CIS 721 - Real-Time Systems

Lecture 23: UPPAAL Internals

Mitch Neilsen
neilsen@ksu.edu

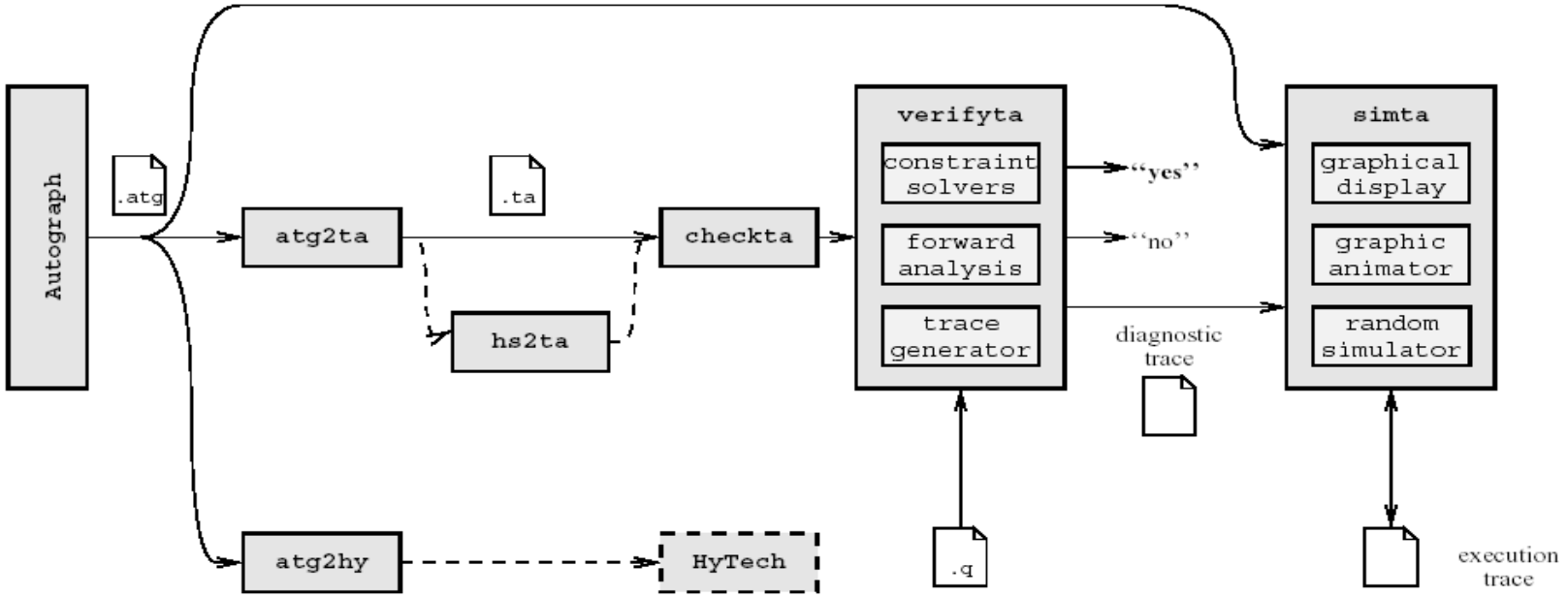
Outline

- Real-Time Verification and Validation Tools
  - **RT-SPIN – Real-Time extensions to SPIN**
  - **UPPAAL – Toolbox for validation and verification of real-time systems**
- Real-Time Communication

UPPAAL Components

- **UPPAAL** consists of three main parts:
  - a **description language**,
  - a **simulator**, and
  - a **model checker**.
- The **description language** is a non-deterministic guarded command language with data types. It can be used to describe a system as a *network of timed automata* using either a graphical (*.atg, *.xml) or textual (*.xta) format.
- The **simulator** enables examination of *possible* dynamic executions of a system during the early modeling stages.
- The **model checker** exhaustively checks *all* possible states.

UPPAAL Tools (earlier version)



- **checkta** – syntax checker
- **simta** – simulator
- **verifyta** – model checker

UPPAAL Specification Language

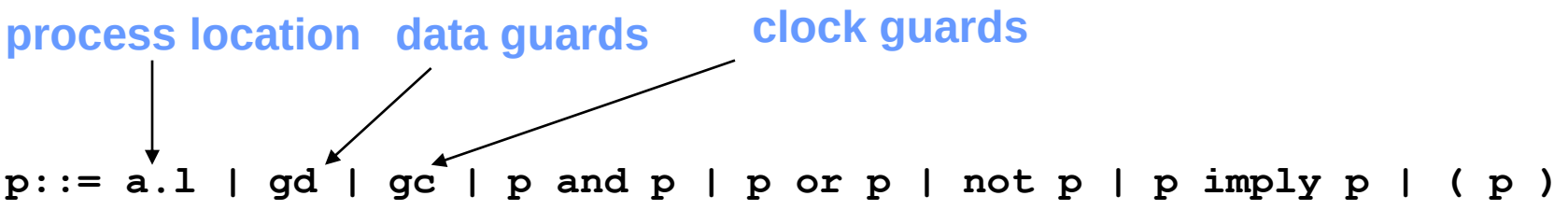
```
A[] p
E<> p
```

(AG p) – on all paths A, always G

(EF p) – on some path E, eventually F

A = on all paths, [ ] = always

E = on some path, <> = eventually



In UPPAAL

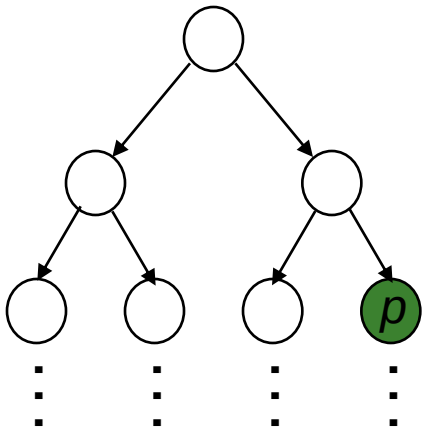
- **Invariantly:** The property **A[ ] p** evaluates to true if and only if every reachable state satisfy p.
- **Possibly:** The property **E<> p** evaluates to true for a timed transition system if and only if there is a sequence of alternating delay transitions and action transitions *s0*-->*s1*-->...-->*sn*, where *s0* is the initial state and *sn* satisfies *p*.
- **Potentially always (on some path always):** The property **E[ ] p** evaluates to true for a timed transition system if and only if there is a sequence of alternating delay or action transitions *s0*-->*s1*-->..*si*-->.. for which *p* holds in all states *si* and which either:
  - is infinite, or
  - ends in a state (*Ln, vn*) such that either
    - for all *d*: (*Ln, vn* + *d*) satisfies *p* and *Inv*(*Ln*), or
    - there is no outgoing transition from (*Ln, vn*)
- **Eventually (on all paths eventually):** The property **A<> p** evaluates to true if (and only if) all possible transition sequences eventually reaches a state satisfying p. An *eventually* property A<> p can be expressed as the *potentially* property not E[ ] not p.

CTL, Derived Operators

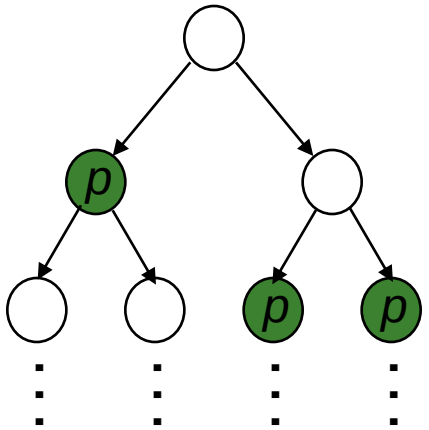
$$\mathsf{EF}\,\phi \equiv \mathsf{E}[\mathrm{true}\,\mathsf{U}\,\phi]$$ *possible*

$$\mathsf{AF}\,\phi \equiv \mathsf{A}[\mathrm{true}\,\mathsf{U}\,\phi].$$ *inevitable*

EF p



AF p



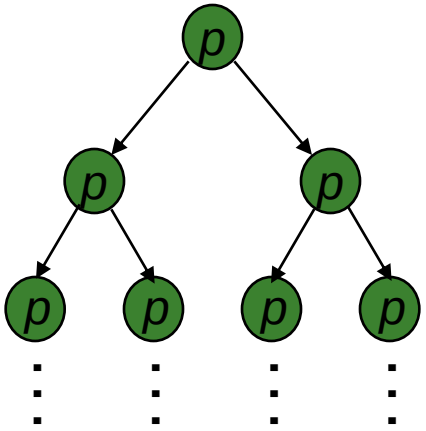
CTL, Derived Operators

$$\mathbf{EG}\,\phi \equiv \neg\,\mathbf{AF}\,\neg\,\phi$$ *potentially always*

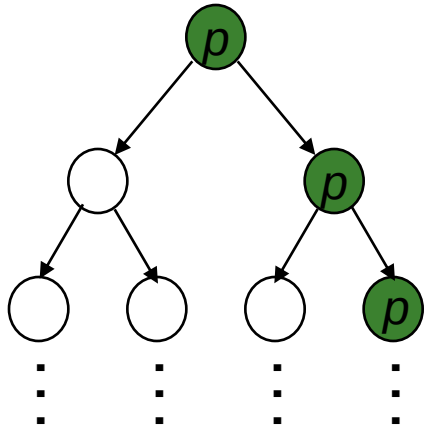
$$\mathbf{AG}\,\phi \equiv \neg\,\mathbf{EF}\,\neg\,\phi$$ *always*

