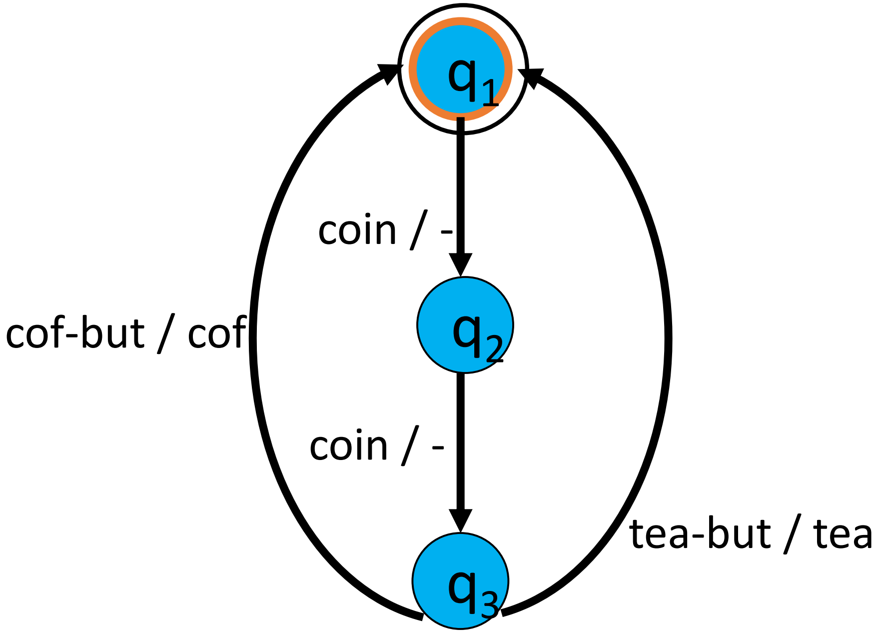
Lecture 6
Module I: Model Checking
Topic: Model checking timed transition systems: timed automata

J.Vain

10.03.2022

Slides by **Brian Nielsen**
(Aalborg Univ. Denmark)

Finite State Machine (Mealy)



condition		effect	
current state	input	output	next state
q_1	coin	-	q_2
q_2	coin	-	q_3
q_3	cof-but	cof	q_1
q_3	tea-but	tea	q_1

Inputs = {cof-but, tea-but, coin}
Outputs = {cof,tea}
States: $\{q_1,q_2,q_3\}$
Initial state = q_1
Transitions= {
$(q_1$, coin, -, $q_2)$,
$(q_2$, coin, -, $q_3)$,
$(q_3$, cof-but, cof, $q_1)$,
$(q_3$, tea-but, tea, $q_1)$
}

Sample run:

$$q_1 \xrightarrow{\text{coin/ -}} q_2 \xrightarrow{\text{coin/-}} q_3 \xrightarrow{\text{cof-but / cof}} q_1 \xrightarrow{\text{coin/ -}}$$

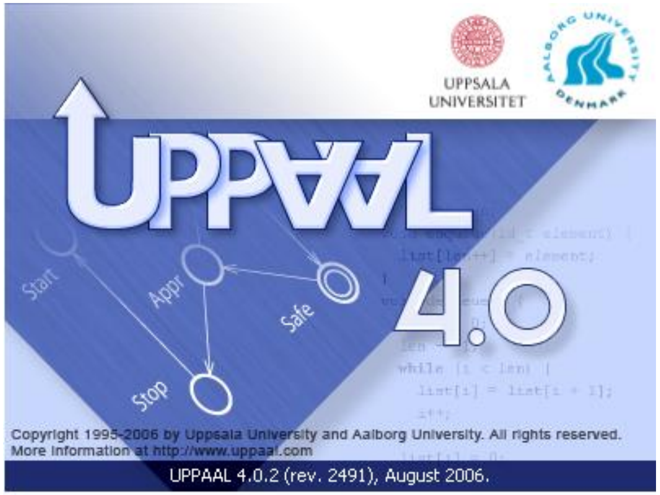
$$q_2 \xrightarrow{\text{coin/ -}} q_3 \xrightarrow{\text{cof-but / cof}} q_1$$

Adding Time

FSM

↓

Timed Automata



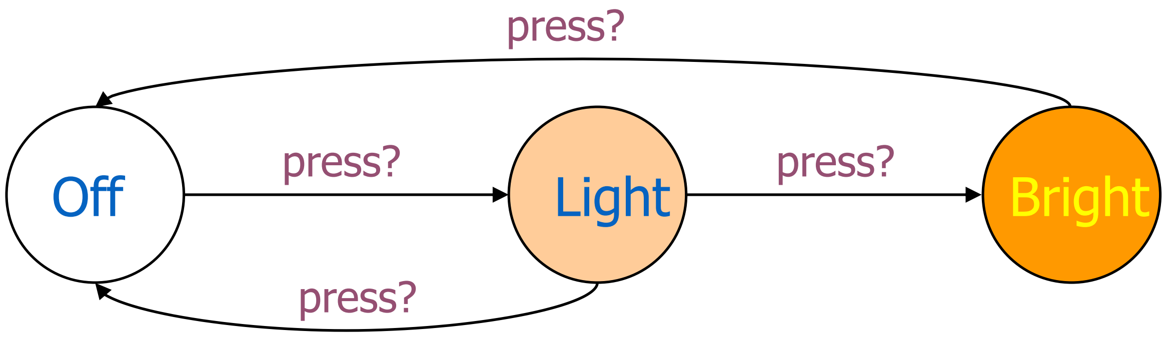
Copyright 1995-2006 by Uppsala University and Aalborg University. All rights reserved.
More information at http://www.uppaal.com

UPPAAL 4.0.2 (rev. 2491), August 2006.

Designing Dumb Light Control

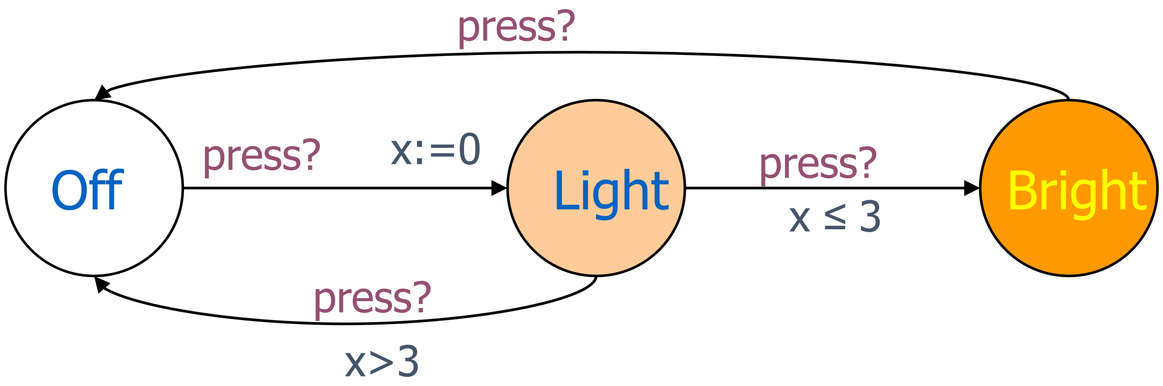
Reguirement: if press is issued twice quickly then the light will get brighter; otherwise the light is turned off.

Solution 1:

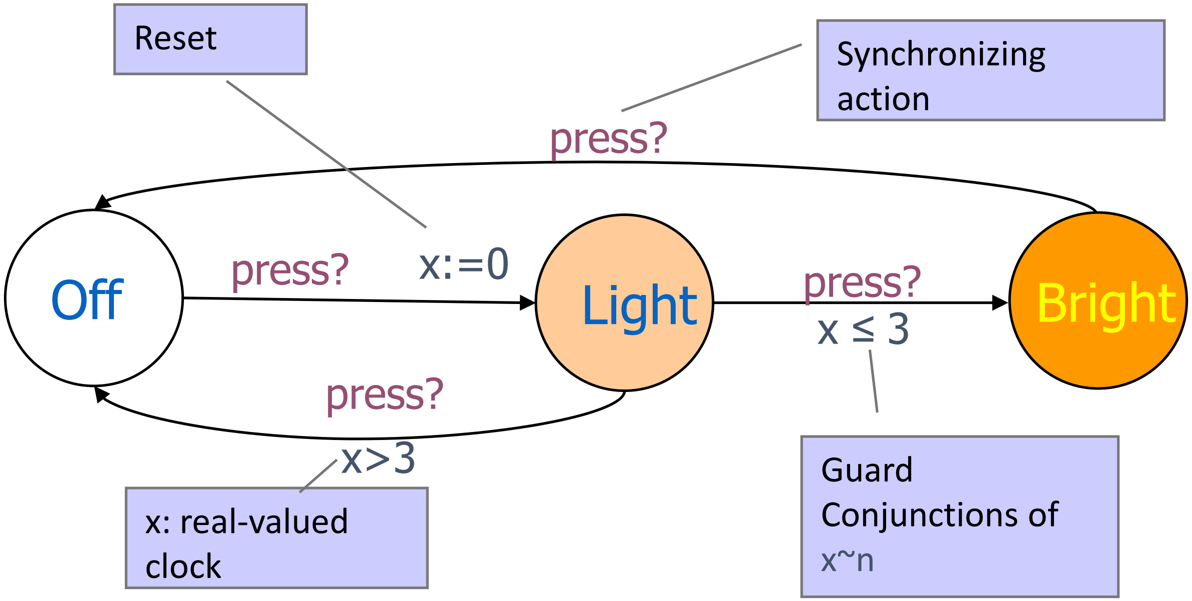


Dumb Light Control

Solution 2: Add real-valued clock x to model the timing requirement *quickly* ≡ $x \leq 3$



Timed Automata



Reset

Synchronizing action

x: real-valued clock

Guard Conjunctions of x~n

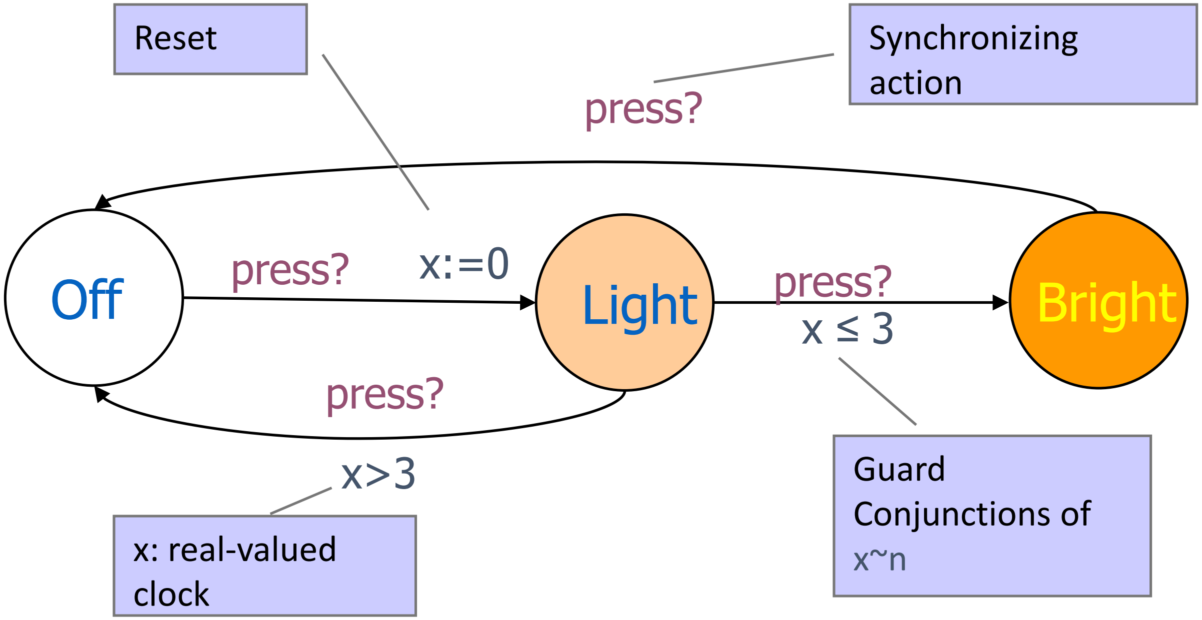
States:

(location , x=v) where v ∈ **R**

Transitions:

(Off , x=0)

Timed Automata



Reset

Synchronizing action

x: real-valued clock

Guard
Conjunctions of x~n

States:

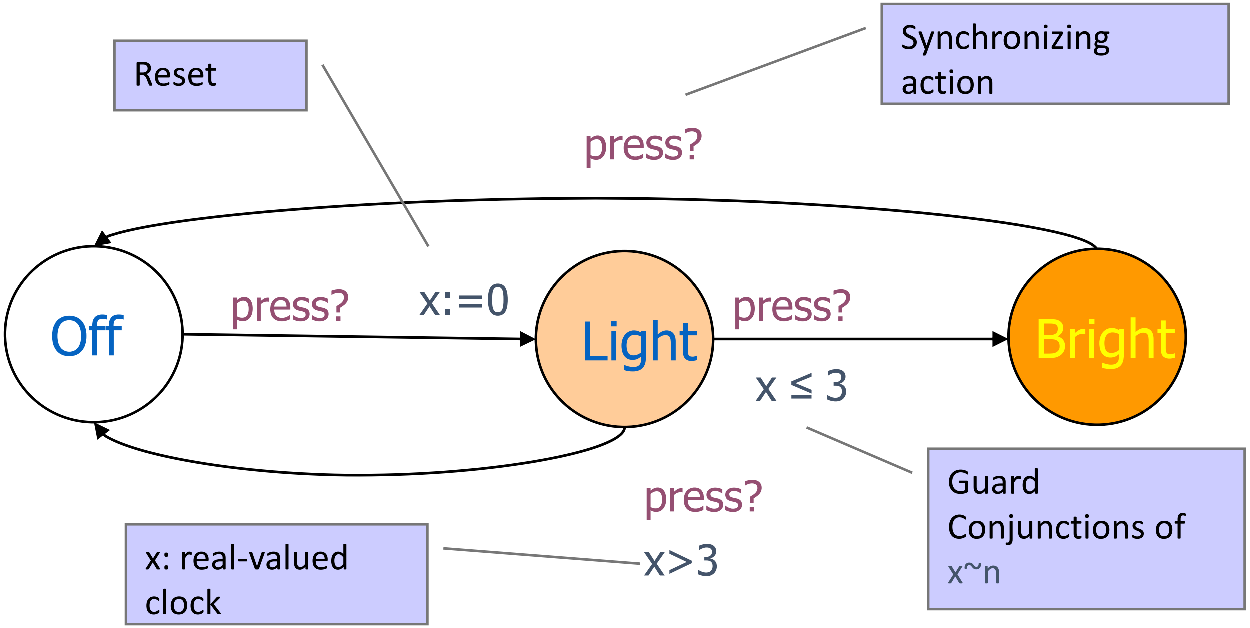
(location , x=v) where v ∈ **R**

Transitions:

(Off , x=0)

delay 4.32 → (Off , x=4.32)

Timed Automata



Reset

Synchronizing action

Guard
Conjunctions of x~n

x: real-valued clock

States:

(location , x=v) where v ∈ **R**

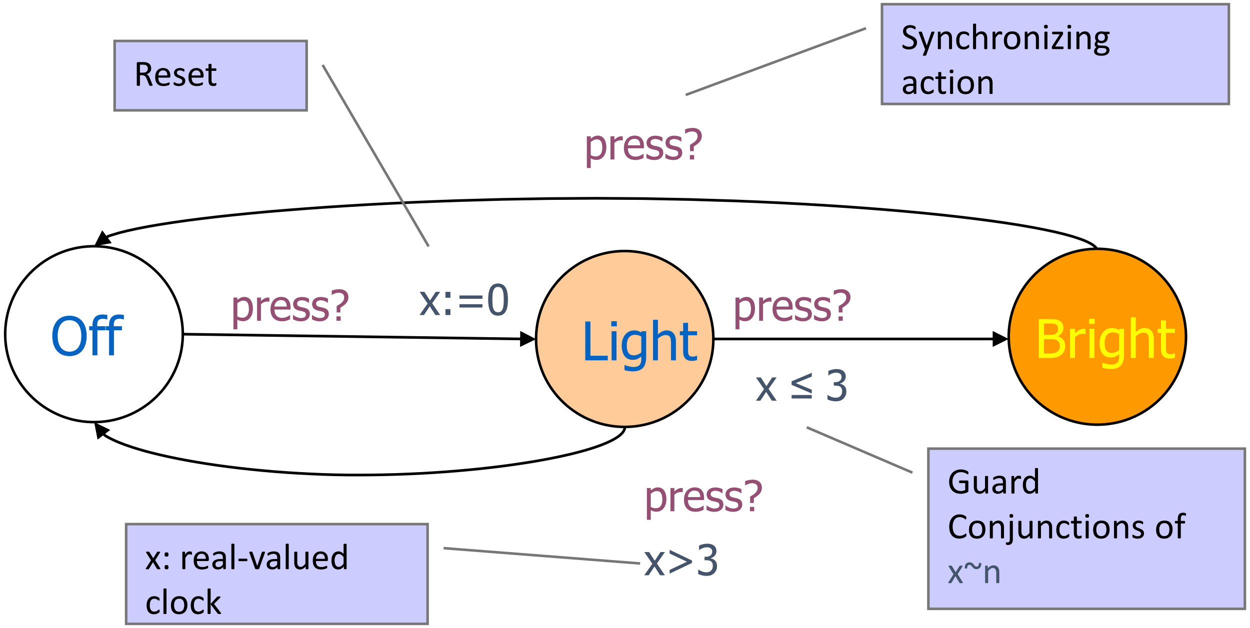
Transitions:

(Off , x=0)

delay 4.32 → (Off , x=4.32)

press? → (Light , x=0)

Timed Automata



Reset

Synchronizing action

Guard
Conjunctions of x~n

x: real-valued clock

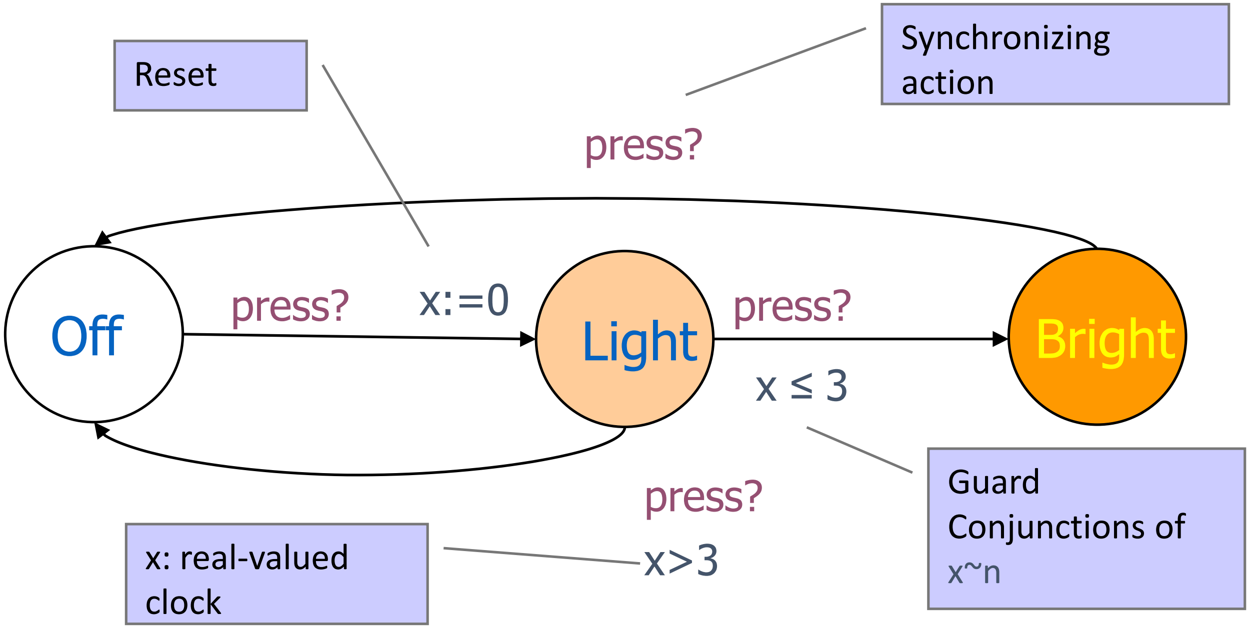
States:

(location , x=v) where v ∈ **R**

Transitions:

(Off , x=0)
delay 4.32 → (Off , x=4.32)
press? → (Light , x=0)
delay 2.51 → (Light , x=2.51)

Timed Automata



States:

(location , x=v) where v ∈ **R**

Transitions:

(Off , x=0)

delay 4.32 → (Off , x=4.32)

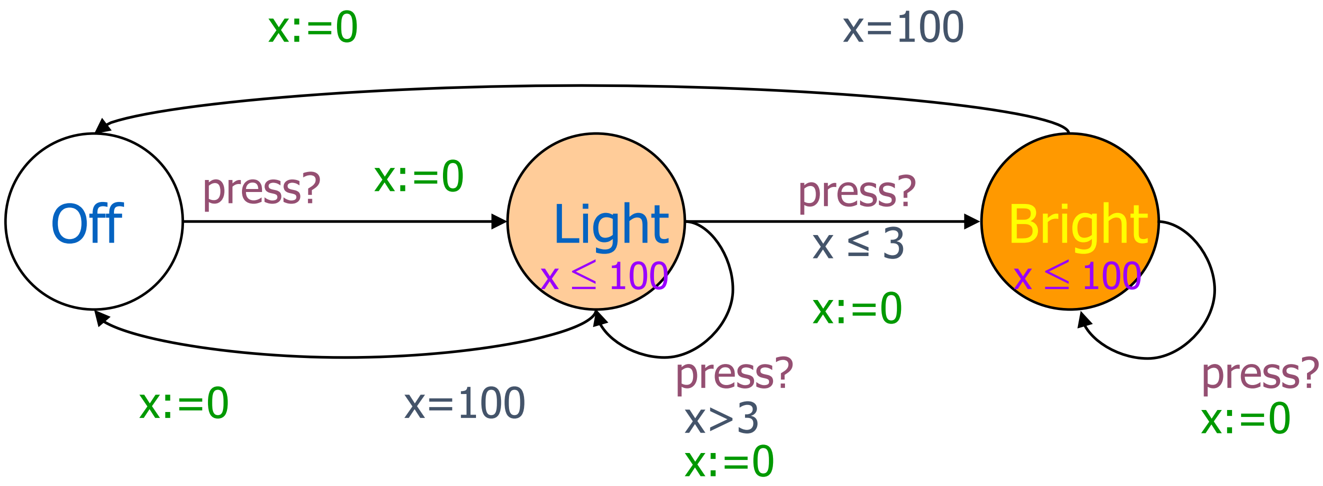
press? → (Light , x=0)

delay 2.51 → (Light , x=2.51)

press? → (Bright , x=2.51)

Intelligent Light Control

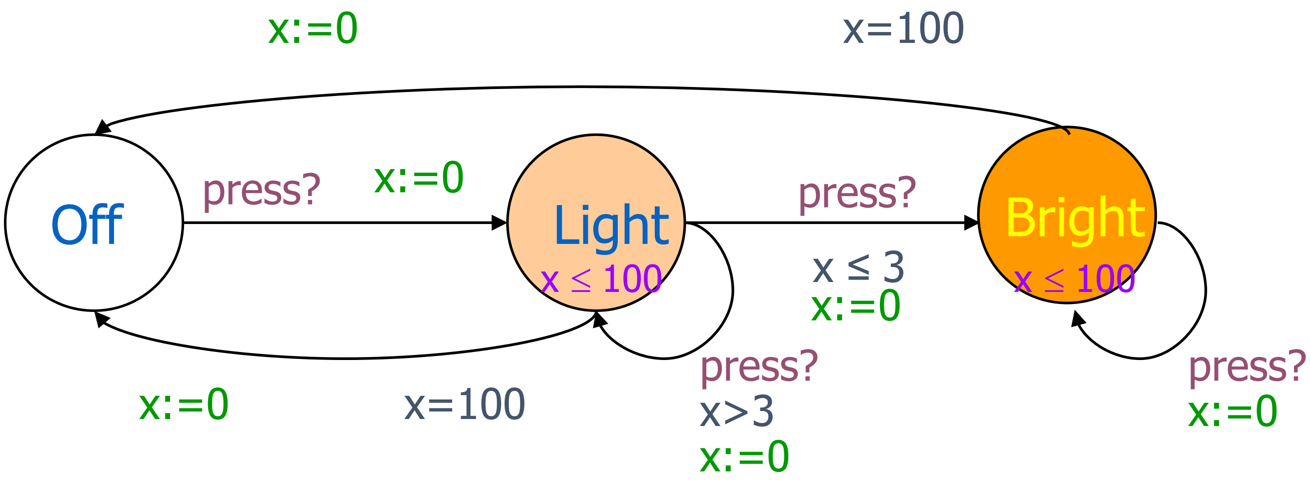
Reguirement: : automatically switch light off after 100 time units



Upper time bound is specified using invariants

Intelligent Light Control

Using Invariants



Transitions:

(Off , x=0)
delay 4.32 → (Off , x=4.32)
press? → (Light , x=0)
delay 4.51 → (Light , x=4.51)
press? → (Light , x=0)
delay 100 → (Light , x=100)
τ → (Off , x=0)

