Tools and Algorithms for Deciding Relations on Timed Automata

B Tech project, supervised by S. Arun-Kumar, verification group

Mihir Mehta

Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

cs1090197@cse.iitd.ac.in

May 13, 2013

Outline

Automata without timing and relations on them

Timed automata and relations on them

Algorithms

Labeled transition systems

Definition

Labelled Transition System: A labelled transition system (LTS) [1] is an automaton which is described by

- S, a set of *states*
- Act, a set of *actions*
- $\rightarrow \subseteq S \times Act \times S$, a *transition relation*.
- optionally, $I \subseteq S$,a set of initial states. If there is exactly one initial state, then the LTS is said to be *rooted*.

Relations on LTS I

Definition

Strong bisimulation:A binary relation R on the states of an LTS is a strong bisimulation if and only if, for all $(s_1, s_2) \epsilon R$ and $a \epsilon Act$.
$\forall s_1'(s_1 \xrightarrow{a} s_1' \Rightarrow \exists s_2'.(s_2 \xrightarrow{a} s_2' \wedge (s_1', s_2') \in R))\wedge$
$\forall s_2'(s_2 \xrightarrow{a} s_2' \Rightarrow \exists s_1'.(s_1 \xrightarrow{a} s_1' \wedge (s_1', s_2') \in R))$

Definition

It can be shown that the union of all strong bisimulations over the set of states is a strong bisimulation. This binary relation is called *strong bisimilarity*, denoted by $\sim$.

Outline

Automata without timing and relations on them

Timed automata and relations on them

Algorithms

Timed Automata

Definition

Timed Automaton: A timed automaton [2] over a finite set of clocks C and a finite set of actions Act is a 4-tuple (L, l_0, E, I).

- L is a finite set of locations.
- l_0 is the initial location.
- $E \subseteq L \times B(C) \times Act \times 2^C \times L$ is a finite set of edges.
- $I : L \rightarrow B(C)$ assigns invariants to each edge location.
- $B(C)$ is the set of clock constraints over C.

Relations on timed automata

Definition

Strong time abstracted bisimulation: A binary relation R over the states of a timed automaton is a strong time abstracted bisimulation (STaB) if and only if, for all $(s_1, s_2) \in R$, $a \in Act$, $d \in R_{\geq 0}$

$$\forall s_1'(s_1 \xrightarrow{a} s_1' \Rightarrow \exists s_2'.(s_2 \xrightarrow{a} s_2' \wedge (s_1', s_2') \in R))\wedge$$
$$\forall s_2'(s_2 \xrightarrow{a} s_2' \Rightarrow \exists s_1'.(s_1 \xrightarrow{a} s_1' \wedge (s_1', s_2') \in R))\wedge$$
$$\forall s_1'(s_1 \xrightarrow{d} s_1' \Rightarrow \exists (s_2', d').(s_2 \xrightarrow{d'} s_2' \wedge (s_1', s_2') \in R))\wedge$$
$$\forall s_2'(s_2 \xrightarrow{d} s_2' \Rightarrow \exists (s_1', d').(s_1 \xrightarrow{d'} s_1' \wedge (s_1', s_2') \in R))$$

It can be shown that the union of all strong time abstracted bisimulations over the set of (location, valuation) pairs is a strong time abstracted bisimulation. This binary relation is called *strong time abstracted bisimilarity*.

Relations on timed automata

Definition

Time abstracted delay bisimulation: A binary relation R over the states of a timed automaton is a time abstracted delay bisimulation (TadB) if and only if, for all $(s_1, s_2) \in R$, $a \in Act$, $d \in R_{\geq 0}$