$$\mathbf{AX}\,\phi \equiv \neg\,\mathbf{EX}\,\neg\,\phi.$$

AG p

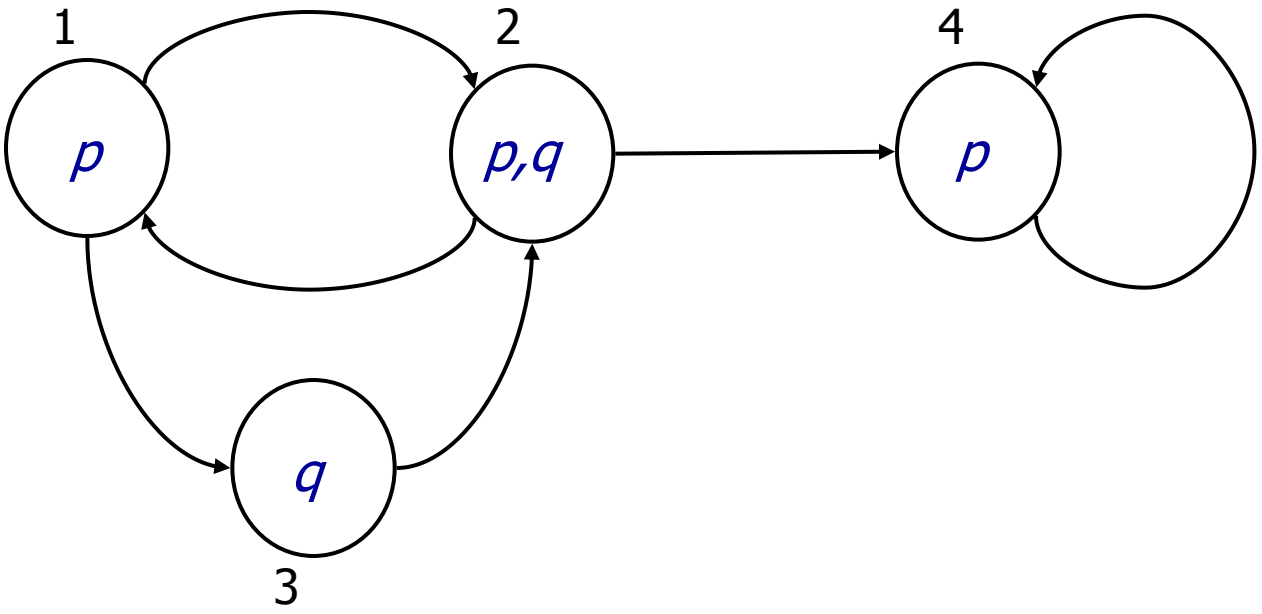


EG p



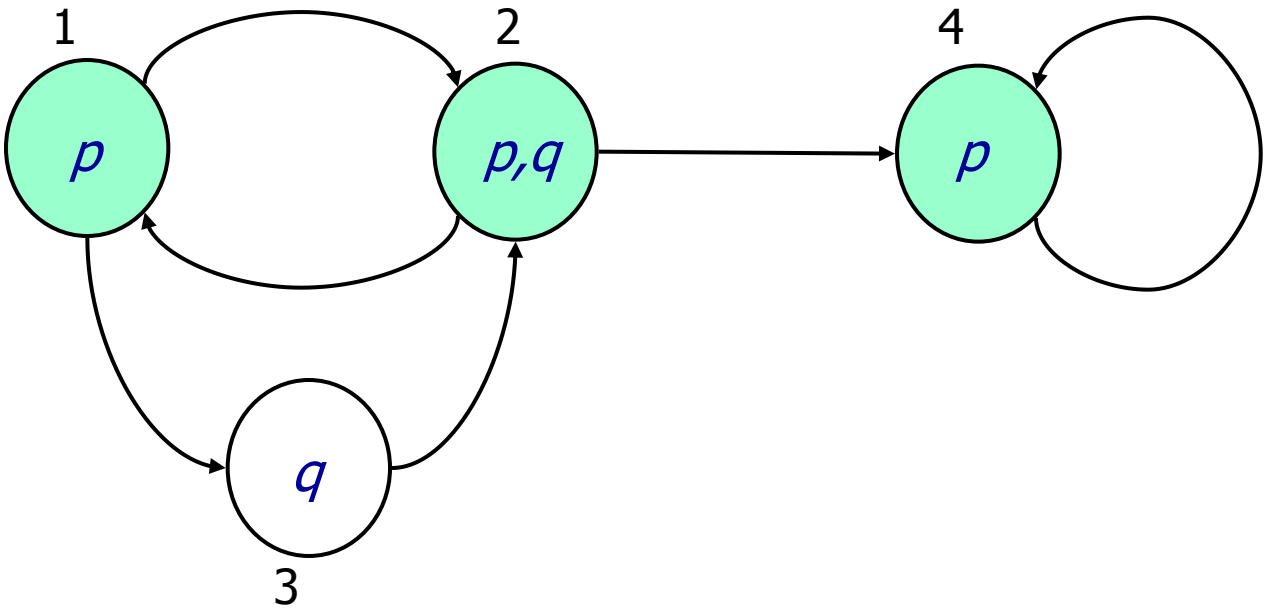
Example

E[] *p*



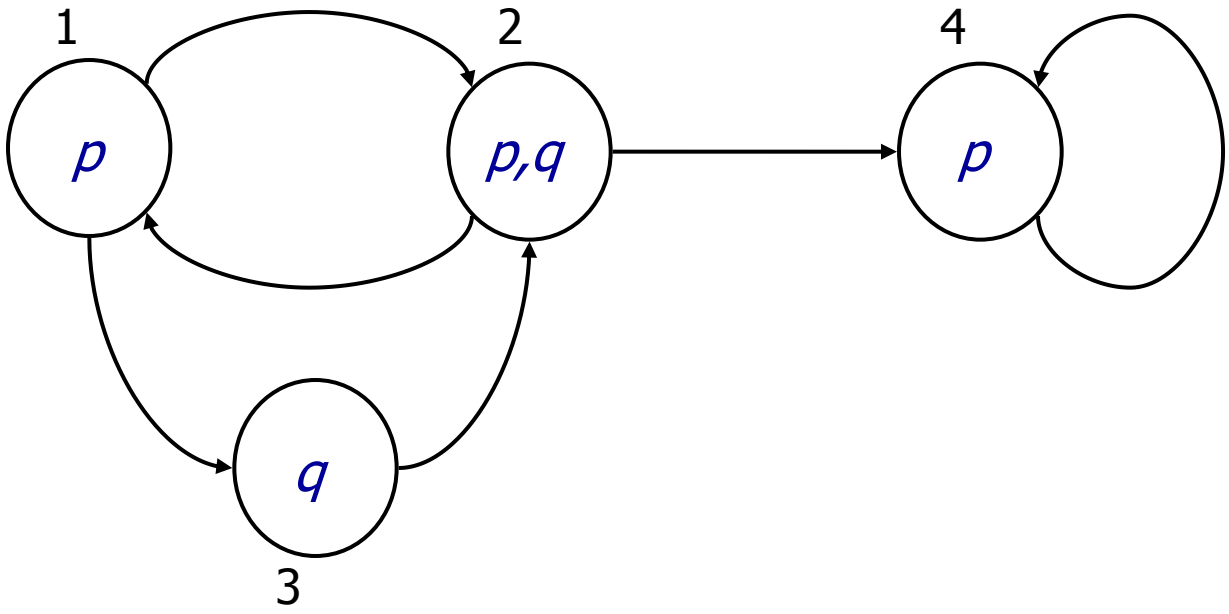
Example

E[] *p*



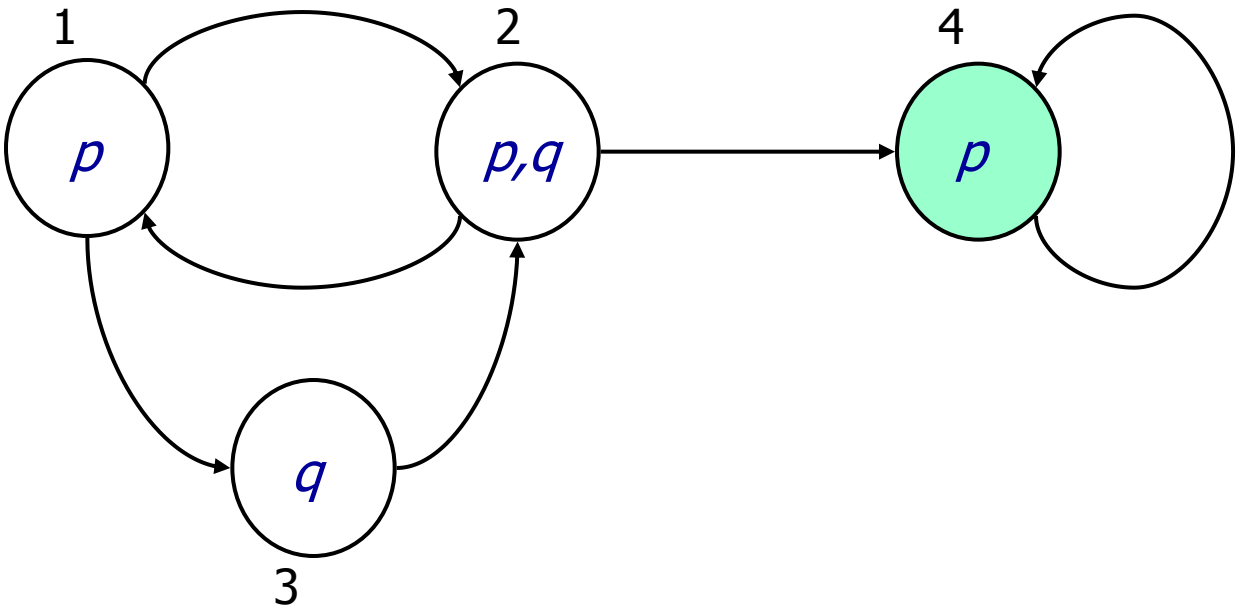
Example

A[] *p*



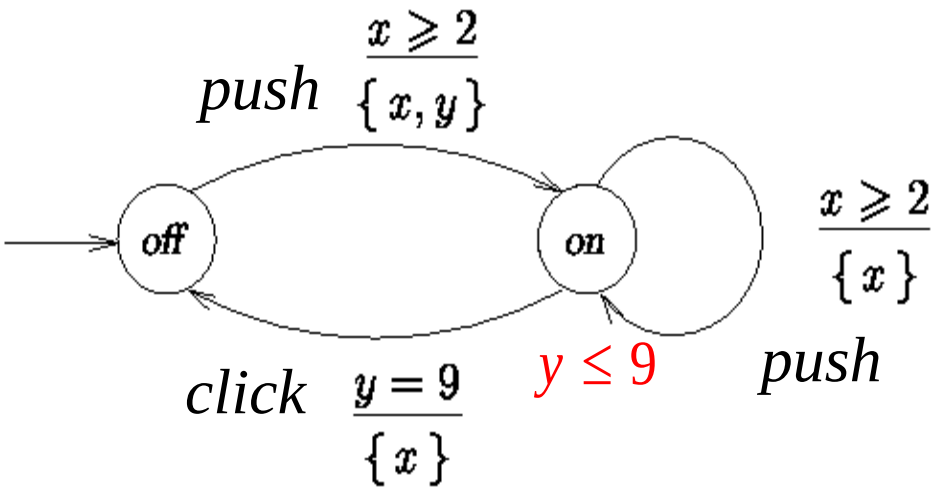
Example

A[] *p*



Timed CTL (TCTL)

Light Switch



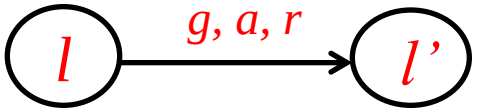
- Switch may be turned on whenever at least 2 time units has elapsed since last “turn off”
- Light automatically switches off after 9 time units.

Semantics

- *clock valuations*: $V(C) \quad v : C \to R_{\geq 0}$
- *state*: $(l, v) \quad where \quad l \in L \ \ and \ \ v \in V(C)$
- Semantics of timed automata is a *labeled transition system* where

$$S = \{ \ (l, v) \ \ | v \in V(C) \ \ and \ \ l \in L \ \}$$

$(S, \to)$



- *action transition*

$$(l, v) \xrightarrow{a} (l', v') \ \ iff$$

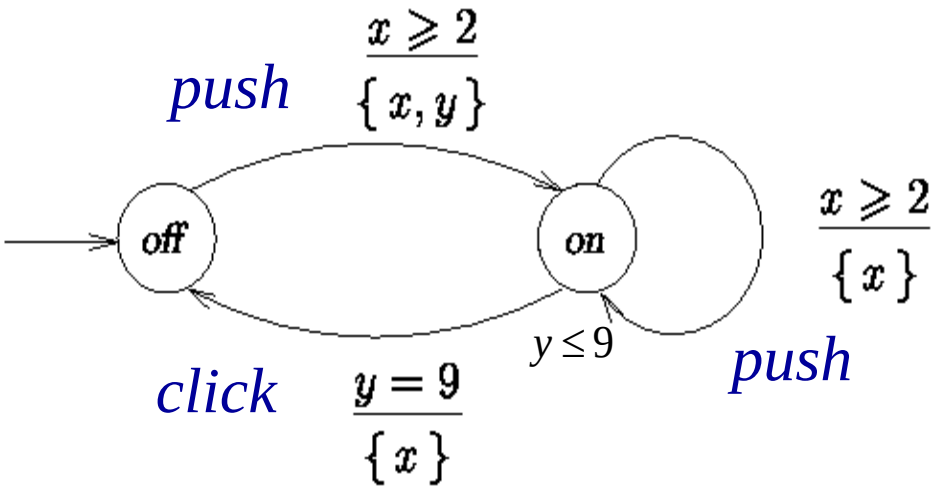
$$g(v) \ \ and \ \ v' = v[r] \ \ and \ \ Inv(l')(v')$$

- *delay Transition*

$$(l, v) \xrightarrow{d} (l, v + d) \ iff$$

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

Semantics: Example



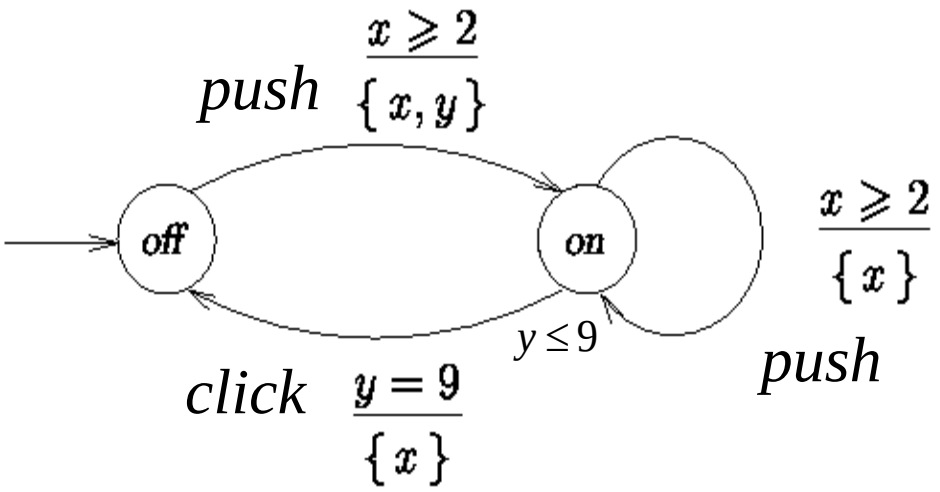
$$(off, x = y = 0) \xrightarrow{3.5} (off, x = y = 3.5) \xrightarrow{push}$$

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

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

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

Light Switch (cont.)




```
A[] (x <= y)
P.on --> P.off
```

$\mathsf{AG}(x \leq y)$

$\mathsf{AG}(on \Rightarrow \mathsf{AF}\, off)$

$\mathsf{AG}(on \Rightarrow \mathsf{AF}_{\leq 9}\, off)$

$\mathsf{A}[off\ \mathsf{U}\ x \geq 2]$

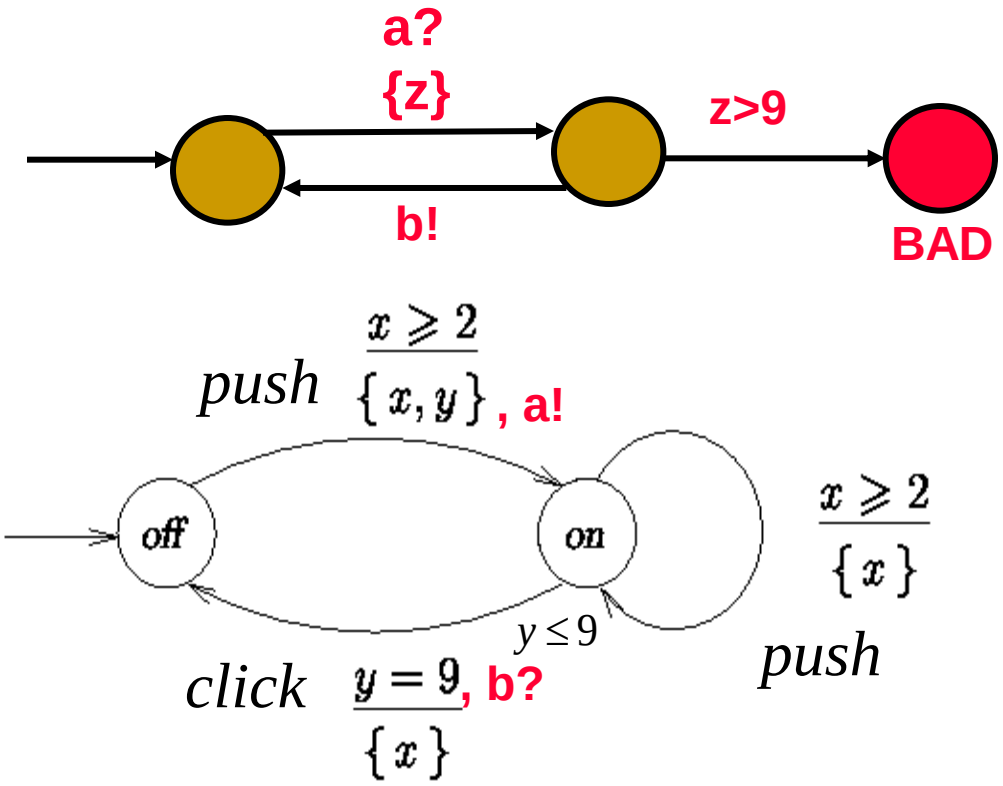
$\mathsf{A}[off\ \mathsf{U}\ x \geq 3]$

$\mathsf{E}[off\ \mathsf{U}\ x \geq 3]$

$\mathsf{A}[x \leq 2\ \mathsf{U}\ on]$

$\mathsf{E}[x \leq 2\ \mathsf{U}\ on]$

Light Switch (Add Observer)



$\mathsf{AG}(x \le y)$

$\mathsf{AG}(on \Rightarrow \mathsf{AF}\, off)$

$\mathsf{AG}(on \Rightarrow \mathsf{AF}_{\le 9}\, off)$

$\mathsf{A}[off\ \mathsf{U}\ x \ge 2]$

$\mathsf{A}[off\ \mathsf{U}\ x \ge 3]$

$\mathsf{E}[off\ \mathsf{U}\ x \ge 3]$

$\mathsf{A}[x \le 2\ \mathsf{U}\ on]$

$\mathsf{E}[x \le 2\ \mathsf{U}\ on]$

Timeliness Properties

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

receive(m) always occurs within 5 time units after *send(m)*

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

receive(m) may occur exactly 11 time units after *send(m)*

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

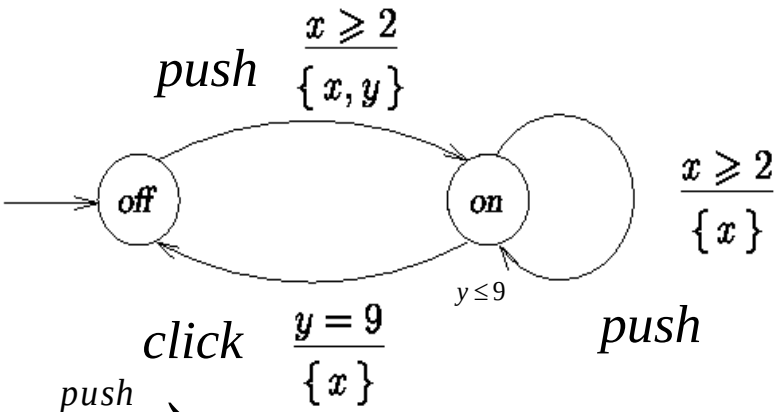
putbox occurs periodically (exactly) every 25 time units

(note: other *putbox*'s may occur in between)

Paths

A *path* is an infinite sequence $s_0\, a_0\, s_1\, a_1\, s_2\, a_2 \ldots$ of states alternated by transition labels such that $s_i \xrightarrow{a_i} s_{i+1}$ for all $i \geqslant 0$.

