Modeling, Vefirication, and Testing of Real-time System

(Adapted from Brian Nielsen's Slides)

Agenda

- Real-time systems
- Timed Automata (TA)
- Modeling real-time systems using Uppaal
- Modelling checking real-time systems
- Model-based testing of real-time systems

Real-time Systems

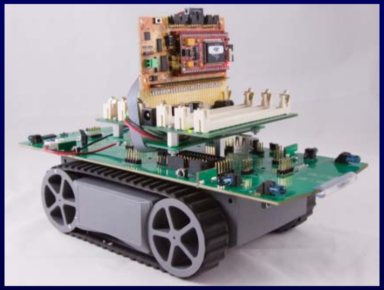
Real-time Systems

Real Time System

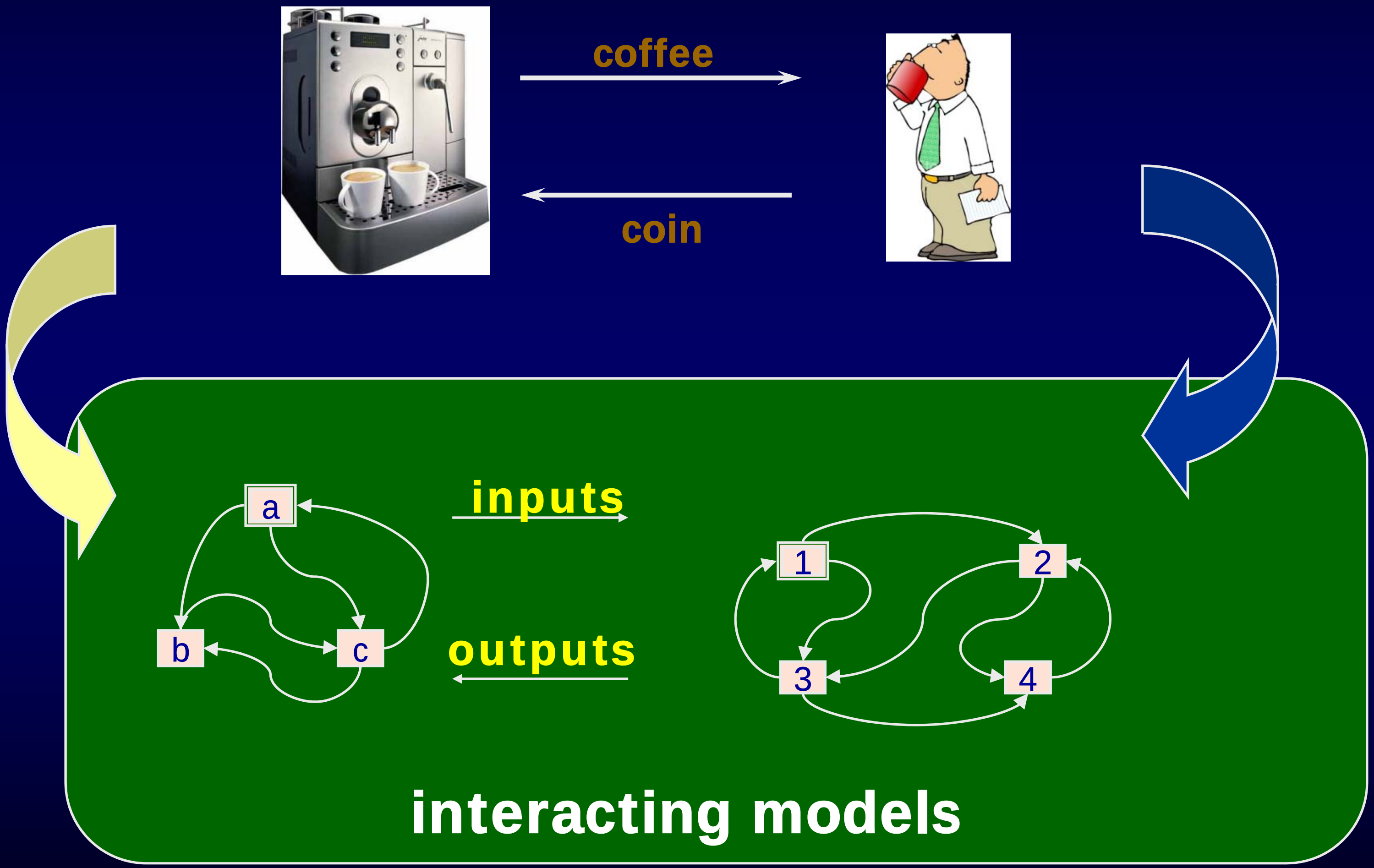
A system where correctness not only depends on the logical order of events, but also on their **timing**!!

Eg.:

- Real-time Protocols
- Pump Control
- Air Bags
- Robots
- Cruise Control
- Drive-by-Wire
- ABS
- CD Players
- Production Lines



Real-time System Modelling



Discrete-Time vs. Continous-Time

- Discrete-time
  - Time equally devided into small peices (slices)
  - Event can only occur at the end of some time slice
  - Approapriate for synchronous systems