$$\forall s_1'(s_1 \xrightarrow{a} s_1' \Rightarrow \exists (s_2', d).(s_2 \xrightarrow{d}\xrightarrow{a} s_2' \wedge (s_1', s_2') \in R))\wedge$$
$$\forall s_2'(s_2 \xrightarrow{a} s_2' \Rightarrow \exists (s_1', d).(s_1 \xrightarrow{d}\xrightarrow{a} s_1' \wedge (s_1', s_2') \epsilon R))\wedge$$
$$\forall s_1'(s_1 \xrightarrow{d} s_1' \Rightarrow \exists (s_2', d').(s_2 \xrightarrow{d'} s_2' \wedge (s_1', s_2') \in R))\wedge$$
$$\forall s_2'(s_2 \xrightarrow{d} s_2' \Rightarrow \exists (s_1', d').(s_1 \xrightarrow{d'} s_1' \wedge (s_1', s_2') \in R))$$

It can be shown that the union of all time abstracted delay bisimulations over the set of (location, valuation) pairs is a time abstracted delay bisimulation. This binary relation is called *time abstracted delay bisimilarity*.

Relations on timed automata

Definition

Time abstracted observational bisimulation: A binary relation R over the states of a timed automaton is a time abstracted observational bisimulation (TaoB) if and only if, for all $(s_1, s_2) \in R$, $a \in Act$, $d \in R_{\geq 0}$

$$\forall s_1'(s_1 \xrightarrow{a} s_1' \Rightarrow \exists(s_2', d, d').(s_2 \xrightarrow{d}\xrightarrow{a}\xrightarrow{d'} s_2' \wedge (s_1', s_2') \in R))\wedge$$
$$\forall s_2'(s_2 \xrightarrow{a} s_2' \Rightarrow \exists(s_1', d, d').(s_1 \xrightarrow{d}\xrightarrow{a}\xrightarrow{d'} s_1' \wedge (s_1', s_2') \in R))\wedge$$
$$\forall s_1'(s_1 \xrightarrow{d} s_1' \Rightarrow \exists(s_2', d').(s_2 \xrightarrow{d'} s_2' \wedge (s_1', s_2') \in R))\wedge$$
$$\forall s_2'(s_2 \xrightarrow{d} s_2' \Rightarrow \exists(s_1', d').(s_1 \xrightarrow{d'} s_1' \wedge (s_1', s_2') \in R))$$

It can be shown that the union of all time abstracted observational bisimulations over the set of (location, valuation) pairs is a time abstracted observational bisimulation. This binary relation is called *time abstracted observational bisimilarity*.

Difference bound matrices

Definition

Difference bound matrix: A difference bound matrix (DBM) is a representation of a convex polyhedron on a set of clocks $\{x_1, \ldots x_n\}$ in the form of an $(n+1) \times (n+1)$ matrix M, each element of which takes the form $(m_{ij}, \smile_{ij})$, where m_{ij} is an integer and $\smile_{ij} \, \epsilon \, \{<, \leq\}$. Assuming x_0 to be a clock always valued at zero, the associated polyhedron is given by

$$\bigcap_{0 \leq i,j \leq n} (x_i - x_j \smile_{ij} m_{ij})$$

Abstractions

Abstractions serve to cap the number of zones in a zone graph by reducing zones to equivalent regions which contain them, thus ensuring termination of zone creation algorithms which use forward analysis.

In this implementation we use a simplified version of the *maximum constants* abstraction described in [3].

Algebraically, our abstraction can be thus described: given a set of clocks $\{x_i | 1 \leq i \leq n\}$, a maximum constant k over all clocks, and a DBM $M = \langle (m_{ij}, \smile_{ij}) \rangle_{0 \leq i,j \leq n}$, we can replace M with $M' = \langle m'_{ij}, \smile'_{ij} \rangle_{0 \leq i,j \leq n}$ where

$$(m'_{ij}, \smile'_{ij}) = \begin{cases} (\infty, <) & \text{if } m_{ij} > k \\ (-k, <) & \text{if } m_{ij} < -k \\ m_{ij}, \smile_{ij} & \text{if } -k \leq m_{ij} \leq k \end{cases}$$

Outline

Creating the minimal zone graph

- Origin of this algorithm - [4]
- Minimal zone graph - stability, reachability, minimality.
- Strategy: forward propagation for reachability and stability, backward propagation for stability.

