CIS 721 - Real-Time Systems

Lecture 21: Real-Time Verification

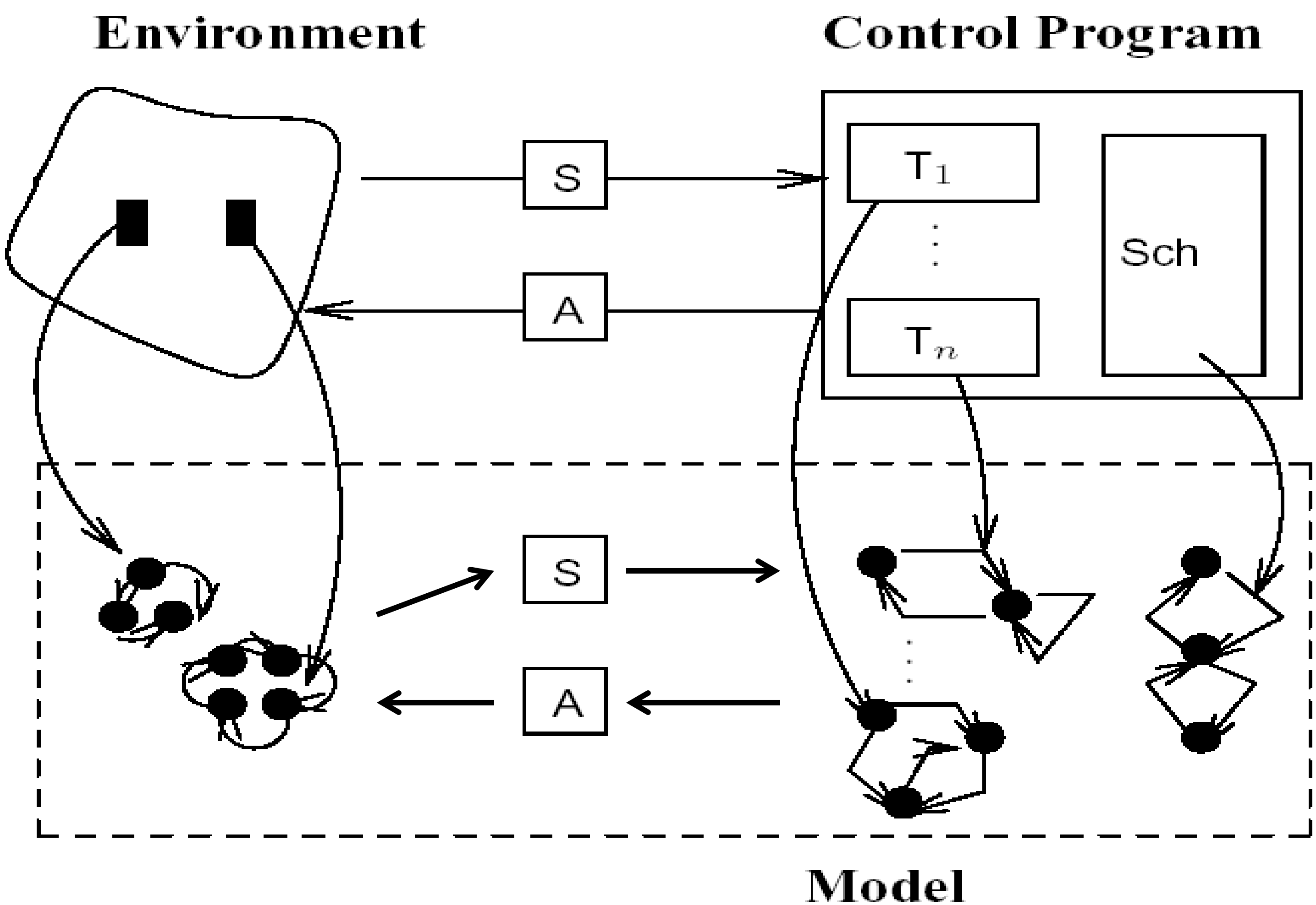
Mitch Neilsen

neilsen@ksu.edu

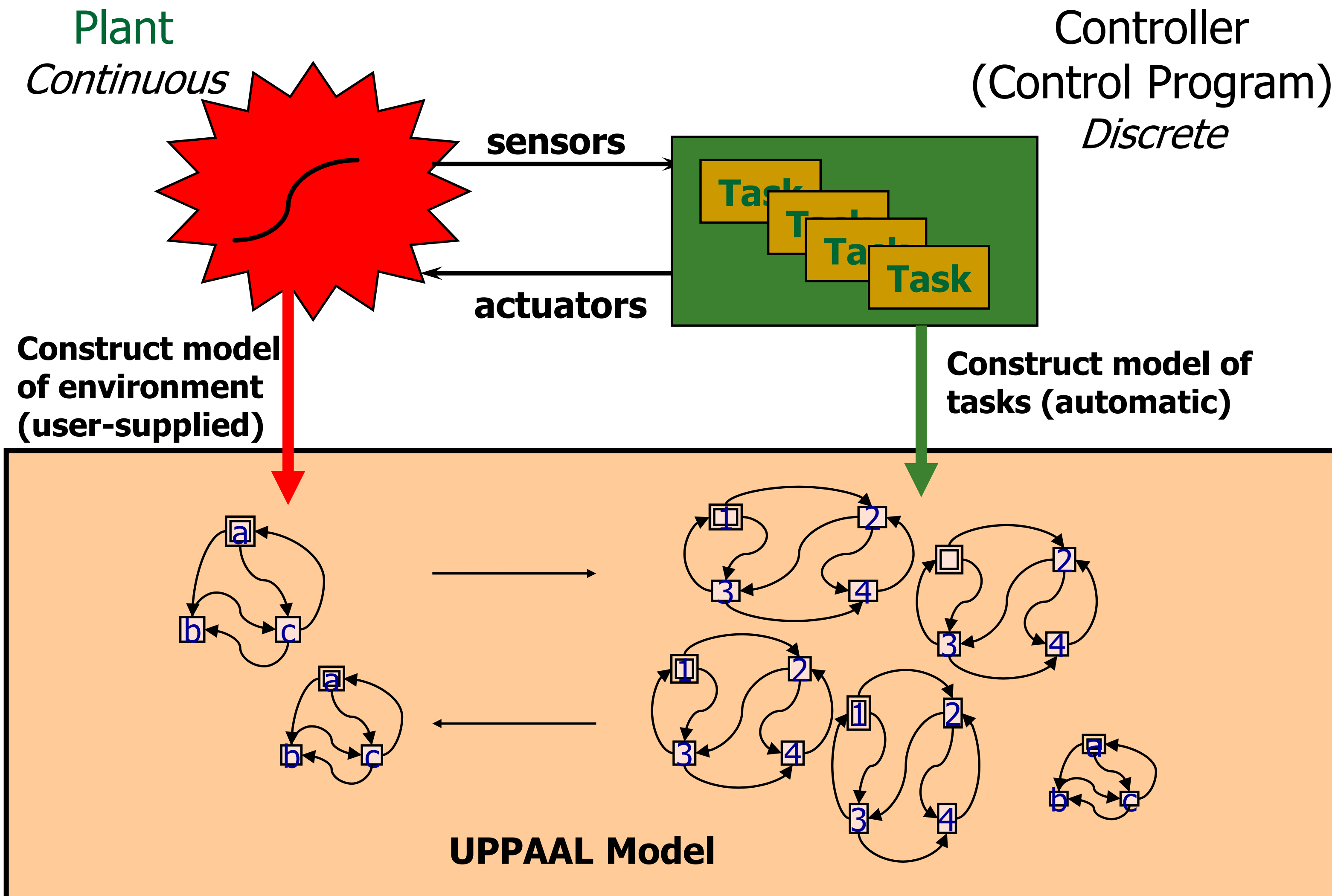
Outline

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

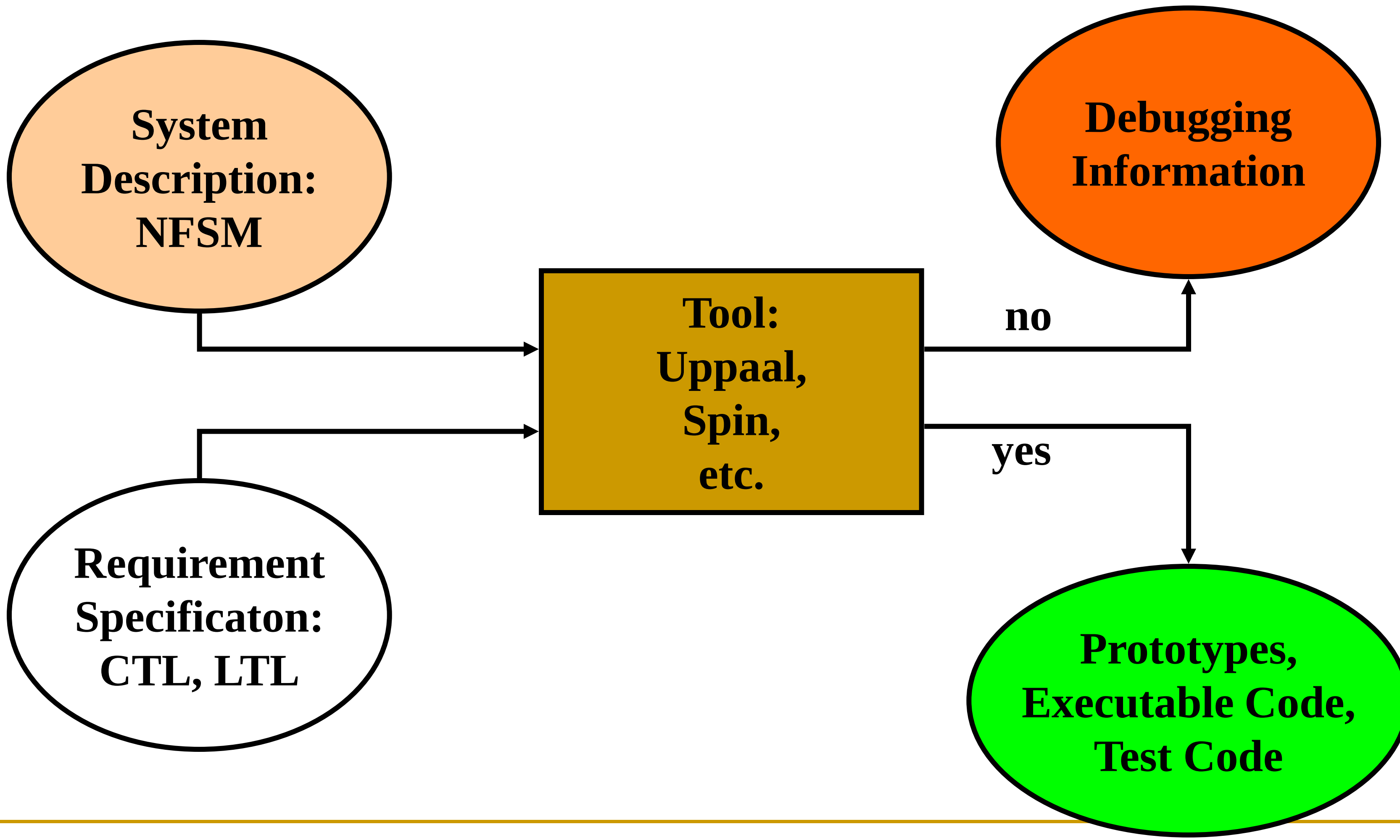
Modelling



UPPAAL Model Construction



Tool Support



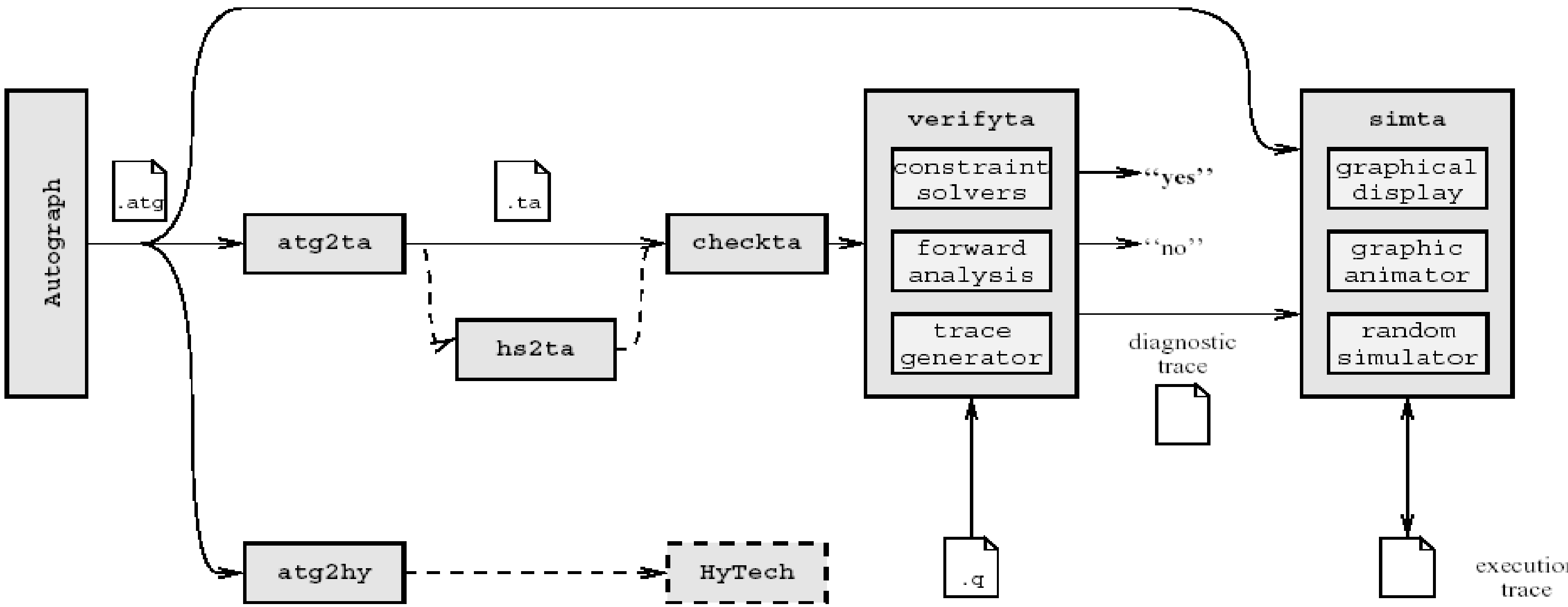
UPPAAL

- **UPPAAL** is a tool box for simulation and verification of real-time systems based on constraint solving and other techniques.
- UPPAAL was developed jointly by **Upp**sala University and **Aal**borg University.
- It can be used for systems that are modeled as a collection of non-deterministic processes w/ finite control structures and real-valued clocks, communicating through channels and/or shared variables.
- It is designed primarily to check both **invariants** and **reachability properties** by exploring the statespace of a system.