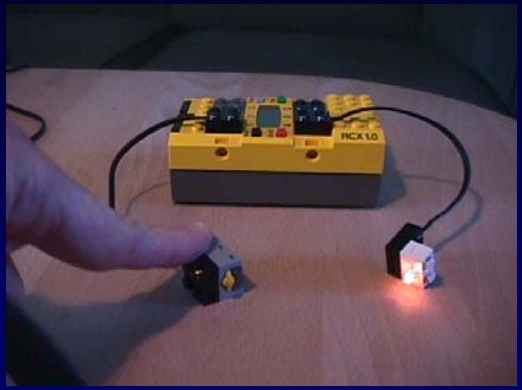
that's what we are concerned with

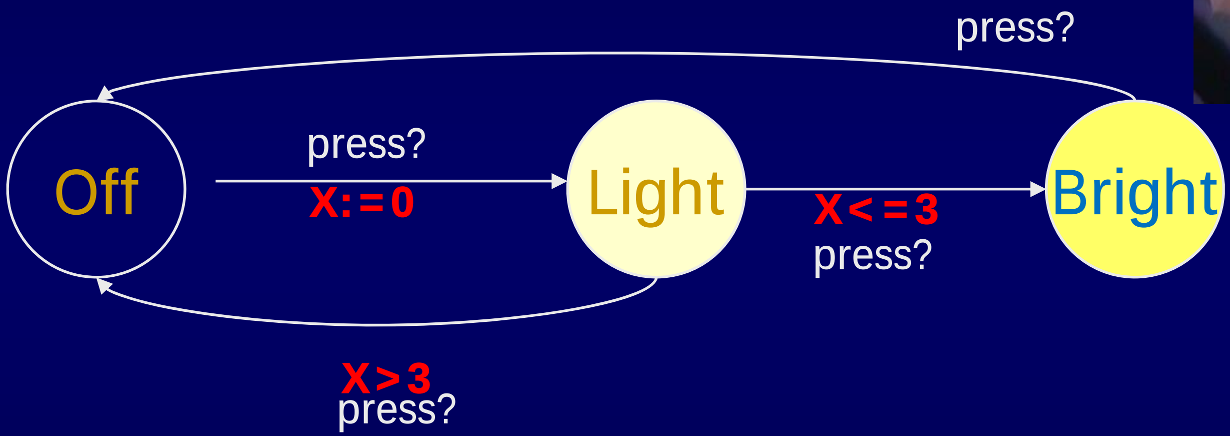
- Continous-time
  - Time slices can be "infinitely" small
  - Event can occur anytime
  - Appropriate for asynchronous systems
    - e.g., distributed systems, comm. Protocols, etc.

Timed Automata

A formalism for Continous-Time Modeling
of Real-time Systems

An Intelligent Light Control



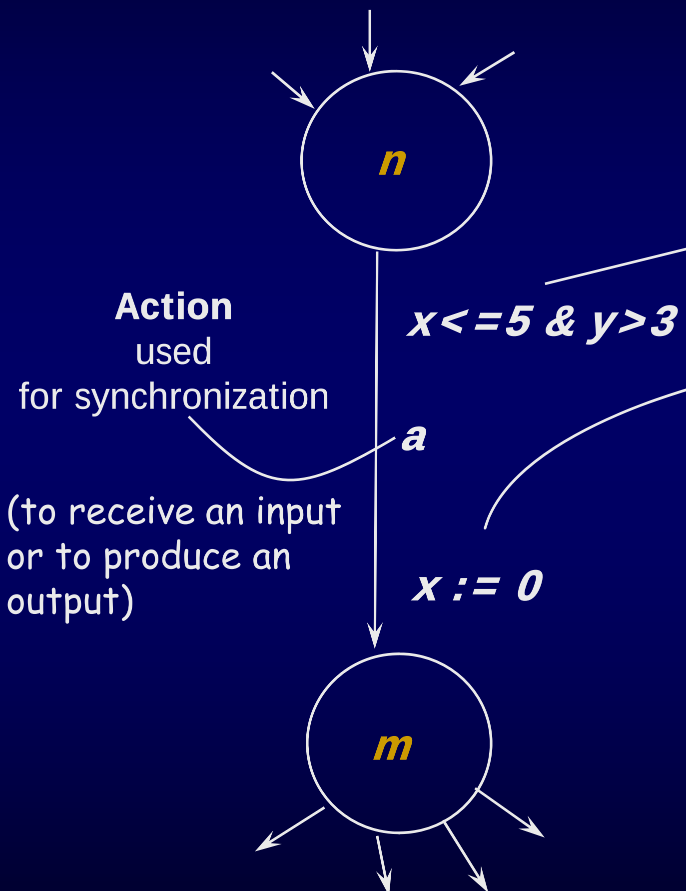


WANT: if "press" is issued twice quickly then the light will get brighter; if "press" is issued twice slowly the light is turned off.

Solution: Add a real-type variable (a real-valued clock) x

Timed Automata

(Alur & Dill 1990)



