Timed Automata

Time-critical systems

- There are systems which are time-critical
  - Their behavior is subject to stringent timing constraints.
  - Example: a train crossing: the gate must be closed within a certain time bound to halt car and pedestrian traffic before the train reaches the crossing.
- We saw “metric” version of LTL with discrete time
- “The traffic light will turn green within the next 30 seconds.”

$$\Box\,(red \;\Rightarrow\; \Diamond^{\leqslant 30}\, green).$$

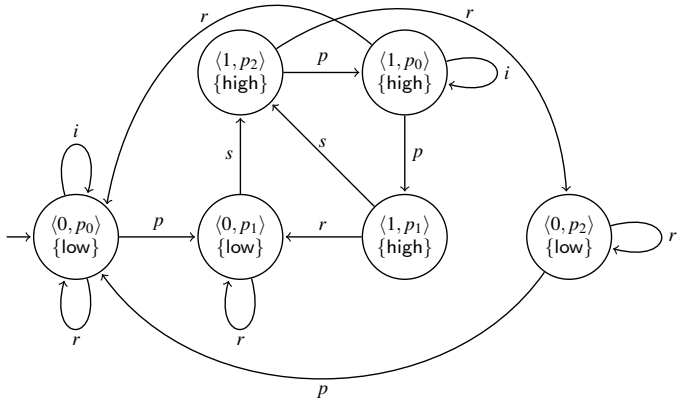
- This version of LTL is enough for *syncronous* systems, where all the components of the system take a transition «at the same time»
- What about asynchronous systems? Concurrent FSM?

Time in Finite State Machines

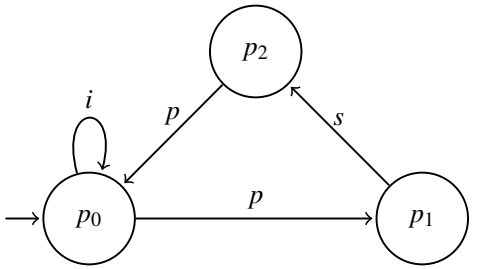
- Time is “ignored” in FSM:
  - **discrete**, **implicit**, **metric**
  - **infinite** with Büchi acceptance condition
- We can assume there is an underlying clock rate, so that each transition is taken synchronously: one transition= 1 time step.
- This is not always satisfactory

Time in FSM: parallel composition

- Complex models built from **composition** of FSM
- Temporal relationships between events in composed automata depend on the notion of composition
  - if fully **synchronous** (composition takes a transition if both automata do) timing relationships are preserved
  - **parallel** (asynchronous) composition does not preserve timing relationships
    - in particular **metric** ones when "transition = time step"



vs.



Decoupling time and transitions

- A rigid notion "transition = time step" is not really suitable when system models become complex
  - they must be built from composition of smaller models
  - Often some or all components are asynchronous
- So we need to introduce different mechanisms for capturing the advancement of time
- Several alternatives are possible...

How to decouple time and transitions

1. Time advances only when certain transitions are taken (or when certain conditions are met)
   - so for example composed automata can "agree" on which transitions make time advance
   - Statecharts
     - still essentially transition-based, good for **discrete** time
2. Time advances independent of transitions; guards on transitions can predicate on time to constrain *when* they are taken with respect to the timeline
   - Timed and Hybrid automata
     - can capture also notion of **continuous** (dense) time

Timed Automata (TA) as Program Graphs

- TA = Program graphs augmented with real-valued variables, called *clocks*
- State is a combination of *location* (the logical state of the automaton) + value of clocks
  - *set of locations is finite*, clocks take value in $R_{\geq 0}$, so state space is *infinite*
  - *often no AP set (we only used the names of locations)*
- A TA evolves in two ways:
  - location jumps (time does not advance)
  - time elapsing (location does not change)
- Clocks measure the time elapsing according to equation $\dot{x} = 1$
  - a clock can be *reset* to 0 by the action on a transition
  - states and transitions can include constraints on clocks

Clocks

- Access to Clocks is limited: they may only be inspected and reset to zero.
- Unless reset to zero, all clocks continuously incremented, as time progresses, at rate one
  - after the elapse of d time units, all clocks advance by d.
  - The value of a clock denotes the amount of time that has been elapsed since its last reset. Clocks can intuitively be considered as stopwatches that can be started and checked independently of one another.
- Conditions on the values of the clocks (clock contraints) are used as enabling conditions (i.e., guards) of transitions: only if the condition is fulfilled is the transition (and its action) is enabled and capable of being taken; otherwise, the action is disabled

A *clock constraint* over set C of clocks is formed according to the grammar

$$g ::= \quad x < c \;\Big|\; x \leqslant c \;\Big|\; x > c \;\Big|\; x \geqslant c \;\Big|\; g \wedge g$$

where $c \in \mathbb{N}$ and $x \in C$. Let $CC(C)$ denote the set of clock constraints over C.

Intuition

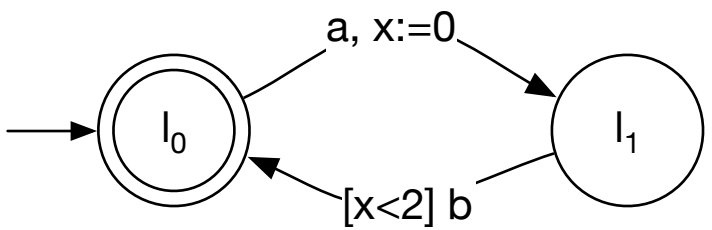
- A timed automaton is a program graph with a finite set C of clocks.
- Edges are labeled with tuples (g, α, D) where
  - g is a clock constraint,
  - α is an action, and
  - D ⊆ C is a set of clocks.
- The intuitive interpretation of:

$$\ell \overset{g:\alpha,D}{\hookrightarrow} \ell'$$

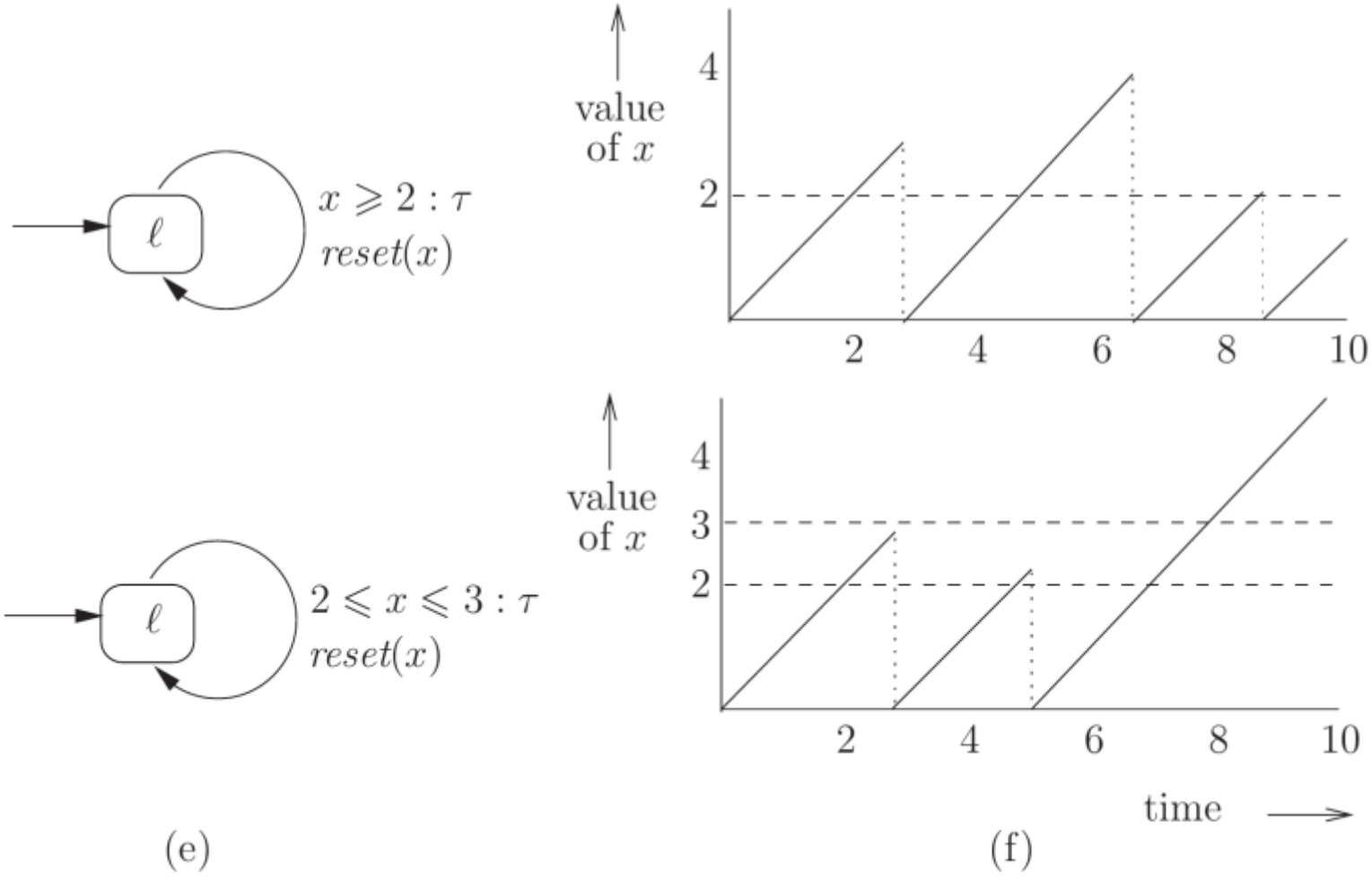
- is that the timed automaton can move from location ℓ to location ℓ' when clock constraint *g* holds. If moving, any clock in *D* will be reset to zero and action α is performed.

First example

- When taking transition from l_0 to l_1, clock x is reset
- Transition from l_1 to l_0 can be taken only if x<2



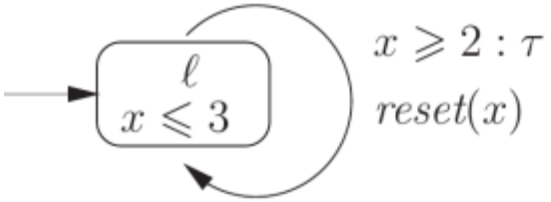
Simple examples of behavior



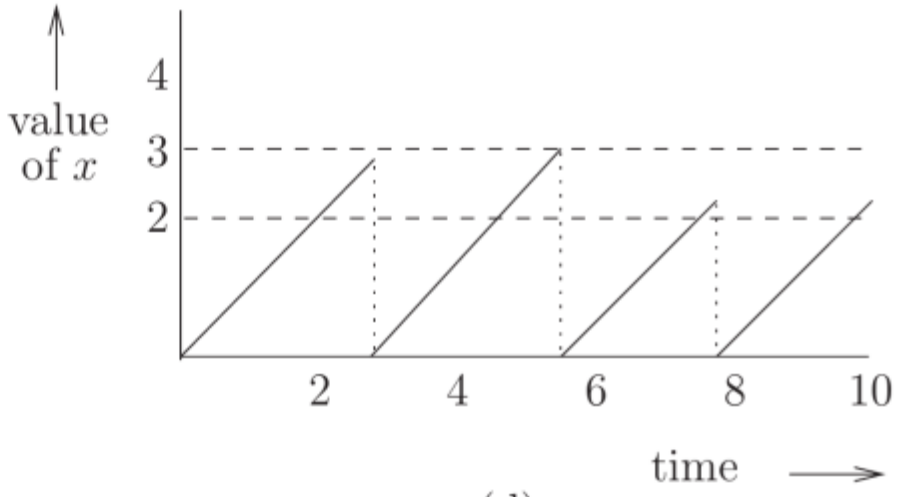
(e) (f)

Invariant

- A location ℓ may have a clock constraint, called its *invariant,* that constrain the amount of time that can be spent in ℓ .
  - Invariants are useful to force liveness constraints