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)



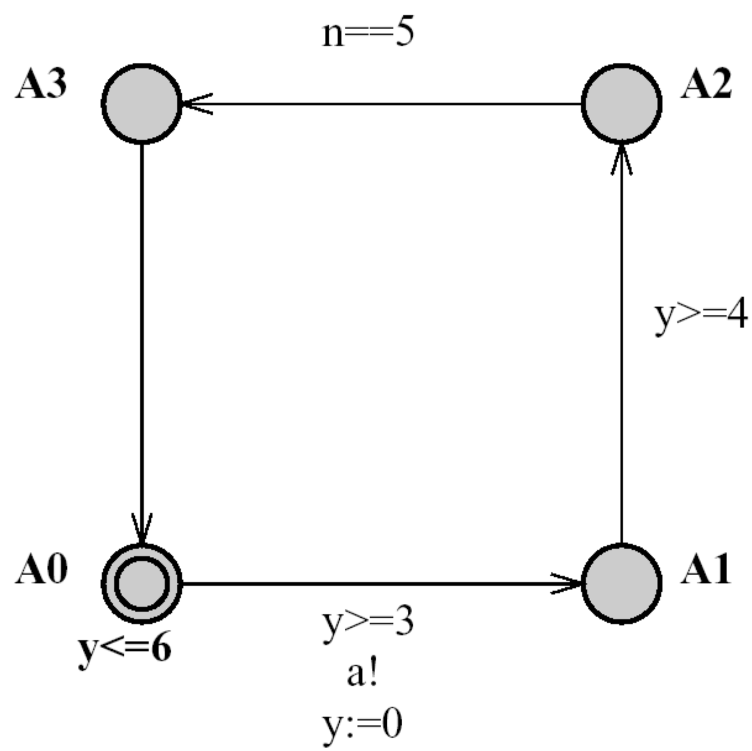
- **checkta** – syntax checker
- **simta** – simulator
- **verifyta** – model checker

Example – .xta file format

(from UPPAAL in a Nutshell)

```
clock x, y;
int n;
chan a;

process A {
  state A0 { y<=6 }, A1, A2, A3;
  init A0;
  trans A0 -> A1 {
    guard y>=3;
    sync a!;
    assign y:=0;
  },
  A1 -> A2 {
    guard y>=4;
  },
  A2 -> A3 {
  guard n==5;
  }, A3 -> A0;
}
```

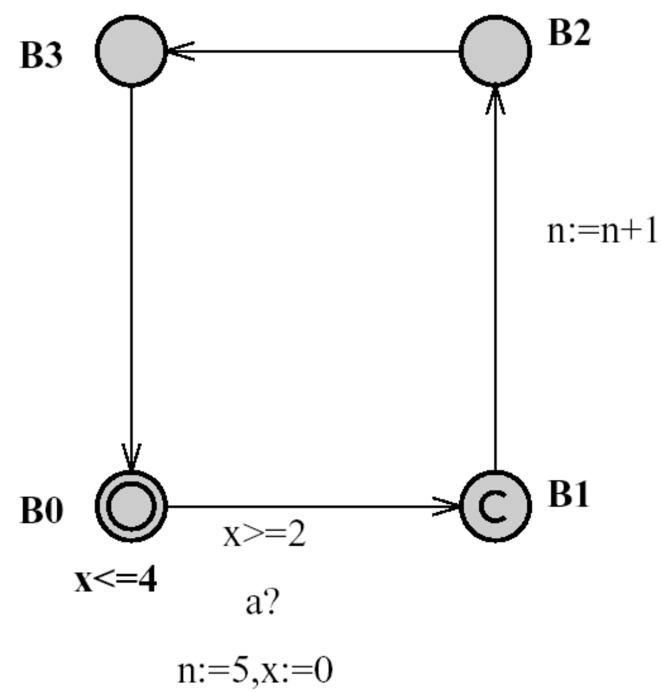


Example (cont.)

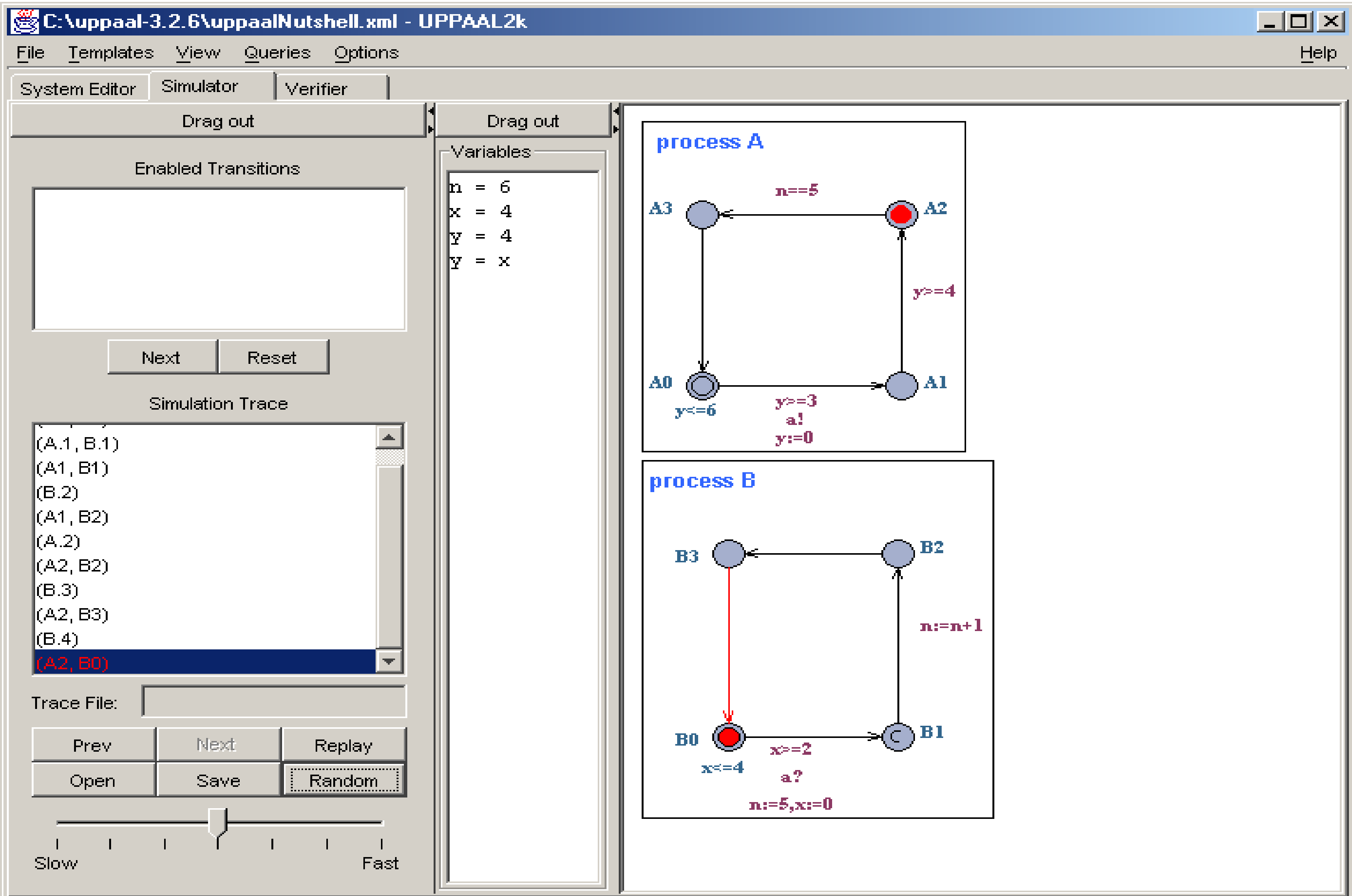
(from UPPAAL in a Nutshell)

```
process B {
  state B0 { x<=4 }, B1, B2, B3;
  commit B1;
  init B0;
  trans B0 -> B1 {
    guard x>=2;
    sync a?;
    assign n:=5,x:=0;
  },
  B1 -> B2 {
    assign n:=n+1;
  },
  B2 -> B3 {
  }, B3 -> B0;
}

system A, B;
```



Example (cont.)



Linear Temporal Logic (LTL)

- LTL formulae are used to specify temporal properties.
- LTL includes propositional logic and temporal operators:
  - **[ ]P =** always P
  - **<>P =** eventually P
  - **P U Q =** P is true until Q becomes true
- **Examples:**
  - Invariance: **[ ] (p)**
  - Response: **[ ] ((p) -> (<> (q)))**
  - Precedence: **[ ] ((p) -> ((q) U (r)))**
  - Objective: **[ ] ((p) -> <>((q) || (r)))**

Labels and Transitions

- The edges of the automata can be labeled with three different types of labels:
  - a **guard** expressing a condition on the values of clocks and integer variables that must be satised in order for the edge to be taken,
  - a **synchronization action** which is performed when the edge is taken, and
  - a **number of clock resets and assignments** to integer variables.
- Nodes may be labeled with **invariants**; that is, conditions expressing constraints on the clock values in order for control to remain in a particular node.

Committed Locations

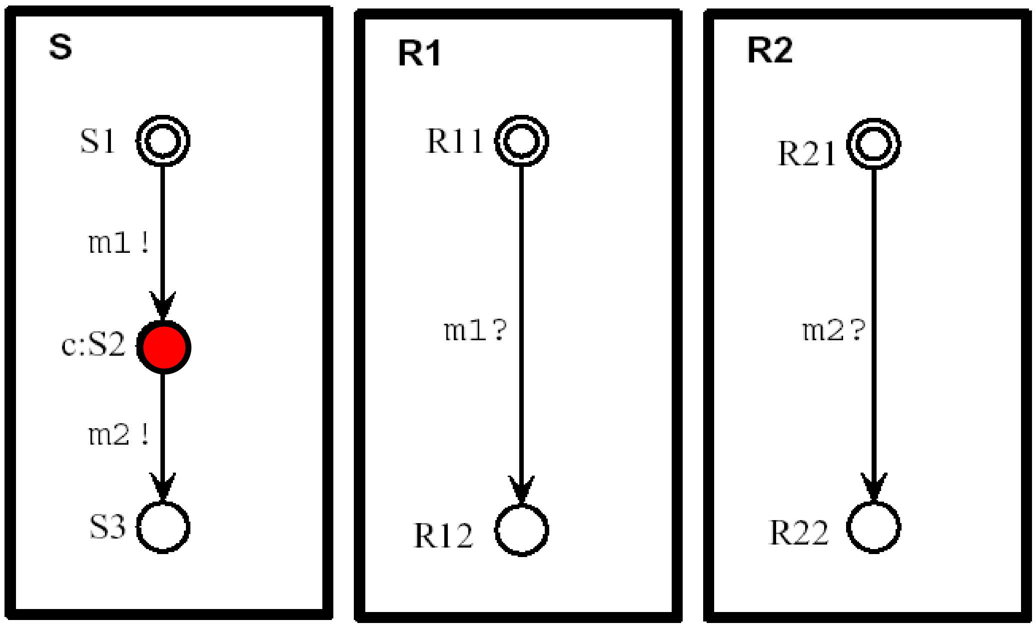
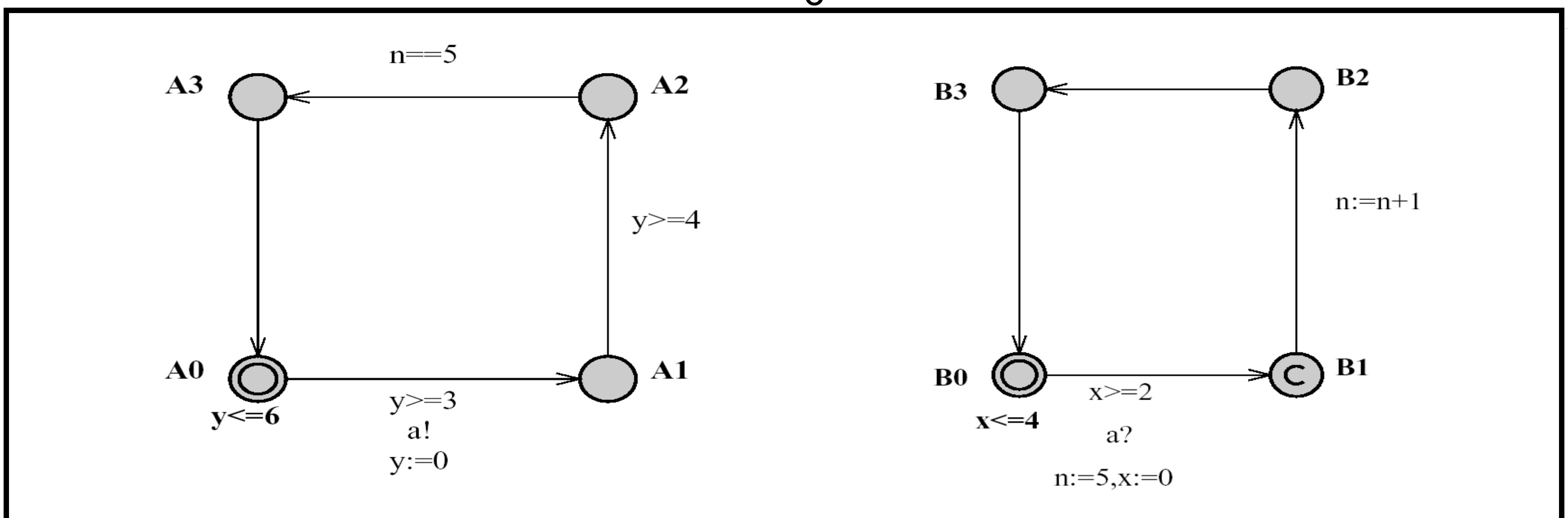


Fig. 4. Broadcasting Communication and Committed Locations.

