Reachability algorithm using zones

B. Srivathsan
Chennai Mathematical Institute, India

In a previous lecture, we asked the following question: given a timed automaton $\mathcal{A} = (Q, \Sigma, X, T, q_0, Acc)$, when is $\mathcal{L}(\mathcal{A})$ empty? $\mathcal{L}(\mathcal{A})$ is non-empty iff there exists a run of the automaton that leads to an accepting state. Note that existence of *an accepting run* does not depend on how the letters of Σ are labeled on the transitions. In fact, it does not depend on Σ itself. Henceforth, we consider automata without an alphabet: $\mathcal{A} = (Q, X, T, q_0, Acc)$. Language emptiness then reduces to asking if an accepting state is *reachable*. The language emptiness problem would now be called the *reachability problem* for timed automata.

We have seen that a solution to this problem proceeds by the region graph construction. As we have seen, the number of regions is exponential in the number of clocks. While modeling a system, each component of the system is modeled as a timed automaton and the entire system is then obtained by a product construction of the individual automata. This immediately gives rise to many states, a phenomenon known as *state-space explosion*. If on top of this, one attaches exponentially many regions to each state, the algorithm runs out of memory. Therefore the region based method is infeasible in practice.

Goal: In this part of the course, we will consider two aspects:

1. How to reduce the number of "time components" attached to each state?
2. How to reduce the number of discrete states themselves?

The broad idea is as follows. We want to design an algorithm that for a timed automaton $\mathcal{A}$ constructs a *finite graph* $\mathsf{Graph}(\mathcal{A})$ with some accepting nodes, that satisfies the following two properties:

soundness: if an accepting node is reacheable in $\mathsf{Graph}(\mathcal{A})$ then there is a run of $\mathcal{A}$ that reaches an accepting state

completeness: if an accepting state is reachable in $\mathcal{A}$ then an accepting node is reachable in $\mathsf{Graph}(\mathcal{A})$

If we manage to define such a $\mathsf{Graph}(\mathcal{A})$, then one could have an algorithm that constructs and simultaneously searches this graph (using standard breadth-first search or depth-search search methods) for an accepting node. The goal is to come up with a $\mathsf{Graph}(\mathcal{A})$ as small as possible and that can be efficiently computed.

1 Zones

Let us first recall the semantics of a timed automaton:

Definition 1 (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 $\mathcal{A}$ and $\mathbf{0}$ the valuation that attaches the value 0 to every clock. 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 $t = (q, g, R, q') \in T$ such that $v \vDash g$ and $v' = [R]v$.

To get a finite $\mathsf{Graph}(\mathcal{A})$, a standard solution is to group together all the valuations reaching a state of the automaton via a particular path. We first define a transition relation $\Rightarrow$ over nodes of the form (q, W) where W is a set of valuations.

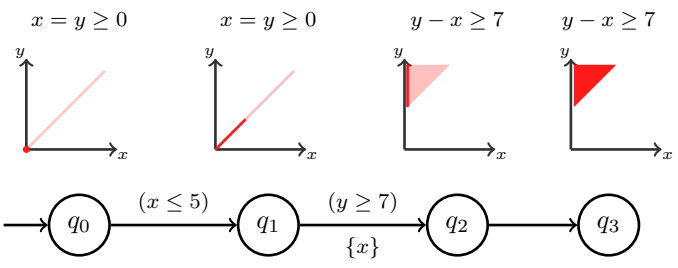
Definition 2 (Symbolic transition $\Rightarrow$) Let $\mathcal{A}$ be a timed automaton. For every transition t of $\mathcal{A}$ and every set of valuations W, we have a transition $\Rightarrow^{t}$ defined as follows:

$$(q, W) \Rightarrow^{t} (q, W') \text{ where } W' = \{v' \mid \exists v \in W,\ \exists \delta \in \mathbb{R}_{\geq 0}.\ (q, v) \rightarrow^{t} \rightarrow^{\delta} (q', v')\}$$

The transition relation $\Rightarrow$ is the union of all $\Rightarrow^{t}$.

The transition relation defined above considers each valuation $v \in W$ that can take the transition t, obtains the valuation after the transition and then collects the time-successors from this obtained valuation. Therefore the symbolic transition $\Rightarrow$ always yields sets closed under time-successors. The initial configuration of the automaton is $(q_0, \mathbf{0})$. Starting from the initial valuation $\mathbf{0}$ the set of valuations reachable by a time elapse at the initial state are given by $\{\mathbf{0} + \delta \mid \delta \in \mathbb{R}_{\geq 0}\}$. Call this W_0. From (q_0, W_0) as the initial node, computing the symbolic transition relation $\Rightarrow$ leads to different nodes (q, W) wherein the sets W are closed under time-successors.

