Tools and Algorithms for Deciding Relations on Timed Automata

Mihir Mehta

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

Abstract. This describes the author's work in implementing the construction of zone-valuation graphs for timed automata and using these to verify certain relations on pairs of timed automata. Timed automata are a widely-used formalism, in verification, for describing systems where the execution is subject to timing constraints. One area of interest is the verification of certain timed and time abstracted relations on pairs of timed automata, which are generally of either an equivalence nature or a pre-ordering nature. Among existing tools, minim [1] serves to verify certain of these relations, but it is limited in its scope and fails to meet certain requirements of reachability in its output. We build upon the work of [2], in which algorithms for building zone-valuation graphs (minimal, discrete equivalent representations of timed automata) are given and used to verify certain pre-ordering relations on pairs of timed automata. We present an modified version of the zone-valuation graph construction algorithm, and a generalisation of the algorithm which uses zone-valuation graphs to verify a larger set of timed and time-abstracted relations on timed automata, where this generalisation builds upon the work of [3]. We also describe a reference implementation of these algorithms in Ocaml.

Table of Contents

List of Figures

1 Introduction

The work described in this thesis builds on the work of [2] in which Guha et al described an algorithm to generate *zone valuation graphs* for timed automata and an algorithm to use such zone-valuation graphs 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 time abstracted relations, such as time abstracted bisimulations [1] and time abstracted simulation equivalence. Towards this end, we studied the literature about timed automata as well as various existing tools for verifying these equivalences (such as `minim`, described in [1]). Our implementation, in OCaml, implements several of these relations and leaves some scope for implementing others.

This document proceeds by developing the relevant theory for labelled transition systems and strong bisimilarity, and then continues with timed automata and time abstracted bisimilarity in analogous fashion. We then detail several methods to discretise the state space of timed automata, including region graphs, zone graphs, and zone valuation graphs, and use region graphs to show that a finite representation of the state space of a timed automaton is always decidable. We mention difference bound matrices (DBM), a widely used representation for the time constraints (i. e. convex polyhedra) that are used to describe zones. Then, we explain abstractions, which are transformations on DBM that replace each DBM by a DBM which is its superset and which is equivalent to it modulo strong time abstracted bisimilarity. This helps us prevent state space explosion in our exploration algorithms which build the state space of a timed automaton. We then explain the common features of certain time abstracted relations which allow us to verify them in a generalised manner, and explain our implementation.

2 Labelled transition systems

Definition 1. Labelled Transition System*: A labelled transition system (LTS) [6] 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.

LTS are useful for describing the behaviour of untimed systems, and serve as the foundation for the development of more complex models such as CCS and timed automata. Thus, equivalences on LTS serve as the theoretical foundation for many timed and time abstracted equivalences on timed automata, and also have direct applications in determining some of these equivalences in cases where timed automata can be reduced to equivalent LTS.

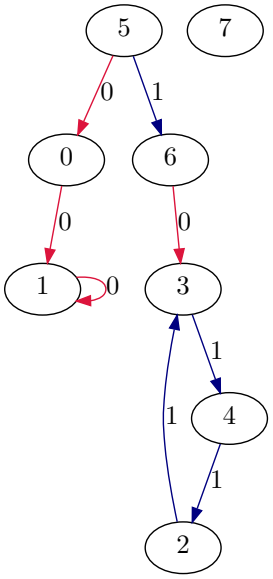


Fig. 1: An example of a labelled transition system. Here, the states are $\{0, 1, 2, \ldots 7\}$ and the actions are $\{0, 1\}$.

3 Equivalences on labelled transition systems

3.1 Strong bisimilarity

A binary relation R is a *strong bisimulation* if and only if, for all $(s_1, s_2) \in R$ and $a \in Act$.
$\forall s_1'(s_1 \xrightarrow{a} s_1' \Rightarrow \exists s_2'.(s_2 \xrightarrow{a} s_2' \wedge (s_1', s_2') \in R)) \wedge$
$\forall s_2'(s_2 \xrightarrow{a} s_2' \Rightarrow \exists s_1'.(s_1 \xrightarrow{a} s_1' \wedge (s_1', s_2') \in R))$

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 $\sim$.