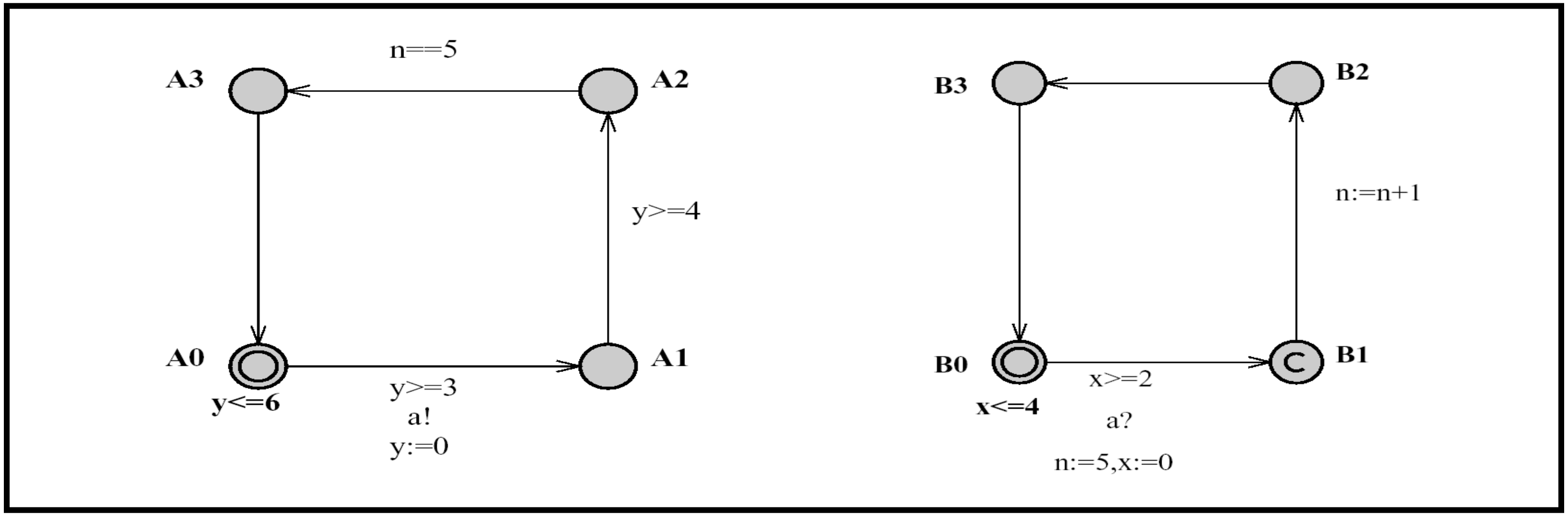
A **committed location** must be left immediately.
A broadcast can be represented by two transitions with a committed state between sends.

Transitions

- **Delay transitions** – if none of the invariants of the nodes in the current state are violated, time may progress without making a transition; e.g., from $((A_0,B_0),x=0,y=0,n=0)$, time may elapse 3.5 units to $((A_0,B_0),x=3.5,y=3.5,n=0)$, but time cannot elapse 5 time units because that would violate the invariant on B_0.



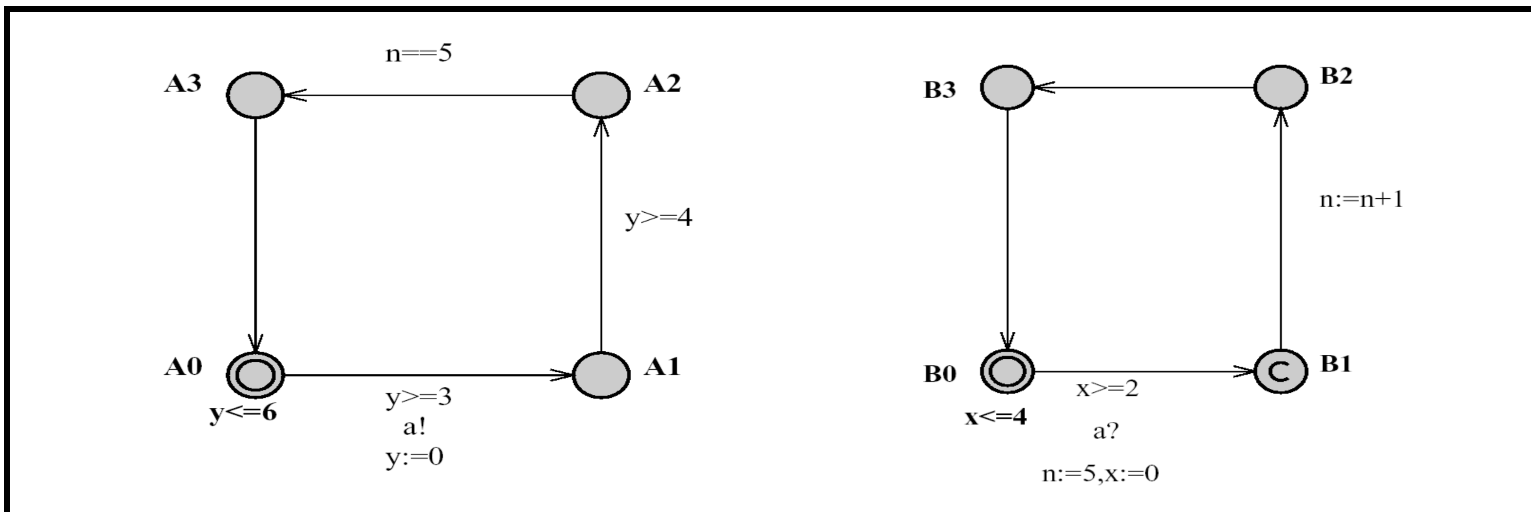
Transitions (cont.)



- **Action transitions** – if two complementary edges of two different components are enabled in a state, then they can synchronize; also, if a component has an enabled internal edge, the edge can be taken without any synchronizaton; e.g., from $((A_0,B_0),x=3.5,y=3.5,n=0)$ the two components can synchronize to $((A_1,B_1),x=0,y=0,n=5)$.

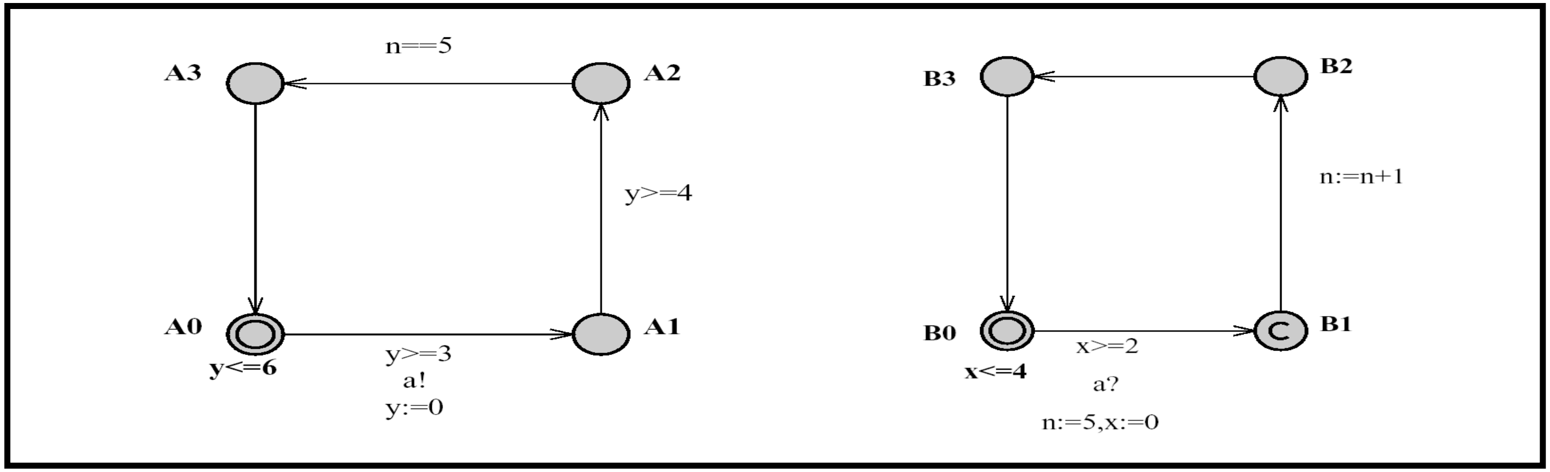
Urgent Channels and Committed Locations

- Transitions can be overruled by the presence of **urgent channels** and **committed locations:**
  - When two components can synchronize on an **urgent channel,** no further delay is allowed; e.g., if channel a is urgent, time can not elapse beyond 3, because in state $((A_0,B_0),x=3,y=3,n=0)$, synchronization on channel a is enabled.



Committed Nodes

- Transitions can be overruled by the presence of **urgent channels** and **committed locations:**

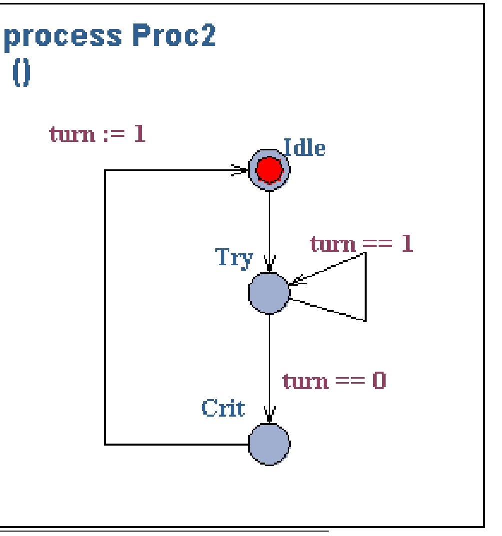


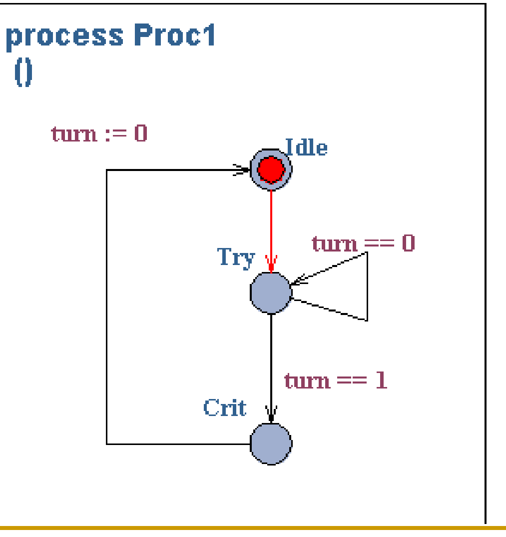
  - If one of the components is in a **committed node**, no delay is allowed to occur and any action transition must involve the component committed to continue; e.g., in state $((A_1,B_1),x=0,y=0,n=5)$, B_1 is committed, so the next state of the network is $((A_1,B_2),x=0,y=0,n=6)$.

Translation to UPPAAL

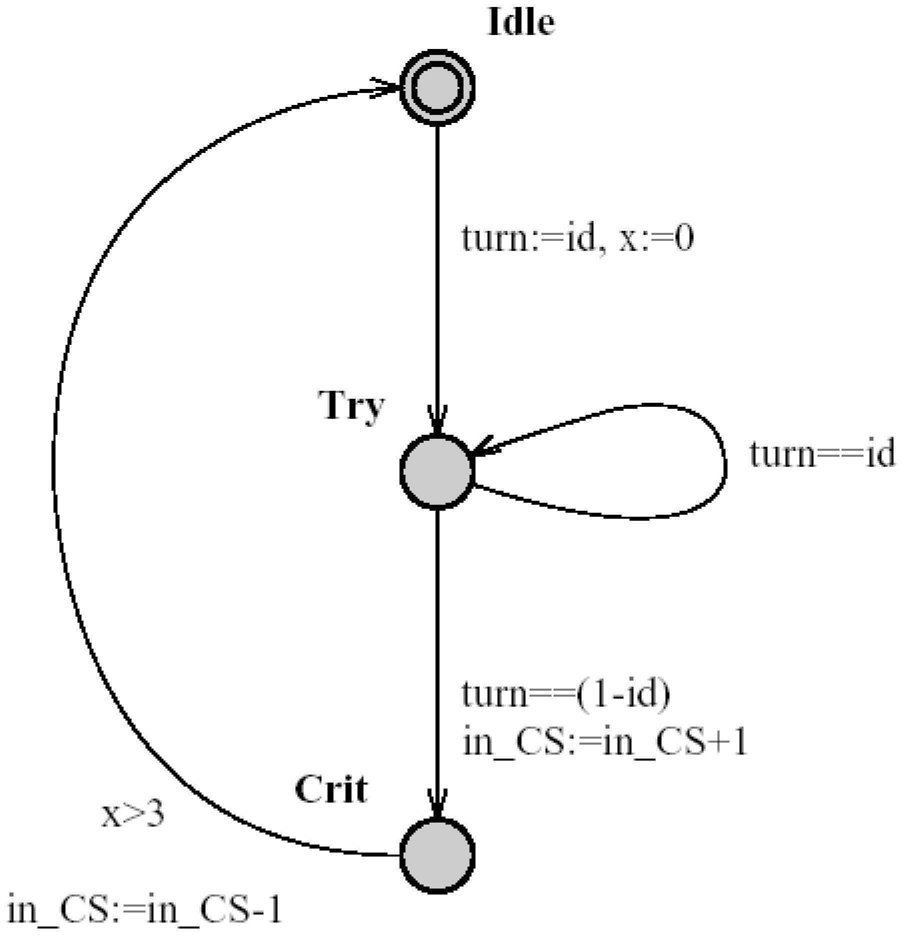
```
P1 :: while True do
        T1 : wait(turn=1)
        C1 : turn:=0
        endwhile
||
P2 :: while True do
        T2 : wait(turn=0)
        C2 : turn:=1
        endwhile
```