Example Path:



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

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

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

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

Elapsed Time in Path

$$\Delta(\sigma,0) = 0$$

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

Example:

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

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

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

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

$\Delta(\sigma,1)=3.5$, $\Delta(\sigma,6)=3.5+9=12.5$

TCTL Semantics

$$
\begin{array}{ll}
s, w \models p & \text{iff } p \in Label(s) \\
s, w \models \alpha & \text{iff } v \cup w \models \alpha \\
s, w \models \neg \phi & \text{iff } \neg (s, w \models \phi) \\
s, w \models \phi \vee \psi & \text{iff } (s, w \models \phi) \vee (s, w \models \psi) \\
s, w \models z \text{ in } \phi & \text{iff } s, \text{reset } z \text{ in } w \models \phi \\
s, w \models \mathsf{E}[\phi \mathsf{U} \psi] & \text{iff } \exists \sigma \in P^{\infty}_{\mathcal{M}}(s). \exists (i, d) \in Pos(\sigma). \\
 & \quad (\sigma(i, d), w{+}\Delta(\sigma, i) \models \psi \wedge \\
 & \quad (\forall (j, d') \ll (i, d). \sigma(j, d'), w{+}\Delta(\sigma, j) \models \phi \vee \psi)) \\
s, w \models \mathsf{A}[\phi \mathsf{U} \psi] & \text{iff } \forall \sigma \in P^{\infty}_{\mathcal{M}}(s). \exists (i, d) \in Pos(\sigma). \\
 & \quad ((\sigma(i, d), w{+}\Delta(\sigma, i)) \models \psi \wedge \\
 & \quad (\forall (j, d') \ll (i, d). (\sigma(j, d'), w{+}\Delta(\sigma, j)) \models \phi \vee \psi)).
\end{array}
$$

s - (location, clock valuation)

w - formula clock valuation

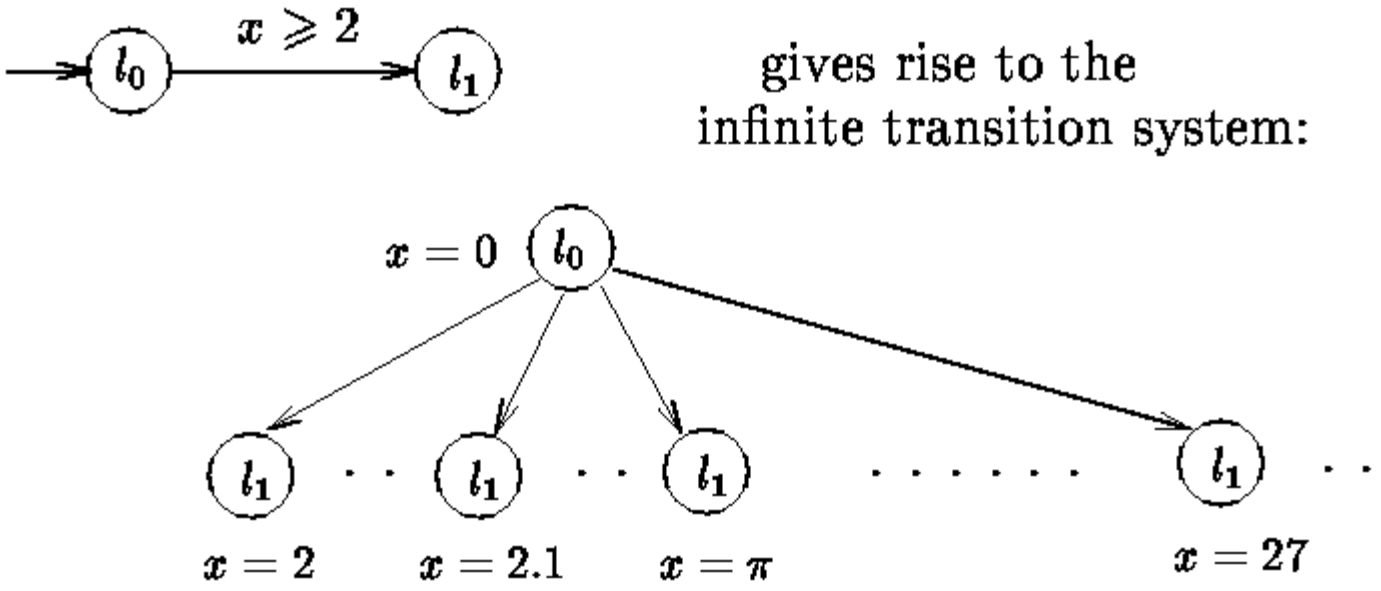
$P^{\infty}_{M}(s)$ - set of paths from *s*

Pos(σ) - positions in σ

Δ(σ,i) - elapsed time

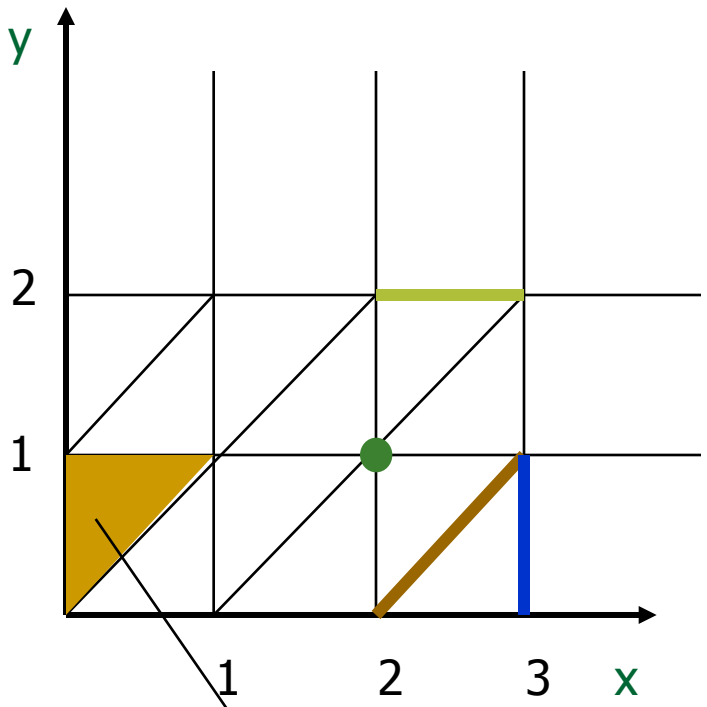
(i,d) <<(j,d') iff (i<j) or ((i=j) and (d<d'))

Infinite State Space?



Regions – two clocks: x, y

Finite partitioning of state space



Definition

$w \approx w'$ iff w and w' satisfy the exact same conditions of the form

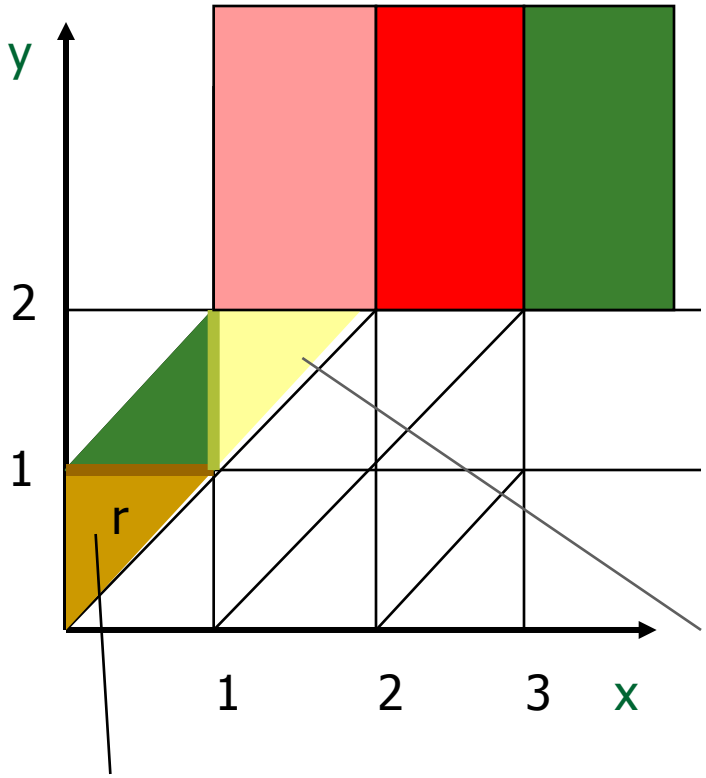
$$x_i \leq n \text{ and } x_i - x_j \leq n$$

where $n \leq \max$

An equivalence class (i.e. a *region*)
in fact there are only a *finite* number of regions!

Regions

Finite partitioning of state space



Definition

$w \approx w'$ iff w and w' satisfy the exact same conditions of the form

$$x_i \leq n \text{ and } x_i - x_j \leq n$$

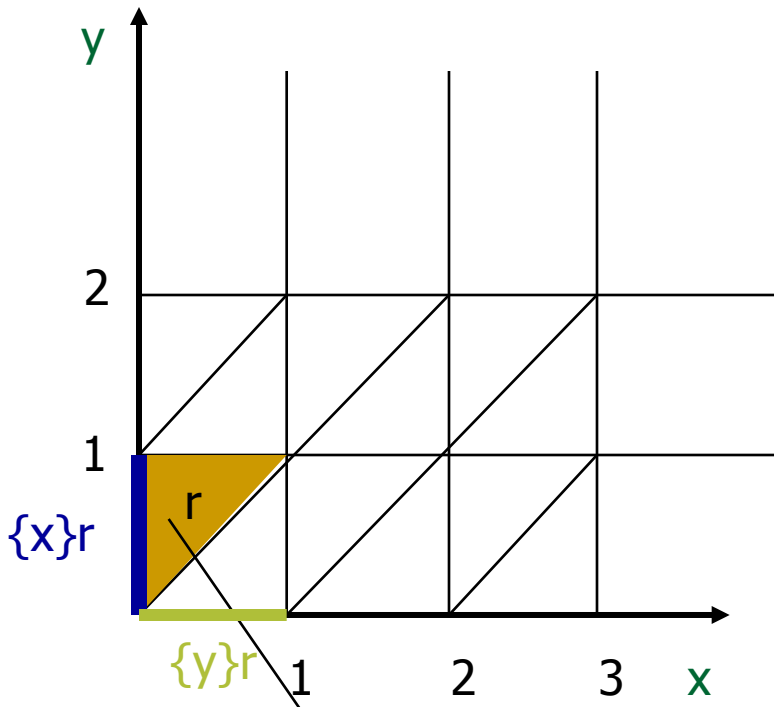
where $n \leq \max$

Successor regions, Succ(r)

An equivalence class (i.e. a *region*)

Regions

Finite partitioning of state space



Reset regions

An equivalence class (i.e. a *region*) r

Definition

$w \approx w'$ iff w and w' satisfy the exact same conditions of the form

$x_i \le n$ and $x_i - x_j \le n$

where $n \le \max$

THEOREM

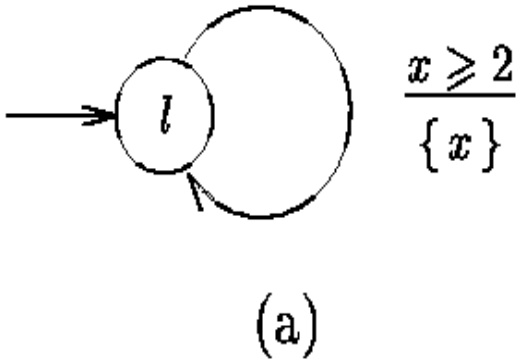
Whenever $uv \approx u'v'$ then

$[(l,u),v]$ sat ϕ

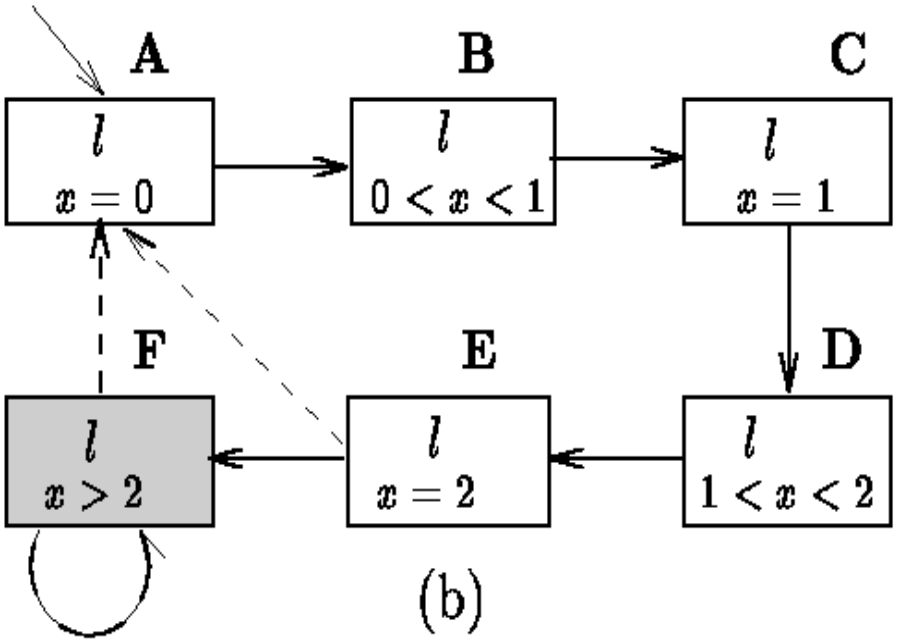
$\Leftrightarrow$

$[(l,u'),v']$ sat ϕ

Region graph of simple timed automata

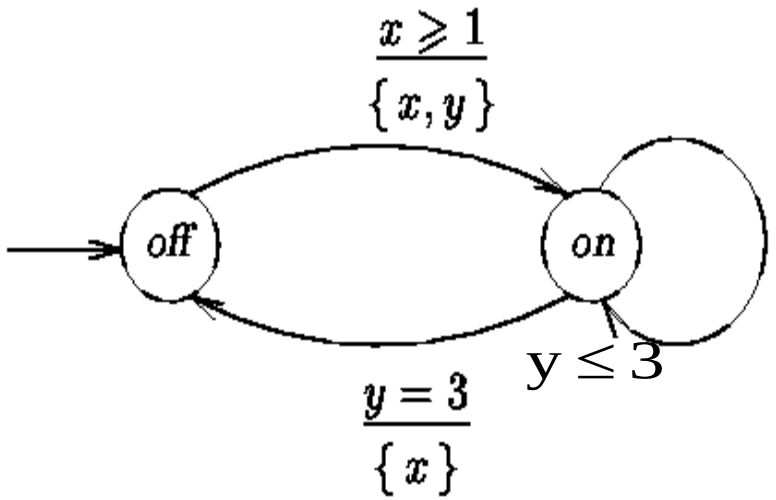


(a)