Strong bisimilarity between two states in an LTS implies, intuitively, that any action performed by the one can be performed by the other and vice versa.

An algorithm to evaluate strong bisimilarity was first given by Kanellakis and Smolka [7]. A faster algorithm for the special case having just one kind of action was given by Paige and Tarjan [8] and made general by Fernandez [9].

4 Clocks, valuations and related operations

The timing model we use relies on the notions of clocks and valuations. The timing information about the possible executions of a timed automaton can, intuitively, be expressed in terms of constraints on clocks. We will make this idea more precise as we go on.

Definition 2. Clock*: A clock is a variable ranging over the set of non-negative real numbers,* $R_{\geq 0}$.

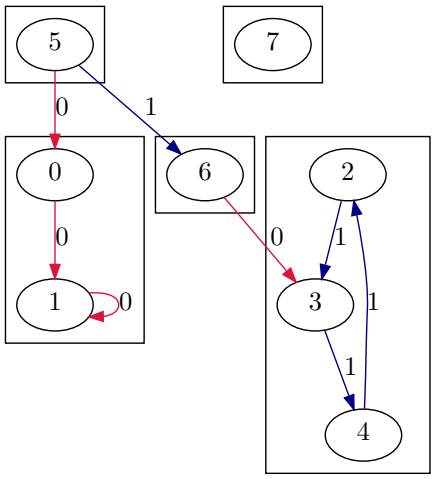


Fig. 2: Strong bisimilarity quotient of the LTS in Figure 1.

Definition 3. Valuation*: A valuation over a set of clocks C is a function $C \to R_{\geq 0}$, assigning non-negative values to each of the clocks.*

Two important operations on clock valuations are clock resets and delays. For a subset X of C, the clock reset of a valuation v is given by

$$v[X := 0](x) = \begin{cases} 0 & \text{if } x \epsilon X \\ v(x) & \text{if } x \epsilon C - X \end{cases}$$

For a real delay $d \, \epsilon \, R_{\geq 0}$, the delay of a valuation v is given by

$$(v + d)(x) = v(x) + d \quad \forall x \epsilon C$$

Definition 4. Hyperplane*: Expressions of the forms $x \backsim c$ and $x - y \backsim c$, where $x, y \epsilon C$, c is an integer, and $\backsim \epsilon \{<, \leq, \geq, >\}$ are known as* atomic clock constraints*. A valuation v over C is said to satisfy the atomic clock constraint $x \backsim c$ if $v(x) \backsim c$ and $x - y \backsim c$ if $v(x) - v(y) \backsim c$. A hyperplane is a set of valuations satisfying an atomic clock constraint.*

Definition 5. Polyhedron*: The set of polyhedra over a set of clocks C is the smallest subset of the set of all valuations over C that*

- *contains all hyperplanes over C*
- *is closed over intersection, union and complementation.*

Definition 6. Convex polyhedron*: A convex polyhedron is a polyhedron that can be represented as the intersection of a set of hyperplanes.*

It should be noted that this definition does not coincide exactly with the notion of a convex set from algebra. For instance, for $C = \{x_1, x_2\}$, the set $\{(x_1, x_2) | x_1 \geq 0, x_2 \geq 0, x_1 + x_2 > 0\}$ is convex but not a convex polyhedron.

Union, intersection, complementation and set difference are well defined operations on polyhedra, but of these, intersection is the only operation which preserves convexity.