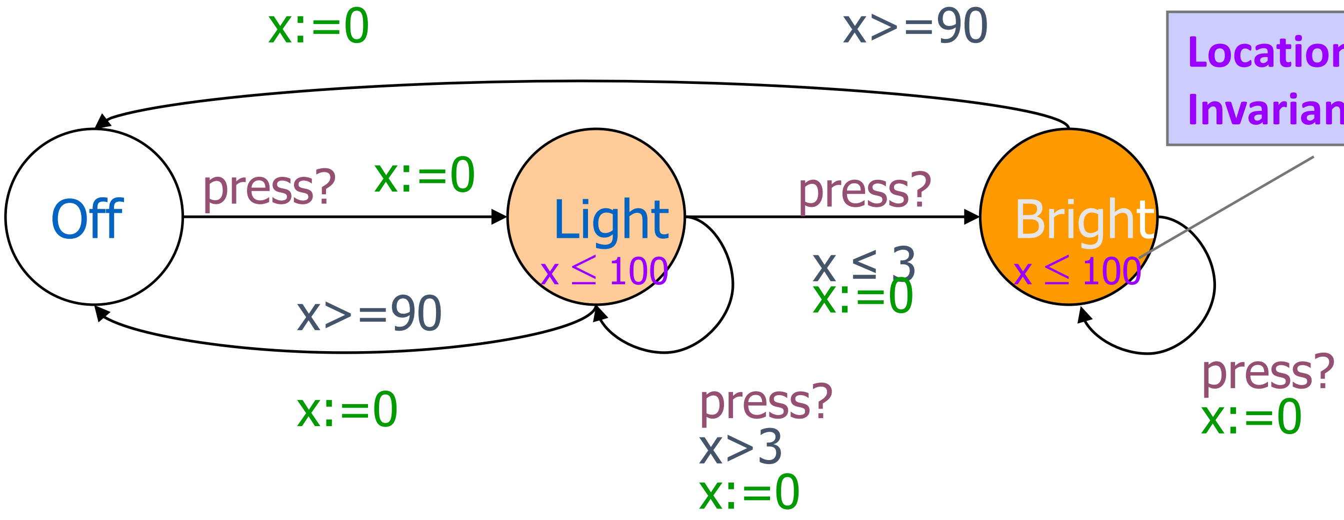
Note:
(Light , x=0) delay 103 → X

Invariants ensures progress

Intelligent Light Control

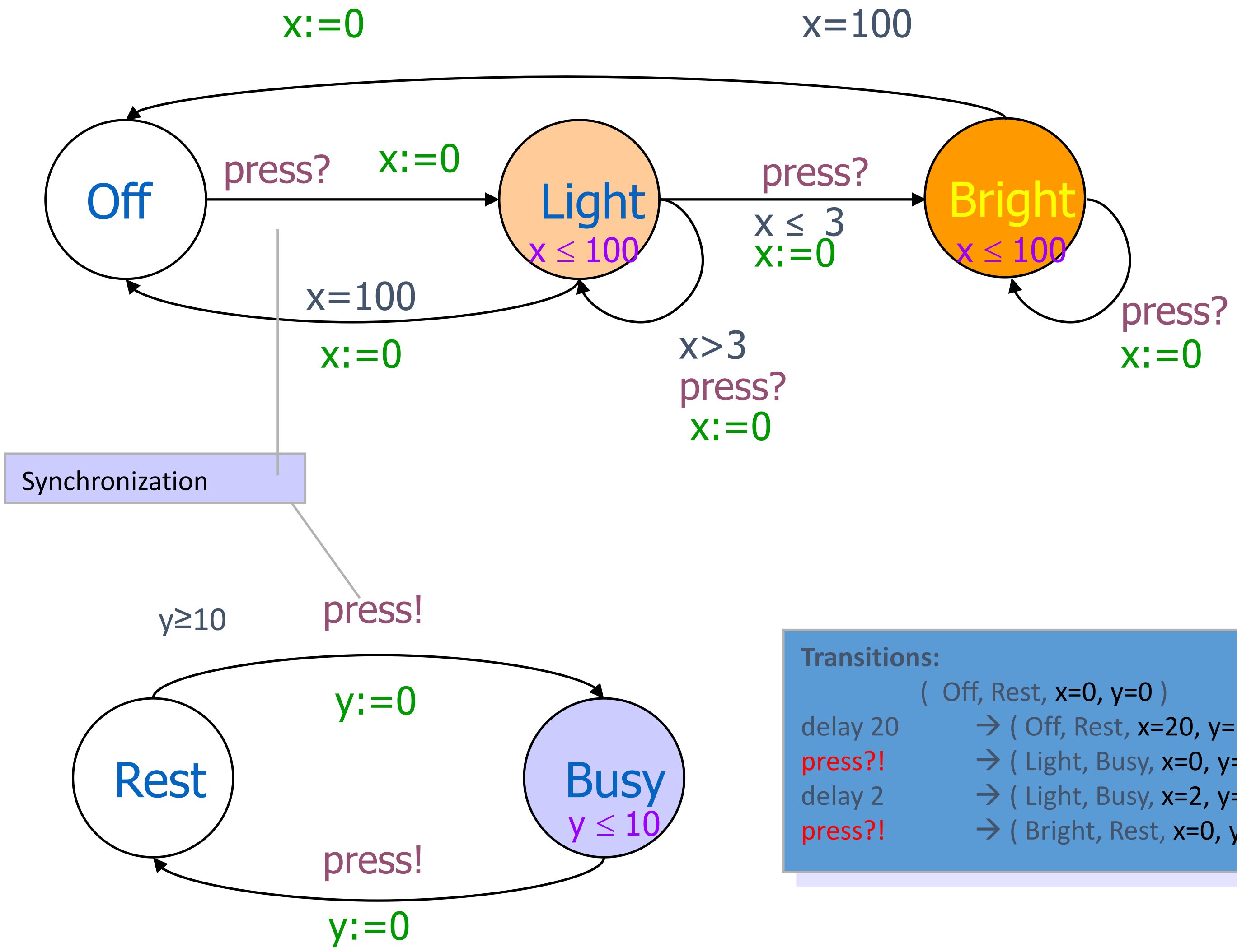
If requirements include **uncertainty**:

Automatically switch light off ***between*** 90-100 time units after switching on.



Location Invariant

Light Controller || User



Transitions:

(Off, Rest, x=0, y=0)

delay 20 → (Off, Rest, x=20, y=20)

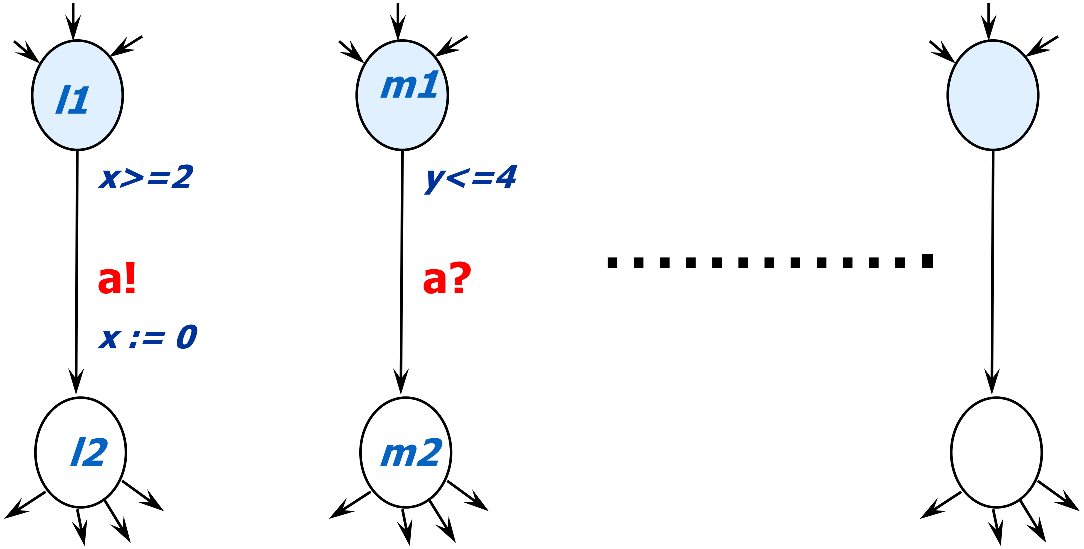
press?! → (Light, Busy, x=0, y=0)

delay 2 → (Light, Busy, x=2, y=2)

press?! → (Bright, Rest, x=0, y=0)

Networks of Timed Automata

(a'la CCS)



Two-way synchronization on *complementary* actions.

Closed Systems!

Example transitions

$(l1, m1, \ldots\ldots, x=2, y=3.5, \ldots\ldots) \xrightarrow{tau} (l2, m2, \ldots\ldots, x=0, y=3.5, \ldots\ldots)$

$\cancel{\xrightarrow{0.2}} (l1, m1, \ldots\ldots, x=2.2, y=3.7, \ldots\ldots)$

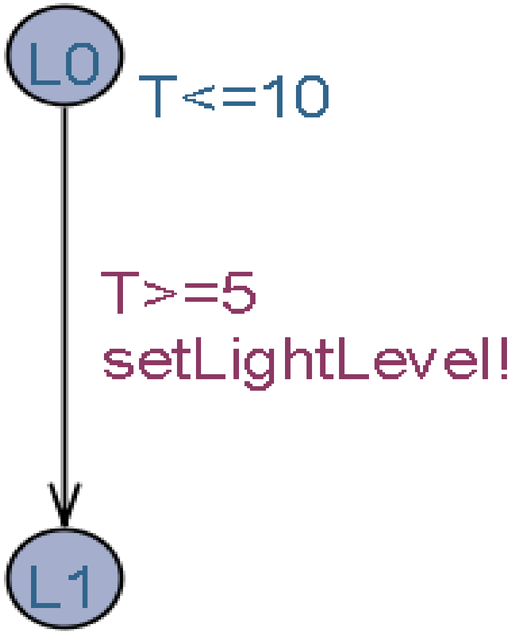
If **a** URGENT CHANNEL

How to model Timing Uncertainty?

- Unpredictable or variable timing
  - response time,
  - computation time
  - transmission time etc:

Example:
Light level must be adjusted
between 5 and 10 time units

Initially T=0

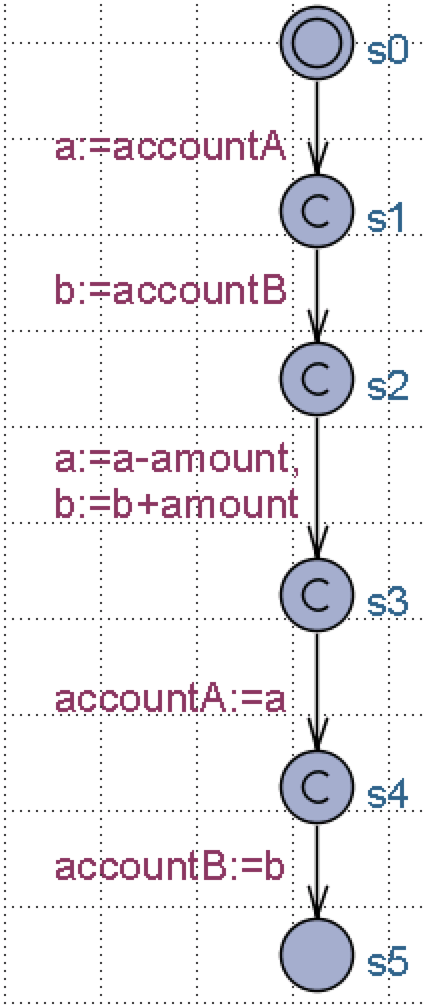


Comitted Locations

- Locations marked **C**
  - ***No delay*** in committed location.
  - No interleaving with parallel transitions
- Handy to model atomic sequences
- The use of committed locations reduces the number of states in a model, and allows for more space and time efficient analysis.

- Example:

Sequence s0 to s5 is executed atomically (no interleavings with concurrent actions)



Urgent Channels and Locations

- Locations marked **U**
  - ***No delay*** like in committed location.
  - But Interleaving permitted
- Channels declared “**`urgent chan`**”
  - Time doesn’t elapse when a synchronization is possible on a pair of urgent channels
  - Interleaving is allowed

Broadcast channel

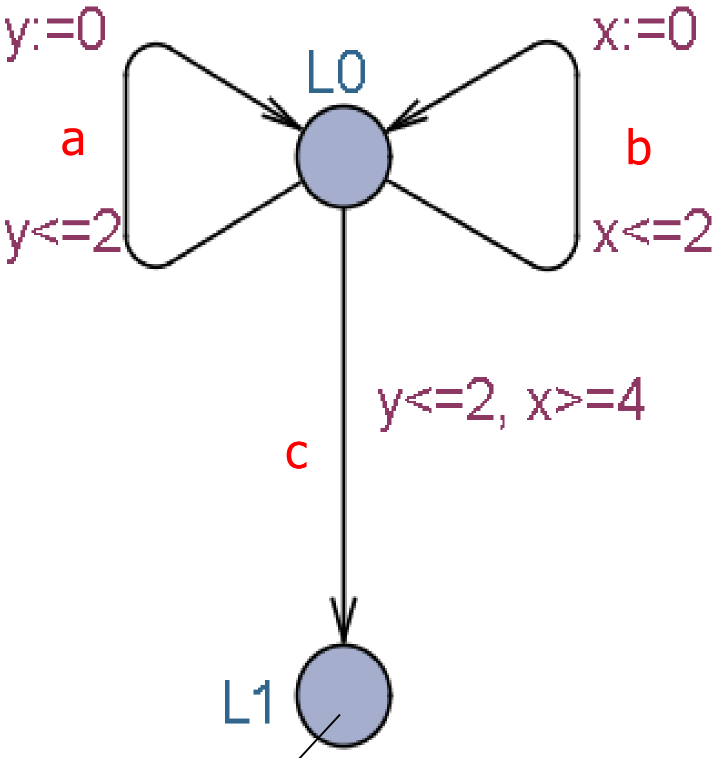
- Declaration: `broadcast chan  ch;`
- Sending process executes output action e.g. `ch!`
- Every automaton that listens to moves on synchronously

  ie. every automaton with enabled transition that is labeled with input action `ch?` moves on one step
- Provides non-blocking synchronization: even if zero listeners are enabled, sender process can progress anyway

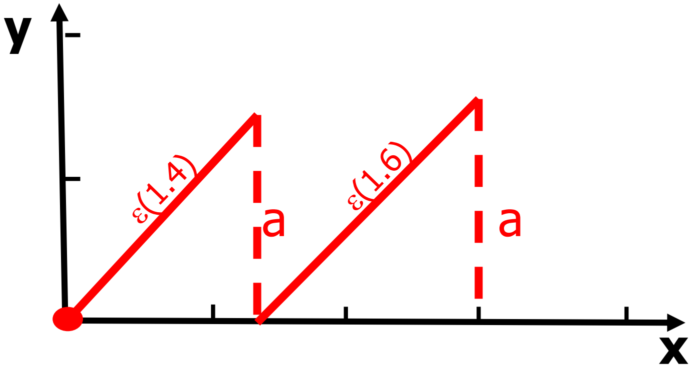
Other Uppaal features

- Bounded variable domains
  - `int [1..4] a;`
- C-like data-structures and user defined functions in declaration section
  - structs, arrays, and type definitions
    - `typedef int [0,n] t_id`
- non-deterministic assignment:
    - `select a:T        // select a random value from T`
- `forall`, `exists` in expressions
- Scalar sets (for giving unique ID’s)
- Process and channel **priorities**

Semantics: Timed traces



Reachable?



