Timed Automata

Lecture #15 of Advanced Model Checking

Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling & Verification

E-mail: `katoen@cs.rwth-aachen.de`

December 18, 2006

Time-critical systems

- Timing issues are of crucial importance for many systems, e.g.,
  - landing gear controller of an airplane, railway crossing, robot controllers
  - steel production controllers, communication protocols . . . . . .
- In time-critical systems correctness depends on:
  - not only on the logical result of the computation, but
  - also on the time at which the results are produced
- How to model timing issues:
  - discrete-time or continuous-time?

A discrete time domain

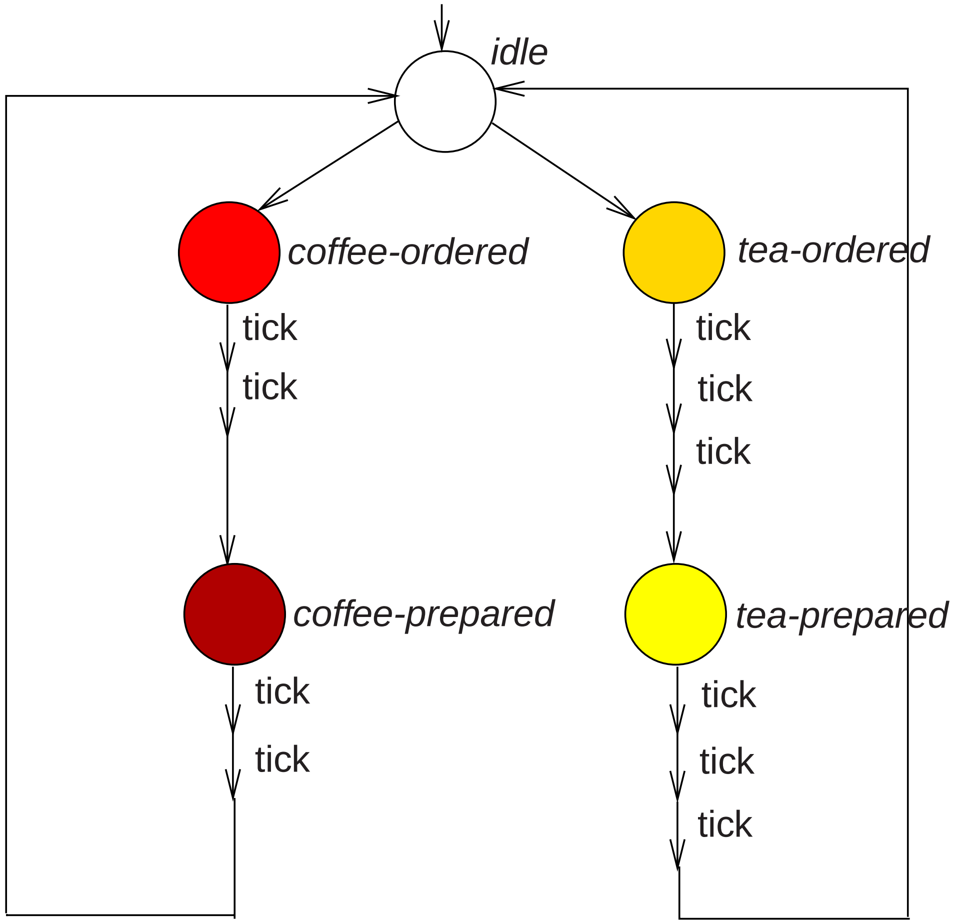
- Time has a *discrete* nature, i.e., time is advanced by discrete steps
  - time is modelled by naturals; actions can only happen at natural time values
  - a specific tick action is used to model the advance of one time unit
  - $\Rightarrow$ delay between any two events is always a multiple of the minimal delay of one time unit

- Properties can be expressed in traditional temporal logic
  - the next-operator "measures" time
  - two time units after being red, the light is green: $\square(red \Rightarrow \bigcirc \bigcirc green)$
  - within two time units after red, the light is green:

$$\square(red \Rightarrow (green \vee \bigcirc green \vee \bigcirc \bigcirc green))$$

- Main application area: synchronous systems, e.g., hardware

A discrete-time coffee machine



A discrete time domain

- Main advantage: conceptual simplicity
  - labeled transition systems equipped with a tick actions suffice
  - standard temporal logics can be used

  ⇒ traditional model-checking algorithms suffice

- Main limitations:
  - (minimal) delay between any pair of actions is a multiple of an *a priori* fixed minimal delay

