Model-checking of Real-Time Systems

Cristina Seceleanu & Paul Pettersson
Embedded Systems, IDT, MDH

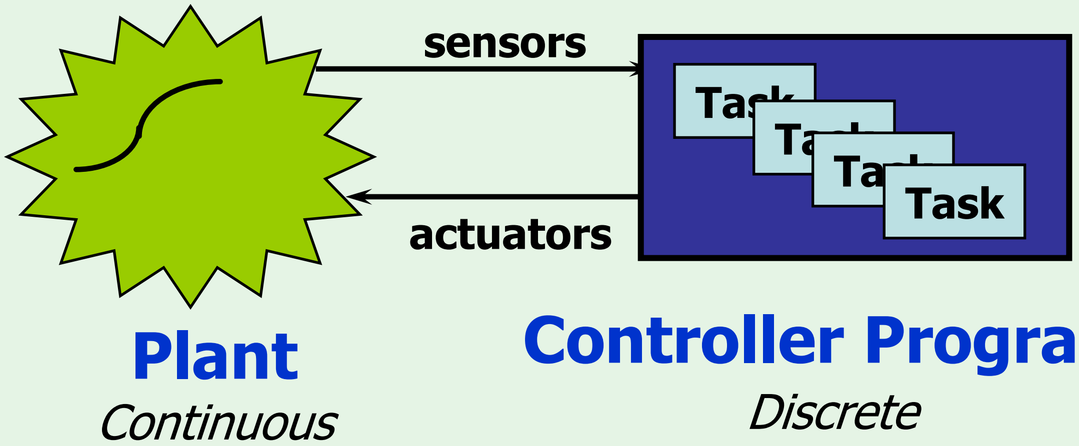
PROGRESS
A national Swedish Strategic Research Centre

MÄLARDALEN UNIVERSITY

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

SCIENCE • TECHNOLOGY • MEDICINE
Swedish Foundation for Strategic Research

Real-Time Systems

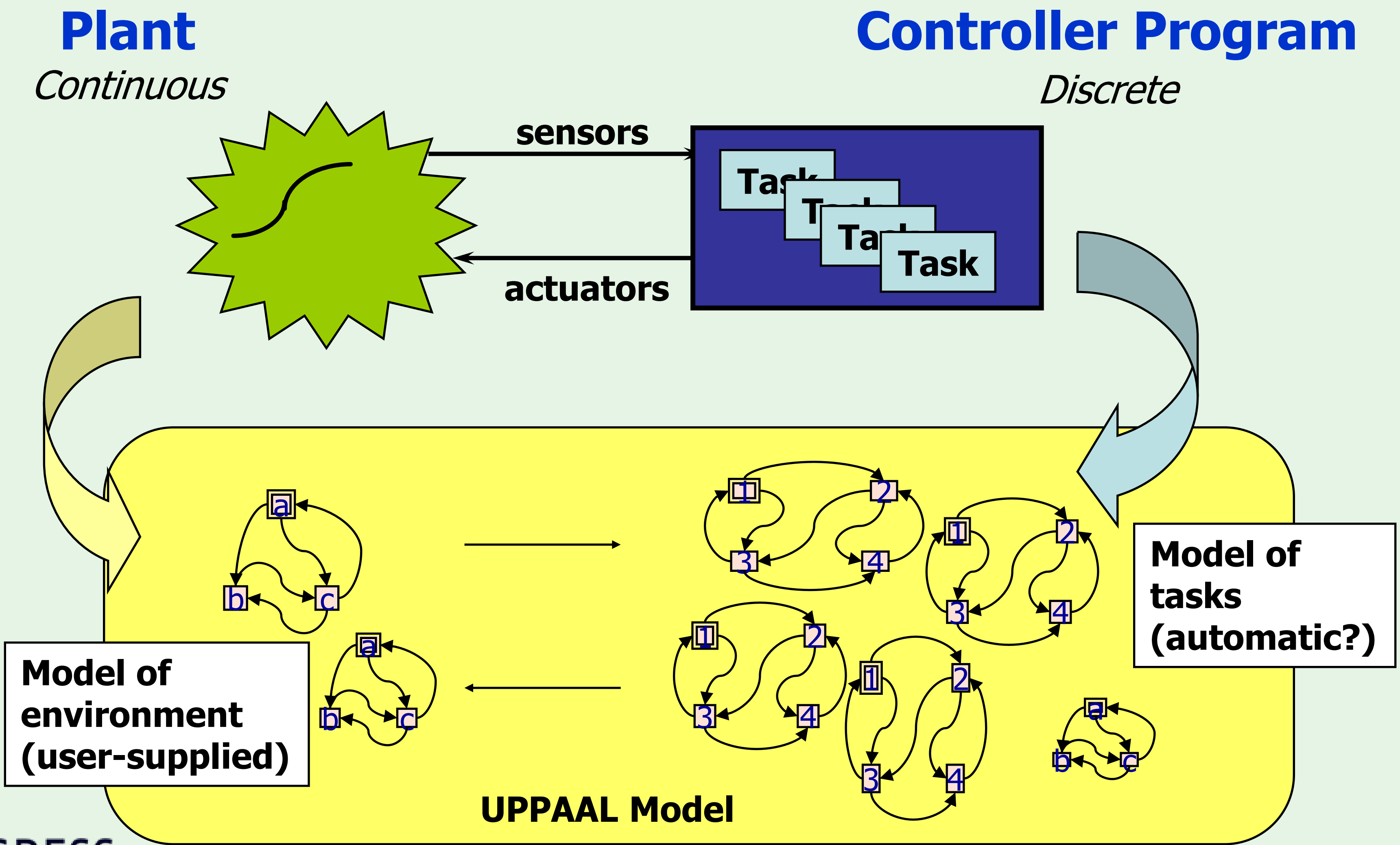


Real-Time System
A system where correctness not only depends on the logical order of events but also on their **timing**!!

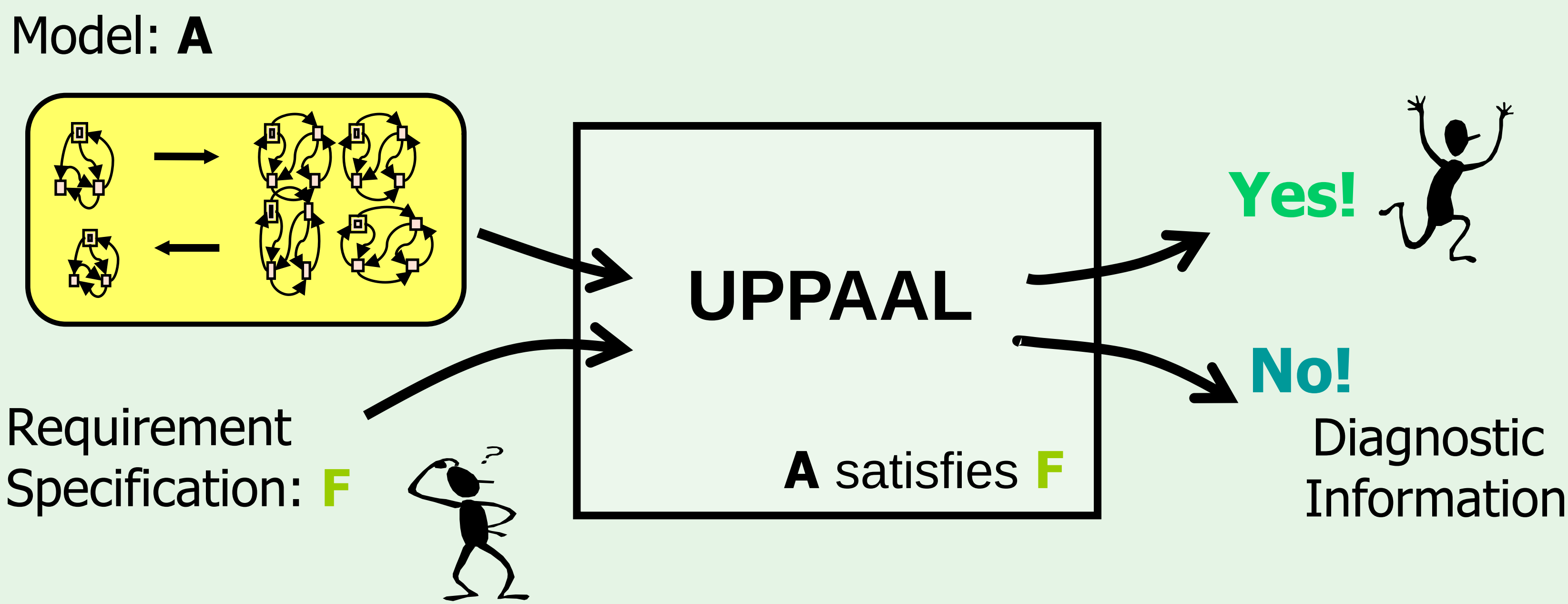
E.g.: Air Bags, Cruise Control, ABS
Process Control, Production Lines, Robots
Real-time Protocols
DVD/CD Players