(L0, x=0,y=0)

$\rightarrow_{\varepsilon(1.4)}$

(L0, x=1.4,y=1.4)

$\rightarrow_{a}$

(L0, x=1.4,y=0)

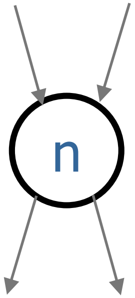
$\rightarrow_{\varepsilon(1.6)}$

(L0, x=3.0,y=1.6)

$\rightarrow_{a}$

(L0,x=3.0,y=0)

From explicit clock values to zones
(*from infinite to finite*)

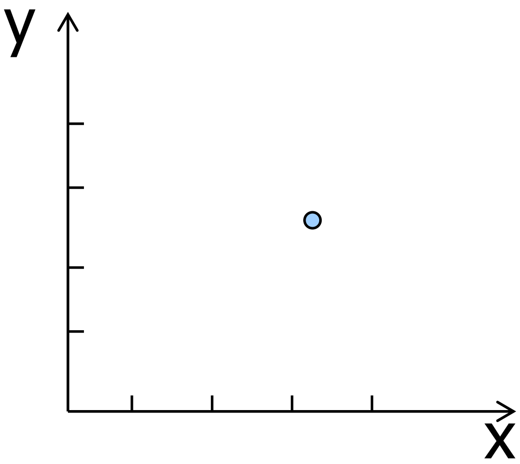


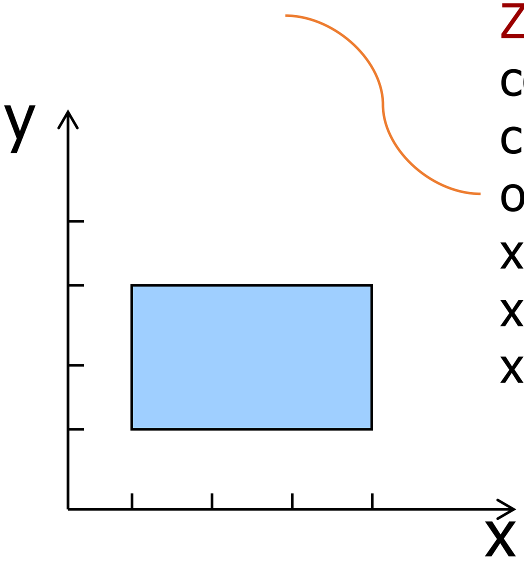
Explicit state

(n, x=3.2, y=2.5)

Symbolic state (set)

(n, 1 ≤ x ≤ 4, 1 ≤ y ≤ 3)

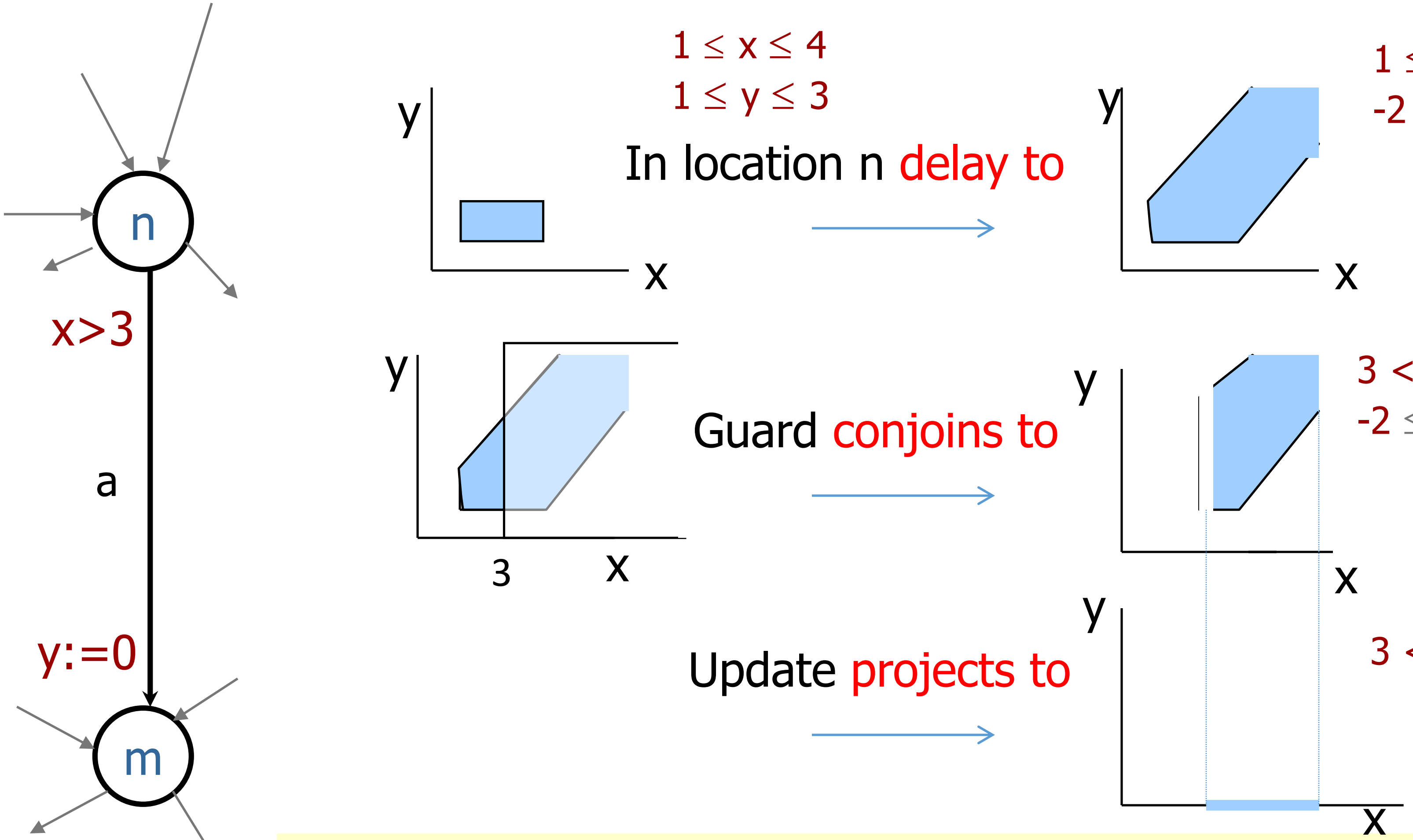




Zone:
conjunction of clock constraints of form:
x-y<=const1,
x<=const2,
x>=const3

Symbolic Transitions

Assume clock values when reaching location n



$1 \leq x \leq 4$
$1 \leq y \leq 3$

In location n delay to →

$1 \leq x$, $1 \leq y$
$-2 \leq x-y \leq 3$

Guard conjoins to →

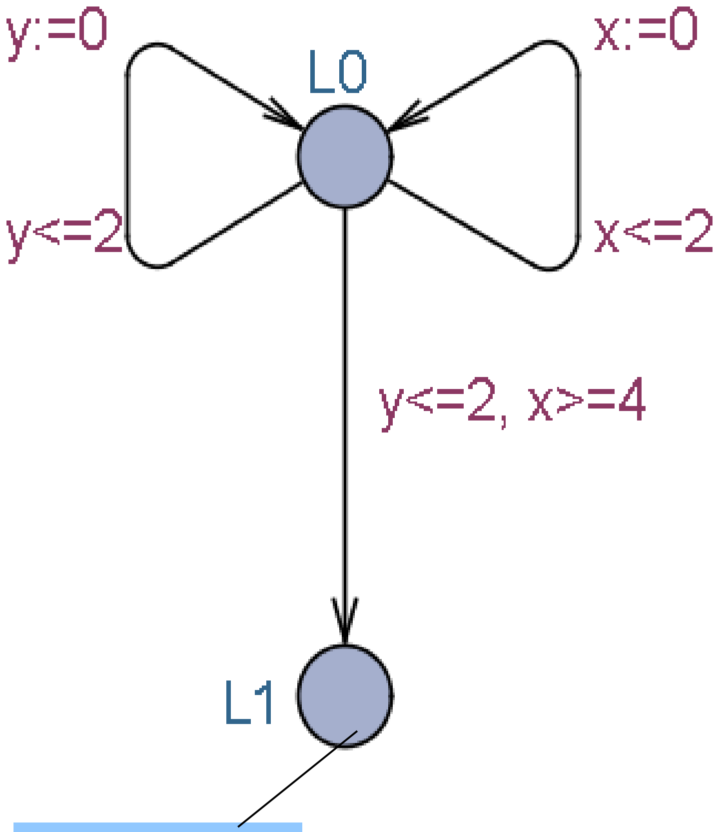
$3 < x$, $1 \leq y$
$-2 \leq x-y \leq 3$

Update projects to →

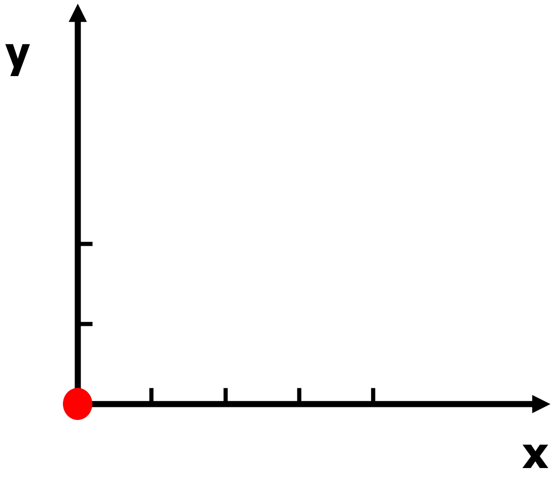
$3 < x$, $y=0$

Thus (n, $1 \leq x \leq 4$, $1 \leq y \leq 3$) $\rightarrow^{a}$ (m, $3 < x$, $y=0$)

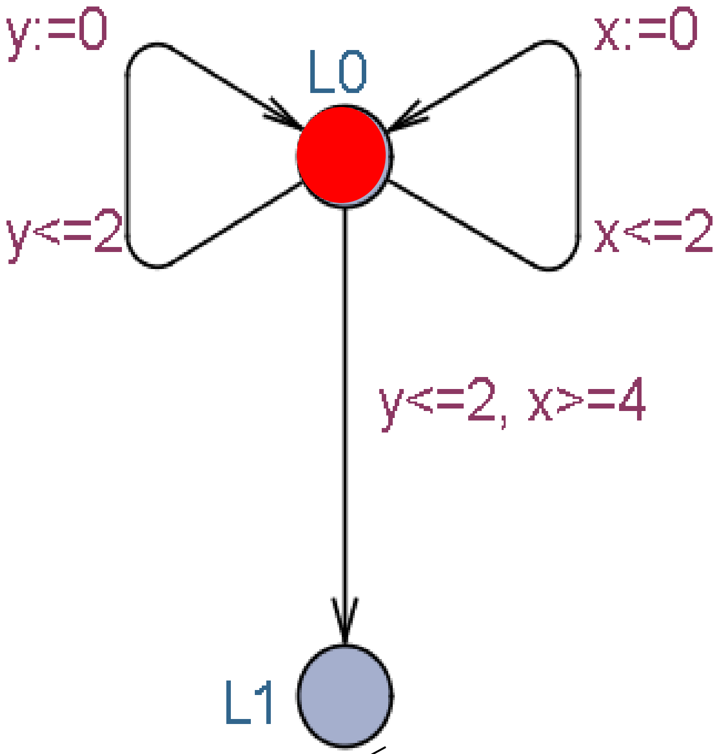
Symbolic state exploration



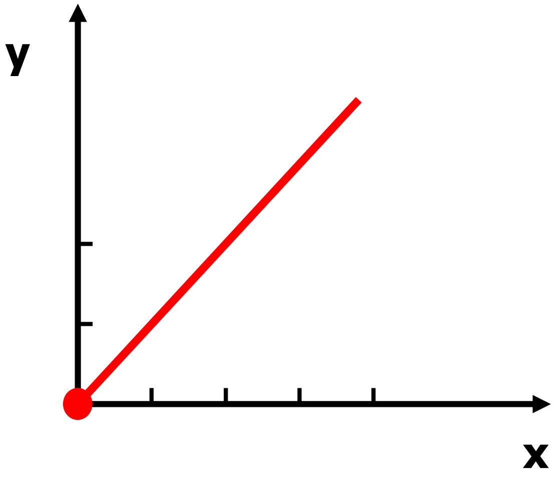
Reachable?