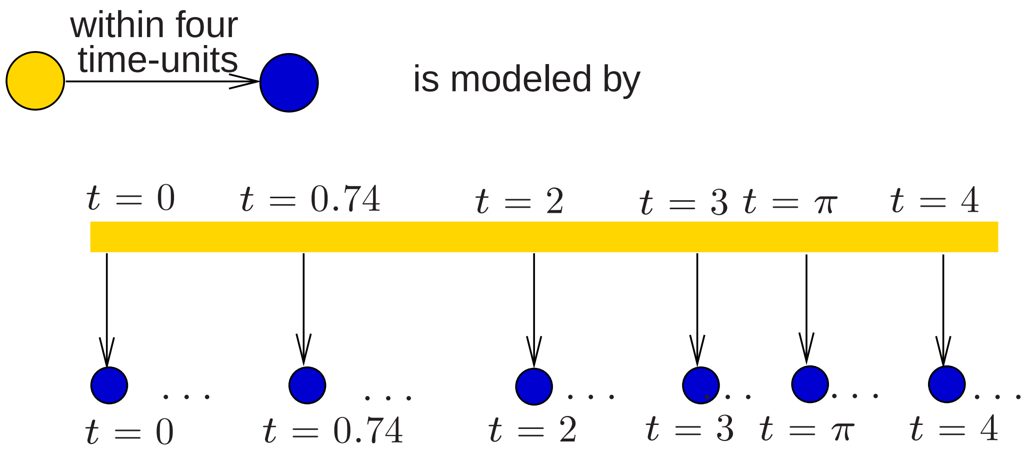
  ⇒ difficult (or impossible) to determine this in practice

  ⇒ limits modeling accuracy

  ⇒ inadequate for *asynchronous* systems. e.g., distributed systems

A continuous time-domain

If time is continuous, state changes can happen at any point in time:



but: infinitely many states and infinite branching

How to check a property like:

once in a yellow state, eventually the system is in a blue state within π time-units?

Approach

- *Restrict expressivity* of the property language
  - e.g., only allow reference to natural time units

$\Longrightarrow$ Timed CTL

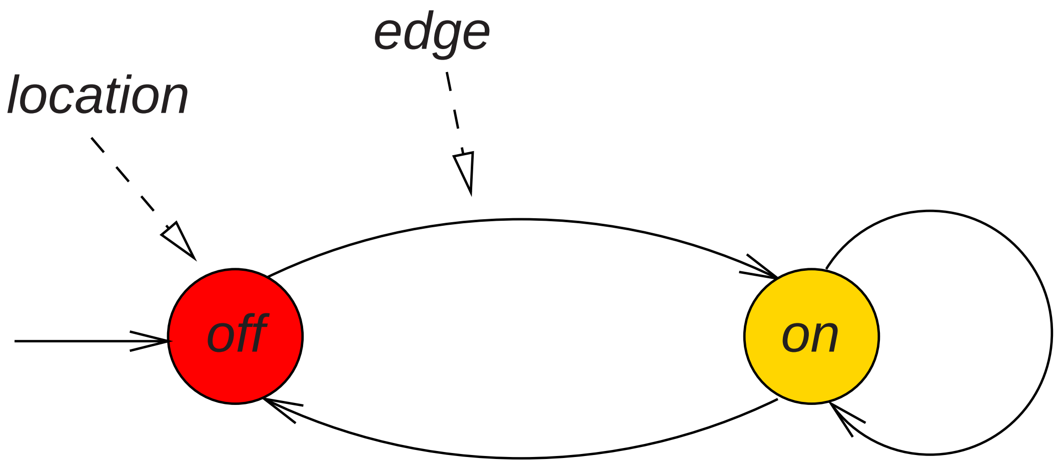
- Model timed systems *symbolically* rather than explicitly
  - in a similar way as program graphs and channel systems

$\Longrightarrow$ Timed Automata

- Consider a *finite quotient* of the infinite state space on-demand
  - i.e., using an equivalence that depends on the property and the timed automaton