PROGRESS

Real-Time Model-Checking



Model-Checking



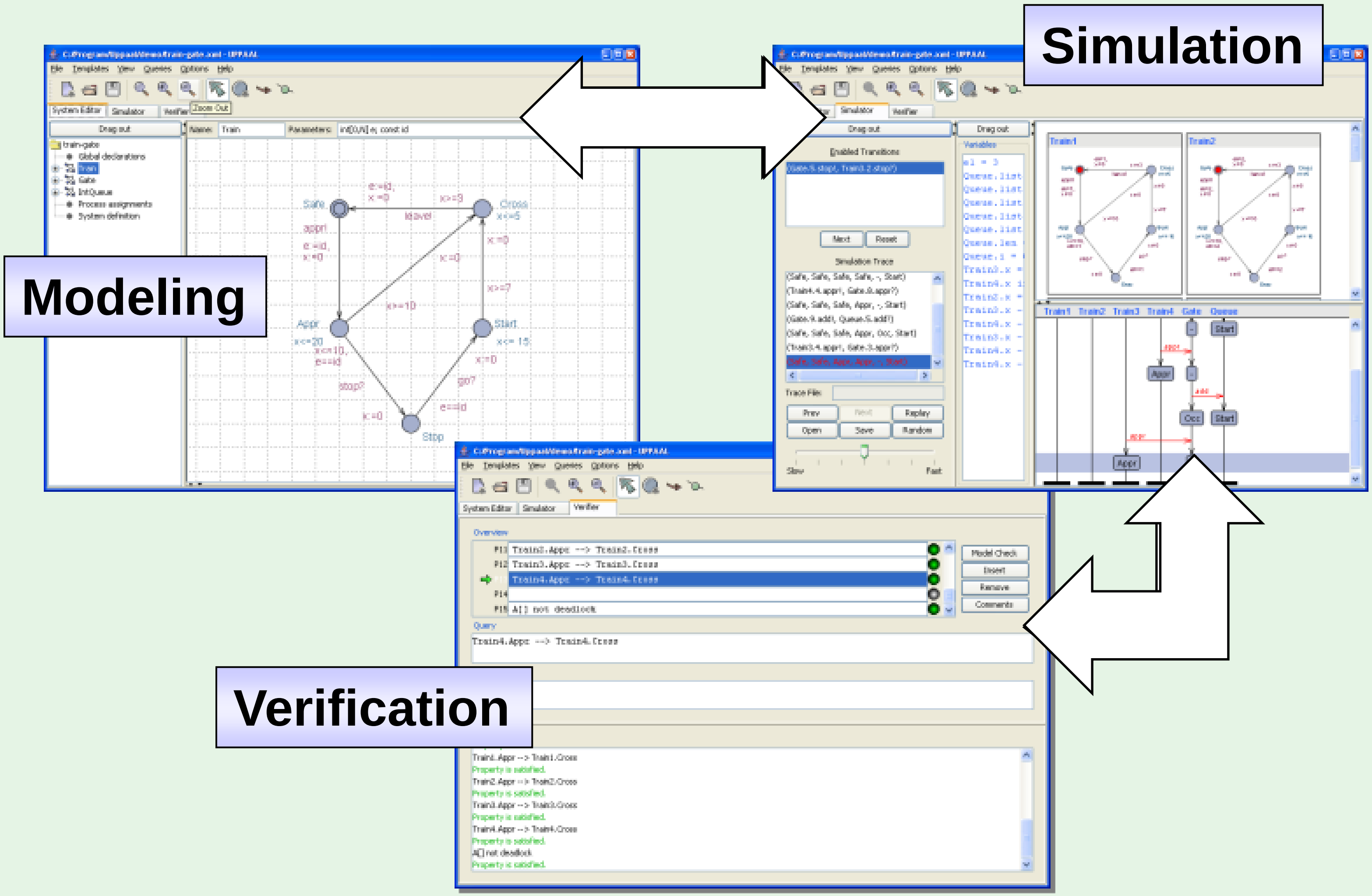
A – Model: Network of Timed Automata

F – Requirement: temporal logical formula, e.g.

- Invariant: something bad will never happen, something may happen
- Liveness: something will eventually happen

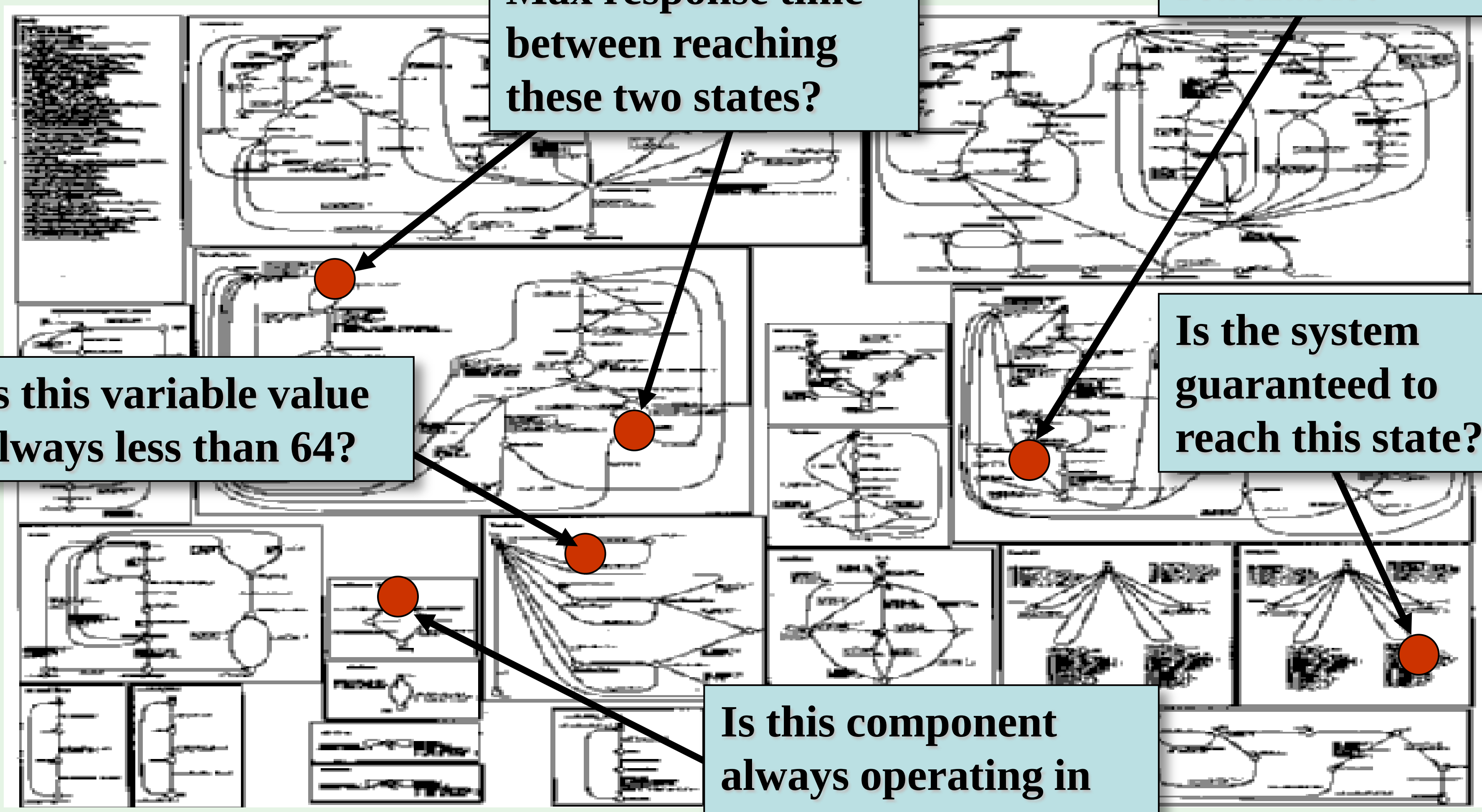
PROGRESS

Formal design and analysis



PROGRESS

Example model-based verification



Max response time between reaching these two states?

Is this error state reachable?

Is this variable value always less than 64?

Is the system guaranteed to reach this state?

Is this component always operating in this state?

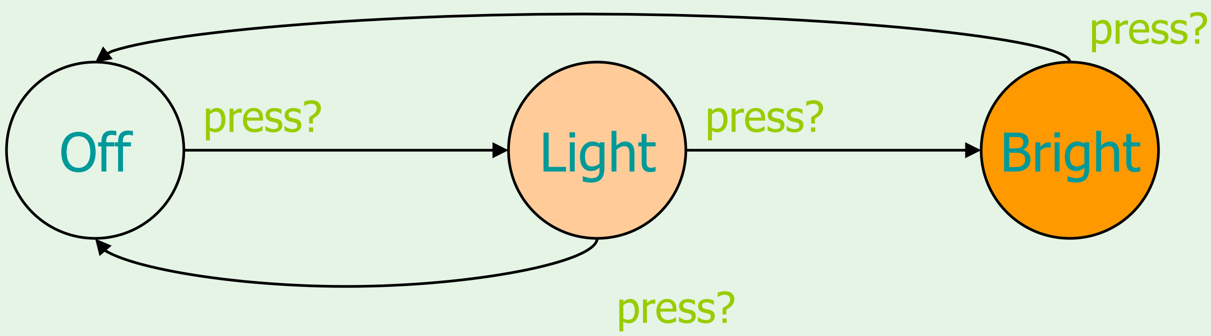
Model-checking of Real-Time Systems

- Modeling Formalism
- A Simple Example

Finite state automata

- Finite state graph, with
  - Set of nodes (states)
  - Set of edges (transitions)
  - Set of labels (actions)

Light Control

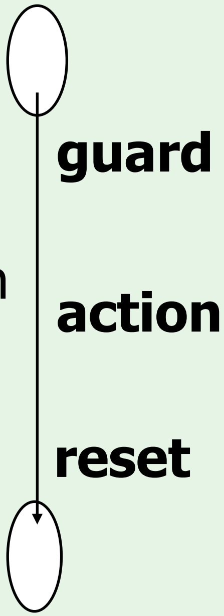


Wanted Behaviour:

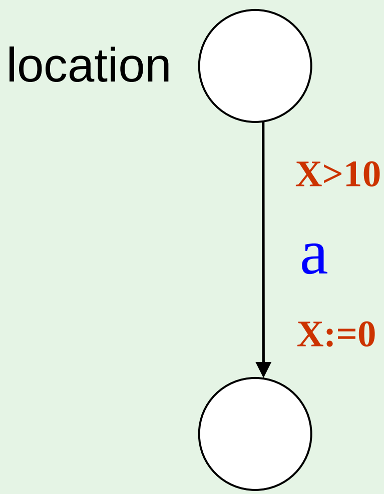
- pressed once = light
- pressed twice quickly = light will get brighter
- pressed again = light off.