Symbolic Exploration

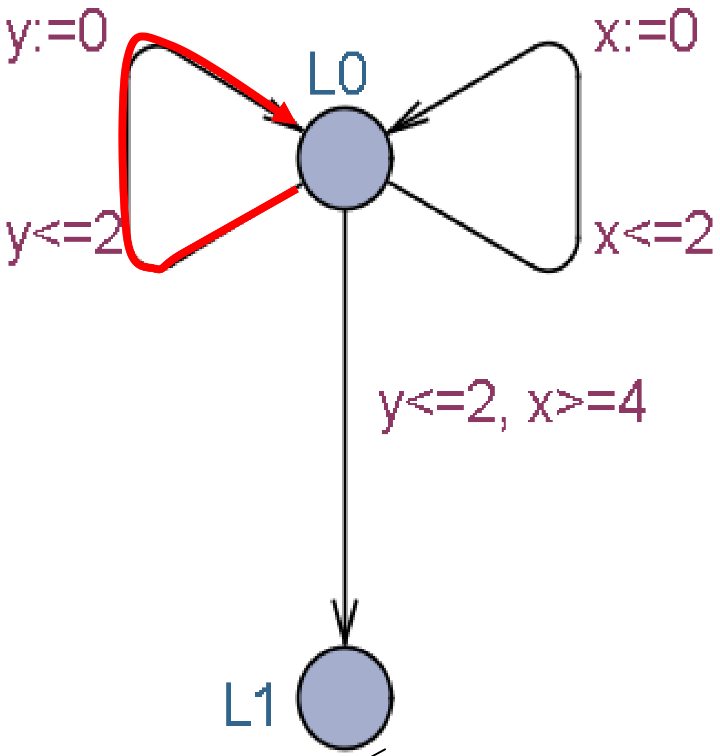


Reachable?

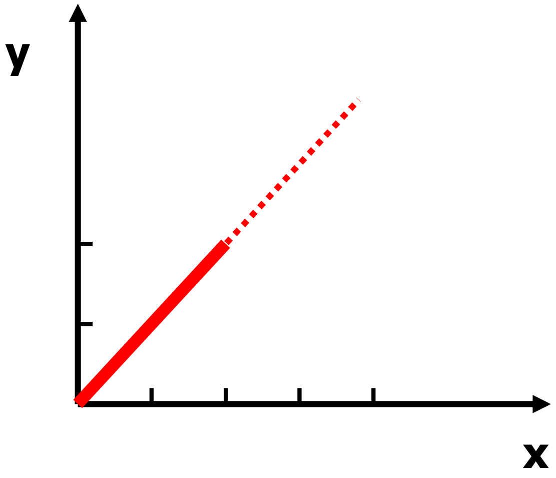


Delay

Symbolic Exploration

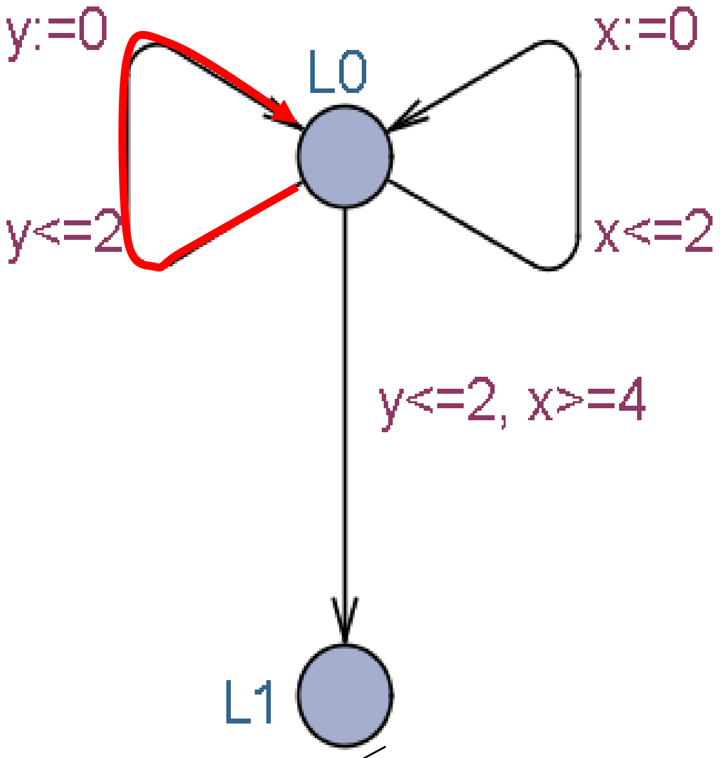


Reachable?

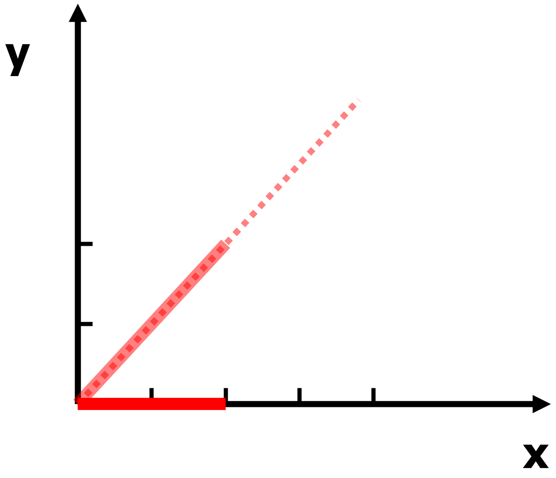


Left

Symbolic Exploration

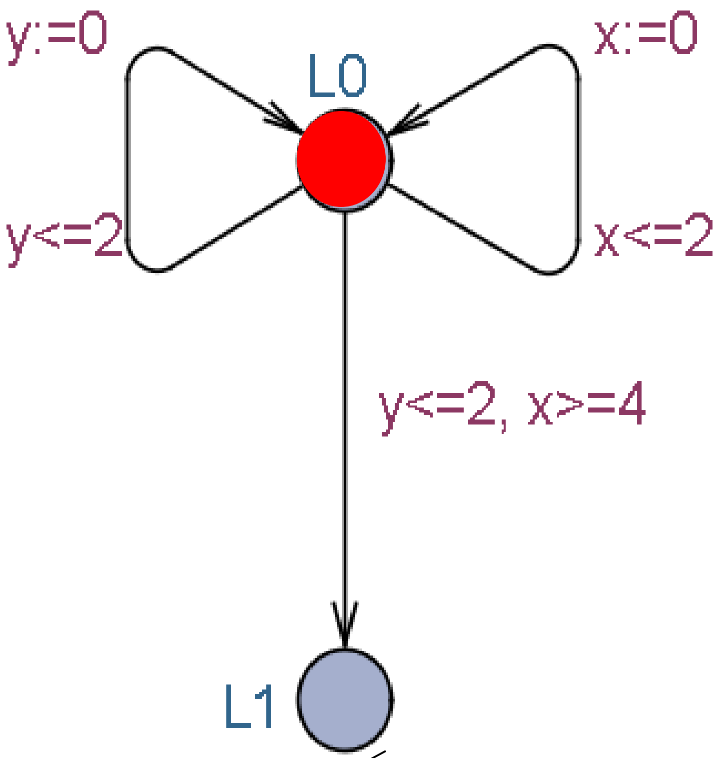


Reachable?

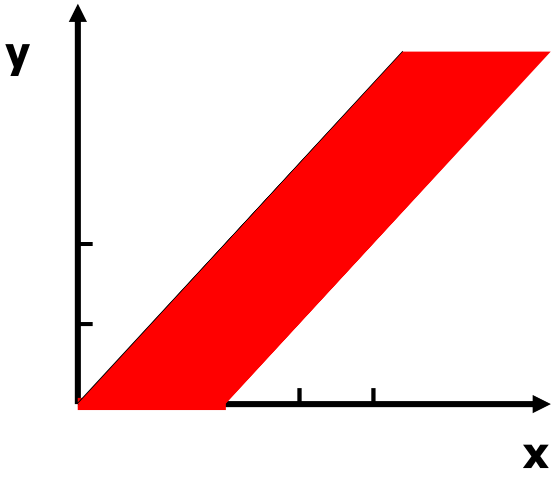


Left

Symbolic Exploration

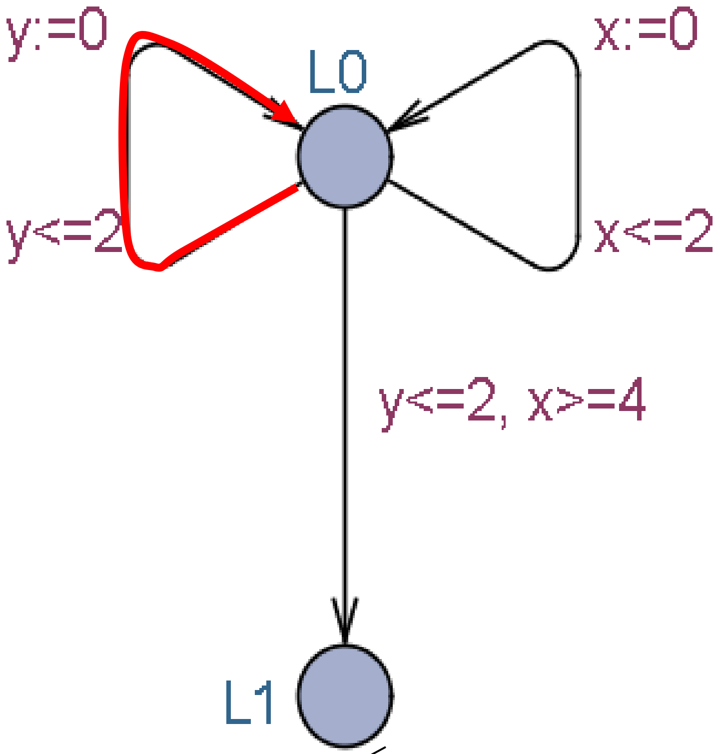


Reachable?

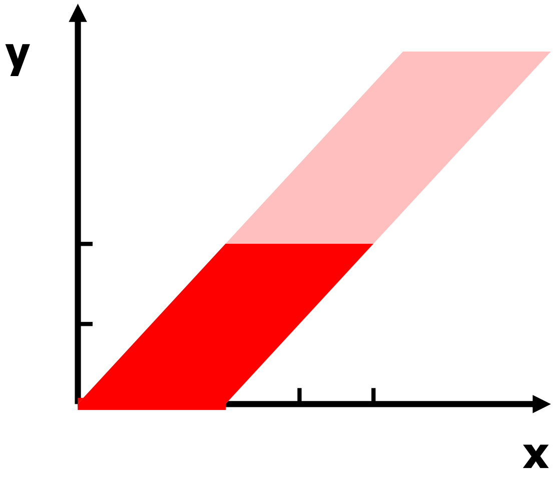


Delay

Symbolic Exploration

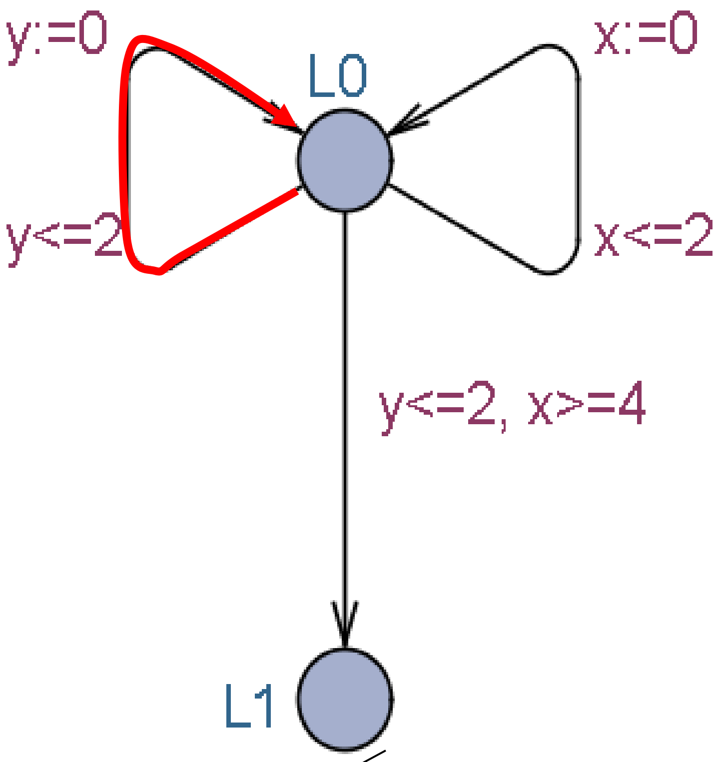


Reachable?

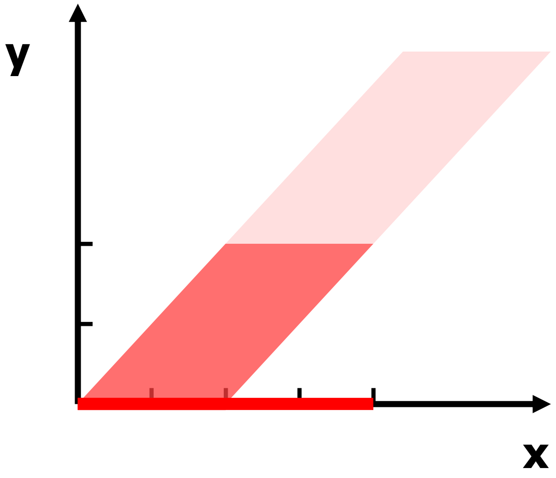


Left

Symbolic Exploration

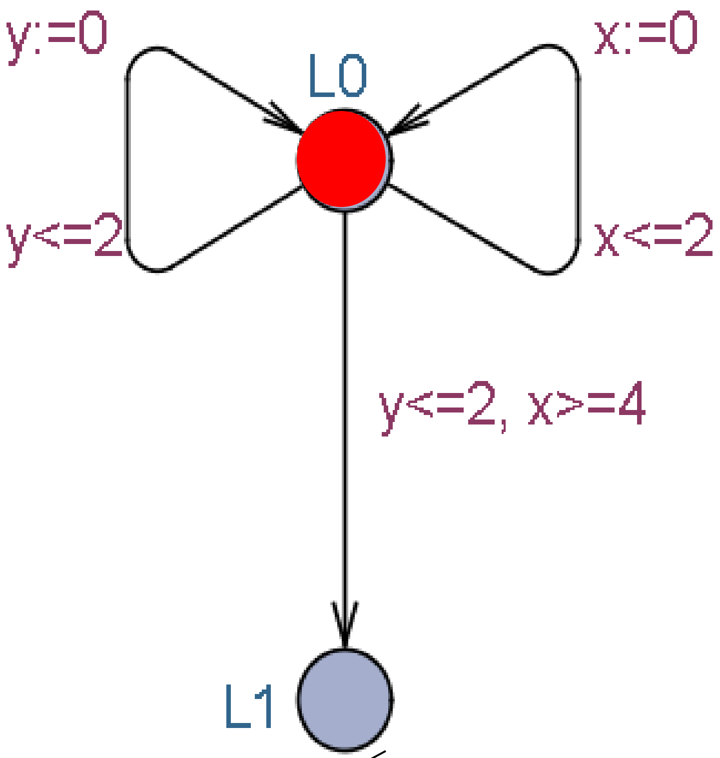


Reachable?