Example 3 Consider the automaton with two clocks shown below. The sets of valuations computed using the above symbolic transition relation is shown on the top of the automaton.



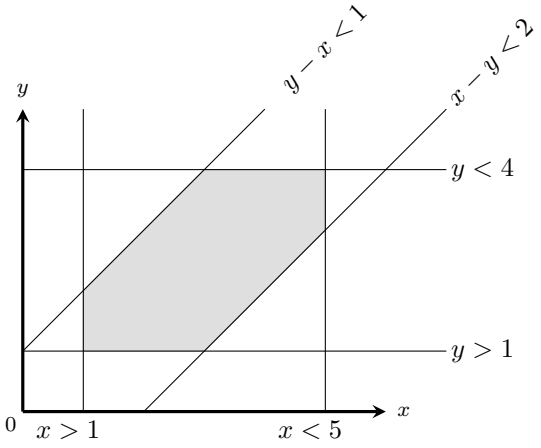


Figure 1.1: An example of a zone

It has additionally been noticed that the sets W obtained in the nodes (q, W) can be described by some simple constraints involving only the difference between clocks [BY04]. This has motivated the definition of *zones*, which are sets of valuations defined by difference constraints.

Definition 4 (Zones [BY04]) A zone is a set of valuations defined by a conjunction of two kinds of clock constraints: for $x, y \in X$

$$x \sim c$$
$$x - y \sim c$$

where, $\sim \in \{\leq, <, =, >, \geq\}$ and $c \in \mathbb{Z}$. For example, $(x > 4 \ \wedge \ y - x \leq 1)$ is a zone.

Another example of a zone is illustrated in Figure 1.1. The sets depicted in Example 3 are zones. We will prove in the next lecture that starting from a node (q, W) with W being a zone, the transition $(q, W) \Rightarrow (q', W')$ leads to a node in which W' is again a zone [BY04]. Observe that the initial set of valuations $W_0 = \{\mathbf{0} + \delta \mid \delta \in \mathbb{R}_{\geq 0}\}$ is indeed a zone: it is given by the constraints

$$\bigwedge_{x,y \in X} (x \geq 0 \ \wedge \ x - y = 0)$$

We will now define a symbolic semantics of timed automata which is a transition system with nodes consisting of zones. This is called the *zone graph* of the automaton. In the sequel, zones are denoted by Z, Z', etc.

Definition 5 (Zone graph) Given a timed automaton $\mathcal{A} = (Q, q_0, X, T, Acc)$, the *zone graph* $ZG(\mathcal{A})$ of $\mathcal{A}$ is a transition system whose nodes are of the form (q, Z) with $q \in Q$ and Z a zone. The initial node is (q_0, Z_0) where $Z_0 = \{\mathbf{0} + \delta \mid \delta \in \mathbb{R}_{\geq 0}\}$ is the set of valuations obtained by elapsing time from $\mathbf{0}$. The transitions are given by the relation $\Rightarrow$ of Definition 2.

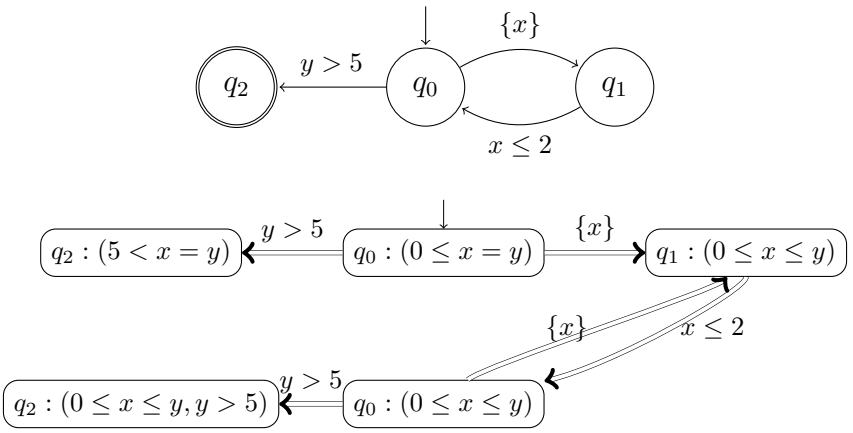


Figure 1.2: An automaton and part of its zone graph

As zones have a simple description, they can be efficiently represented using what are called Difference-Bound Matrices (DBMs) [Dil90]. Figure 1.1 shows another example of an automaton and a part of its zone graph. We will now see how the successor of a node (q, Z) is computed. Successors of a node (q, Z) are the of nodes (q', Z') such that there exists a transition t and $(q, Z) \Rightarrow^t (q', Z')$

Successor computation in the zone graph

The successor computation $(q, Z) \Rightarrow^t (q', Z')$ for a transition $t = (q, g, R, q')$ proceeds in the following steps.

$$(q, Z) \xrightarrow{guard} (q, Z \wedge g) \xrightarrow{reset} (q, [R](Z \wedge g)) \xrightarrow{elapse} (q, Z')$$

In the above, $Z \wedge g$ represents the set of valuations that satisfy the constraints of both Z and g; the set $[R](Z \wedge g)$ represents the set of valuations obtained by resetting clocks in R from every valuation in $Z \wedge g$ and finally Z' is the set of valuations obtained by elapsing an arbitrary amount of time from $[R](Z \wedge g)$. All these operations can be computed efficiently using DBMs.