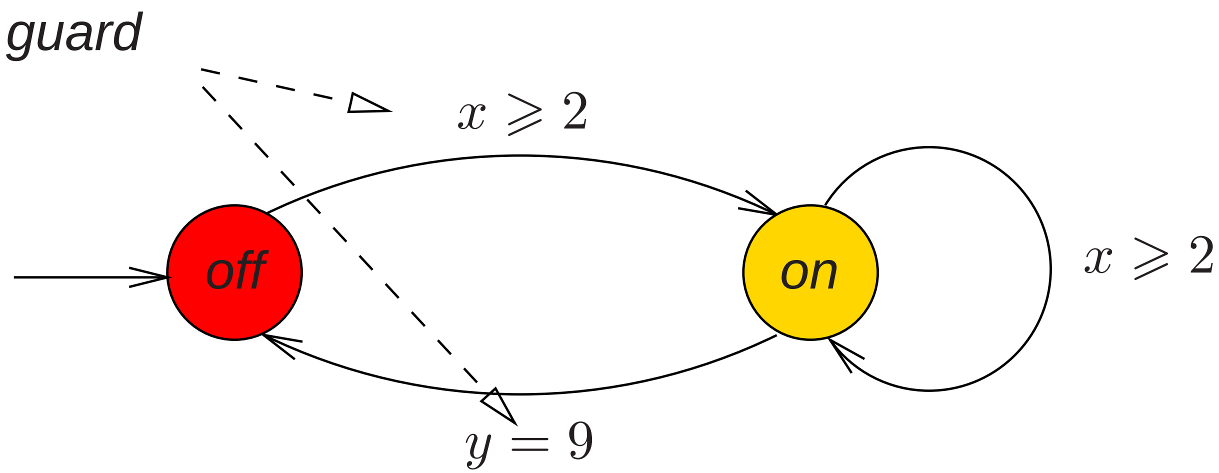
$\Longrightarrow$ Region Automata

What is a timed automaton?



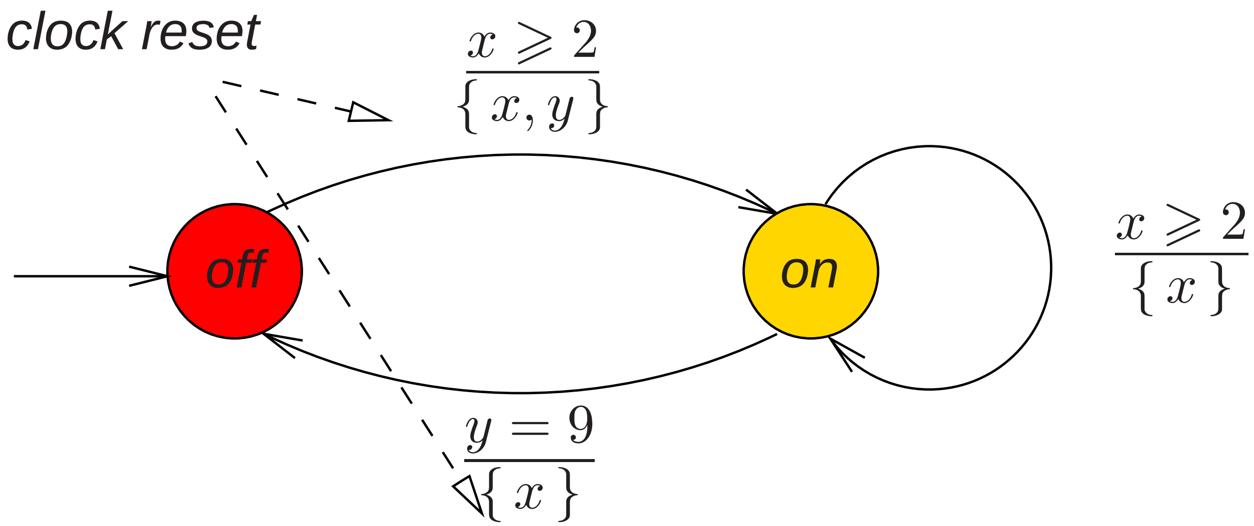
- a program graph with *locations* and *edges*
- a location is labeled with the valid *atomic propositions*
- *taking an edge is instantaneous*, i.e, consumes no time

What is a timed automaton?