Clocks: ***x, y***

Guard
Boolean combination of comp with integer bounds

Reset
Action performed on clocks

State

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

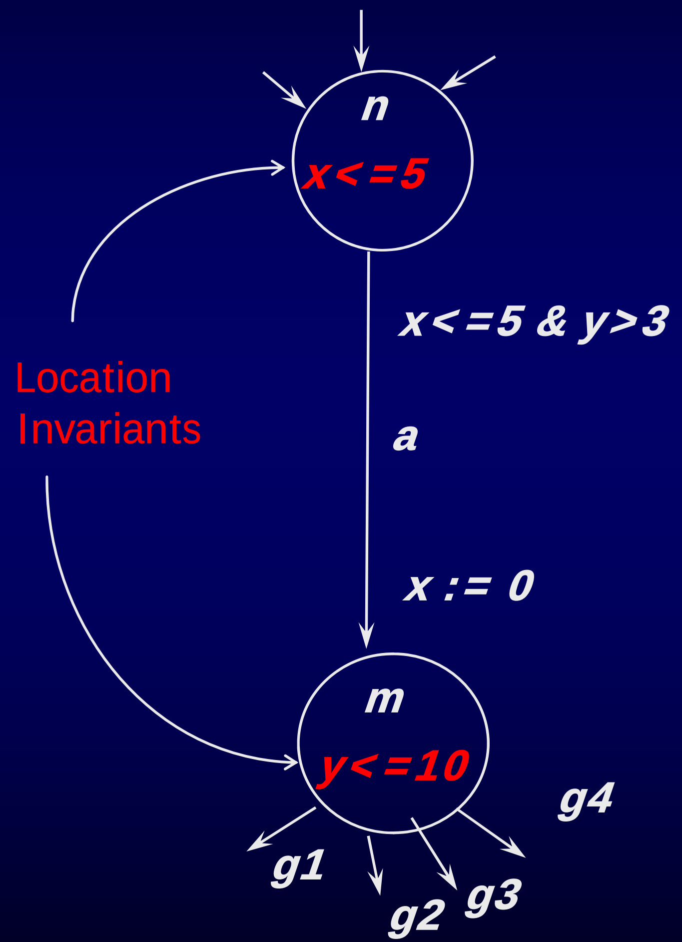
Transitions

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

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

Timed Automata

location invariants



Clocks: *x, y*

Transitions

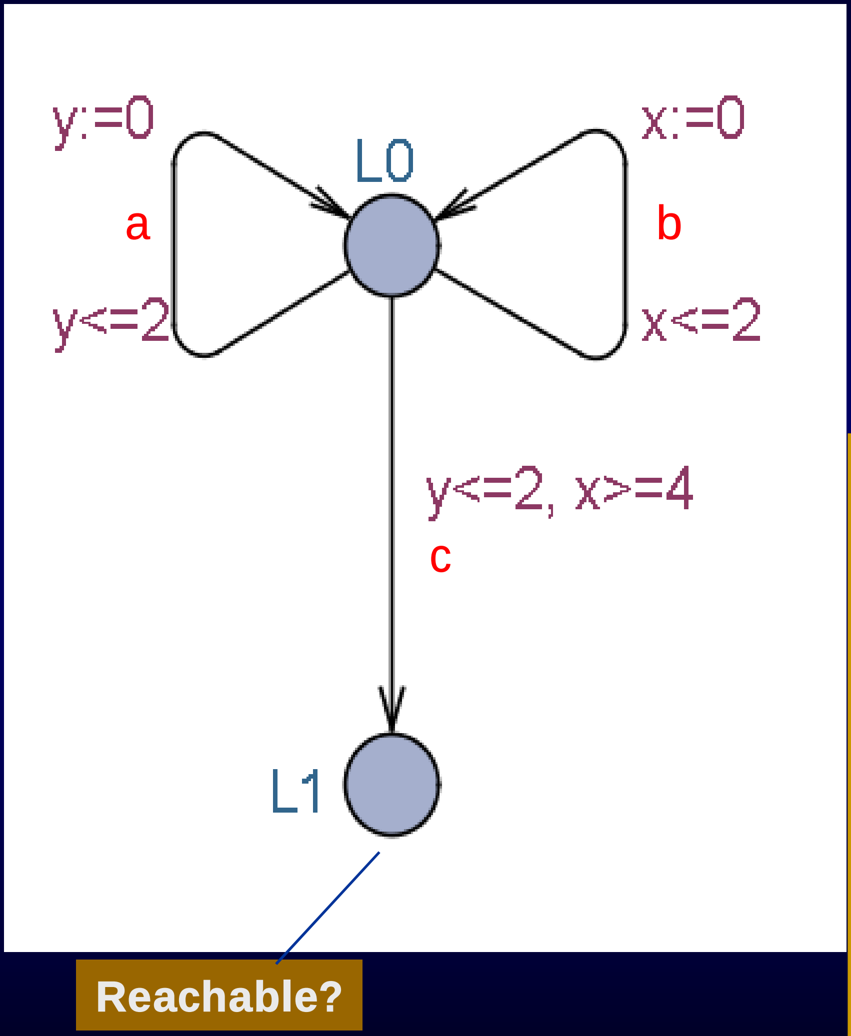
(***n*** , ***x***=2.4 , ***y***=3.1415) ~~*e(3.2)*~~ →

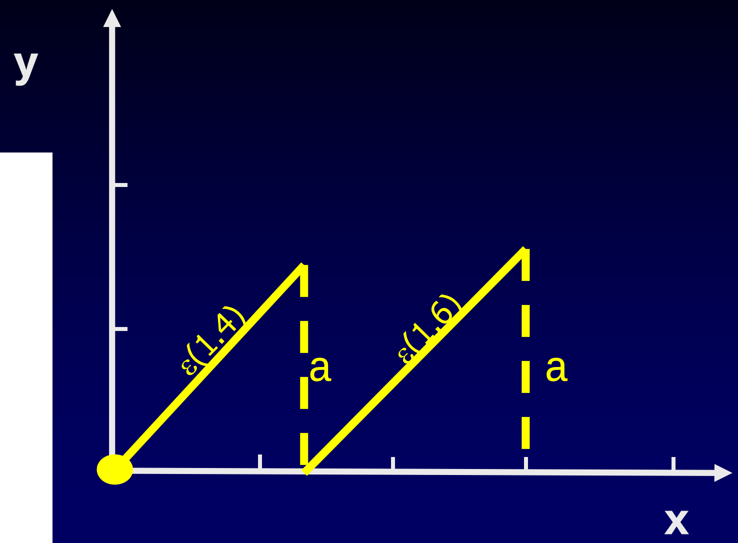
(***n*** , ***x***=2.4 , ***y***=3.1415) *e(1.1)* → (***n*** , ***x***=3.5 , ***y***=4.2415)

Invariants ensure progress!!

you cannot stay in this location forever;
you must leave before the deadline!

