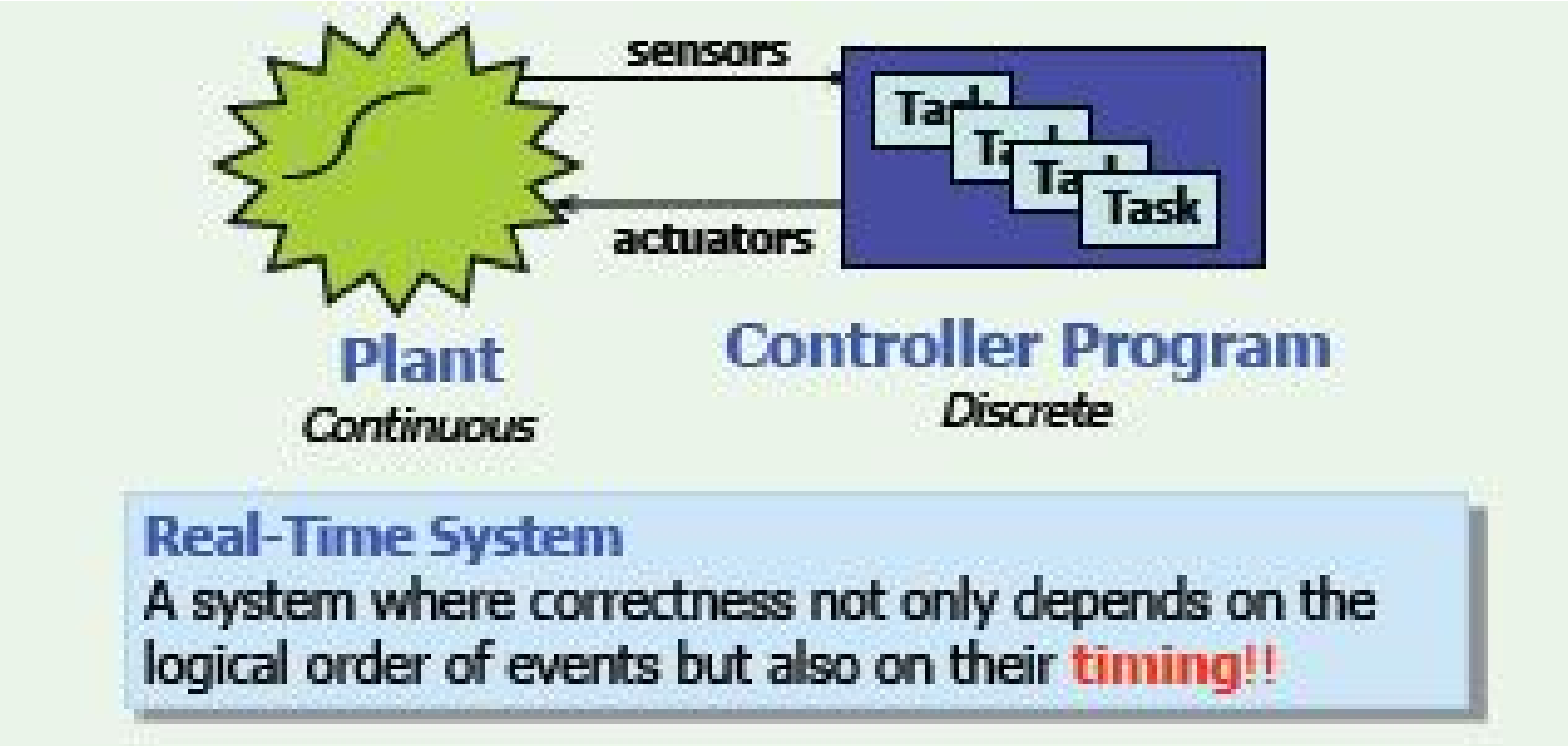
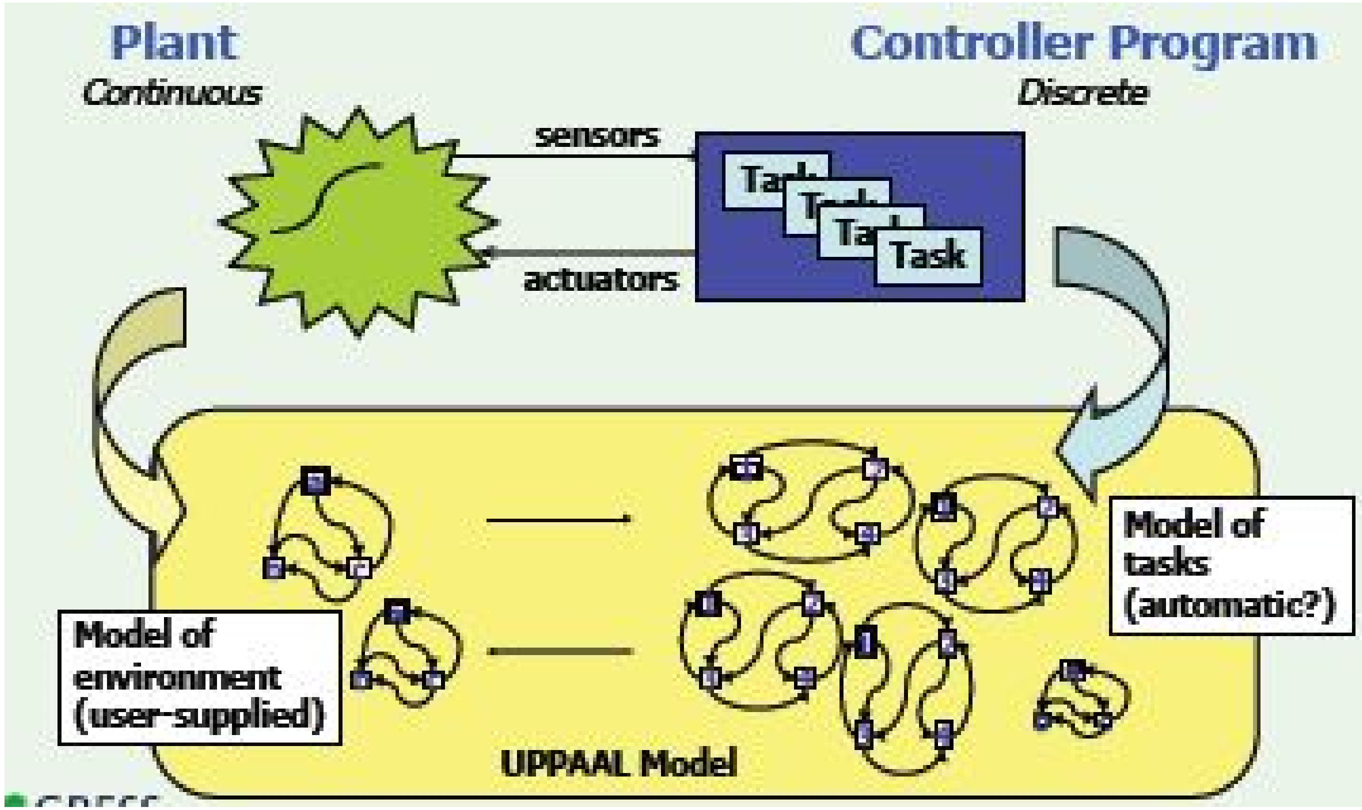
Real-Time Systems

