VERIFICA DI PROCESSI CONCORRENTI
VPC 19-20

Timed models

Prof.ssa Susanna Donatelli
Universita' di Torino
www.di.unito.it
susi@di.unito.it

Reference material books:

Concepts, Algorithms, and Tools

for

Model Checking

Joost-Pieter Katoen

Lehrstuhl für Informatik VII

Friedrich-Alexander Universität Erlangen-Nürnberg

Lecture Notes of the Course

"Mechanised Validation of Parallel Systems"

(course number 10359)

Semester 1998/1999

Prof. Jost-Pieter Katoen

(University of Aachen, D)

Acknowledgements

Transparencies adapted from the course notes and trasparencies of

- Prof. Jost-Pieter Katoen, University of Aachen (Germany)
- ....

Dealing with time

Why is time introduced in formalisms for system verification?

- Correctness may depend also on time (think of the operations of a pipelined CPU)
- Usefulness may depend on time (when I call the lift, when the lift will come , or I want to compute how long does a production line takes)

To check a timed property the time should be explicitely represented in the model

We can check also untimed property on a timed model (example: reachability of a marking in a timed Petri net)

Dealing with time

What is a time specification

1. A value → fixed delay for an activity
2. An interval (min-max) → the duration of an activity is a non deterministic value in the interval
3. A stochastic distribution → the duration of an activity is a value is extracted from a distribution
4. The possibility of defining clocks as variables that increase constantly
5. A mix of the above

Dealing with time

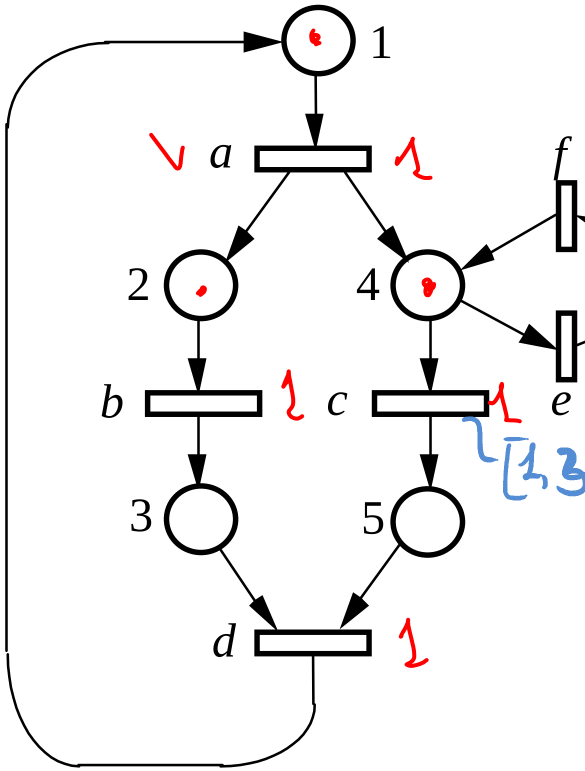
Continuous or discrete?

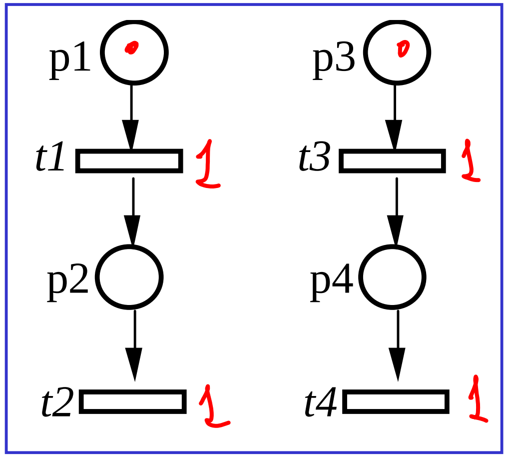
Discrete:

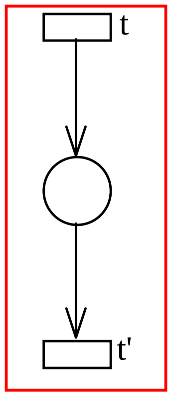
- time is a discrete entity
- time elapses in regular ticks
- events/activities can happen only at ticks
- between two ticks the system stays unchanged
- used to represent synchronous system (system with a global discrete clock)
- can represent an abstraction of a continuous system
- ex1: imagine a discrete time Petri net, in which all firing have equal duration
- ex2: imagine a discrete time Petri net, in which firings have different discrete durations