The costliest operation is the computation of the intersection of a zone with a guard. In the general case when the guards are diagonals like $x - y \leq 5$, the intersection takes $\mathcal{O}(|X|^3)$. However when the guards are diagonal free, it has been shown in [ZLZ05] that the intersection operation can be done in $\mathcal{O}(|X|^2)$ time. Another crucial operation required in algorithms using zones is to know when a zone Z is included in another zone Z'. We list the common operations on zones and the complexity required to perform these operations in Table 1.1. We will see more details about these operations in the next lecture. From the table it can be inferred that computing the successor in the zone graph has a complexity quadratic in the number of clocks.

Operation	Complexity	
$Z \wedge g$	$\mathcal{O}(\|X\|^2)$	$Z \wedge g = \{v \mid v \in Z \text{ and } v \vDash g\}$
$[R](Z)$	$\mathcal{O}(\|X\|.\|R\|)$	$[R](Z) = \{[R]v \mid v \in Z\}$
elapse(Z)	$\mathcal{O}(\|X\|)$	elapse$(Z) = \{v + \delta \mid v \in Z \text{ and } \delta \in \mathbb{R}_{\geq 0}\}$
$Z \subseteq Z'$	$\mathcal{O}(\|X\|^2)$	

Table 1.1: Operations on zones (**note that g is diagonal free)**

2 A naive reachability algorithm

Algorithm 1.1 shows a reachability algorithm that uses zones. The procedure starts from (q_0, Z_0) and repeatedly computes and searches for a node that has an accepting state.

Algorithm 1.1: Reachability procedure using zones

```
1   Input: Timed automaton A = (Q, X = {x_1, ..., x_k}, q_0, T, Acc)
2   Output: Yes, when some state in Acc is reachable from q_0; No otherwise
3
4   function main()
5
6   Waiting := ∅;
7   Passed := ∅;
8   Z_0 := 0 ≤ x_1 = x_2 = ··· = x_k
9
10  Add (q_0, Z_0) to Waiting
11
12  while (Waiting ≠ ∅)
13      Remove (q, Z) from Waiting
14
15      if (q is accepting)
16          exit Yes
17      else if (∃ (q, Z') ∈ Passed s.t. Z ⊆ Z')
18          continue
19      else
20          for each (q_s, Z_s) s.t. (q, Z) ⇒ (q_s, Z_s) do
21              if (Z_s ≠ ∅)
22                  Add (q_s, Z_s) to Waiting
23          Add (q, Z) to Passed
24
25  return No
```

At any point of time during the execution of the algorithm, the part of the zone graph computed by the algorithm is depicted in Figure 1.3.

Some remarks about the algorithm

1. There are three kinds of nodes computed by the algorithm: the ones in the *Passed* list whose successors have been computed, the ones in the *Waiting* list that are yet to be *explored* and the *Covered* nodes which are included in an existing *Passed* node.

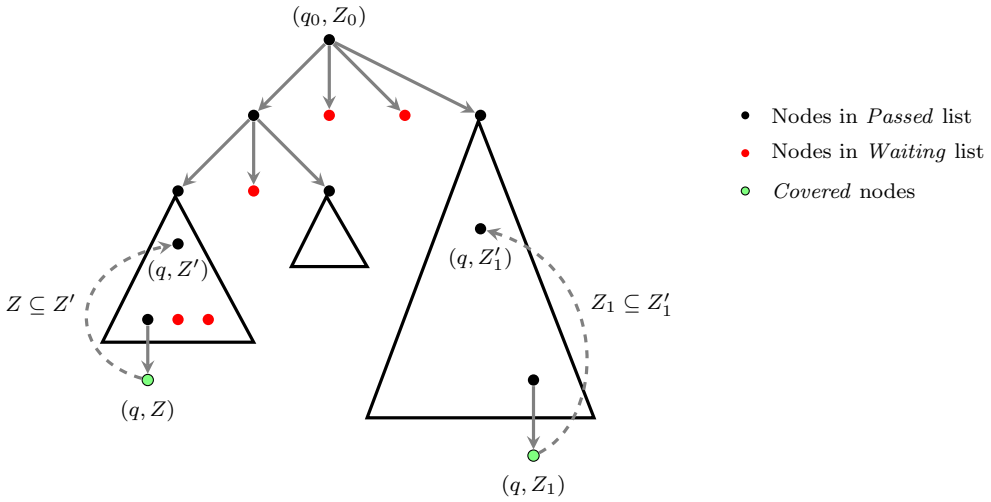


Figure 1.3: Snapshot of the partially computed zone graph during the execution of the algorithm

Such covered nodes are not explored further. The covered nodes are not explicitly stored.