Finite state automata 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
  - Timed automata is another example (clocks)
    - Guards and resets over clock variables on transitions
    - Infinite state!
- Semantics: Transition Systems

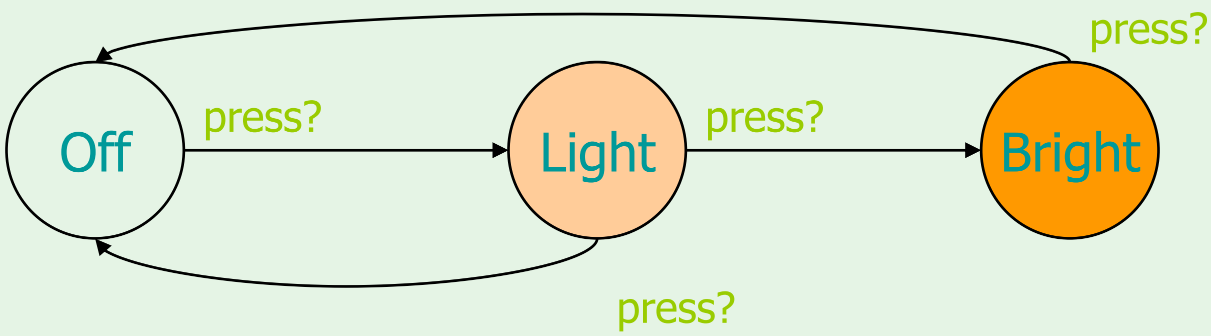


Timed Automata *Alur & Dill 1990*



- Guard
  - Timing constraints e.g. X>10
- Action
  - Synchronization e.g. a (handshake: a! for send, a? for receive)
- Clock reset
  - Reset clock to 0 e.g. X:=0

Light Control

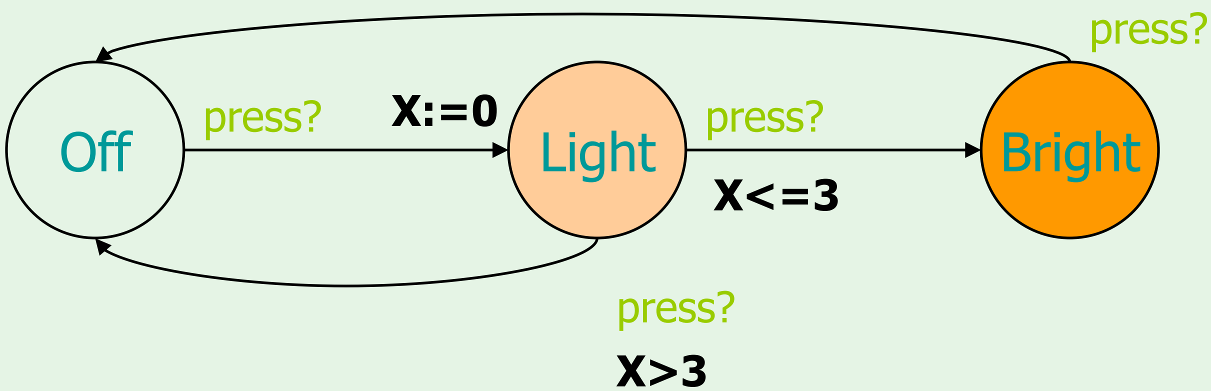


Wanted Behaviour:

- pressed once = light
- pressed twice quickly = light will get brighter
- pressed again = light off.

Timed Automata:

Light Control with Timing



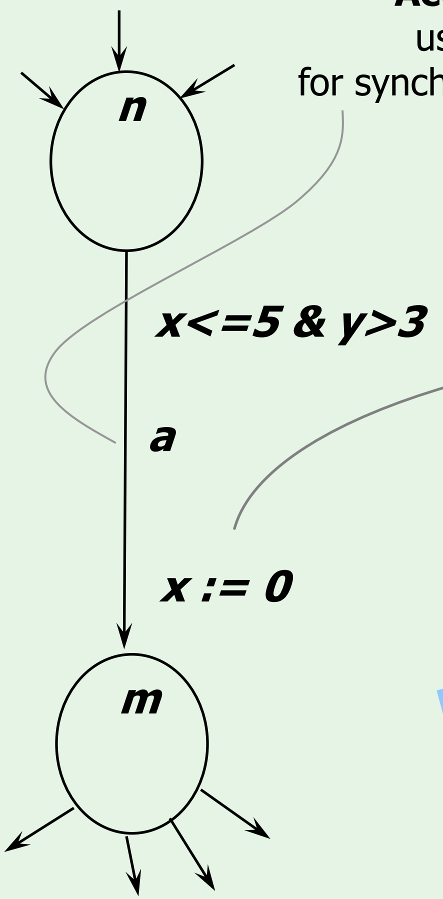
SOLUTION: Add real-valued clock **x** to measure the delay between press events

PROGRESS

Alur & Dill 1990

Timed Automata: Semantics

Action
used
for synchronization



Clocks: *x, y*

Guard
Boolean combination of **integer bounds** on **clocks**

Reset
Action performed on clocks

State
(***location*** , ***x***=v , ***y***=u) where v,u are in $\mathbf{R}$

Transitions

Discrete Trans

(***n*** , ***x***=2.4 , ***y***=3.1415) $\xrightarrow{a}$ (***m*** , ***x***=0 , ***y***=3.1415)

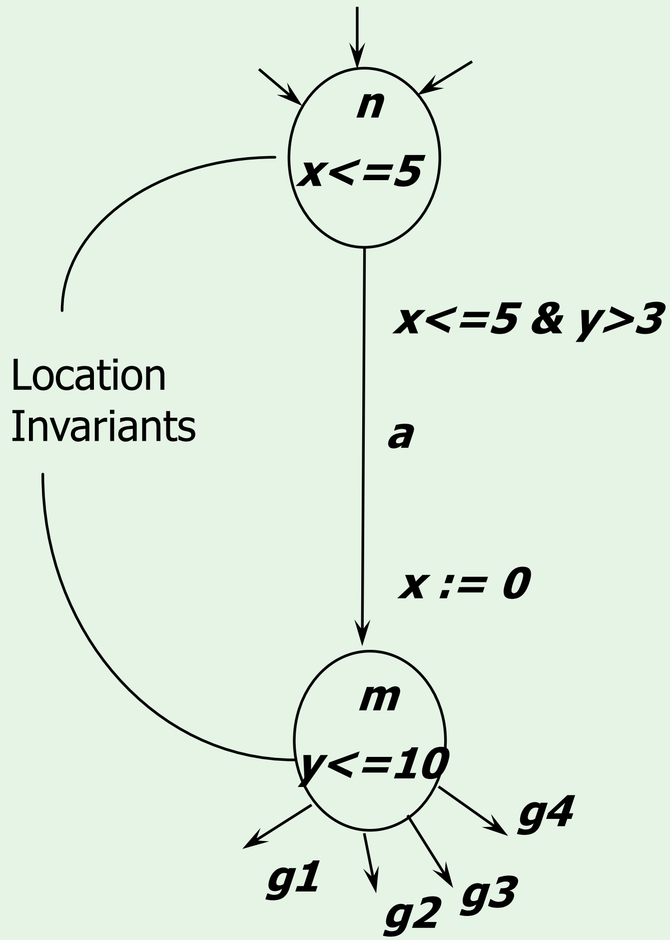
Delay Trans

(***n*** , ***x***=2.4 , ***y***=3.1415) $\xrightarrow{e(1.1)}$ (***n*** , ***x***=3.5 , ***y***=4.2415)

PROGRESS

Timed Automata

with Invariants



Clocks: *x, y*

Transitions

(***n*** , ***x***=2.4 , ***y***=3.1415) $\xrightarrow{\cancel{e(3.2)}}$

(***n*** , ***x***=2.4 , ***y***=3.1415) $\xrightarrow{e(1.1)}$ (***n*** , ***x***=3.5 , ***y***=4.2415)

Invariants ensure progress!!

PROGRESS

Clock Constraints

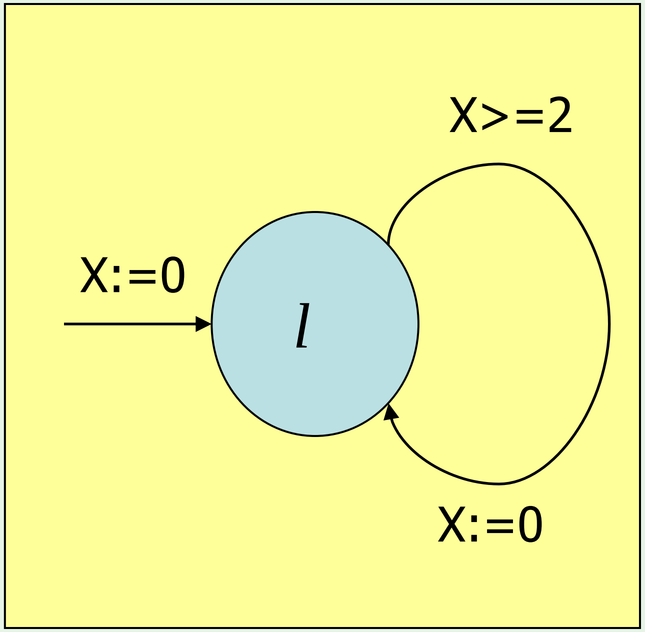
For set C of clocks with $x, y \in C$, the set of *clock constraints* over C, $\Psi(C)$, is defined by

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

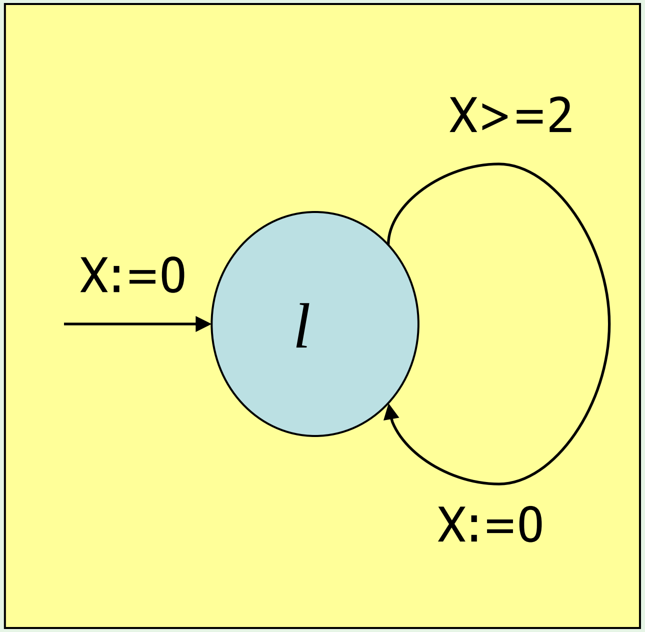
where $c \in \mathbb{N}$ and $\prec \in \{<, \leqslant\}$.