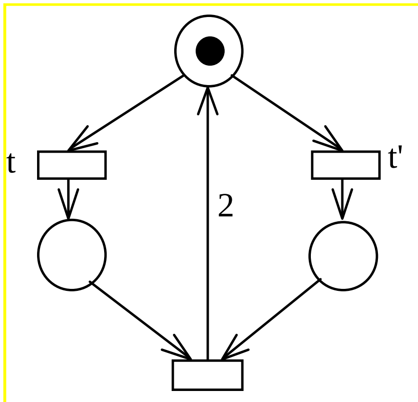
PN examples

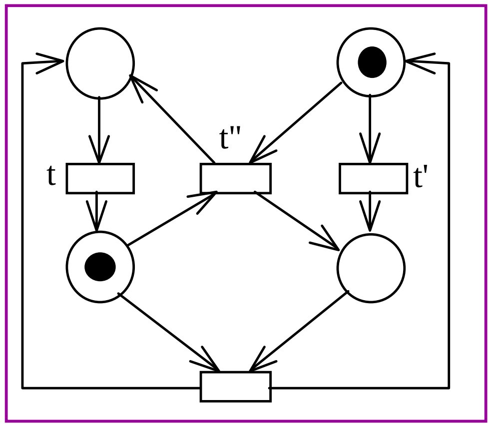
p1 + p3 t=0
t1+t3 t=1
P2+P4
t2+t4 t=2
∅



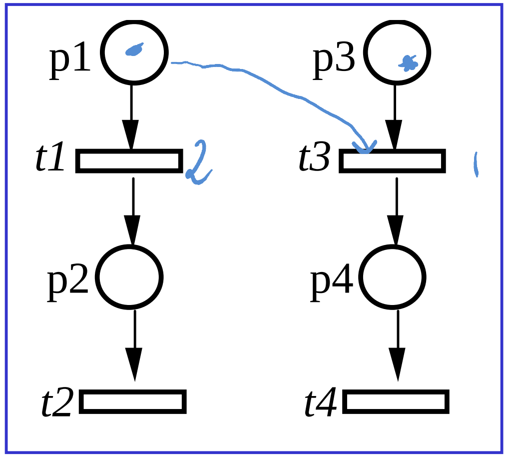








PN examples



STEP

p1 + p3 t=0

↓ t=1

p2 + p4

↓ t=2

∅

INTERLEAVING

p1 + p3

t1 ↙ ↘ t3

p2 + p3 p1 + p4

t3 ↘ ↙ t1

p2 + p4

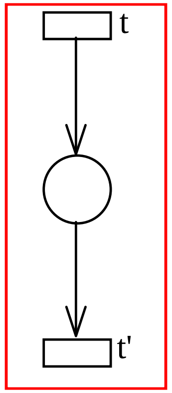
↓ 1 unità

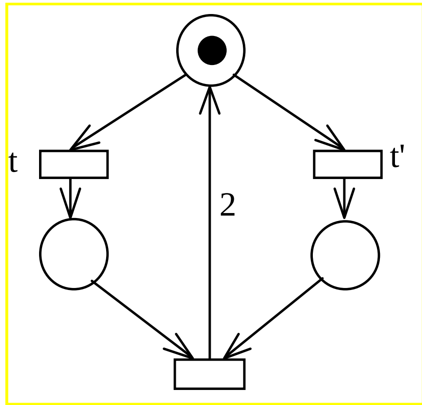
t = 1

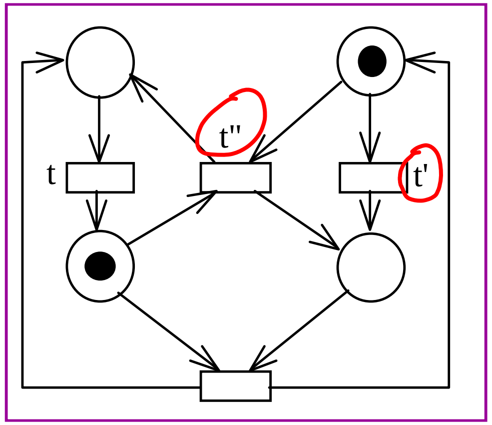
↓ tempo ∅

t = 1

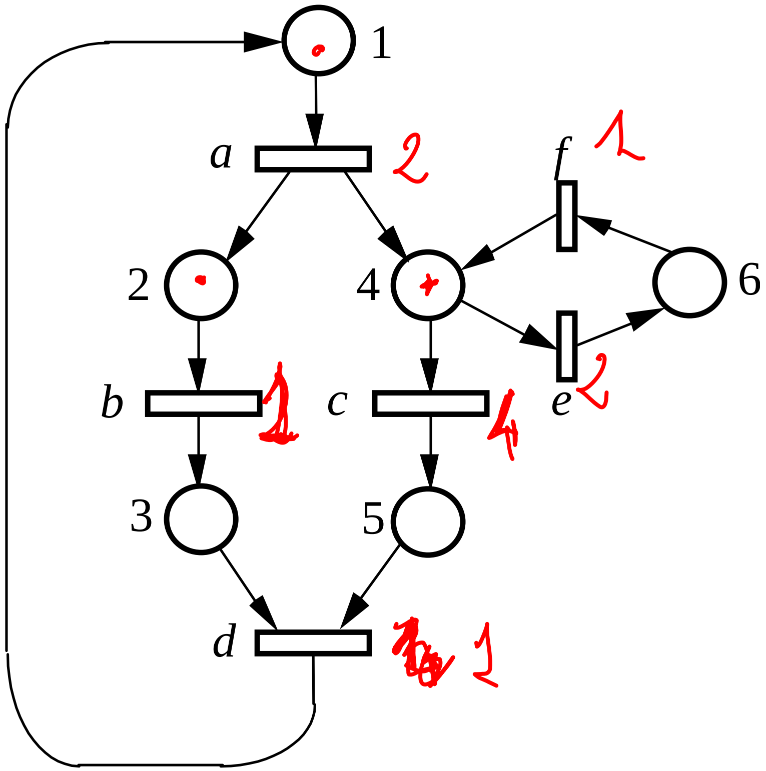
PN examples



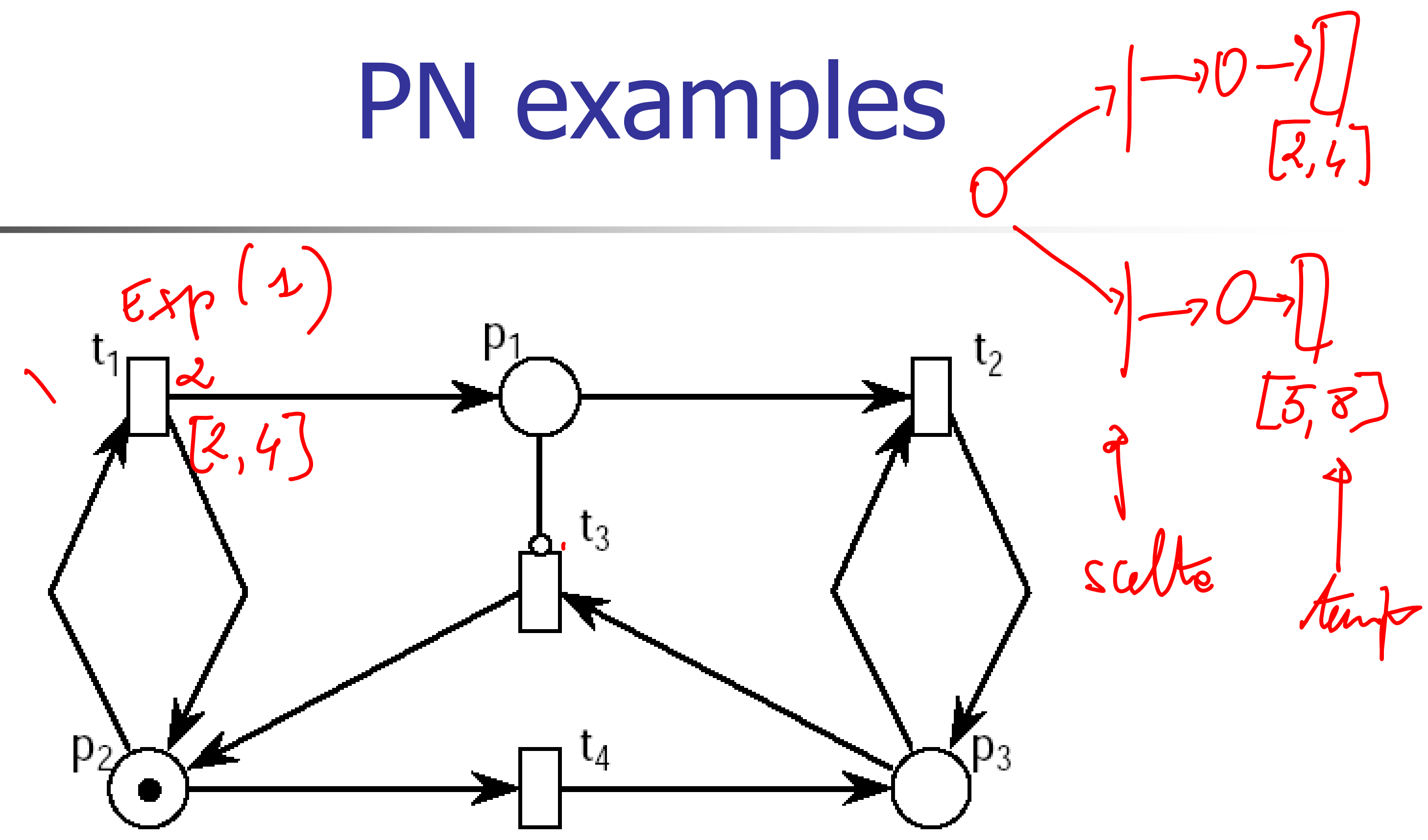




PN examples



PN examples



Property of interest: how often does the lazy chap cooks, for how long does it cook?

Exists an execution in which he eats only for X unit time?

Dealing with time

What is the state of the system?

- value of the variables plus value of clocks
- marking plus value of clocks
- process algebra terms plus the value of clocks

How many states do we have?

How do we express temporal properties?

- use temporal logic without time (X "accounts" for time)
- use temporal logic in which temporal operators have a time interval $A[\varphi \, \mathcal{U}^{[t1,t2]} \psi]$

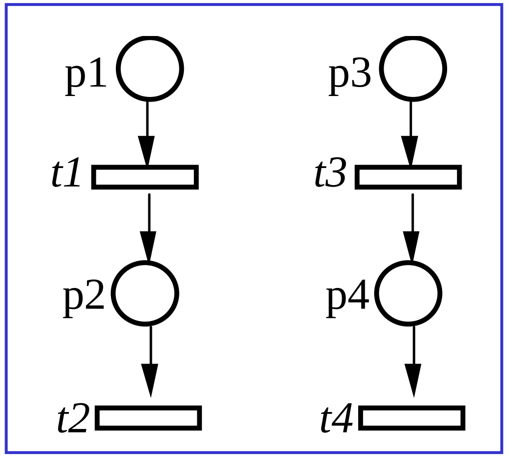
Dealing with time

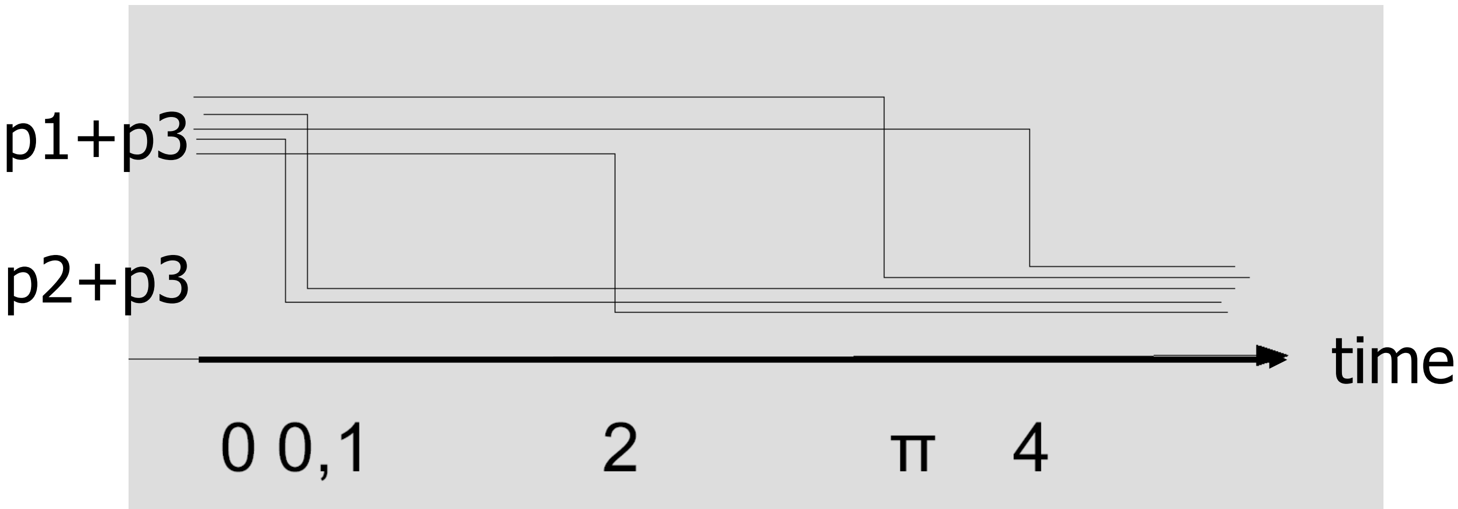
Continuous or discrete?

Continous:

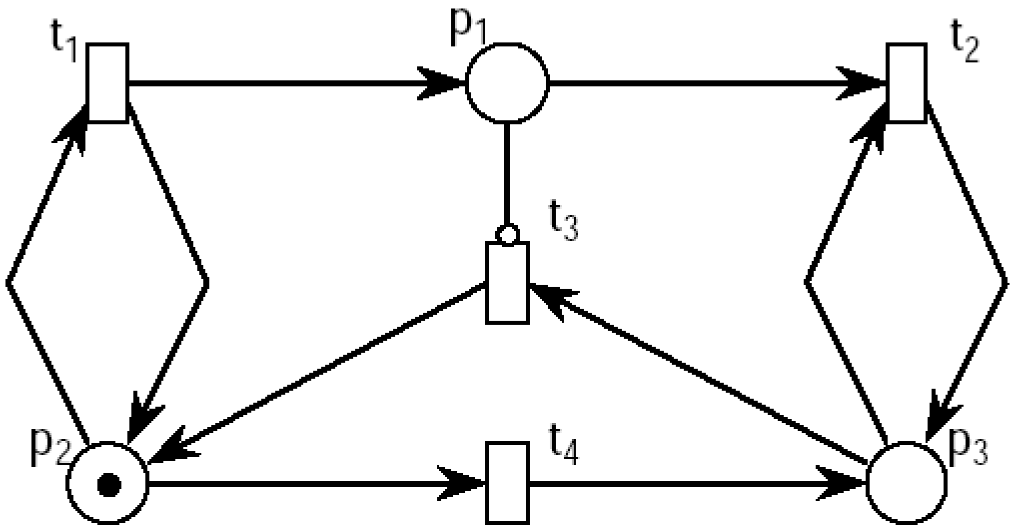
- time is a continuous entity (modelled as a real non-negative variable)
- time elapses continuously
- events/activities can happen at any instant of time
- time between two events can be arbitrarily small
- used to represent asynchronous system
- ex1: imagine a continuous time Petri net, in which all activities have equal duration
- ex2: imagine a continuous time Petri net, in which activities have different durations
- ex2: imagine a continuous time Petri net, in which activities have different durations, chosen non deterministically in a given interval

PN examples





PN examples



Property of interest: how often does the lazy chap cooks, for how long does it cook?

Exists an execution in which he eats only for X unit time?

Dealing with time

What is the state of the system?

- value of the variables plus value of clocks
- marking plus value of clocks
- process algebra terms plus the value of clocks

How many states do we have?

How do we express temporal properties?