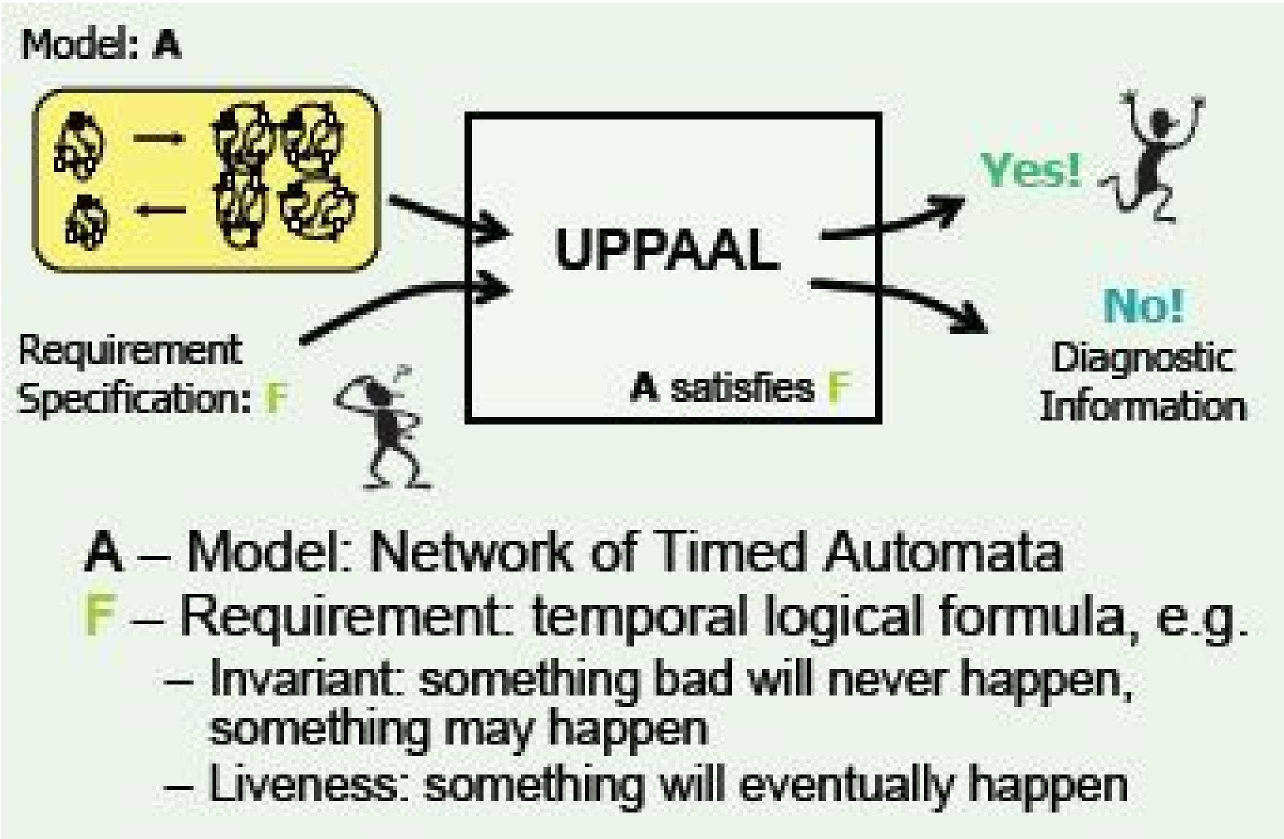


E.g.: Automotive systems – Cruise Control, ABS, Air bags,
Robots, Real-time protocols, DVD/CD Players, Process Control, etc.

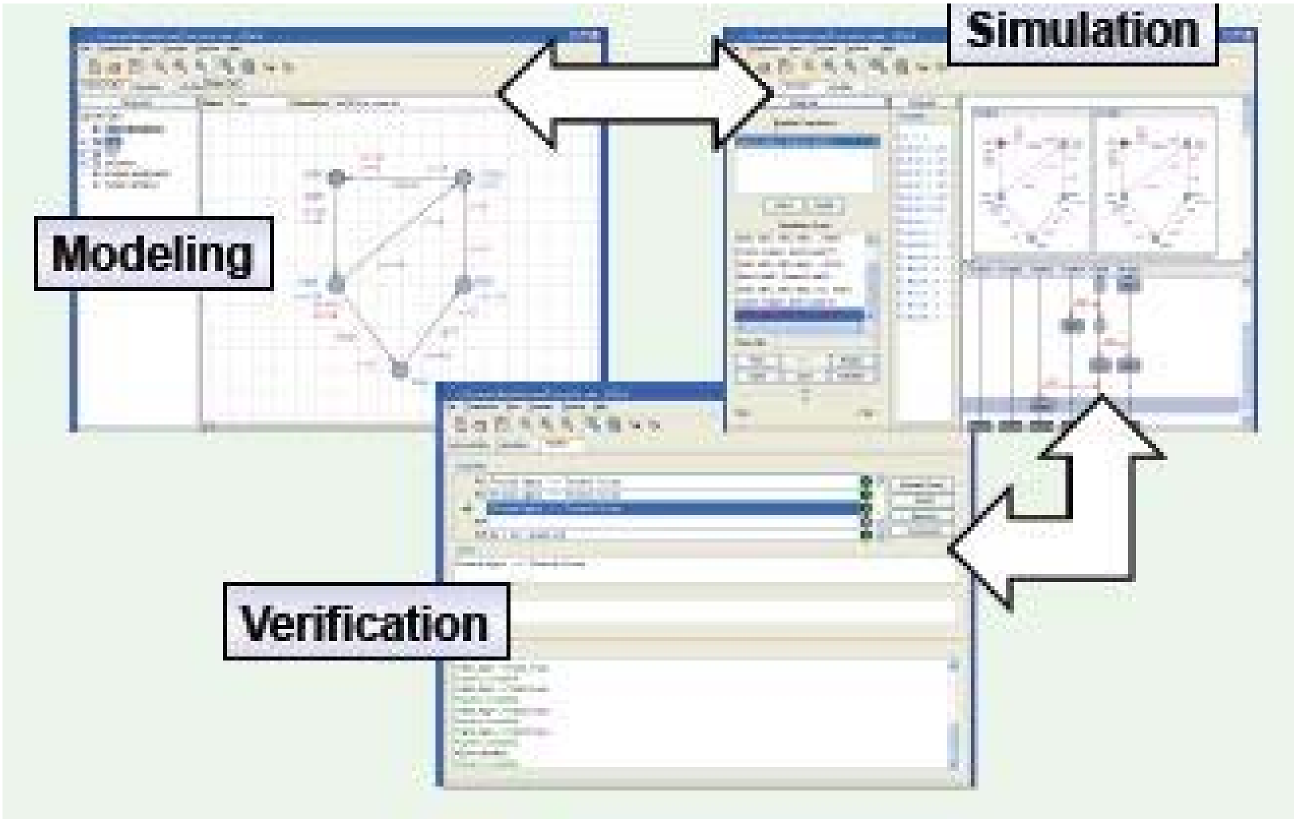
Real-Time Model Checking (RTMC)