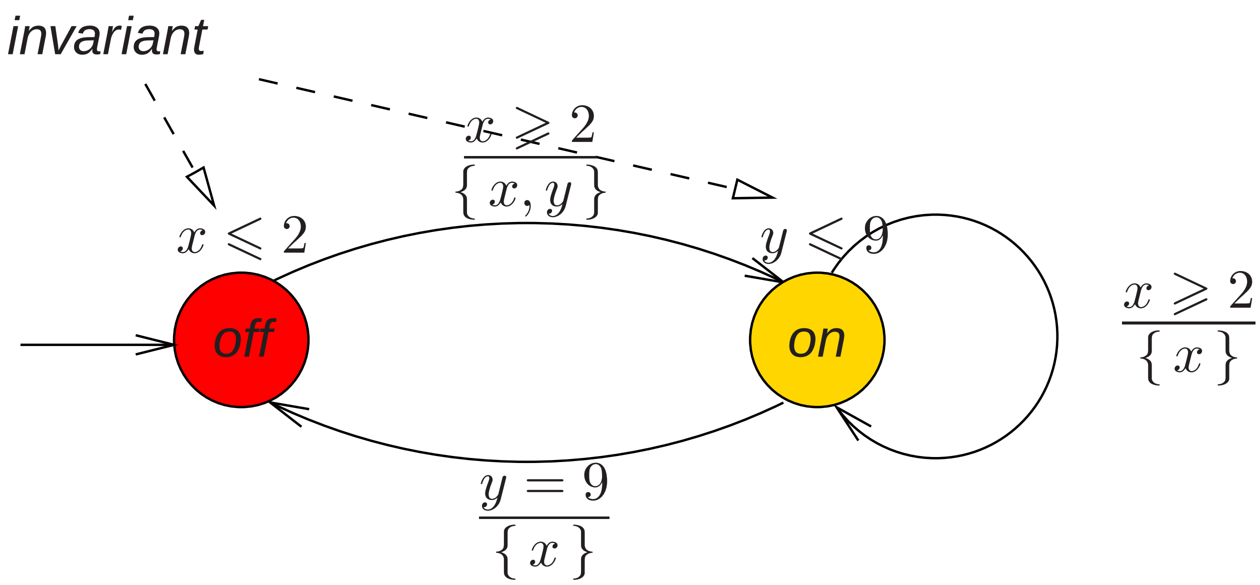
- equipped with real-valued *clocks* $x, y, z, \ldots$
- clocks advance implicitly, all at the *same speed*
- logical constraints on clocks can be used as *guards* of actions

What is a timed automaton?



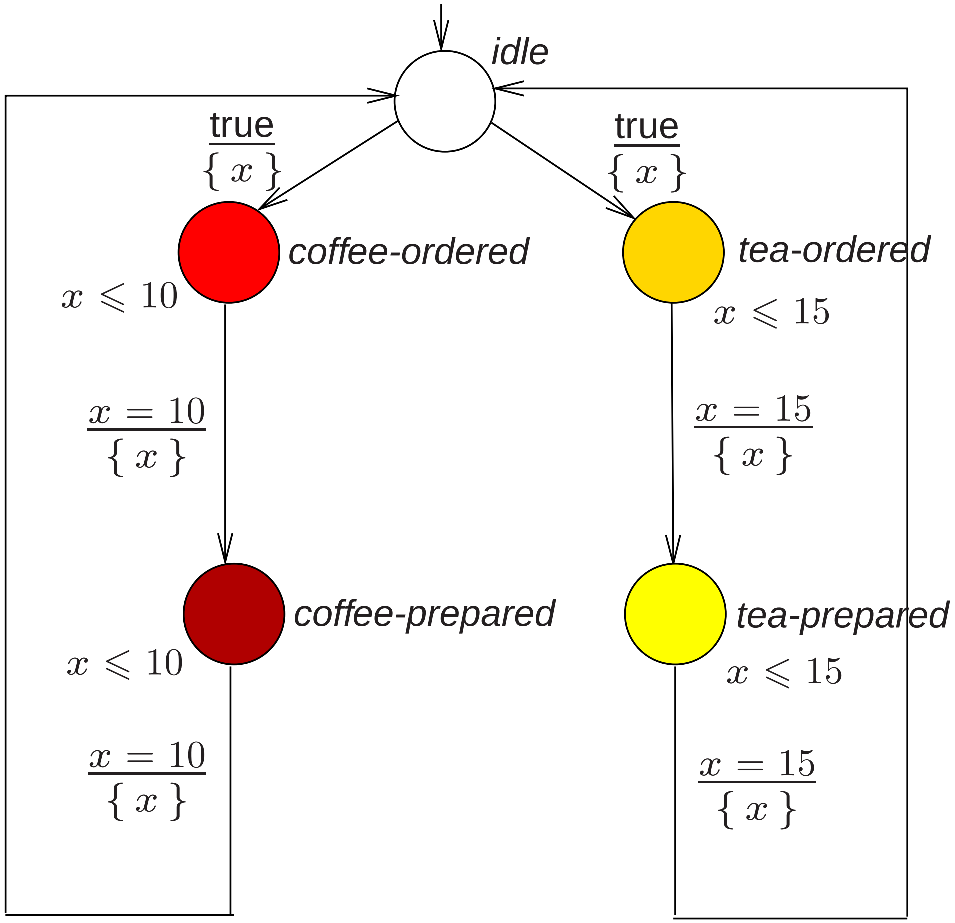
- clocks can be *reset* when taking an edge
- assumption:
  all clocks are zero when entering the initial location initially

What is a timed automaton?