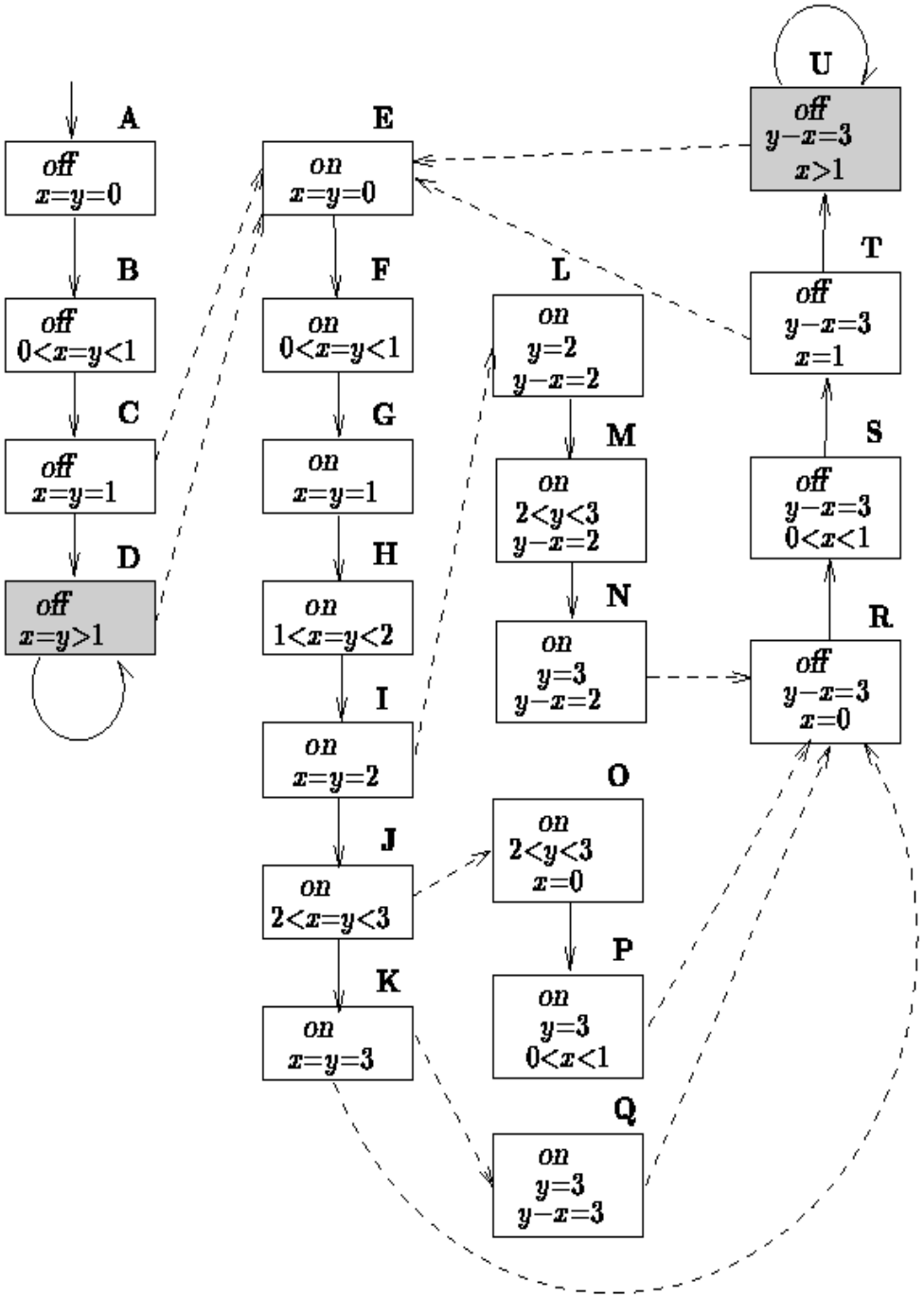
(b)

Modified light switch



$inv(off) = \text{true}$

$inv(on) = y \leqslant 3$



Reachable part of region graph

Properties

$\mathrm{AG}(x \le y)$

$\mathrm{AG}(on \Rightarrow \mathrm{AF}off)$

$\mathrm{AG}(on \Rightarrow \mathrm{AF}_{\le 3}off)$

Roughly speaking....

Model checking a timed automata against a TCTL-formula amounts to model checking its region graph against a CTL-formula

Problem to be solved

The worst-case time complexity of model checking TCTL*-formula* ϕ *over timed automaton* $\mathcal{A}$*, with the clock constraints of* ϕ *and* $\mathcal{A}$ *in* Ψ *is:*

$$\mathcal{O}\left(|\phi| \times \left(n! \times 2^n \times \prod_{x \in \Psi} c_x \times |L|^2\right)\right).$$

(i) linear in the length of the formula ϕ

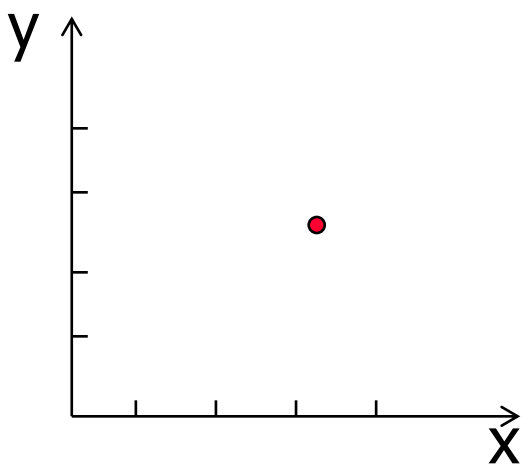
(ii) exponential in the number of clocks in $\mathcal{A}$ and ϕ

(iii) exponential in the maximal constants with which clocks are compared in $\mathcal{A}$ and ϕ.

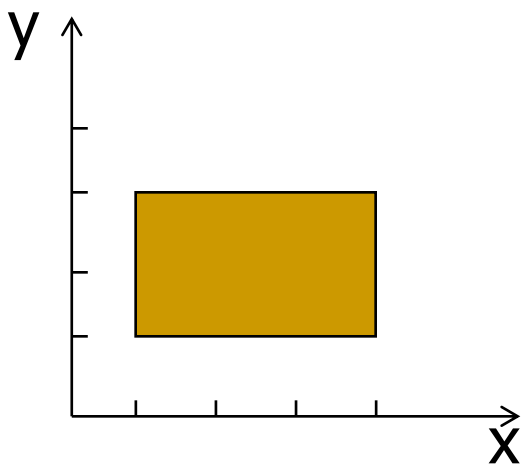
Model Checking TCTL is PSPACE-hard

Zones: From Infinite to Finite

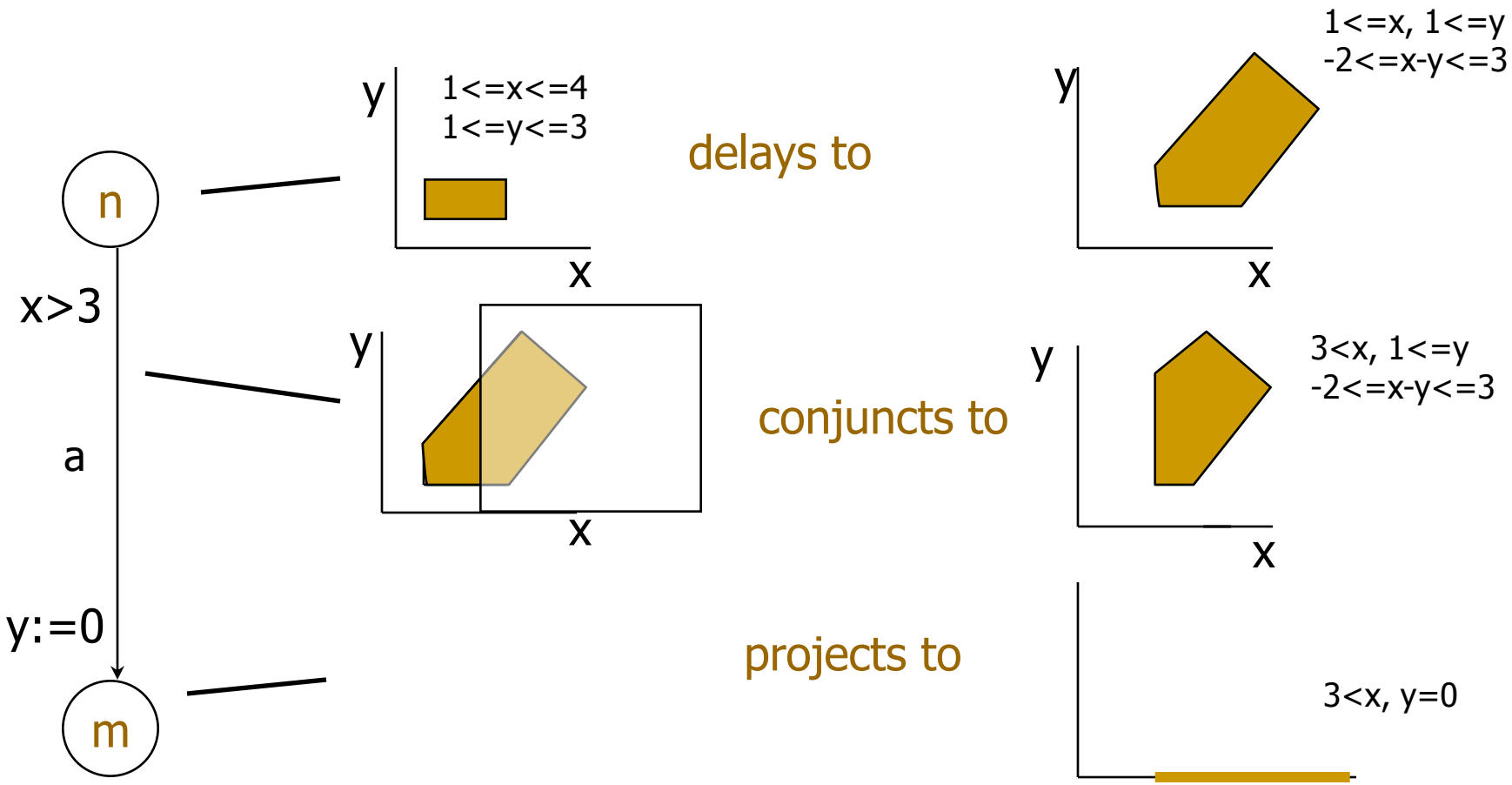
State
(n, x=3.2, y=2.5)



Symbolic state (set)
(n, $1 \le x \le 4, 1 \le y \le 3$)



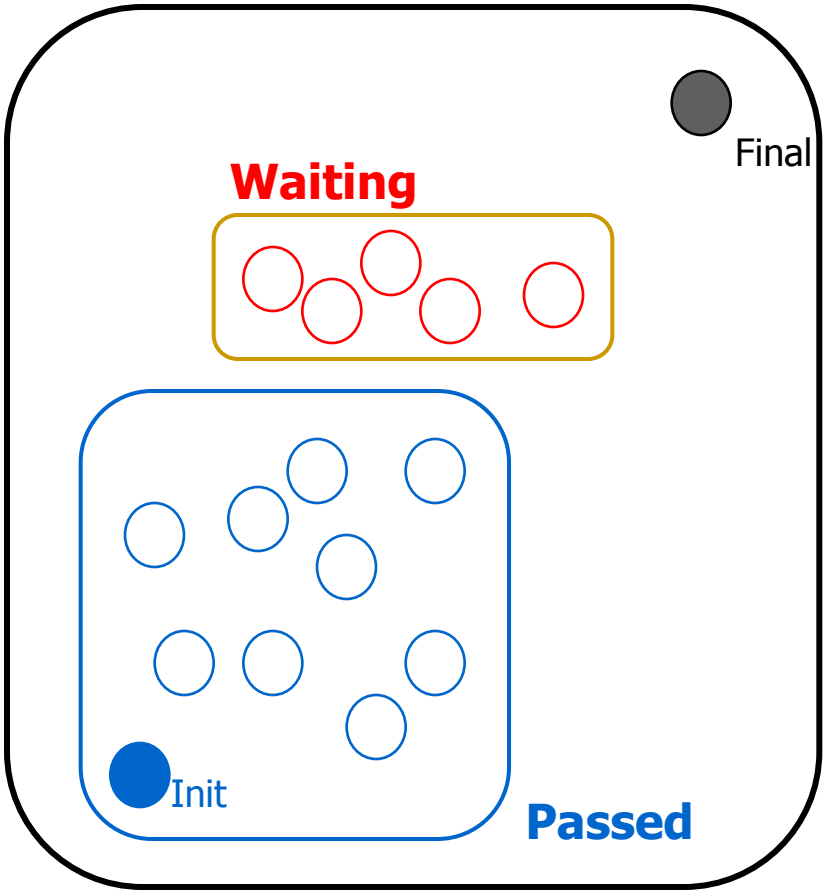
Symbolic Transitions



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

Forward Reachability

Init -> Final ?