Mutual Exclusion Program

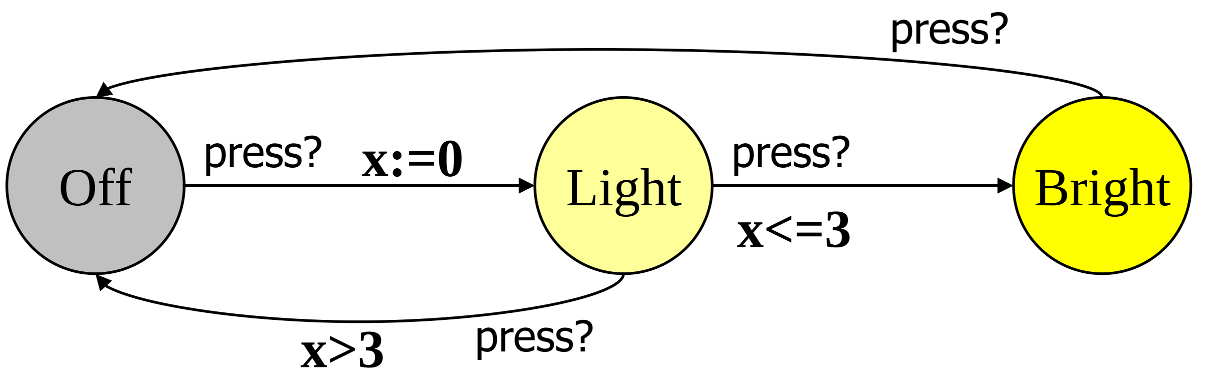




Example: Mutual Exclusion

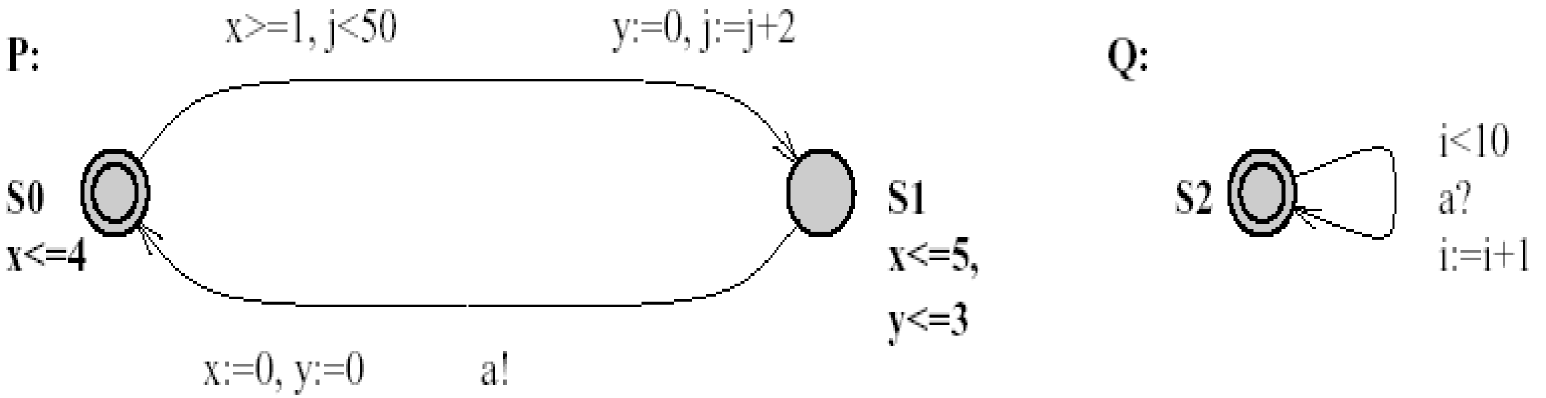


Intelligent Light Control



- **Requirements:** If a user **quickly** presses the light control twice, then the light should get brighter; on the other hand, if the user **slowly** presses the light control twice, the light should turn off.
- **Solution:** Add a real-valued clock, x.

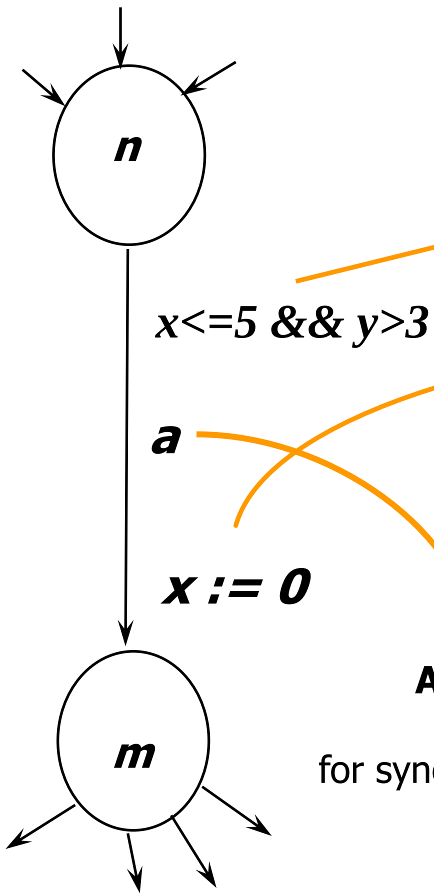
UPPAAL Model =
Networks of Timed Automata



A **timed automaton** is a standard finite state automaton extended with a finite collection of real-valued clocks.

Timed Automata

Alur & Dill 1990



Action
used
for synchronization

Clocks: x, y

Guard
Boolean combination of comp with integer bounds

Reset
Action perfomed on clocks

State

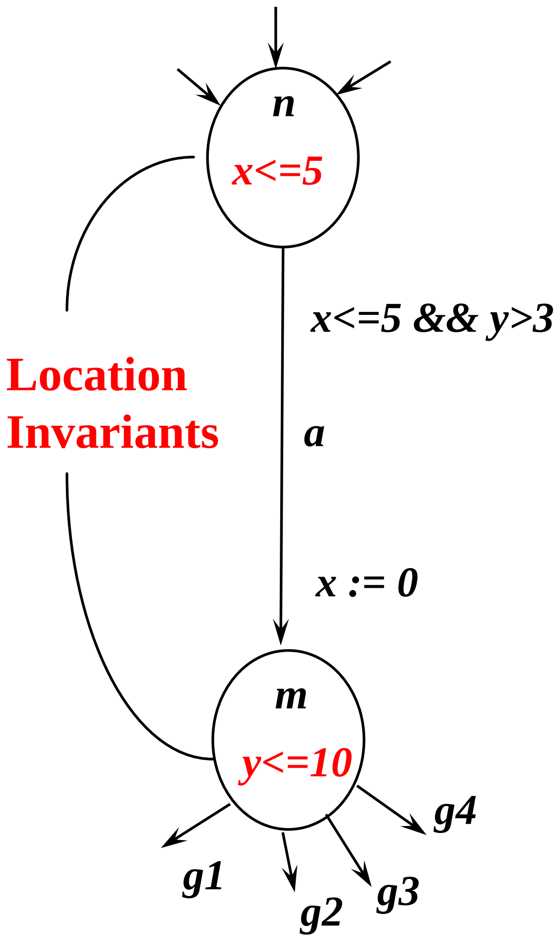
(***location*** , ***x***=v , ***y***=u)   where v,u are in $\mathbf{R}$

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 - Invariants



Clocks: ***x, y***

Transitions

(***n*** , *x*=2.4 , *y*=3.1415) $\not\xrightarrow{e(3.2)}$

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

Invariants ensure progress.

A simple program

```
Int x

Process P
        do
        :: x<2000 → x:=x+1
        od

Process Q
        do
        :: x>0 → x:=x-1
        od

Process R
        do
        :: x=2000 → x:=0
        od

fork P; fork Q; fork R
```

What are possible values for x?

Questions/Properties:

E<>(x>1000)
E<>(x>2000)
A[](x<=2000)
E<>(x<0)
A[](x>=0)

Possible

Always

Verification (example.xta)

```
int x:=0;
process P{
state S0;
init S0;
trans S0 -> S0{guard x<2000; assign x:=x+1; };
}
process Q{
state S1;
init S1;
trans S1 -> S1{guard x>0; assign x:=x-1; };
}
process R{
state S2;
init S2;
trans S2 -> S2{guard x==0; assign x:=0; };
}
p1:=P();
q1:=Q();
r1:=R();
system p1,q1,r1;
```

Int x

Process P
do
:: x<2000 → x:=x+1
od

Process Q
do
:: x>0 → x:=x-1
od

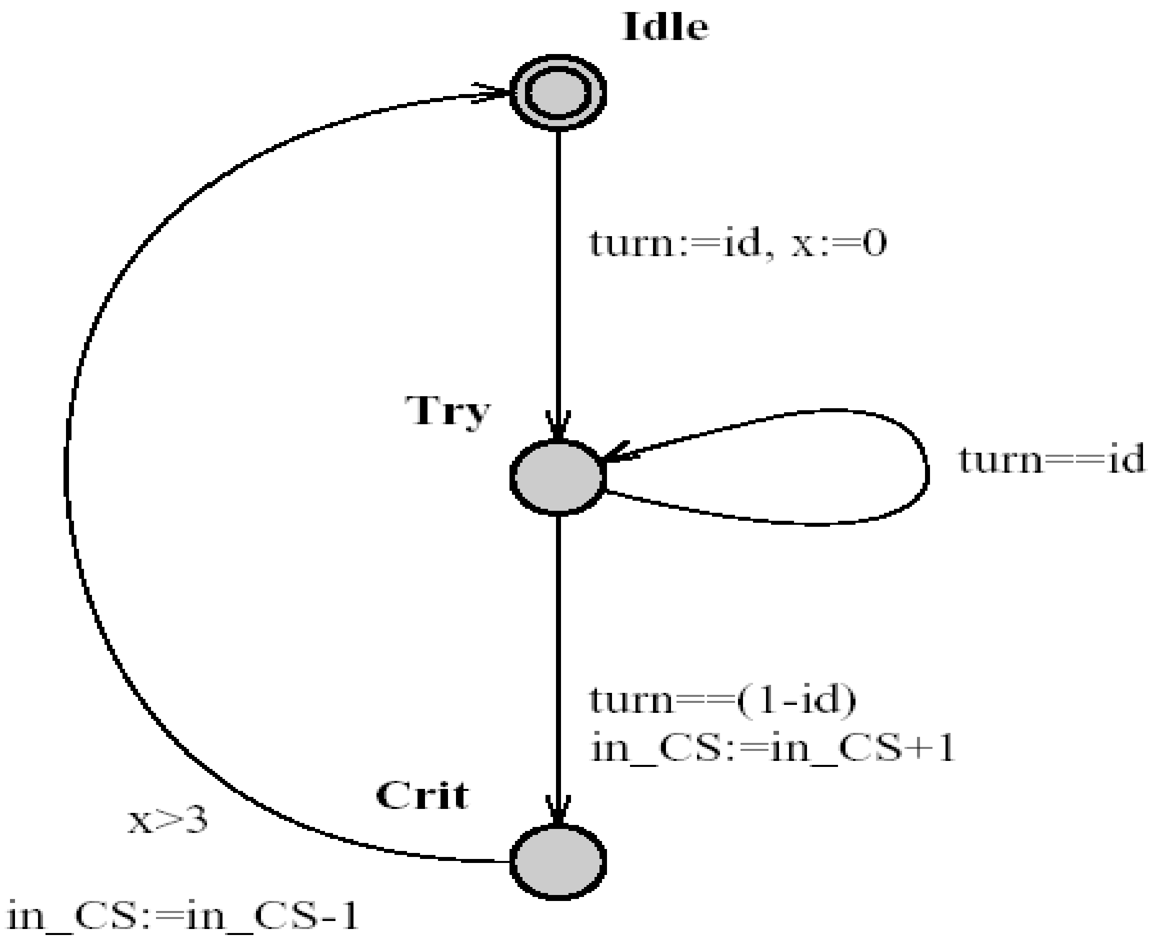
Process R
do
:: x=2000 → x:=0
od

fork P; fork Q; fork R

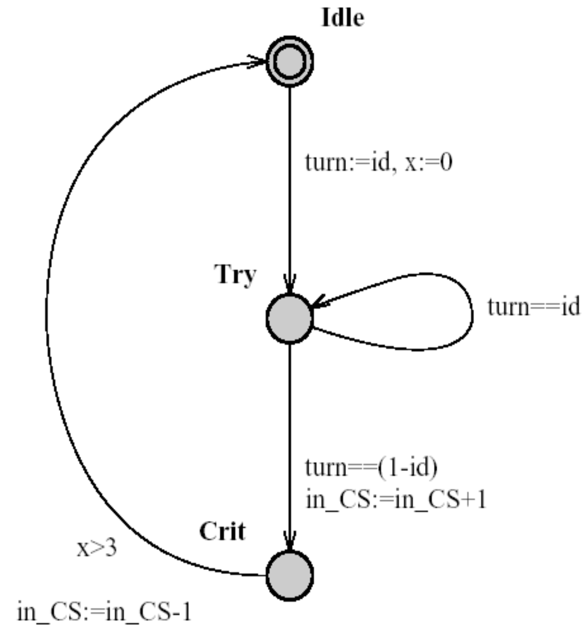
Appendix B: BNF for q-format

$$
\begin{array}{lcl}
\mathit{Prop} & \rightarrow & \texttt{E<>}\ \mathit{StateProp} \mid \texttt{A[]}\ \mathit{StateProp} \\
\mathit{StateProp} & \rightarrow & \mathit{AtomicProp} \mid \texttt{(}\ \mathit{StateProp}\ \texttt{)} \\
 & & \mid \texttt{not}\ \mathit{StateProp} \\
 & & \mid \mathit{StateProp}\ \texttt{or}\ \mathit{StateProp} \\
 & & \mid \mathit{StateProp}\ \texttt{and}\ \mathit{StateProp} \\
 & & \mid \mathit{StateProp}\ \texttt{imply}\ \mathit{StateProp} \\
\mathit{AtomicProp} & \rightarrow & \mathit{Id}\texttt{.}\mathit{Id} \mid \mathit{Id}\ \mathit{RelOp}\ \mathit{Nat} \\
 & & \mid \mathit{Id}\ \mathit{RelOp}\ \mathit{Id}\ \mathit{Op}\ \mathit{Nat} \\
\mathit{RelOp} & \rightarrow & \texttt{<} \mid \texttt{<=} \mid \texttt{>=} \mid \texttt{>} \mid \texttt{==} \\
\mathit{Op} & \rightarrow & \texttt{+} \mid \texttt{-} \\
\mathit{Id} & \rightarrow & \mathit{Alpha} \mid \mathit{Id}\ \mathit{AlphaNum} \\
\mathit{Nat} & \rightarrow & \mathit{Num} \mid \mathit{Num}\ \mathit{Nat} \\
\mathit{Alpha} & \rightarrow & \texttt{A} \mid \texttt{...} \mid \texttt{Z} \mid \texttt{a} \mid \texttt{...} \mid \texttt{z} \\
\mathit{Num} & \rightarrow & \texttt{0} \mid \texttt{...} \mid \texttt{9} \\
\mathit{AlphaNum} & \rightarrow & \mathit{Alpha} \mid \mathit{Num} \mid \texttt{\_}
\end{array}
$$

Example: Mutual Exclusion



Example (mutex2.xta)