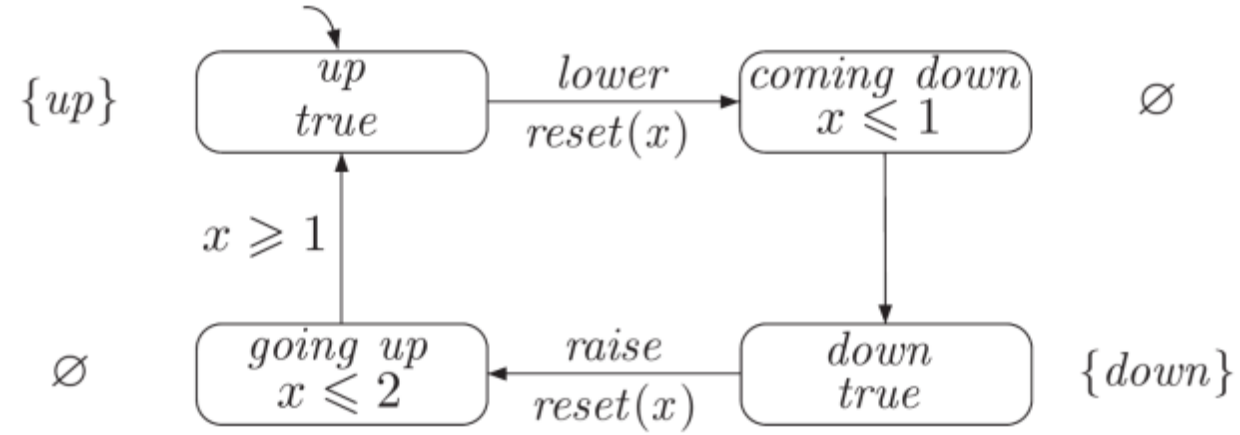


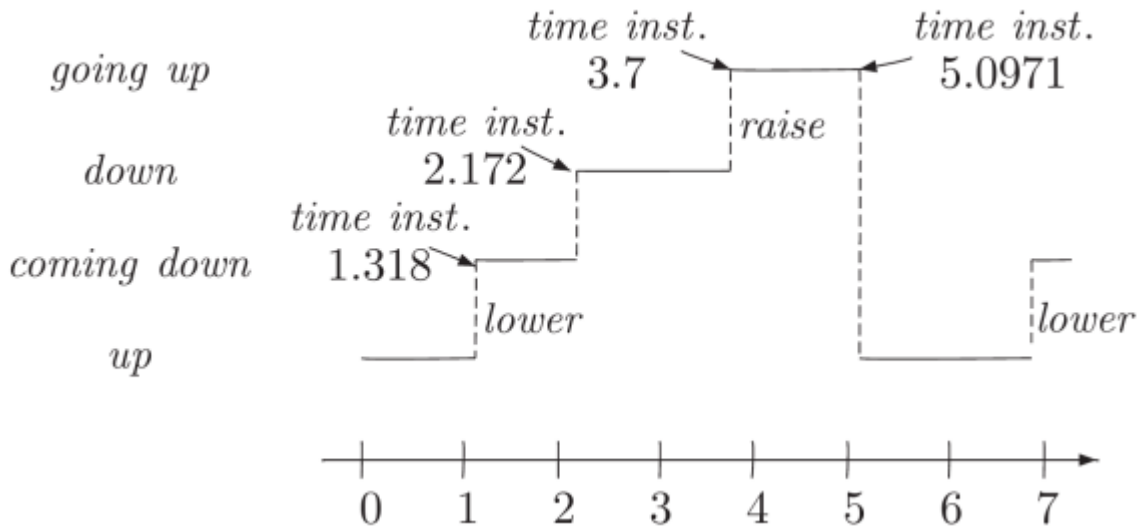
(c)



(d)

Another, larger example: gate of KRC





Formal Definition of TA as a TS

- TA = (Loc, Act, C, $\rightarrow$, Loc_0, Inv, AP,L), where:
- Loc is a finite set of locations
- $Loc_0 \subseteq$ Loc is a set of initial locations
- Act is a finite set of actions
- C is a finite set of clocks,
- $\rightarrow \subseteq Loc \times CC(C) \times Act \times 2^C \times Loc$ is a transition relation,
- Inv : Loc $\rightarrow$ CC(C) is an invariant-assignment function,
- AP is a finite set of atomic propositions,
- L : Loc $\rightarrow 2^{AP}$ is a labeling function for the locations.

Handshaking for parallel composition

Let $TA_i = (Loc_i, Act_i, C_i, \hookrightarrow_i, Loc_{0,i}, Inv_i, AP_i, L_i)$, $i = 1, 2$ be timed automata with $H \subseteq Act_1 \cap Act_2$, $C_1 \cap C_2 = \varnothing$ and $AP_1 \cap AP_2 = \varnothing$. The timed automaton $TA_1 \|_H TA_2$ is defined as

$$(Loc_1 \times Loc_2, Act_1 \cup Act_2, C_1 \cup C_2, \hookrightarrow, Loc_{0,1} \times Loc_{0,2}, Inv, AP_1 \cup AP_2, L)$$

where $L(\langle \ell_1, \ell_2 \rangle) = L_1(\ell_1) \cup L_2(\ell_2)$ and $Inv(\langle \ell_1, \ell_2 \rangle) = Inv_1(\ell_1) \wedge Inv_2(\ell_2)$. The transition relation $\hookrightarrow$ is defined by the following rules:

- for $\alpha \in H$:

$$\frac{\ell_1 \xhookrightarrow{g_1:\alpha, D_1}_1 \ell_1' \ \wedge \ \ell_2 \xhookrightarrow{g_2:\alpha, D_2}_2 \ell_2'}{\langle \ell_1, \ell_2 \rangle \xhookrightarrow{g_1 \wedge g_2:\alpha, D_1 \cup D_2} \langle \ell_1', \ell_2' \rangle}$$

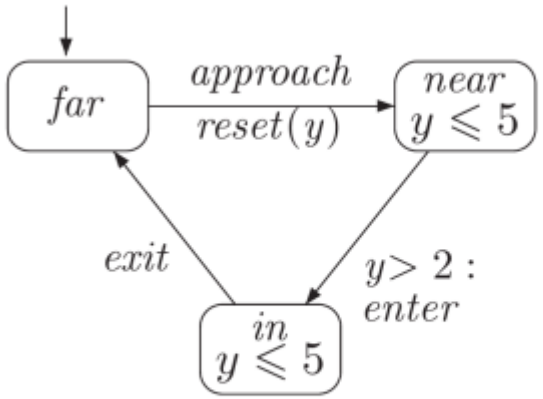
- for $\alpha \notin H$:

$$\frac{\ell_1 \xhookrightarrow{g:\alpha, D}_1 \ell_1'}{\langle \ell_1, \ell_2 \rangle \xhookrightarrow{g:\alpha, D} \langle \ell_1', \ell_2 \rangle} \quad \text{and} \quad \frac{\ell_2 \xhookrightarrow{g:\alpha, D}_2 \ell_2'}{\langle \ell_1, \ell_2 \rangle \xhookrightarrow{g:\alpha, D} \langle \ell_1, \ell_2' \rangle}$$