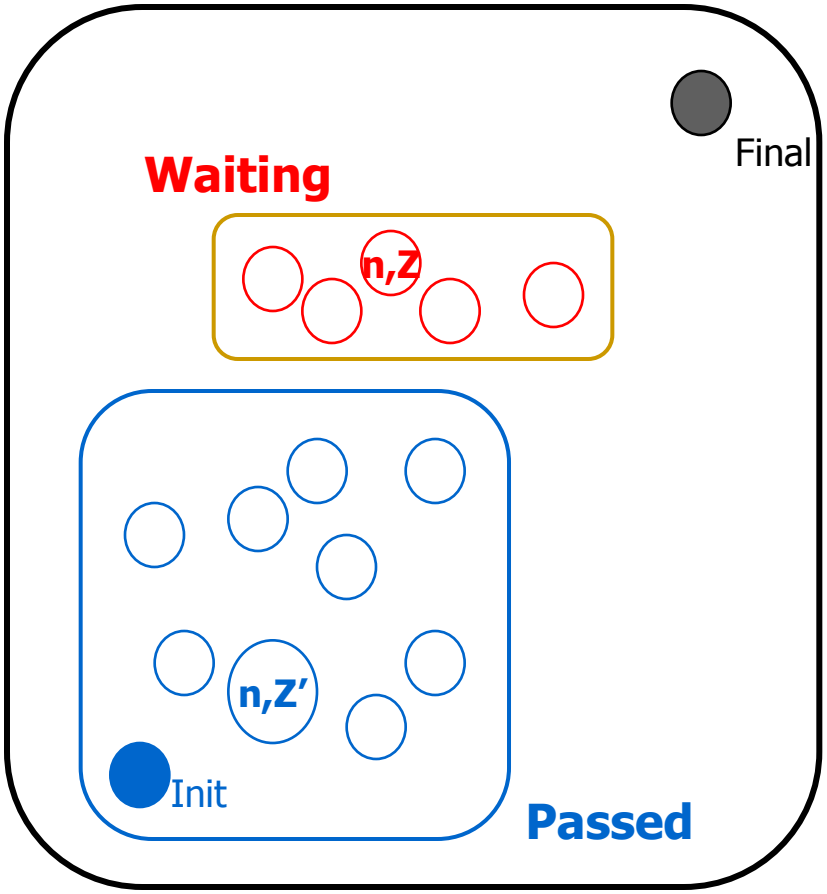
INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

REPEAT

UNTIL **Waiting** = Ø
or
Final is in **Waiting**

Forward Reachability

Init -> Final ?



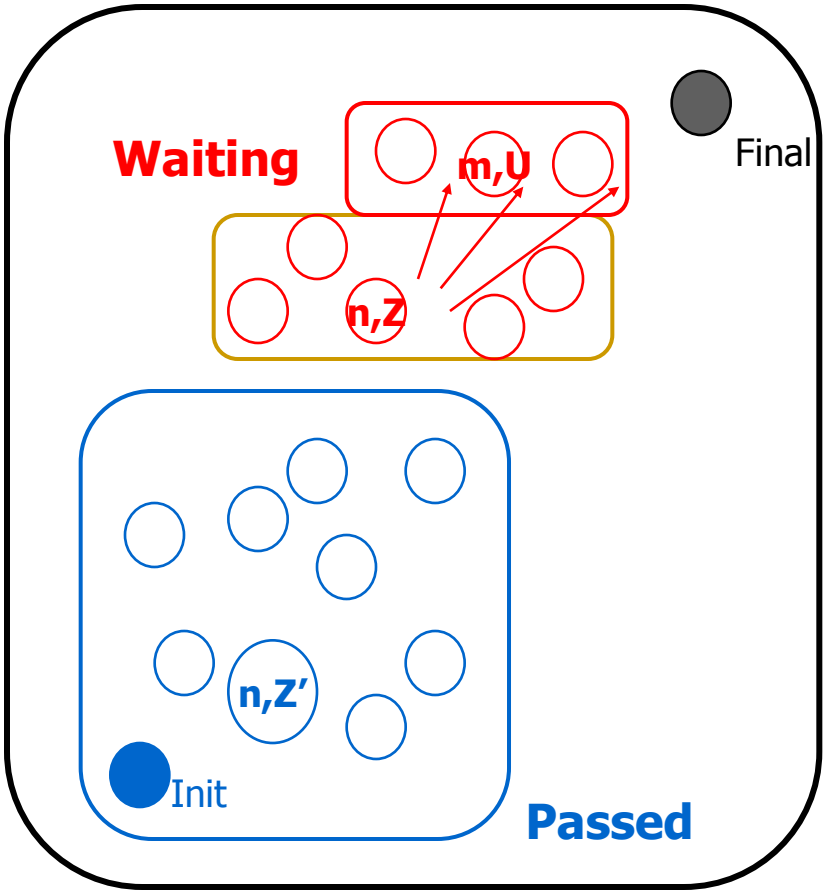
INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

REPEAT
- pick (n,Z) in **Waiting**
- **if** for some Z' ⊇ Z
(n,Z') in **Passed** **then** **STOP**

UNTIL **Waiting** = Ø
or
Final is in **Waiting**

Forward Reachability

Init -> Final ?



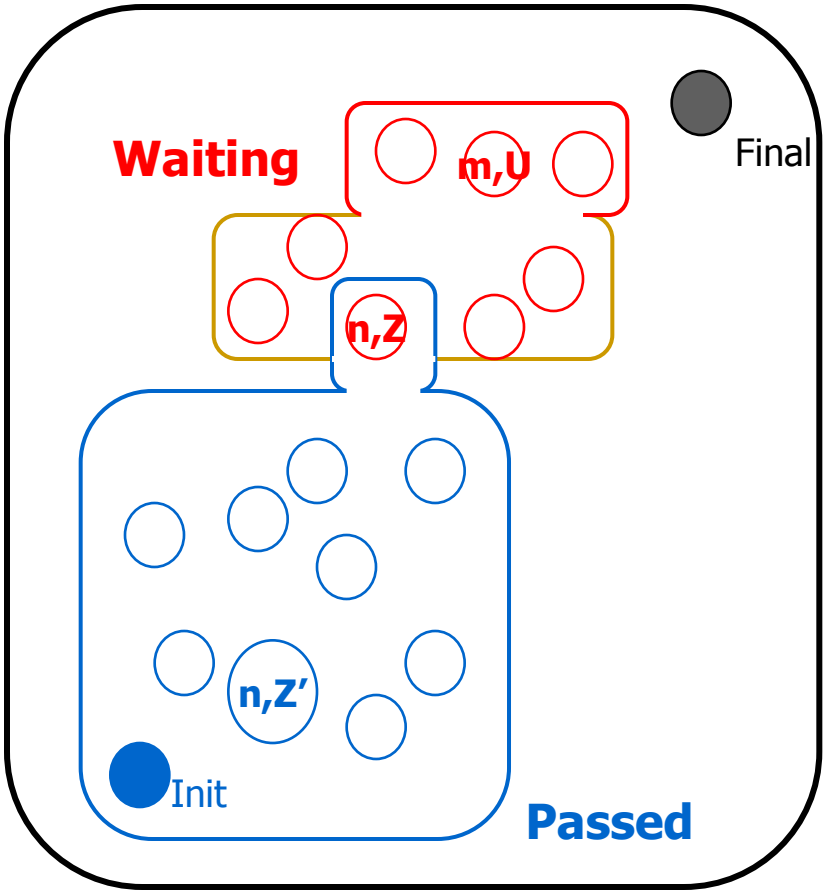
INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

REPEAT
- pick (n,Z) in **Waiting**
- **if** for some Z' ⊇ Z
  (n,Z') in **Passed** **then** **STOP**
- **else** /explore/ add
  { (m,U) : (n,Z) => (m,U) }
  to **Waiting**;

UNTIL **Waiting** = Ø
or
Final is in **Waiting**

Forward Reachability

Init -> Final ?

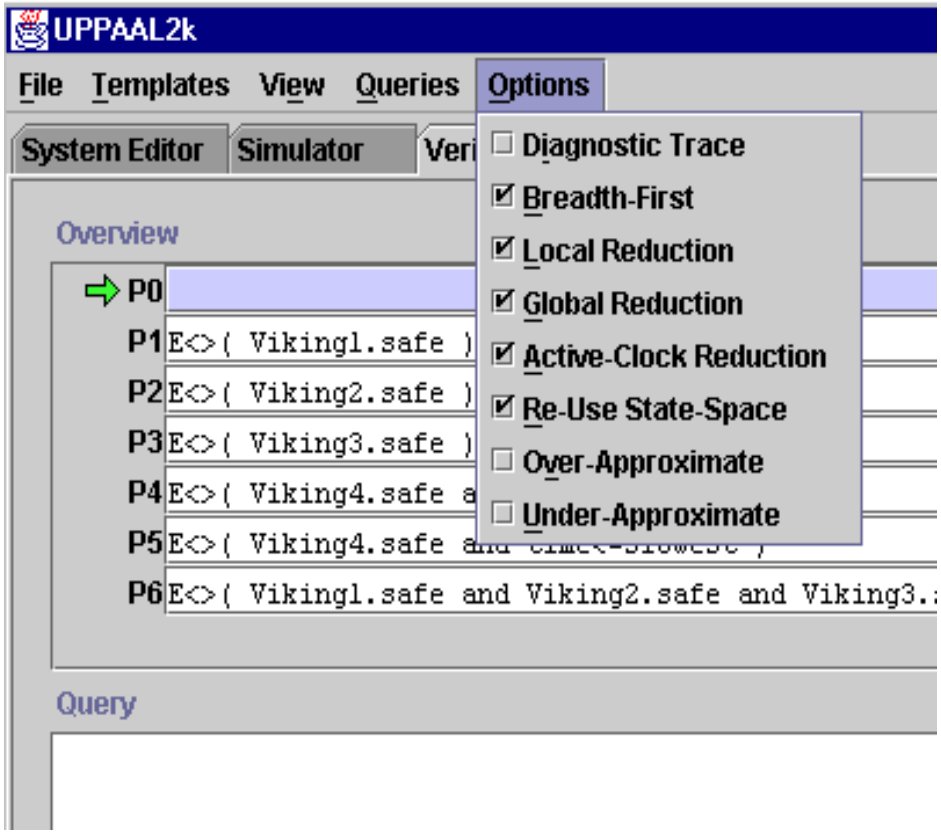


INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

REPEAT
- pick (n,Z) in **Waiting**
- **if** for some Z' ⊇ Z
(n,Z') in **Passed** **then** **STOP**
- **else** /explore/ add
{ (m,U) : (n,Z) => (m,U) }
to **Waiting**;
Add (n,Z) to **Passed**

UNTIL **Waiting** = Ø
or
Final is in **Waiting**

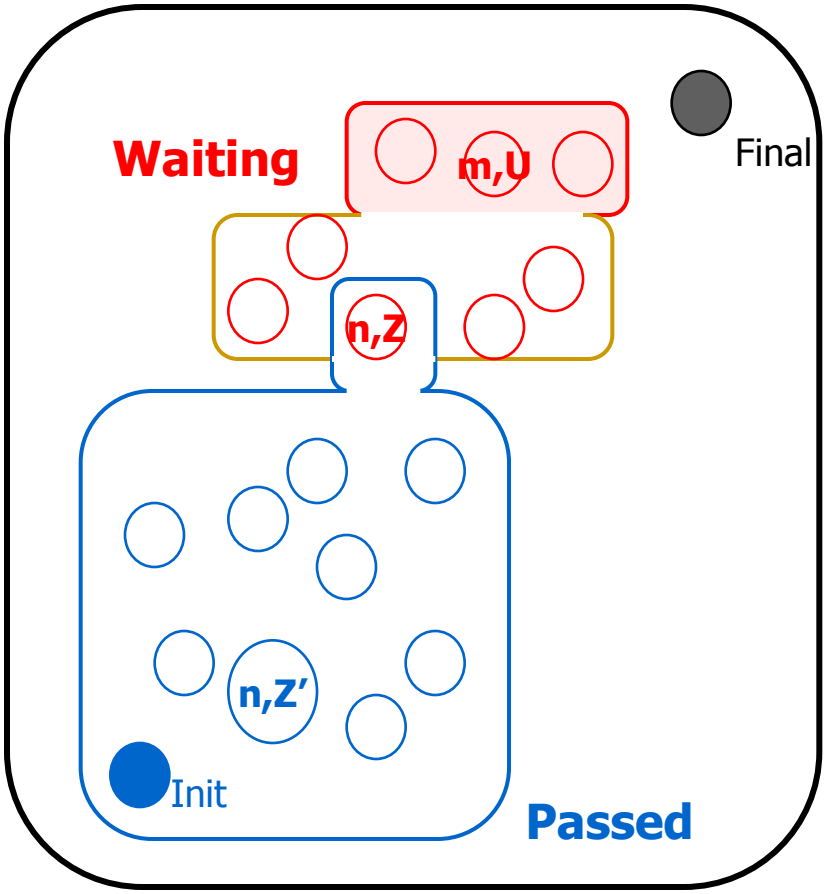
UPPAAL Verification Options



- Diagnostic Trace
- Breadth-First
  - Depth-First
- Local Reduction
- Global Reduction
- Active-Clock Reduction
- Re-Use State-Space
- Over-Approximation
- Under-Approximation

Forward Rechability

Init -> Final ?



INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

REPEAT
- pick (n,Z) in **Waiting**
- **if** for some Z' ⊇ Z
  (n,Z') in **Passed** **then** **STOP**
- **else** /explore/ add
  { (m,U) : (n,Z) => (m,U) }
  to **Waiting**;
  Add (n,Z) to **Passed**

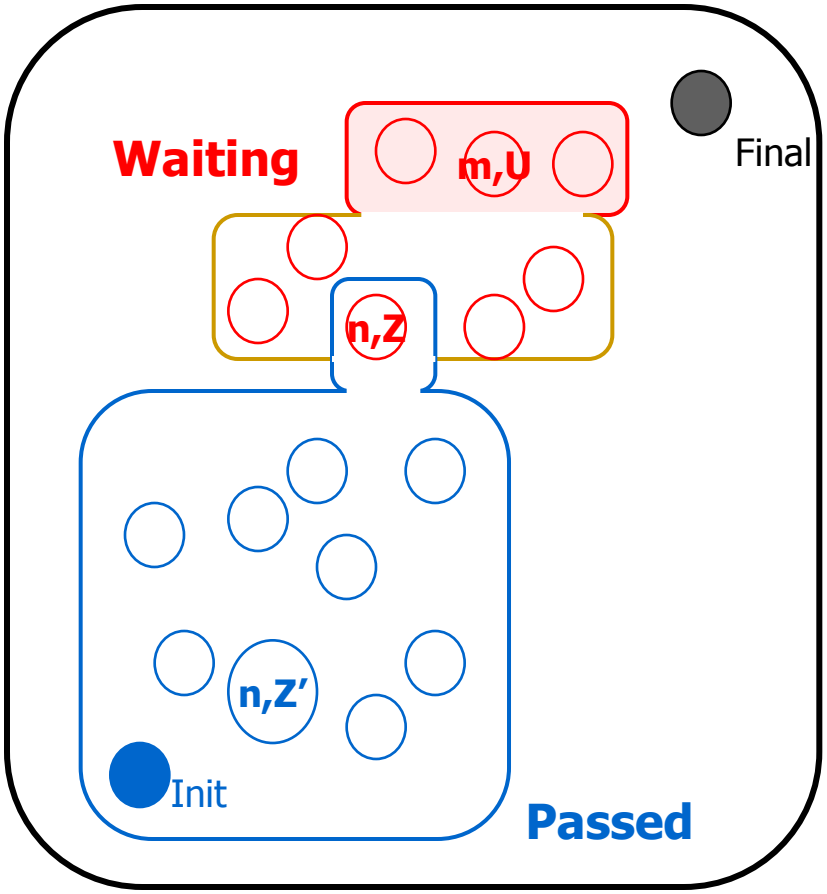
location

zone

UNTIL **Waiting** = Ø
or
Final is in **Waiting**

Order of Exploration

Depth-First vs Breadth-First



Depth-First
Waiting stored on *stack*

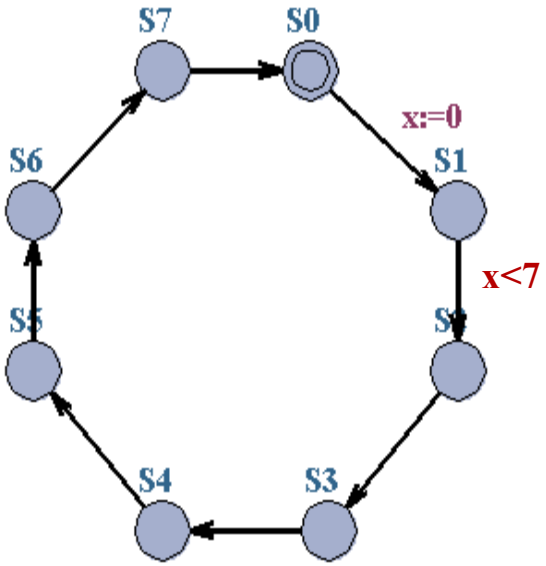
Breadth-First
Waiting stored in *queue*

In most cases **BF** is preferred because it allows for generation of "shortest" traces.