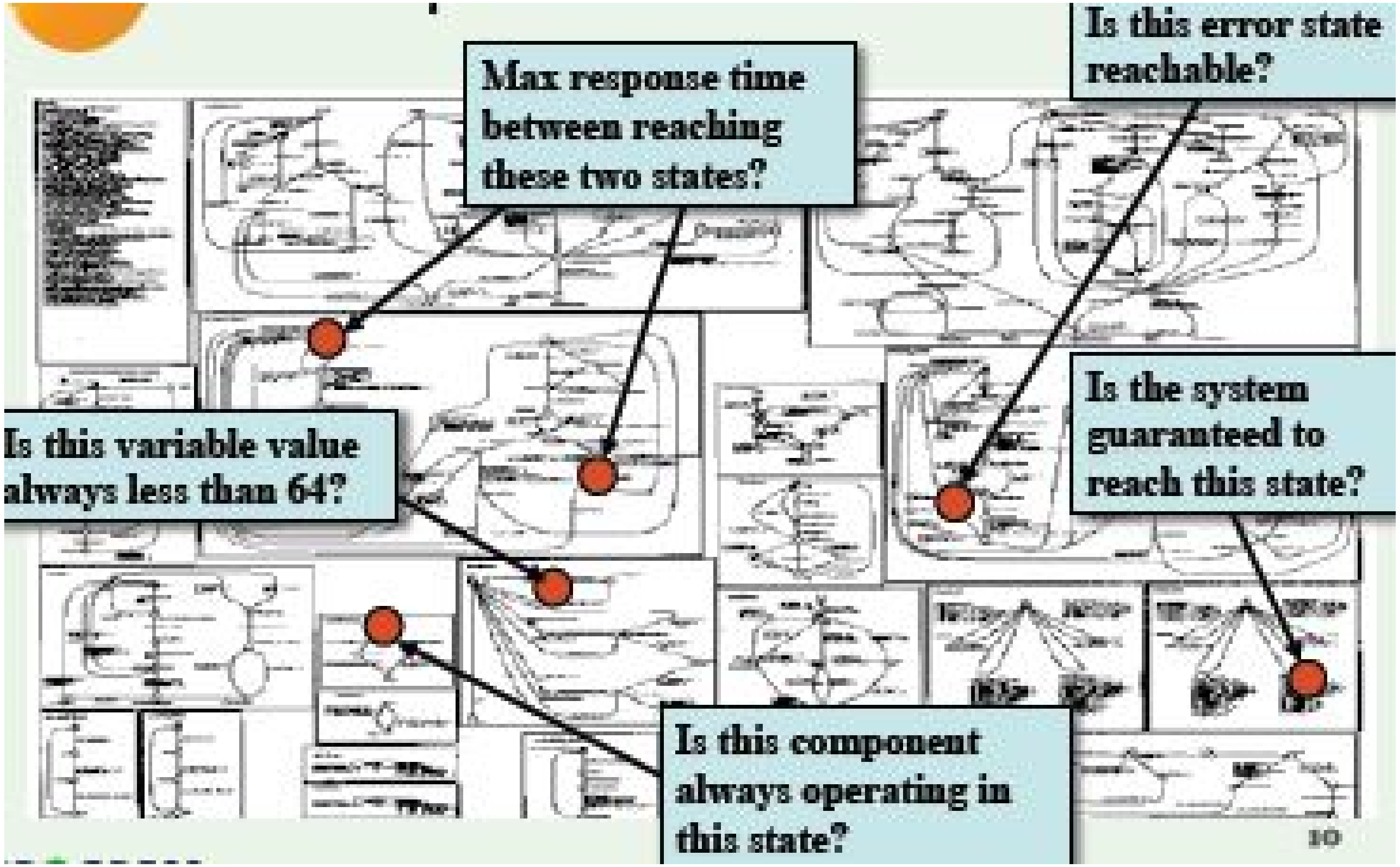
Model Checking



Formal Design and Analysis



Example model based verification



Finite State Automata: Light Control

Finite State Automata:

Finite State Graph with

Finite State Automata (FSA): Light Control

Finite State Automata:

Finite State Graph with

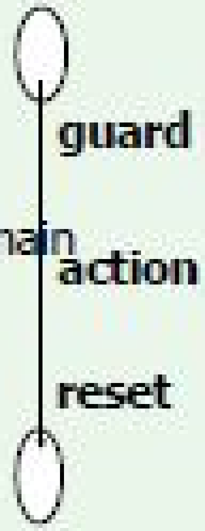
1. Set of nodes (states)

2. Set of edges (transitions)

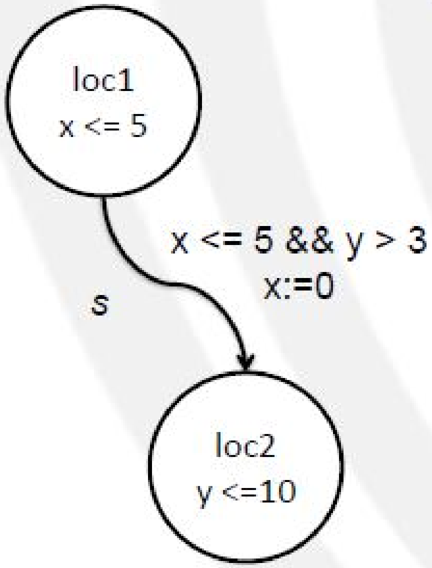
3. Set of labels (actions)

FSA with variables

- Extend FSA with variables e.g.
  - Relational automata and/or guarded commands
    - Guards and assignments on transitions
    - Maybe infinite state, but finite state for bounded domain
  - Time automata is another exmaple
    - Guards and reset over clock variables on transitions
    - Infinite state!
- Semantics: Transition Systems

