Language emptiness for timed automata

B. Srivathsan

Chennai Mathematical Institute, India

In this document, we'll consider the following question:

Given a timed automaton $\mathcal{A}$, is the language $\mathcal{L}(\mathcal{A})$ empty?

This problem is known as the *emptiness* or the *reachability* problem for timed automaton. It essentially amounts to checking if there is an execution of the automaton from an initial state to a final state. In practice, solutions to this problem can be used to check if a bad state of a system modelled as a timed automaton can ever be reached.

Figure 1.1 gives an example of a time automaton. Figure 1.2 shows the different behaviours of this automaton.

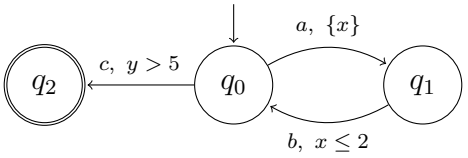

Figure 1.1: A timed automaton $\mathcal{A}_1$

Nodes in Figure 1.2 are configurations of the form (q, v) where q is a state of the automaton and v is a valuation giving values for each of the clocks. Each arrow gives the amount of time spent at a configuration and the action that is taken subsequently. For instance, $(q_1, \langle 0, 1.3\rangle) \xrightarrow{2,b} (q_0, \langle 2, 3.3\rangle)$ means that at state q_1 with the clock values being $\langle 0, 1.3\rangle$ the automaton spends 2 time units and takes the transition b. Hence the values of the clocks would become $\langle 2, 3.3\rangle$.

As we see, the space of configurations is uncountably infinite due to the dense time component. To solve the emptiness problem, one needs to know if there is a sequence of enabled transitions leading to the final state. To know if a transition is enabled, one needs to check if the values of clocks before taking the transition satisfy the corresponding guard.

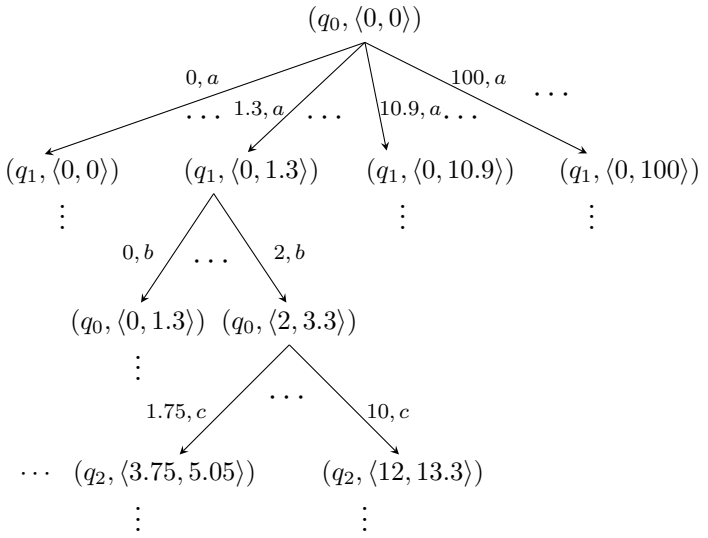

Figure 1.2: Part of the transition system showing the behaviours of the timed automaton $\mathcal{A}_1$ of Figure 1.1

This calls for an effective and efficient handling of the uncountably infinite space of clock valuations. This is the main challenge faced in the analysis.

The first solution to this problem was given in the paper that introduced timed automata [AD94]. We will now see this solution, before which we would need some prelimary definitions.

1 Preliminaries

Let $\mathbb{R}_{\geq 0}$ denote the set of non-negative reals. A clock is a variable that ranges over $\mathbb{R}_{\geq 0}$. Let X be a set of clocks. A *clock constraint* ϕ is a conjunction of comparisons of a clock with a constant, given by the following grammar:

$$\phi := x \sim c \mid \phi \wedge \phi$$

where $x \in X$, $\sim \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$. For example, $(x \leq 3 \wedge y > 0)$ is a clock constraint. Let $\Phi(X)$ denote the set of clock constraints over the set of clocks X.

Remark 1 In the above, we have chosen only integer constants in the clock constraints. We will see later that for the emptiness problem, choosing c to be in $\mathbb{N}$ or in $\mathbb{Q}$ does not make a difference.

Definition 2 (Clock valuation) A *clock valuation* over X is a function $v : X \to \mathbb{R}_{\geq 0}$. The set of all clock valuations is denoted by $\mathbb{R}_{\geq 0}^X$.