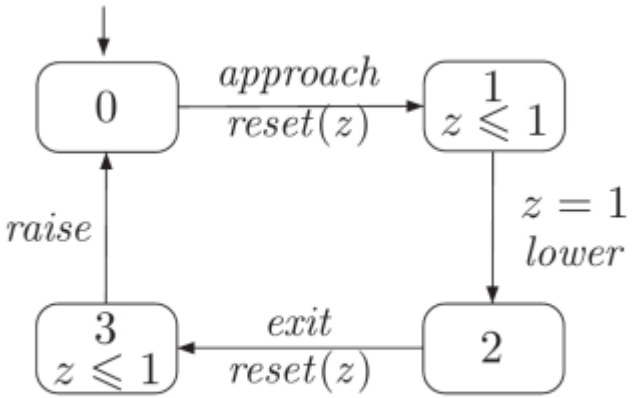
Example: KRC

$$(Train \|_{H_1} Controller) \|_{H_2} Gate$$

$$H_1 = \{ approach, exit \} \text{ and } H_2 = \{ lower, raise \}.$$

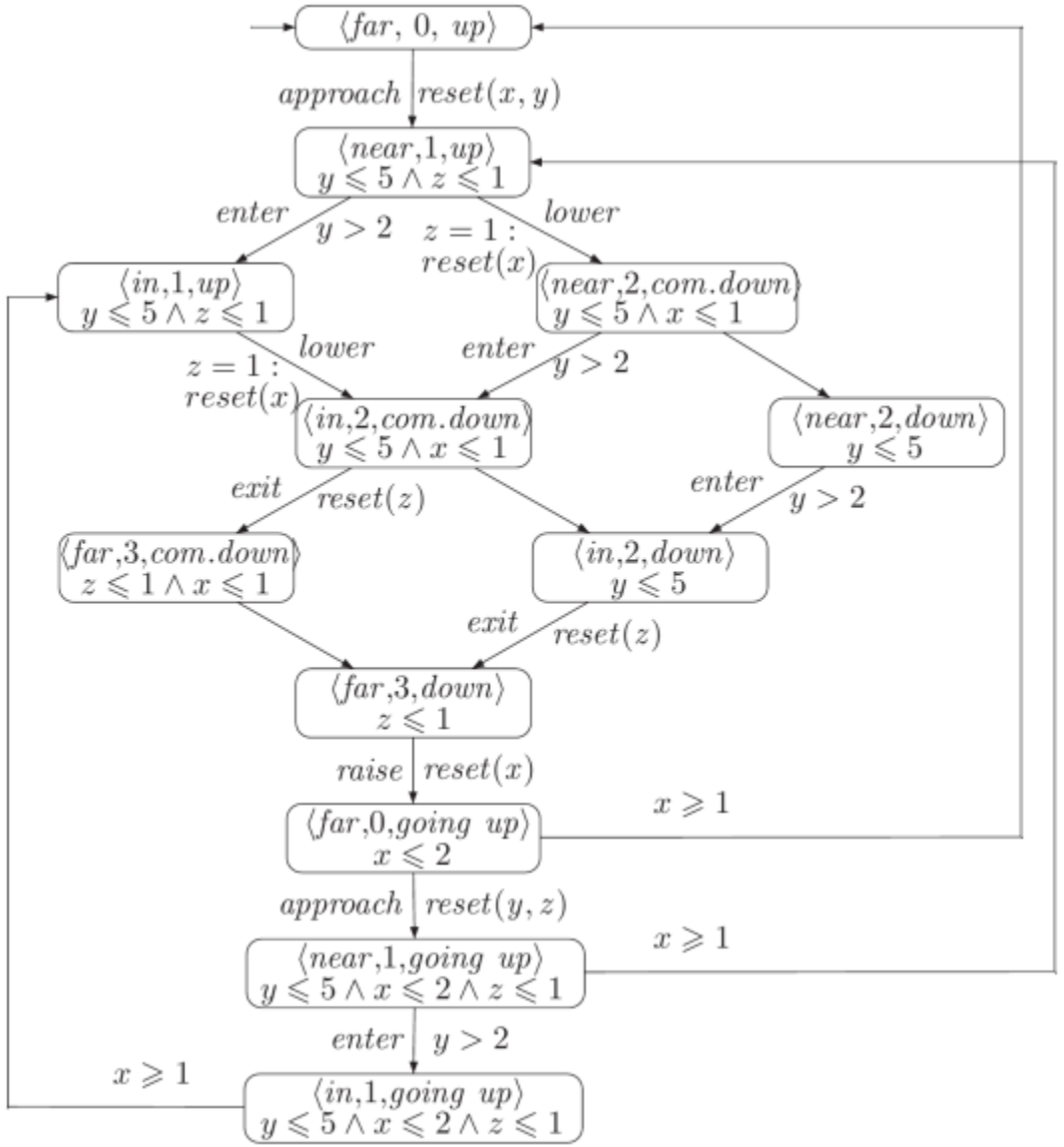


Train



Controller

The resulting TA



Semantics: clock valuation

- A *clock valuation* is a function η: $C \rightarrow \mathcal{R}_{\geq 0}$ assigning a nonnegative *real* value to each clock $x \in C$
- Eval(C) denotes the set of all clock valuations over C.
- A clock valuation η *satisfies* a clock constraint $g \in CC$ iff g evaluates to true when the values of the clocks are as in η
  - e.g. if C = {x, y}, η = {x =3.67, y =7}
    g_1 = x ≥ 2 and x < 5 and y ≤ 9
    g_2 = x ≥ 3 and y > 7 and y ≤ 10
    η satisfies g_1, but not g_2 (it violates constraint y > 7)

Formal definitions

Definition 9.10. Satisfaction Relation for Clock Constraints

For set C of clocks, $x \in C$, $\eta \in Eval(C)$, $c \in \mathbb{N}$, and $g, g' \in CC(C)$, let $\models \subseteq Eval(C) \times CC(C)$ be defined by

$$
\begin{array}{ll}
\eta \models \text{true} & \\
\eta \models x < c & \text{iff } \eta(x) < c \\
\eta \models x \leqslant c & \text{iff } \eta(x) \leqslant c \\
\eta \models \neg g & \text{iff } \eta \not\models g \\
\eta \models g \wedge g' & \text{iff } \eta \models g \ \wedge \ \eta \models g'
\end{array}
$$

■

Let η be a clock valuation on C. For positive real d, $\eta{+}d$ denotes the clock valuation where all clocks of η are increased by d. Formally, $(\eta{+}d)(x) = \eta(x) + d$ for all clocks $x \in C$. reset x in η denotes the clock valuation which is equal to η except that clock x reset. Formally:

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

Transitions and Delays

- Semantics defined in terms of an underlying transition system TS
- There are two possible ways in which a timed automaton can proceed: by taking a transition in the timed automaton, or by letting time progress while staying in a location.
  - Taking a transition = a discrete transition in the TS, labeled with the same action
  - Time progressing= a delay transition in the TS, labeled with a positive real number indicating the amount of time that has elapsed.
- The elapse of time in timed automata only takes place in locations, while actions are instantaneous (zero duration). As a result, at a single time instant, several actions may take place.

Transition System Semantics of a TA

- Let TA = (Loc, Act, C, $\rightarrow$, Loc_0, Inv, AP, L)
- Let TS(TA) = (S, Act' , $\rightarrow$, I, AP', L), called a Timed Transition System, be such that:
- S = Loc $\times$ Eval(C)
- Act' = Act $\cup$ $\mathscr{R}_{\geq 0}$
- I = { $\langle \ell_0, \eta \rangle \mid \ell_0 \in Loc_0 \wedge \eta(x) = 0$ for all $x \in C$ }
- AP' = AP $\cup$ ACC(C)
  - (ACC(C) is the set of atomic clock constrains
- L'($\langle \ell, \eta \rangle$) = L($\ell$) $\cup$ { $g \in ACC(C) \mid \eta \models g$ }
- the transition relation is defined next:

Transition relation of the TS

- To deal with time, the transition relation distinguishes two kinds of transitions
  - discrete (location change, zero time)
  - delay (time elapsing, no location change)

discrete transition: $\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \eta' \rangle$ if the following conditions hold:

(a) there is a transition $\ell \overset{g:\alpha,D}{\hookrightarrow} \ell'$ in TA

(b) $\eta \models g$

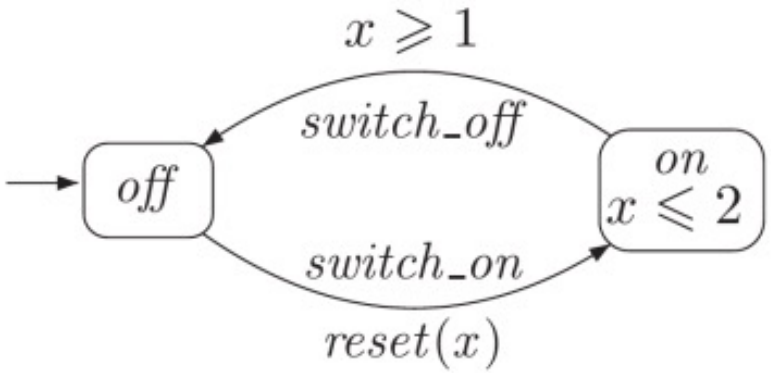
(c) $\eta' = \mathsf{reset}\ D\ \mathsf{in}\ \eta$