- use temporal logic in which temporal operators have a time interval $A[\varphi U^{[t1,t2]} \psi]$

Some timed formalisms and logics

Timed automata

Timed Petri nets

Timed process algebra

Semantics is given in terms of Timed transition system, the system of timed executions

Timed CTL (TCTL) example: a leader is elected within 3 seconds

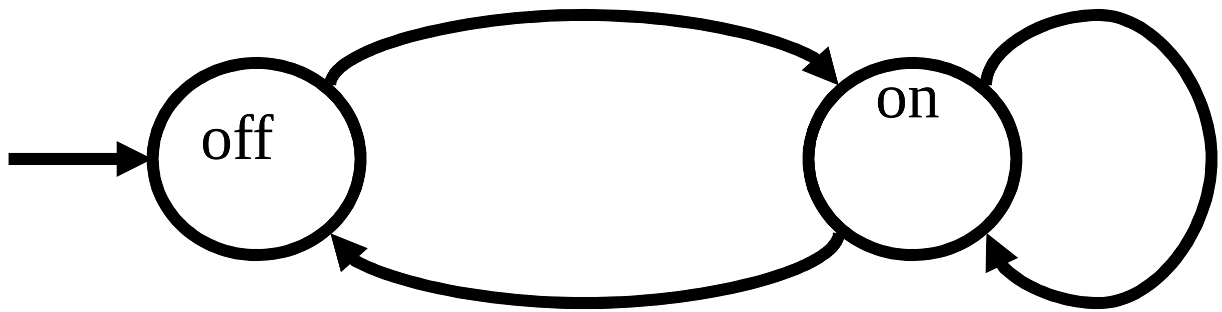
Note: there is no probabilistic reasoning, only non determinism and "possibility"

Timed automata part

- Syntax of timed automata
- State of a finite automata, execution paths and timed transition systems
- Semantics of timed automata (in terms of a timed transition system)
- TCTL syntax and semantics
- Model checking TCTL
- Timed automata and temporal logic in Uppaal

Timed automata

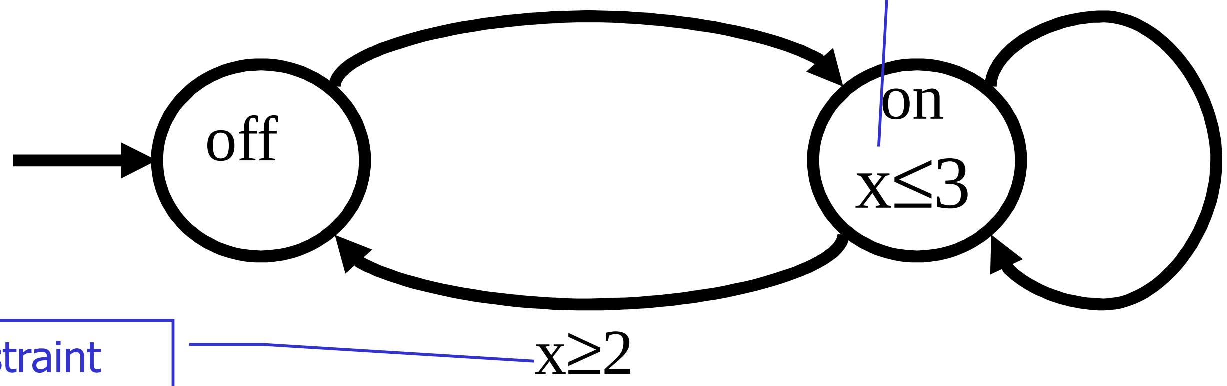
- Finite-state graph with locations and edges
  + clock variables
  + ……
- Time elapses in location, not in edges
- Example: light switch, with clock x



Timed automata

- Finite-state graph with locations and edges
  + clocks variables (run at the same speed)
  + clock constraints that "constrain" the behaviour (examples: $x \leq 3$, $x - y > 5$)

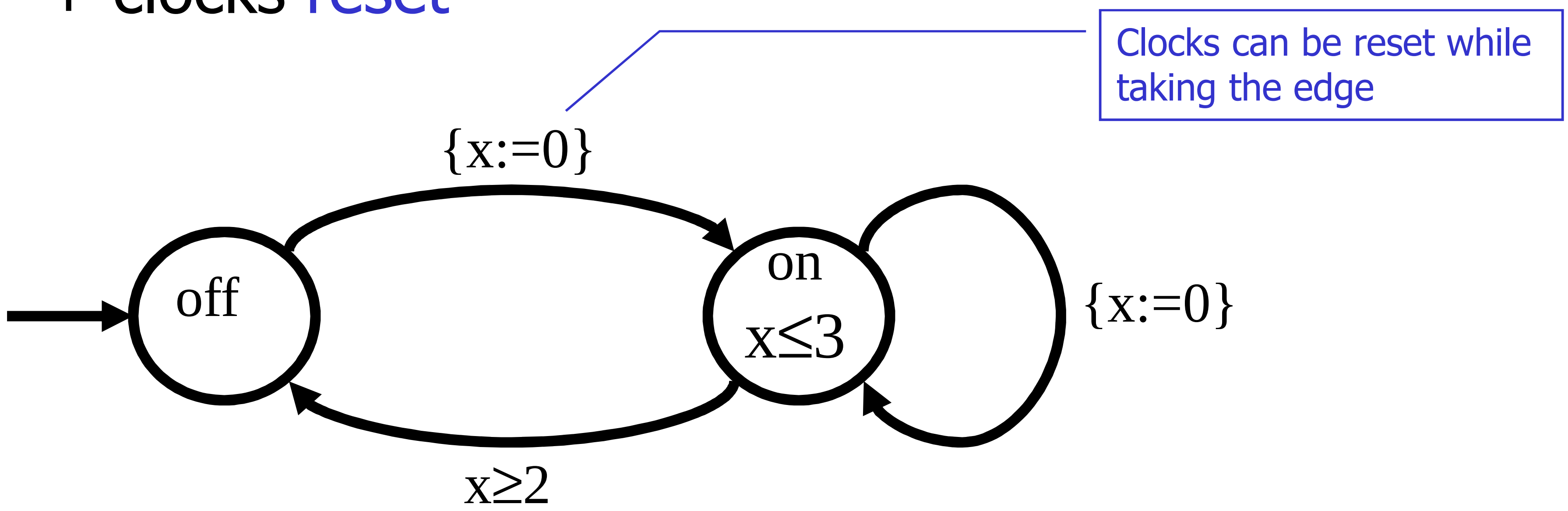
A clock constraint can be an invariant for a location



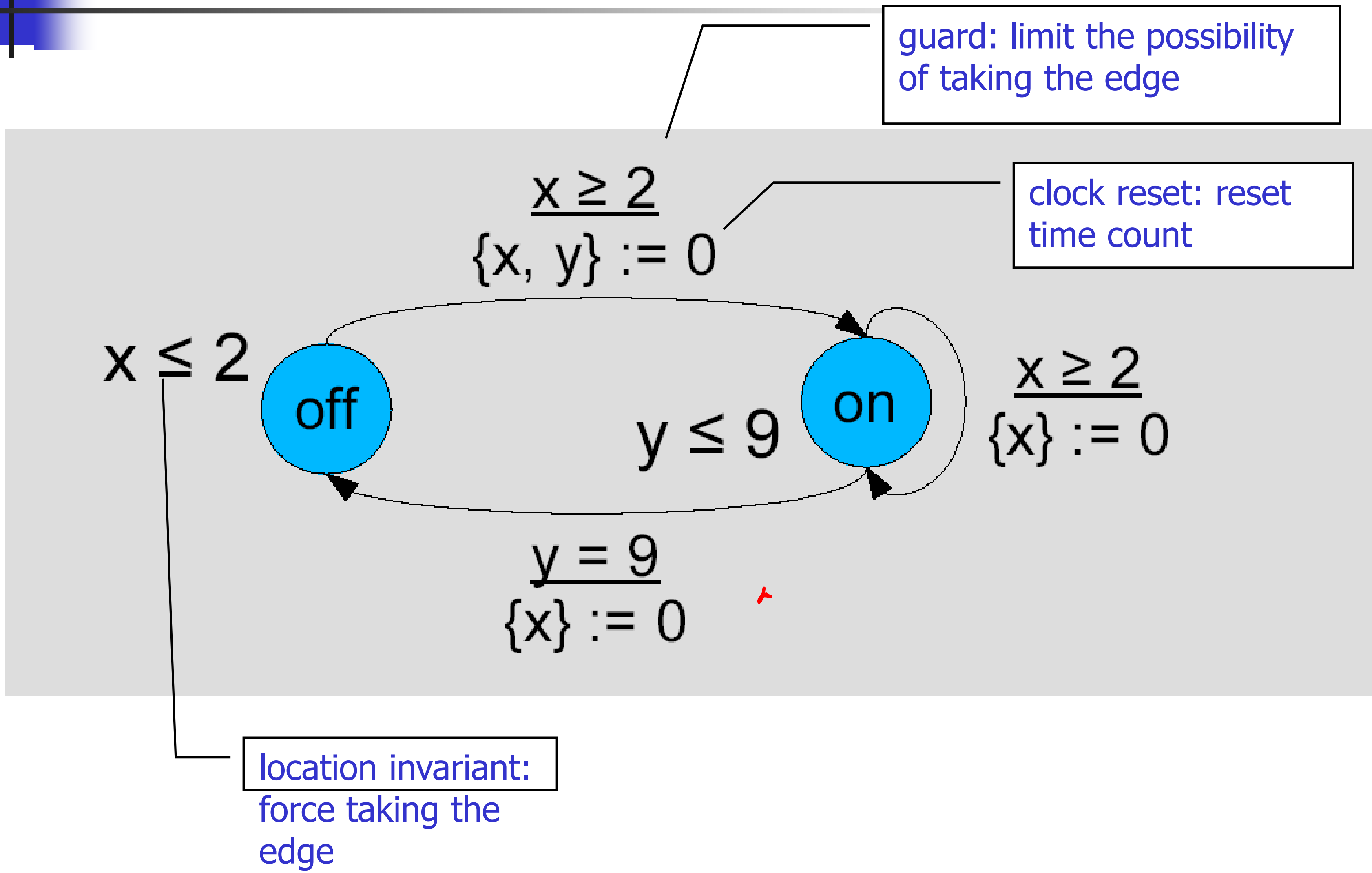
A clock constraint can be a guard on an edge

Timed automata

- Finite-state graph with locations and edges
  + clocks variables (run at the same speed)
  + clock constraints
  + clocks reset

