Timed Automata

Jennifer Nist

11^{th} February 2016

Chair of Software Engineering
Albert-Ludwigs Universität Freiburg

Outline

Timed Automata

Timed automata are used to model and verify the behaviour of real-time systems over time.

A timed automaton consists of

- vertices l_i called locations,
- edges e_i,
- and real-valued variables $t_i \in \mathbb{R}$ called clocks.

Timed Automata: Clocks

Clocks

- model time,
- increase monotonically with $t_0 \leq t_1 \leq \cdots \leq t_n$,
- and proceed at rate one, i.e after d time steps every clock increased by d.

Time (clock variables) can only increase while being in a location.

Timed Automata: Example

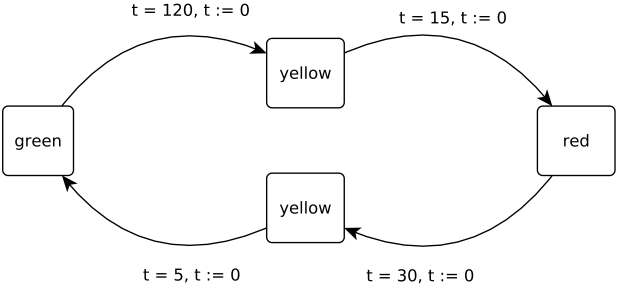


Figure : A simplified example of a timed automaton

Timed Automata: Actions and clock constraints

Every **edge** can be combined with

- actions, and
- clock constraints called guards.

Guards enable the transition if satisfied and disable it otherwise.

Every **location** can contain

- clock constraints called invariants.

Invariants limit the time allowed to spend in the location.

Timed Automata: Example

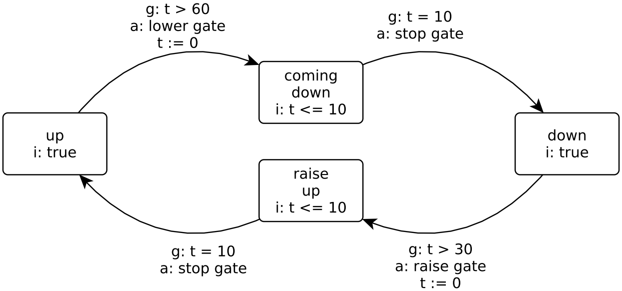


Figure : Timed automaton of a crossing gate

i: invariant, g: guard, a: action

Timed Automata: Definitions

Definition (Guard)

For a set $\mathcal{C}$ of clocks, with constants $c \in \mathbb{Q}$ and $t \in \mathcal{C}$, the set G over $\mathcal{C}$ of *clock constraints g*, called *guard* is defined by the grammar:

$$g ::= t < c \mid t \leq c \mid t > c \mid t \geq c \mid g \wedge g$$

Definition (Clock valuation)

For a given set of clocks $\mathcal{C}$, a *clock valuation* $\nu : \mathcal{C} \rightarrow \mathbb{R}_{\geq 0}$ is a mapping which assigns a real, non-negative value to each clock.

Timed Automata: Definitions

Definition (Timed Automaton, Syntax)

A *timed automaton* $A = (Loc, Act, \mathcal{C}, Edge, Inv, Init, Fin)$ is a tuple with

- *Loc* is a finite set of *locations*,
- *Act* is a finite set of *actions*,
- $\mathcal{C}$ is a finite set of *clocks*,
- $Edge \subseteq Loc \times Act \times CC(\mathcal{C}) \times 2^{\mathcal{C}} \times Loc$ is finite set of *edges*,
- $Inv : Loc \to CC(\mathcal{C})$ is a mapping which assigns an *invariant* to each location,
- $Init \subseteq Loc$ with $\nu(t_i) = 0$ for all $t_i \in \mathcal{C}$ is the finite set of *initial locations*, and
- $Fin \subseteq Loc$ is a finite set of *final locations*.

Timed Automata: Definitions

Definition (Timed Automaton, Semantics)

- Any timed automaton T can be interpreted as a transition system TS with infinitely many states.
- A state of TS is a pair (l, ν) with $l \in Loc$ of T and ν is a clock valuation for $\mathcal{C}$ of T.
- A path is a sequence of states $s_0 \to s_1 \to \cdots \to s_n$.
- A run is a path starting in a initial state $s_0 \to \cdots \to s_n$ with $s_0 = (l_0, \nu)$, $l_0 \in Init$.