The *pre-reset* and *post-reset* of a set $X \epsilon C$ on a polyhedron ζ over the set C of clocks are respectively defined by

$$[X := 0]\zeta = \{v | v[X := 0]\epsilon\zeta\}$$
$$\zeta[X := 0] = \{v[X := 0] | v\epsilon\zeta\}$$

The *future* of a polyhedron ζ over a set C of clocks is given by

$$\zeta \uparrow = \{v + d | v\epsilon\zeta, d\epsilon R_{\geq 0}\}$$

The operators intersection, future, pre-reset, post-reset and future preserve convexity of polyhedra.

Definition 7. Clock constraint*: A clock constraint is a polyhedron which can be expressed as an intersection of hyperplanes of the form* $x \smile c$.

A related term, *extended clock constraint*, can be used interchangeably with *polyhedron.*

5 Timed automata

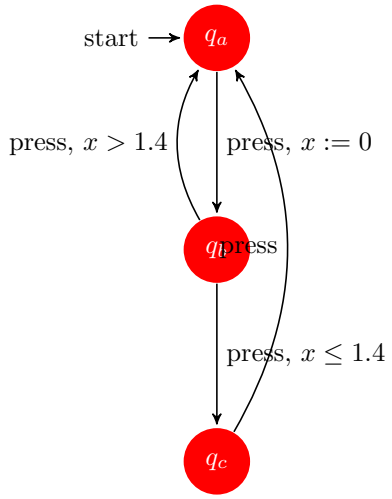


Fig. 3: Timed automaton representing a light bulb with two brightness settings, example taken from [4]

Definition 8. Timed Automaton*: A timed automaton [10] 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.*
- $I : L \to B(C)$ *assigns invariants to each edge location.*
- $B(C)$ *is the set of clock constraints over* C*.*

This is a useful formalism for describing the behaviour of a system with timing requirements on its behaviour. For a better understanding of its semantics, it is useful to describe a labeled transition system corresponding to it (called a timed LTS, or TLTS) that describes the behaviour of this timed automaton.

Definition 9. Timed LTS*: We define the timed LTS* $T(A)$ *defined by a timed automaton* (L, l_0, E, I) *over a set of clocks* C *and a set of actions* Act *as*

$$T(A) = (Proc, Lab, \to)$$

where

- $Proc = \{(l, v) | l \in L, v \in I(l)\}$
- $Lab = Act \cup R_{\geq 0}$
- $\to$ *is given by*
  - $(l, v) \xrightarrow{a} (l', v')$ *iff* $a \in Act$, $\exists (l \xrightarrow{g,a,r} l') \in E$ *such that* $v \models g \wedge v' = v[r := 0] \wedge v' \models I(l')$
  - $(l, v) \xrightarrow{d} (l, v')$ *iff* $d \in R_{\geq 0} \wedge v \models I(l) \wedge v' \models I(l)$

We will use the object-oriented notation `l.invar` to refer to the invariant $I(l)$ of a location l and `e.source`, `e.guard`, `e.action`, `e.resets` and `e.target` to refer to l, g, a, r, l' of an edge $e = l \xrightarrow{g,a,r} l'$ unless otherwise noted.

6 Region graphs

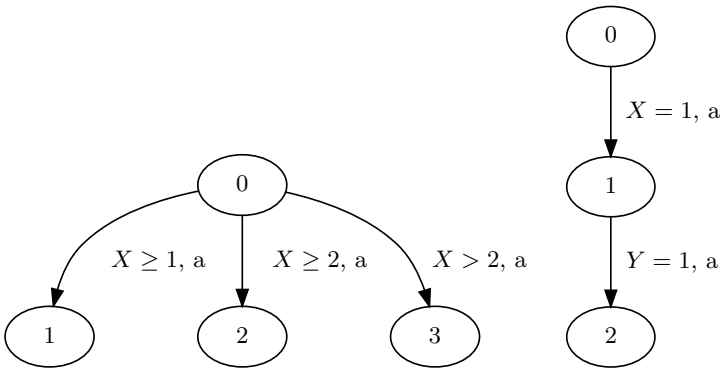
The notion of a region graph was first introduced in [10] and used to show that discretisation of the state space of a timed automaton is decidable under a certain abstraction of exact timing values while examining the possible states and actions of a timed automaton.