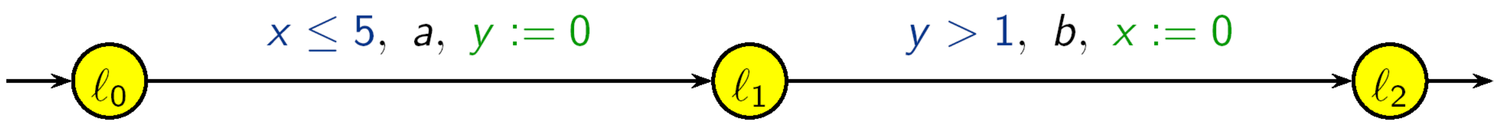


Timed automata



Timed automata

x, y clocks



A possible execution

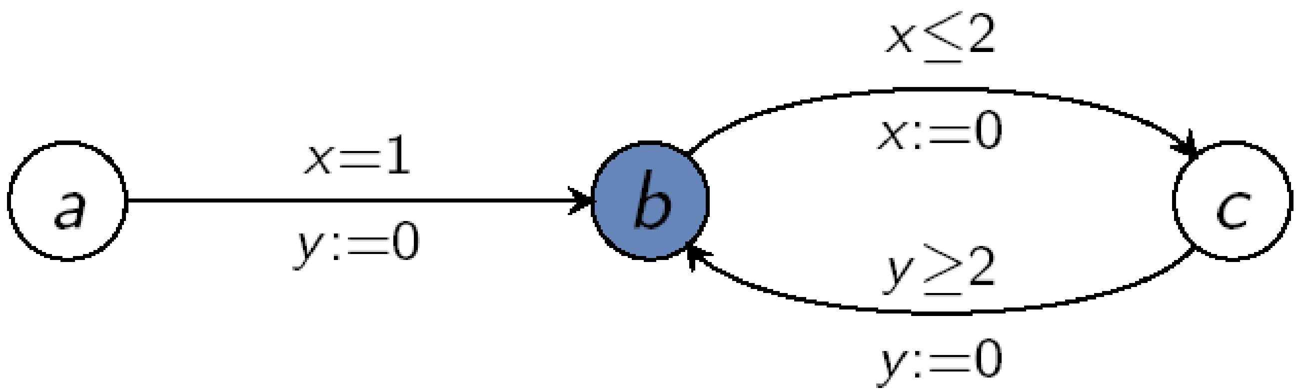
δ(0.1) · δ(6.01) ·

	ℓ_0	$\xrightarrow{\delta(4.1)}$	ℓ_0	$\xrightarrow{a}$	ℓ_1	$\xrightarrow{\delta(1.4)}$	ℓ_1	$\xrightarrow{b}$	ℓ_2
x	0		4.1		4.1		5.5		0
y	0		4.1		0		1.4		1.4

Clock valuation

stato del timed automata

Timed automata: another example



Timed automata

Def.: a clock is a variable ranging over R^+

Def. Clock constraints. Let C be a set of clocks, with $x \in C$ and c a natural value, then

1. $x < c$ and $x \leqslant c$ are clock constraints
2. If α is a clock constraint, then $\neg\alpha$ is a clock constraint
3. If α and β are clock constraints, then $\alpha \wedge \beta$ is a clock constraint
4. Anything else is not a clock constraint.

The set of clock constraints over C is indicated with $\Psi(C)$ or Cstr(C)

Timed automata

Note: adding x+y to clock constraints makes the MC undecidable (and x-y)?

Note: taking c over the real makes the MC undecidable (and for rational?)

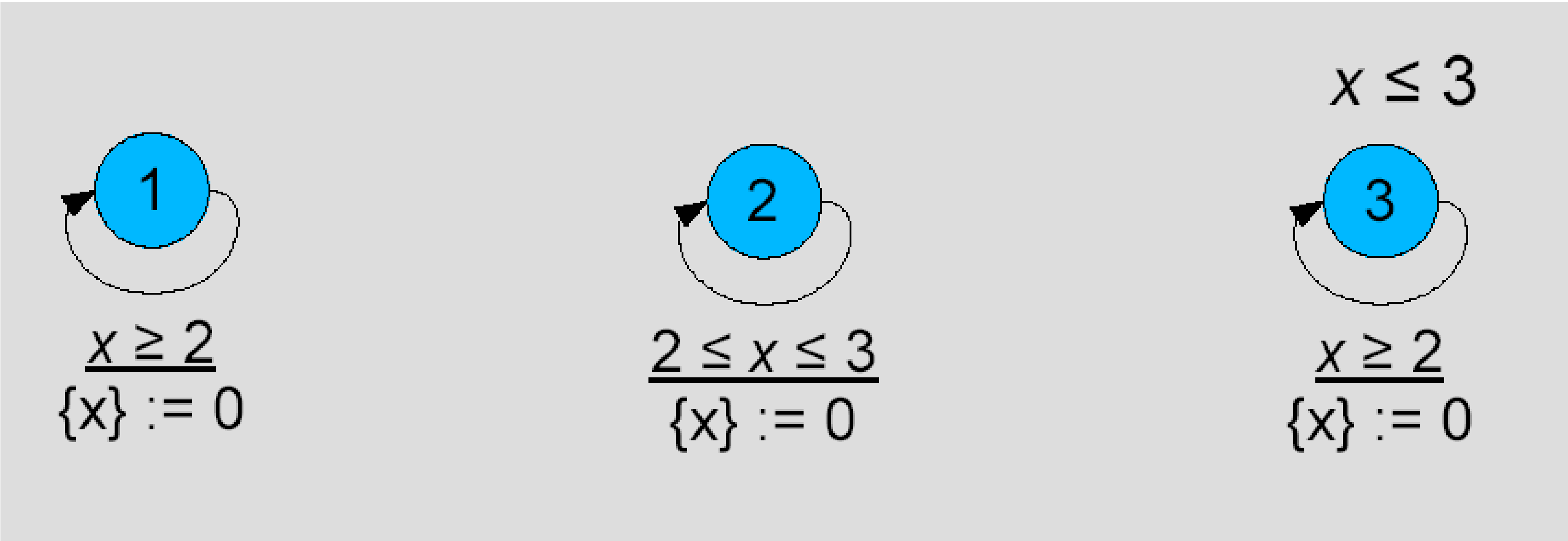
Definition of timed automata

Def.: A timed automata *A* is a tuple (L, l_0, E, Label, C, clocks, guard, inv) with

- L a non empty and finite set of locations with initial location l_0
- $E \subseteq L \times L$, a set of edges
- Label: L--> 2^{AP} a function that assigns to each location a set Label(l) of atomic propositions
- C, a finite set of clocks
- clocks: E --> 2^C, a function that assign to each edge $e \in E$ a set of clocks clocks(e) --*clocks to be reset*
- guard: E --> Cstr(C), a function that assign to each edge $e \in E$ a clock constraint guard(e)
- inv: L --> Cstr(C), a function that assign to each location $l \in L$ a clock constraint inv(l)

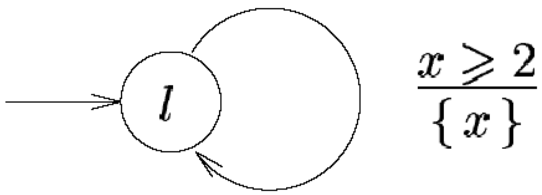
Timed automata

Guards or invariants

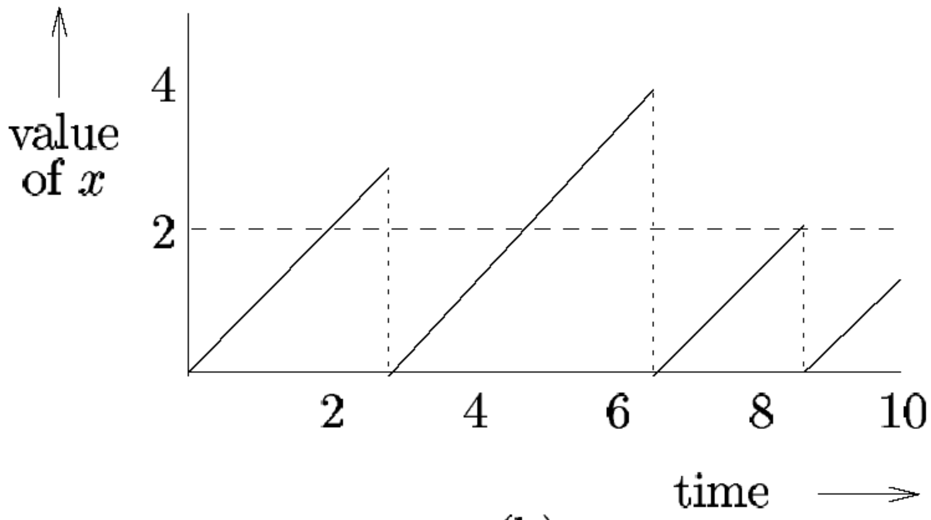


Timed automata

Time diagram



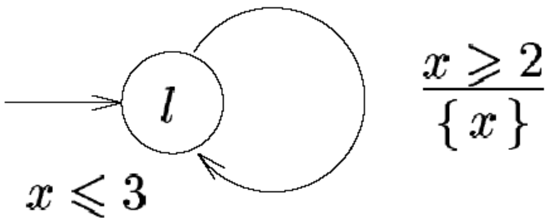
(a)



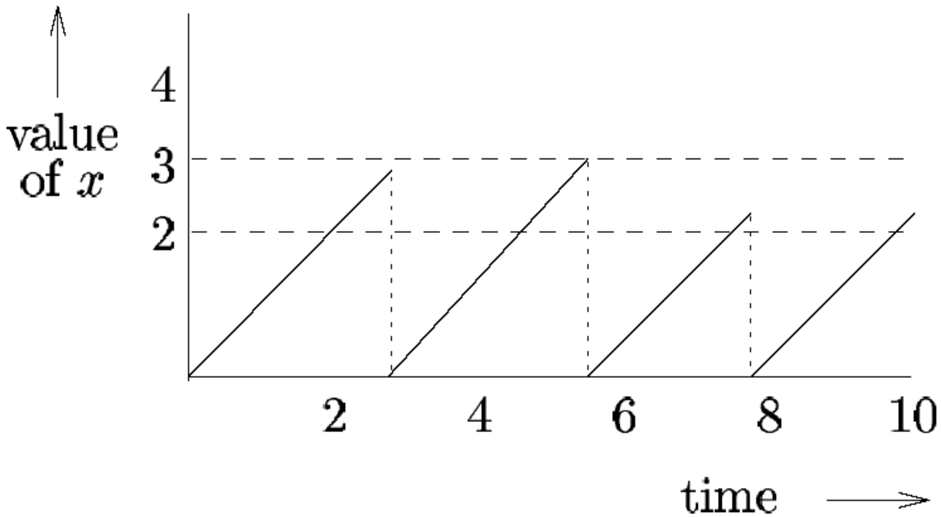
(b)