DF is useful in situations when reachability may be concluded without generating the full state-space.
Easy computation of traces.

Representation of Symbolic States

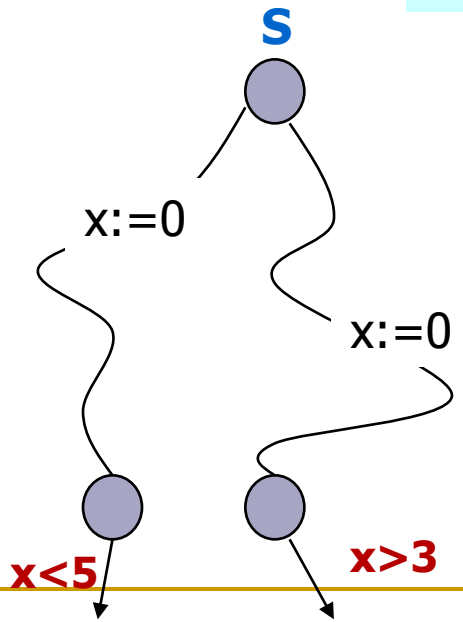
(In)Active Clock Reduction



x is only ***active*** in location **S1**

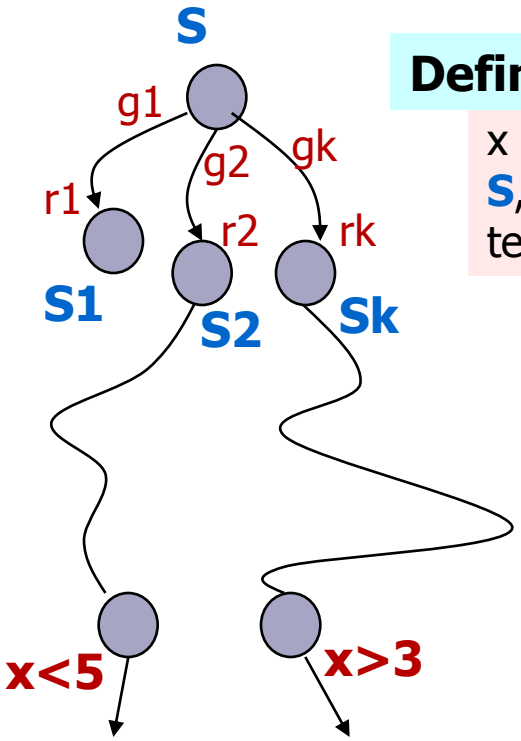
Definition

x is ***inactive*** at **S** if on all paths from **S**, x is always reset before being tested.



Representation of Symbolic States

Active Clock Reduction



Definition

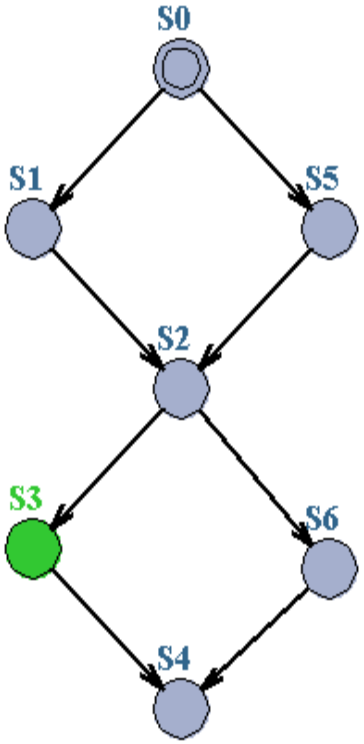
x is ***inactive*** at **S** if on all paths from **S**, x is always reset before being tested.

$$Act(S) = \bigcup_i Clocks(g_i) \cup \bigcup_i \left(Act(S_i) - Clocks(r_i)\right)$$

Idea: We only need to save constraints on active clocks.

When to Store Symbolic State

Global Reduction

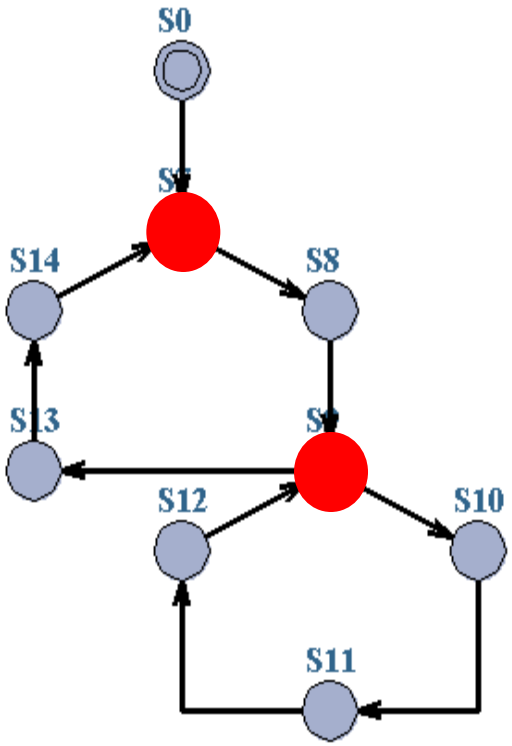


However, the **Passed** list is useful for efficiency.

If there are No Cycles: The **Passed** list is not needed for *termination.*

When to Store Symbolic State

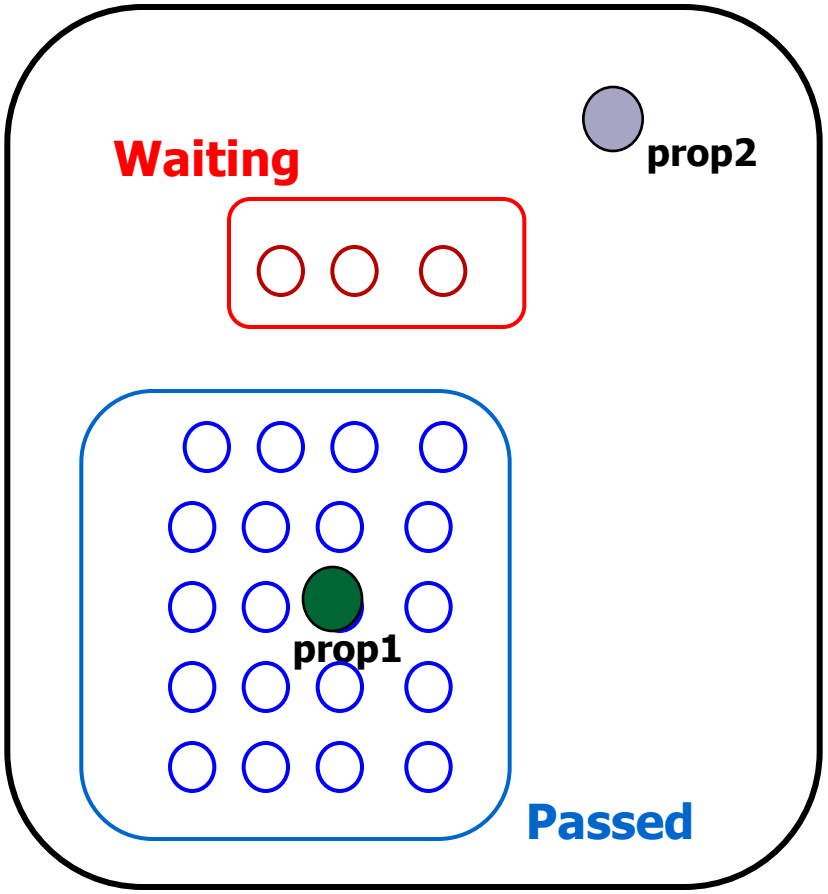
Global Reduction



Cycles:

Only symbolic states involving loop-entry points need to be saved on **Passed** list.

Reuse State Space

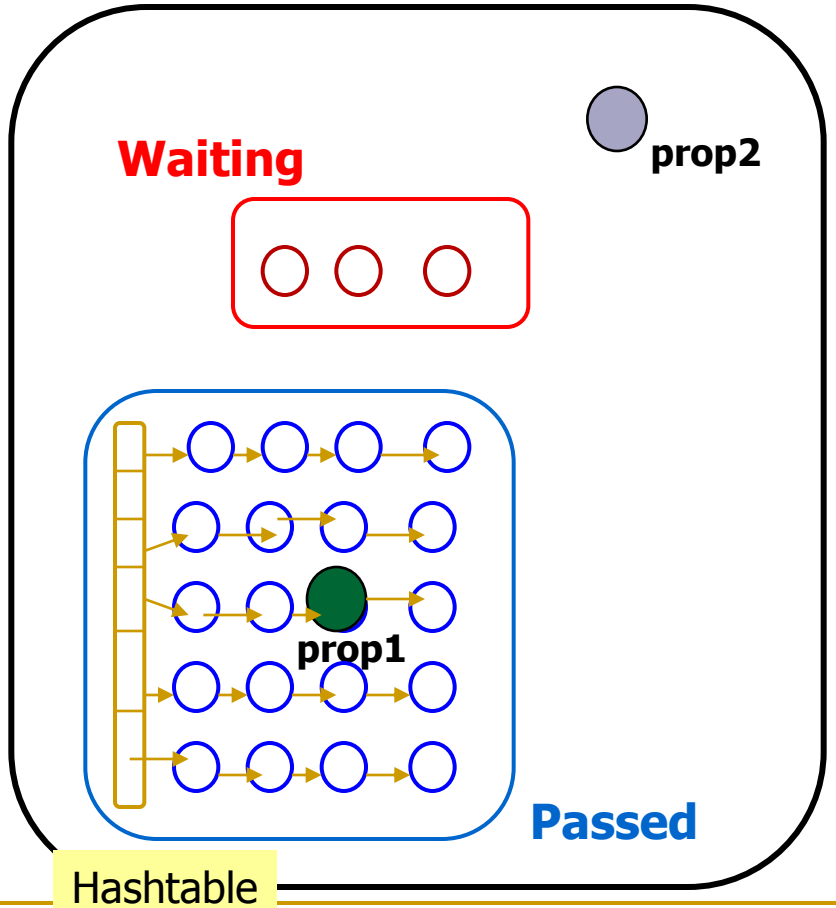



```
A[]  prop1

A[]  prop2
A[]  prop3
A[]  prop4
A[]  prop5
.
.
.
A[]  propn
```

Search in existing **Passed** list before continuing search.

Which order to search?

Reuse State Space



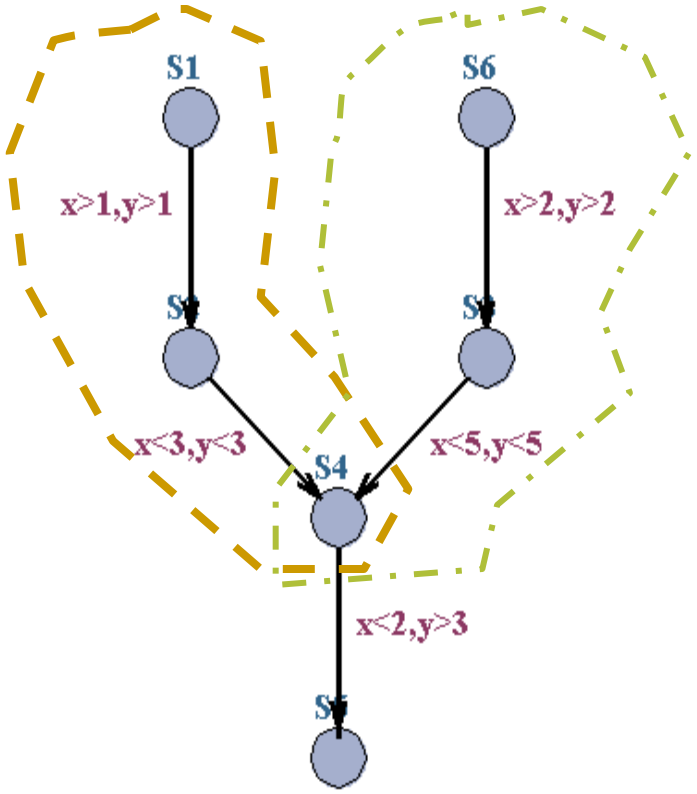

```
A[]  prop1

A[]  prop2
A[]  prop3
A[]  prop4
A[]  prop5
.
.
.
A[]  propn
```

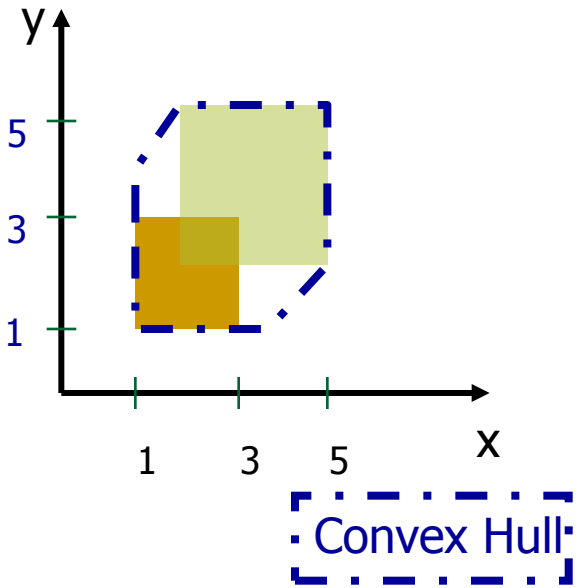
Search in existing **Passed** list before continuing search.

Which order to search?