Example





(L0,x=0,y=0)

→ $_{\varepsilon(1.4)}$

(L0,x=1.4,y=1.4)

→ $_{a}$

(L0,x=1.4,y=0)

→ $_{\varepsilon(1.6)}$

(L0,x=3.0,y=1.6)

→ $_{a}$

(L0,x=3.0,y=0)

Zones

from infinite to finite

a bunch of concrete states

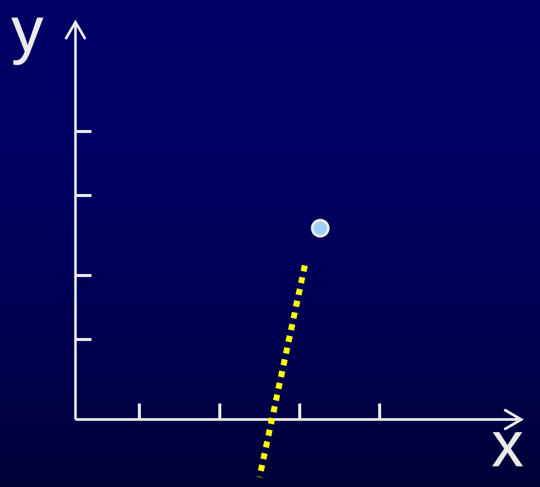
a state

(n, x=3.2, y=2.5)

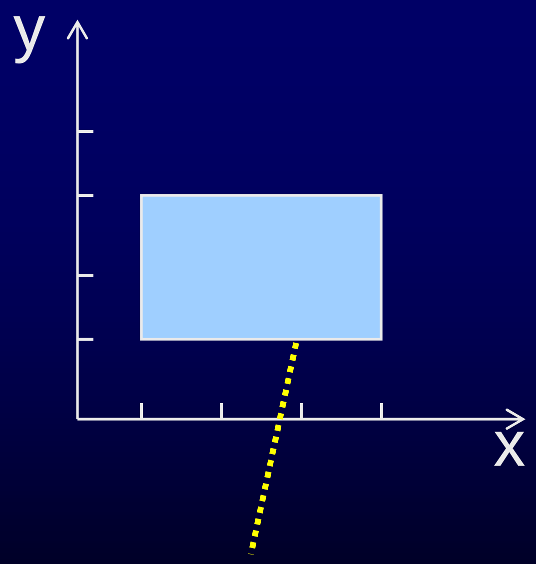
a symbolic state (set)

(n, $1 \leq x \leq 4$, $1 \leq y \leq 3$)