2. The algorithm computes a tree. However, the exact tree structure is not maintained. Only two lists of nodes are maintained. Although, for proofs it is convenient to visualize the tree.

3. Depending on the data-structures used for the list, the search order is determined. If the lists are stacks, then the search is depth-first. If the lists are queues, then the search is breadth-first.

4. The algorithm is *on-the-fly*. We do not need the entire zone graph beforehand to start the search for the accepting state. Note that we have only the automaton with us and the nodes are computed as and when they are required.

Correctness of the algorithm

Note that the algorithm does not compute the entire zone graph. It does some optimization by stopping the exploration of a node (q, Z) if it is already covered by a *Passed* node. Therefore it is important to justify the correctness of this procedure.

Lemma 6 (Soundness) If there is a path $(q_0, Z_0) \Rightarrow^{t_1} (q_1, Z_1) \Rightarrow^{t_2} \ldots \Rightarrow^{t_n} (q_n, Z_n)$ in the tree computed by Algorithm 1.1, then there is a run of the automaton $\mathcal{A}$:

$$(q_0, v_0) \xrightarrow{\delta_0, t_0} (q_1, v_1) \xrightarrow{\delta_1, t_1} \ldots \xrightarrow{\delta_n, t_n} (q_n, v_n)$$

such that $v_i \in Z_i$ for all $i \leq n$.

Proof
Direct from the definition of the symbolic transition (c.f. Definition 2). □

It is the completeness aspect that needs some justification as we stop exploration sooner.

Lemma 7 (Completeness) Let $\rho := (q_0, v_0) \xrightarrow{\delta_0, t_0} (q_1, v_1) \xrightarrow{\delta_1, t_1} \ldots \xrightarrow{\delta_n, t_n} (q_n, v_n)$ be a run of $\mathcal{A}$ such that for $0 \leq i \leq n-1$, $q_i \notin Acc$, and q_n could either be in Acc or not. Then for each $i \leq n-1$, there exists a node (q_i, Z_i) in the Passed list of Algorithm 1.1 such that $v_i \in Z_i$.

Proof
We proceed by an induction on the length of the run leading to (q_n, v_n).

Base case: We know that $v_0 \in Z_0$. The node (q_0, Z_0) is added to the Waiting list in Line 10. When Line 14 is executed the first time, the node (q_0, Z_0) is added to the Passed list. For the base case, (q_0, Z_0) is the required node.

Induction case: Assume that for all $0 \leq i \leq m$, there exists (q_i, Z_i) in Passed such that $v_i \in Z_i$. We will now show that there exists (q_{m+1}, Z_{m+1}) in Passed such that $v_{m+1} \in Z_{m+1}$.

By the induction hypothesis, we have (q_m, Z_m) in Passed such that $v_m \in Z_m$. Consider the transition $(q_m, v_m) \xrightarrow{\delta_m, t_m} (q_{m+1}, v_{m+1})$ of the run ρ. As (q_m, Z_m) is in Passed, the transition $\Rightarrow^{t_m}$ would have been considered in the for loop of Line 20. Let $(q_m, Z_m) \Rightarrow^{t_m} (q_{m+1}, Z_{m+1})$ be the transition in the zone graph. By definition of the symbolic transition, $v_{m+1} \in Z_{m+1}$.

If (q_{m+1}, Z_{m+1}) is in Passed, we are done. If not, either $q_{m+1} \in Acc$ and the algorithm would have exited at Line 16. In this case, due to the condition required by the Lemma, $m = n-1$ and we are done. The only other case when (q_{m+1}, Z_{m+1}) is not in Passed is when there exists (q_{m+1}, Z'_{m+1}) in Passed such that $Z_{m+1} \subseteq Z'_{m+1}$). Therefore, $v_{m+1} \in Z'_{m+1}$ and since (q_{m+1}, Z'_{m+1}) is in Passed, our required node would be (q_{m+1}, Z'_{m+1}).

□

The above two lemmas tell that the algorithm can detect reachability of an accepting state correctly. However, we have not given justifications about its termination. In fact, it turns out that the algorithm might not terminate for some automata.

Lemma 8 (Non-termination) There exist automata for which Algorithm 1.1 might not terminate.