Over-Approximation

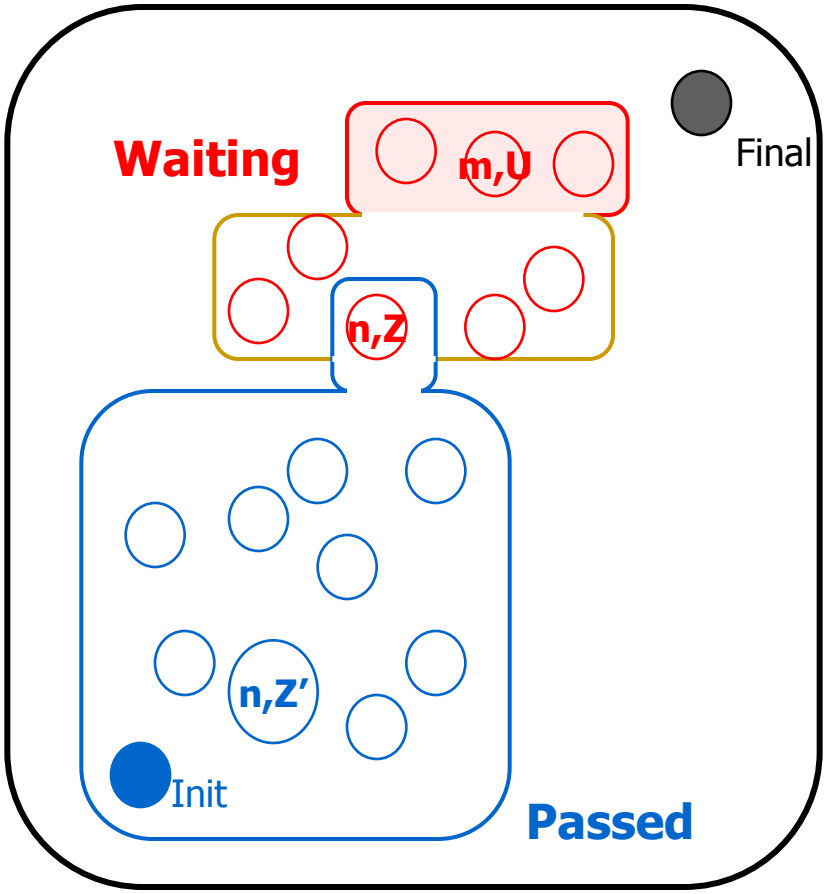
Convex Hull





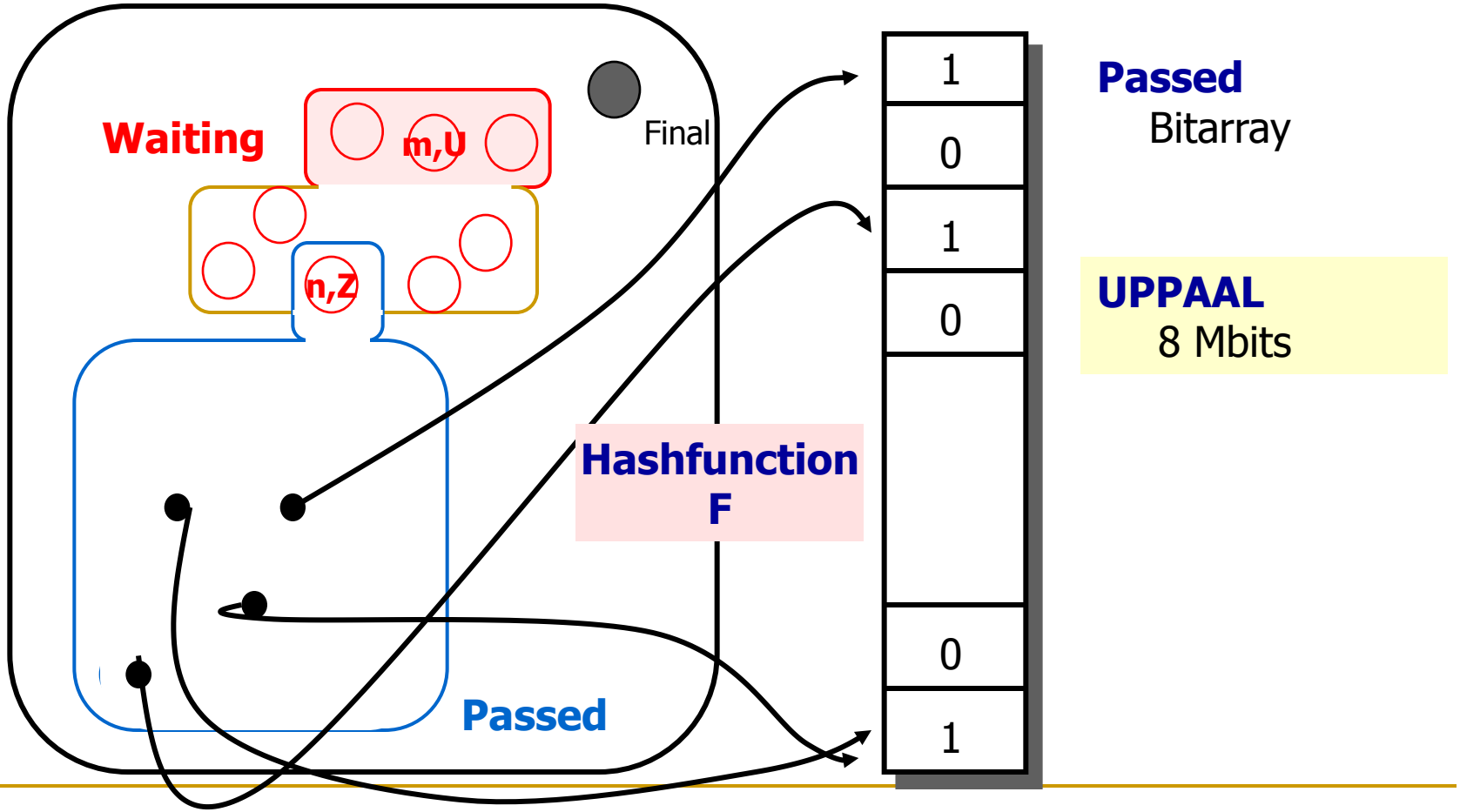
Under-Approximation

Bitstate Hashing

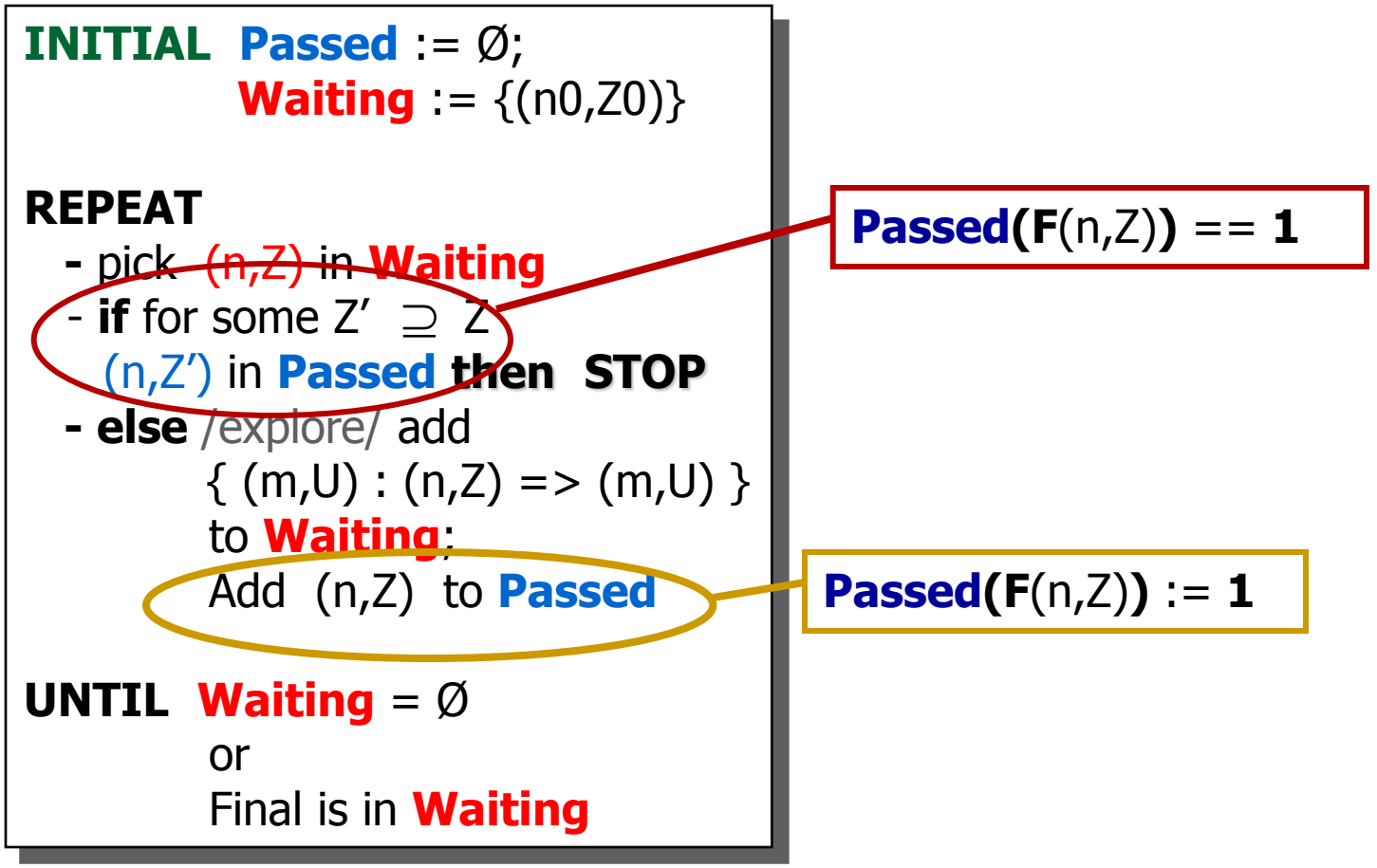


Under-Approximation

Bitstate Hashing



Bitstate Hashing



Options in UPPAAL 3.4.9

- **Search Order:** perform (symbolic) state-space exploration in breadth-first or depth-first order.
- **State Space Reduction:** determines if control-structure analysis should be performed to reduce the space requirements during verification. Possible values are *none*, *conservative* (control-structure reduction saving all non-comitted states and all loop-entry points), and *aggressive* (control-structure reduction saving only loop-entry points). Note that there is normally a tradeoff between space requirement and speed.
- **State Space Representation:** determines how the state-space should be represented in the model checker. Possible values are *DBM* (Difference Bound Matrices), the *Compact Data Structure*, *Under Approximation* (by bit-state hashing), and *Over Approximation* (by convex-hull approximation).

Options in UPPAAL 3.4.9

- **Clock Reduction:** activates (in-)active clock reduction.
- **Reuse State-Space:** instructs, the verifier to (whenever possible) reuse the generated portion of the state-space when several properties of the same system are checked.
- **Diagnostic Trace:** generates a trace (if there is one) that shows how the checked property is (or is not) satisfied. The trace is automatically loaded into the simulator. The possible values are *none*, *some*, *shortest* (to generate an trace with minimum number of transitions), and *fastest* (to generate a trace with the shortest accumulated time delay).

Options in UPPAAL 3.4.9

- **Note:** there are certain dependencies restricting how the various options can be combined. Activating one option may therefore deactivate other options. The Options menu is always updated to show the currently activated options.
- **More Inforation:** control-structure reduction and compact data structure is described in the paper:

  Efficient Verification of Real-Time Systems: Compact Data Structure and State-Space Reduction, Kim G. Larsen, Fredrik Larsson, Paul Pettersson and Wang Yi. In Proceedings of the 18th IEEE Real-Time Systems Symposium, pages 14-24. San Francisco, California, USA, 3-5 December 1997.

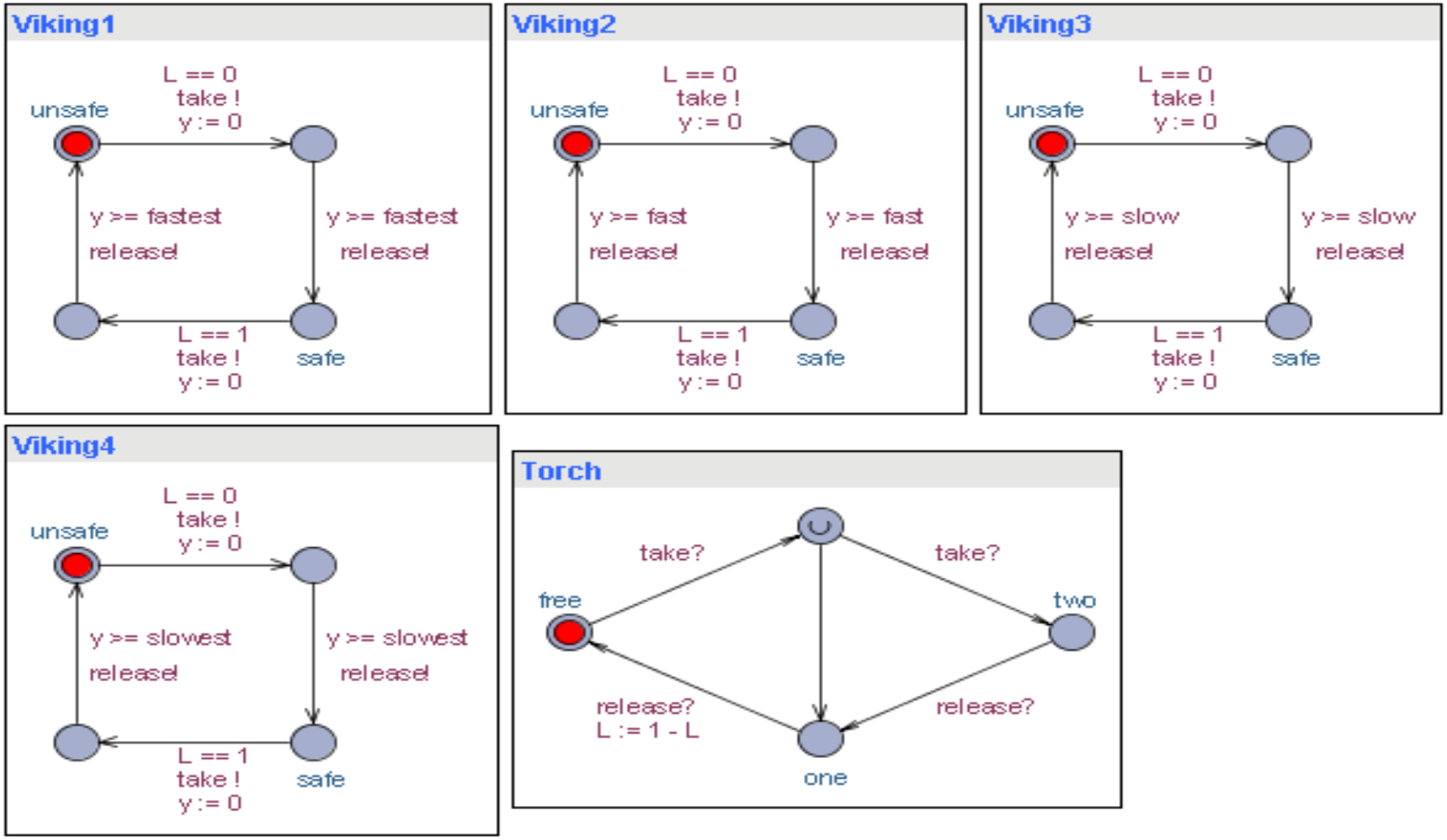
Example: Vikings' Problem

- Four vikings are about to cross a damaged bridge in the middle of the night.
- The bridge can only carry two of the vikings at the time and to find the way over the bridge the vikings need to bring a torch.
- The vikings need 5, 10, 20 and 25 minutes (one-way) respectively to cross the bridge.