To motivate this, we should note that in figure 5a, we can distinguish between the valuations $[X = 1.1]$, $[X = 2.0]$ and $[X = 2.2]$ at location 0 on the basis of the possibility of their outgoing a-transitions: the first can only move to 1, the second can move to 1 and 2, and the third can move to 1, 2, and 3.

Also, in figure 5b, we can distinguish between valuations $[X = 0.6, Y = 0.4]$ and $[X = 0.4; Y = 0.6]$ at location 0 since the first can take a delay of 0.4 units, then an a-transition to 1, then a delay of 0.2, then an a-transition to 2, but the second cannot move to 2 since it must take a delay of 0.6 to move to 1 which leaves no possibility of it moving to 2 after any amount of delay.

For each clock x, we define c_x to be the greatest constant compared to x over all guards and invariants in the automaton. It is evident that any two valuations

Fig. 4: Motivating examples for region graphs.



which both assign values greater than c_x to x should not be distinguished, but we should distinguish between valuations differing in $\lfloor v(x) \rfloor$ x for any clock x, between valuations having the same $v(x)$ but in which one has $frac(v(x)) = 0$ but the other does not, and between valuations which have different relative orderings of $v(x)$ and $v(y)$ for any pair of clocks (x, y). We formalise these notions in this definition.

Definition 10. Region graph*: A region graph is a partitioning of the set of valuations for a given timed automaton into equivalence classes under* region equivalence*.*

Valuations v_1 *and* v_2 *are related under region equivalence if*

- $\forall x \in C$*:* $(\lfloor v_1(x) \rfloor = \lfloor v_2(x) \rfloor) \leq c_x$ *or* $v_1(x) > c_x$ *and* $v_2(x) > c_x$.
- $\forall x \in C$*:* $v_1(x) = 0$ *iff* $v_2(x) = 0$.
- $\forall$ *pairs of clocks* (x, y)*:* $frac(v_1(x)) \leq frac(v_1(y))$ *iff* $frac(v_2(x)) \leq frac(v_2(y))$.

It is evident that any two states sharing a location and satisfying region equivalence of their states are going to, under an abstraction of exact values of time delays (a notion we will formalise in 16). This result is important as it shows that there are finitely many categories of states (which we will talk about in 13) which we need to consider.

7 Zone graphs

It is evident that the state space in a timed automaton is, in general, uncountably infinite. However, we have seen that region graphs can be used to make the state space finite, but at the cost of state explosion, since the number of regions is generally exponential in the number of clocks. Some alternative, less expensive methods for discretising the state space follow.

Definition 11. State*: A state of a timed automaton is a pair* (l, v) *where* l *is a location in the automaton and* v *is a clock valuation satisfying* `l.invar`*.*

Definition 12. Symbolic state*: A symbolic state is a set of states in the timed automaton. The constituent states do not necessarily share a location.*

Definition 13. Zone*: A zone is a symbolic state where the all the constituent states share a location and the set of valuations of these states forms a convex polyhedron on the valuation space.*

Definition 14. Zone graph*: A zone graph of a timed automaton is an LTS, where the states are zones, the actions consist of the actions of the timed automaton and an* ϵ *action, which represents a timed transition, and the transition relation respects the invariants, guards and clock resets of the original timed automaton.*

8 Difference bound matrices

Definition 15. 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* $(m_{ij}, \backsim_{ij})$*, where* m_{ij} *is an integer and* $\backsim_{ij} \epsilon \{<, \leq\}$*. Assuming* x_0 *to be a clock always valued at zero, the associated polyhedron is given by*

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

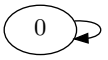
DBM offer a convenient method to represent polyhedra for most of the common operations required on zones, including intersection, clock resets, future, and abstraction. The UPPAAL DBM libraries [11] [12] implement many common functions on DBM and have been extensively used in our implementation.