We denote by $\mathbf{0}$ the valuation that associates 0 to every clock in X. A valuation v is said to satisfy a constraint ϕ, written as $v \vDash \phi$, when every constraint in ϕ holds after replacing every x by $v(x)$. For example, $\langle x = 0.4, y = 9.3, z = 4.6\rangle \vDash (x < 5) \wedge y > 4$.

For $\delta \in \mathbb{R}_{\geq 0}$, let $v + \delta$ be the valuation that associates $v(x) + \delta$ to every clock x. For instance, $\langle 0.4, 9.3, 4.6\rangle + 5.1 = \langle 5.5, 14.4, 9.7\rangle$. We will use this notation to talk about the valuations that arise after some δ time-elapse.

For $R \subseteq X$, let $[R]v$ be the valuation that sets x to 0 if $x \in R$, and that sets x to $v(x)$ otherwise. For example, $[\{x, y\}]\langle 0.4, 9.3, 4.6\rangle = \langle 0, 0, 4.6\rangle$. This notation would be used to denote the valuation that is obtained from v after resetting clocks in R.

For a valuation v and a clock x, we denote the integral part of $v(x)$ by $\lfloor v(x) \rfloor$ and the fractional part of $v(x)$ by $\{v(x)\}$. We write $\lessdot$ to mean either $\leq$ or $<$, and $\gtrdot$ to mean either $\geq$ or $>$.

Let us start with a formal definition of a timed automaton.

Definition 3 (Timed automaton [AD94]) A *Timed Automaton (TA)* is a tuple $\mathcal{A} = (Q, \Sigma, X, T, q_0, Acc)$ where

- Q is a finite set of states,
- Σ is a finite alphabet,
- X is a finite set of clocks,
- $q_0 \in Q$ is the initial state,
- $Acc \subseteq Q$ is a set of accepting states,
- $T \subseteq Q \times \Sigma \times \Phi(X) \times 2^X \times Q$ is a finite set of transitions (q, a, g, R, q') where a is a letter in Σ, g is a clock constraint called the *guard*, and R is the set of clocks that are *reset* on the transition.

Remark 4 For technical convenience, we choose to have a single initial state $q_0 \in Q$ instead of a set of initial states $Q_0 \subseteq Q$. For the algorithm that we propose, this does not make any difference. One can easily extrapolate it to the case of multiple initial states.

The behaviour of a timed automaton is described by a graph as in Figure 1.2. We will call it the *semantics* of a timed automaton. Here is the formal definition of this infinite graph.

Definition 5 (Semantics of a timed automaton) Let $\mathcal{A}$ be a timed automaton. The semantics of $\mathcal{A}$ is given by a transition system $\mathcal{S}_\mathcal{A}$ whose nodes are configurations (q, v) consisting of a state q of $\mathcal{A}$ and a valuation v giving the values of clocks. The initial configuration is given by $(q_0, \mathbf{0})$ with q_0 being the initial state of $\mathcal{A}$. The transition relation $\rightarrow$ is a union of two kinds of transitions:

delay $(q, v) \rightarrow^\delta (q, v + \delta)$ for some $\delta \in \mathbb{R}_{\geq 0}$;

action $(q, v) \rightarrow^t (q', v')$ for some transition $t = (q, a, g, R, q') \in T$ such that $v \vDash g$ and $v' = [R]v$.

A *run* of $\mathcal{A}$ is a finite sequence of transitions starting from the initial configuration $(q_0, \mathbf{0})$. Without loss of generality, we can assume that the first transition is a delay transition and that delay and action transitions alternate (Why?). We write $(q, v) \xrightarrow{\delta,t} (q', v')$ if there is a delay transition $(q, v) \to^{\delta} (q, v + \delta)$ followed by an action transition $(q, v + \delta) \to^{t} (q', v')$. So a run of $\mathcal{A}$ can be written as:

$$(q_0, v_0) \xrightarrow{\delta_0,t_0} (q_1, v_1) \xrightarrow{\delta_1,t_1} (q_2, v_2) \xrightarrow{\delta_2,t_2} \dots \xrightarrow{\delta_{n-1},t_{n-1}} (q_n, v_n)$$

where (q_0, v_0) represents the initial configuration $(q_0, \mathbf{0})$.

A run is *accepting* if it ends in a configuration (q_n, v_n) with $q_n \in Acc$.

Definition 6 (Emptiness problem) The *emptiness problem* for timed automata is to decide whether a given automaton has an accepting run. This problem is known to be decidable [AD94, CY92].

