Algorithms for Model Checking (2IW55)

Lecture 11

Timed Verification: Timed Automata

Chapter 16, 17

Tim Willemse
(timw@win.tue.nl)
http://www.win.tue.nl/∼timw
HG 6.81

Outline

Timed Systems

Informally: Timed Automata

Formaly: Timed Automata

Summary

Exercise

Timed Systems

So far, we have only considered untimed systems.

- Timing is of crucial importance for many systems:
  - controllers found in airplanes (landing gear, collision avoidance).
  - controllers found in cars (airbag, future *drive-by-wire* systems).
  - communication protocols (re-routing upon timeouts).

Functional correctness is only one of many aspect:

- the correct timing of an event is crucial.
- timing influences behaviour: the passing of time may disable events.

which model of time to use:

- Discrete time.
- Continuous time.

Timed Systems

In discrete time, time has a discrete nature:

- Time can be described by natural numbers
- A special tick action is used to model the advance of a single time unit

Advantage: standard temporal logic can be used to express timing properties: The next-operator measures time.

Example

A timeout is set two time units after a message is sent:

$$\mathsf{A}\,\mathsf{G}\,(\text{sent} \rightarrow \mathsf{X}\,\mathsf{X}\,(\text{timeout}))$$

Discrete time is mainly used for synchronous systems, such as hardware.

Timed Systems

Simplicity is the key advantage to discrete time:

- We can reuse mixed Kripke Structures: timed transitions are labelled with a tick action.
- We can check properties using existing languages such as CTL^*.

This means that traditional model checking algorithms are applicable.

Main disadvantages of discrete time:

- delay between any pair of actions is a multiple of an a priori fixed minimal delay.
- model is therefore only accurate up-to this minimal delay.
- finding the minimal delay is difficult in practice:
  - how to find the minimal delay in a distributed, asynchronous system?

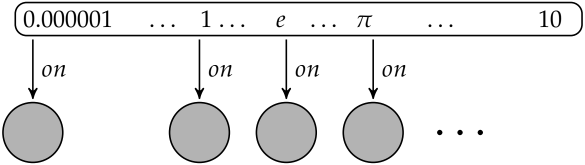
Timed Systems

In continuous time, time has a continuous nature:

- Time can be described by a dense domain, such as real numbers
- State changes can happen at any point in time

Example

An event on that must take place between time 0 and time 10 can be executed at time $0.000001, 1, e, \pi, \ldots$:



Problem: there are infinitely many moments that action *on* can happen. How to check that it happens before time t?

Timed Systems

Approach by Alur and Dill:

- Restrict expressive power of the temporal logic . . . . Timed CTL
- Describe timed systems symbolically . . . . Timed Automata
- Compute a finite representation of the infinite state space on-demand . . . Region Automata

Outline

Timed Systems

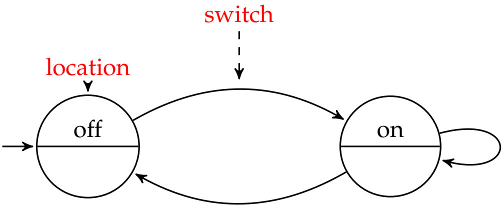
Informally: Timed Automata

Formaly: Timed Automata

Summary

Exercise

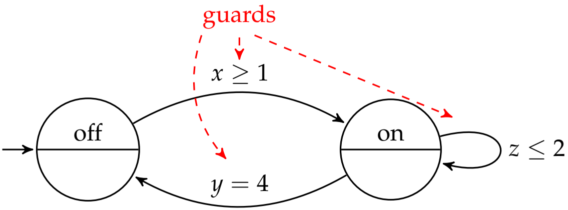
Informally: Timed Automata



A Timed Automaton:

- has vertices called locations,
- has edges called switches which are labelled with actions (not shown),
- Intuition: executing a switch consumes no time, i.e. it is instantaneous.
- time progresses in locations.