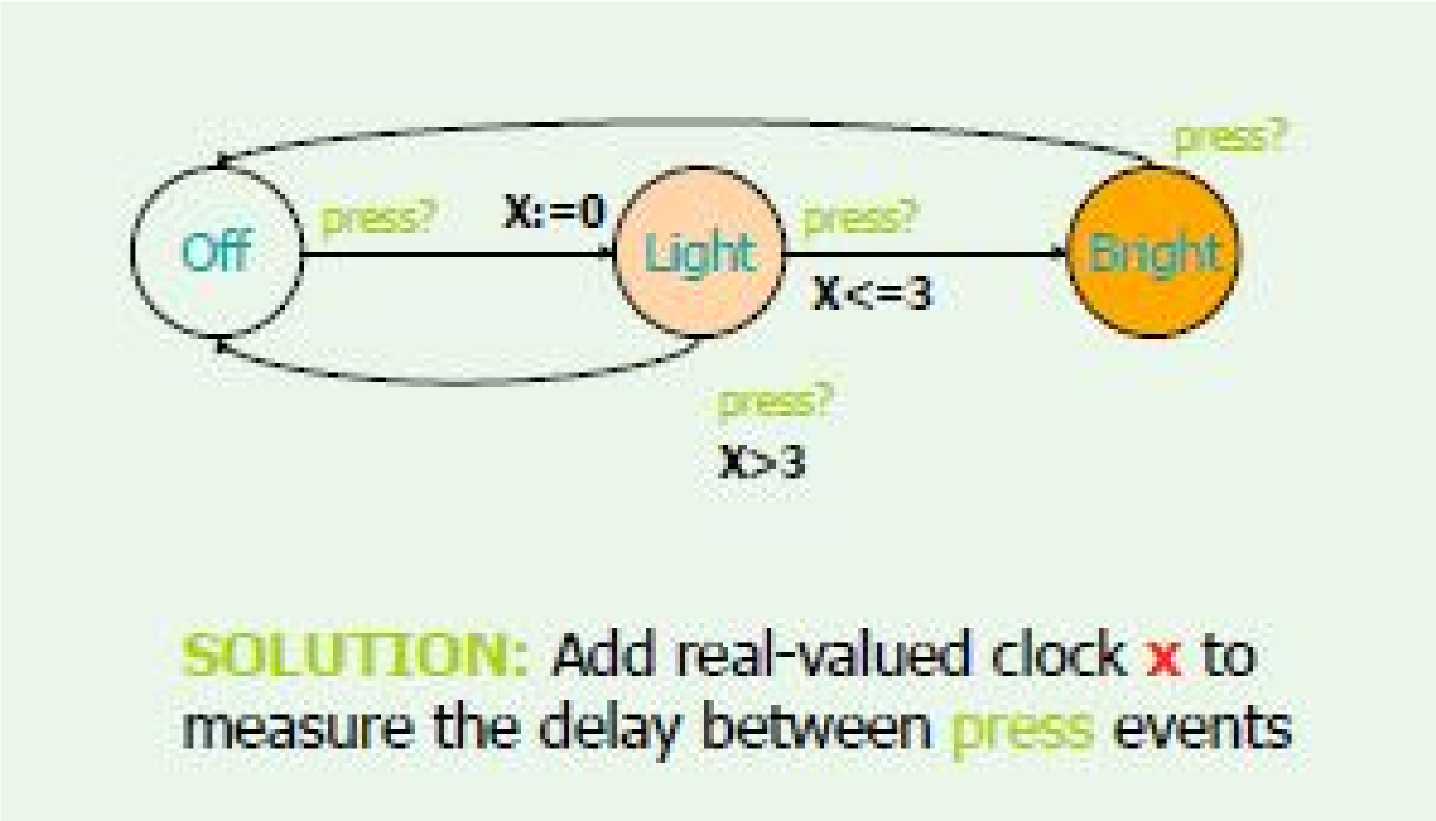


Timed Automata: Alur & Dill 1990

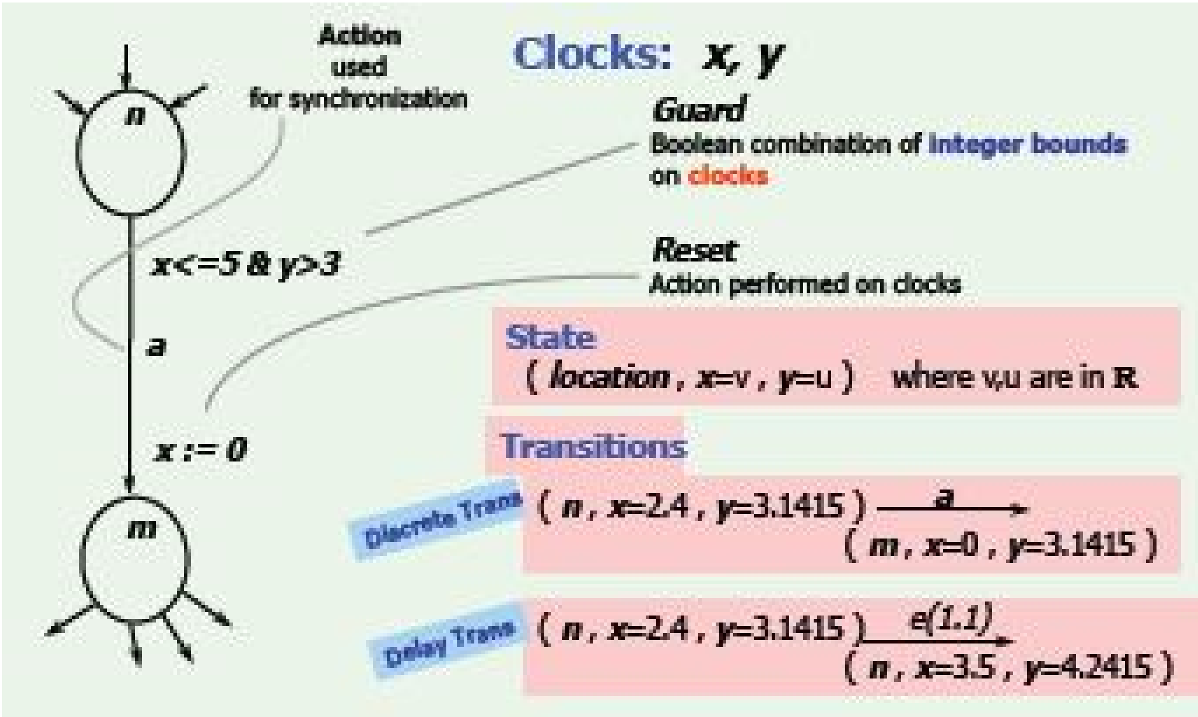


- A timed automaton is a finite state machine extended with clock variables
  - Locations
    - Invariants
  - Edges
    - Guards
    - Resets
    - Synchronization labels
  - Clocks
    - Variables evaluate real numbers

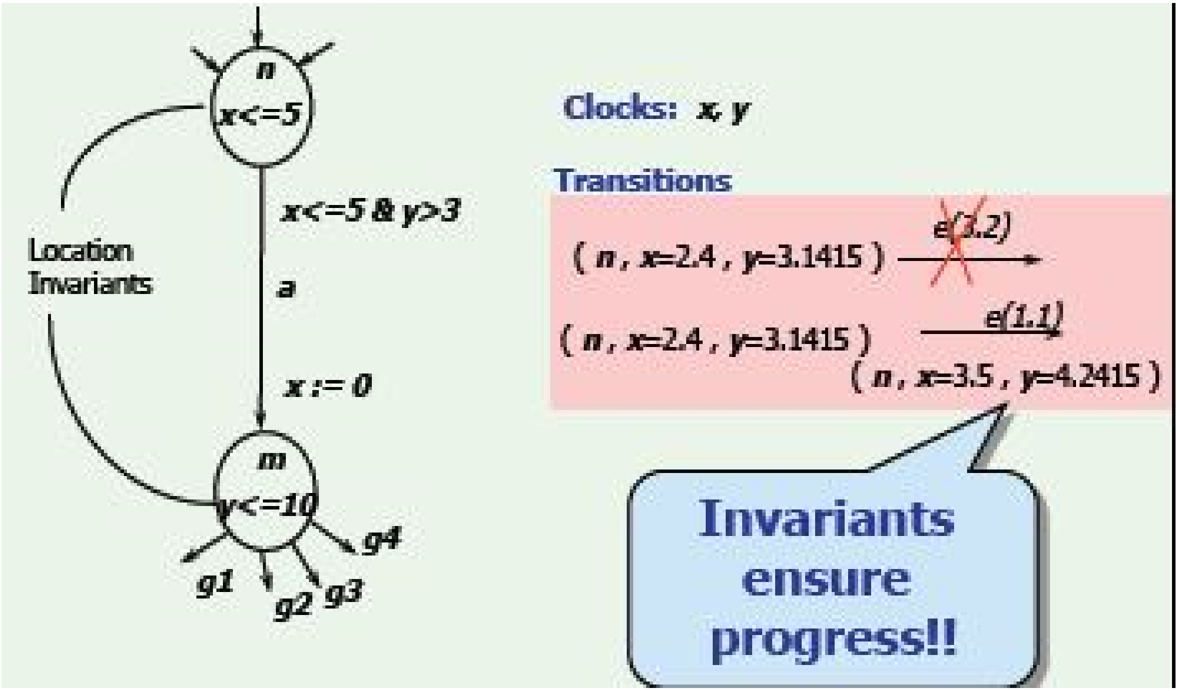
Timed Automata: Light Control with Timing