Timed automata

Time diagram



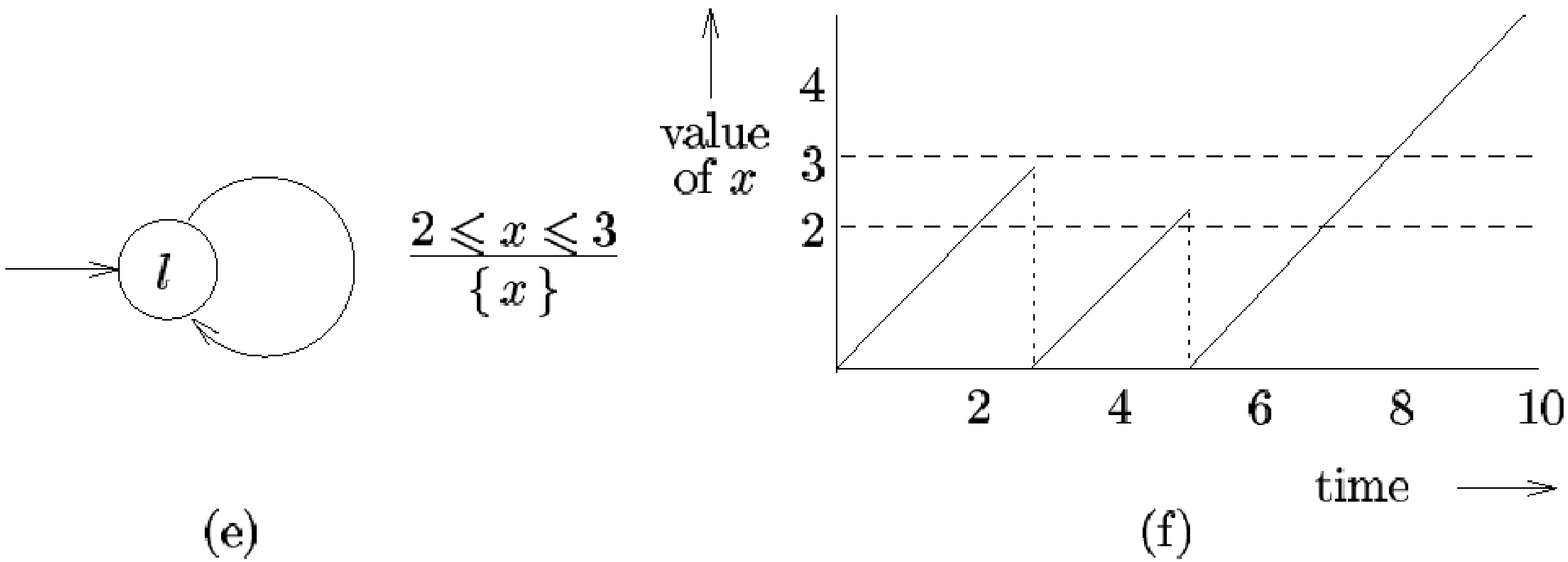
(c)



(d)

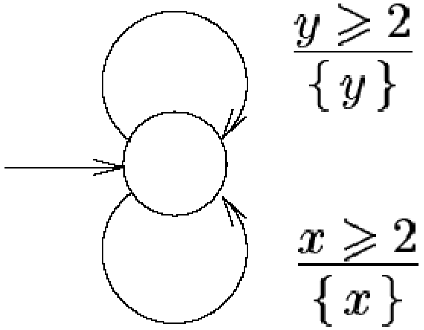
Timed automata

Time diagram

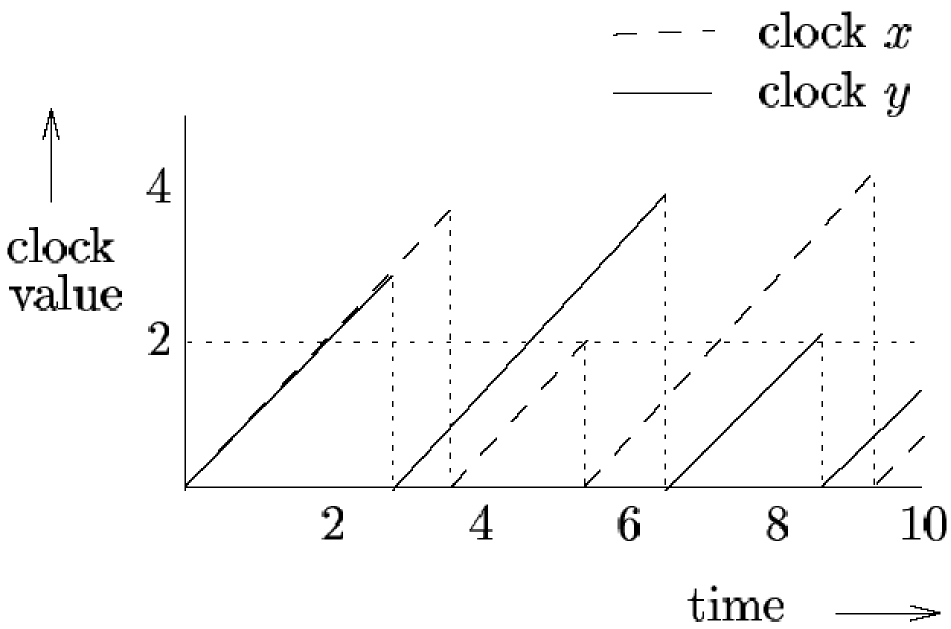


Timed automata

Time diagram



(a)



(b)

Timed automata

Def.: clock valuation v for a set of clocks C is a function

v: C-->R$^+$, assigning to each clock x in C its current value v(x)

Def.: Let V(C) denote the set of all clock valuations over C. A state of a timed automata *A* is a pair

(l,v)

with l a location of *A* and v a valuation over C, the clocks of *A*

For positive real *d*, v+*d* is the valuation where each clock is incremented by *d*. The valuation v with clock x reset is

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

Timed automata

Def.: evaluation of clock constraints. For $x \in C$, $v \in V(C)$, natural c and α and $\beta \in Cstr(C)$, we have

$$
\begin{array}{ll}
v \models x \leqslant c & \text{iff } v(x) \leqslant c \\
v \models x < c & \text{iff } v(x) < c \\
v \models \neg \alpha & \text{iff } v \not\models \alpha \\
v \models \alpha \wedge \beta & \text{iff } v \models \alpha \wedge v \models \beta.
\end{array}
$$

Timed Transition System (TTS)

Def.: Timed transition system underlying a timed automata *A*, *M*(*A*), is defined as (S,s_0,-->) where

- $S = \{ (l, v) \in L \times V(C) \mid v \models inv(l) \}$
- $s_0 = (l_0, v_0)$ where $v_0(x) = 0$ for all $x \in C$
- the transition relation $\longrightarrow \subseteq S \times (\mathbb{R}^+ \cup \{*\}) \times S$ is defined by the rules:

1. $(l, v) \xrightarrow{*} (l', \text{reset } clocks(e) \text{ in } v)$ if the following conditions hold:
   (a) $e = (l, l') \in E$
   (b) $v \models guard(e)$, and
   (c) $(\text{reset } clocks(e) \text{ in } v) \models inv(l')$
2. $(l, v) \xrightarrow{d} (l, v+d)$, for positive real d, if the following condition holds:

$$\forall d' \leqslant d.\, v+d' \models inv(l).$$

Path of a TTS

Def.: a path is an infinite sequence $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots$ where, for all *i*, $s_i \xrightarrow{a_i} s_{i+1}$ is a transition in the TTS

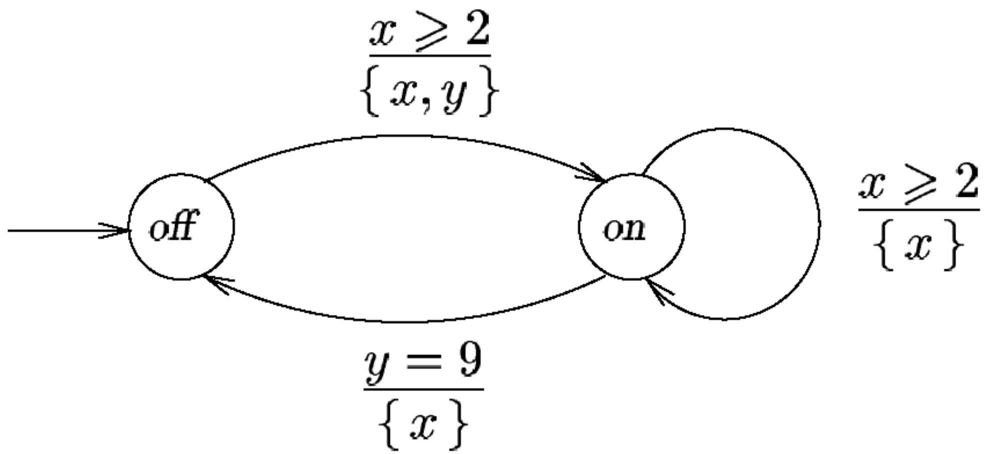
An execution of a timed automata *A* is a path through its timed transition system *M*(*A*).

The elapsed time on a path is defined as follow:

Definition 44. (Elapsed time on a path)
For path $\sigma = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots$ and natural i, the time elapsed from s_0 to s_i, denoted $\Delta(\sigma, i)$, is defined by:

$$\begin{aligned} \Delta(\sigma,0) &= 0 \\ \Delta(\sigma,i{+}1) &= \Delta(\sigma,i) + \begin{cases} 0 & \text{if } a_i = * \\ a_i & \text{if } a_i \in \mathbb{R}^+. \end{cases} \end{aligned}$$

Path of a TTS



$$\begin{aligned}\sigma \quad = \quad & (\text{off}, v_0) \xrightarrow{3} (\text{off}, v_1) \xrightarrow{*} (\text{on}, v_2) \xrightarrow{4} (\text{on}, v_3) \xrightarrow{*} (\text{on}, v_4) \\ & \xrightarrow{1} (\text{on}, v_5) \xrightarrow{2} (\text{on}, v_6) \xrightarrow{2} (\text{on}, v_7) \xrightarrow{*} (\text{off}, v_8) \ldots\end{aligned}$$

	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
x	0	3	0	4	0	1	3	5	0
y	0	3	0	4	4	5	7	9	9

Path of a TTS

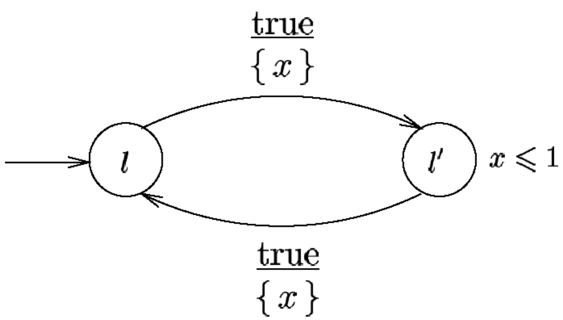
Def.: a path is called time-divergent if $\lim_{i \to \infty} \Delta(\sigma, i) = \infty$.