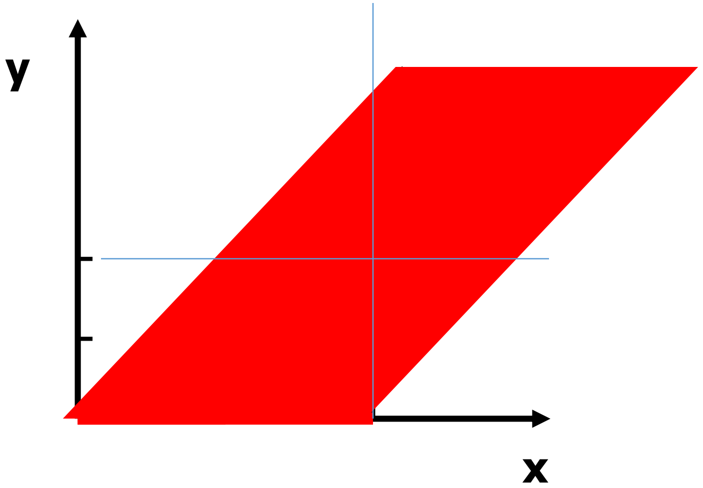


Left

Symbolic Exploration

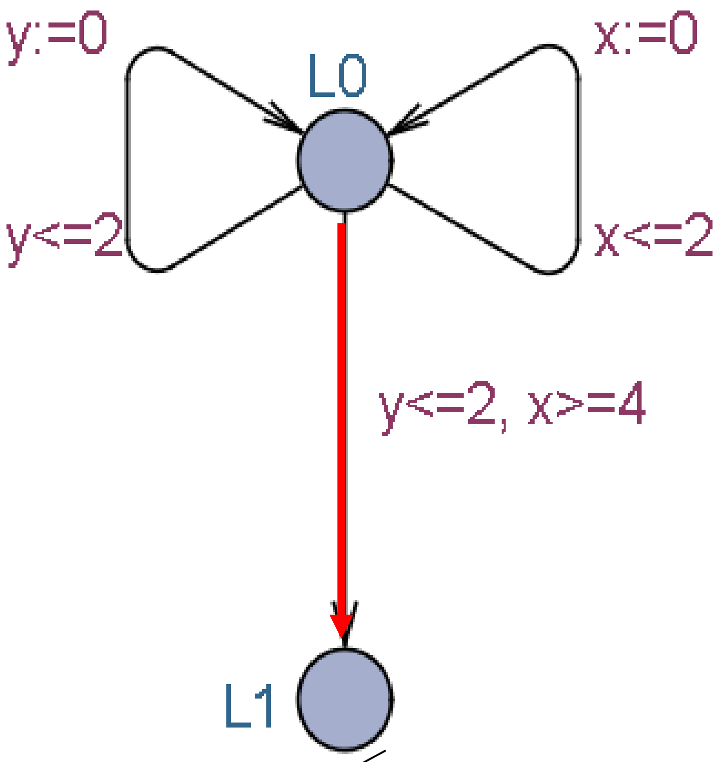


Reachable?

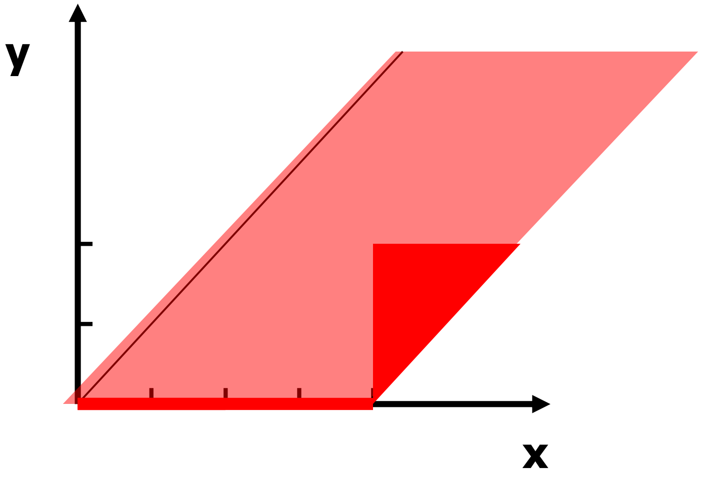


Delay

Symbolic Exploration



Reachable?

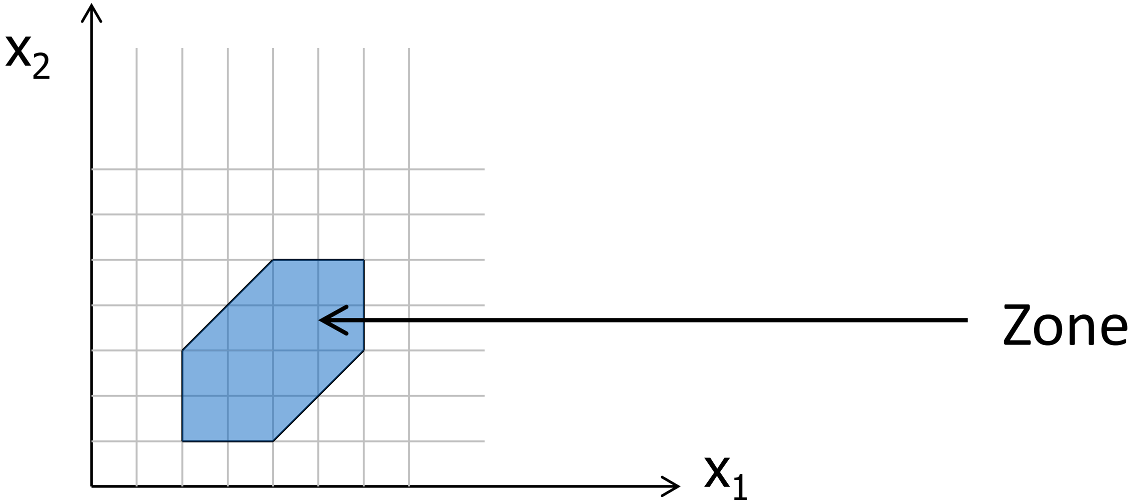


Down

Difference Bound Matrix

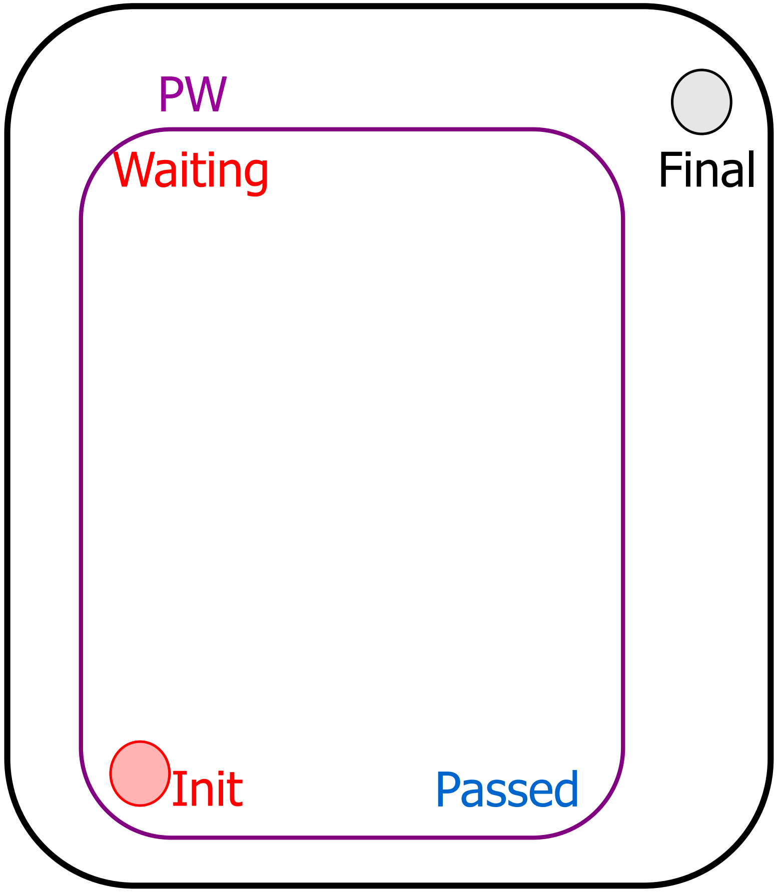
$x_0-x_0<=0$	$x_0-x_1<=-2$	$x_0-x_2<=-1$
$x_1-x_0<=6$	$x_1-x_1<=0$	$x_1-x_2<=3$
$x_2-x_0<=5$	$x_2-x_1<=1$	$x_2-x_2<=0$

$$x_i-x_j<=c_{ij}$$



Forward Reachability Algorithm

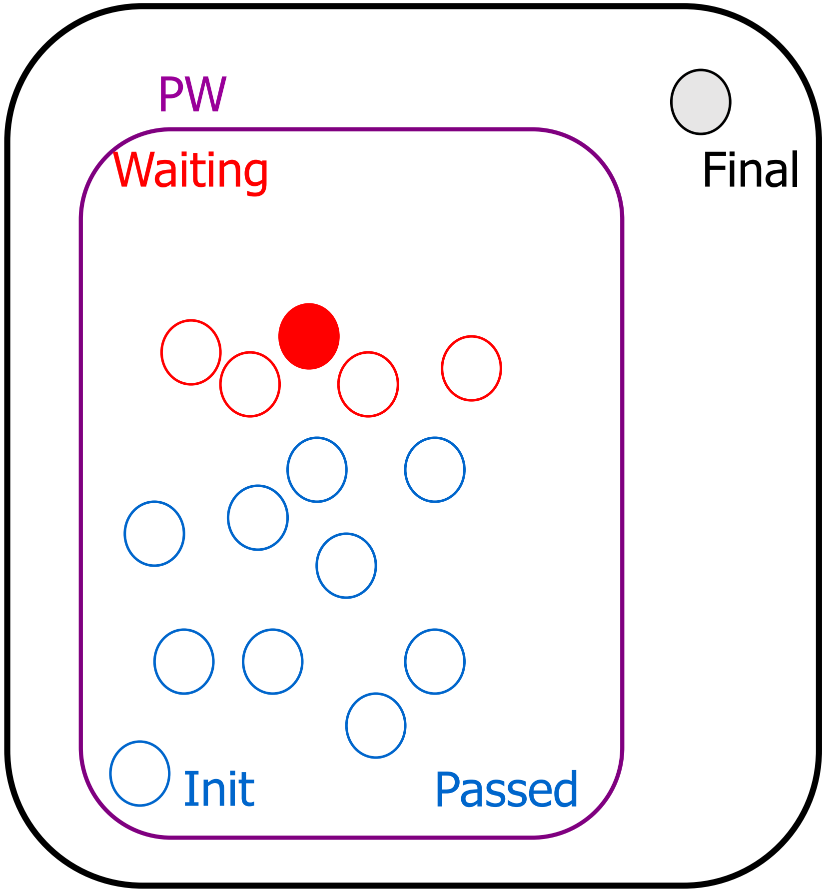
Init -> Final ?



```
INITIAL  Passed := Ø;
         Waiting := {($n_0$,$Z_0$)}

REPEAT
   pick (n,Z) in Waiting
   if (n,Z) = Final return true
   for all $(n,Z) \rightarrow (n',Z')$:
      if for some (n',Z'') in Passed $Z' \subseteq Z''$
      then continue
      else add (n',Z') to Waiting
      move (n,Z) to Passed

UNTIL  Waiting = Ø
return false
```