(d) $\eta' \models Inv(\ell')$

delay transition: $\langle \ell, \eta \rangle \xrightarrow{d} \langle \ell, \eta{+}d \rangle$ for $d \in \mathbb{R}_{\geqslant 0}$

(e) if $\eta{+}d \models Inv(\ell)$

Example: a Light Switch



The transition system $TS(Switch)$ has the state space

$$S = \{\langle off, t\rangle \mid t \in \mathbb{R}_{\geqslant 0}\} \cup \{\langle on, t\rangle \mid t \in \mathbb{R}_{\geqslant 0}\}$$

$TS(Switch)$ has the following transitions

$$\begin{array}{lcll}
\langle off, t\rangle & \xrightarrow{d} & \langle off, t+d\rangle & \text{for all } t \geqslant 0 \text{ and } d \geqslant 0 \\
\langle off, t\rangle & \xrightarrow{switch_on} & \langle on, 0\rangle & \text{for all } t \geqslant 0 \\
\langle on, t\rangle & \xrightarrow{d} & \langle on, t+d\rangle & \text{for all } t \geqslant 0 \text{ and } d \geqslant 0 \text{ with } t+d \leqslant 2 \\
\langle on, t\rangle & \xrightarrow{switch_off} & \langle off, t\rangle & \text{for all } 1 \leqslant t \leqslant 2.
\end{array}$$

The set of reachable states in $TS(Switch)$ from state $\langle off, 0\rangle$ is

$$\{\langle off, t\rangle \mid t \in \mathbb{R}_{\geqslant 0}\} \cup \{\langle on, t\rangle \mid 0 \leqslant t \leqslant 2\}$$

$$\langle off, 0\rangle \xrightarrow{0.57} \langle off, 0.57\rangle \xrightarrow{switch\_on} \langle on, 0\rangle \xrightarrow{\sqrt{2}} \langle on, \sqrt{2}\rangle \xrightarrow{0.2}$$

$$\langle on, \sqrt{2}{+}0.2\rangle \xrightarrow{switch\_off} \langle off, \sqrt{2}{+}0.2\rangle \xrightarrow{switch\_on} \langle on, 0\rangle \xrightarrow{1.7} \langle on, 1.7\rangle \ldots$$

Timelock and Zeno Paths

- **Time convergent paths**: time does not progress beyond a value
  - Example: path that visits infinitely many states in the interval [1/2, 1]:

$$\langle off, 0\rangle \xrightarrow{2^{-1}} \langle off, 1-2^{-1}\rangle \xrightarrow{2^{-2}} \langle off, 1-2^{-2}\rangle \xrightarrow{2^{-3}} \langle off, 1-2^{-3}\rangle \ \ldots\ldots$$

- In time *divergent* (i.e., non-convergent) paths, time is progressing "to infinity"

- **Timelock**: State s in TS(TA) contains a timelock if there is no time-divergent path starting in s.
  - Such states are unrealistic as time cannot progress for ever from these states.
  - Timelocks are undesired and need to be avoided when modeling a time-critical system by means of a timed automaton.

- **Zeno paths**: An infinite path fragment π in TS(TA) is *zeno* if it is time-convergent and infinitely many actions α ∈ Act are executed along π.
  - A Zeno path represents nonrealizable behavior, since it would require infinitely fast processors.

Example: Zeno paths for the light switch

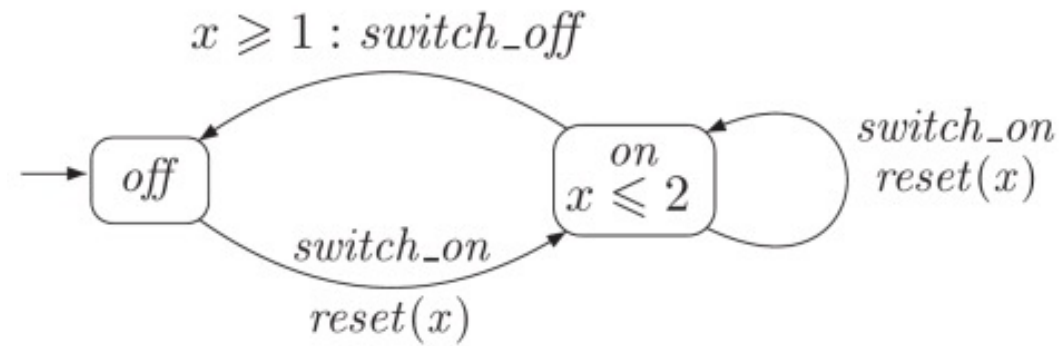
- A version of the Switch where the user may push the button when light is on, to extend it for other two minutes
- However, user may press the *on* button infinitely fast, or just faster and faster:

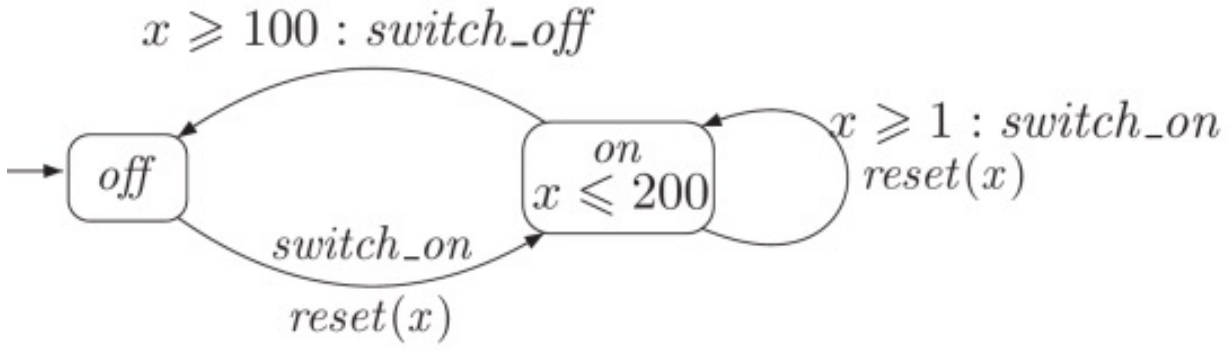
$$\langle off, 0\rangle \xrightarrow{switch\_on} \langle on, 0\rangle \xrightarrow{switch\_on} \langle on, 0\rangle \xrightarrow{switch\_on} \langle on, 0\rangle \xrightarrow{switch\_on} \ldots$$

$$\langle off, 0\rangle \xrightarrow{switch\_on} \langle on, 0\rangle \xrightarrow{0.5} \langle on, 0.5\rangle \xrightarrow{switch\_on} \langle on, 0\rangle \xrightarrow{0.25} \langle on, 0.25\rangle \xrightarrow{switch\_on} \ldots,$$

- Both behaviors are of course impossible.
- Just add a non-zero delay between successive button pushings: Inv(on) = $x \geq c$ for some natural number $c>0$
  - NB: we use natural numbers in clock constraints, but *rational* numbers could be used, since they can be scaled up to integers, e.g. if c=1/100, just multiply every constant by 100.