Non timed-divergent paths in previous automata?

Ex. of non time-div path: $s_0 \xrightarrow{2^{-1}} s_1 \xrightarrow{2^{-2}} s_2 \xrightarrow{2^{-3}} s_3 \ldots s_k \xrightarrow{2^{-k+1}} s_{k+1} \ldots$

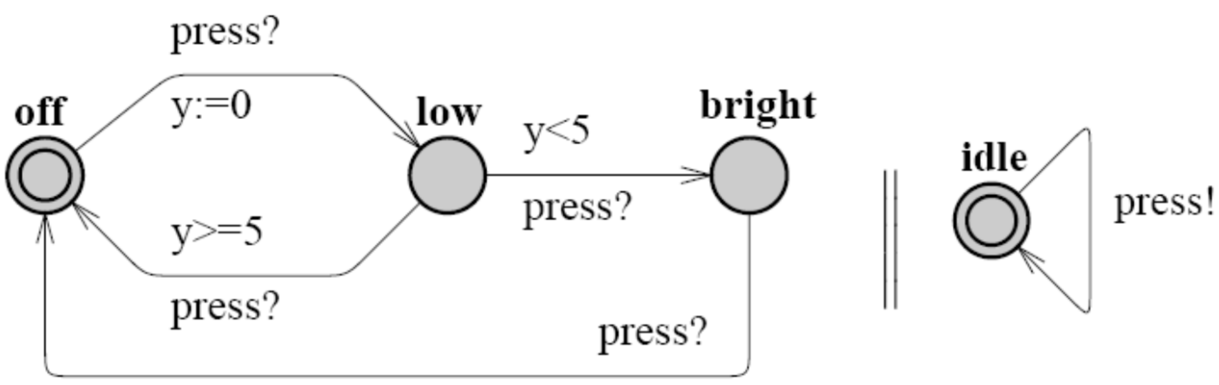
The set of time-divergent paths from a state s is $Paths^{\infty}(s)$

Def: A timed automata A is called non-Zeno if from any state some time-divergent path can start



Example of a Timed automata

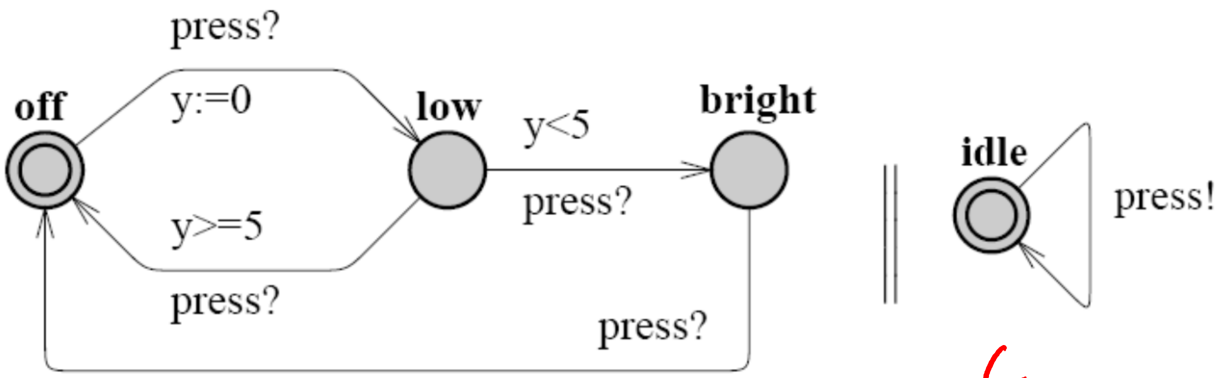
L'esempio della lampadina a due livelli



(a) Lamp. (b) User.

Example of a Timed automata

L'esempio della lampadina a due livelli



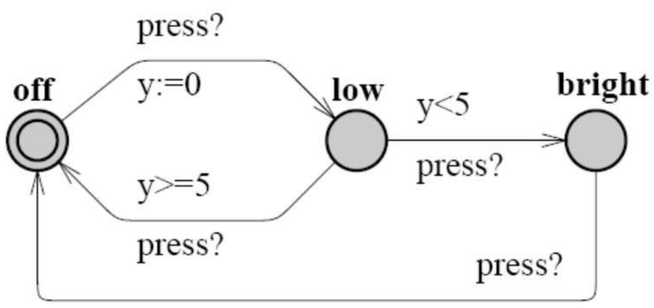
(a) Lamp. (b) User.

$(low, \nu_0) \models EF(bright \text{ and } y = 7)$ ($y \in C$)

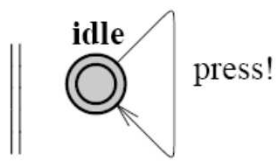
$(off, \nu_0) \models EF^{\leq 4}\, bright \equiv z \text{ in } EF(bright \wedge z \leq 4)$

FALSE $(low, \nu_t) \models EF^{>6}\, bright \equiv z \text{ in } EF(bright \wedge z > 6)$

Examples of Timed automata



(a) Lamp.



(b) User.

Timed Computational Tree Logic (TCTL)

Syntax: CTL + formula clocks that can be reset in formula

Semantics defined over TTS

Example of properties that can be expressed in TCTL

- If a message is sent,
  it is received within at most 5 time units.
  $AG\ (\ \mathit{send}(m) \rightarrow AF^{\leq 5}\ \mathit{receive}(m)\ )$
- It is possible to reach a red state
  from each blue state immediately.
  $AG\ (\ \mathit{blue} \rightarrow EF^{=0}\ \mathit{red}\ )$
- The program finishes exactly after 5 time units.
  $A\ (\ \neg \mathit{finished}\ U^{=5}\ \mathit{finished}\ )$

Timed Computational Tree Logic

Def: For $p \in AP$, $z \in D$, D the set of formula clocks, and $\alpha \in Cstr(C \cup D)$, the set of TCTL formulae is given by:

$$\phi ::= p \mid \alpha \mid \neg\phi \mid \phi \vee \phi \mid z \text{ in } \phi \mid \mathsf{E}[\phi \,\mathsf{U}\, \phi] \mid \mathsf{A}[\phi \,\mathsf{U}\, \phi].$$

Clock z in "z in Φ" is called a freeze identifier, and it means: "z in Φ" is valid in state s if Φ holds in s where clock z starts from 0

For example: "z in (z=0)" is valid (true in any state) while "z in (z>1)" is not

Clocks have to be bounded to the formula

Timed Computational Tree Logic

Not a very convenient way to express timed properties, and a number of derived operators have been defined:

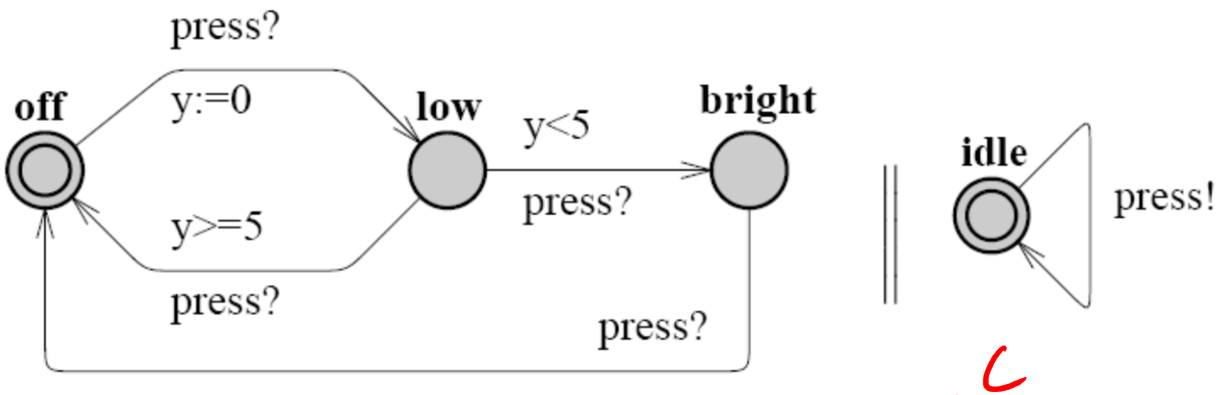
- $E(\Phi\ U^{\leq n}\ \Psi) = \text{reset } z \text{ in } E(\Phi\ U\ (z \leq n \wedge \Psi))$
- $A(\Phi\ U^{<n}\ \Psi) = \text{reset } z \text{ in } A(\Phi\ U\ (z < n \wedge \Psi))$

- $EF^{=n}\ \Phi = \text{reset } z \text{ in } EF\ (z = n \wedge \Phi)$
- $AF^{\leq n}\ \Phi = \text{reset } z \text{ in } AF\ (z \leq n \wedge \Phi)$

- $EG^{<n}\ \Phi = \neg AF^{<n}\ \neg\Phi$
- $AG^{=n}\ \Phi = \neg EF^{=n}\ \neg\Phi$

Example of a Timed automata

L'esempio della lampadina a due livelli



(a) Lamp. (b) User.