Forward Reachability Algorithm

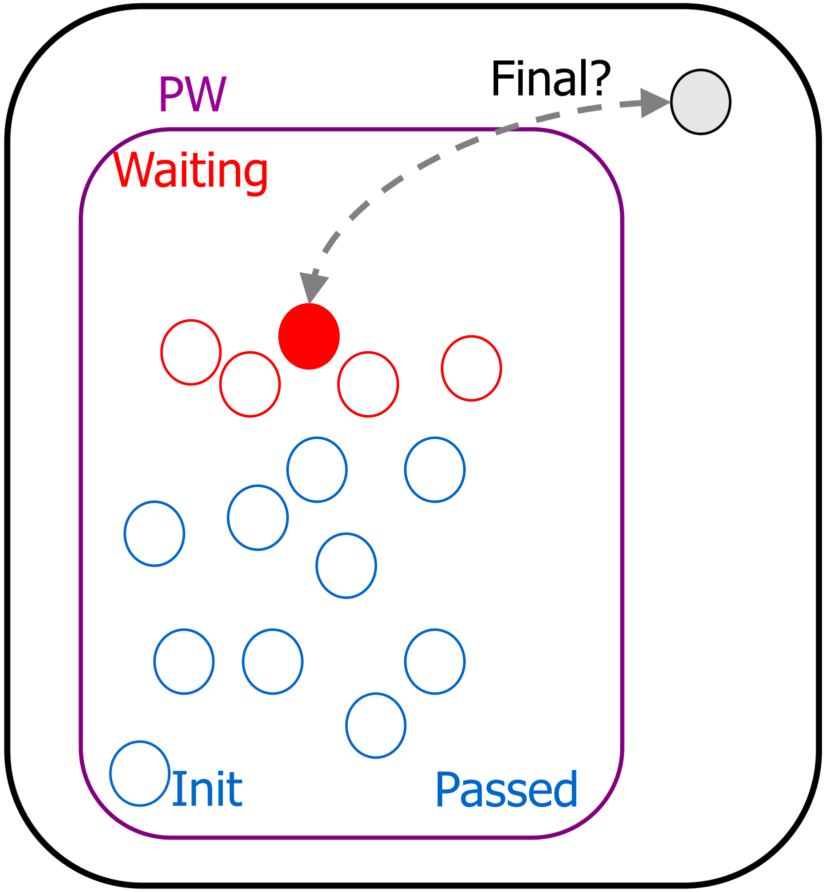
Init -> Final ?



```
INITIAL  Passed := Ø;
         Waiting := {($n_0$,$Z_0$)}

REPEAT
   pick (n,Z) in Waiting
   if (n,Z) = Final return true
   for all (n,Z)→(n′,Z′):
      if for some (n′,Z″) $Z' \subseteq Z''$ continue
      else add (n′,Z′) to Waiting
      move (n,Z) to Passed

UNTIL  Waiting = Ø
return false
```

Forward Reachability Algorithm

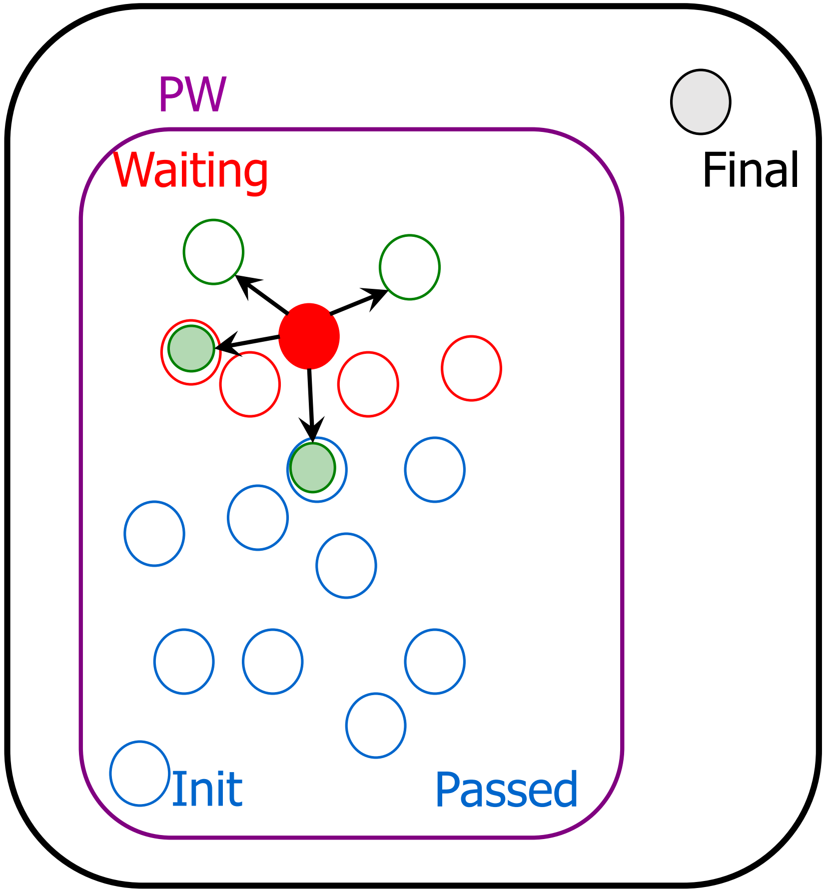
Init -> Final ?



```
INITIAL  Passed := $\emptyset$;
         Waiting := $\{(n_0,Z_0)\}$

REPEAT
   pick (n,Z) in Waiting
   if (n,Z) = Final return true
   for all $(n,Z)\rightarrow(n',Z')$:
      if for some $(n',Z'')$ $Z' \subseteq Z''$ continue
      else add $(n',Z')$ to Waiting
      move (n,Z) to Passed

UNTIL  Waiting = $\emptyset$
return false
```

Forward Reachability Algorithm

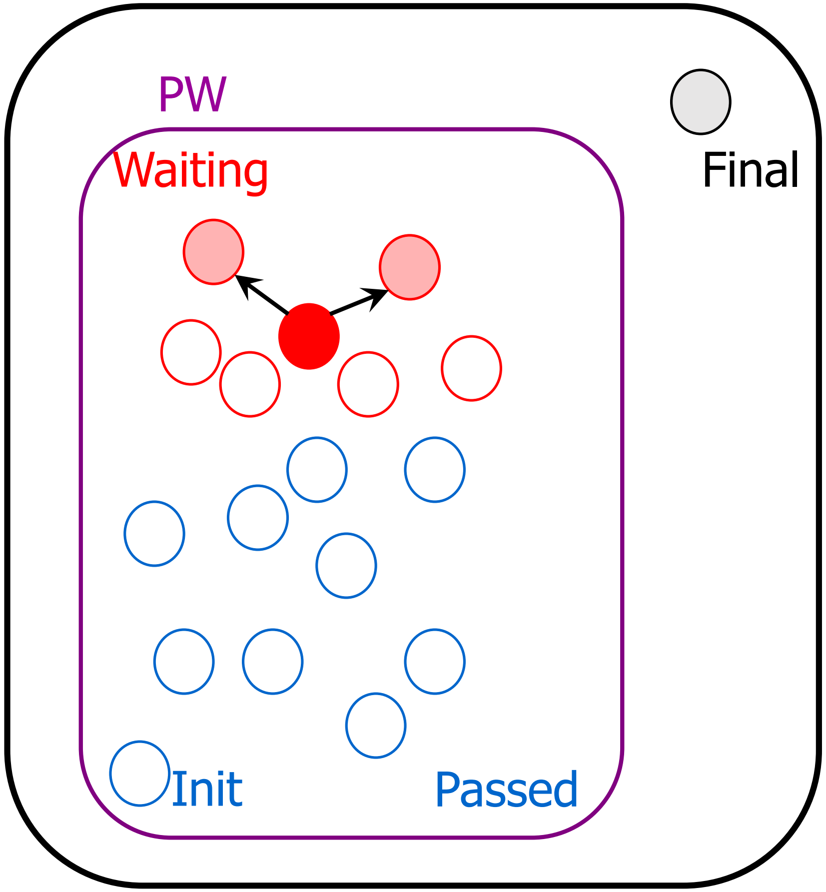
Init -> Final ?



```
INITIAL  Passed := Ø;
         Waiting := {(n_0,Z_0)}

REPEAT
   pick (n,Z) in Waiting
   if (n,Z) = Final return true
   for all (n,Z)→(n',Z'):
      if for some (n',Z'') Z'⊆ Z'' continue
      else add (n',Z') to Waiting
      move (n,Z) to Passed

UNTIL  Waiting = Ø
return false
```

Forward Reachability Algorithm

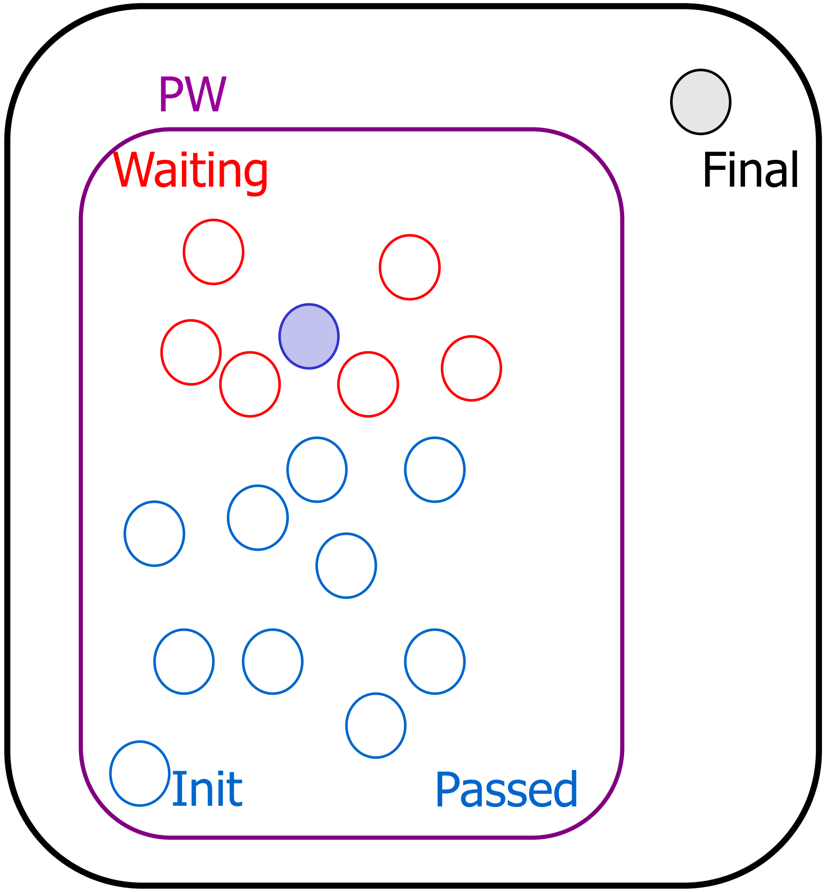
Init -> Final ?



```
INITIAL  Passed := Ø;
         Waiting := $\{(n_0,Z_0)\}$

REPEAT
    pick (n,Z) in Waiting
    if (n,Z) = Final return true
    for all (n,Z)→(n',Z'):
        if for some (n',Z'') $Z' \subseteq Z''$ continue
        else add (n',Z') to Waiting
        move (n,Z) to Passed

UNTIL  Waiting = Ø
return false
```

Forward Reachability Algorithm

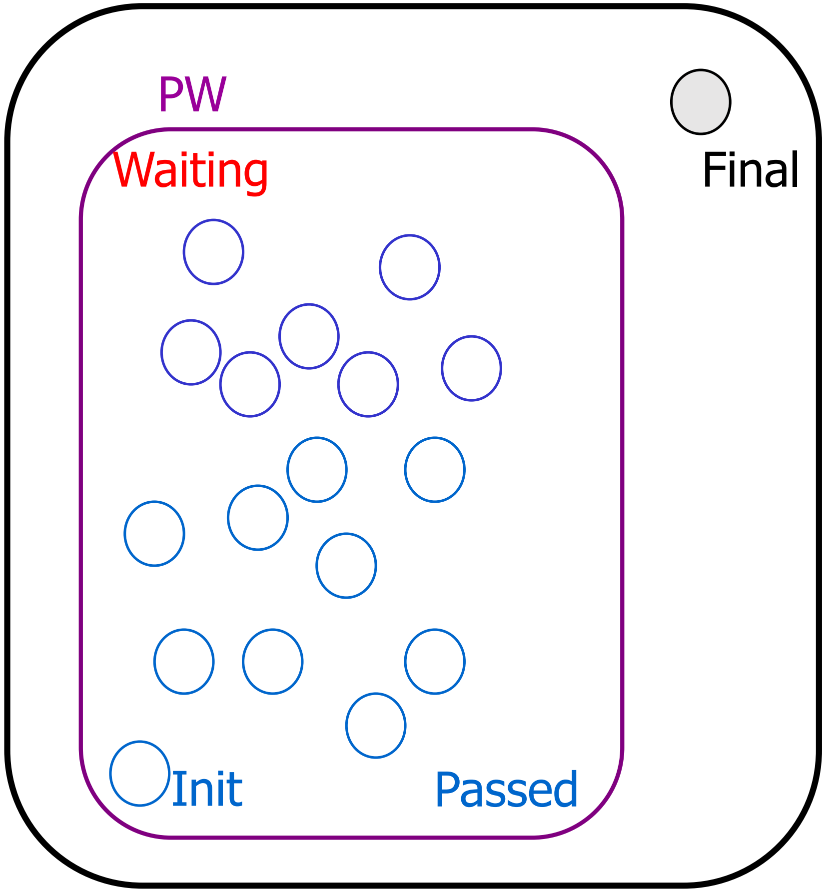
Init -> Final ?



```
INITIAL  Passed := Ø;
             Waiting := {($n_0$,$Z_0$)}

REPEAT
   pick (n,Z) in Waiting
   if (n,Z) = Final return true
   for all (n,Z)→(n′,Z′):
      if for some (n′,Z′) Z′⊆ Z″ continue
      else add (n′,Z′) to Waiting
      move (n,Z) to Passed

UNTIL  Waiting = Ø
return false
```

Forward Reachability Algorithm

Init -> Final ?