TA: Semantics



TA with invariants



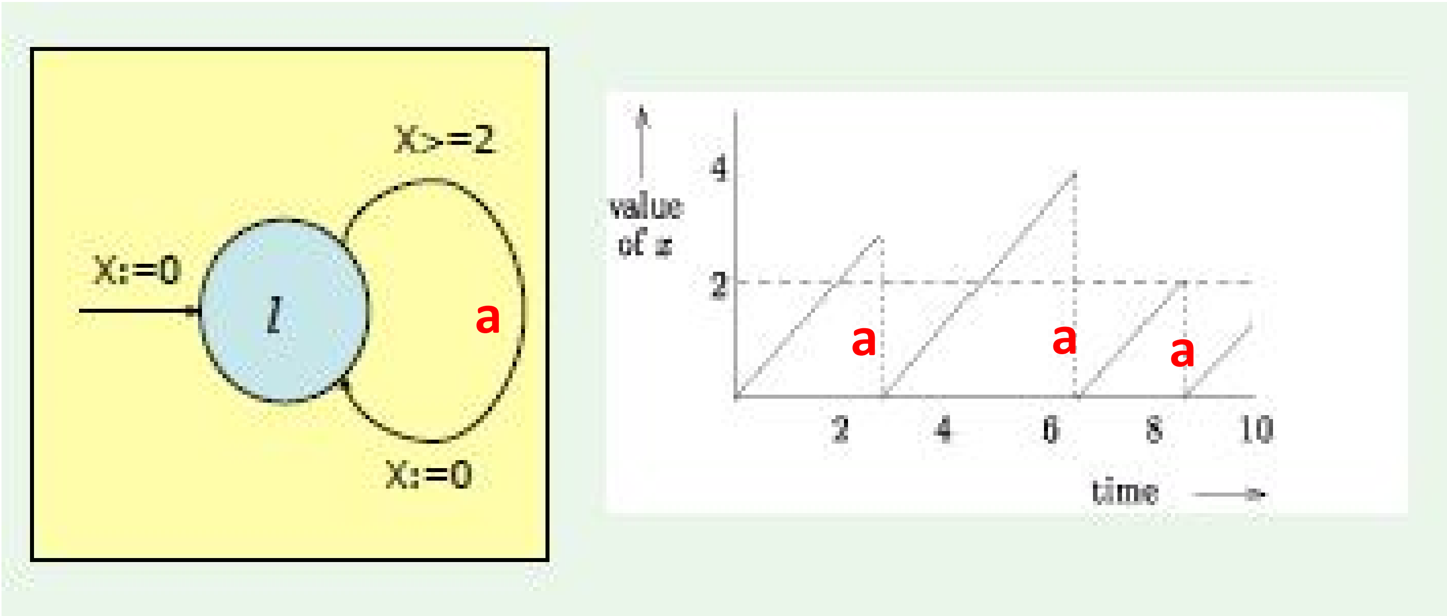
Clock constraints

For set C of clocks with $x, y \in C$, the set of *clock constraints* over C, $\Psi(C)$, is defined by

$$\alpha ::= x \prec c \;\Big|\; x - y \prec c \;\Big|\; \neg\alpha \;\Big|\; (\alpha \,\wedge\, \alpha)$$

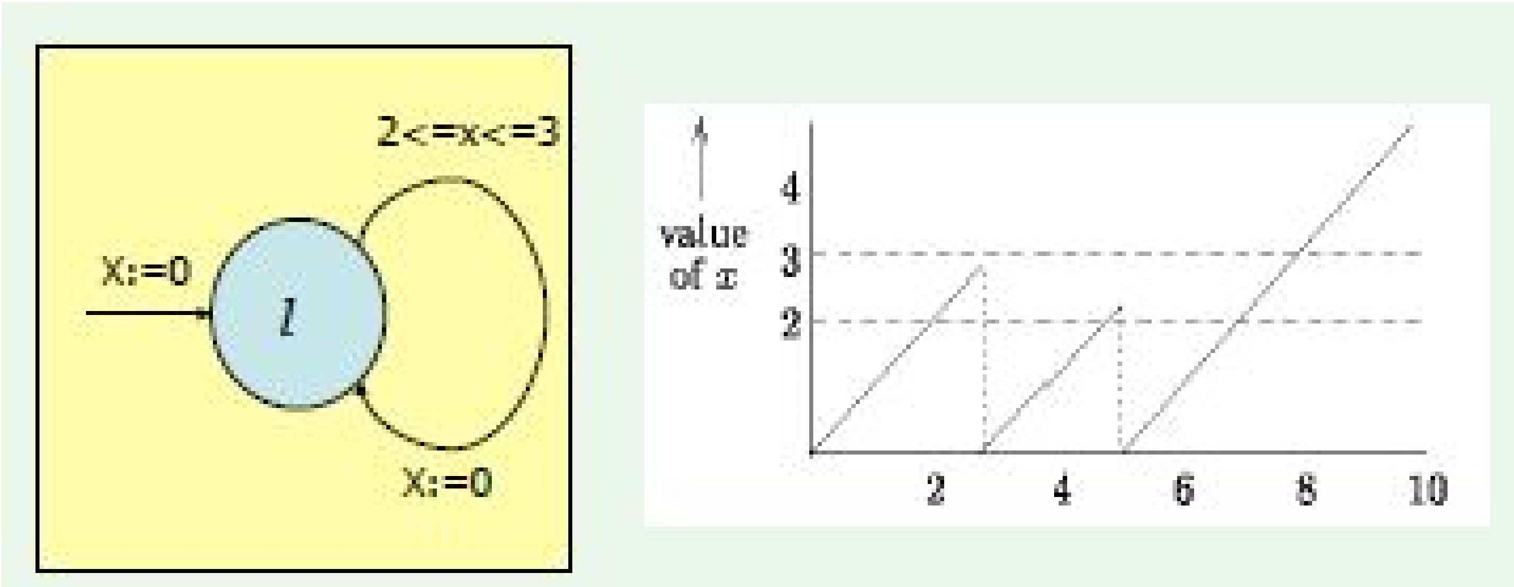
where $c \in \mathbb{N}$ and $\prec \in \{<, \leqslant\}$.