Proof
Consider the automaton $\mathcal{A}_{inf}$ shown in Figure 1.4, with two clocks $\{x, y\}$ and no accepting state. The initial node is given by $(q_0, x = y \ \wedge \ x \geq 0)$. The transition to q_1 gives the node $(q_1, x = y \ \wedge \ x \geq 0)$. The only transition from q_1 taken from this node gives the node $(q_1, x - y = 1 \ \wedge \ x \geq 0)$, which is a new node. This node has its own successors and the process continues. Finally at q_1 we have the following zones in the zone graph $ZG(\mathcal{A}_{inf})$, all of which will be computed by Algorithm 1.1 as q_1 is non-accepting:

$$(\ x - y = k \ \wedge \ x \geq 0 \ ) \ \text{ for all } k \in \mathbb{N}$$

This is pictorially shown in Figure 1.4.

□

The next section explains methods to make the algorithm terminating: in other words, methods to get a *finite abstraction* of the zone graph.

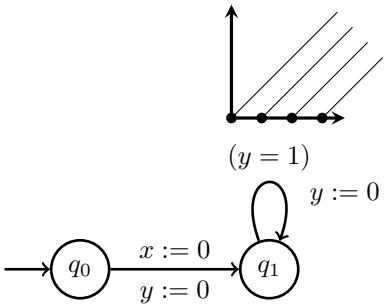


Figure 1.4: Automaton $\mathcal{A}_{inf}$ and the graph of zones obtained at q_1

3 Finite abstractions of the zone graph

As seen from the example in Figure 1.4, it is useless to explore the state q_1 again and again even if the zones are not included in each other. Ideally, we want to something like this:

> Let (q, Z) be a node newly removed from Waiting list. If there exists a node (q, Z') in Passed such that *all sequences of transitions* that can be seen from (q, Z) can be seen from (q, Z') too, then we don't want to explore (q, Z).

This is because to check for state reachability, it is enough to determine one path that leads to that state. Therefore, the criterion given above preserves the set of reachable states. A convenient way of formalizing the above criterion is by making use of *simulation relations*.

Definition 9 ((Time-abstract) Simulation) A *(time-abstract) simulation* between two states of transition system $\mathcal{S}_{\mathcal{A}}$ (Definition 1) is a relation $(q, v) \preceq_{\mathcal{A}} (q', v')$ such that:

- $q = q'$,
- if $(q, v) \to^{\delta} (q, v + \delta) \to^{t} (q_1, v_1)$, then there exists a $\delta' \in \mathbb{R}_{\geq 0}$ such that $(q, v') \to^{\delta'} (q, v' + \delta') \to^{t} (q_1, v_1')$ satisfying $(q_1, v_1) \preceq_{\mathcal{A}} (q_1, v_1')$ for the same transition t.

We say that (q, v) is simulated by (q', v').

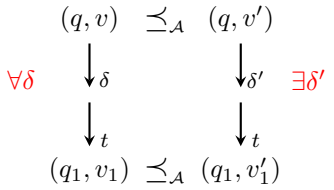


Figure 1.5: Illustration of Definition 9

The definition is pictorially represented in Figure 1.5. Essentially the above definition says that if the configuration (q, v) can elapse δ time units and take a transition t, then the same transition can be taken from (q, v') after an elapse of some δ' time units. We don't need δ and δ' to be the same. It is enough for us to ensure that transition t is taken. The fact that the resulting configuration (q_1, v_1) is simulated by (q_1, v_1') ensures that all sequences of transitions that can be seen from (q, v) can be seen from (q', v') too.

We now need to extend these simulation relation to *sets of valuations*.

Definition 10 (Simulation on sets of valuations) Let $\mathcal{A}$ be an automaton W and W' be two sets of valuations of $\mathcal{A}$. For each state q, we say that (q, W) is simulated by (q, W') written as $(q, W) \preceq_{\mathcal{A}} (q, W')$ if:

$$\text{for every } v \in W, \text{ exists } v' \in W' \text{ s.t. } (q, v) \preceq_{\mathcal{A}} (q, v')$$

We state below a property of simulation relations that is easy to prove.

Proposition 11 Simulation relations are reflexive and transitive.

Note that if (q, W) is simulated by (q, W') then all states that are reachable from (q, W) would be contained in the set of states reachable from (q, W'). This fact can be used to give a correct criterion in Algorithm 1.1 to stop exploration of a zone (q, Z) that is newly removed from the Waiting list. Currently the algorithm uses set inclusion $Z \subseteq Z'$, which is an example of a simulation relation.

Extending the naive algorithm with simulation relations

The modified algorithm is given below. The only change is the test $(q, Z) \preceq_{\mathcal{A}} (q, Z')$ instead of $Z \subseteq Z'$ when (q, Z) is newly removed from the Waiting list.