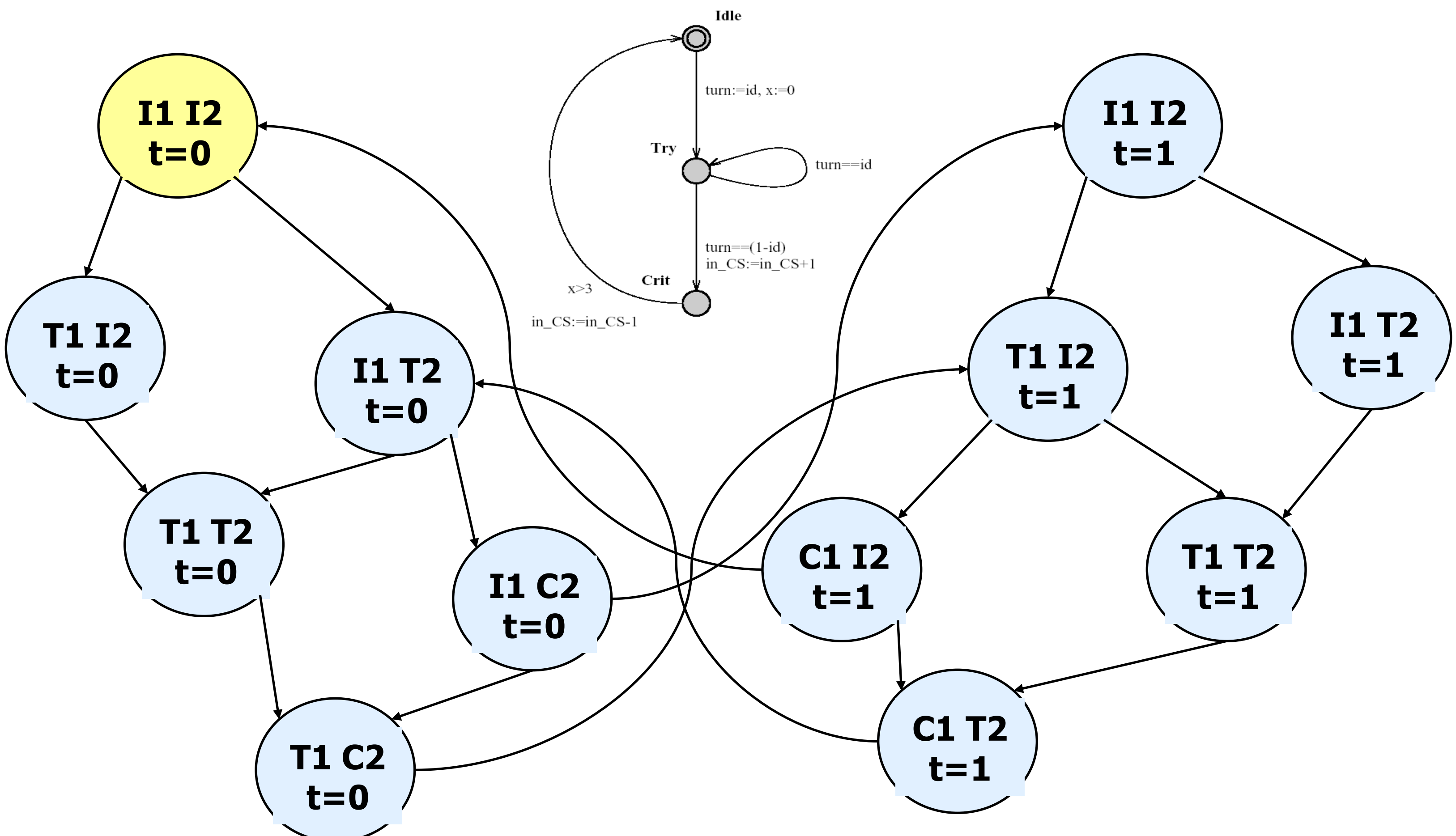

```
//Global declarations
int turn;
int in_CS;

//Process template
process P(const id){
clock x;
state Idle, Try, Crit;
init Idle;
trans Idle -> Try{assign turn:=id, x:=0; },
Try -> Crit{guard turn==(1-id); assign in_CS:=in_CS+1; },
Try -> Try{guard turn==id; },
Crit -> Idle{guard x>3; assign in_CS:=in_CS-1; };
}

//Process assignments
P1:=P(1);
P2:=P(0);

//System definition.
system P1, P2;
```

From UPPAAL$_{-time}$ Models to *Kripke Structures*



CTL Models

A CTL-model is a triple $\mathcal{M} = (S, R, Label)$ where

- S is a non-empty set of states,
- $R \subseteq S \times S$ is a total relation on S, which relates to $s \in S$ its possible successor states,
- $Label : S \longrightarrow 2^{AP}$, assigns to each state $s \in S$ the atomic propositions $Label(s)$ that are valid in s.

Computation Tree Logic, CTL

(Clarke and Emerson, 1980)

Syntax

$$\boxed{\phi \quad ::= \quad p \mid \neg\phi \mid \phi \vee \phi \mid \mathsf{EX}\,\phi \mid \mathsf{E}\,[\phi\,\mathsf{U}\,\phi] \mid \mathsf{A}\,[\phi\,\mathsf{U}\,\phi].}$$

- EX (pronounced "for some path next")
- E (pronounced "for some path")
- A (pronounced "for all paths") and
- U (pronounced "until").

Example

(from UPPAAL2k: Small Tutorial)

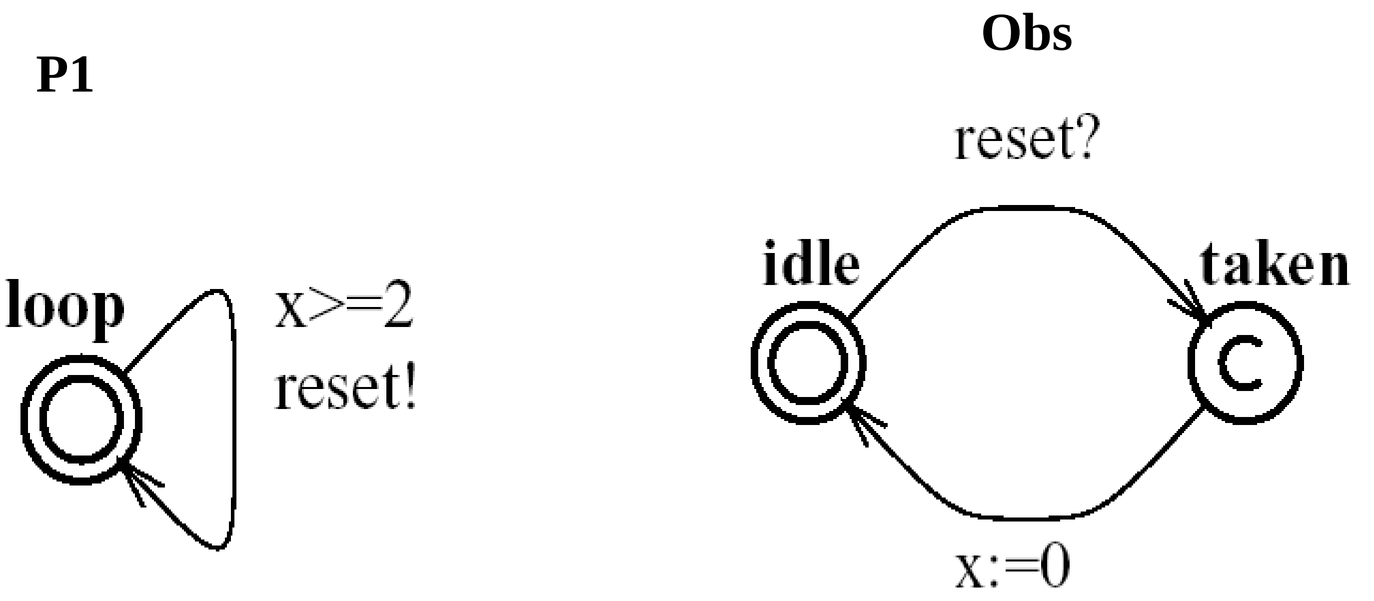
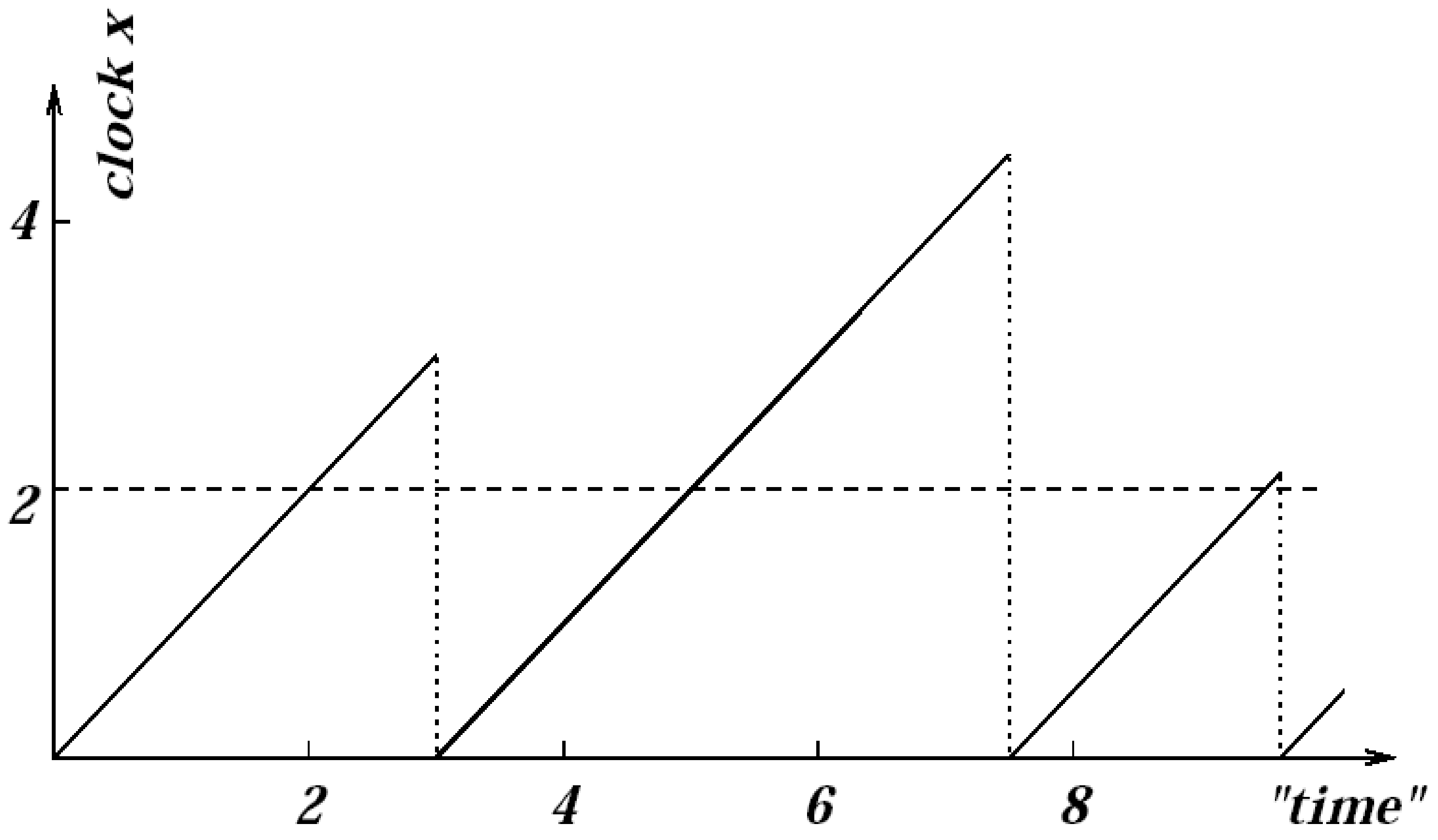


Figure 5: First example with the observer.

Example (cont.)



Example (cont.)

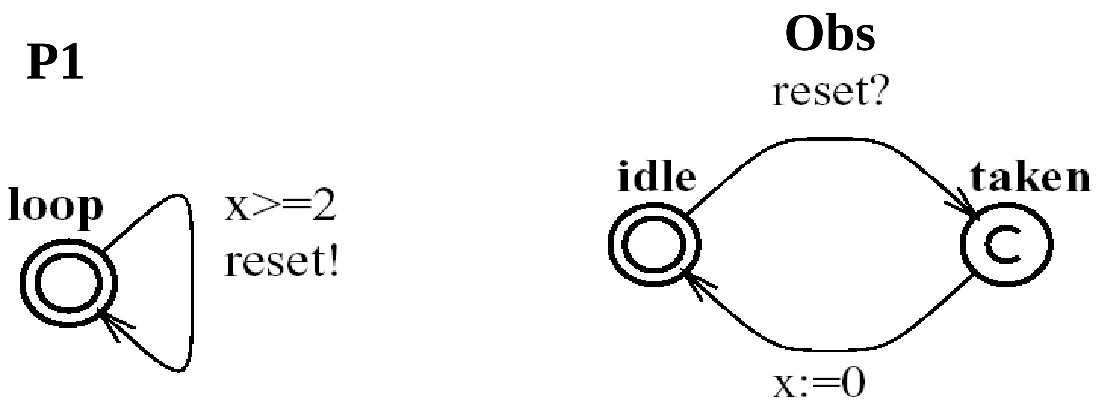


Figure 5: First example with the observer.

- Verification:
  - A[](Obs.taken imply x>=2)
  - E<>(Obs.idle and x>3) - for some path E, there is eventually <> a state in which Obs is in the idle state and x > 3.
  - E<>(Obs.idle and x>3000)

Example (cont.)