Creating the minimal zone graph - forward analysis

- We use a queue, as in BFS, but we may visit a location multiple times unlike BFS, and unreachable locations may never be visited.
- Each queue element stores a location l_{succ}, its predecessor in the current path l_{pred}, and the transition t from l_{pred} to l_{succ}. It
- Each time a location is visited, we create therein, new zones which are reachable from the zones of the predecessor. Thus, for each predecessor zone $(l_{pred}, \zeta_{pred_i})$, the derived successor zone is $(l_{succ}, \zeta_{succ_i})$, where

$$\zeta_{succ_i} = ((\zeta_{pred_i} \uparrow \cap \mathtt{t.guard})[\mathtt{t.resets} := 0]) \uparrow$$

Creating the minimal zone graph - forward analysis

- Thus, if we let $(l_{pred}, \zeta_{pred_i})$ range over the existing zones of l_{pred}, and if we let $(l_{succ}, \zeta_{succ_j})$ range over the existing zones of l_{succ}, then the new zones in l_{succ} will cover

$$\zeta_{succ_{new}} = (\bigcup_i \zeta_{succ_i}) - (\bigcup_j \zeta_{succ_j})$$

- Since $\zeta_{succ_{new}}$ is not necessarily convex, we may need to split it into multiple convex polyhedra before creating the corresponding zones in the l_{succ}.
- We may also need to split it further in order for the zones to be stable with respect to its outgoing edge guards, and also in order to filter out zones which do not satisfy its invariant.
- We enqueue all the successors if any new zones are created this way, or if we are visiting the initial location for the first time.
- We terminate when the queue is empty.

Creating the minimal zone graph - backward analysis

- We iterate through the transitions of the timed automaton, multiple times if necessary, splitting the zones of the source of each transition with respect to the zones of the transition's target, until we achieve stability of each zone of each location.
- We recall that for stability, whenever we have an edge in the zone valuation graph from a zone (l_{pred}, ζ_{pred}) to (l_{succ}, ζ_{succ}) corresponding to a transition t in the timed automaton, we require

$$\zeta_{pred} \subseteq \texttt{t.guard} \wedge \zeta_{pred}[\texttt{t.resets} := 0] \subseteq \zeta_{succ}$$

Thus, when this does not hold, we split (l_{pred}, ζ_{pred}) into

$$(l_{pred}, \zeta_{pred} \cap (\texttt{t.guard} \cap [\texttt{t.resets} := 0]\zeta_{succ}))$$

(which is convex and has an edge to (l_{succ}, ζ_{succ})) and

$$(l_{pred}, \zeta_{pred} - (\texttt{t.guard} \cap [\texttt{t.resets} := 0]\zeta_{succ}))$$

(which does not have an edge to (l_{succ}, ζ_{succ}) and may need to be split into convex zones.)
This generates the zone valuation graph.

Creating the minimal zone graph - forward analysis

Initialise the queue Q with a single element $(null, null, l_0)$;
Initialise the graph *zone_graph* with a single node $(l_0, v_0\uparrow)$ with an ϵ self-loop;
while *Q is not empty* **do**
    Dequeue $(l_{parent}, t, l_{child})$ from Q;
    if $l_{parent} \neq null$ **then**
        foreach *zone Z_{parent} of l_{parent}* **do**
            Add new zones to the zones of l_{child} so that all zones reachable from Z_{parent} are represented;
            Abstract if necessary;
            Update edges from Z_{parent} to the new zones of l_{child} **if** *new zones are created in l_{child} or l_{parent} is null* **then**
                foreach *outgoing transition t' of l_{child}* **do**
                    Enqueue $(l_{child}, t', \texttt{t'.target})$ in Q;
                end
            end
        end
    end
end

Creating the minimal zone graph - backward analysis

```
Set new_zone;
while new_zone do
    Reset new_zone;
    foreach transition t in the timed automaton do
        Split the zones of t.source to be stable with respect to the zones of
        t.target;
        Update edges accordingly;
        if new zones are created in t.source then
            Set new_zone;
        end
    end
end
Generate minimal_zone_graph by applying Fernandez' algorithm to zone_graph;
Return zone_graph;
```

Minimal zone graph example

Figure: Timed automaton. Here, the states are $\{0\}$, the actions are $\{a\}$, and the clocks are $\{X, Y\}$.