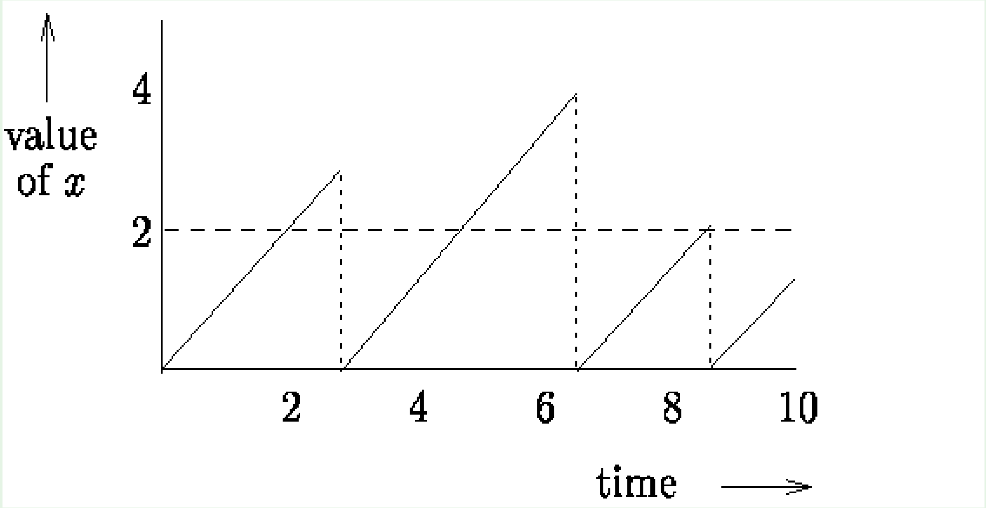
PROGRESS

Timed Automata: Example

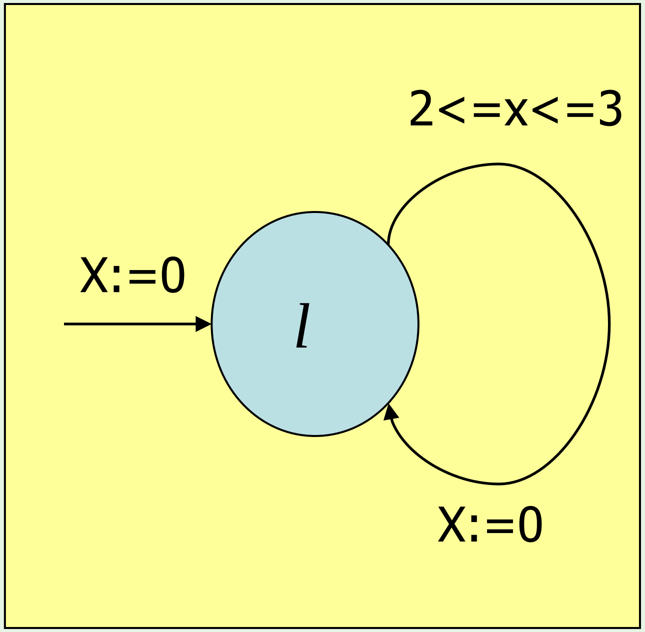


Timed Automata: Example



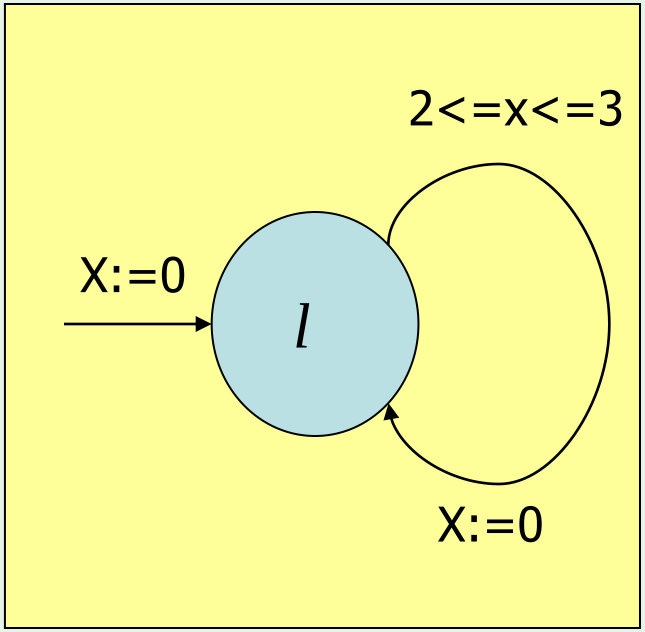


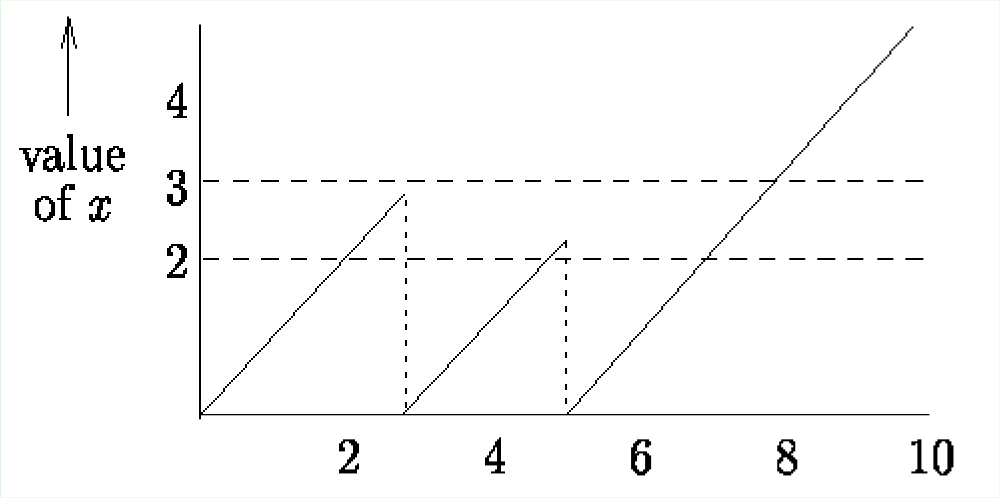
Timed Automata: Example



PROGRESS

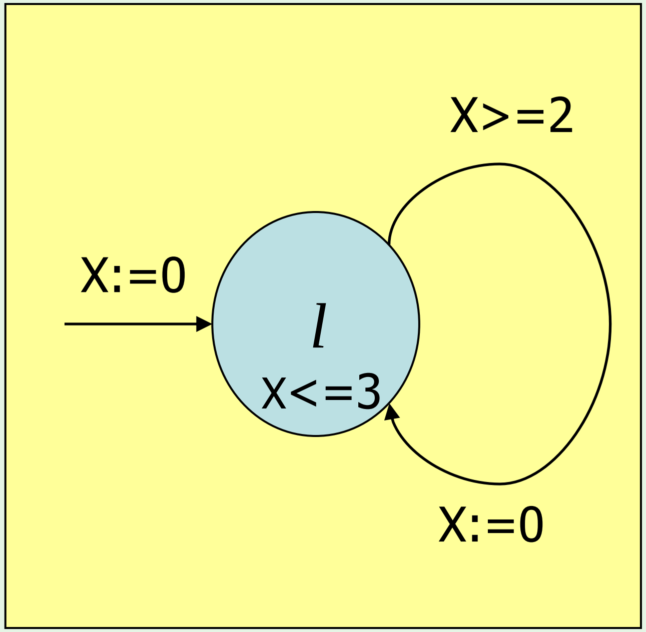
Timed Automata: Example





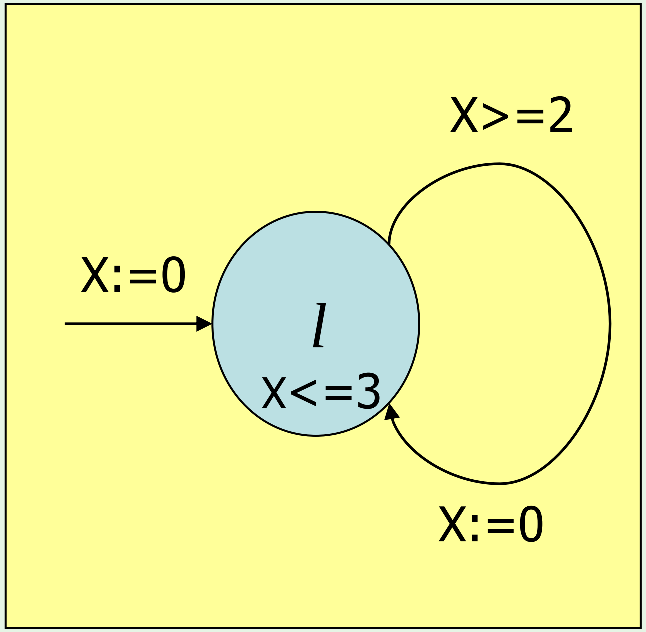
PROGRESS

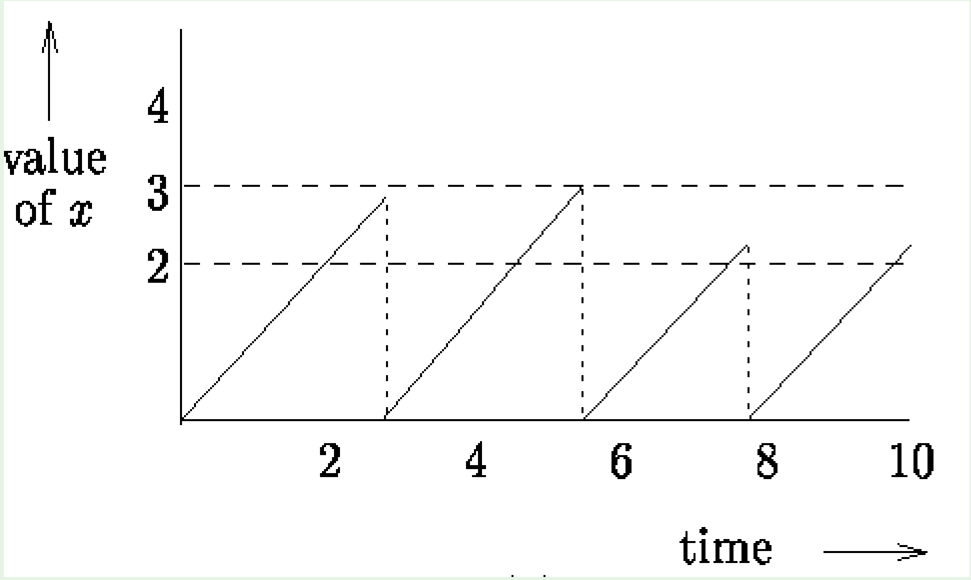
Timed Automata: Example



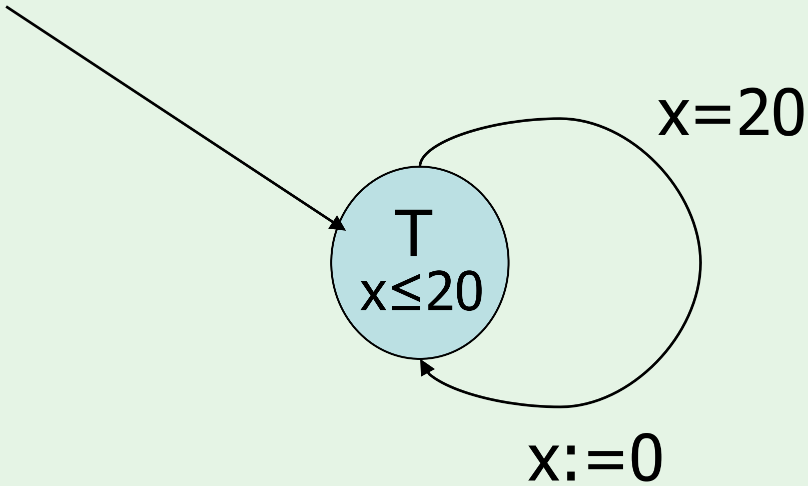
PROGRESS

Timed Automata: Example



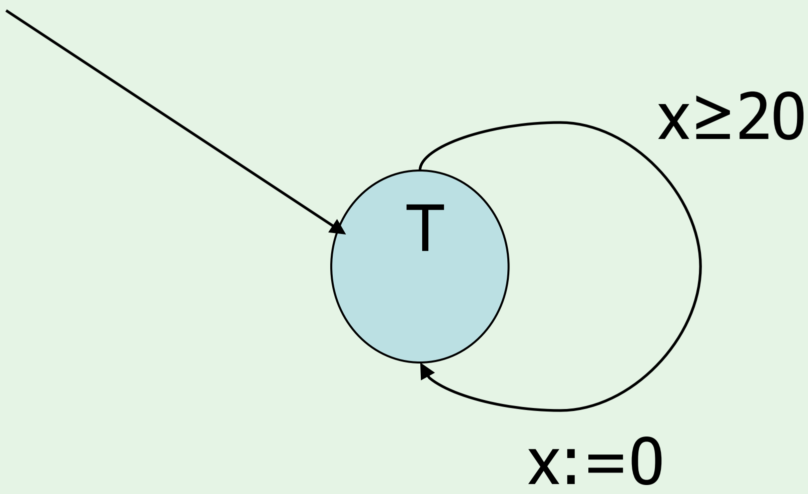


Timed Automata: Example
(periodic task, period 20)

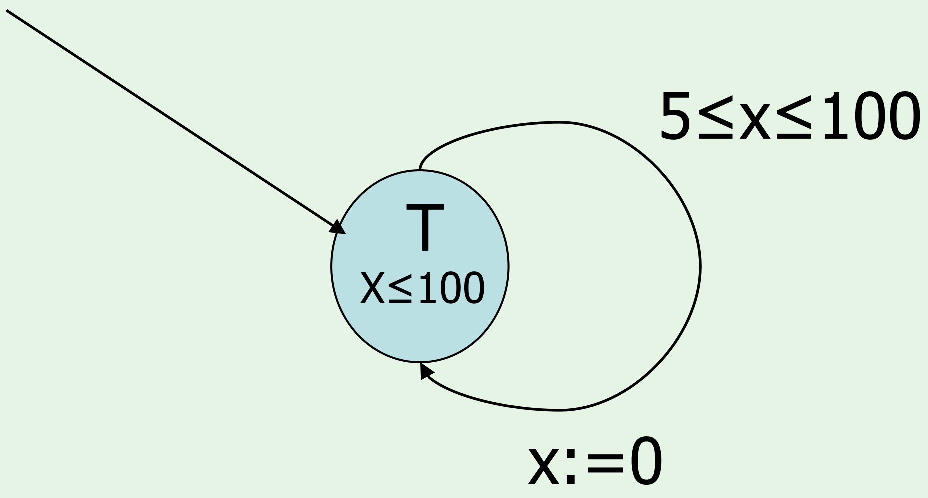


PROGRESS

Timed Automata: Example (sporadic task w min period 20)

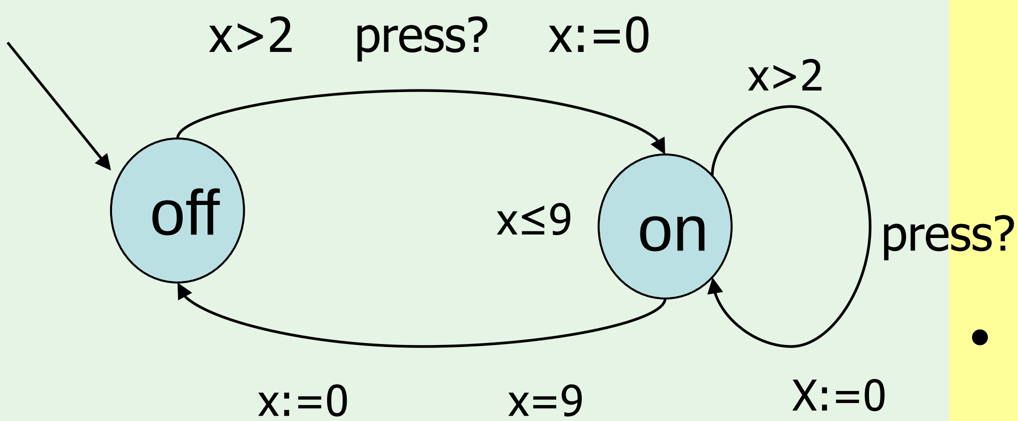