The «Zeno Switch» and its (rescaled) correction





Diagonal constraints

- What about introducing more general constraints?
- *Diagonal constraints* have the form $x<y+c$, $x\leq y+c$, $x=y+c$
- They can be replaced by storing information in the states about the last reset of the two clocks
  - This means introducing new states
  - $x<y$ if last reset of x x was after last reset of y;
  - $x=y$ if their last reset was at the same time
  - …
- More general arithmetic constraints (e.g., $x+y<z+2$) lead to undecidability of emptiness

ANALYZING TIMED AUTOMATA

Exhaustive exploration of the state space

- To check whether properties hold for a system modeled through a TA, we should perform an exhaustive exploration of the state space of the TA
- "State" of a TA: ⟨location, clock evaluation⟩
- Set Eval(C) of possible clock interpretations: $R_{\geq 0}^{|C|}$
  - it's an *infinite* set
- We need some *finite* abstraction of the state space to be able to exhaustively explore it
  - **clock regions**

Clock regions: intuition

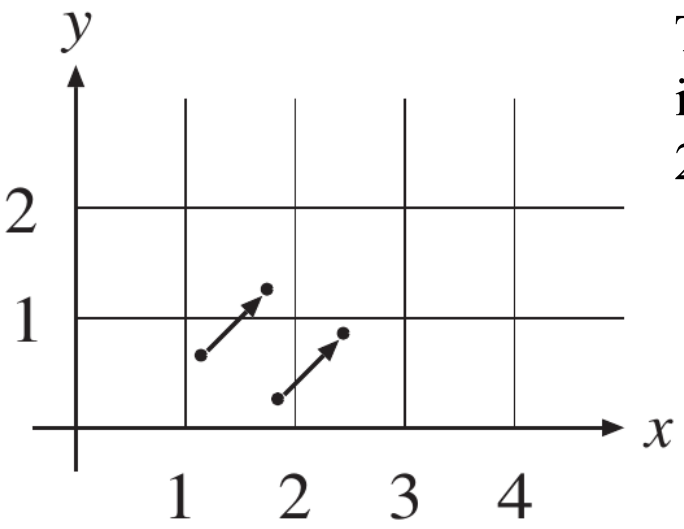
- Two clock evaluation η_1 and η_2 are in the same region iff
  - they agree on the integral parts of the clock values
    - i.e. for each clock $x \in C$, the values $\eta_1(x)$ and $\eta_2(x)$ have the same integral part
  - they agree on the ordering of the fractional parts of all clock values
    - this means that if we have two clocks x and y, if starting from η_1 they will reach the next integer values in some order, they will reach them in the same order starting from η_2

Ex: $\eta_1 = \{x \mapsto 1.75, y \mapsto 7.83\}$, $\eta_2 = \{x \mapsto 1.24, y \mapsto 7.78\}$

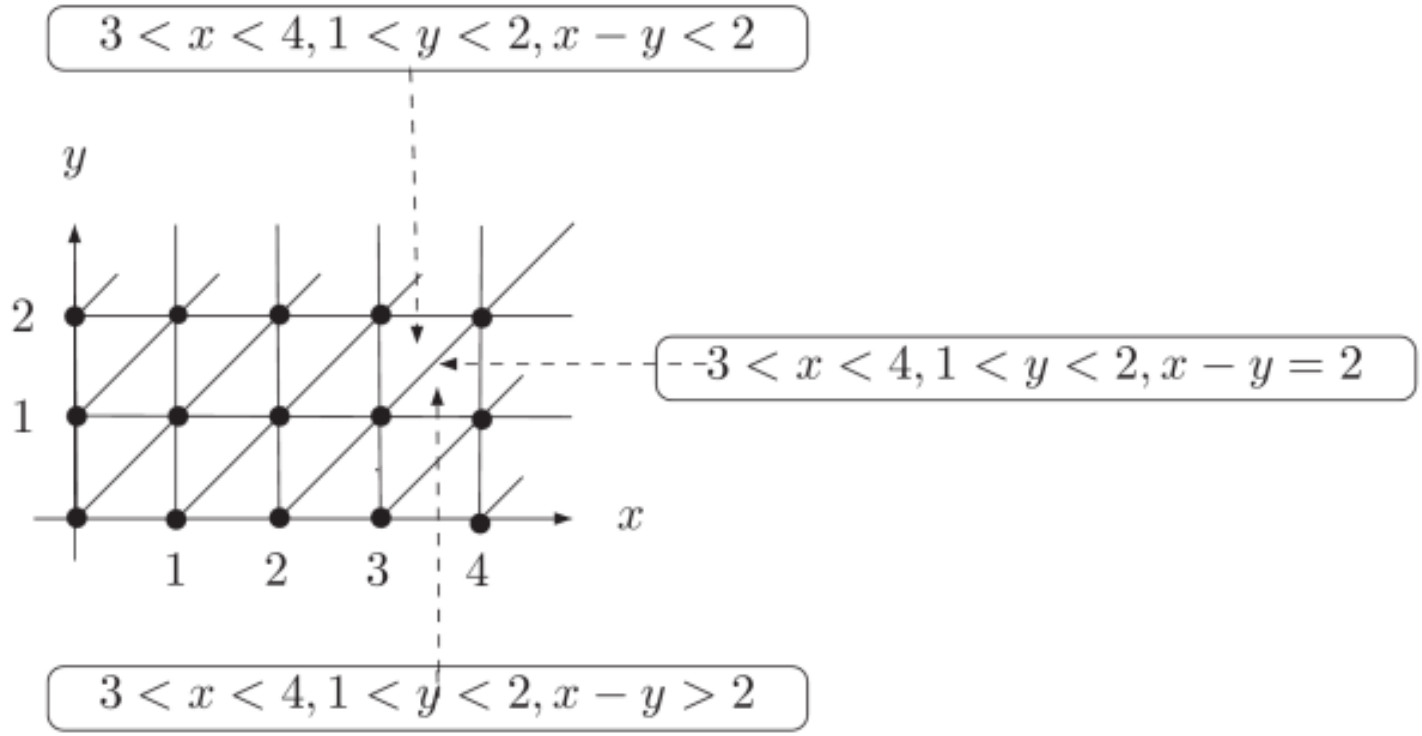
are in the same clock region, while:

$\eta'_1 = \{x \mapsto 1.86, y \mapsto 7.83\}$, is not in the same clock region of η_2 (do not agree on fractional part)

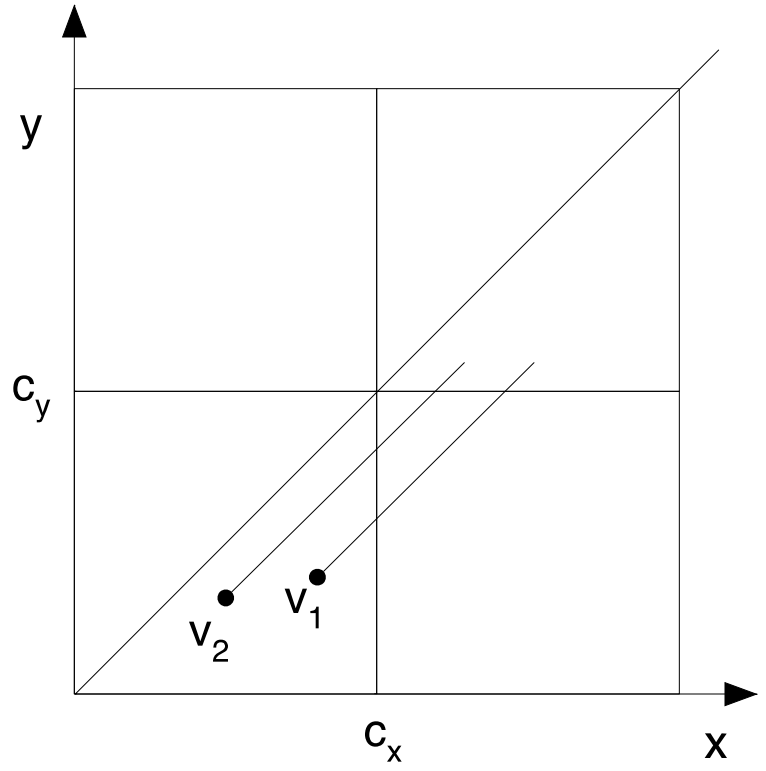
The case for considering the order of the fractional parts