- Does a schedule exist which gets all four vikings over the bridge within 60 minutes? What is the minimum time required to get all four vikings over the bridge?

Example: Vikings' Problem Model

- Model the four vikings as four processes, and the torch as a single process (see bridge.xml).



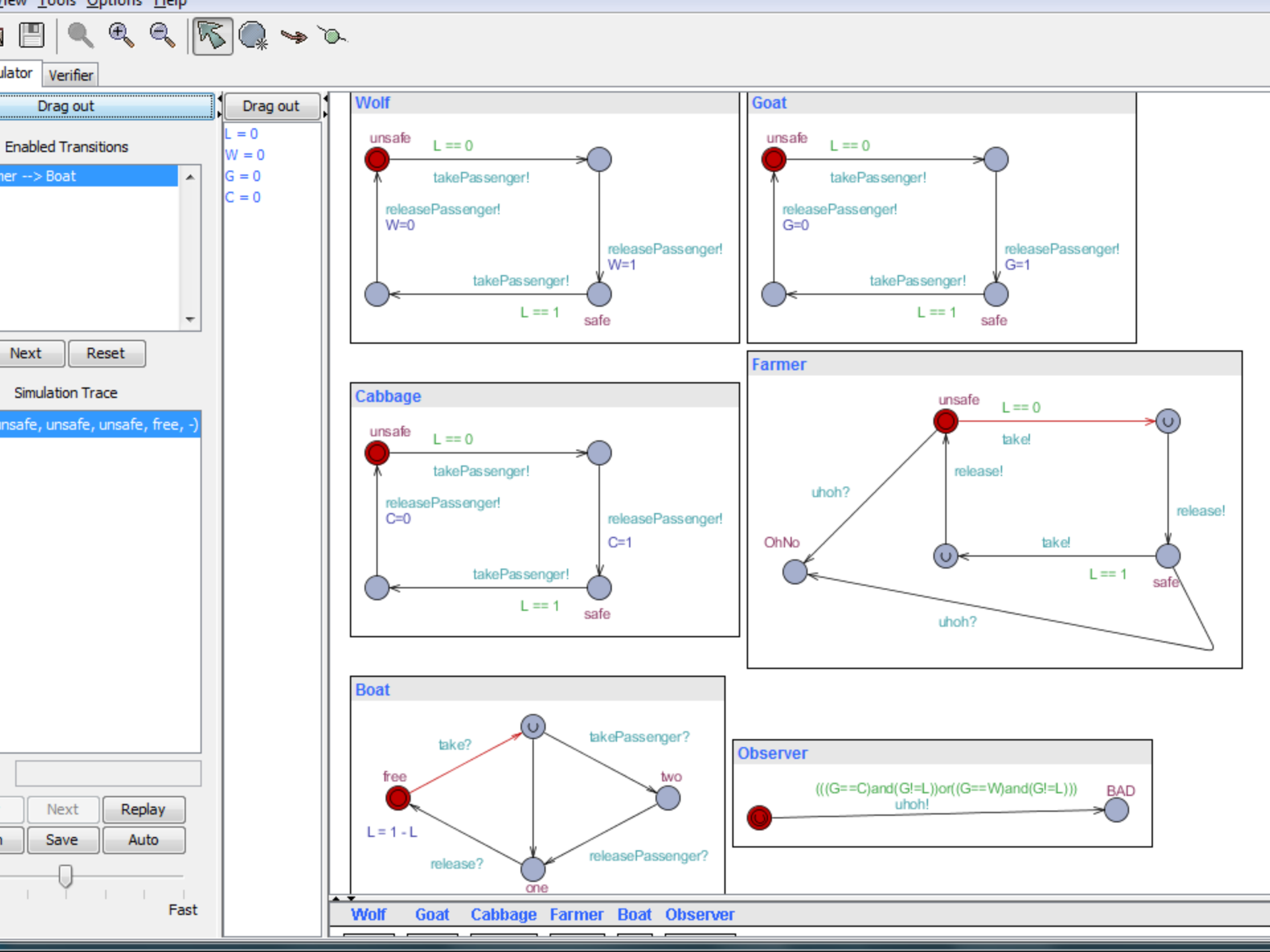
Example: Vikings' Problem

- Four vikings are about to cross a damaged bridge in the middle of the night.
- The bridge can only carry two of the vikings at the time and to find the way over the bridge the vikings need to bring a torch.
- The vikings need 5, 10, 20 and 25 minutes (one-way) respectively to cross the bridge.

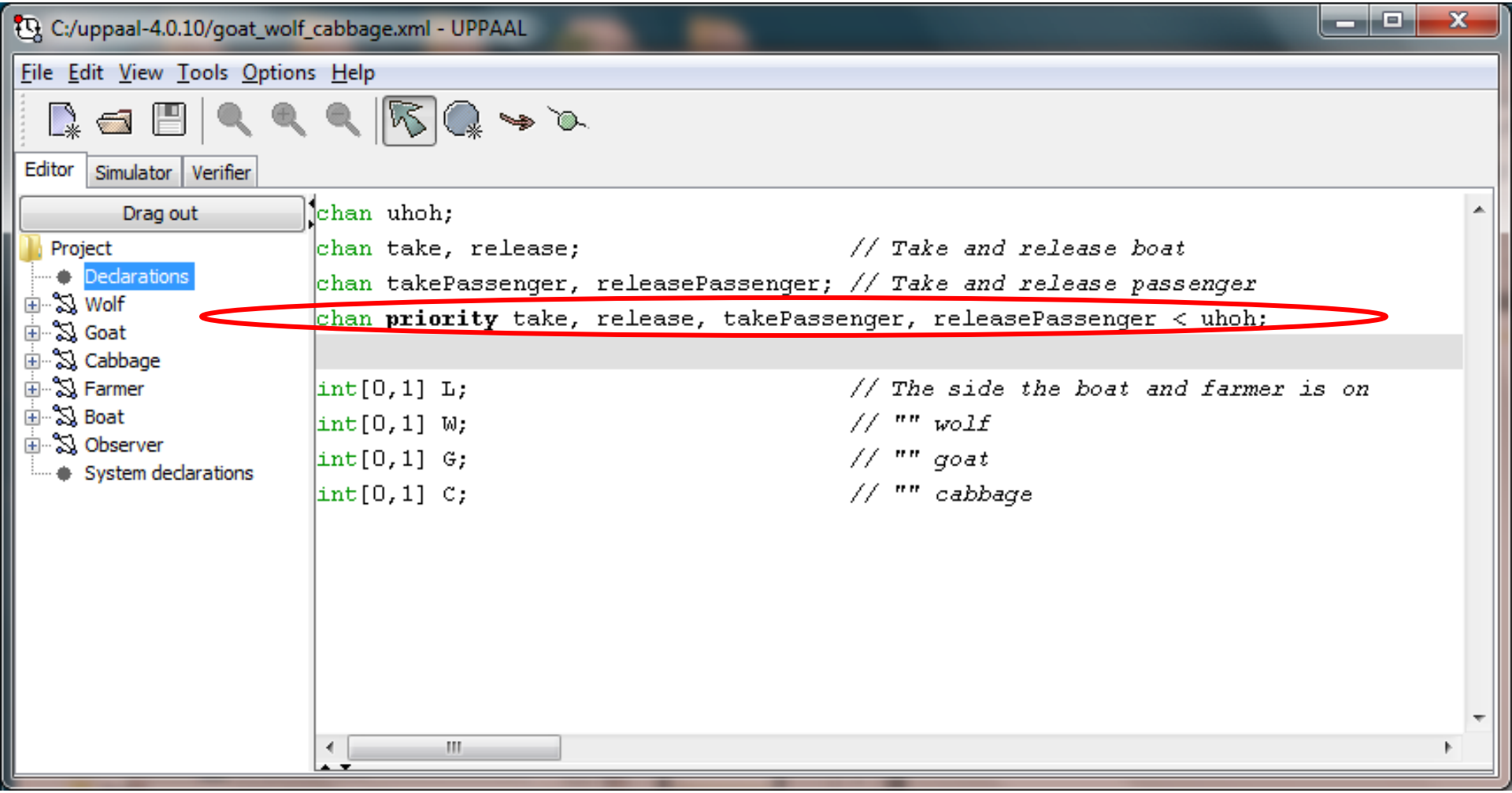
- **Question:** What is the minimum time required for all four vikings to safely cross the bridge?
- **Answer:** 60 minutes: use E<>(Viking1.safe and Viking2.safe and Viking3.safe and Viking4.safe) with the additional Option: **Diagnostic Trace: Fastest.**

UPPAAL Example

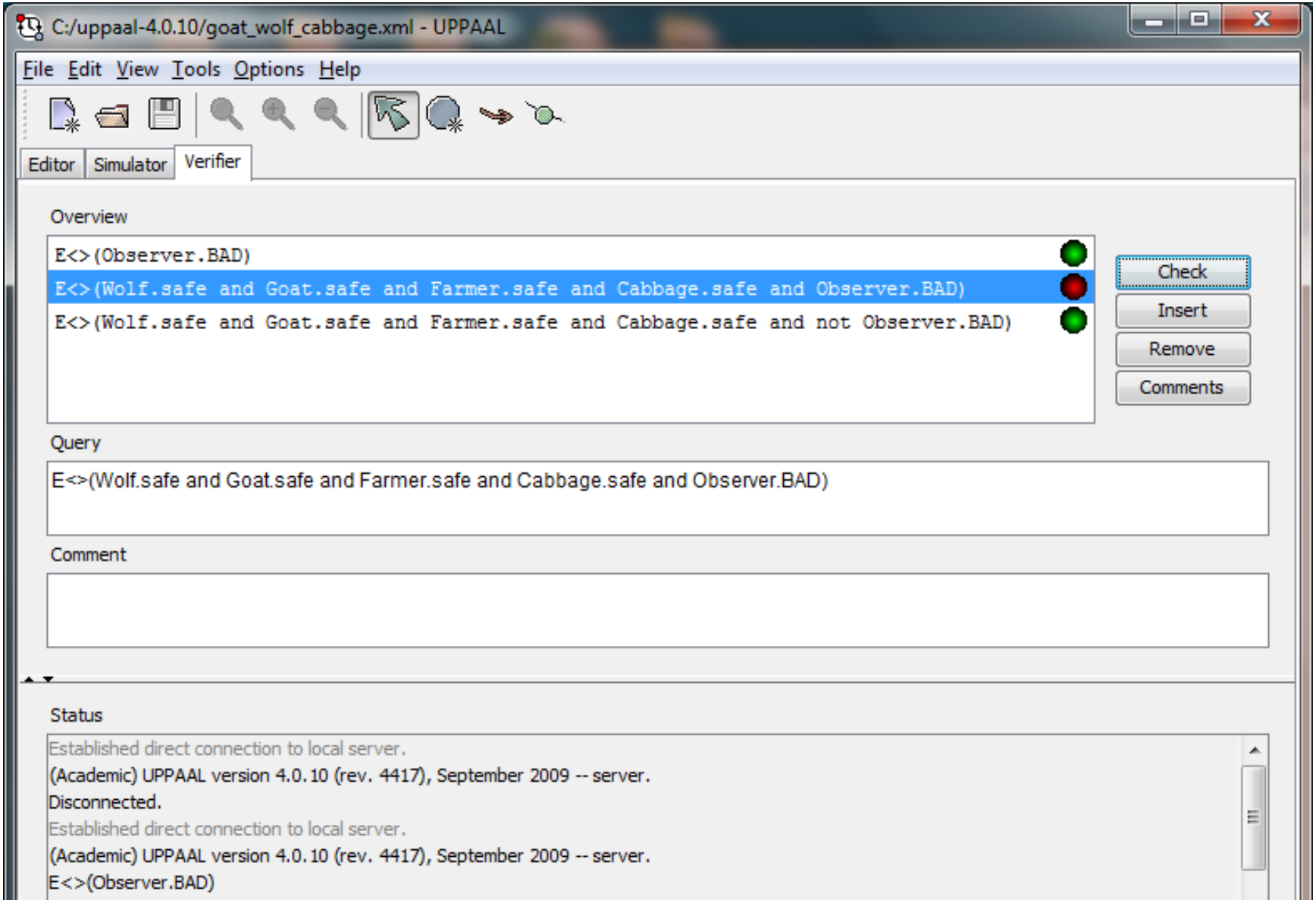
- Wolf, goat, cabbage, farmer problem
  - The farmer needs to move the wolf, goat, and cabbage from one side of the river to the other side. The farmer can only carry one passenger.
  - If the wolf and goat are left alone, the wolf will eat the goat. If the goat and cabbage are left alone, the goat will eat the cabbage.
  - How can the farmer transport the passengers without allowing one to be eaten?


Verifier
Drag out
Enabled Transitions
Boat
Next
Reset
Simulation Trace
Next
Replay
Save
Auto
Fast
Drag out
L = 0
W = 0
G = 0
C = 0
Wolf
unsafe
L == 0
takePassenger!
releasePassenger!
W=0
releasePassenger!
W=1
takePassenger!
L == 1
safe
Goat
unsafe
L == 0
takePassenger!
releasePassenger!
G=0
releasePassenger!
G=1
takePassenger!
L == 1
safe
Cabbage
unsafe
L == 0
takePassenger!
releasePassenger!
C=0
releasePassenger!
C=1
takePassenger!
L == 1
safe
Farmer
unsafe
L == 0
take!
release!
uhoh?
release!
OhNo
take!
L == 1
safe
uhoh?
Boat
take?
takePassenger?
free
two
L = 1 - L
release?
releasePassenger?
one
Observer
(((G==C)and(G!=L))or((G==W)and(G!=L)))
uhoh!
BAD
Wolf
Goat
Cabbage
Farmer
Boat
Observer

UPPAAL Model



UPPAAL Model



Summary

- **Next Time:** SPIN