9 Abstraction

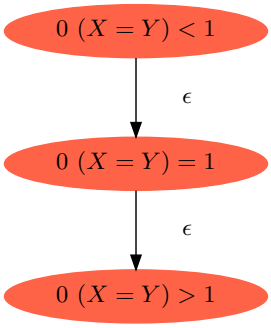
A common feature of algorithms that generate zone graphs for timed automata is a *forward propagation* step in which the algorithm attempts to create reachable zones in reachable locations by traversing the timed automaton. However, this introduces a vulnerability to certain pathological cases in which the number of zones expands indefinitely, preventing termination of the algorithm. For example, in the automaton in Figure 6a, the number of zones may expand in each iteration, as shown in Figure 6b, Figure 6c, Figure 6d.

However, this is inconsistent with what we know about region graphs and their implication of finiteness for the state spaces of timed automata, thus, we have abstractions which 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.

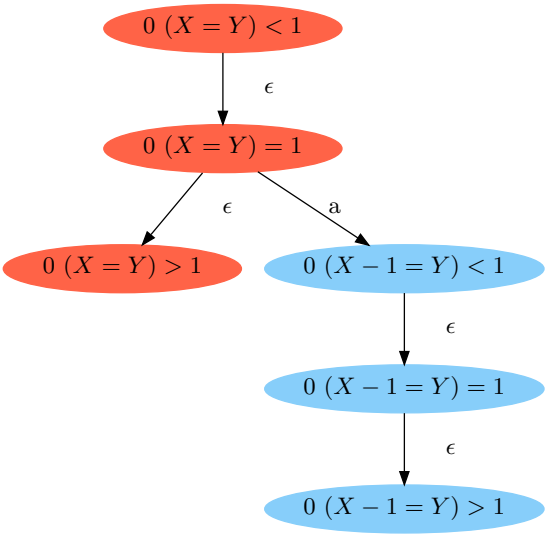
Fig. 5: Timed automaton with a potentially infinite set of zones, example taken from [5].



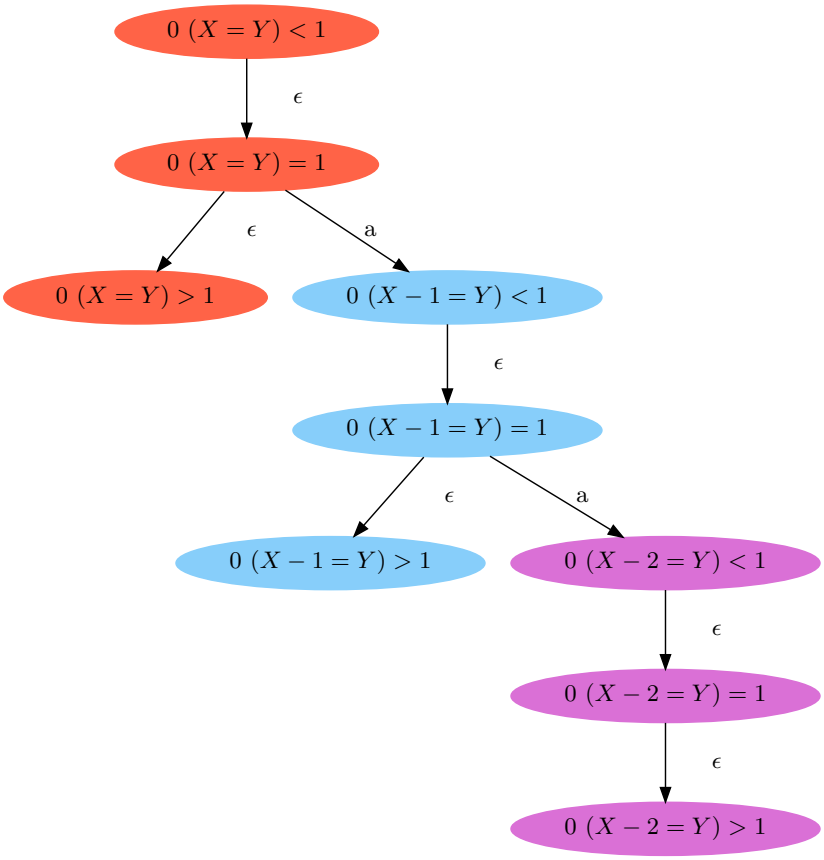
Timed automaton with potentially infinite state space.