$(low, v_0) \models EF(bright \text{ and } y = 7)$ ($y \in C$)

$(off, v_0) \models EF^{\leq 4}\, bright \equiv z \text{ in } EF(bright \wedge z \leq 4)$

FALSE $(low, v_t) \models EF^{>6}\, bright \equiv z \text{ in } EF(bright \wedge z > 6)$

Timed Computational Tree Logic

Semantics: need to define (i,d), position over a path and an order relationship on position

This definition is wrong in Katoen's notes (as was in the original paper of Alur and Dill of 1989/90)

Timed Computational Tree Logic

Def.: A RT-trajectory σ is an infinite sequence of states $s_i = (l_i, v_i)$ and delays δ_ι:

$$\sigma = s_0 \; -\delta_0\!-> \; s_1 \; -\delta_1\!-> \; s_2 \; -\delta_2\!-> \; s_3 \; -\delta_3\!-> \ldots\ldots\ldots\ldots$$

Def.: A position in σ is the pair (i,δ): $i \in \mathcal{N}$ *and* $\delta \le \delta_\iota$

Def.: location in the position (i,δ) is $loc(i,\delta) = l_i$

Def.: valuation in the position (i,δ) is $val(i,\delta) = v_i + \delta$

Def.: state in position (i,δ) is

$$\sigma(i,\delta) = (\, loc(i,\delta), \, val(i,\delta) \,)$$

Timed Computational Tree Logic

Def.: time elapsed at position (i,δ) is

$$\tau_\sigma(i,\delta) = \delta + \Sigma_{0 \le j < i}\, \delta_i$$

Def. of precedence on positions: we say that (i,δ) precedes (j,δ') and we write $(i,\delta) << (j,\delta')$ if:

- $i < j$ *or*
- $i = j$ and $\delta \le \delta'$

Timed Computational Tree Logic

Def: Semantics of TCTL. Let $p \in AP$, $z \in D$, $w \in V(D)$, $s \in S$ (States of the TTS), $\alpha \in Cstr(C \cup D)$, $s=(l,v)$, $v \in V(C)$, the set of TCTL formulae is given by:

$$s, w \models p \quad \text{iff } p \in Label(s) = Label(l)$$

$$s, w \models \alpha \quad \text{iff } v \cup w \models \alpha$$

$$s, w \models \neg \phi \quad \text{iff } \neg (s, w \models \phi)$$

$$s, w \models \phi \vee \psi \quad \text{iff } (s, w \models \phi) \vee (s, w \models \psi)$$

$$s, w \models z \textbf{ in } \phi \quad \text{iff } s, \textbf{reset } z \textbf{ in } w \models \phi$$

$s = (l, v)$, $v \in V(C)$; $w \in V(D)$

Timed Computational Tree Logic

.........cont.: let p∈AP, z∈D, w∈V(D), s∈S, α∈Cstr(C ∪D), $P^{\infty}_{\mathcal{M}}(s)$ the RT-trajectories starting in s,

$$s, w \models \mathbf{E}\,[\phi\,\mathsf{U}\,\psi] \quad \text{iff } \exists\, \sigma \in P^{\infty}_{\mathcal{M}}(s).\, \exists\,(i,d) \in Pos(\sigma).$$
$$(\sigma(i,d), w{+}\Delta(\sigma,i) \models \psi \;\wedge$$
$$(\forall\,(j,d') \ll (i,d).\, \sigma(j,d'), w{+}\Delta(\sigma,j) \models \phi \;\vee\; \psi))$$
$$s, w \models \mathbf{A}\,[\phi\,\mathsf{U}\,\psi] \quad \text{iff } \forall\, \sigma \in P^{\infty}_{\mathcal{M}}(s).\, \exists\,(i,d) \in Pos(\sigma).$$
$$((\sigma(i,d), w{+}\Delta(\sigma,i)) \models \psi \;\wedge$$
$$(\forall\,(j,d') \ll (i,d).\, (\sigma(j,d'), w{+}\Delta(\sigma,j)) \models \phi \;\vee\; \psi))$$

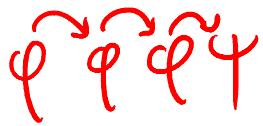
Timed Computational Tree Logic

Why it is necessary that … $w+\Delta(\sigma, j)) \models \phi \vee \psi)$

Consider the formula

~~reset~~ z in E(z<=5 U z>5)

then on paths on which the delays on the paths are almost zero we approach 5: it is not possible to find “the point” in which z become >5 for the first time



Example of TCTL

Promptness requirement: maximal delay between an event and its reaction

$$\mathsf{AG}\left[send(m) \;\Rightarrow\; \mathsf{AF}_{<5}\, receive\,(r_m)\right]$$

z in AF(receive(r_m) and z<5)

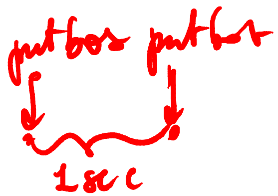
Punctuality requirement: exact delay between events

$$\mathsf{EG}\left[send(m) \;\Rightarrow\; \mathsf{AF}_{=11}\, receive\,(r_m)\right]$$

Example of TCTL

Periodicity requirement: an event occur within a certain period
Example: a machine that put boxes on a belt every 25 time units

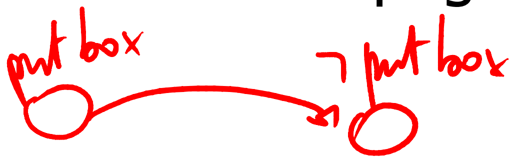
$$\mathsf{AG}\,[\mathsf{AF}_{=25}\; putbox]$$



$$\mathsf{AG}\,[putbox \;\Rightarrow\; \neg\, putbox\,\mathsf{U}_{=25}\; putbox]$$

Attention: the correct version of the above formula is
AG (putbox ➔ z in [(not(putbox) or z=0) U (putbox and z =25)])

Same correction for the formulas in the next pages



Example of TCTL

Minimal delay: minimal delay between events

Example: the delay between two trains at a crossing (tac) should be at least 180

$$\mathsf{AG}\,[tac \;\Rightarrow\; \neg\, tac\,\mathsf{U}_{\geqslant 180}\, tac]$$

Interval delay: an event must occur within a certain interval from another event

Example: trains should have a maximal distance of 900 time units (the minimal delay still holds)

$$\mathsf{AG}\,[tac \;\Rightarrow\; (\neg\, tac\,\mathsf{U}_{\geqslant 180}\, tac \;\wedge\; \neg\, tac\,\mathsf{U}_{\leqslant 900}\, tac)]$$

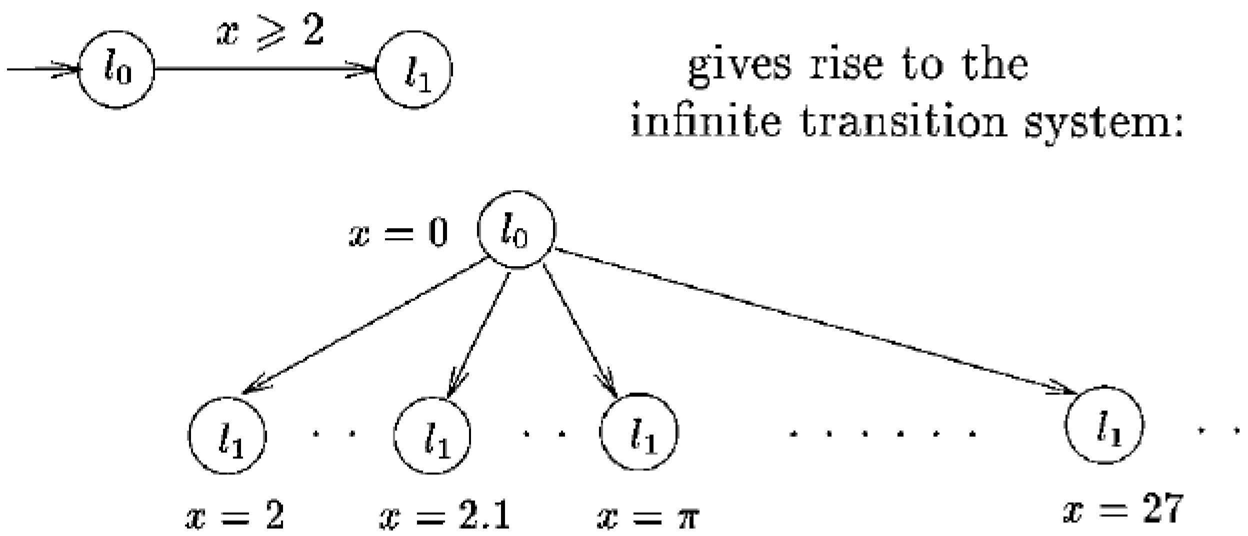
$$\mathsf{AG}\,[tac \;\Rightarrow\; \neg\, tac\,\mathsf{U}_{=180}\,(\mathsf{AF}_{\leqslant 720}\, tac)]$$

Clock equivalence

Even simple automata give rise to infinite TTS, the infinite number of states is due to the real valuations of clocks

Solution: a finite number of equivalence classes on the clock valuations. Equivalence should maintain…..

Question: what could be such equivalence on the TA below?



Clock equivalence

Solution: a finite number of equivalence classes on the clock valuations.

Define an equivalence ≈ that should have the following characteristics:

- correctness: (v,w) ≈ (v′,w′) ==> $\forall\Phi$: (v,w)|=Φ sse (v′,w′)|=Φ
- finiteness: the number of equivalence classes of ≈ is finite

Approach: we present the definition and we explain why each constraint is needed

Clock equivalence

Approach: we present the definition and we explain why each constraint is needed. Lex c_x be the maximal constant that appears in a constraint on x

Definition 49. (**Clock equivalence**)
Let $\mathcal{A}$ be a timed automaton with set of clocks C and $v, v' \in V(C)$. Then $v \approx v'$ if and only if

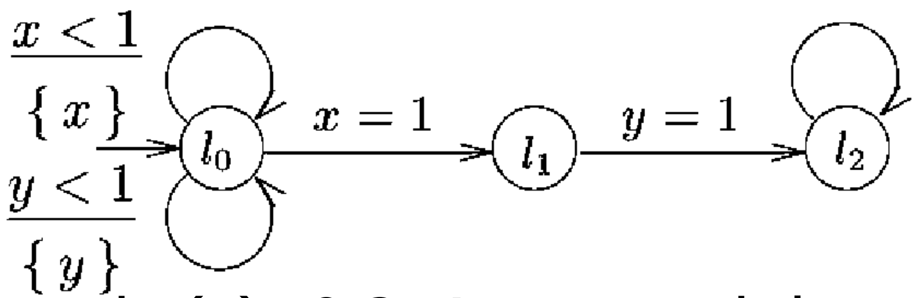
1. $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$ or $v(x) > c_x$ and $v'(x) > c_x$, for all $x \in C$, and
2. $\mathit{frac}(v(x)) \leqslant \mathit{frac}(v(y))$ iff $\mathit{frac}(v'(x)) \leqslant \mathit{frac}(v'(y))$ for all $x, y \in C$ with $v(x) \leqslant c_x$ and $v(y) \leqslant c_y$, and
3. $\mathit{frac}(v(x)) = 0$ iff $\mathit{frac}(v'(x)) = 0$ for all $x \in C$ with $v(x) \leqslant c_x$.