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

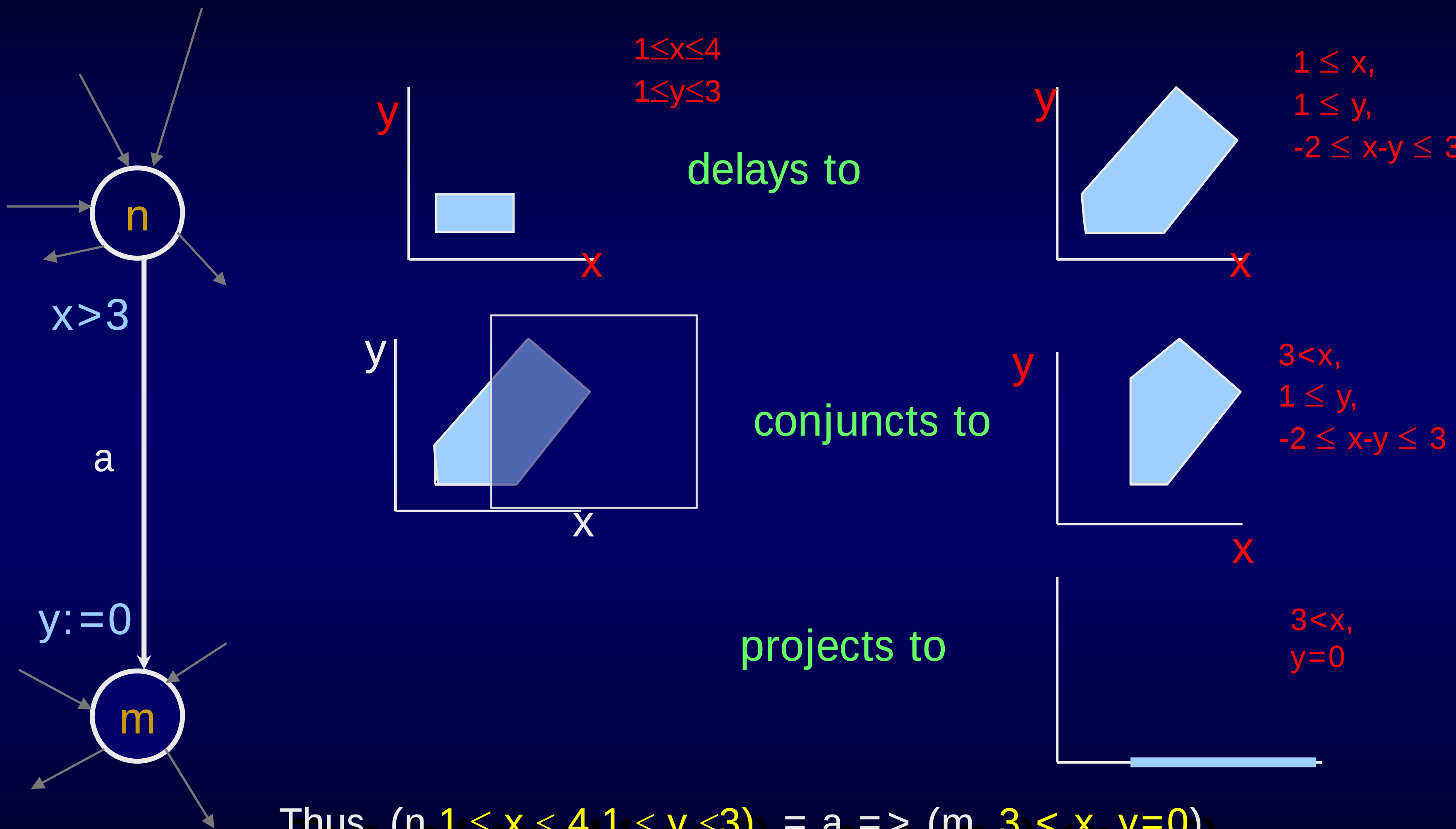


this is a time point



this is a time zone

Symbolic Transition



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

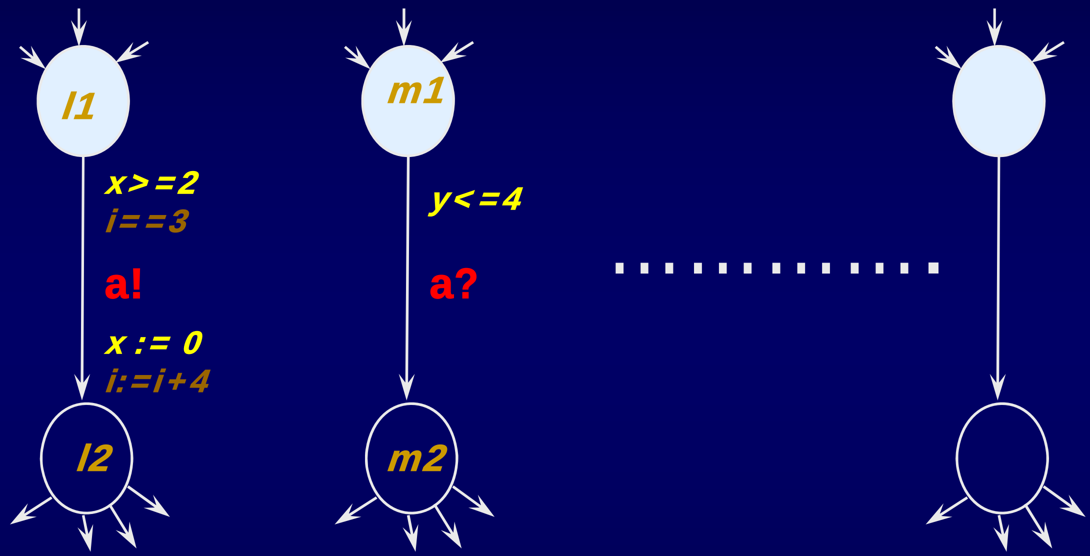
Finite symbolic simulation graph and reachable states can be computed

this is a symbolic transition (a bunch of concrete transitions)

Modelling Real-timeSystems using Uppaal

The Uppaal Model

= Networks of Timed Automata + Integer Variables + ….



Two-way synchronization on *complementary* actions.

Closed Systems!

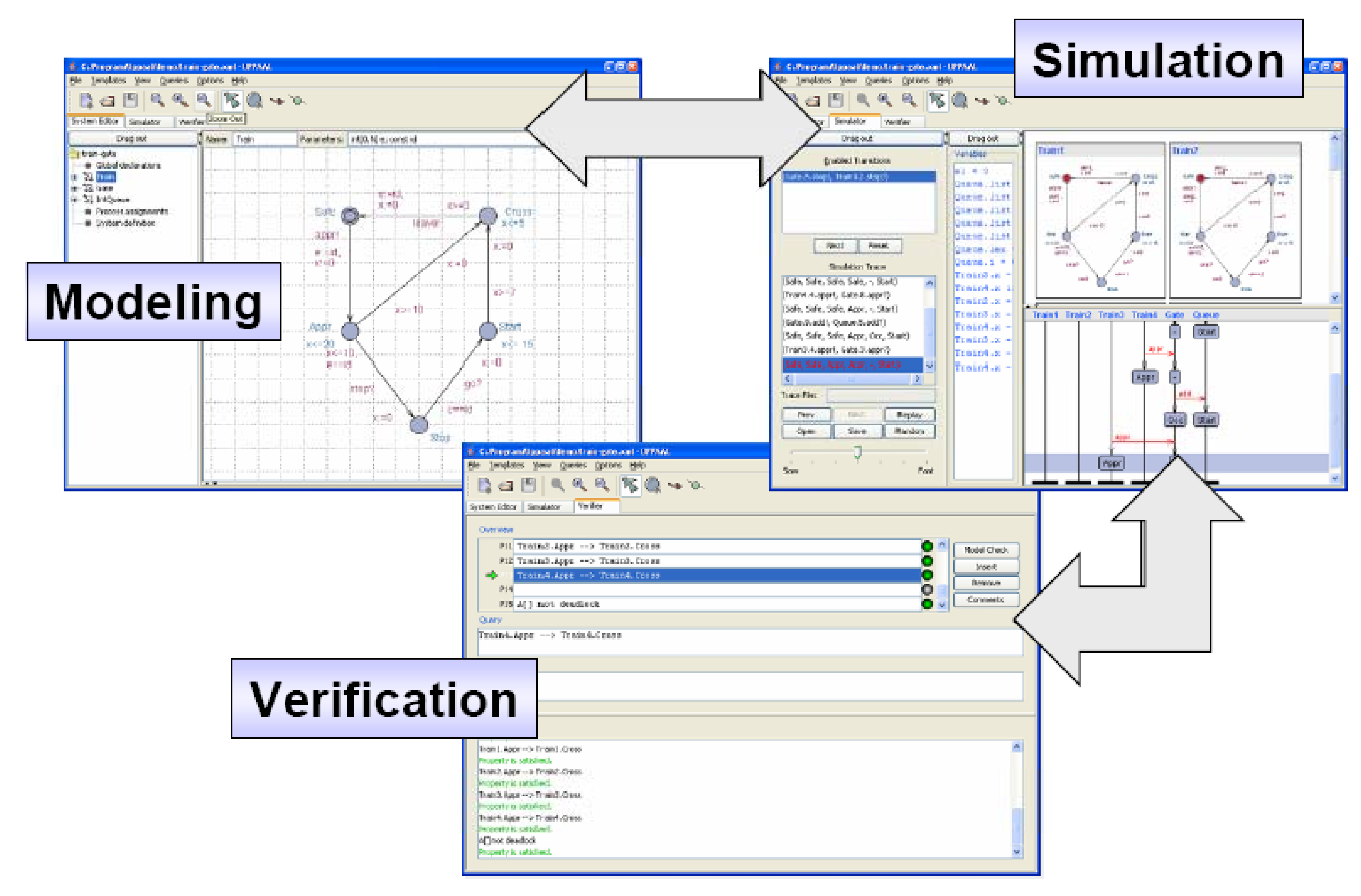
a! + a? --> tau (internal component interaction)

Example transitions

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

$\xrightarrow{0.2}$ (*l1*,*m1*,………,*x*=2.2, *y*=3.7, *i*=3,…..)