Timed Automata: Example (aperiodic task, every 5 to 100)



PROGRESS

Timed Automata: 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.

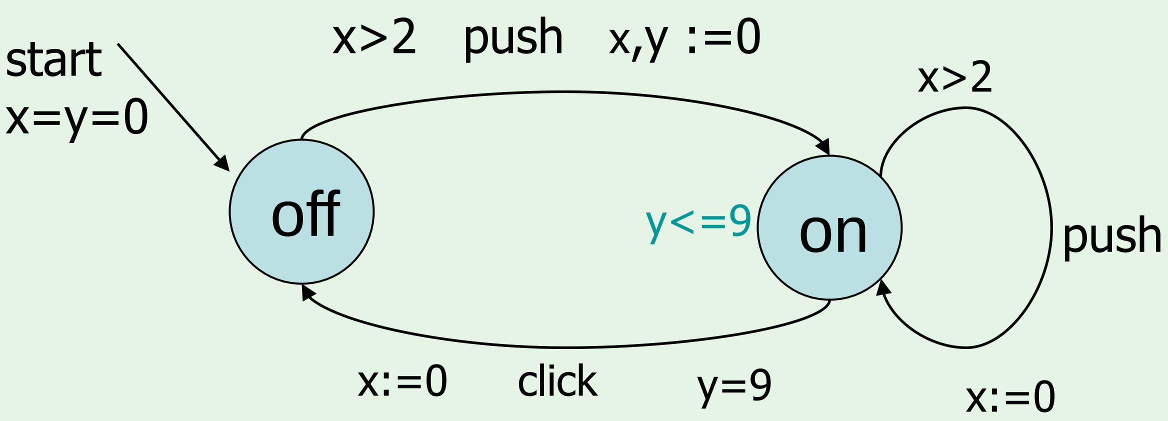
Semantics Definition

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

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

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

Timed Automata: Example



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

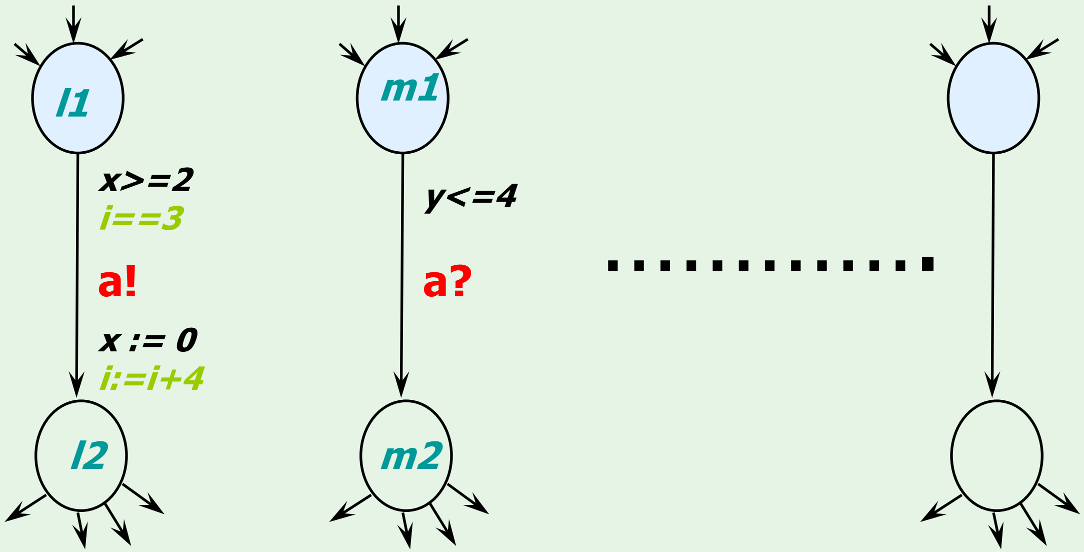
$$(on, x=y=0) \xrightarrow{\pi} (on, x=y=\pi) \xrightarrow{push}$$

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

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

Networks of Timed Automata

with (finite) integer variables



Two-way synchronization on *complementary* actions.

Closed Systems!