Zones of Figure 6a after 1 iteration.



Zones of Figure 6a after 2 iterations.



Zones of Figure 6a after 3 iterations.

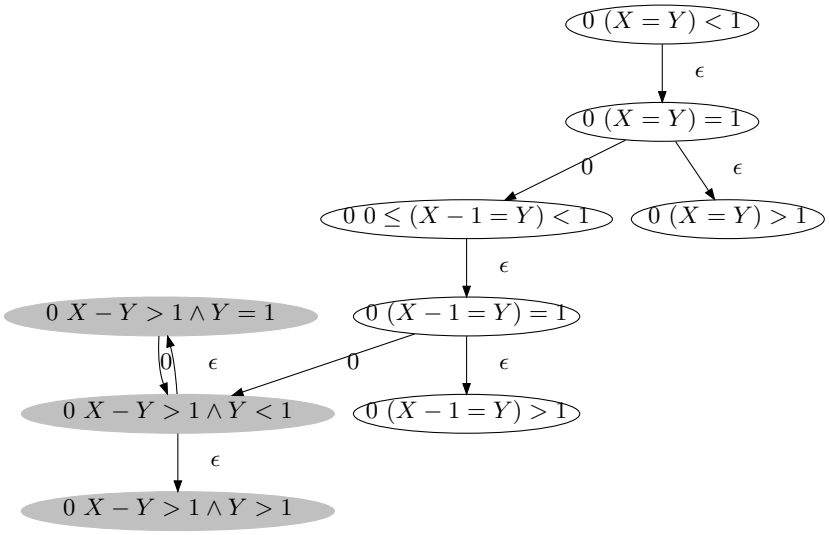


Fig. 6: Zones of Figure 6a after abstraction.

In this implementation we use a simplified version of the *maximum constants* abstraction described in [5].