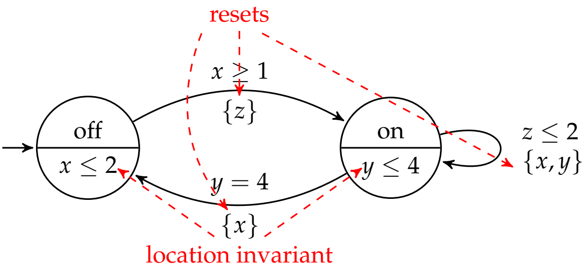
Informally: Timed Automata



…

- Has real-valued clocks $x, y, z, \ldots$, which all advance with the same speed,
- Has guards indicating when an edge may be taken.
- Intuition: Guards express at which moments in time a transition is *enabled*.
- Enabledness depends on the constraints on clocks.

Informally: Timed Automata



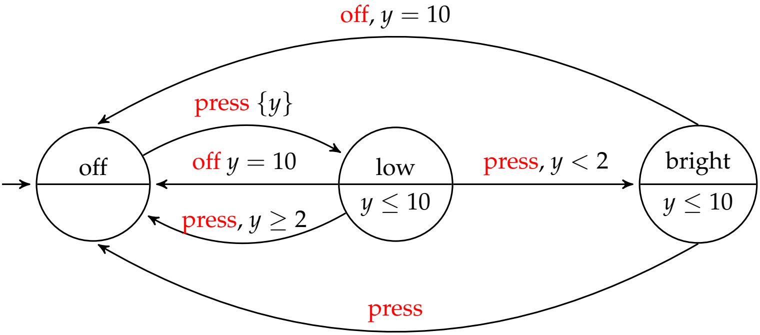
...

- Switches can reset clocks upon execution, i.e. set some clocks to 0.
- Time can only increase as long as the location invariant holds.
- A switch must be taken before the invariant becomes invalid.

Informally: Timed Automata

Example

The following timed automaton models a simple lamp with three locations: off, low and bright. If a button is pressed the lamp is turned on for at most ten time-units. If the button is pressed again, the lamp is turned off. However, if the button is pressed rapidly, the lamp becomes bright.



Outline

Timed Systems

Informally: Timed Automata

Formaly: Timed Automata

Summary

Exercise

Formally: Timed Automata

Timing constraints are provided by clock constraints:

$$\phi ::= \textsf{true} \mid x < c \mid x - y < c \mid x \leq c \mid x - y \leq c \mid \neg\phi \mid \phi \wedge \phi$$

- $c \in \mathbb{N}$ are constants (sometimes rational numbers);
- $x, y \in C$ are clocks
- As usual:
  - $x \geq c$ is short for $\neg(x < c)$;
  - $x \in [c_1, c_2)$ is short for $\neg(x < c_1) \wedge (x < c_2)$

The set of clock constraints over a set of clocks C is denoted $\mathcal{C}(C)$.

Formally: Timed Automata

A timed automaton is a tuple

$$\mathcal{T} = \langle L, L_0, Act, C, \longrightarrow, \iota \rangle$$

- L is a finite set of locations; $L_0 \subseteq L$ is a non-empty set of initial locations
- Act is the set of actions
- C is a finite set of clock variables
- $\longrightarrow \subseteq L \times \mathcal{C}(C) \times Act \times 2^C \times L$ is the set of switches
- $\iota : L \to \mathcal{C}(C)$ is the invariant assignment function

Formally: Timed Automata

- A clock constraint ϕ contains free variables
- The truth of a clock constraint ϕ depends on the values of the clocks
- A clock valuation ν for a set C of clocks is a function $\nu : C \to \mathbb{R}_{\geq 0}$
- Clock constraints are evaluated in the context of a clock valuation ν:
  - $[\mathsf{true}]_\nu = \mathsf{true}$
  - $[x < c]_\nu = \nu(x) < c$
  - $[x - y < c]_\nu = \nu(x) - \nu(y) < c$
  - $[x \leq c]_\nu = \nu(x) \leq c$
  - $[x - y \leq c]_\nu = \nu(x) - \nu(y) \leq c$
  - $[\neg\phi]_\nu = \text{not } [\phi]_\nu$
  - $[\phi_1 \wedge \phi_2]_\nu = [\phi_1]_\nu \text{ and } [\phi_2]_\nu$
- We write $\nu \models \phi$ iff $[\phi]_\nu = \mathsf{true}$.
- Clock valuation update: $\nu + d$ is defined as: $(\nu + d)(x) = \nu(x) + d$ for all $d \in \mathbb{R}_{\geq 0}$.
- Clock valuation reset: $[\nu]_R$ is defined as: $[\nu]_R(x) = 0$ if $x \in R$, else $\nu(x)$.