Timed Automata: Definitions

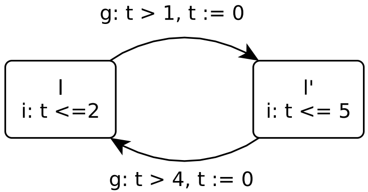
Definition (Transition semantics)

Edge :

$$\frac{\nu \models g \quad \nu' = \text{reset } C \text{ in } \nu \quad \nu' \models Inv(l')}{(l,\nu) \overset{a}{\rightarrow} (l',\nu')} \tag{1}$$

Location :

$$\frac{t > 0 \;\; \nu' = \nu + t \quad \nu' \models Inv(l)}{(l,\nu) \overset{t}{\rightarrow} (l,\nu')} \tag{2}$$



Timed Language

Definition (Timed words)

A *timed word* over an alphabet Σ is a sequence $(a_0, t_0), (a_1, t_1), \ldots, (a_k, t_k)$, where each $a_i \in \Sigma$ and each t_i in $\mathbb{R}$.

Definition (Untimed words)

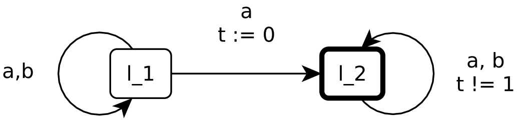
The *untimed word* v of a timed word w is the sequence of the actions without the occurrence times.

Example

The correspondent untimed word v for the timed word $w = (a_0, t_1), (a_1, t_1), (a_2, t_2)$ is $v = a_0 a_1 a_2$.

Timed language: Example

Set of accepted words:
$\{w \mid \text{action } a \text{ at some time } t, \text{ and no action at time } t+1\}$.



Accepted timed words w and untimed words v

- $w_0 = (a, 0) \rightarrow v_0 = a$
- $w_1 = (ab, 1), (ab, 2), (ab, 3), (a, 0) \rightarrow v_1 = abababa$
- $w_2 = (ab, 1), (a, 0), (ab, 0.99), (ab, 1.01) \rightarrow v_2 = abaabab$

Timed language

A *timed language* over the alphabet Σ is a set of timed words over Σ and is denoted $L(A)$.

Definition (Time regular language, Oliver Finkel)

A timed language L is said to be *timed regular* if there exists a timed automaton A such that $L(A) = L$.

Timed Language

Theorem (Alur et al.)

The set of timed regular languages is closed under union, intersection, but not under complementation.

First part: Closed under union and intersection.

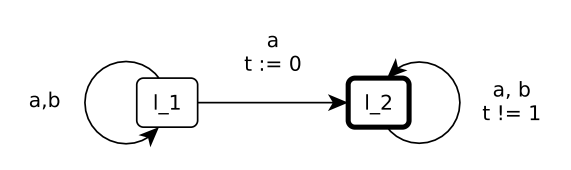
Proof: Extend the classical product construction to timed automata.

Timed Language

Second part:
Show, that there exists a timed automaton that generates a timed regular language L whose complementation $\overline{L}$ is not time regular.

Proof.

- Let $\Sigma = \{a, b\}$ and L be the timed language.
- The words $w \in L$ contain an action a at time t such that no action occurs at time $t + 1$.

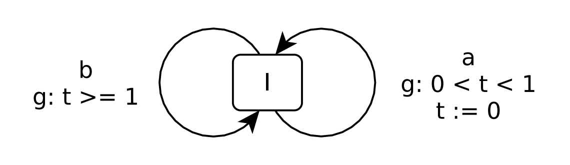


- The timed automaton in the figure above accepts L.

Proof

Proof.

- Construct L' which consists of timed words w' such that
  - all the a actions happen before time 1,
  - no two a actions happen at the same time and
  - the untimed word v matches the regular expression $a^* b^*$.
- It can be verified, that L' is timed regular.



- The timed automaton in the figure above accepts L'.

Proof

Proof.

- Observe that $untime(\overline{L} \cap L')$ is the language consisting of the words $\{a^n b^m | m \geq n\}$.
- Regarding to the theorem, the intersection of two timed regular languages is again timed regular.
- But the language $\{a^n b^m | m \geq n\}$ is not regular. This leaves the conclusion, that $\overline{L}$ is not timed regular.

□

Emptiness problem

Problem: Decide whether the language $L(A)$ for a given timed automaton is empty.

Detect if there exists a final state that is reachable from an initial state.

New Problem: Solve a reachability problem.

$\Rightarrow$ To decide the reachability problem, wee need a finite state space abstraction.

Solution: Construct a region automaton.

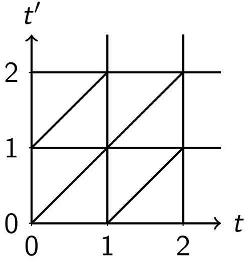
Region automaton

Idea: Divide the infinite state space of each location into a finite number of regions.

Region: Each state of a region is equivalent regarding to a defined equivalence relation.

Clock equivalence

For two clocks t, t' with $c_t, c_{t'} = 2$ every intersection of two integers, horizontal, vertical, upper and lower triangle, and diagonal line is a clock region.