Modelling using Uppaal …



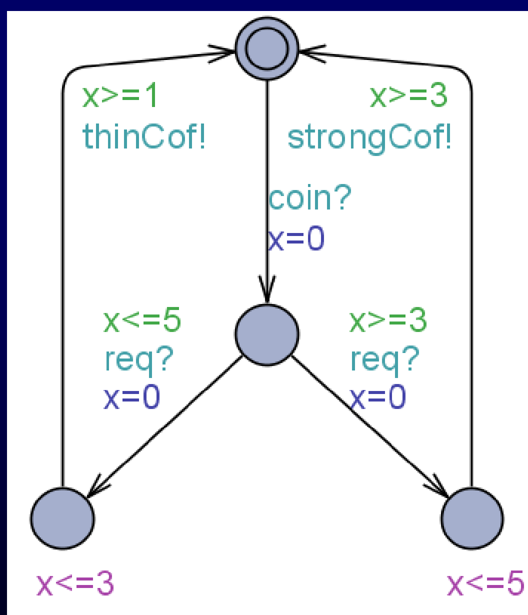
Timed Automaton of Coffee Machine

coin?

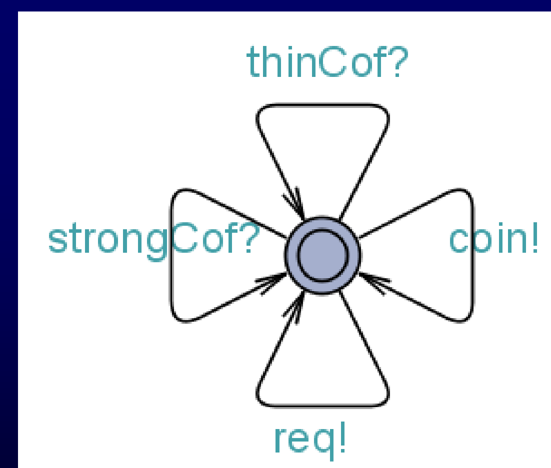
request?

thinCof!

strongCof!