Formally: Timed Automata

Example

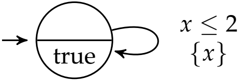
Let x, y be clocks and $\nu : \{x, y\} \to \mathbb{R}_{\geq 0}$ a clock valuation.

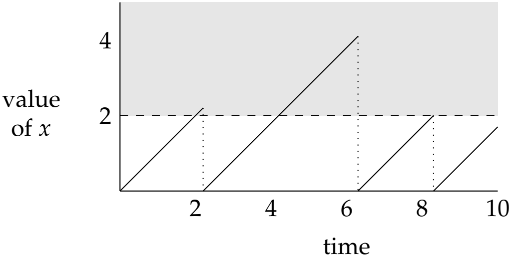
- if $\nu(x) = 2$ and $\nu(y) = \pi$, then $x < 3 \wedge y \geq 3$ holds
- the clock constraint $x - y > 2$ is valid whenever $\nu(x) - \nu(y) > 2$.
- the clock constraint $x \geq 2 \wedge x \leq 2$ is only valid whenever $\nu(x) = 2$.
- the clock constraint $x \geq 2 \wedge x - y < 2$ is only valid for $\nu(x) \geq 2$ and $\nu(y) > \nu(x) - 2$

Formally: Timed Automata

Example

The effect of a lower bound guarding a switch:

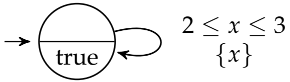


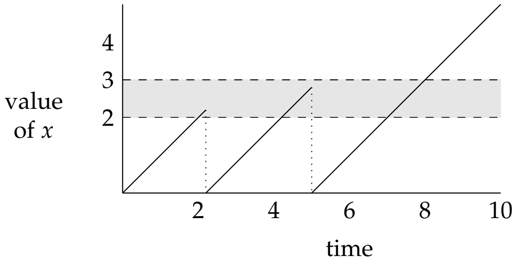


Formally: Timed Automata

Example

The effect of a lower bound and upper bound guarding a switch:

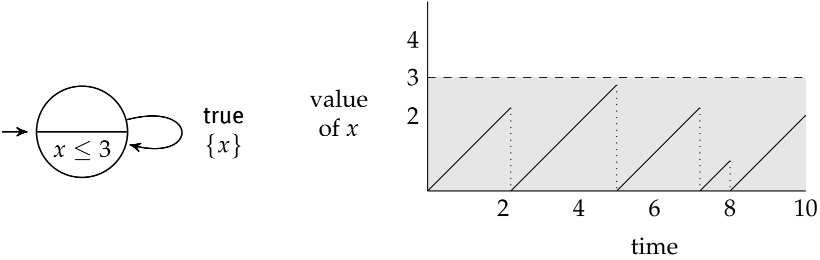




Formally: Timed Automata

Example

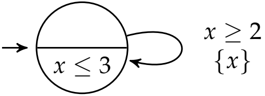
The effect of an invariant:

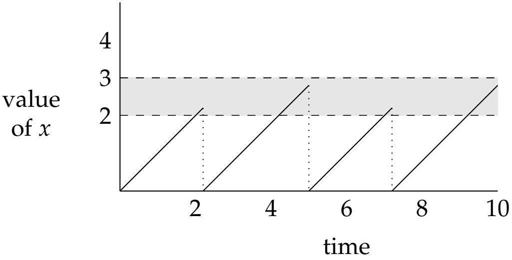


Formally: Timed Automata

Example

The effect of an invariant and guard combined:

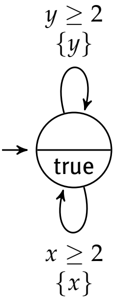


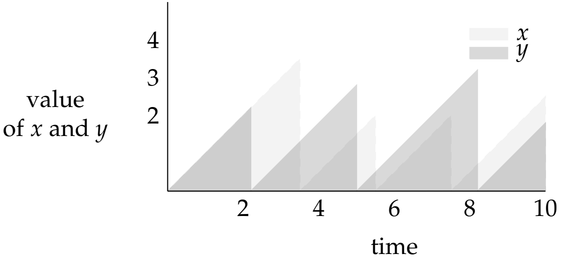


Formally: Timed Automata

Example

Switches that reset different clocks can cause an arbitrary difference between clock values. This is impossible to describe in a discrete time setting.