2 Algorithm for the emptiness problem

Since the transition system determined by an automaton is infinite, the standard solution is to find a finite approximation of this transition system by grouping valuations together. The grouping should be done in such a way that v and v' are in the same group if all the states of the automaton that are reachable from (q, v) and (q, v') are the same, for all q. That is, if there is the following run from (q, v):

$$(q, v) \xrightarrow{\delta_0,t_0} (q_1, v_1) \xrightarrow{\delta_1,t_1} (q_2, v_2) \xrightarrow{\delta_2,t_2} \quad \dots \quad \xrightarrow{\delta_{n-1},t_{n-1}} (q_n, v_n)$$

then there should be a run with the same sequence of discrete states $q_1, \dots, q_n$ from (q, v'). That is, there exist $\delta'_0, \delta'_1, \dots, \delta'_{n-1} \in \mathbb{R}_{\geq 0}$ such that the following is a valid run from (q, v'):

$$(q, v') \xrightarrow{\delta'_0,t_0} (q_1, v'_1) \xrightarrow{\delta'_1,t_1} (q_2, v'_2) \xrightarrow{\delta'_2,t_2} \quad \dots \quad \xrightarrow{\delta'_{n-1},t_{n-1}} (q_n, v'_n)$$

Note that for the emptiness problem, we do not care about the intermediate times spent in a run. We just insist that v' should be able to take the same sequence of transitions as v.

One such grouping is given by the *region abstraction* [AD94]. The space of valuations is partitioned into a finite number of *regions*. Two valuations within a region are indistinguishable with respect to reachability. Having formed these finite number of regions, a cross product with the states of the automaton is taken to give state-augmented regions. These state-augmented regions act as nodes of what is called the *region graph* of the automaton whose transitions are defined in a natural way, using the valuations present in the region. The analysis of the automaton is then performed using this finite region graph.

Let X be a finite set of clocks. Let $M : X \mapsto \mathbb{N} \cup \{-\infty\}$ be a *bound function* that associates a constant $M_x \in \mathbb{N}$ to every clock x. This is a slight generalization from what we saw in class, where considered a single constant M for all clocks.

Definition 7 (Region equivalence [AD94]) Two valuations $v, v' \in \mathbb{R}^X_{\geq 0}$ are *region equivalent* w.r.t. M, denoted $v \sim_M v'$ iff for every $x, y \in X$:

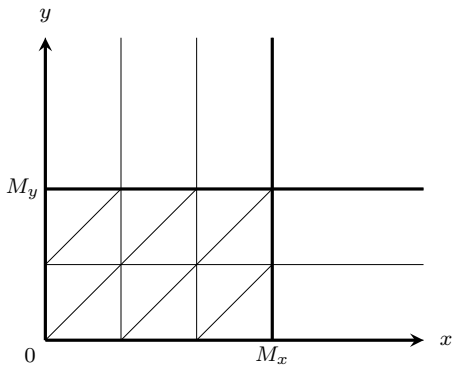

Figure 1.3: Division into regions with two clocks x and y [AD94].

1. $v(x) > M_x$ iff $v'(x) > M_x$;
2. if $v(x) \leq M_x$, then $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$;
3. if $v(x) \leq M_x$, then $\{v(x)\} = 0$ iff $\{v'(x)\} = 0$;
4. if $v(x) \leq M_x$ and $v(y) \leq M_y$ then $\{v(x)\} \leq \{v(y)\}$ iff $\{v'(x)\} \leq \{v'(y)\}$.

Given an automaton $\mathcal{A}$, a bound function is obtained by choosing for a clock x, the maximum constant appearing in a guard involving x. Then, the first three conditions in the above definition ensure that the two valuations satisfy the same guards. The last one enforces that for every $\delta \in \mathbb{R}_{\geq 0}$ there is $\delta' \in \mathbb{R}_{\geq 0}$, such that valuations $v + \delta$ and $v' + \delta'$ satisfy the same guards.

Definition 8 (Region [AD94]) Let $M : X \mapsto \mathbb{N} \cup \{-\infty\}$ be a bound function. A *region* is an equivalence class of $\sim_M$. We write $[v]^M$ for the region of v, and $\mathcal{R}_M$ for the set of all regions with respect to M.

Figure 1.3 shows the division into regions when there are two clocks x and y. We also give below a constructive definition of regions which would be useful to estimate the number of regions.

Definition 9 (Region: constructive definition [AD94]) A region with respect to bound function M is a set of valuations specified as follows:

1. for each clock $x \in X$, one constraint from the set:
   $\{x = c \mid c = 0, \ldots, M_x\} \cup \{c - 1 < x < c \mid c = 1, \ldots, M_x\} \cup \{x > M_x\}$
2. for each pair of clocks x, y having interval constraints: $c-1 < x < c$ and $d-1 < y < d$, it is specified if $\{x\}$ is less than, equal to or greater than $\{y\}$.