TA examples (1/4)



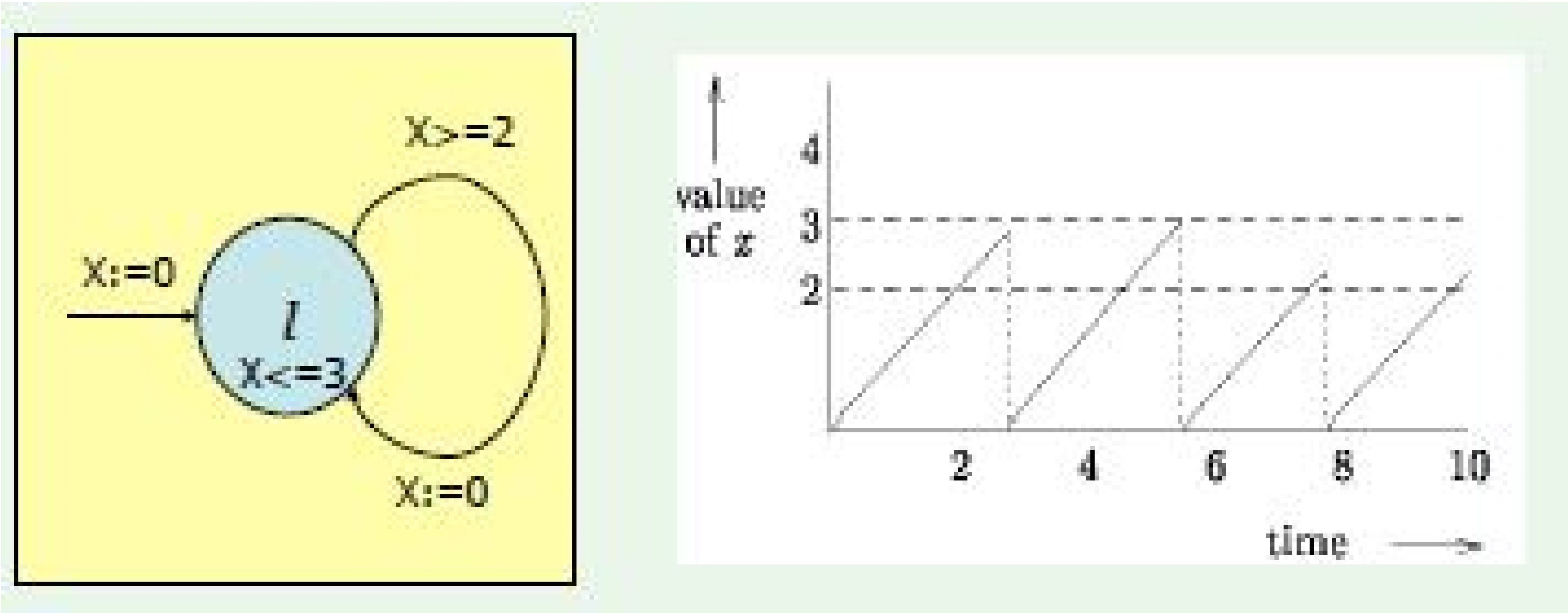
- The transition can be taken after 2 seconds

TA examples (2/4)



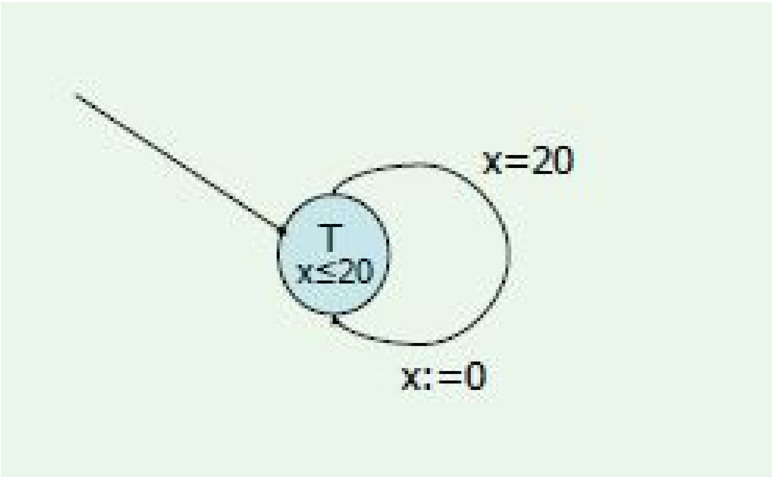
- The transition can be taken between 2 and 3 seconds
- When x > 3 let time pass, no transition can be taken

TA examples (3/4)



- The transition can be taken after 2 seconds
- The transition must be taken within 3 second

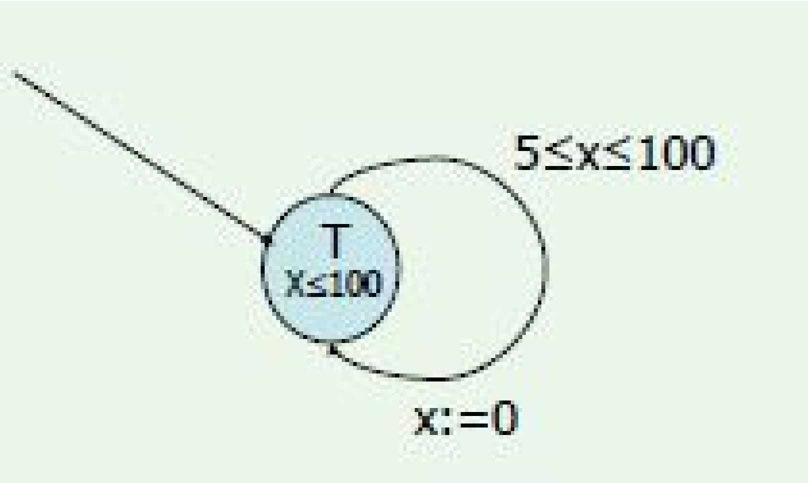
TA examples (4/4)-(a): Periodic Task



- Has regular arrival times and hard deadlines
- Executes its invocation within regular time interval

Periodic task: period 20

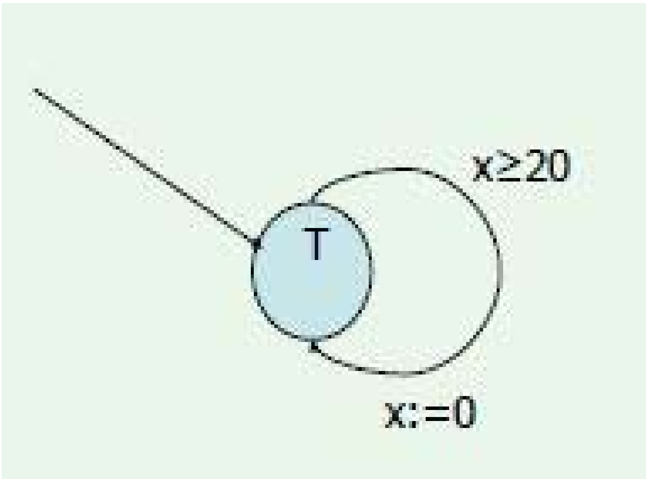
TA examples (4/4)-(b): Aperiodic Task



- Invoked only once and arises at random invocations.
- Has irregular arrival times.
- Its arrival times are unknown at design time.

aperiodic task, every 5 to 100

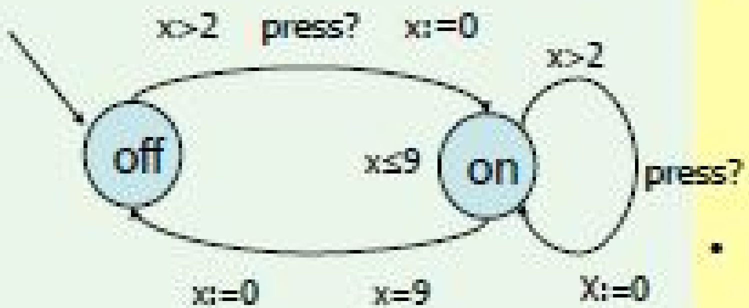
TA examples (4/4)-(c): Sporadic Task



sporadic task min period 20

- Arrives to the system at random points in time, but with defined minimum inter-arrival times (= the minimum separation = period 20) between two consecutive invocations.
- We don't know when they arrive to the system (is the relative deadline).
- The minimum separation btw two consecutive invocations of the task implies that once an invocation of a sporadic task occurs, the next invocation cannot occur before 20 time units have elapsed.