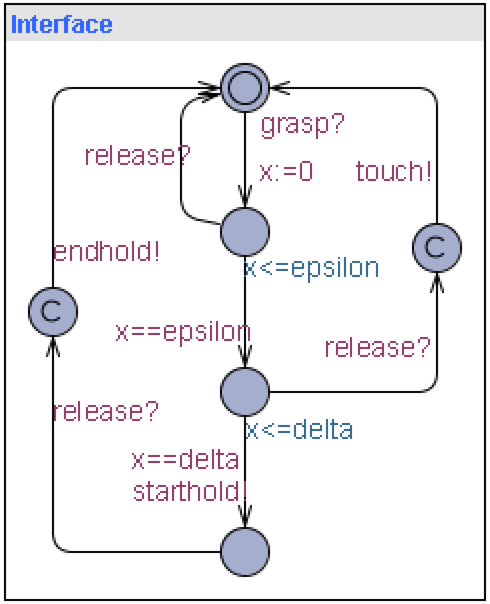


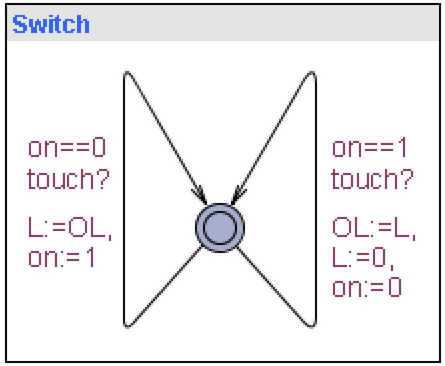
Machine Model

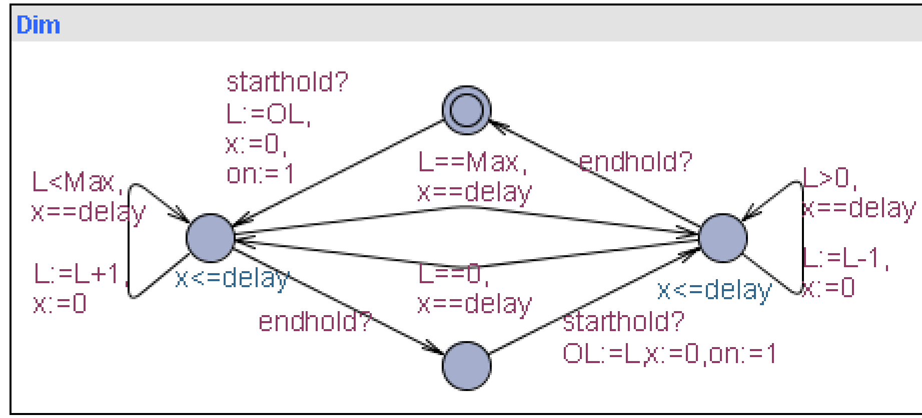


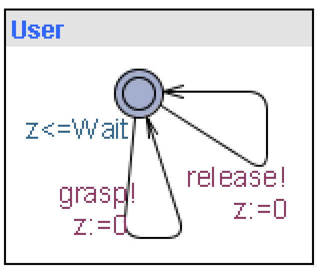
Possible users-model

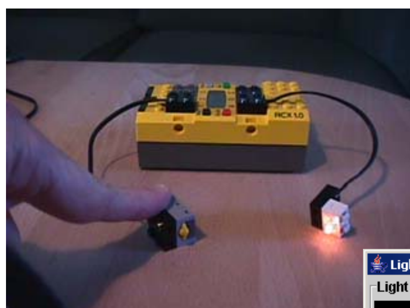
A Touch Sensative Light Controller

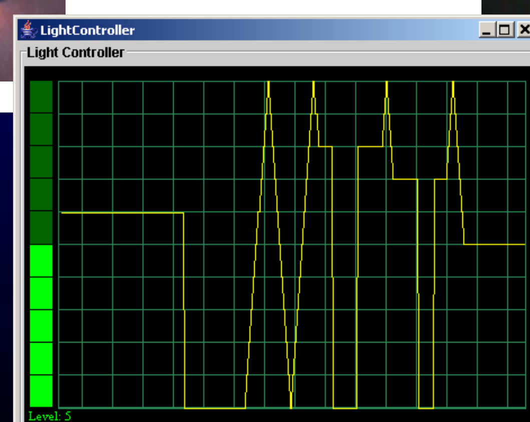








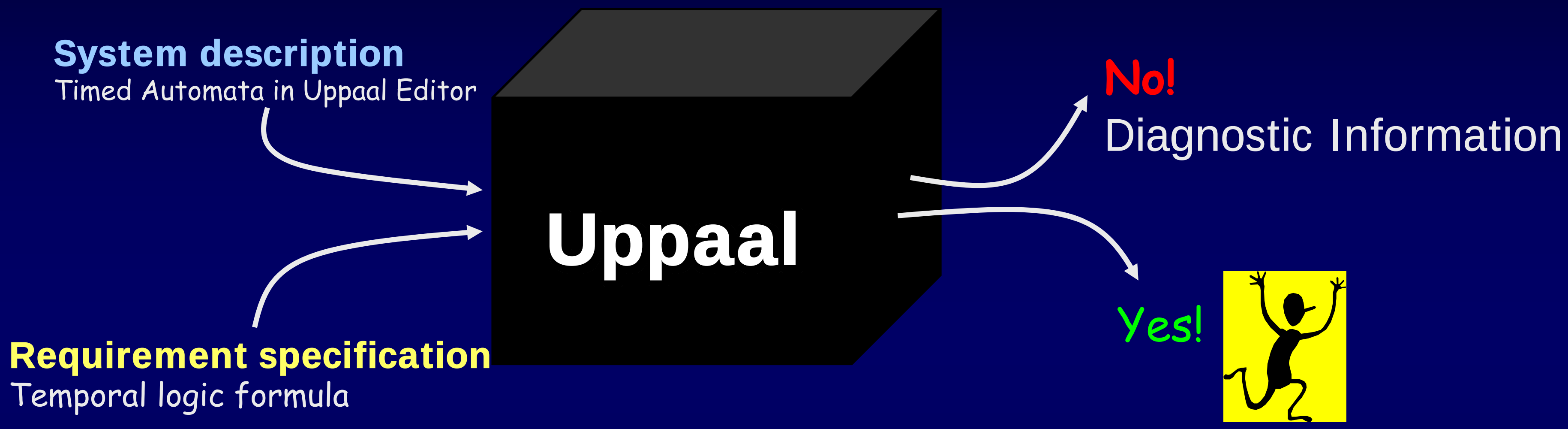


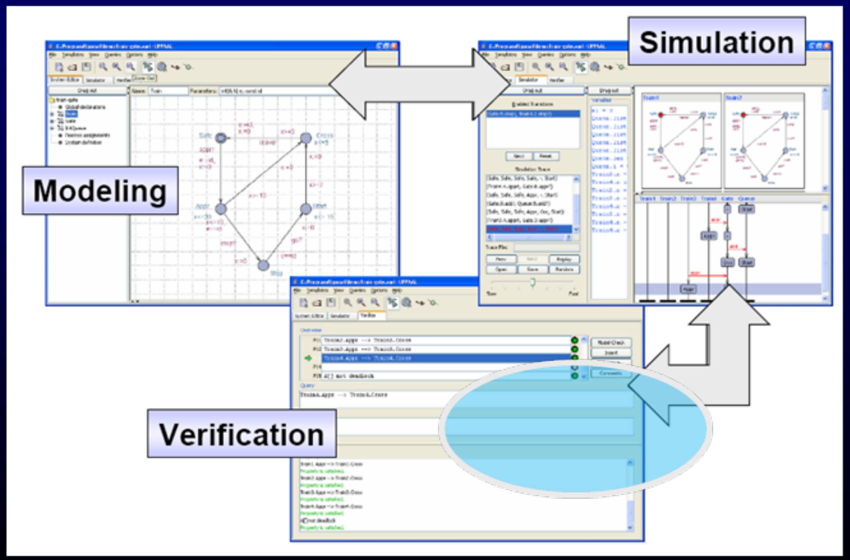


- Patient user: Wait=∞
- Impatient: Wait=15

Model Checking Real-time Systems

Uppaal as a box…





What does Verification do

- Compute *all* possible execution sequences
- And consequently to examine *all* states of the system
- *Symbolic approach to infinite state space exploration*
- Check if
  - every state encountered does not have the undesired property --> safety property
  - some state encountered has the desired property --> reachability property

Properties

- ***Safety***
  - Nothing bad happens during execution
  - System never enters a bad state
    - Eg. mutual exclusion on shared resource
- ***Liveness*** — diffent from reachability property
  - Something good eventually happens
  - Eventually reaching a desired state
    - Eg. a process' request for a shared resource is eventually granted

UPPAAL Property Specification Language

To quantitatively describe timing constraints.