- guards indicate when an edge *may* be taken
- a location invariant specifies the *amount of time that may be spent in a location*
  - when a *location invariant* becomes invalid, an edge must be taken

A real-time coffee machine



Clock constraints

- *Clock constraints* over set C of clocks are defined by:

$$g ::= \text{true} \;\big|\; x < c \;\big|\; x - y < c \;\big|\; x \leqslant c \;\big|\; x - y \leqslant c \;\big|\; \neg g \;\big|\; g \wedge g$$

  - where $c \in \mathbb{N}$ and clocks $x, y \in C$
  - rational constants would do; neither reals nor addition of clocks!
  - let *CC*(C) denote the set of clock constraints over C
  - shorthands: $x \geqslant c$ denotes $\neg(x < c)$ and $x \in [c_1, c_2)$ or $c_1 \leqslant x < c_2$ denotes $\neg(x < c_1) \wedge (x < c_2)$

- *Atomic clock constraints* do not contain true, $\neg$ and $\wedge$
  - let *ACC*(C) denote the set of atomic clock constraints over C

- *Diagonal-free constraints* do neither contain $x - y \leqslant q$ nor $x - y < q$
  - let *DCC*(C) be the set of diagonal-free clock constraints over C

Timed automaton

A *timed automaton* is a tuple

$$TA = \big(Loc, Act, C, \rightsquigarrow, Loc_0, inv, AP, L \big) \quad \text{where:}$$

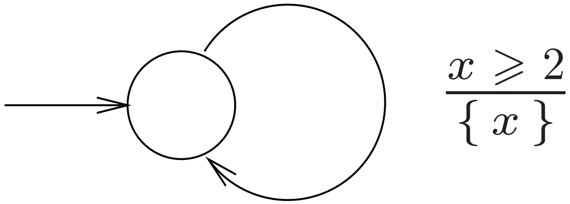
- Loc is a finite set of locations.
- $Loc_0 \subseteq Loc$ is a set of initial locations
- C is a finite set of clocks
- $L : Loc \to 2^{AP}$ is a labeling function for the locations
- $\rightsquigarrow \subseteq Loc \times CC(C) \times Act \times 2^C \times Loc$ is a transition relation, and
- $inv : Loc \to CC(C)$ is an invariant-assignment function

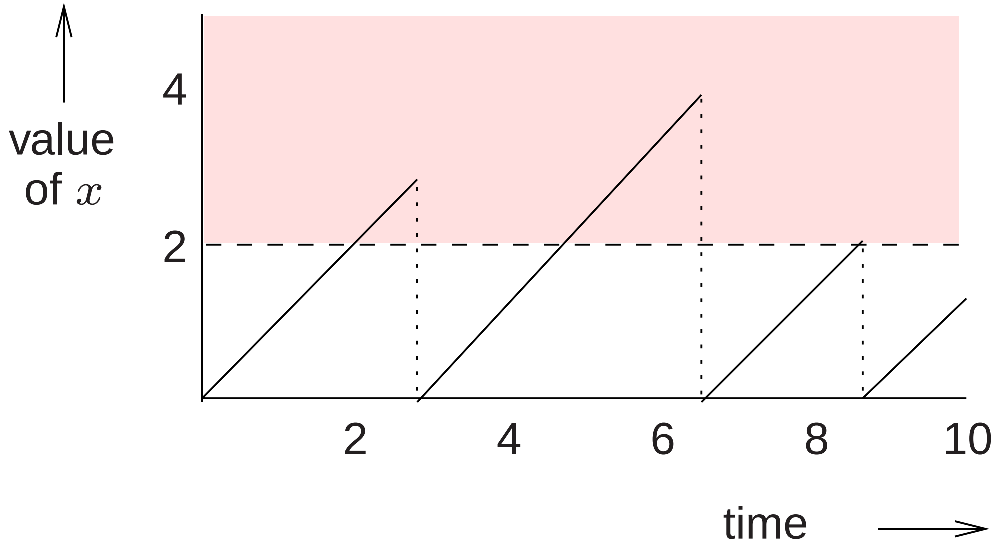
Intuitive interpretation

- Edge $\ell \xrightarrow{g:\alpha,C'} \ell'$ means:
  - action α is enabled once guard g holds
  - when moving from location ℓ to ℓ', any clock in C' will be reset to zero
- *inv*(ℓ) constrains the amount of time that may be spent in location ℓ
  - once the invariant *inv*(ℓ) becomes invalid, the location ℓ must be left immediately
  - if this is not possible – no enabled outgoing transition – no further progress is possible

