Tools and Algorithms for Deciding Timed Relations

Mihir Mehta

Department of Computer Science and Engineering,
Indian Institute of Technology, Delhi.
`cs1090197@cse.iitd.ac.in`

April 2013

Abstract

This is a report summarising the author's project on their B Tech Project for the academic year 2012-2013.

1 Objectives

The work described in this thesis builds on the work of [1] in which the authors described an algorithm method to generate *zone-valuation graphs* for timed automata and an algorithm to use such zone-valuation graphs to implement an on-the-fly analysis to determine *timed performance prebisimilarity* on pairs of timed automata. Our aim was to implement these algorithms in a generalised manner in order to verify various other timed relations, such as time abstracted bisimulations [2] and timed simulation equivalence. Towards this end, we studied the literature about various kinds of timed and untimed automata as well as various existing tools for verifying these equivalences (such as `minim`, described in [2]. Our implementation, in OCaml, implements several of these relations and leaves some scope for implementing others.

2 Timed automata

start → q_a

press, $x > 1.4$

press, $x := 0$

q_b press

press, $x \leq 1.4$

q_c
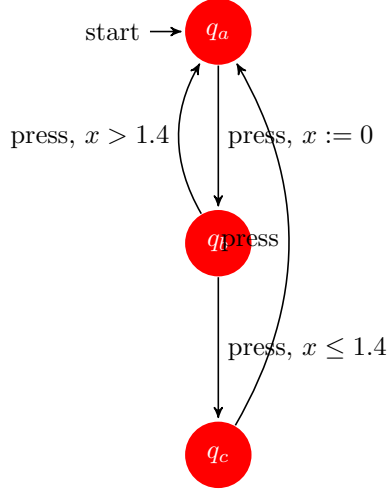
Timed automaton representing a light bulb with two brightness settings, example taken from [3]

- Formally, a timed automaton 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. An element of $B(C)$ can be an equality, a slack inequality, or a strict inequality on $v(x)$ for some $x \ \epsilon$ C, or an AND combination of such constraints.

- The state of the automaton at any particular instant is given by the ordered pair (l, v) which gives the location and assigns a value to each clock.

- A transition is either a delay transition where the automaton stays at the same location while advancing each clock by the same time delay, or an action transition where the automaton performs a state change while resetting some of its clocks.

3 Timed relations on timed automata

3.1 Time abstracted bisimilarity

- *Strong time abstracted bisimulation*: A binary relation R is a strong time abstracted bisimulation (STaB) if and only if, for all $(s_1, s_2) \, \epsilon \, R$, $a \, \epsilon \, Act$, $d \, \epsilon \, R_{\geq 0}$

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

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

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

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

- 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*.

- *Time abstracted delay bisimulation*: A binary relation R is a time abstracted delay bisimulation (TadB) if and only if, for all $(s_1, s_2) \, \epsilon \, R$, $a \, \epsilon \, Act$, $d \, \epsilon \, R_{\geq 0}$

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

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

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

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

- 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*.

- *Time abstracted observational bisimulation*: A binary relation R is a time abstracted observational bisimulation (TaoB) if and only if, for all $(s_1, s_2) \, \epsilon \, R$, $a \, \epsilon \, Act$, $d \, \epsilon \, R_{\geq 0}$

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

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

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

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

- 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*.

4   Abstractions

5   Algorithms

5.1   Creation of the zone valuation graph

This algorithm is adapted from the algorithm in [1]. We changed it to make the correctness more evident. In summary, the algorithm consists of a *forward propagation* which ensures that all *reachable* zones are created, and a *backward propagation* which ensures that each zone is *stable* with respect to its successors. For the forward propagation, we use a queue, akin to the queue of the classic *breadth-first search* algorithm for graphs, to traverse the timed automaton, starting from the inital location, to ensure that all reachable zones are created. In each queue element (with the exception of the first element containing the inital location, which has no predecessor), we store a location l_{succ}, its predecessor in the current path l_{pred}, and the transition t from l_{pred} to l_{succ}. It should be noted that this may result in some locations being visited multiple times, and in unreachable locations never being visited. 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

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

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}. Then, we split the zones of l_{succ} to ensure stability with respect to its invariant and outgoing edge constraints. If any new zones are thus created, we enqueue each of the location's successors, in order to ensure that all reachable zones are created. The forward propagation ends when the queue is empty. In the backward propagation, 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. This generates the zone valuation graph.

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 $(l_{parent}, transition, l_{child})$ 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} (given by the equation $Z_{child} = (((Z_{parent} \uparrow$
            $) \cap transition.guard)[transition.resets := 0]) \uparrow)$ 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} **then**
                foreach *outgoing transition t of l_{child}* **do**
                    Enqueue $(l_{child}, t, t.target)$ in Q;
                end
            end
        end
    end
    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 $t.target$, so that each source zone Z_{parent} with an
            edge to a target zone Z_child satisfies
            $Z_{parent} \subseteq t.guard \cap ([t.resets := 0]Z_{child})$;
            Update edges accordingly;
            if *new zones are created in $t.source$* **then**
                Set new_zone;
            end
        end
    end
end
Return $zone\_graph$;

- **Termination**: The algorithm, in the worst case, will create as many zones as there are regions in the region graph, as the region graph abstraction ensures that the number of zones cannot exceed the number of regions. Since the number of regions is known to be bounded, termination of the algorithm is guaranteed.

- **Reachability**: Since the initial zone is the future of the zero valuation in the initial zone, it is reachable by definition. A new zone is only created when it is reachable from some zone which has already been created, thus

5

each zone which is created is reachable by induction.

- **Stability**: Since the termination of the backward propagation step only occurs after an iteration in which all the edges of the timed automaton are traversed without causing any splitting of states, it follows that the zone graph is stable with respect to itself after this last iteration.

References

[1] S. Guha, C. Narayan, and S. Arun-Kumar, "On decidability of prebisimulation for timed automata," in *CAV* (P. Madhusudan and S. A. Seshia, eds.), vol. 7358 of *Lecture Notes in Computer Science*, pp. 444–461, Springer, 2012.

[2] S. Tripakis and S. Yovine, "Analysis of timed systems using time-abstracting bisimulations," *Formal Methods in System Design*, vol. 18, no. 1, pp. 25–68, 2001.

[3] L. Aceto, A. Ingólfsdóttir, K. G. Larsen, and J. Srba, *Reactive systems: modelling, specification and verification.* Cambridge University Press, 2007.