The successors of the two points in the same square are different: it is impossible to modify the delays so the leftmost point goes to the region 2<x<3, 0<y<1. The squares are not fine enough.



Time advancement from clock regions



Clock region: formal definition

- Given a set C of clocks, for each x $\in$ C let c_x be the largest integer with which x is compared
  - i.e. c_x is the largest value c s.t. either x$\leq$c or c$\leq$x appears in CC(C)
- Def: equivalence relation $\simeq$ over the set of clock valuations:
  η, η' clock evaluations for C: $\eta \simeq \eta'$ iff:

  for any $x \in C$ it holds that $\eta(x) > c_x$ and $\eta'(x) > c_x$, or

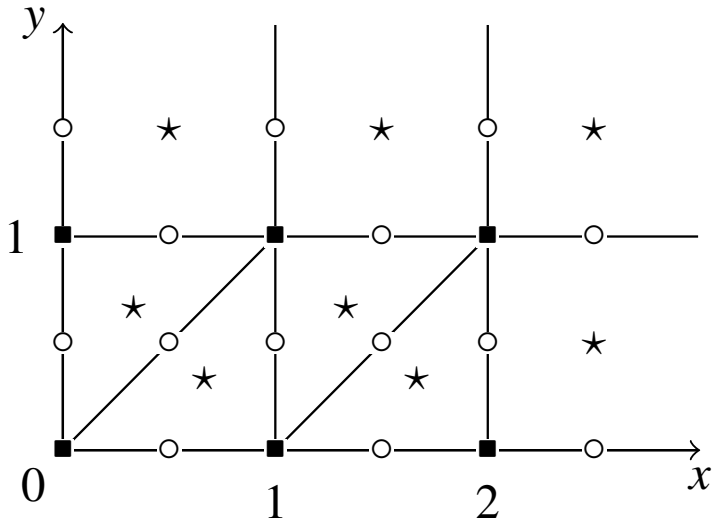
  for any $x, y \in C$ with $\eta(x), \eta'(x) \leqslant c_x$ and $\eta(y), \eta'(y) \leqslant c_y$ all the following conditions hold:

  - $\lfloor \eta(x) \rfloor = \lfloor \eta'(x) \rfloor$ and $frac(\eta(x)) = 0$ iff $frac(\eta'(x)) = 0$,
  - $frac(\eta(x)) \leqslant frac(\eta(y))$ iff $frac(\eta'(x)) \leqslant frac(\eta'(y))$.

- Def: We define a clock region as the *equivalence class* of clock evaluations induced by the equivalence relation $\simeq$

Clock regions: example

- Consider C = {x,y}, $c_x = 2$ and $c_y = 1$, we have the following 28 clock regions



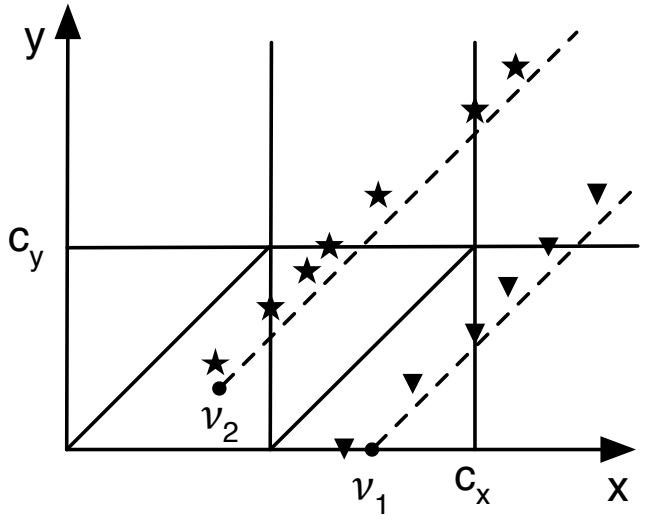
- Given a clock interpretation η, let $[\eta]$ its corresponding clock region
  - e.g., the clock region $[\eta_1]$ of evaluation $\eta_1 = \{x \longmapsto 1.75, y \longmapsto 0.83\}$ is the one s.t. $1 < x < 2$ and $0 < y < 1$ and frac(y) > frac(x)

Towards a “region -based” automaton

- Using clock regions, we can give a finite representation to the possible states $\langle l_i, \eta_i \rangle$ in which TS(TA) can be
- Key idea: a state is a pair $\langle l_i, [\eta_i] \rangle$
  - for a given TA, there is a *finite* number of clock regions $[\eta]$
  - Finite because there is a max constant in the constraint
- Given a TA, we build a TS whose set of states is made of pairs $\langle l_i, [\eta_i] \rangle$

Time-successor relation

- To define the transitions of the region-TS, we need to define the successors of a clock region
- **Def:** a clock region α' is a time-successor of a clock region α iff for each $\eta \in \alpha$ there is $t \in R_{\geq 0}$ s.t. $\eta + t \in \alpha'$
  (i.e, every lock region that can be reached along a diagonal)



Time-abstracted bisimulation

- The idea is that, from two equivalent states, the automaton can take the same transitions, except that the values of the delays might have to be changed.
  - It must be possible to adjust the delays to go the same region
- Clock equivalence is a special case of a time-abstracted bisimulation:

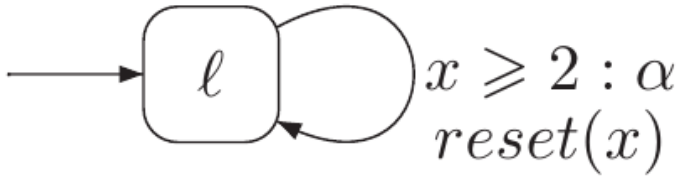
Definition 2 A relation R on the states of A is a *time-abstracted bisimulation* if $(\ell_1, v_1)R(\ell_2, v_2)$ and $(\ell_1, v_1) \xrightarrow{d_1,a} (\ell'_1, v'_1)$ for some $d_1 \in \mathbb{R}_{\geq 0}$ and $a \in \Sigma$ imply $(\ell_2, v_2) \xrightarrow{d_2,a} (\ell'_2, v'_2)$ for some $d_2 \in \mathbb{R}_{\geq 0}$, with $(\ell'_1, v'_1)R(\ell'_2, v'_2)$ and vice versa.