Example transitions

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

Datastructure: Zones

From infinite to finite

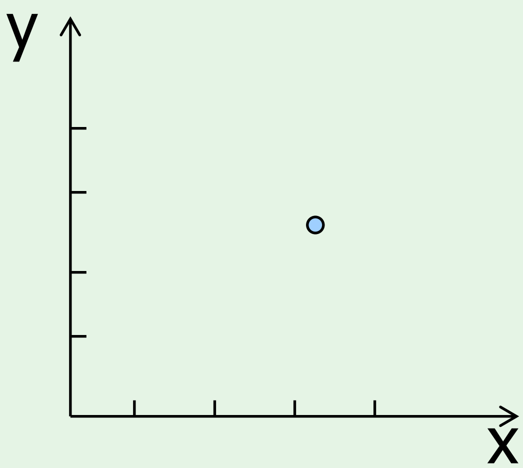
State

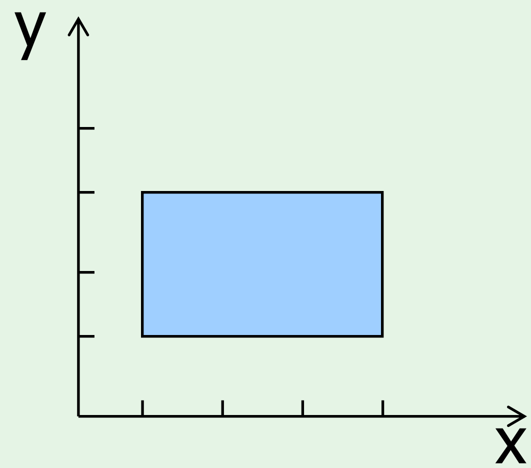
(n, x=3.2, y=2.5)

Symbolic state (set)

(n, 1≤x≤4, 1≤y≤3)

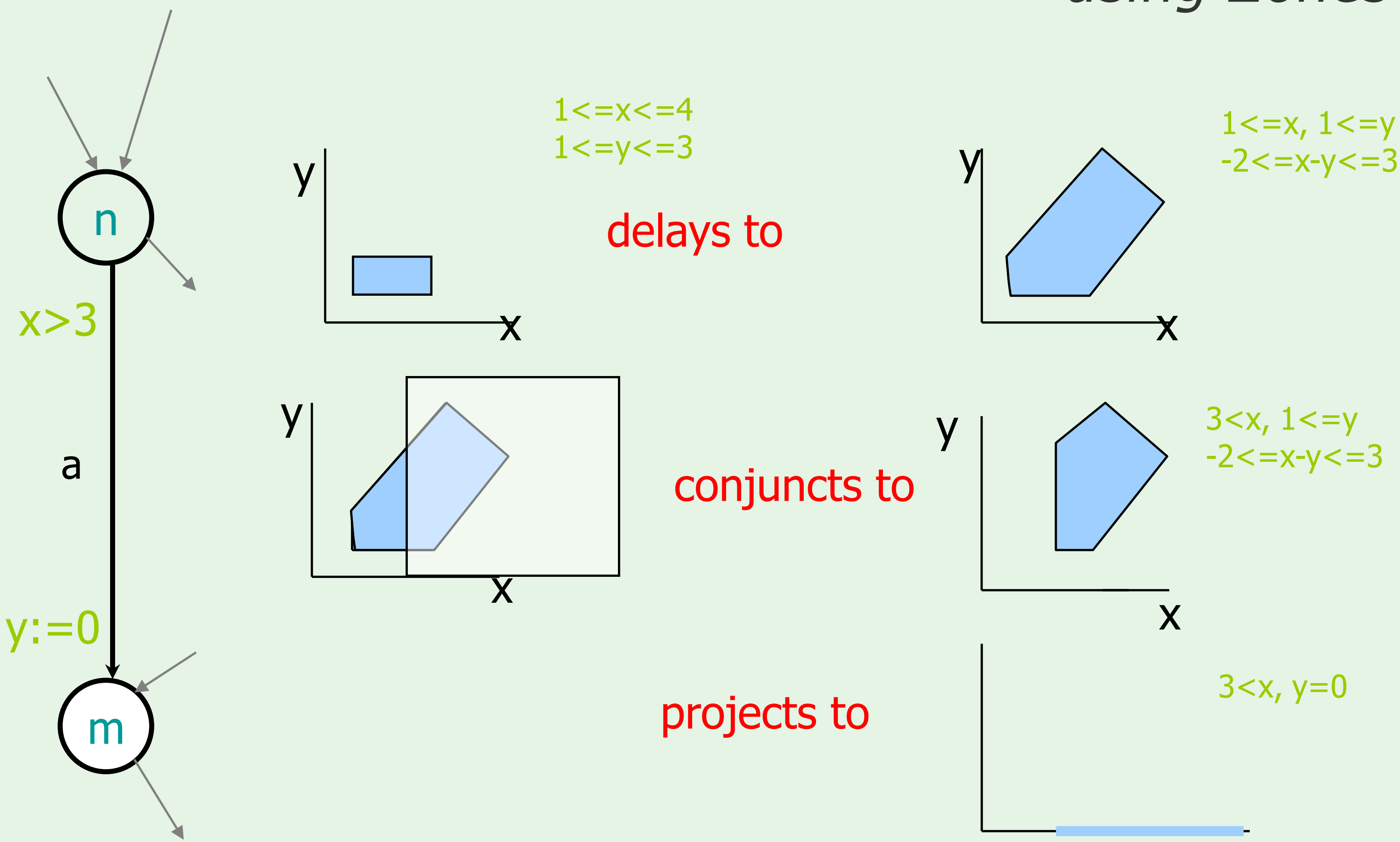
Zone:
conjunction of
x-y<=n, x<=>n





Symbolic Transitions

using Zones



Thus (n,1<=x<=4,1<=y<=3) =a => (m,3<x, y=0)

Zones = Conjuctive constraints

- A zone Z is a conjunctive formula:

  g_1 & g_2 & ... & g_n

  where g_i is a clock constraint:

  $x_i \sim b_i$ or $x_i - x_j \sim b_{ij}$

- Use a zero-clock x_0 (constant 0)
- A zone can be re-written as a set:

  $\{x_i - x_j \sim b_{ij} \mid \sim \text{ is } < \text{ or } \leq,\ i,j \leq n\}$

- This can be represented as a MATRIX, DBM (Difference Bound Matrices)

PROGRESS

Operations on Zones

- Delay: SP(Z) or $Z\uparrow$
  - $[Z\uparrow] = \{u+d \mid d \in R, u \in [Z]\}$

- Weakest pre-condition: WP(Z) or $Z\downarrow$ (the dual of $Z\uparrow$)
  - $[Z\downarrow] = \{u \mid u+d \in [Z]$ for some $d \in R\}$

- Reset: $\{x\}Z$ or $Z(x:=0)$
  - $[\{x\}Z] = \{u[0/x] \mid u \in [Z]\}$

- Conjunction
  - $[Z\&g] = [Z] \cap [g]$

PROGRESS

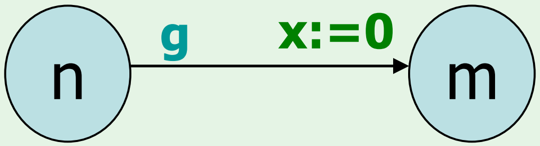
An important theorem on Zones

- The set of zones is closed under all constraint operations (including x:=x-c or x:=x+c)
- That is, the result of the operations on a zone is a zone
- That is, there will be a zone (a finite object i.e a zone/constraints) to represent the sets: $[Z\uparrow]$, $[Z\downarrow]$, $[\{x\}Z]$

PROGRESS

One-step reachability: Si→Sj

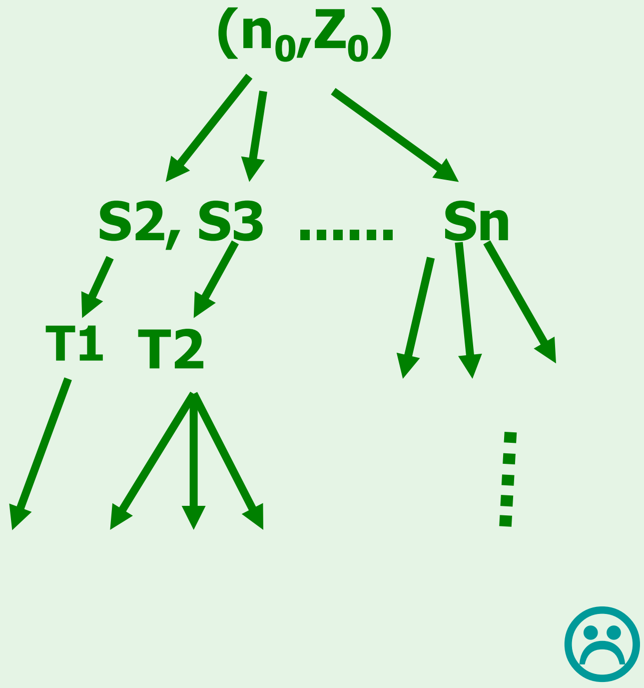
- Delay: (n,Z) → (n,Z') where $Z' = Z\uparrow \wedge inv(n)$
- Action: (n,Z) → (m,Z') where $Z' = \{x\}(Z \wedge g)$

  if 

- Successors(n,Z)={(m,Z') | (n,Z) →→(m,Z'), $Z' \neq \emptyset$}
  - Sometime we write: (n,Z)→(m,Z') if (m,Z') is a successor of (n,Z)