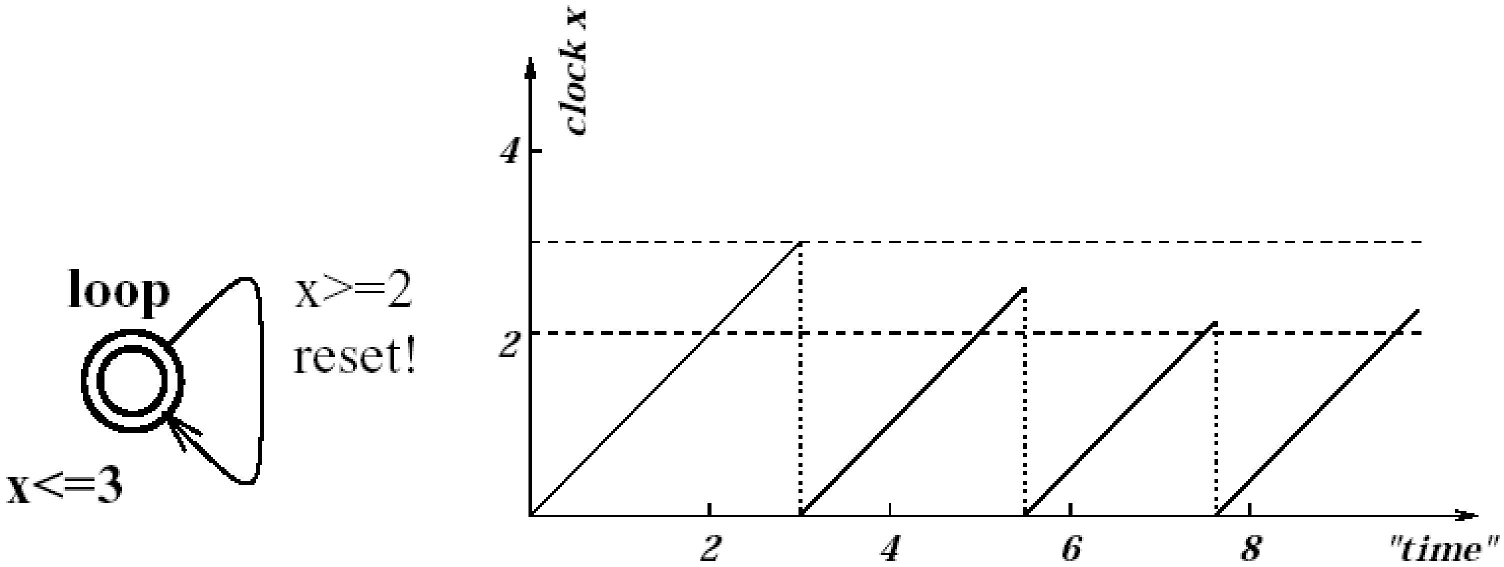


Figure 7: Adding an invariant: the new behaviour.

Example (cont.)

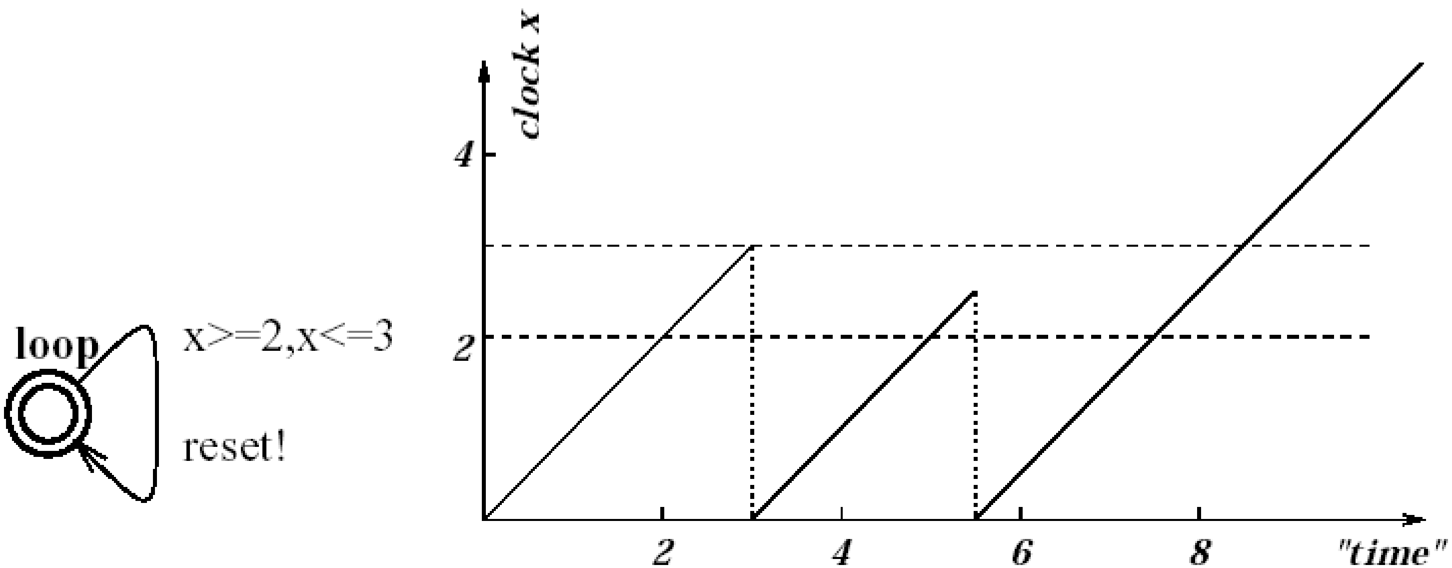


Figure 8: No invariant and a new guard: the new behaviour.

Computation Tree Logic, CTL
(Clarke and Emerson, 1980)

Syntax

$$\phi \quad ::= \quad p \;\mid\; \neg\phi \;\mid\; \phi \vee \phi \;\mid\; \mathsf{EX}\,\phi \;\mid\; \mathsf{E}\,[\phi\,\mathsf{U}\,\phi] \;\mid\; \mathsf{A}\,[\phi\,\mathsf{U}\,\phi].$$

- EX (pronounced "for some path next")
- E (pronounced "for some path")
- A (pronounced "for all paths") and
- U (pronounced "until").

UPPAAL Specification Language

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

A = on all paths, [ ] = always

E = on some path, <> = eventually

(AG p) – all paths, always

(EF p) – some path, eventually

process location   data guards   clock guards

p::= a.l | g_d | g_c | p and p |

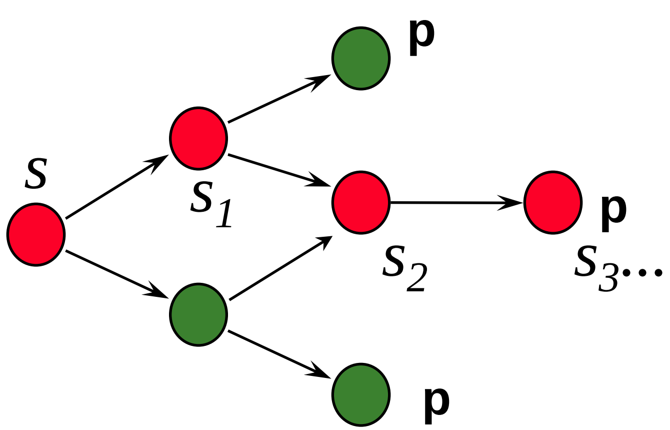
p or p | not p | p imply p |

(p)

Path

Definition 20. (Path)

A *path* is an infinite sequence of states $s_0\, s_1\, s_2 \ldots$ such that $(s_i, s_{i+1}) \in R$ for all $i \geqslant 0$.



The set of paths starting in s

$P_{\mathcal{M}}(s)$

Formal Semantics

satisfaction relation $\models$

$$
\begin{array}{ll}
s \models p & \text{iff } p \in Label(s) \\
s \models \neg\phi & \text{iff } \neg(s \models \phi) \\
s \models \phi \vee \psi & \text{iff } (s \models \phi) \vee (s \models \psi) \\
s \models \mathsf{EX}\,\phi & \text{iff } \exists \sigma \in P_{\mathcal{M}}(s).\, \sigma[1] \models \phi \\
s \models \mathsf{E}[\phi\,\mathsf{U}\,\psi] & \text{iff } \exists \sigma \in P_{\mathcal{M}}(s).(\exists j \geqslant 0.\, \sigma[j] \models \psi \wedge (\forall 0 \leqslant k < j.\, \sigma[k] \models \phi)) \\
s \models \mathsf{A}[\phi\,\mathsf{U}\,\psi] & \text{iff } \forall \sigma \in P_{\mathcal{M}}(s).(\exists j \geqslant 0.\, \sigma[j] \models \psi \wedge (\forall 0 \leqslant k < j.\, \sigma[k] \models \phi)).
\end{array}
$$

CTL, Derived Operators

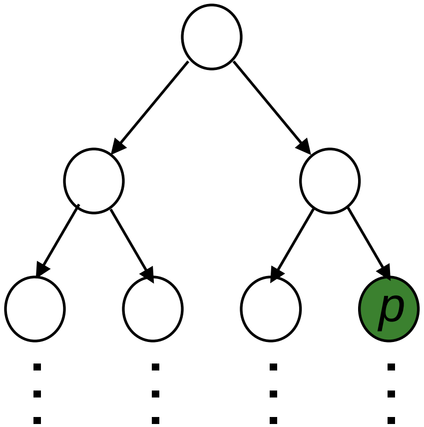
$$\mathsf{EF}\,\phi \;\equiv\; \mathsf{E}\left[\mathrm{true}\,\mathsf{U}\,\phi\right]$$

possible

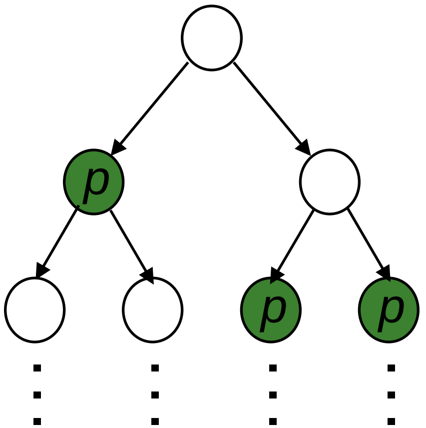
$$\mathsf{AF}\,\phi \;\equiv\; \mathsf{A}\left[\mathrm{true}\,\mathsf{U}\,\phi\right].$$

inevitable

EF p



AF p



CTL, Derived Operators

$$\mathsf{EG}\,\phi \equiv \neg\,\mathsf{AF}\,\neg\,\phi$$

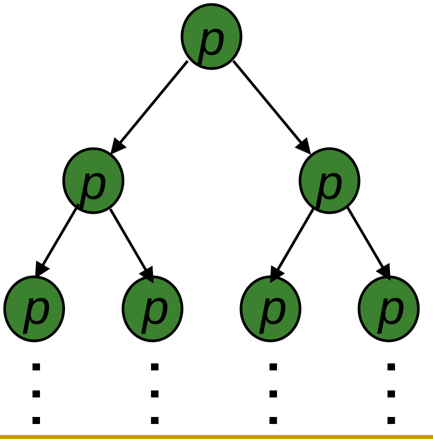
potentially always

$$\mathsf{AG}\,\phi \equiv \neg\,\mathsf{EF}\,\neg\,\phi$$

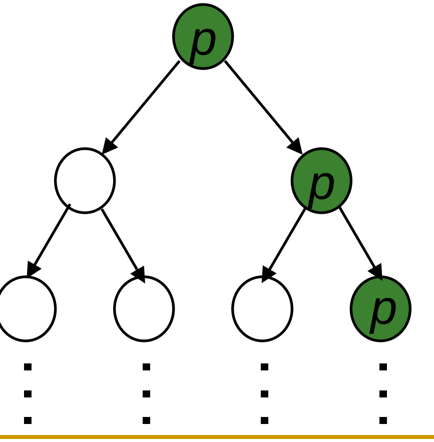
always

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

AG p



EG p



Theorem

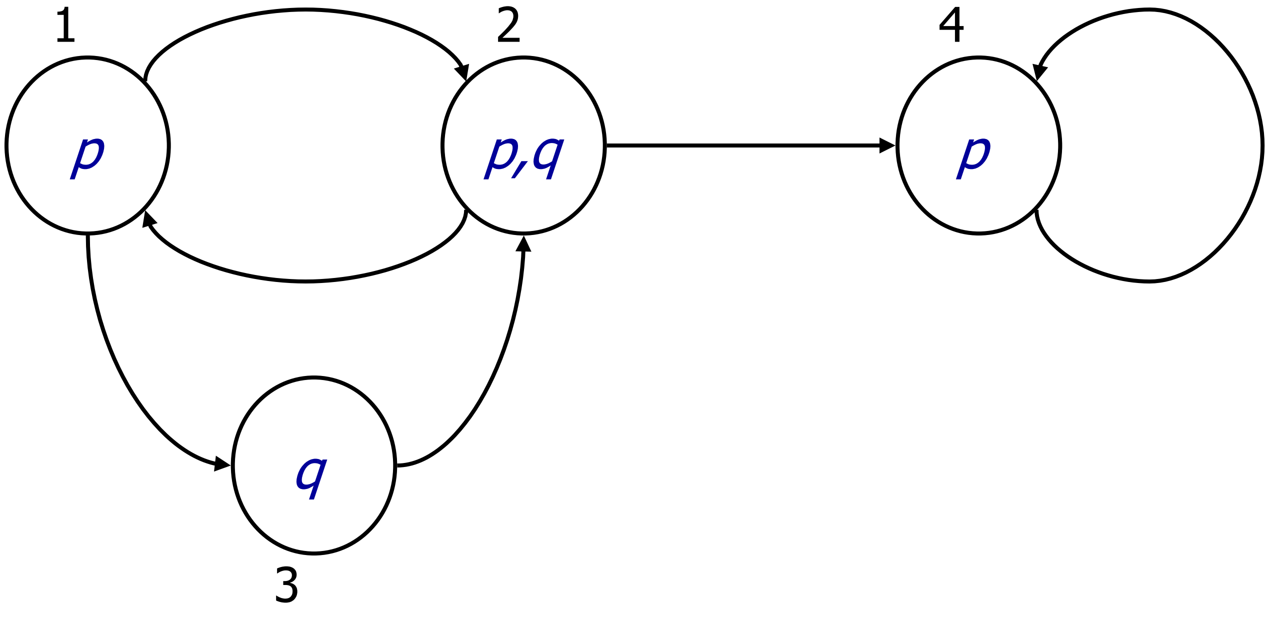
All operators are derivable from

- EX f
- EG f
- E[f U g]

and boolean connectives

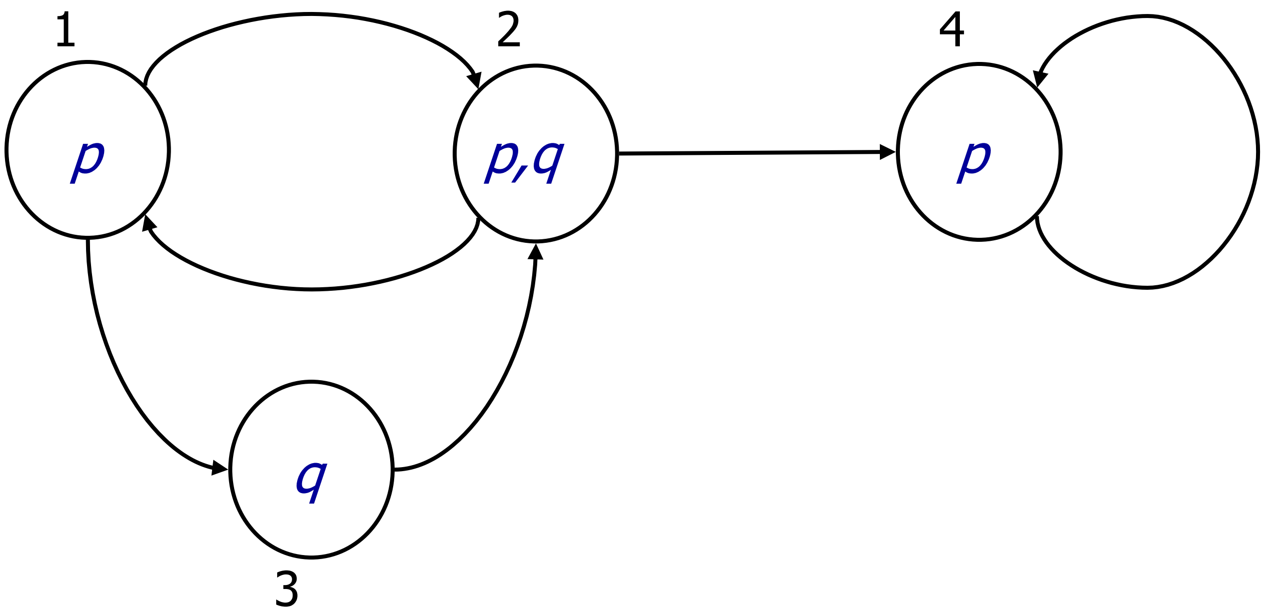
$$\mathrm{A}[f\ \mathrm{U}\ g] \equiv \neg \mathrm{E}[\neg g \mathrm{U}(\neg f \wedge \neg g)] \wedge \neg \mathrm{EG} \neg g$$