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

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

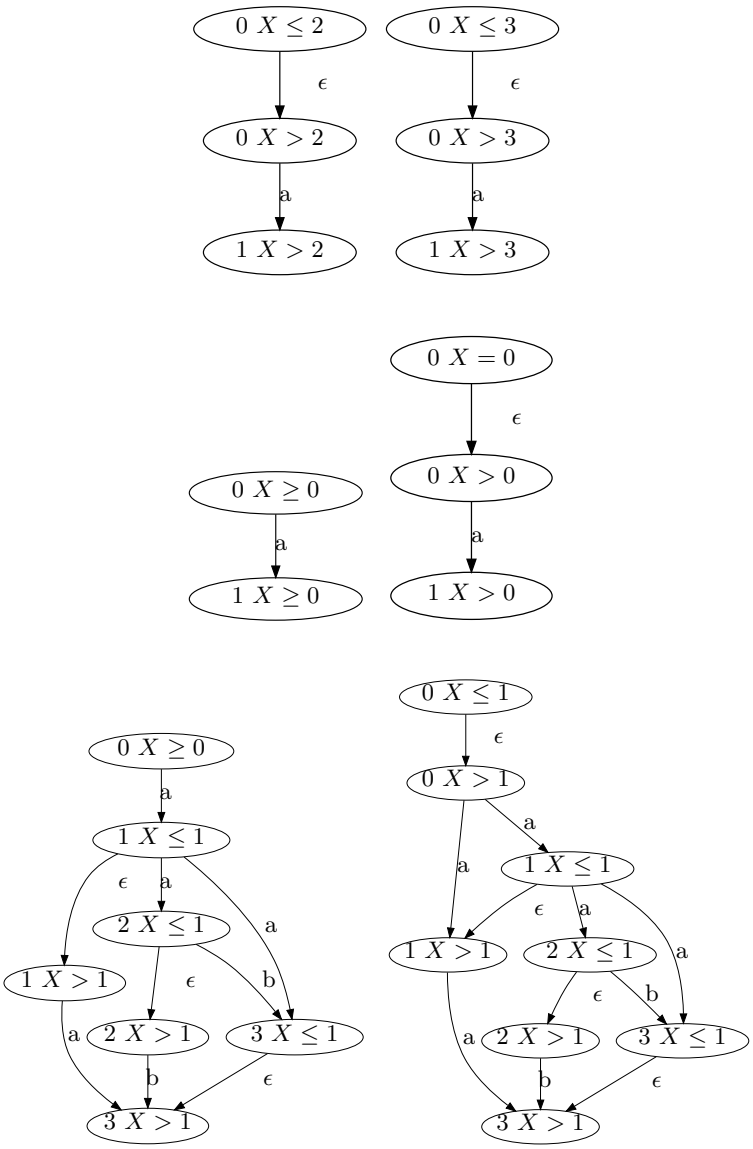
10 Time abstracted relations on timed automata

10.1 Time abstracted bisimilarity

Definition 16. Strong time abstracted bisimulation*: A binary relation R is a strong time abstracted bisimulation (STaB) if and only if, for all $(s_1, s_2) \epsilon R$, $a \epsilon Act$, $d \epsilon R_{\geq 0}$*

$\forall s'_1 (s_1 \xrightarrow{a} s'_1 \Rightarrow \exists s'_2 . (s_2 \xrightarrow{a} s'_2 \wedge (s'_1, s'_2) \epsilon R)) \wedge$
$\forall s'_2 (s_2 \xrightarrow{a} s'_2 \Rightarrow \exists s'_1 . (s_1 \xrightarrow{a} s'_1 \wedge (s'_1, s'_2) \epsilon R)) \wedge$
$\forall s'_1 (s_1 \xrightarrow{d} s'_1 \Rightarrow \exists (s'_2, d') . (s_2 \xrightarrow{d'} s'_2 \wedge (s'_1, s'_2) \epsilon R)) \wedge$
$\forall s'_2 (s_2 \xrightarrow{d} s'_2 \Rightarrow \exists (s'_1, d') . (s_1 \xrightarrow{d'} s'_1 \wedge (s'_1, s'_2) \epsilon R))$

Fig. 7: Examples for time abstracted bisimilarities.



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.*

Definition 17. 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.*

Definition 18. 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.*

Figure 7 shows examples for each kind of time abstracted bisimilarity. The initial zones of Figure 8a and Figure 8b show STaB, TadB and TaoB, while the initial zones of Figure 8c and Figure 8d show TadB and TaoB but not STaB, and the initial zones of Figure 8e and Figure 8f show TaoB but not STaB or TadB.

11 Algorithms

11.1 Fernandez' algorithm

This algorithm is summarised here from [9]. This algorithm follows a strategy of repeatedly splitting a partition of the states of the LTS, into a finer partition, until a fixpoint is reached and no further refinement is possible. The details of two-way splitting and three-way splitting are omitted for brevity.

Initialise $\pi = \{Pr\}$;
Initialise W $= \{Pr\}$ **while** *W is not empty* **do**
  Choose a splitter B from W, removing it;
  if *B is a simple splitter* **then**
    Perform a two-way split on each action with respect to B and update W;
  else
    Perform a three-way split on each action with respect to B and update W;
  end
end

11.2 Creation of the zone valuation graph

Overview This algorithm is adapted from the algorithms in [2] and [13]. We changed it to make the correctness more evident. This 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

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

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. 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.