If r is a region then we will write $r \vDash g$ to mean that every valuation in r satisfies the guard g. It is straightforward to see that if a valuation $v \in r$ satisfies the guard g, then every valuation $v' \in r$ satisfies g. We will now show the other property with respect to time-elapse mentioned above.

Lemma 10 Let v, v' be valuations such that $v' \sim_M v$. Then, for all $\delta \in \mathbb{R}_{\geq 0}$, there exists a $\delta' \in \mathbb{R}_{\geq 0}$ such that $v' + \delta' \sim_M v + \delta$.

Proof
We know $v' \sim_M v$ and we are given δ. We need to choose δ'. Put $\lfloor \delta' \rfloor$ to be $\lfloor \delta \rfloor$. Clearly, we have $v' + \lfloor \delta' \rfloor \sim_M v + \lfloor \delta \rfloor$: that is, valuations $v' + \lfloor \delta' \rfloor$ and $v + \lfloor \delta \rfloor$ have the same integral parts and the same ordering of fractional parts (modulo M). Let $x_1 \preccurlyeq_1 x_2 \preccurlyeq_2 \ldots \preccurlyeq_{k-1} x_k$ be the ordering of fractional parts of clocks less than M in both the valuations. Here $\preccurlyeq$ denotes either $<$ or $=$.

From $v + \lfloor \delta \rfloor$, elapsing a fractional amount $\{\delta\}$ might move some of the clocks up to the next integer. Let $x_j, x_{j+1}, \ldots, x_k$ be the clocks that have their integral values increased from $v + \lfloor \delta \rfloor$ due to the fractional elapse $\{\delta\}$. Thanks to the denseness of the real line, one can choose $\{\delta'\}$ between the fractional values of clocks x_{j-1} and x_j in $v' + \lfloor \delta' \rfloor$ so that $v' + \delta'$ has the same integers as $v + \delta$ and the same ordering of fractional parts (modulo M). □

Given a bound function M, the number of regions in $\mathcal{R}_M$ is finite. Once this finite partition of the valuations is obtained, we proceed to define a finite graph built from these regions, that captures the behaviour of the timed automaton.

For an automaton $\mathcal{A}$, to define its region graph, we consider a bound function $M_{\mathcal{A}}$ that is obtained from the automaton's definition.

Definition 11 (Maximal bounds) Given an automaton $\mathcal{A}$, the *maximal bounds function* $M_{\mathcal{A}} : X \mapsto \mathbb{N} \cup \{-\infty\}$ associates to each clock x the biggest constant appearing in a guard of the automaton that involves x. If there is no guard involving x, then $M_{\mathcal{A}}(x)$ is assigned $-\infty$.

We define the region graph of an automaton $\mathcal{A}$ using the $\sim_{M_{\mathcal{A}}}$ relation. Note that in class, we used the terminology *region automaton*. As we are not interested in the language accepted by this automaton, and instead we care if there is a path to the accepting state, we will stick to calling this a region graph.

Definition 12 (Region graph [AD94]) Nodes of the *region graph* denoted by $RG(\mathcal{A})$ are of the form (q, r) for q a state of $\mathcal{A}$ and $r \in \mathcal{R}_{M_{\mathcal{A}}}$ a region. There is a transition $(q, r) \xrightarrow{t} (q', r')$ if there are $v \in r$, $\delta \in \mathbb{R}_{\geq 0}$ and $v' \in r'$ with $(q, v) \xrightarrow{\delta, t} (q', v')$. The initial node of the region graph is $(q_0, [\mathbf{0}]_{\sim_{M_{\mathcal{A}}}})$ where $[\mathbf{0}]_{\sim_{M_{\mathcal{A}}}}$ represents the region to which the initial valuation $\mathbf{0}$ belongs to. A node (q, r) is said to be an *accepting node* if $q \in Acc$.

Observe that a transition in the region graph is not decorated with a delay. Figure 1.4 shows a part of the region graph $RG(\mathcal{A}_1)$ of the automaton $\mathcal{A}_1$ shown in Figure 1.1.

It will be important to understand the property of pre-stability of regions [TYB05].