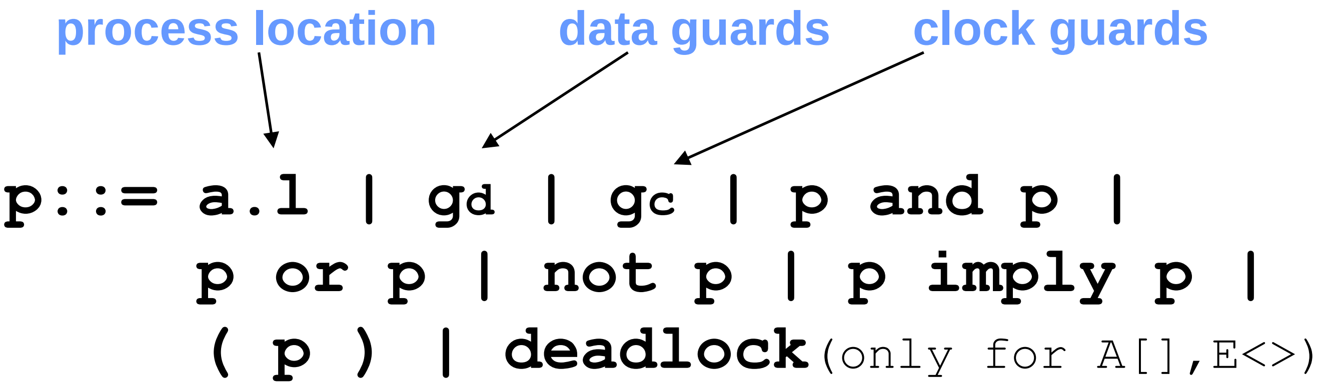
```
INITIAL  Passed := Ø;
              Waiting := {$(n_0,Z_0)$}

REPEAT
    pick (n,Z) in Waiting
    if (n,Z) = Final return true
    for all (n,Z)→(n',Z'):
        if for some (n',Z'') $Z' \subseteq Z''$ continue
        else add (n',Z') to Waiting
        move (n,Z) to Passed

UNTIL  Waiting = Ø
return false
```

Specification (Query) Language
TCTL

UPPAAL Property Specification Language

- **A[] p** *always*
- **A<> p** *inevitable*

- **E<> p** *Possible*
- **E[] p** *potentially always*
- **P --> q** *leads-to*



process location

data guards

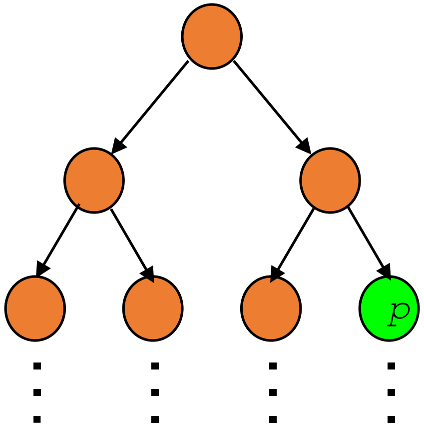
clock guards

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

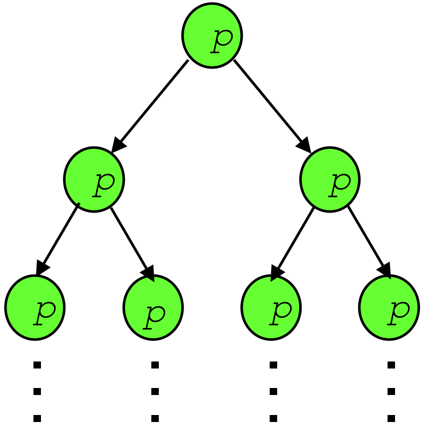
Example:
`A[] (mc1.finished and mc2.finished) imply (accountA+accountB==200)`

Uppaal "Computation Tree Logic"

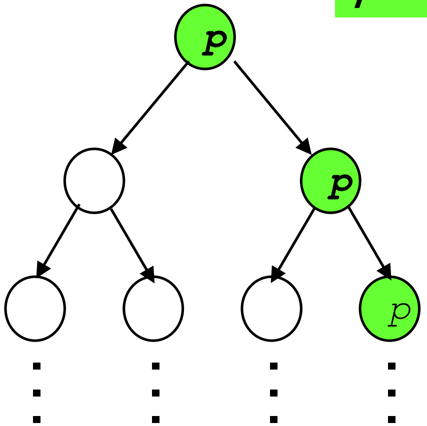
`E<> p` *Possible*



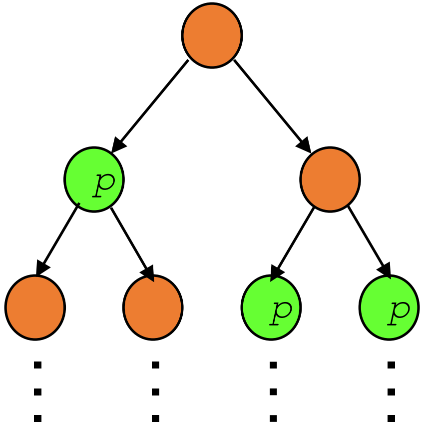
`A[] p` *always*



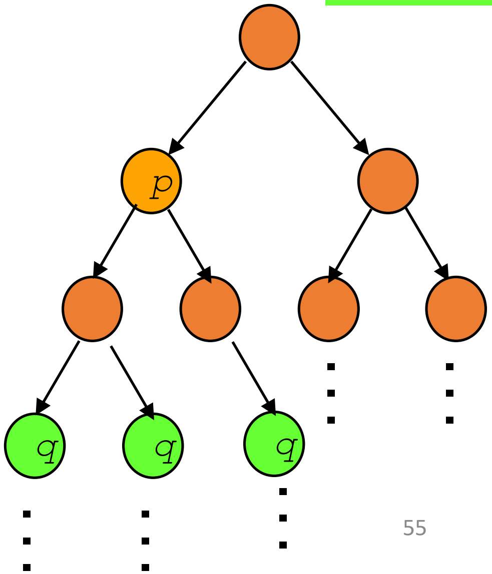
`E[] p` *potentially always*



`A<> p` *inevitable*



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



Logical Specifications

- Validation Properties
  - Possibly: E<> p
- Safety Properties
  - Invariant: A[] p
  - Possibly Inv.: E[] P
- Liveness Properties
  - Eventually: A<> p
  - Leads_to: p --> p
- Bounded Liveness
  - Leads to within: p --> $_{\leq t}$ q

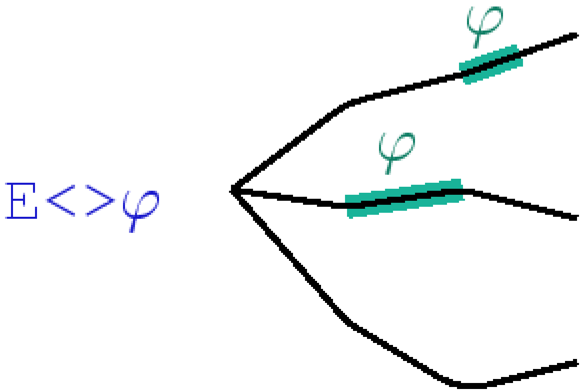
The expressions p and q

- must be type safe, side effect free, and evaluate to a boolean.

- only references to integer variables, constants, clocks, and locations are allowed (and arrays of these).

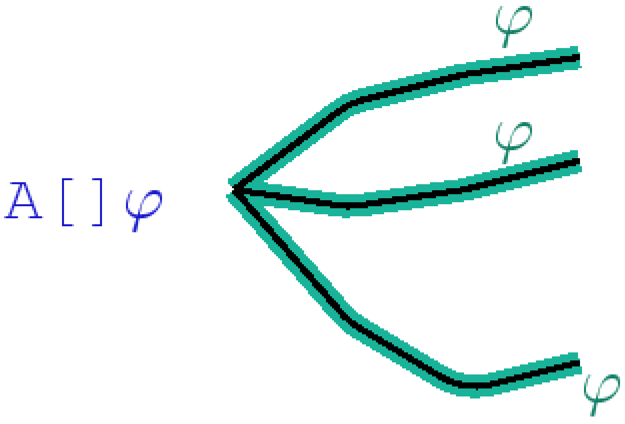
Logical Specifications

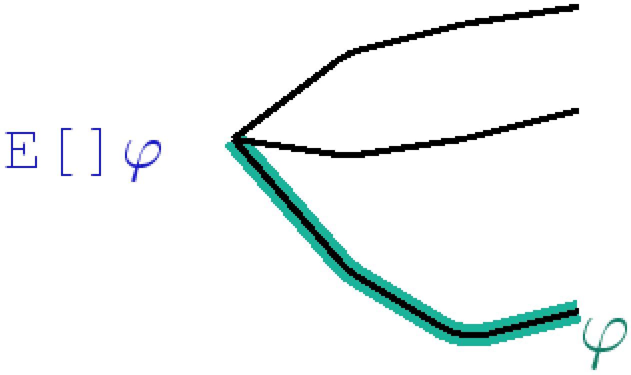
- Validation Properties
  - Possibly: E<> φ
- Safety Properties
  - Invariant: A[] φ
  - Pos. Inv.: E[] φ
- Liveness Properties
  - Eventually: A<> φ
  - Leadsto: φ --> ψ
- Bounded Liveness
  - Leads to within: φ --> $_{\leq t}$ ψ



Logical Specifications

- Validation Properties
  - Possibly: E<> *p*
- Safety Properties
  - Invariant: A[] φ
  - Pos. Inv.: E[] φ
- Liveness Properties
  - Eventually: A<> φ
  - Leadsto: φ --> ψ
- Bounded Liveness
  - Leads to within: φ --> $_{\leq t}$ ψ





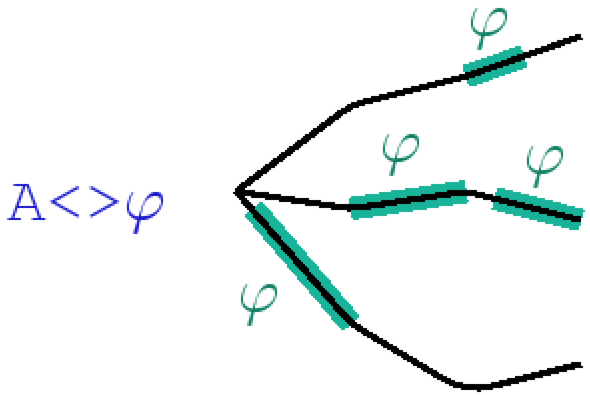
Logical Specifications

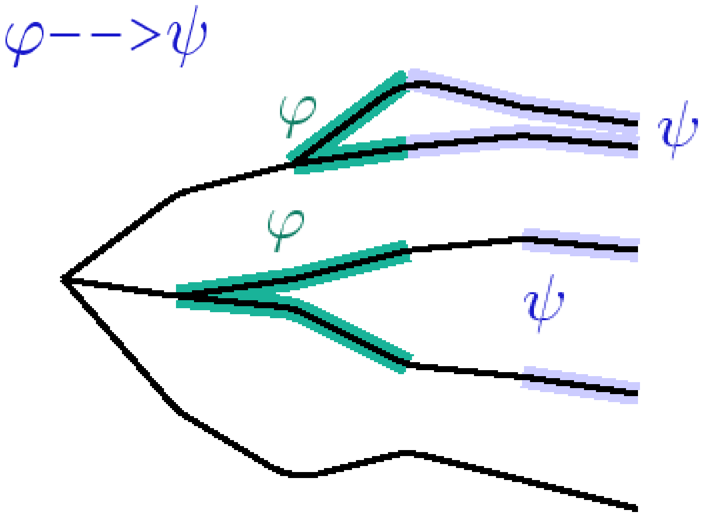
- Validation Properties
  - Possibly: E<> φ

- Safety Properties
  - Invariant: A[] φ
  - Pos. Inv.: E[] φ

- Liveness Properties
  - Eventually: A<> φ
  - Leads to: φ --> ψ

- Bounded Liveness
  - Leads to within: φ --> $_{\leq t}$ ψ





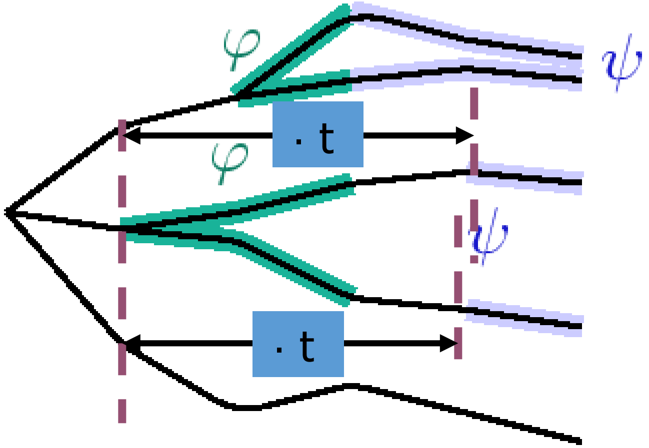
Logical Specifications

- Validation Properties
  - Possibly: E<> φ

- Safety Properties
  - Invariant: A[] φ
  - Pos. Inv.: E[] φ

- Liveness Properties
  - Eventually: A<> φ
  - Leadsto: φ --> ψ

- Bounded Liveness
  - Leads to within: φ --> $_{\leq t}$ ψ



Jug Example

- Safety: Never overflow.
  - A[] forall(i:id_t) level[i] <= capa[i]

- Validation/Reachability: How to get 1 unit.
  - E<> exists(i:id_t) level[i] == 1

Train-Gate Crossing Example

- Safety: One train on crossing at a time.
  - `A[] forall (i : id_t) forall (j : id_t)`
    `Train(i).Cross && Train(j).Cross imply i == j`
- Liveness: Approaching trains eventually arrive on crossing.
  - `Train(0).Appr --> Train(0).Cross`
  - `Train(1).Appr --> Train(1).Cross`
  - …
- No deadlock.
  - `A[] not deadlock`