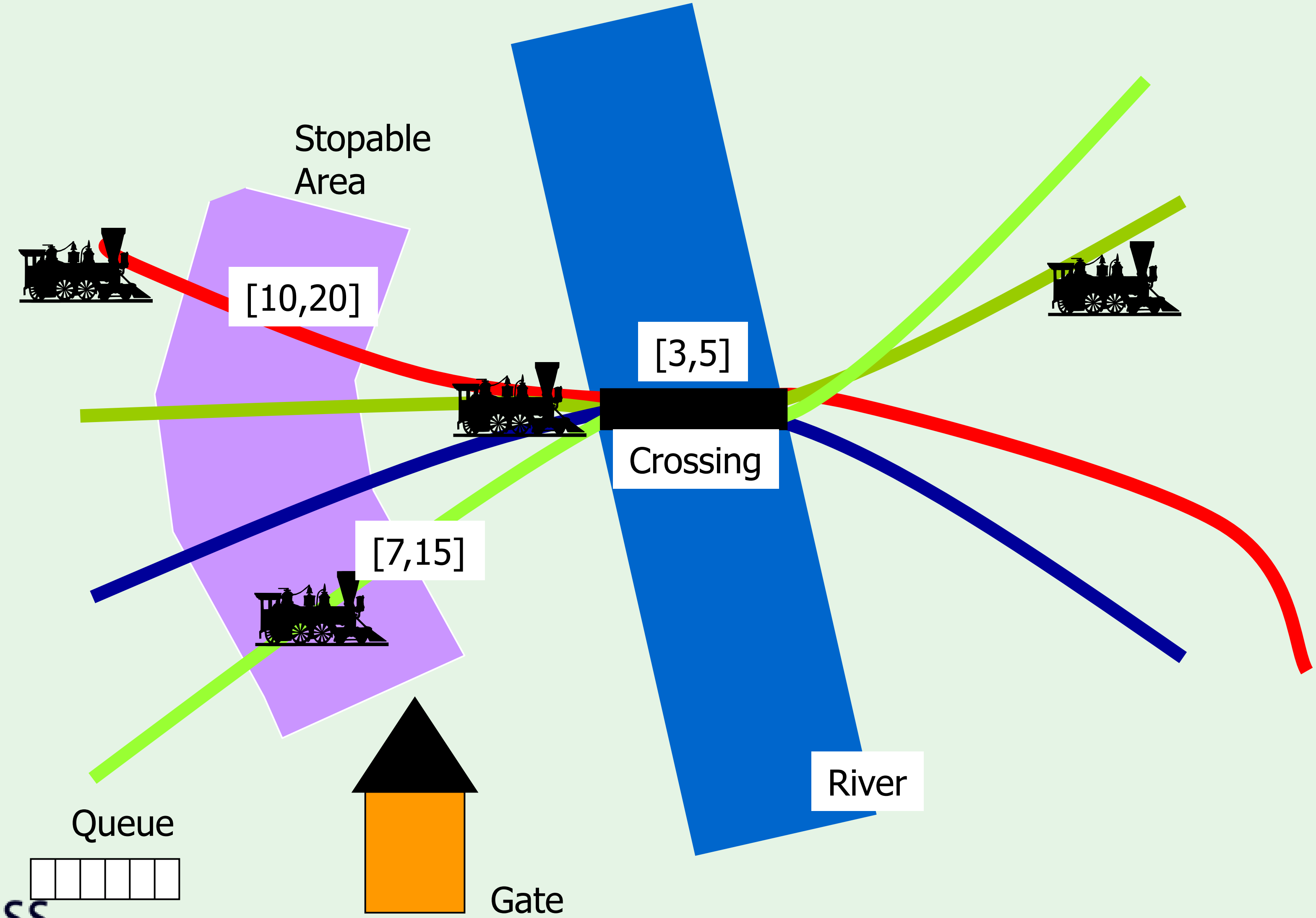
PROGRESS

Now, we have a search problem



PROGRESS

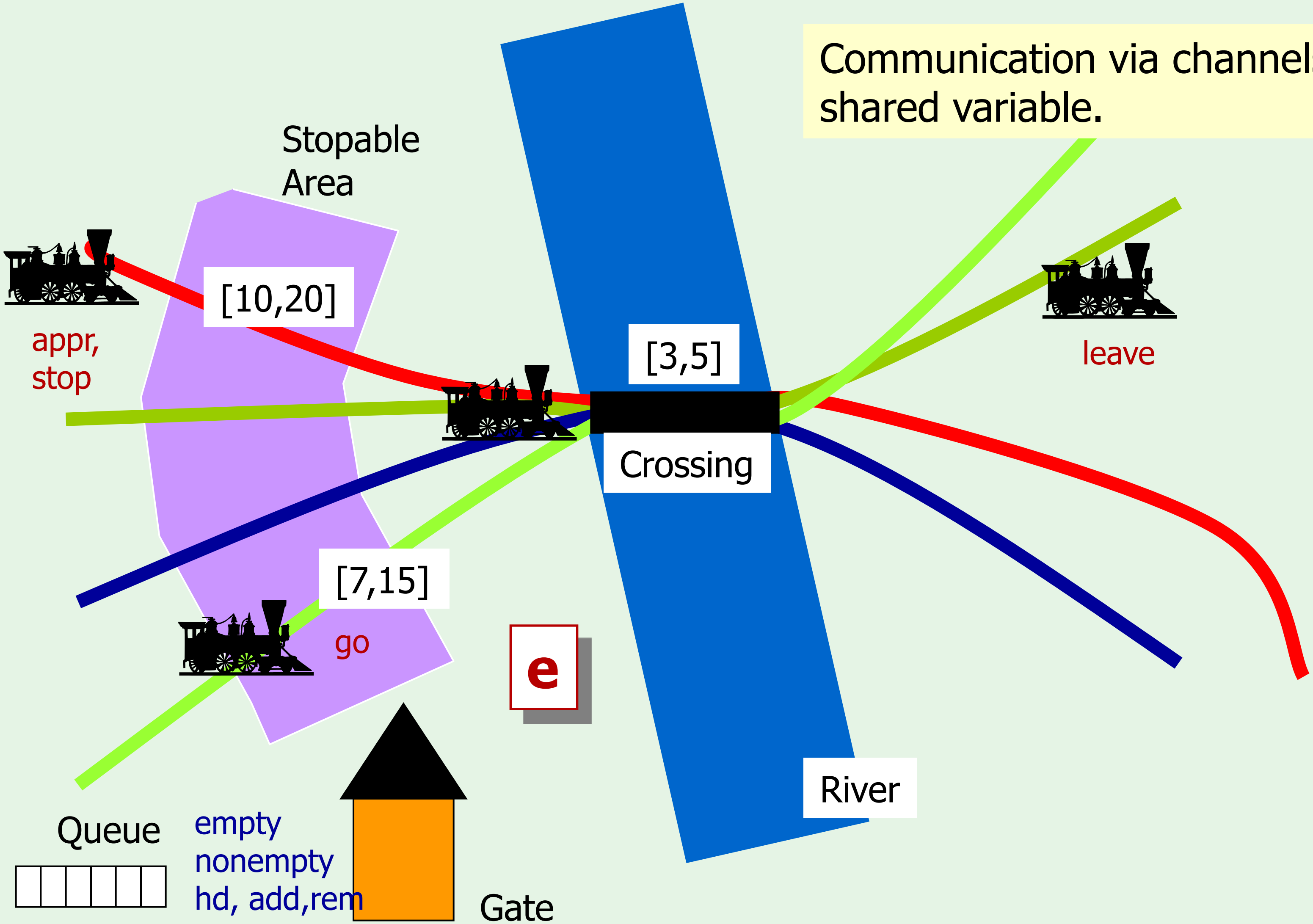
Train Crossing



PROGRESS

Train Crossing

Communication via channels and shared variable.

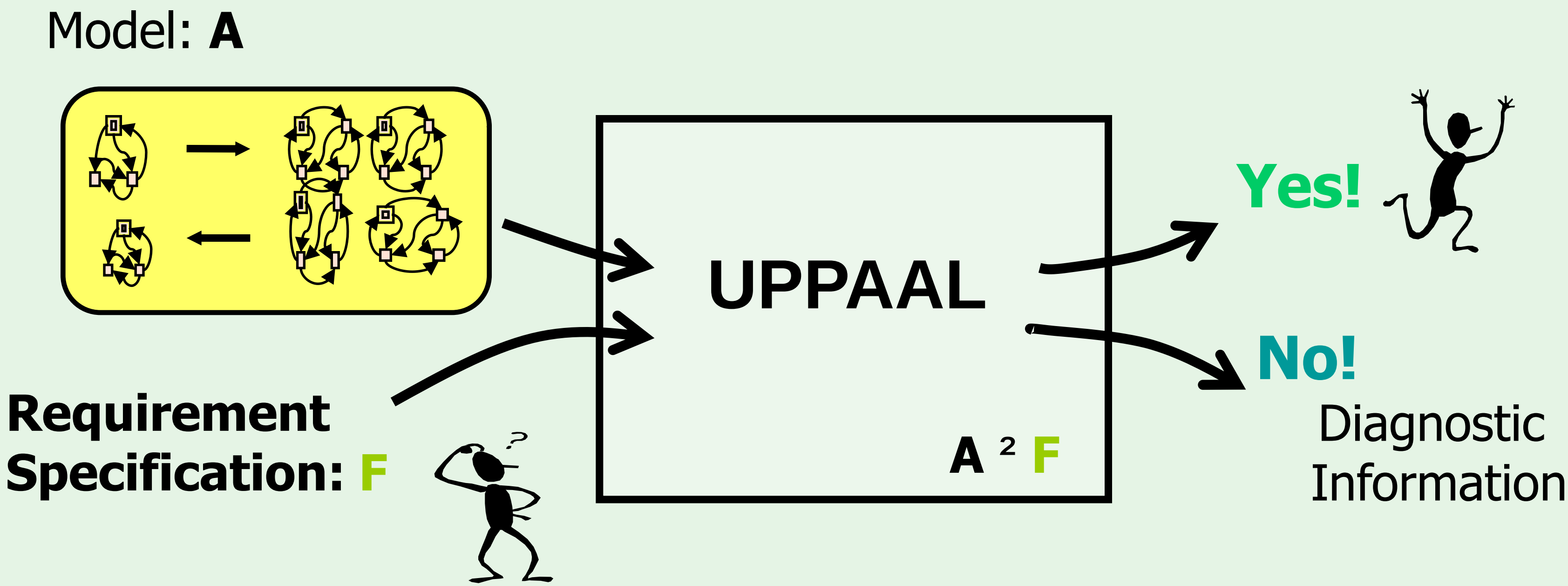


PROGRESS

How to specify what to check

SPECIFICATION OF REQUIRE- MENTS

How to specify what to check?!?



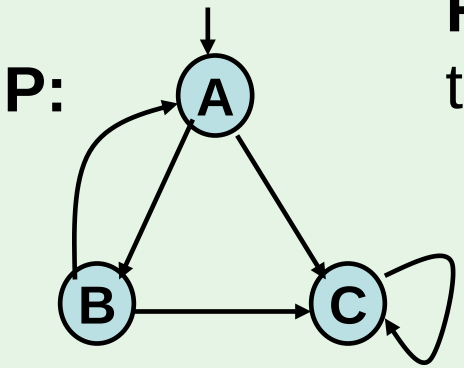
A – Model: Network of Timed Automata

F – **Requirement:** temporal logical formula, e.g.

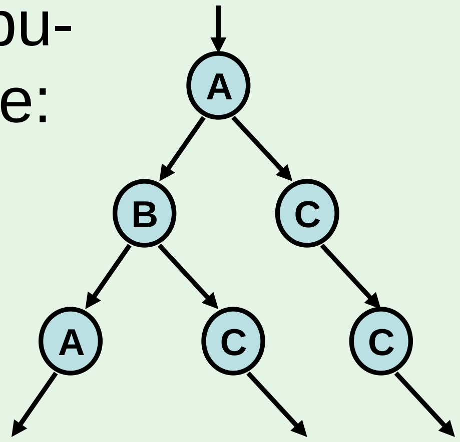
- Invariant: something bad will never happen, something may happen
- Liveness: something will eventually happen

Specification of Requirements

- TCTL - Timed Computation Tree Logic



P's compu-
tation tree:



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

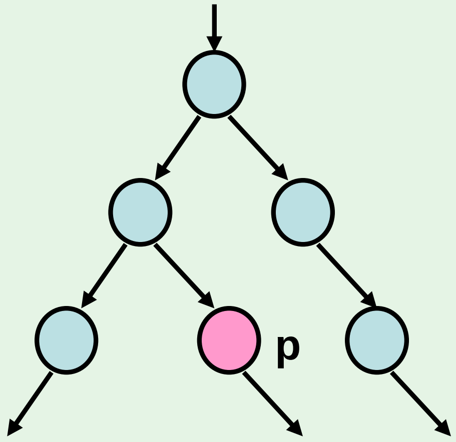
Quantifiers in TCTL

- 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[]`**.