Algorithm 1.2: Reachability procedure using zones and simulation relations

```
1   Input: Timed automaton A = (Q, X = {x_1, ..., x_k}, q_0, T, Acc) and
2          ⪯_A a simulation relation
3   Output: Yes, when some state in Acc is reachable from q_0; No otherwise
4
5   function main()
6
7   Waiting := ∅;
8   Passed := ∅;
9   Z_0 := 0 ≤ x_1 = x_2 = ··· = x_k
10
11  Add (q_0, Z_0) to Waiting
12
13  while (Waiting ≠ ∅)
14      Remove (q, Z) from Waiting
15
16      if (q is accepting)
17          exit Yes
18      else if (∃ (q, Z') ∈ Passed s.t. (q, Z) ⪯_A (q, Z'))
19          continue
20      else
21          for each (q_s, Z_s) s.t. (q, Z) ⇒ (q_s, Z_s) do
22              if (Z_s ≠ ∅)
23                  Add (q_s, Z_s) to Waiting
24          Add (q, Z) to Passed
25
26  return No
```

The goal is to come up with *finite simulation relations*.

Definition 12 (Finite simulations) A simulation relation $\preceq$ on a transition system $(S, \rightarrow)$ is finite if there exists a natural number N such that in every run:

$$s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_K$$

where $K \geq N$, there exist i, j with $i < j \leq K$ satisfying $s_j \preceq s_i$.

If we can come up with a finite simulation relation $\preceq_{\mathcal{A}}$ in Algorithm 1.2, then we are guaranteed to terminate. Before coming up with finite simulation relations, we need to justify correctness of Algorithm 1.2. Yet again, it is completeness that is non-trivial.

Lemma 13 (Soundness) If there is a path $(q_0, Z_0) \Rightarrow^{t_1} (q_1, Z_1) \Rightarrow^{t_2} \ldots \Rightarrow^{t_n} (q_n, Z_n)$ in the tree computed by Algorithm 1.2, then there is a run of the automaton $\mathcal{A}$:

$$(q_0, v_0) \xrightarrow{\delta_0, t_0} (q_1, v_1) \xrightarrow{\delta_1, t_1} \ldots \xrightarrow{\delta_n, t_n} (q_n, v_n)$$

such that $v_i \in Z_i$ for all $i \leq n$.

Proof
Direct from the definition of the symbolic transition (c.f. Definition 2). □

Lemma 14 (Completeness) Let $\preceq_{\mathcal{A}}$ be a time-abstract simulation on the semantics of an automaton $\mathcal{A}$. Let

$$\rho := (q_0, v_0) \xrightarrow{\delta_0, t_0} (q_1, v_1) \xrightarrow{\delta_1, t_1} \ldots \xrightarrow{\delta_n, t_n} (q_n, v_n)$$

be a run of $\mathcal{A}$ such that for $0 \leq i \leq n-1$, $q_i \notin Acc$, and q_n could either be in Acc or not. Then for each $i \leq n-1$, there exists a node (q_i, Z_i) in the Passed list of Algorithm 1.2 that contains a valuation which simulates v_i: that is, there exists $v_i' \in Z_i$ such that $(q_i, v_i) \preceq_{\mathcal{A}} (q_i, v_i')$.

Proof
We proceed by an induction on the length of the run leading to (q_n, v_n).

Base case: As $\preceq_{\mathcal{A}}$ is reflexive, we have $(q_0, v_0) \preceq_{\mathcal{A}} (q_0, v_0)$. Moreover, we know that $v_0 \in Z_0$ and (q_0, Z_0) is added to the Passed list of Algorithm 1.2. This proves the base case.

Induction case: Assume that for all $0 \leq i \leq m$, there exists (q_i, Z_i) in Passed and $v_i' \in Z_i$ such that $(q_i, v_i) \preceq_{\mathcal{A}} (q_i, v_i')$. We will now show that there exist (q_{m+1}, Z_{m+1}) in Passed and $v_{m+1}' \in Z_{m+1}$ such that $(q_{m+1}, v_{m+1}) \preceq_{\mathcal{A}} (q_{m+1}, v_{m+1}')$.

By the induction hypothesis, we have (q_m, Z_m) in Passed and $v_m' \in Z_m$ such that $(q_m, v_m) \preceq_{\mathcal{A}} (q_m, v_m')$. Consider the transition $(q_m, v_m) \xrightarrow{\delta_m, t_m} (q_{m+1}, v_{m+1})$ of the run ρ. Firstly, as (q_m, v_m') simulates (q_m, v_m) there exists a δ_m' such that:

$$(q_m, v_m') \xrightarrow{\delta_m', t_m} (q_{m+1}, v_{m+1}') \quad \text{and} \quad (q_{m+1}, v_{m+1}) \preceq_{\mathcal{A}} (q_{m+1}, v_{m+1}')$$

Secondly, as (q_m, Z_m) is in Passed, the transition $\Rightarrow^{t_m}$ would have been considered in the for loop of Line 20. Let $(q_m, Z_m) \Rightarrow^{t_m} (q_{m+1}, Z_{m+1})$ be the transition in the zone graph. By definition of the symbolic transition, $v'_{m+1} \in Z_{m+1}$.