TA: Light Switch



- Switch may be turned on whenever at least 2 time units has elapsed since last "turn off"
- Light automatically switches off after 9 time units if it is not pressed.

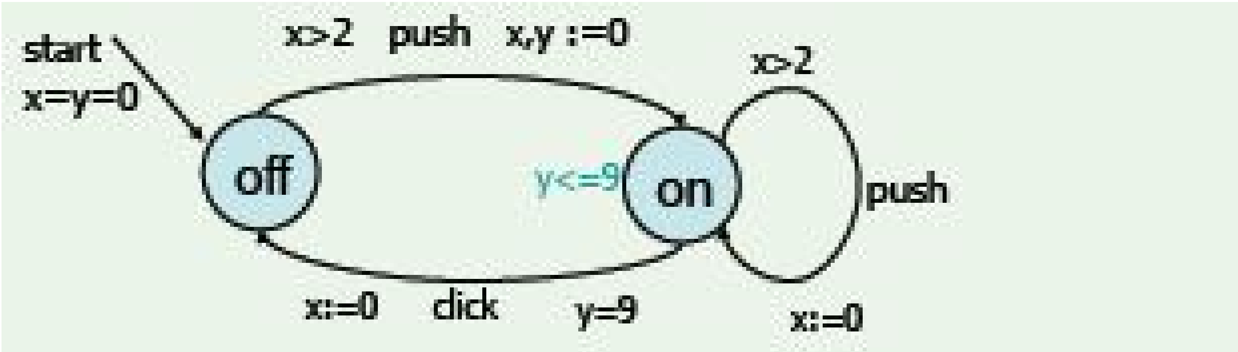
Semantics Definition

- <u>Clock valuations</u>: $V(C)$ $\quad v : C \to R_{\geq 0}$
- <u>State</u>: (l, v) *where* $l \in L$ *and* $v \in V(C)$
- <u>Action transition</u> $(l, v) \xrightarrow{a} (l', v')$ *iff* $l \xrightarrow{g\ a\ r} l$

  $g(v)$ *and* $v' = v[r]$ *and* $Inv(l')(v')$
- <u>Delay transition</u> $(l, v) \xrightarrow{d} (l, v + d)$ *iff*

  $Inv(l)(v + d')$ *whenever* $d' \leq d \in R_{\geq 0}$

TA: Example



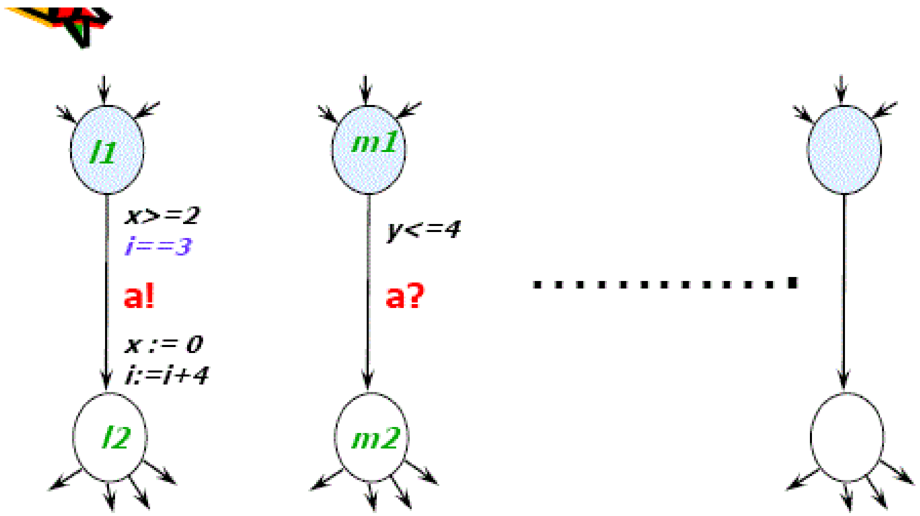
$$(off, x = y = 0) \xrightarrow{3.5} (off, x = y = 3.5) \xrightarrow{push}$$

$$(on, x = y = 0) \xrightarrow{\pi} (on, x = y = \pi) \xrightarrow{push}$$

$$(on, x = 0, y = \pi) \xrightarrow{3} (on, x = 3, y = \pi + 3) \xrightarrow{9-(\pi+3)}$$

$$(on, x = 9 - (\pi + 3), y = 9) \xrightarrow{click} (off, x = 0, y = 9) \ldots$$

Networks of TA with (finite) integer variables



with (finite) integer variables

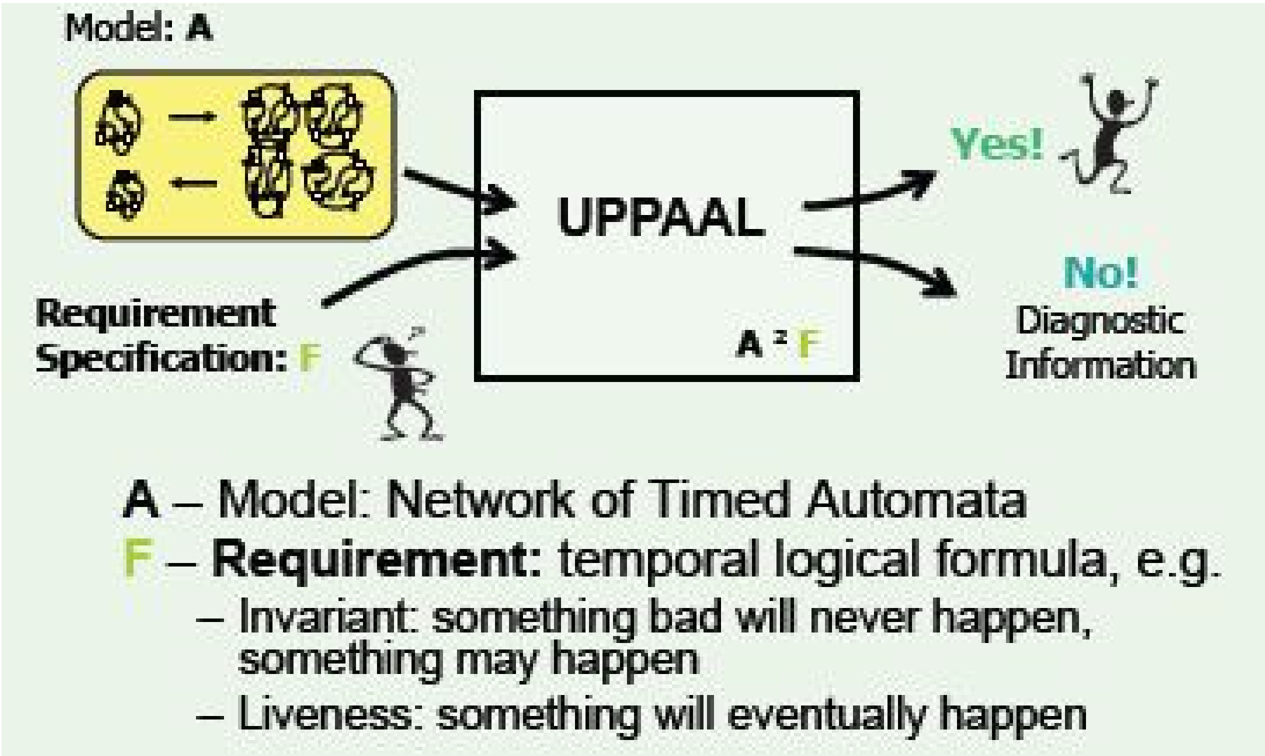
Two-way synchronization on *complementary* actions.