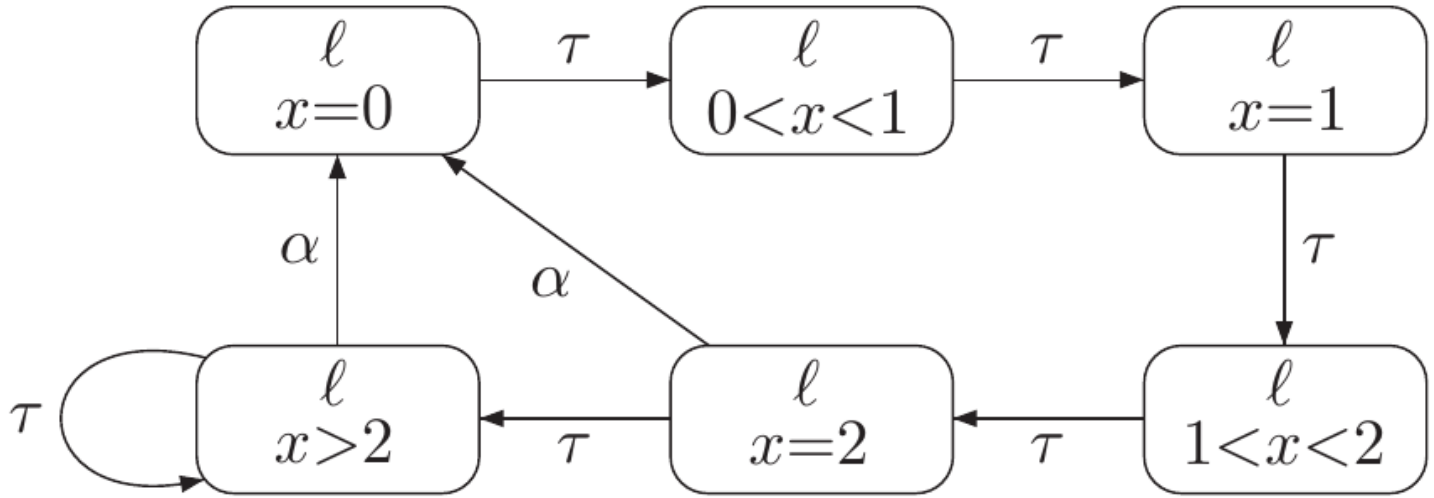
Clock Equivalence is a Bisimulation

- **Theorem**. *Clock equivalence is a bisimulation equivalence over AP'*.
- Hence, we can build the quotient TS of TS(TA), called the **Region TS of the TA**, *RTS(TA)*.
- Since clock equivalence is a relation of finite-index, then clock regions are finite; also locations are finite, therefore *RTS(TA)* is finite!

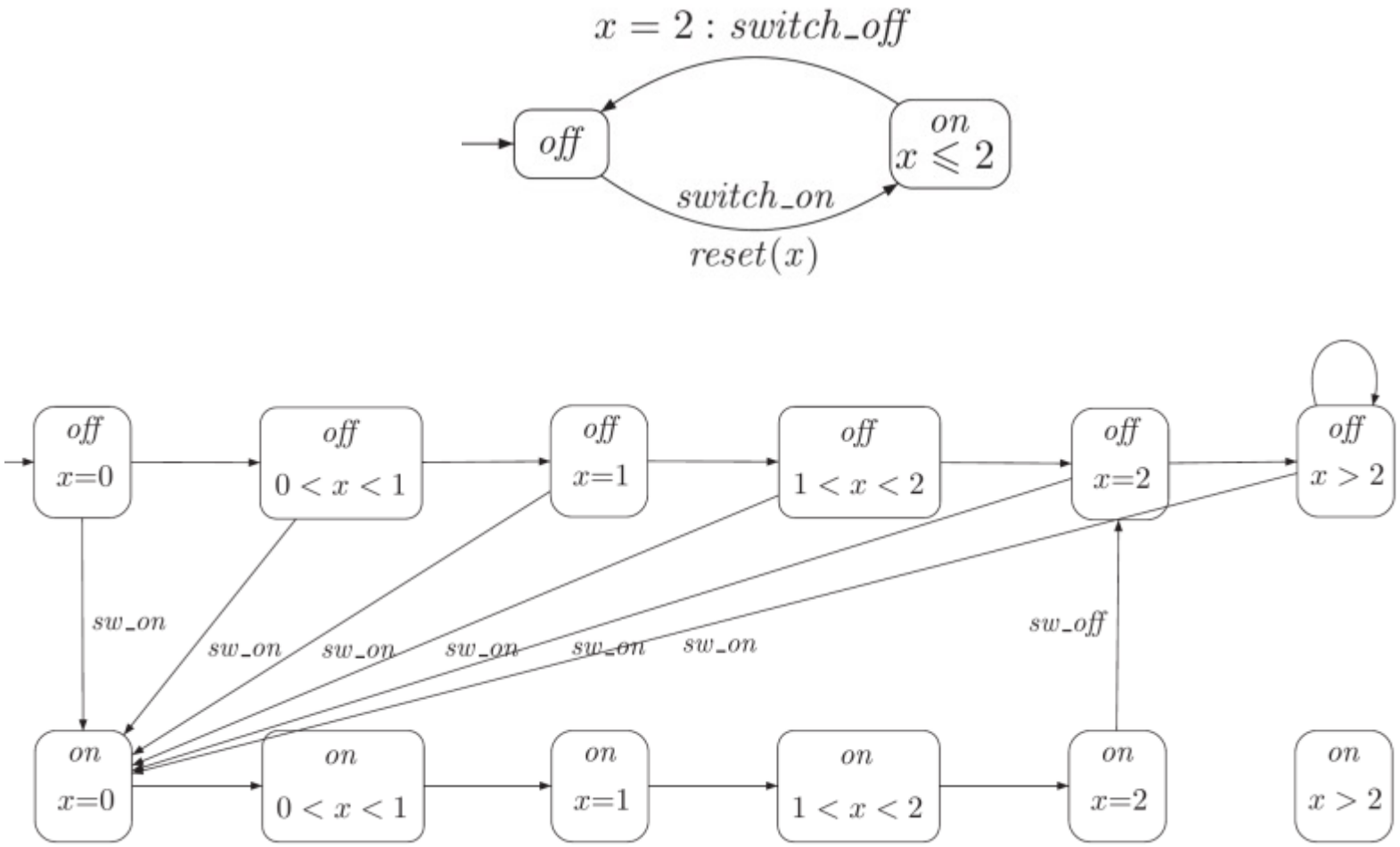
- Bisimulation equivalence ensures that:
  - Every path in the infinite TS(TA) has a corresponding path in the finite RTS(TA)
  - Every path in RTS(TA) corresponds to an infinite set of paths of TS(TA)
  - … and also CTL-style formulae are preserved!

Example


ℓ
$x \geqslant 2 : \alpha$
$reset(x)$


ℓ
$x{=}0$
τ
ℓ
$0{<}x{<}1$
τ
ℓ
$x{=}1$
α
α
τ
τ
ℓ
$x{>}2$
τ
ℓ
$x{=}2$
τ
ℓ
$1{<}x{<}2$

Another example



Example with two clocks