If (q_{m+1}, Z_{m+1}) is in Passed, we are done as this gives us a node in Passed and a valuation $v'_{m+1} \in Z_{m+1}$ such that $(q_{m+1}, v_{m+1} \preceq_{\mathcal{A}} (q_{m+1}, v'_{m+1})$.

If not, either $q_{m+1} \in Acc$ and the algorithm would have exited at Line 16. In this case, due to the condition required by the Lemma, $m = n - 1$ and we are done. The only other case when (q_{m+1}, Z_{m+1}) is not in Passed is when there exists (q_{m+1}, Z'_{m+1}) in Passed such that $Z_{m+1} \preceq_{\mathcal{A}} Z'_{m+1}$). By Definition 10, there exists a u in Z'_{m+1} such that $(q_{m+1}, v_{m+1}) \preceq_{\mathcal{A}} (q_{m+1}, u)$. As simulation relations are transitive, we have $(q_{m+1}, v_{m+1}) \preceq_{\mathcal{A}} (q_{m+1}, u)$. This proves the induction case. □

The above two lemmas tell us that the algorithm is correct. If additionally, we can come up with a finite simulation relation, the algorithm terminates.

Theorem 15 *Let $\mathcal{A}$ be a timed automaton. Algorithm 1.2 terminates if $\preceq_{\mathcal{A}}$ is a simulation relation. The set of reachable states in the tree computed by Algorithm 1.2 is the same as the set of reachable states in $\mathcal{A}$.*

4 Simulations from the region equivalence

The previous section dealt with a general notion of simulation relation $\preceq_{\mathcal{A}}$. In this section, we will see an example of a simulation relation that can be computed on zones: that is, given (q, Z) and (q, Z') we will give relation $\preceq$ that is finite, and for which $(q, Z) \preceq (q, Z')$ can be checked. For this, we will yet again make use of regions.

Definition 16 For an automaton $\mathcal{A}$, let $M : X \mapsto \mathbb{N} \cup \{-\infty\}$ be the maximum bounds function. We have seen the region equivalence $v \sim_M v'$ in the lecture on regions. Using this we define a simulation relation $\preccurlyeq_M$. For each state q and pair of valuations v, v', we define:

$$(q, v) \preccurlyeq_M (q, v') \quad \text{if} \quad v \sim_M v'$$

Lemma 17 The relation $\preccurlyeq_M$ is a finite simulation.

Proof
$\preccurlyeq_M$ being a simulation is a consequence of the pre-stability property of regions (Lemma 10 and 13 of the notes titled "Language emptiness for timed automata"). The fact that it is finite is because the number of regions is finite. □

In fact, $\preccurlyeq_M$ is also symmetric. Such a simulation is also known as *bisimulation*. Extending $\preccurlyeq_M$ to sets of valuations gives us the $Closure_M$ abstraction.

Definition 18 ($Closure_M$ abstraction) Let W be a set of valuations. Then:

$$Closure_M(W) := \{v \mid \text{ exists } v' \in W \text{ s.t. } v \sim_M v'\}$$

In other words, $Closure_M(W)$ is the union of regions that intersect W.

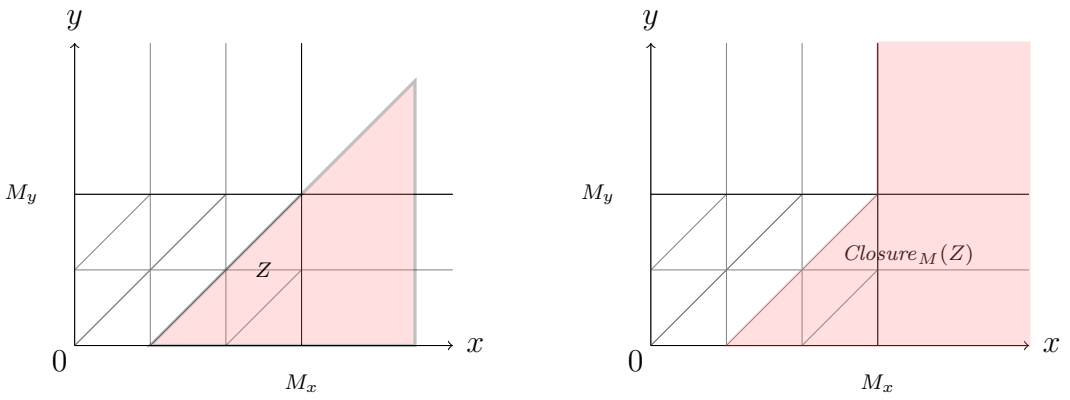


Figure 1.6: A zone and its Closure with respect to the maximum bounds M_x and M_y

Figure 1.6 gives an example of a zone over two clocks and its closure. The thin gray lines give the division of the x-y plane into regions.