Minimal zone graph example

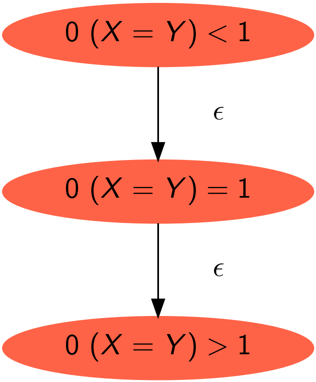


Figure: Zones after one iteration.

Minimal zone graph example

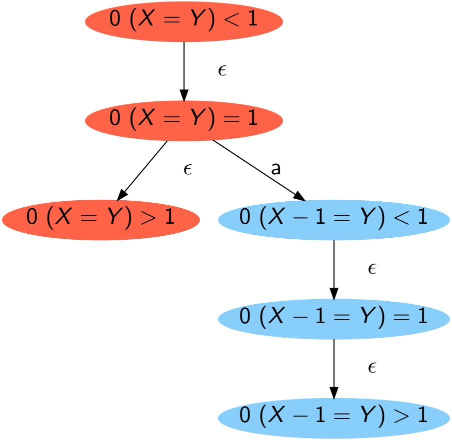


Figure: Zones after two iterations.

Minimal zone graph example

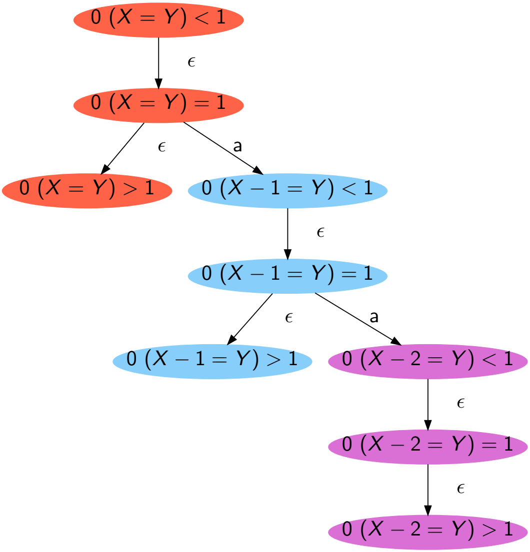


Figure: Zones after three iterations without abstraction.

Minimal zone graph example

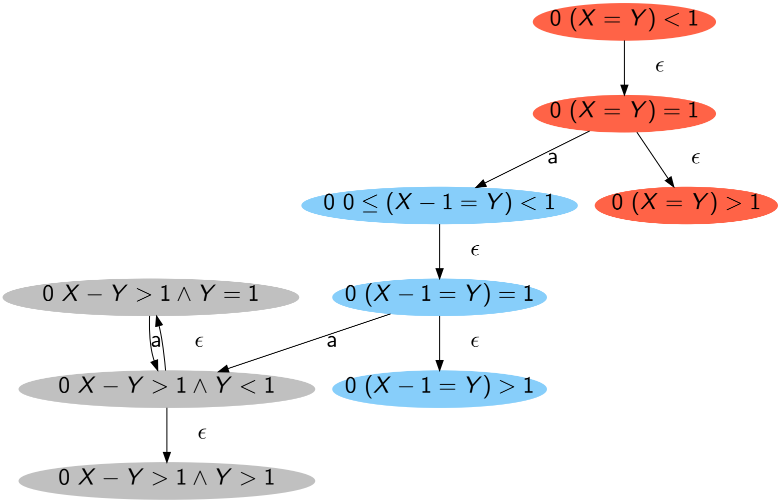


Figure: Zones after three iterations with abstraction.