Guards versus location invariants

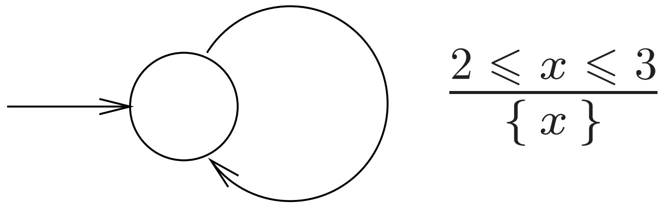
The effect of a lowerbound guard:

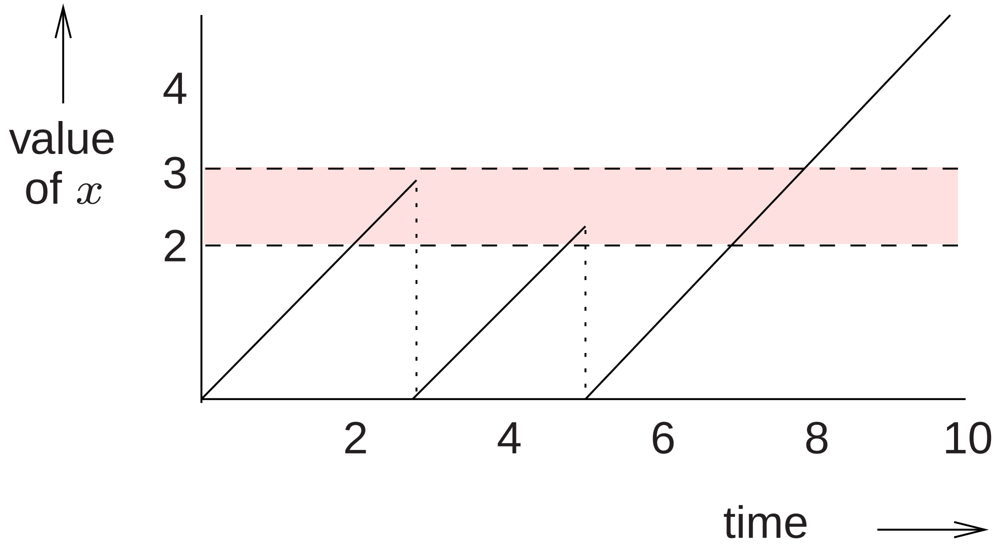




Guards versus location invariants

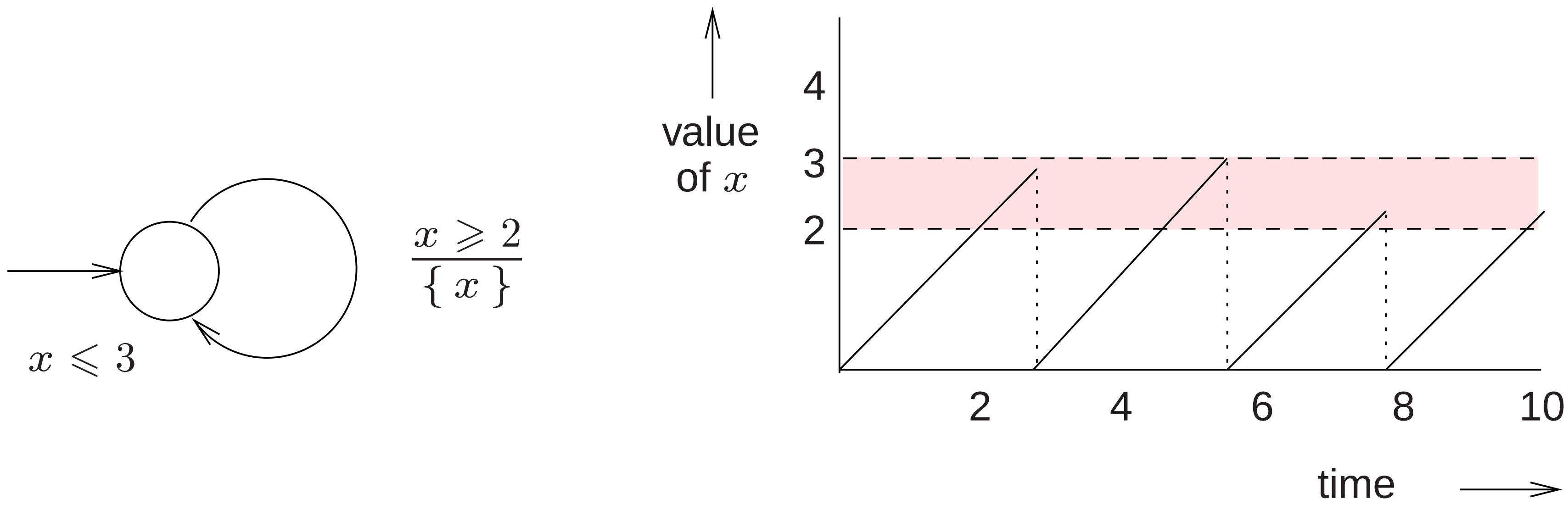
The effect of a lowerbound and upperbound guard:



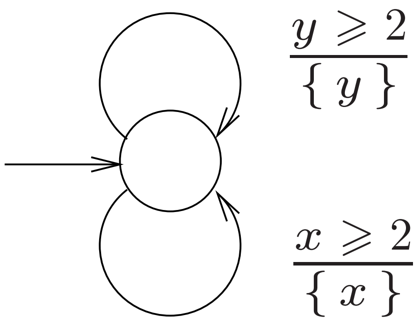


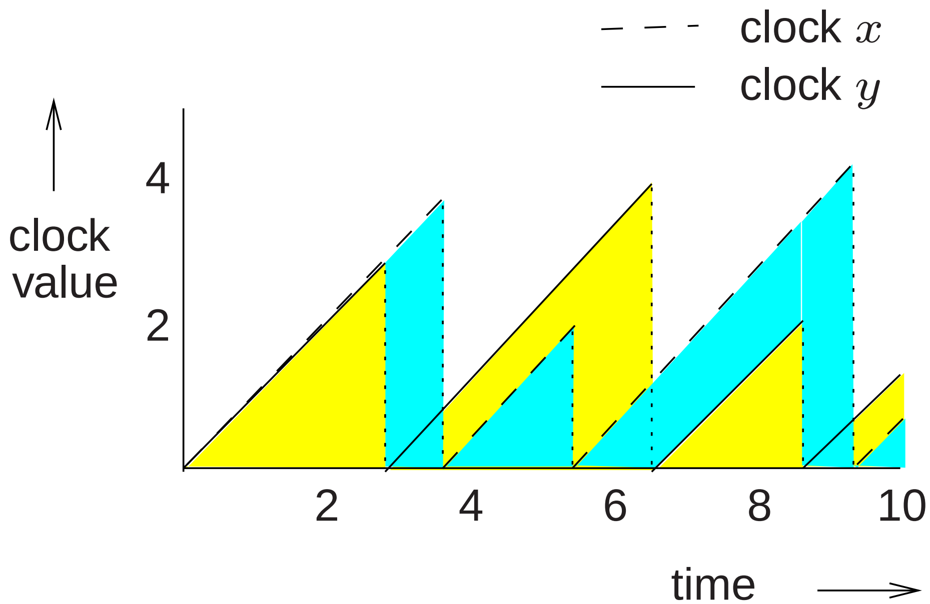
Guards versus location invariants

The effect of a guard and an invariant:



Arbitrary clock differences





This is impossible to model in a discrete-time setting

A timed mutual exclusion protocol

Composing timed automata

Let $\mathit{TA}_i = \big(\mathit{Loc}_i, \mathit{Act}_i, C_i, \rightsquigarrow_i, \mathit{Loc}_{0,i}, \mathit{inv}_i, \mathit{AP}, L_i\big)$ and H an action-set

$$\mathit{TA}_1 \mid\mid_H \mathit{TA}_2 = \big(\mathit{Loc}, \mathit{Act}_1 \cup \mathit{Act}_2, C, \rightsquigarrow, \mathit{Loc}_0, \mathit{inv}, \mathit{AP}, L\big) \quad \text{where:}$$