Example



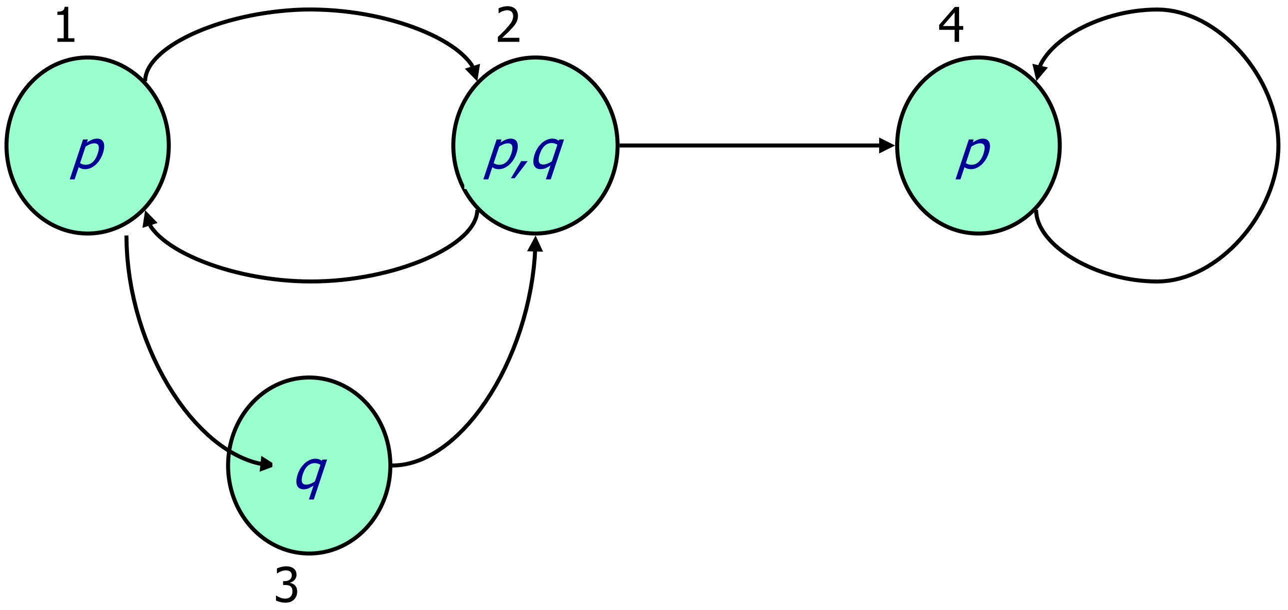
Example

EX *p*



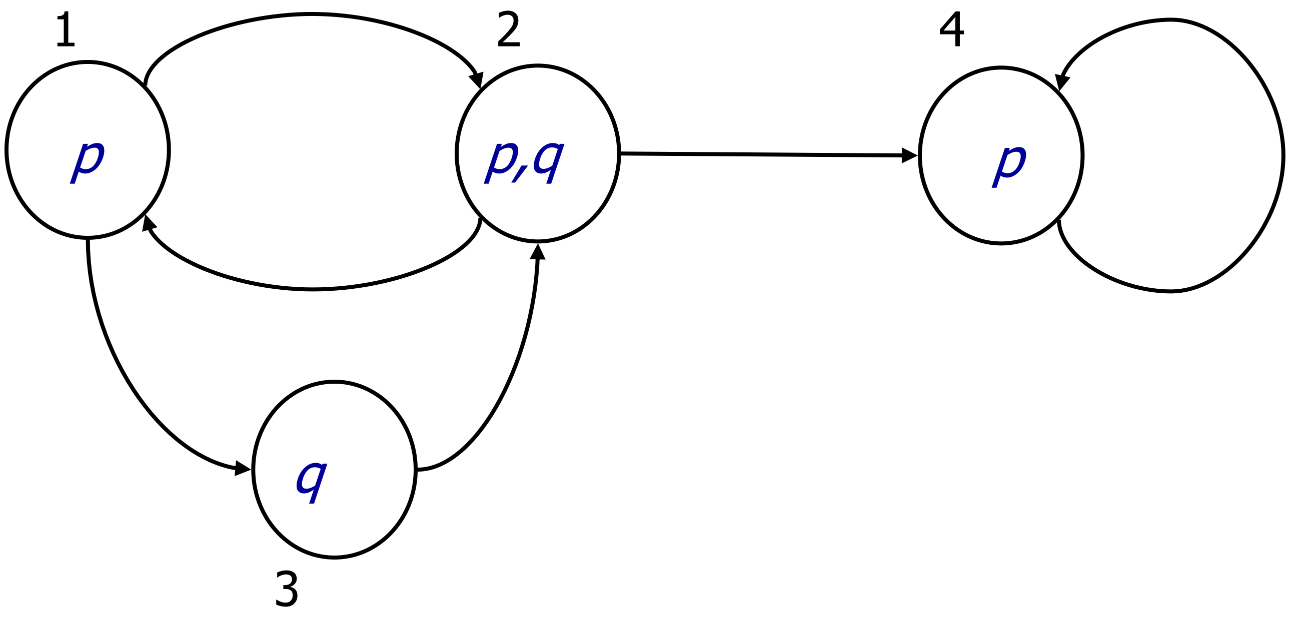
Example

EX *p*



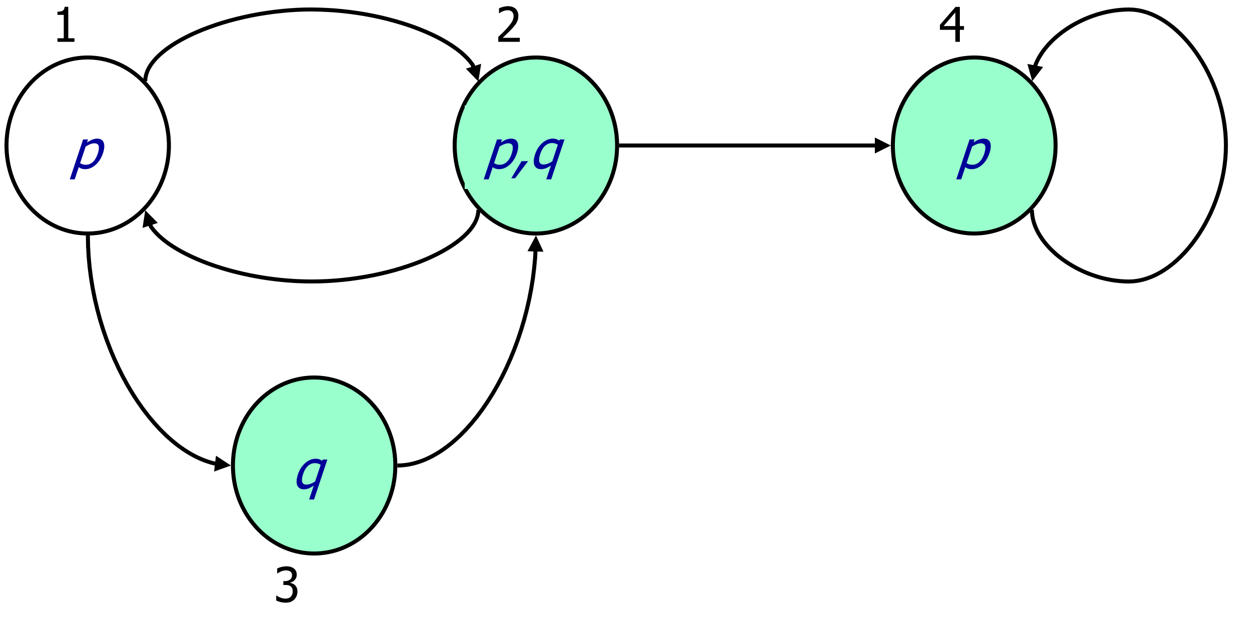
Example

AX *p*



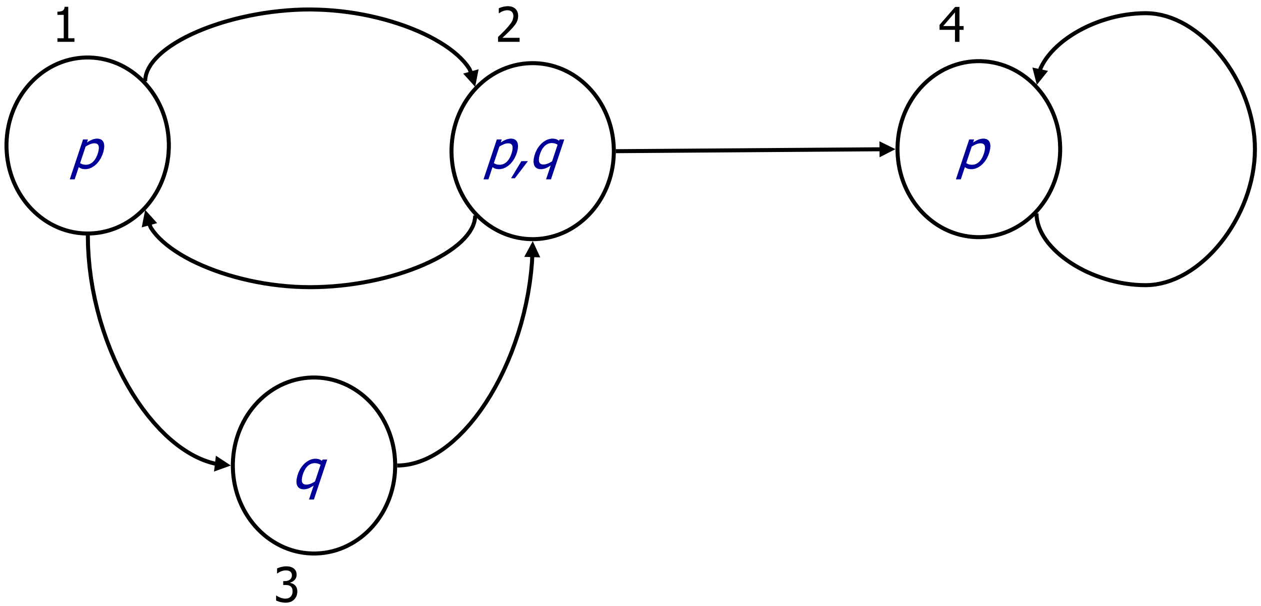
Example

AX *p*



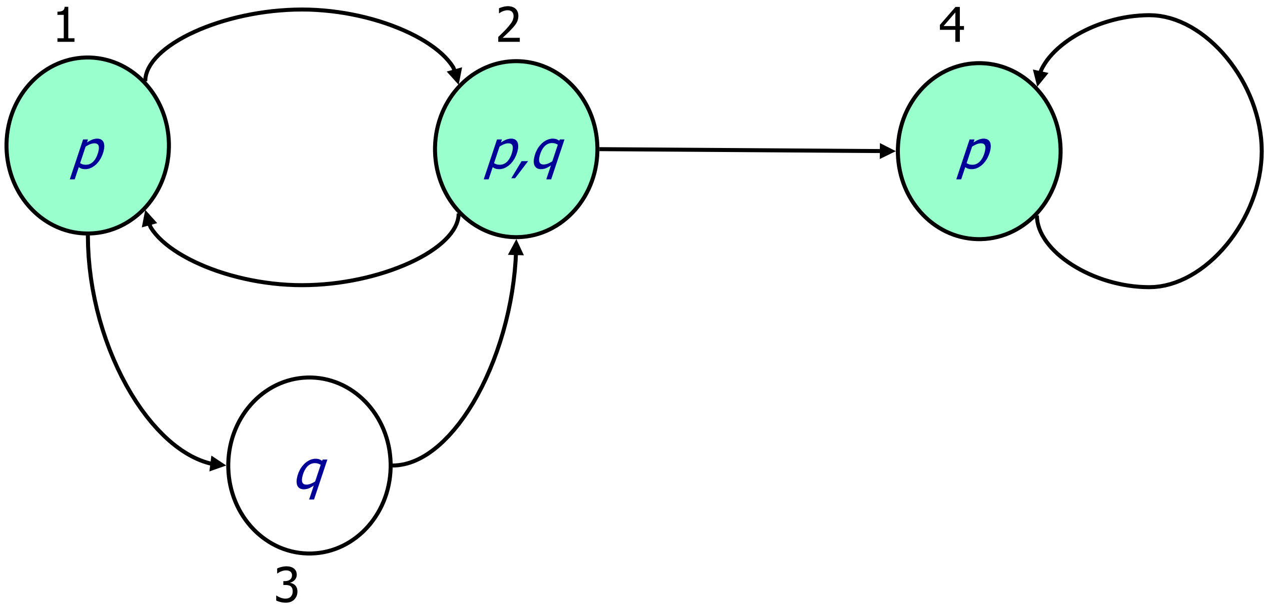
Example

EG *p*



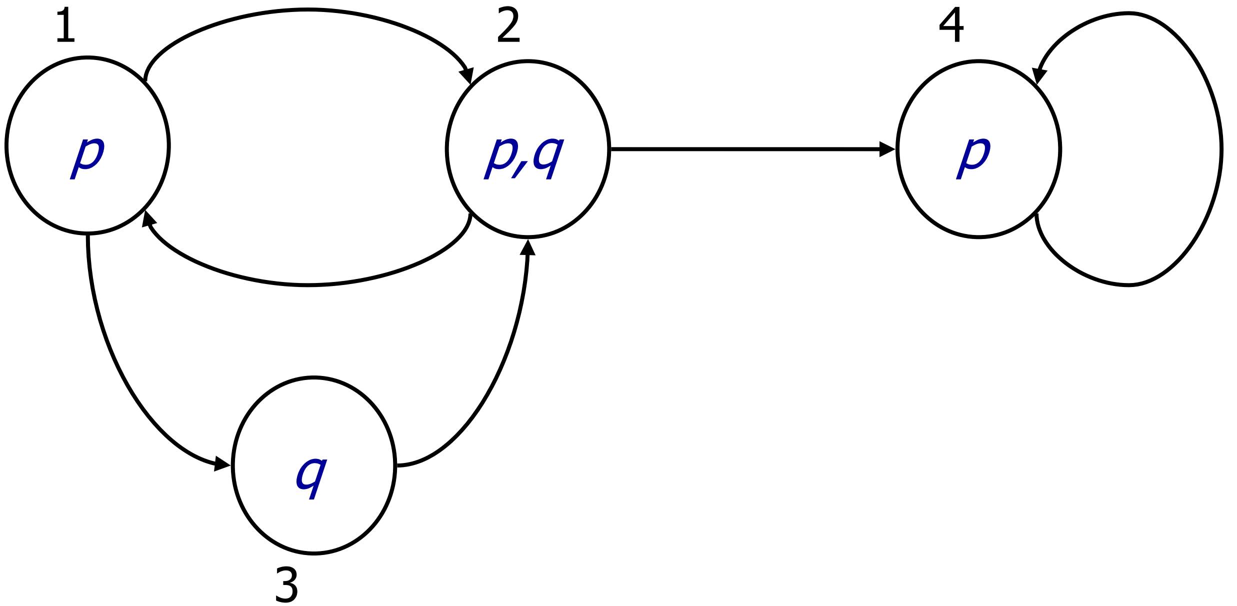
Example

EG *p*



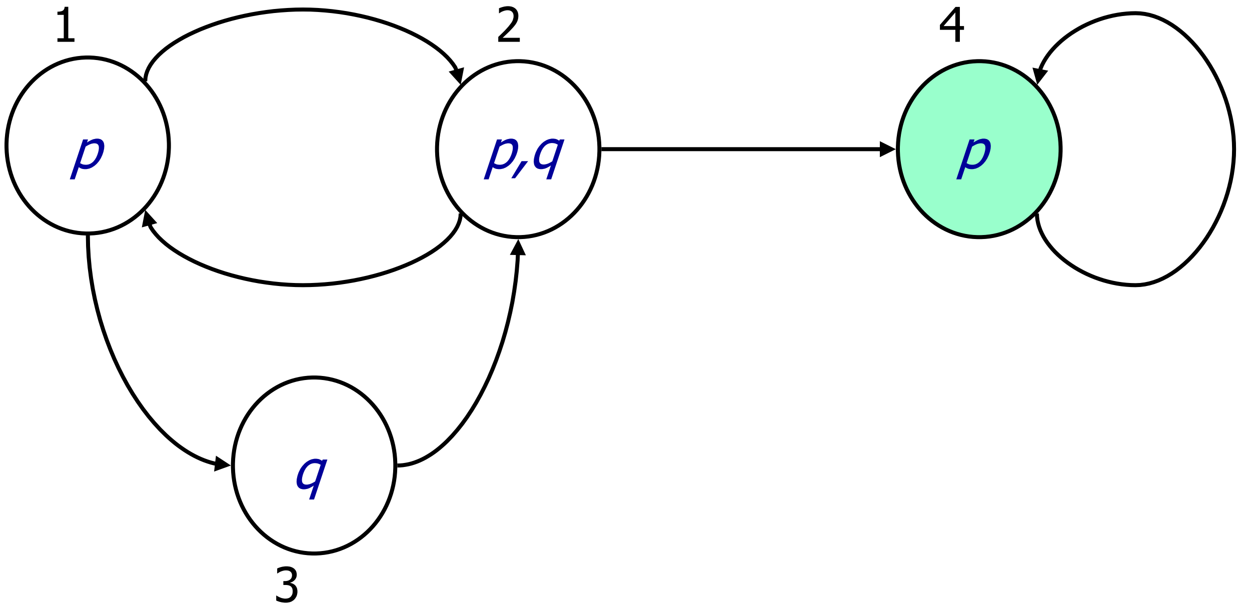
Example

AG p