Formally: Timed Automata

Let $\mathcal{T} = \langle L, L_0, Act, C, \longrightarrow, \iota \rangle$ be a Timed Automaton. Its semantics is defined as a timed transition system: $[\mathcal{T}] = \langle S, S_0, Act, \rightarrow, \mapsto \rangle$

- $S = \{(l, \nu) \mid l \in L \wedge \nu : C \rightarrow \mathbb{R}_{\geq 0} \wedge \nu \models \iota(l)\}$, i.e. all combinations of locations and clock valuations that do not violate the location invariant.
- $S = \{(l, \nu) \mid l \in L_0 \wedge \nu : C \rightarrow \mathbb{R}_{\geq 0} \wedge \nu \models \iota(l)\}$.
- $\longrightarrow \subseteq S \times Act \times S$ is defined as follows:

$$\frac{l \xrightarrow{g\ a\ R} l' \qquad \nu \models g \wedge \iota(l) \qquad \nu' = [\nu]_R \qquad \nu' \models \iota(l')}{(l, \nu) \xrightarrow{a} (l', \nu')}$$

- $\mapsto \subseteq S \times \mathbb{R}_{\geq 0} \times S$ is defined as follows:

$$\frac{\nu \models \iota(l) \qquad \forall 0 \leq d' \leq d : \nu + d' \models \iota(l)}{(l, \nu) \overset{d}{\mapsto} (l, \nu + d)}$$

Formally: Timed Automata

Lemma

Let $\iota(l)$ be a negation-free location invariant. Then for all $d \in \mathbb{R}_{\geq 0}$ and all ν:

$$\nu \models \iota(l) \text{ and } \nu + d \models \iota(l) \text{ implies } \forall 0 \leq d' \leq d : \nu + d' \models \iota(l)$$

- The proof follows by a structural induction on $\iota(l)$.
- This means that for negation-free location invariants, we can simplify the rule for timed transition relations:

$$\frac{\nu \models \iota(l) \qquad \nu + d \models \iota(l)}{(l, \nu) \overset{d}{\mapsto} (l, \nu + d)}$$

Formally: Timed Automata

Recalling intuition:

- A switch $l \xrightarrow{g\ a\ R} l'$ means that:
  - action a is enabled whenever guard g evaluates to true.
  - upon executing the switch, we move from location l to location l' and reset all clocks in R to zero.
  - only locations l' that can be reached with clock values that satisfy the location invariant.
- an invariant $\iota(l)$ limits the time that can be spent in location l.
  - staying in location l only is allowed as long as the invariant evaluates to true.
  - before the invariant becomes invalid location l must be left.
  - if no switch is enabled when the invariant becomes invalid no further progress is possible.
- Thus, we need to determining when a clock constraint is valid or invalid.

Outline

Timed Systems

Informally: Timed Automata

Formaly: Timed Automata

Summary

Exercise

Summary

- Timed Systems can be modelled by discrete time or continuous time.
- For discrete time, existing model checking can be reused.
- For continuous time, a new model is introduced: Timed Automata.
- Timed Automata give rise to infinite transition systems.
- Timed Automata can model systems that cannot be described by means of discrete time.

Outline

Timed Systems

Informally: Timed Automata

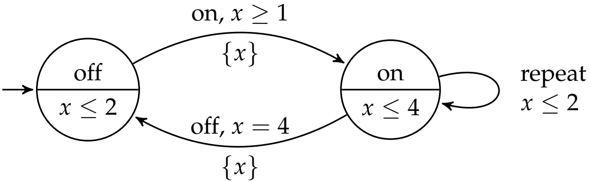
Formaly: Timed Automata

Summary

Exercise

Exercise

Consider the following Timed Automaton.



- Explain which switches can be executed.
- Is there a possibility that the Timed Automaton enters a state in which time cannot progress anymore?
- Give the Timed Transition System for the Timed Automaton.