Pseudocode for this follows.

```
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 ↑) with an ε
self-loop;
while Q is not empty do
    Dequeue (l_parent, t, l_child) from Q;
    if l_parent ≠ 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', 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;
            Update edges accordingly;
            if new zones are created in t.source then
                Set new_zone;
            end
        end
    end
end
Generate zone_valuation_graph by applying Fernandez' algorithm to
zone_graph;
Return zone_graph;
```

Proof of correctness

- `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 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.

11.3 Checking timed and untimed relations on timed automata

Overview Many important relations on timed automata can be verified by arguing about the existence of winning strategies on two-player games. The algorithm described here uses this fact to verify these relations by simulating the two-player games, using a memoisation approach.

The relations which can be verified in this fashion are those in which functions f_P, f_Q exist such that the question 'Are symbolic states s_P and s_Q in timed automata P and Q related under R? ' can have three possible answers:

1. yes
2. no
3. if and only if

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

This property is satisfied by timed bisimulation, STaB, TadB, TaoB, and the corresponding simulation equivalences, among others.

So, 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

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

This formulation naturally suggests an approach: use tables to store pairs of states in the two timed automata which are known to be related or known to be unrelated, and use a dynamic programming approach to verify the relation for any given pair of states. Pseudocode for this is given in the procedure CheckAutomataRelation which calls procedure CheckStatesRelation recursively to return either `true` or `false` for the given pair of 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*)

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)

12 Conclusions and future work

We designed and implemented algorithms for verifying timed and time abstracted relations on pairs of timed automata, and provided reference implementations for STaB, TaDB and TaOB. However, some scope for further work remains.

- Using the framework we have developed for verifying timed relations, such as time abstracted bisimilarity from [2].
- Investigation along the lines of the original zone-valuation graph algorithm in [2] in order to see if the efficiency of this algorithm can be improved while retaining correctness.
- Investigation along the lines of proving cycles cannot exist in the product LTS of two zone-valuation graphs in order to simplify the current algorithm for verifying relations by omitting cycle detection.
- Cleaner code through better integration with the UPPAAL libraries for DBM operations and parsing of timed automata, and with Ocamlgraph, a set of Ocaml libraries for implementing common operations for graphs.

References

1. Tripakis, S., Yovine, S.: Analysis of timed systems using time-abstracting bisimulations. Formal Methods in System Design **18**(1) (2001) 25–68
2. 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
3. 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
4. Aceto, L., Ingólfsdóttir, A., Larsen, K.G., Srba, J.: Reactive systems: modelling, specification and verification. Cambridge University Press (2007)
5. Behrmann, G., Bouyer, P., Fleury, E., Larsen, K.G.: Static guard analysis in timed automata verification. In: In TACAS, Springer (2003) 254–277
6. Keller, R.M.: Formal verification of parallel programs. Commun. ACM **19**(7) (July 1976) 371–384
7. Kanellakis, P.C., Smolka, S.A.: Ccs expressions, finite state processes, and three problems of equivalence. Information and Computation **86**(1) (1990) 43–68

8. Paige, R., Tarjan, R.E.: Three partition refinement algorithms. SIAM Journal on Computing **16**(6) (1987) 973–989
9. Fernandez, J.C.: An implementation of an efficient algorithm for bisimulation equivalence. Science of Computer Programming **13**(2) (1990) 219–236
10. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126** (1994) 183–235
11. David, A.: Uppaal dbm library programmer's reference (2006)
12. Bengtsson, J., Yi, W.: Timed automata: Semantics, algorithms and tools. In: Lectures on Concurrency and Petri Nets. Springer (2004) 87–124
13. Guha, S.: An algorithm to generate zone valuation graph. unpublished