Lemma 13 (Pre-stability of regions) Let $\mathcal{A}$ be an automaton. Transitions in $RG(\mathcal{A})$ are pre-stable: in each transition $(q, r) \xrightarrow{t} (q', r')$, for every $v \in r$ there is a $\delta \in \mathbb{R}_{\geq 0}$ and a valuation $v' \in r'$ such that $(q, v) \xrightarrow{\delta, t} (q', v')$

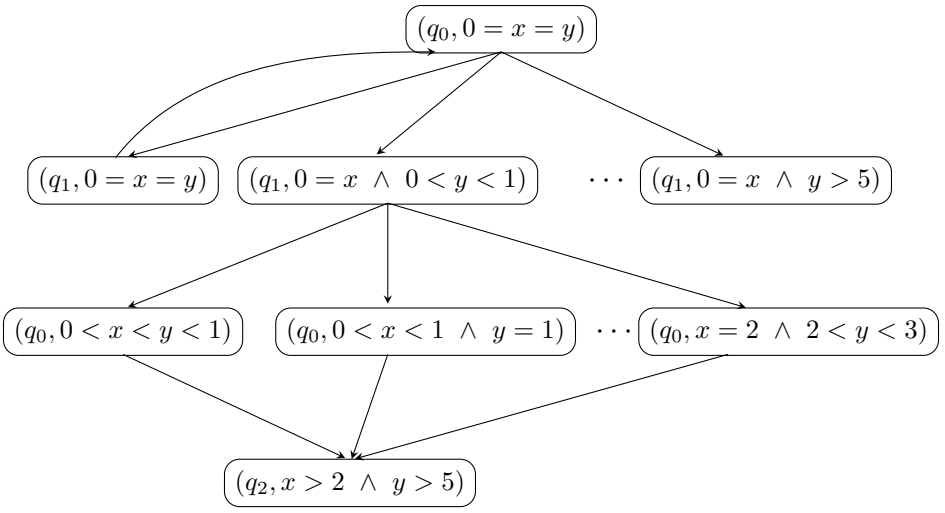

Figure 1.4: Part of region graph of the automaton $\mathcal{A}_1$ shown in Figure 1.1

Proof
By definition of the region graph, a transition $(q_1, r_1) \xrightarrow{t} (q_2, r_2)$ exists in $RG(\mathcal{A})$ if there are $v_1 \in r_1$, $\delta \in \mathbb{R}_{\geq 0}$ and $v_2 \in r_2$ with $(q_1, v_1) \xrightarrow{\delta, t} (q_2, v_2)$.

Let the transition t be (q_1, g, R, q_2). Pick a valuation $v_1' \in r_1$. By Lemma 10, there exists a δ' such that $v_1 + \delta$ and $v_1' + \delta'$ belong to the same region. We know that valuations within the same region satisfy the same guards. Therefore since $v_1 + \delta \vDash g$, we get that $v_1' + \delta' \vDash g$ too. From the definition of region equivalence, we get that regions are stable under projection to a subset of clocks and in particular, this entails that $[R](v_1' + \delta')$ belongs to the same region as $[R](v_1 + \delta)$. □

We will now establish the correspondence between paths of the region graph and runs of the automaton. Consider two sequences

$$(q_0, v_0) \xrightarrow{\delta_0, t_0} (q_1, v_1) \xrightarrow{\delta_1, t_1} \cdots (q_n, v_n) \tag{1.1}$$

$$(q_0, r_0) \xrightarrow{t_0} (q_1, r_1) \xrightarrow{t_1} \cdots (q_n, r_n) \tag{1.2}$$

where the first is a run in $\mathcal{A}$, and the second is a path in $RG(\mathcal{A})$. We say that the first is an *instantiation* of the second if $v_i \in r_i$ for all $i \in \{1, \ldots, n\}$. Equivalently, we say that the second is an *abstraction* of the first. The following lemma is a direct consequence of the pre-stability property.

Lemma 14 Every path in $RG(\mathcal{A})$ is an abstraction of a run of $\mathcal{A}$, and conversely, every run of $\mathcal{A}$ is an instantiation of a path in $RG(\mathcal{A})$.

The above lemma shows that the region graph is sound and complete for state reachability.

Theorem 15 ([AD94]) *Automaton $\mathcal{A}$ has an accepting run iff there is a path in the region graph $RG(\mathcal{A})$ starting from its initial node to an accepting node.*

While this theorem gives an algorithm for solving our problem, it turns out that this method is very impractical. The number of regions obtained using a bound function M is $\mathcal{O}\big(|X|! \cdot 2^{|X|} \cdot \prod_{x \in X}(2M_x + 2)\big)$ [AD94] and constructing all of them, or even searching through them on-the-fly, has proved to be very costly. Later during the course, we will look at more efficient solutions to this problem.

References

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

[CY92] C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. *Form. Methods Syst. Des.*, 1(4):385–415, 1992.

[TYB05] S. Tripakis, S. Yovine, and A. Bouajjani. Checking timed Büchi automata emptiness efficiently. *Formal Methods in System Design*, 26(3):267–292, 2005.