- **A[] p**
- **A<> p**
- **E<> p**
- **E[] p**
- **P --> q**

"p leads to p":
A[ ] (p imply A< > q)

process location | data guards | clock guards

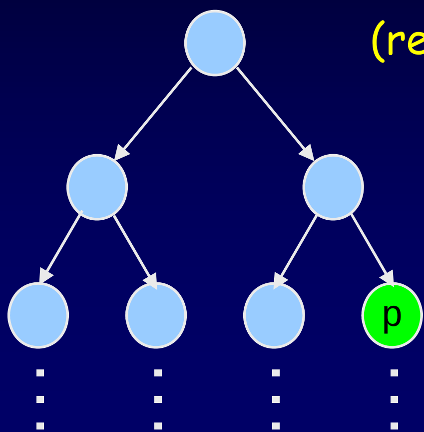
p::= a.l | g_d | g_c | p and p |
p or p | not p | p imply p |
(p) | deadlock(only for A[],E<>)

```
A[] (mc1.finished and mc2.finished) imply (accountA+accountB==200)
```

Uppaal "Computation Tree Logic"

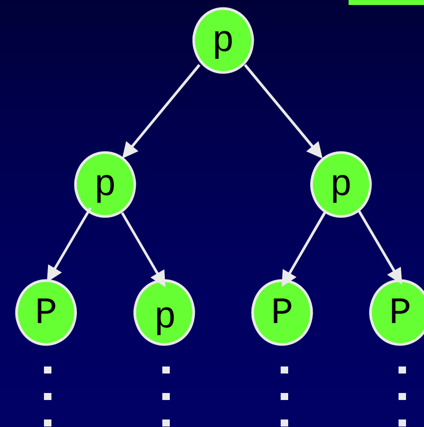
`E<> p` *Possible*

(reachability)

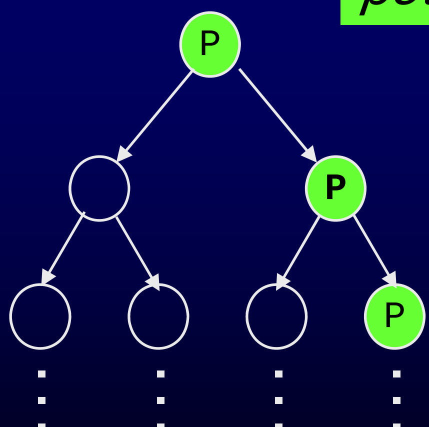


`A[] p` *always*

(safety)

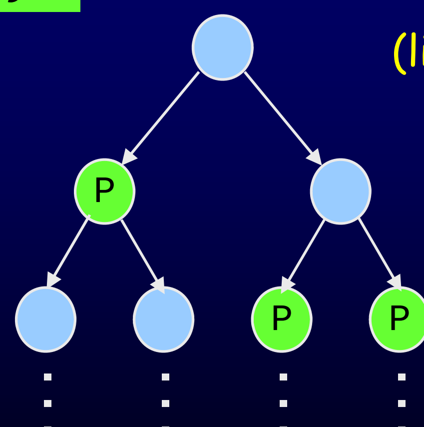


`E[] p` *potentially always*



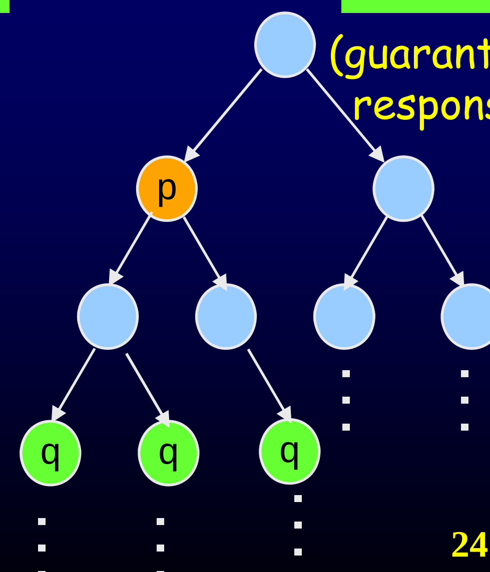
`A<> p` *inevitable*

(liveness)



`p --> q` *leads-to*

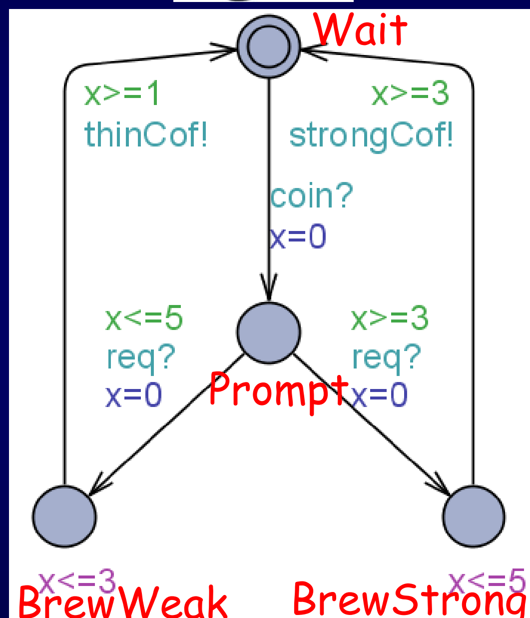
(guaranteed response)

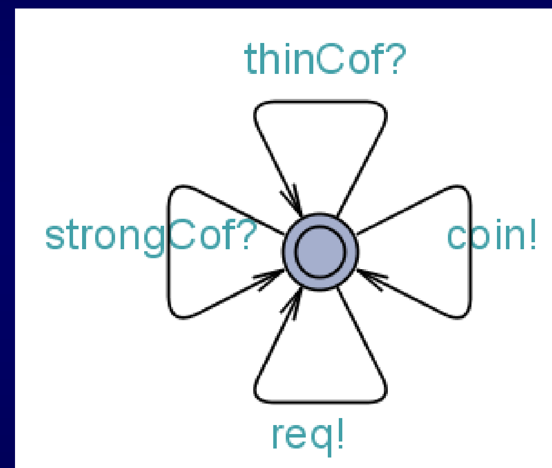


Example Properties

CVM

USER






```
E<> deadlock

E<> (x==2) && (CVM.BrewWeak)

A[] (x==6) imply (CVM.Wait || CVM.Prompt)
```