- $\mathit{Loc} = \mathit{Loc}_1 \times \mathit{Loc}_2$ and $\mathit{Loc}_0 = \mathit{Loc}_{0,1} \times \mathit{Loc}_{0,2}$ and $C = C_1 \cup C_2$
- $\mathit{inv}(\langle \ell_1, \ell_2 \rangle) = \mathit{inv}_1(\ell_1) \;\wedge\; \mathit{inv}_2(\ell_2)$ and $L(\langle \ell_1, \ell_2 \rangle) = L_1(\ell_1) \cup L_2(\ell_2)$
- $\rightsquigarrow$ is defined by the inference rules: for $\alpha \in H$ $\dfrac{\ell_1 \overset{g_1:\alpha,D_1}{\rightsquigarrow_1} \ell_1' \;\wedge\; \ell_2 \overset{g_2:\alpha,D_2}{\rightsquigarrow_2} \ell_2'}{\langle \ell_1, \ell_2 \rangle \overset{g_1 \wedge g_2:\alpha,D_1 \cup D_2}{\rightsquigarrow} \langle \ell_1', \ell_2' \rangle}$

for $\alpha \notin H$: $\dfrac{\ell_1 \overset{g:\alpha,D}{\rightsquigarrow_1} \ell_1'}{\langle \ell_1, \ell_2 \rangle \overset{g:\alpha,D}{\rightsquigarrow} \langle \ell_1', \ell_2 \rangle}$ and $\dfrac{\ell_2 \overset{g:\alpha,D}{\rightsquigarrow_2} \ell_2'}{\langle \ell_1, \ell_2 \rangle \overset{g:\alpha,D}{\rightsquigarrow} \langle \ell_1, \ell_2' \rangle}$

An abstract example

Example: a railroad crossing

Clock valuations

- A *clock valuation* v for set C of clocks is a function $v : C \longrightarrow \mathbb{R}_{\geqslant 0}$
  - assigning to each clock $x \in C$ its current value $v(x)$
- Clock valuation $v{+}d$ for $d \in \mathbb{R}_{\geqslant 0}$ is defined by:
  - $(v{+}d)(x) = v(x) + d$ for all clocks $x \in C$
- Clock valuation reset x in v for clock x is defined by:

$$(\text{reset } x \text{ in } v)(y) = \begin{cases} v(y) & \text{if } y \neq x \\ 0 & \text{if } y = x. \end{cases}$$

  - reset x in $(\text{reset } y \text{ in } v)$ is abbreviated by reset x, y in v

Timed automaton semantics

For timed automaton $\mathit{TA} = \big(\mathit{Loc}, \mathit{Act}, C, \rightsquigarrow, \mathit{Loc}_0, \mathit{inv}, \mathit{AP}, L\big)$:

Transition system $\mathit{TS}(\mathit{TA}) = (S, \mathit{Act}', \rightarrow, I, \mathit{AP}', L')$ where:

- $S = \mathit{Loc} \times \mathit{val}(C)$, state $s = \langle \ell, v \rangle$ for location ℓ and clock valuation v
- $\mathit{Act}' = \mathit{Act} \cup \mathbb{R}_{\geqslant 0}$, (discrete) actions and time passage actions
- $I = \{ \langle \ell_0, v_0 \rangle \mid \ell_0 \in \mathit{Loc}_0 \ \wedge \ v_0(x) = 0$ for all $x \in C \}$
- $\mathit{AP}' = \mathit{AP} \cup \mathit{ACC}(C)$
- $L'(\langle \ell, v \rangle) = L(\ell) \cup \{ g \in \mathit{ACC}(C) \mid v \models g \}$
- $\rightarrow$ is the transition relation defined on the next slide

Timed automaton semantics

The transition relation $\rightarrow$ is defined by the following two rules:

- Discrete transition: $\langle \ell, v \rangle \xrightarrow{d} \langle \ell', v' \rangle$ if all following conditions hold:
  - there is an edge labeled $(g : \alpha, D)$ from location ℓ to ℓ' such that:
  - g is satisfied by v, i.e., $v \models g$
  - $v' = v$ with all clocks in D reset to 0, i.e., $v' =$ reset D in v
  - v' fulfills the invariant of location ℓ', i.e., $v' \models \textit{inv}(\ell')$
- Delay transition: $\langle \ell, v \rangle \xrightarrow{d} \langle \ell, v{+}d \rangle$ for positive real d
  - if for any $0 \leqslant d' \leqslant d$ the invariant of ℓ holds for $v{+}d'$, i.e. $v{+}d' \models \textit{inv}(\ell)$

Example

Merry Xmas and a happy new year