Clock equivalence

1st observation: may be we can use only the integral part

$$v \approx v' \text{ if and only if } \lfloor v(x) \rfloor = \lfloor v'(x) \rfloor \text{ for all } x \in C.$$

2nd observation: the integral part is not enough, also the relative order of clocks should be taken into account



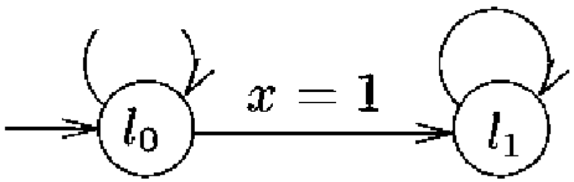
When v(x)=0.4 and v(y)=0.3, A can reach l_2

when v(x)=0.2 and v(y)=0.3, A cannot reach l_2

$$v(x) \leqslant v(y) \text{ if and only if } v'(x) \leqslant v'(y) \text{ for all } x, y \in C$$

Clock equivalence

3rd observation: since in the constraint the comparison is with natural numbers, it can make a difference whether v(x)=n or v(x)=n.m



When v(x)=1.1 and v'(x)=1, the clocks have the same integral part but only from v' we can take the transition to l_1

$$frac(v(x)) = 0 \text{ if and only if } frac(v'(x)) = 0 \text{ for all } x \in C.$$

4th observation: all valuation are of interest only when they do not pass c_x be the maximal constant that appears in a constraint on x

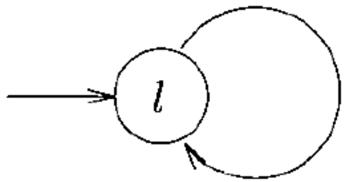
Clock equivalence

This lead to the following definition (Alur-Dill 1994):

Definition 49. (**Clock equivalence**)
Let $\mathcal{A}$ be a timed automaton with set of clocks C and $v, v' \in V(C)$. Then $v \approx v'$ if and only if

1. $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$ or $v(x) > c_x$ and $v'(x) > c_x$, for all $x \in C$, and
2. $\mathit{frac}(v(x)) \leqslant \mathit{frac}(v(y))$ iff $\mathit{frac}(v'(x)) \leqslant \mathit{frac}(v'(y))$ for all $x, y \in C$ with $v(x) \leqslant c_x$ and $v(y) \leqslant c_y$, and
3. $\mathit{frac}(v(x)) = 0$ iff $\mathit{frac}(v'(x)) = 0$ for all $x \in C$ with $v(x) \leqslant c_x$.

Equivalence - example



$$\frac{x \geqslant 2}{\{x\}}$$

The first requirement leads to the following eq. classes

$$[0 \leqslant x < 1], [1 \leqslant x < 2], [2 \leqslant x < 3], [3 \leqslant x < 4], \ldots$$

Since the biggest constant with which x is compared is 2,

$$[0 \leqslant x < 1], [1 \leqslant x < 2], [x = 2], \ \ and \ [x > 2]$$

Separating according to the fractional part

$$[x = 0], [0 < x < 1], [x = 1], [1 < x < 2], [x = 2], \ \ and \ [x > 2]$$

Clock ordering irrelevant (only one clock)

Equivalence - example

Def.: the equivalence classes according to the previous definition can be constructed using a partition refinement algorithm (there is an example of application on page 220 of the book, that leads to the following construction)

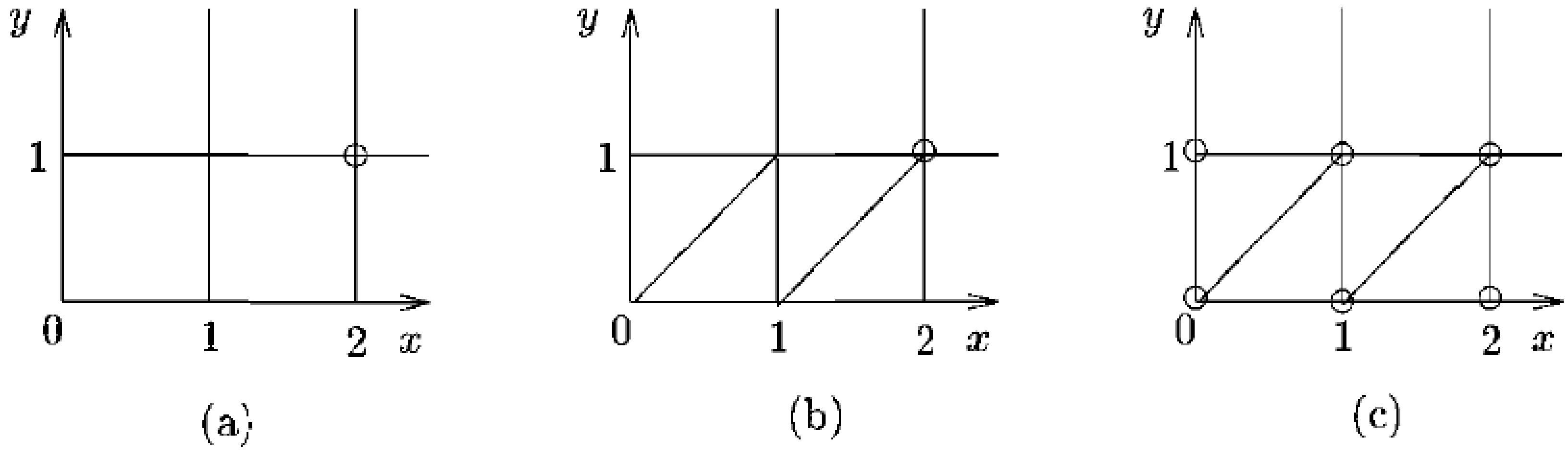


Figure 4.7: Partitioning of $\mathbb{R}^+ \times \mathbb{R}^+$ according to $\approx$ for $c_x = 2$ and $c_y = 1$

Equivalence and TCTL

The theorem below (Alur-Dill-Courcoubetis) states that regions can be safely used for TCTL model checking

Theorem 51.

Let $s, s' \in S$ such that $s, w \approx s', w'$. For any TCTL*-formula ϕ, we have:*

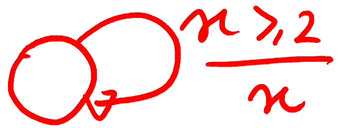
$$\mathcal{M}(\mathcal{A}), (s, w) \models \phi \textit{ if and only if } \mathcal{M}(\mathcal{A}), (s', w') \models \phi.$$

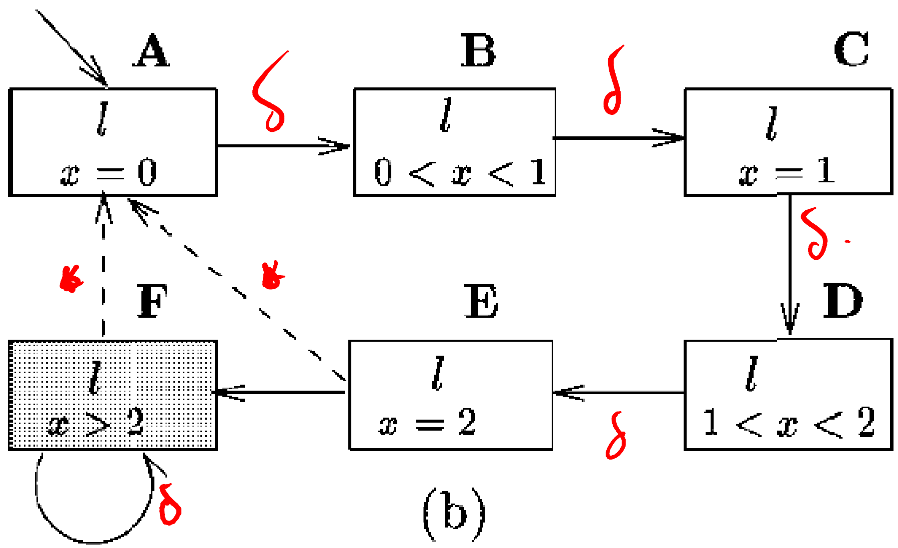
Region automata

Def.: a region is a pair (l,[v]), where l is a location and [v] an equivalence class over clock valuations

We can build a finite state automata over region, called region automata.

In region automata there are two types of transitions: let time elapse or take a transition in the TA





(b)

region automata for the single location automata used before

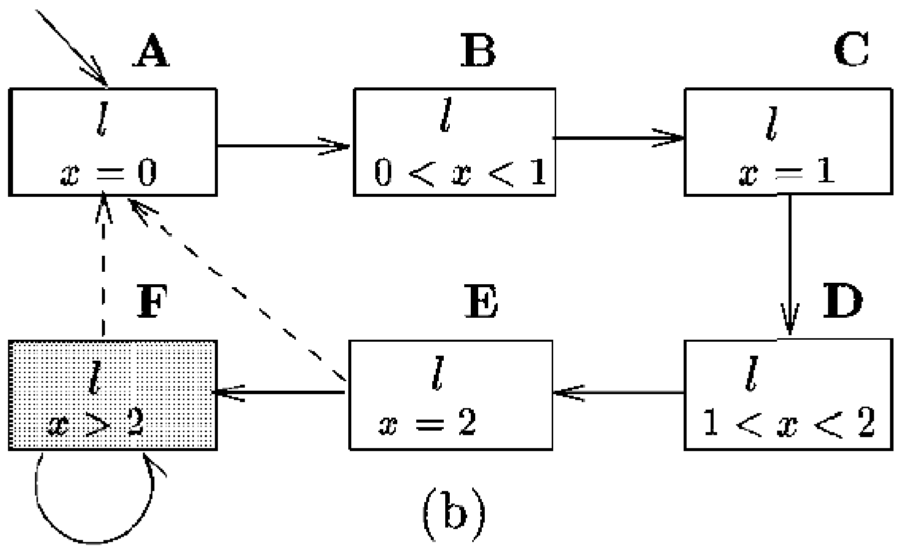
δ : let time elapse

Region automata

Def.: a region is a pair (l,[v]), where l is a location and [v] an equivalence class over clock valuations

We can build a finite state automata over region, called region automata.

In region automata there are two types of transitions: let time elapse or take a transition in the TA



(b)

region automata for the single location automata used before

Region automata

What happens when there are also formula clocks? We have to include also formula clocks in the computation of the equivalences