We are now in our last step in defining a simulation relation over sets of valuations.

Definition 19 For each state q and each pair of valuation sets W, W' define:

$$(q, W) \sqsubseteq_M (q, W') \text{ if } W \subseteq Closure_M(W')$$

Lemma 20 The relation $\sqsubseteq_M$ defined above is a finite simulation relation.

Proof
From Definitions 19 and 18, it is clear that for every $v \in W$, there exists a $v' \in W'$ such that $v \sim_M v'$. But then, this also means that $v \preccurlyeq_M v'$ (Definition 16). Finally from Lemma 17, we know that $\preccurlyeq_M$ is a simulation. Hence for every $v \in W$ there exists a $v' \in W'$ such that $v \preccurlyeq_M v'$. Therefore, $\sqsubseteq$ is a simulation relation.

A region closure is a subset of regions. As the number of regions is finite, the number of region closures is also finite. □

The above lemma and definition give us a way of plugging in a simulation relation in Algorithm 1.2. We give the final version in Algorithm 1.3.

Example 21 We saw an automaton $\mathcal{A}_{inf}$ in Figure 1.4 for which the zone graph was infinite. For the same automaton, Algorithm 1.3 computes a finite graph as illustrated in Figure 1.7.

5 Summary

In this lecture, we have seen a new procedure (Algorithm 1.3) that uses special sets of valuations called zones and an inclusion checking $Z \subseteq Closure_M(Z')$. The correctness of this algorithm follows from the general theorem about simulation relations (Theorem 15) and the fact that the relation $Z \sqsubseteq_M Z'$ if $Z \subseteq Closure_M(Z')$ is a simulation relation (Lemma 20). However, we have not given an algorithm that can compute $Z \subseteq Closure_M(Z')$ efficiently.

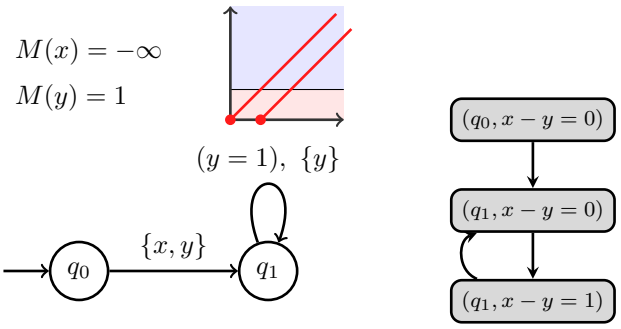


Figure 1.7: Automaton $\mathcal{A}_{inf}$ has finite zone graph when $Closure_M$ is used

Algorithm 1.3: Reachability procedure using $Closure_M$

```
1   Input: Timed automaton A = (Q, X = {x_1, ..., x_k}, q_0, T, Acc)
2   Output: Yes, when some state in Acc is reachable from q_0; No otherwise
3
4   function main()
5
6   let M be the maximum bounds function for A
7
8   Waiting := ∅;
9   Passed := ∅;
10  Z_0 := 0 ≤ x_1 = x_2 = ··· = x_k
11
12  Add (q_0, Z_0) to Waiting
13
14  while (Waiting ≠ ∅)
15      Remove (q, Z) from Waiting
16
17      if (q is accepting)
18          exit Yes
19      else if (∃ (q, Z') ∈ Passed s.t. Z ⊆ Closure_M(Z'))
20          continue
21      else
22          for each (q_s, Z_s) s.t. (q, Z) ⇒ (q_s, Z_s) do
23              if (Z_s ≠ ∅)
24                  Add (q_s, Z_s) to Waiting
25          Add (q, Z) to Passed
26
27  return No
```

This will be the subject of the next two lectures. In the next lecture, we will define some convenient tools that can help understand the operations on zones. In the subsequent lecture, we provide an algorithm that checks $Z \subseteq Closure_M(Z')$ in $\mathcal{O}(|X|^2)$ steps.

References

[BY04] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets*, pages 87–124. Springer, 2004.

[Dil90] David L Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic verification methods for finite state systems*, pages 197–212. Springer, 1990.

[ZLZ05] Jianhua Zhao, Xuandong Li, and Guoliang Zheng. A quadratic-time dbm-based successor algorithm for checking timed automata. *Information processing letters*, 96(3):101–105, 2005.