Minimal zone graph example

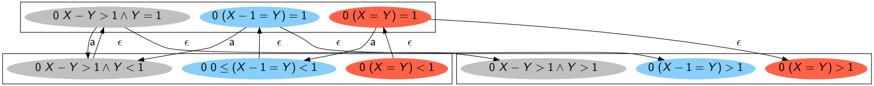


Figure: Zone graph with bisimilarity classes.

Verifying relations on pairs of timed automata.

- Origin for this algorithm - [5]
- General method to compute (ρ, σ)-bisimilarities on two LTS, starting from their initial locations.
- Can be adapted for a certain class of timed and time abstracted relations by using zone valuation graphs.
- For every relation R satisfying this property, functions f_P and f_Q must exist such that the proposition $s_P R s_Q$ resolves to one of these:
  - yes
  - no
  - if and only if

$$\forall (s'_P, L'_Q) \in f_P(s_P) : \exists s'_Q \in L'_Q : s'_P R s'_Q \quad \wedge$$
$$\forall (L'_P, s'_Q) \in f_Q(s_Q) : \exists s'_P \in L'_P : s'_P R s'_Q$$

Verifying relations on pairs of timed automata.

- For STaB, we define f_P and f_Q as

$$\begin{aligned} f_P(s_P) =& \{(s'_P, L'_Q) | s_P \xrightarrow{a} s'_P, L_Q = \{s'_Q | s_Q \xrightarrow{a} s'_Q\}\} \\ & \cup \{(s'_P, L'_Q) | s_P \xrightarrow{\epsilon} s'_P, L_Q = \{s'_Q | s_Q \xrightarrow{\epsilon} s'_Q\}\} \\ f_Q(s_Q) =& \{(L'_P, s'_Q) | s_Q \xrightarrow{a} s'_Q, L_P = \{s'_P | s_P \xrightarrow{a} s'_P\}\} \\ & \cup \{(L'_P, s'_Q) | s_Q \xrightarrow{\epsilon} s'_Q, L_P = \{s'_P | s_P \xrightarrow{\epsilon} s'_P\}\} \end{aligned}$$

- For TadB, we define f_P and f_Q as

$$\begin{aligned} f_P(s_P) =& \{(s'_P, L'_Q) | s_P \xrightarrow{a} s'_P, L_Q = \{s'_Q | s_Q \xrightarrow{\epsilon}\xrightarrow{a} s'_Q\}\} \\ & \cup \{(s'_P, L'_Q) | s_P \xrightarrow{\epsilon} s'_P, L_Q = \{s'_Q | s_Q \xrightarrow{\epsilon} s'_Q\}\} \\ f_Q(s_Q) =& \{(L'_P, s'_Q) | s_Q \xrightarrow{a} s'_Q, L_P = \{s'_P | s_P \xrightarrow{\epsilon}\xrightarrow{a} s'_P\}\} \\ & \cup \{(L'_P, s'_Q) | s_Q \xrightarrow{\epsilon} s'_Q, L_P = \{s'_P | s_P \xrightarrow{\epsilon} s'_P\}\} \end{aligned}$$

- For TaoB, we define f_P and f_Q as

$$\begin{aligned} f_P(s_P) =& \{(s'_P, L'_Q) | s_P \xrightarrow{a} s'_P, L_Q = \{s'_Q | s_Q \xrightarrow{\epsilon}\xrightarrow{a}\xrightarrow{\epsilon} s'_Q\}\} \\ & \cup \{(s'_P, L'_Q) | s_P \xrightarrow{\epsilon} s'_P, L_Q = \{s'_Q | s_Q \xrightarrow{\epsilon} s'_Q\}\} \\ f_Q(s_Q) =& \{(L'_P, s'_Q) | s_Q \xrightarrow{a} s'_Q, L_P = \{s'_P | s_P \xrightarrow{\epsilon}\xrightarrow{a}\xrightarrow{\epsilon} s'_P\}\} \\ & \cup \{(L'_P, s'_Q) | s_Q \xrightarrow{\epsilon} s'_Q, L_P = \{s'_P | s_P \xrightarrow{\epsilon} s'_P\}\} \end{aligned}$$