- The equivalence class $[\nu]$ is called *clock region*.
- For a timed automaton the number of clock regions is finite.

Clock equivalence

Let A be a timed automaton, $\mathcal{C}$ the set of clocks and c_t the largest constant which a clock $t \in \mathcal{C}$ is compared to.

Definition (Clock equivalence)

Two clock valuations ν and ν' are *clock equivalent* $\nu \cong \nu'$, if and only if either

- for all $t \in \mathcal{C}$ $\nu(t) > c_t \wedge \nu'(t) > c_t$ or
- for all $t, t' \in \mathcal{C}$ with $\nu(t), \nu'(t) \leq c_t$ and $\nu(t'), \nu'(t') \leq c_{t'}$ all the following conditions hold:

$$\lfloor \nu(t) \rfloor = \lfloor \nu'(t) \rfloor \wedge (\langle \nu(t) \rangle = 0 \Leftrightarrow \langle \nu'(t) \rangle = 0)$$

$$\langle \nu(t) \rangle \leq \langle \nu(t') \rangle \Leftrightarrow \langle \nu'(t) \rangle \leq \langle \nu'(t') \rangle$$

$\langle t \rangle$ denotes the fractional part, and $\lfloor t \rfloor$ the integral part of $t \in \mathbb{R}$.

Region equivalence

Definition (Region equivalence)

Two states (l, ν) and (l', ν') are *region equivalent* $(l, \nu) \cong (l', \nu')$ iff $l = l'$ and $\nu \cong \nu'$

- The equivalence class $[s]$ are called *state regions*.
- A state region $[s] = (l, [\nu])$ is a pair where l is a location and $[\nu]$ is a clock region.

Region automaton

Given a timed automaton A.

Definition (Region automaton)

The *region automaton* with respect to the region equivalence consists of

- state regions $[s] = (l, [\nu])$
- edges.

- The region automaton of A is denoted $R(A)$.
- The language of $R(A)$ is the untimed language of $L(A)$.

Region automaton: Example

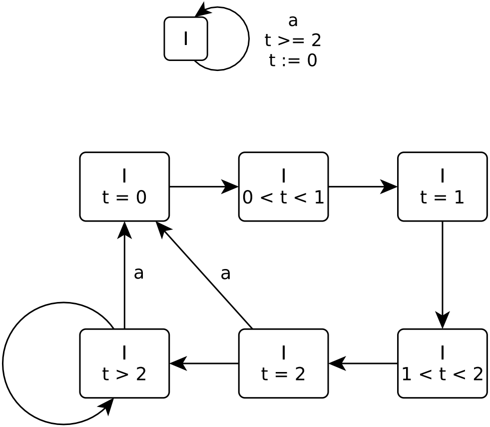


Figure : Region automaton

Region automaton

The reachability and language emptiness of timed automata can now be solved in time linear in the number of vertices and edges of the region automaton.

The size of the region automaton itself is

- linear in the number of locations and edges of the timed automaton, and
- exponential in the number of clocks.

Theorem (Alur et al.)

The language emptiness question for timed automata is PSPACE-complete.

Determinization

- Deterministic timed automata are strictly less expressive than timed automata.
- For a given (non-deterministic) timed automaton A, there does not always exists a deterministic timed automaton accepting the same language.

  $\Rightarrow$ The problem of checking whether there exists an equivalent deterministic timed automaton is not even known to be decidable[4].

  $\Rightarrow$ It is not possible to use the powerset construction to generate a deterministic finite automaton.

Summary

- Timed automata model and verify the behaviour of real-time systems over time.
- Timed automata are neither determinizable nor complementable.
- Region automata have a finite state space.
- Region automata are used to decide the reachability problem.
- The emptiness and reachability problem are decidable.

Sources

- Tripakis, Stavros. "Folk theorems on the determinization and minimization of timed automata." Formal Modeling and Analysis of Timed Systems. Springer Berlin Heidelberg, 2004. 182-188.
- Alur, Rajeev, and Parthasarathy Madhusudan. "Decision problems for timed automata: A survey." Formal Methods for the Design of Real-Time Systems. Springer Berlin Heidelberg, 2004. 1-24.
- Finkel, Olivier. "On decision problems for timed automata." Bulletin of the European Association for Theoretical Computer Science 87 (2005): 185-190.
- Brard, Batrice. "An Introduction to Timed Automata." Control of Discrete-Event Systems. Springer London, 2013. 169-187.

Sources

- Baier, Christel, and Joost-Pieter Katoen. Principles of model checking. Vol. 26202649. Cambridge: MIT press, 2008.
- Alur, Rajeev, and David L. Dill. "A theory of timed automata." Theoretical computer science 126.2 (1994): 183-235.