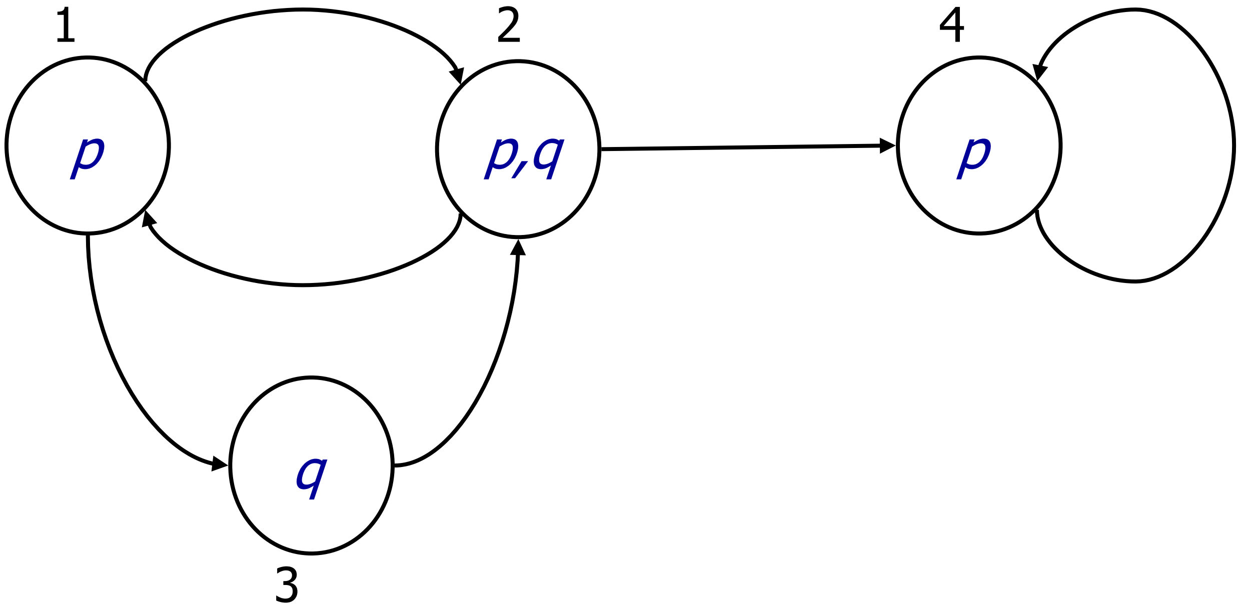
Example

AG *p*



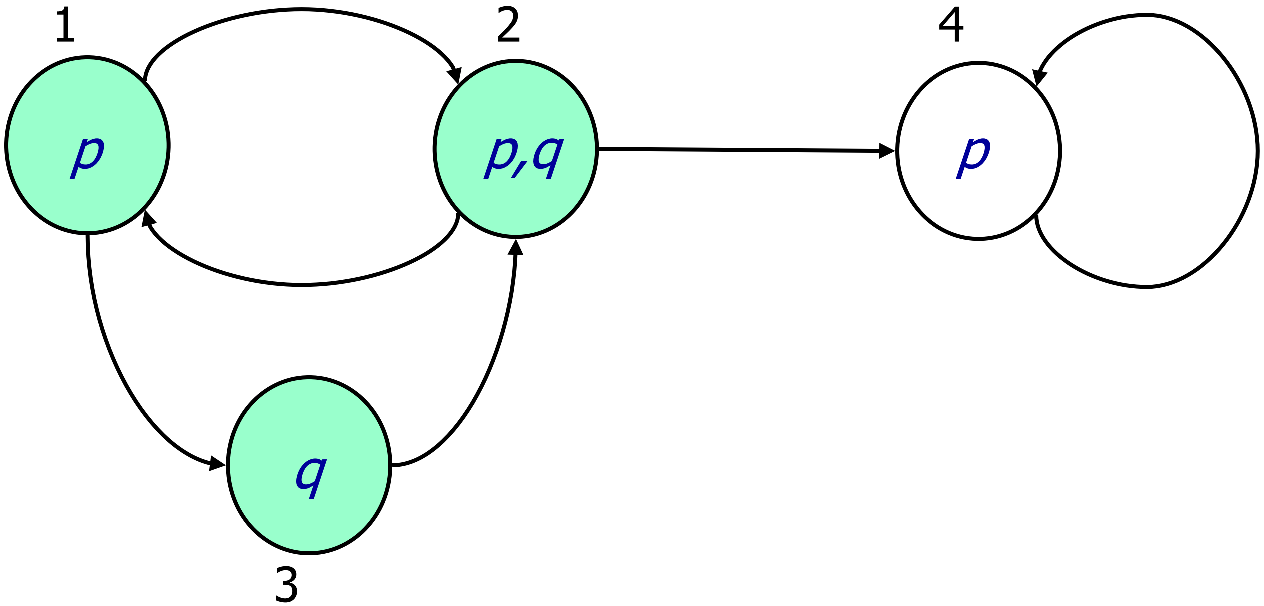
Example

A[p **U** q **]**



Example

A[p **U** q **]**



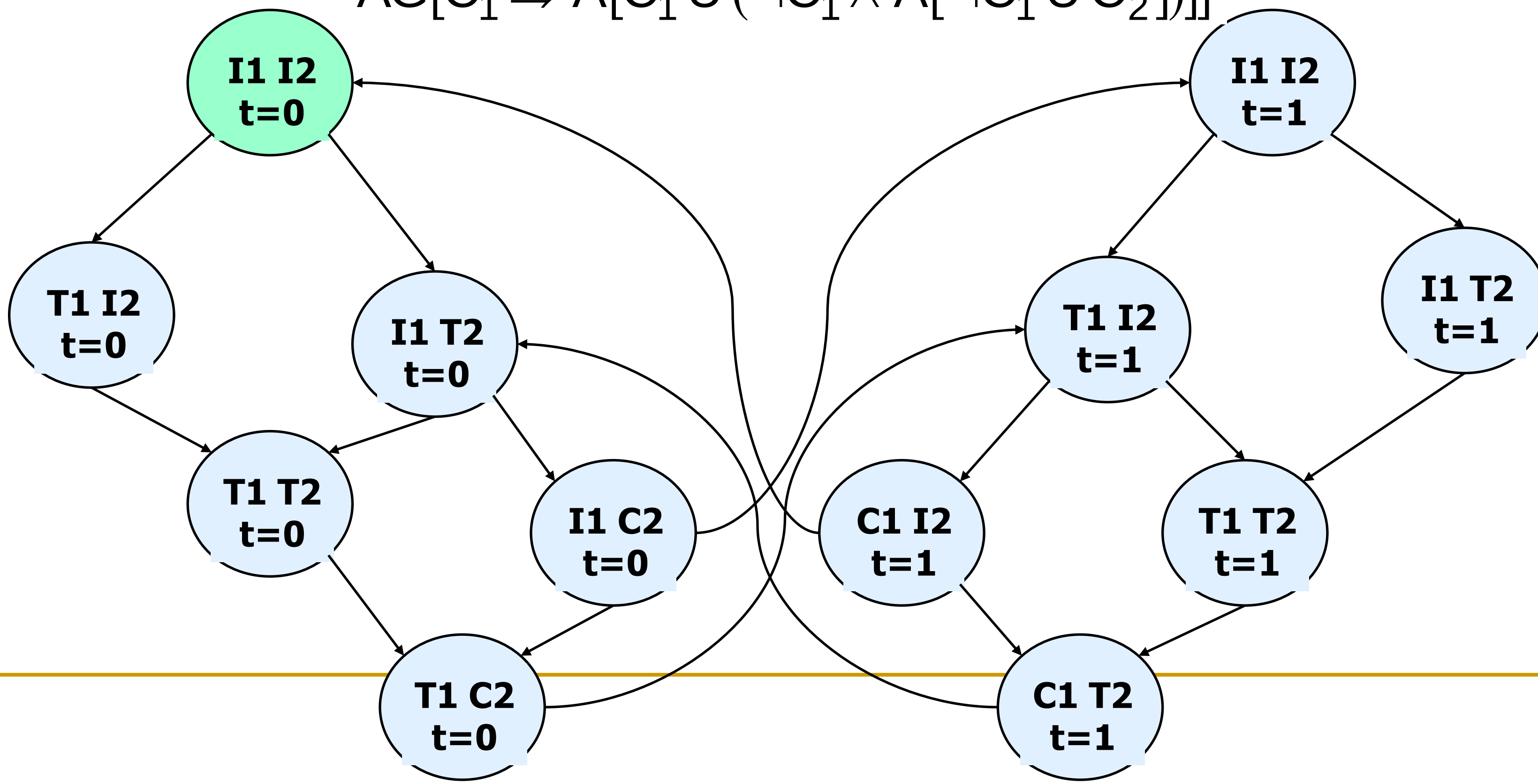
Properties of MUTEX example ?

$$AG\ \neg(C_1 \wedge C_2)$$

$$AG[\ T_1 \Rightarrow AF(C_1)]$$

$$EG[\neg C_1]$$

$$AG[C_1 \Rightarrow A[C_1\ U\ (\neg C_1 \wedge A[\neg C_1\ U\ C_2\ ])]]$$



Summary

- **Next Time:** UPPAAL Logic