Verifying relations on pairs of timed automata.

```
begin
  if lookup(yes_table, s_P, s_Q) then
    return true;
  else
    if lookup(yes_table, s_P, s_Q) then
      return false;
    else
      insert(yes_table, s_P, s_Q);
      Set v_P;
      foreach (s'_P, L'_Q) in f_P(s_P) do
        Reset v_P;
        foreach s'_Q in L'_Q do
          if CheckStatesRelation(P, Q, s'_P, s'_Q, yes_table, no_table) then
            Set v_P;
          end
        end
      end
      Set v_Q;
      foreach (s'_Q, L'_P) in f_Q(s_Q) do
        Reset v_Q;
        foreach s'_P in L'_P do
          if CheckStatesRelation(P, Q, s'_P, s'_Q, yes_table, no_table) then
            Set v_Q;
          end
        end
      end
      if v_P ∧ v_Q then
        return true;
      else
        remove(yes_table, s_P, s_Q);
        insert(yes_table, s_P, s_Q);
        return false;
      end
    end
  end
end
```

Procedure CheckStatesRelation(P, Q, s_P, s_Q, *yes_table*, *no_table*)

Verifying relations on pairs of timed automata.

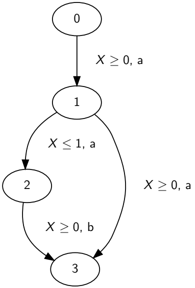
begin

Create zone valuation graphs G_P, G_Q of T_P, T_Q;
Find the zone s_P in G_P which contains the initial state of T_P;
Find the zone s_Q in G_Q which contains the initial state of T_Q;
Initialise *yes_table* and *no_table* to empty tables;
return `CheckStatesRelation`(G_P, G_Q, s_P, s_Q, *yes_table*, *no_table*);

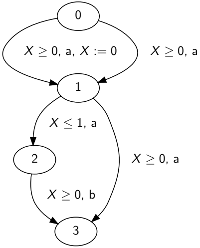
end

Procedure CheckAutomataRelation(T_P, T_Q)

Verifying relations: example



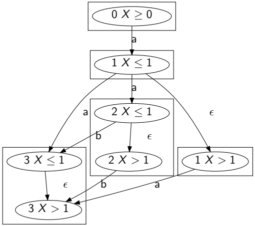
(a) First



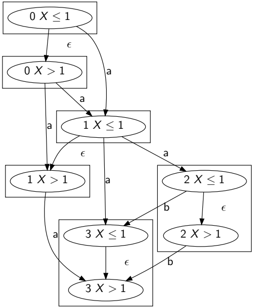
(b) Second

Figure: Timed automata.

Verifying relations: example



(a) First



(b) Second

Figure: Zone valuation graphs.

References I

Keller, R.M.:
Formal verification of parallel programs.
Commun. ACM **19**(7) (July 1976) 371–384

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

Behrmann, G., Bouyer, P., Fleury, E., Larsen, K.G.:
Static guard analysis in timed automata verification.
In: In TACAS, Springer (2003) 254–277

Guha, S., Narayan, C., Arun-Kumar, S.:
On decidability of prebisimulation for timed automata.
In Madhusudan, P., Seshia, S.A., eds.: CAV. Volume 7358 of Lecture Notes in Computer Science., Springer (2012) 444–461

References II

Arun-Kumar, S.:
On bisimilarities induced by relations on actions.
In: Software Engineering and Formal Methods, 2006. SEFM 2006. Fourth IEEE International Conference on, IEEE (2006) 41–49