E<>p – "p Reachable"

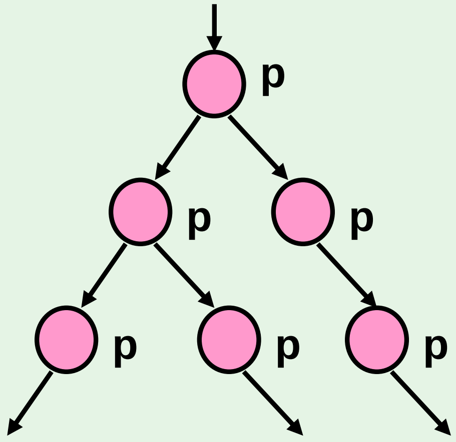
- It is possible to reach a state in which p is satisfied.



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

A[]p – “Invariantly p”

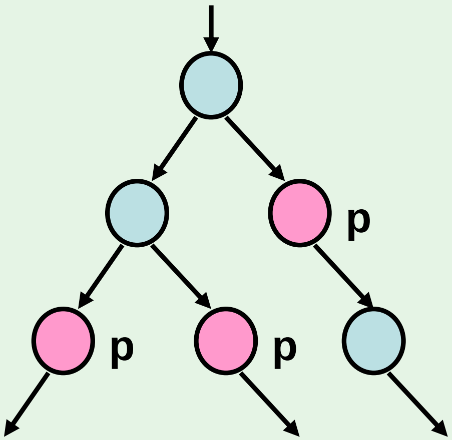
- p holds invariantly.



- p is true in all reachable states.

A<>p – “Inevitable p”

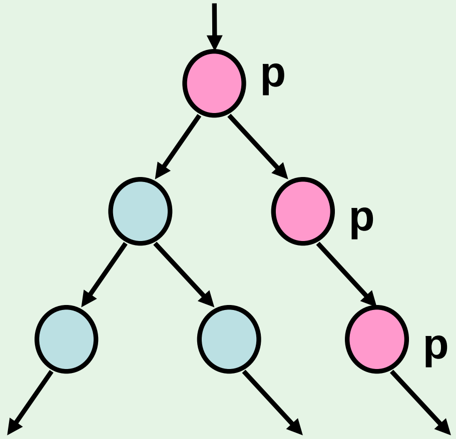
- p will inevitable become true
  - the automaton is guaranteed to eventually reach a state in which p is true.



- p is true in some state of all paths.

E[ ] p – “Potentially Always p”

- p is potentially always true.

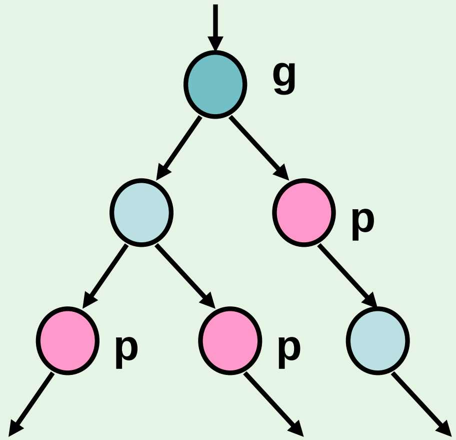


- There exists a path in which p is true in all states.

A[](g imply A<> p)

A[](g imply A<> p)

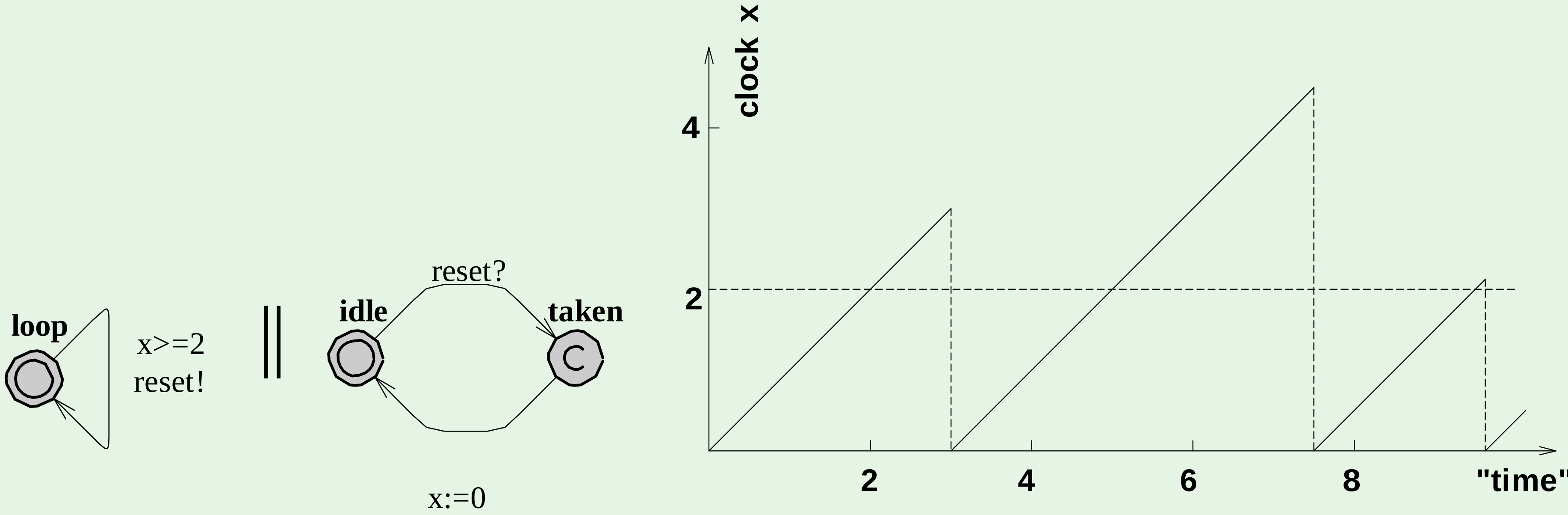
- g leads to p: whenever g is true, p will inevitable become true.



- In UPPAAL: g --> p

A Simple Example

- Uppaal uses a continuous time model.
- Concept of time:
  - a simple example that makes use of an observer.



(a) Test. (b) Observer. (c) Behaviour: one possible run.

First example with an observer.

A Simple Example (cont’d)

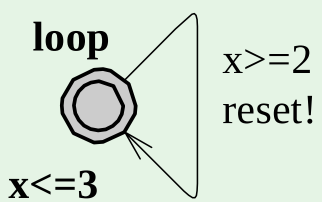
- Properties to be verified in Uppaal:
  - A [] Obs.taken imply x>=2

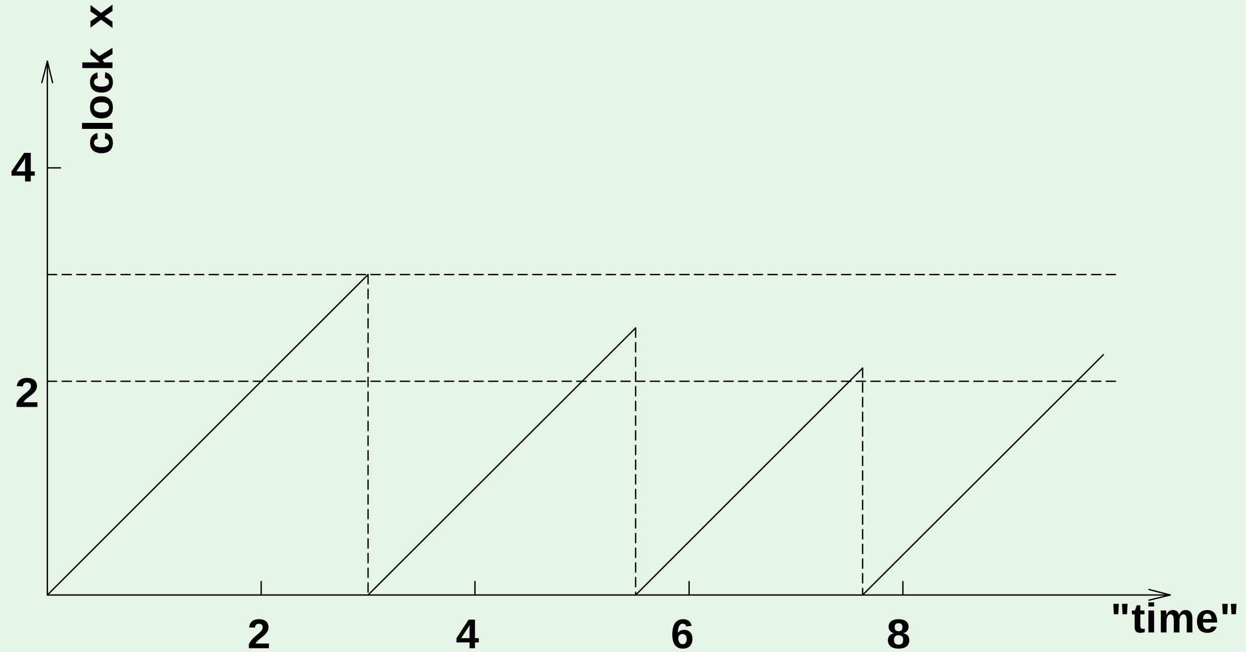
    all resets off x will happen when x is above 2
  - E<> Obs.idle and x>3

    this property requires, that it is possible to reach a state where Obs is in the location idle and x is bigger than 3.

A Simple Example: Invariant

- Add an invariant to the location loop, as shown in figure below
- The system is not allowed to stay in the state more than 3 time units, so that the transition has to be taken, and the clock reset in our example holds.





(a) Test. (b) Updated behavior with an invariant.

A Simple Example (cont'd)

- Properties that hold in Uppaal:
  - A[] Obs.taken imply (x>=2 and x<=3)
    - shows that the transition is taken when x is between 2 and 3, i.e., after a delay between 2 and 3.
  - E<> Obs.idle and x>2
    - it is possible to take the transition when x is between 2 and 3. The upper bound 3 is checked with the next property.
  - A[] Obs.idle imply x<=3
    - to show that the upper bound is respected.

The former property E<> Obs.idle and x>3 no longer holds.

References

1. A Tutorial on UPPAAL 4.0
(Gerd Behrmann, Alexandre David, and Kim G. Larsen)

http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf

2. UPPAAL Tool :

http://www.uppaal.org/

PROGRESS