Closed Systems!

Example transitions

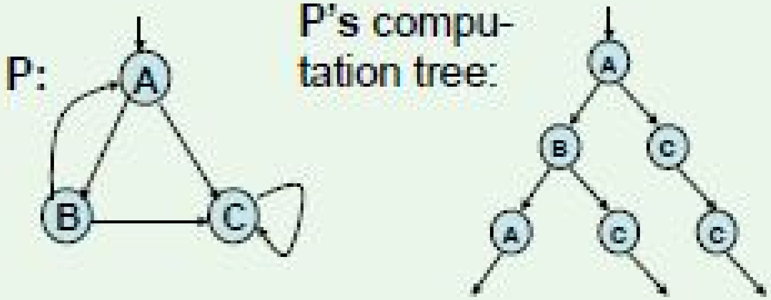
(*l1*, *m1*,........, $x=2$, $y=3.5$, $i=3$,.....) $\xrightarrow{\text{tau}}$ (*l2*,*m2*,........,$x=0$, $y=3.5$, $i=7$,.....)

How to specify what to check?
Specification of requirements



Specification of Requirements: Quantifiers in TCTL

- TCTL - Timed Computation Tree Logic



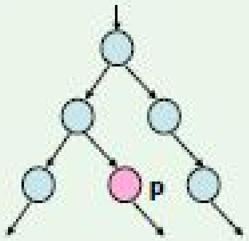
- A → C→ C → C → … a path
- (A,v) → (C,v') → …+ time = a timed path

- E - exists a path (∃).
- A - for all paths (∀).
- [ ] - all states in a path (☐ or G).
- <> - some state in a path (◊ or F).

- We shall look at the following combinations:
  - `A[]`, `A<>`, `E<>`, and `E[]`.

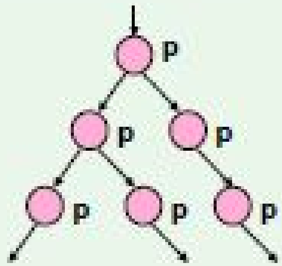
1. EF p : “p reachable” :: 2. AG p : “invariantly p”
3. AF “inevitable p” :: 4. EG p : “potentially always p”

- It is possible to reach a state in which p is satisfied.



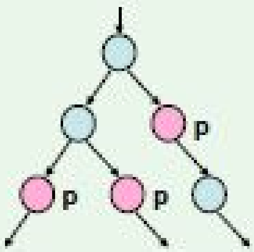
- p is true in (at least) one reachable state.

- p holds invariantly.



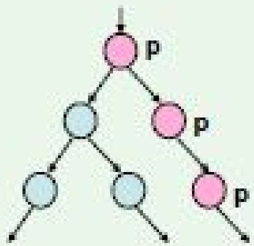
- p is true in all reachable states.

- p will inevitable become true
  - the automaton is guaranteed to eventually reach a state in which p is true.



- p is true in some state of all paths.

- p is potentially always true.



- There exists a path in which p is true in all states.

Recall: Queries Examples

- Nothing bad can happen:

  AG p

- Infinitely often p (i.e., it is repeatedly satisfied)

Recall: Queries Examples

- Nothing bad can happen:

  AG p

- Infinitely often p (i.e., it is repeatedly satisfied):

  AGAF p

- Always p is possible

Recall: Queries Examples

- Nothing bad can happen:

  AG p

- Infinitely often p (i.e., it is repeatedly satisfied):

  AGAF p

- Always p is possible

  AGEF p

- There exists a state from which p always holds

Recall: Queries Examples

- Nothing bad can happen:

  AG p

- Infinitely often p (i.e., it is repeatedly satisfied):

  AGAF p

- Always p is possible:

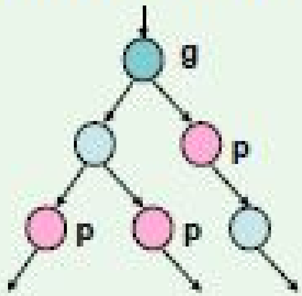
  AGEF p

- There exists a state from which p always holds:

  EFAG p

AG (g imply AF p)

- g leads to p: whenever **g** is true, **p** will inevitable become true.



- In UPPAAL: g --> p

- ***Promptness*** requirement: specify a maximum delay btw the occurrence of an event and its reaction. E.g., every transmission of a msg is followed by a reply within 5 units of time.

  $AG[send(m) \Rightarrow AF(<5)\ receive(m)]$

Queries Examples (1/3)

- ***Punctuality*** requirement: specify an exact delay btw events. E.g., there exists a computation during which the delay between transmitting m and receiving its reply is exactly 11 units of time.

$$EG[send(m) \Rightarrow AF\ (=11)\ receive\ (m)]$$

- ***Periodicity*** requirement: specify that an event occurs with a certain period.
  - A machine puts boxes on a moving belt that moves a constant speed.
  - To maintain an equal distance between successive boxes on the belt, the machine needs to put boxes periodically with a period of 25 time-units. (Specify Periodic behavior).

$$AG[AF(=25)\ putbox]$$

$$AG[putbox \Rightarrow \neg putbox\ U(=25)\ putbox]$$

Queries Examples (2/3)

- ***Minimal delay*** requirement: specify a minimal delay btw events. E.g., to ensure the safety of a railway system, the delay between two trains at a crossing should be at least 180 time units:

$$AG[\ train@cross \Rightarrow \neg train@cross\ U(>=180)\ train@cross\ ]$$

- ***Interval delay*** requirement: specify that an event must occur within a certain interval after another event.
  - Improve the throughput of the railway system -- Trains should have a maximal distance of 900 time-units.
  - The safety of the system must be remained.
  - Extend the previous minimal delay requirement:

$$AG[\ tac \Rightarrow \neg tac\ U(>=180) \land \neg tac\ U(<=900) tac\ ]$$

Queries Examples (3/3)

- ***Interval delay*** requirement: specify that an event must occur within a certain interval after another event.

$$AG[\ tac \Rightarrow \neg tac\ U(>=180) \wedge \neg tac\ U(<=900) tac\ ]$$

$$AG[\ tac \Rightarrow \neg tac\ U(=180)\ (AF(<=720)\ tac)\ ]$$

"After a train at the crossing it lasts 180 time units (safety requirement) before the next train arrives, and in addition this next train arrives within 720+180=900 time-units (the throughput requirement)".