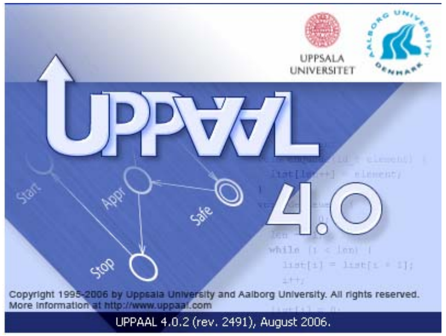
Test and Verification

Of Real-Time Systems using UPPAAL

Brian Nielsen

bnielsen@cs.aau.dk



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.

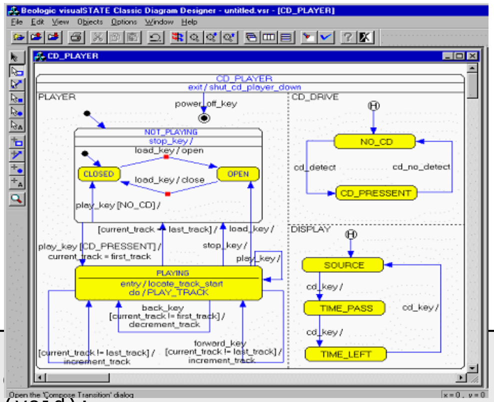
AALBORG UNIVERSITY DENMARK

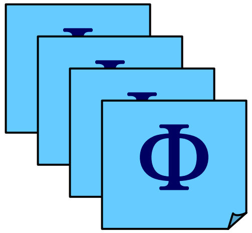
BRICS
Basic Research
in Computer Science

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Verifikation og Test

Model





Spec

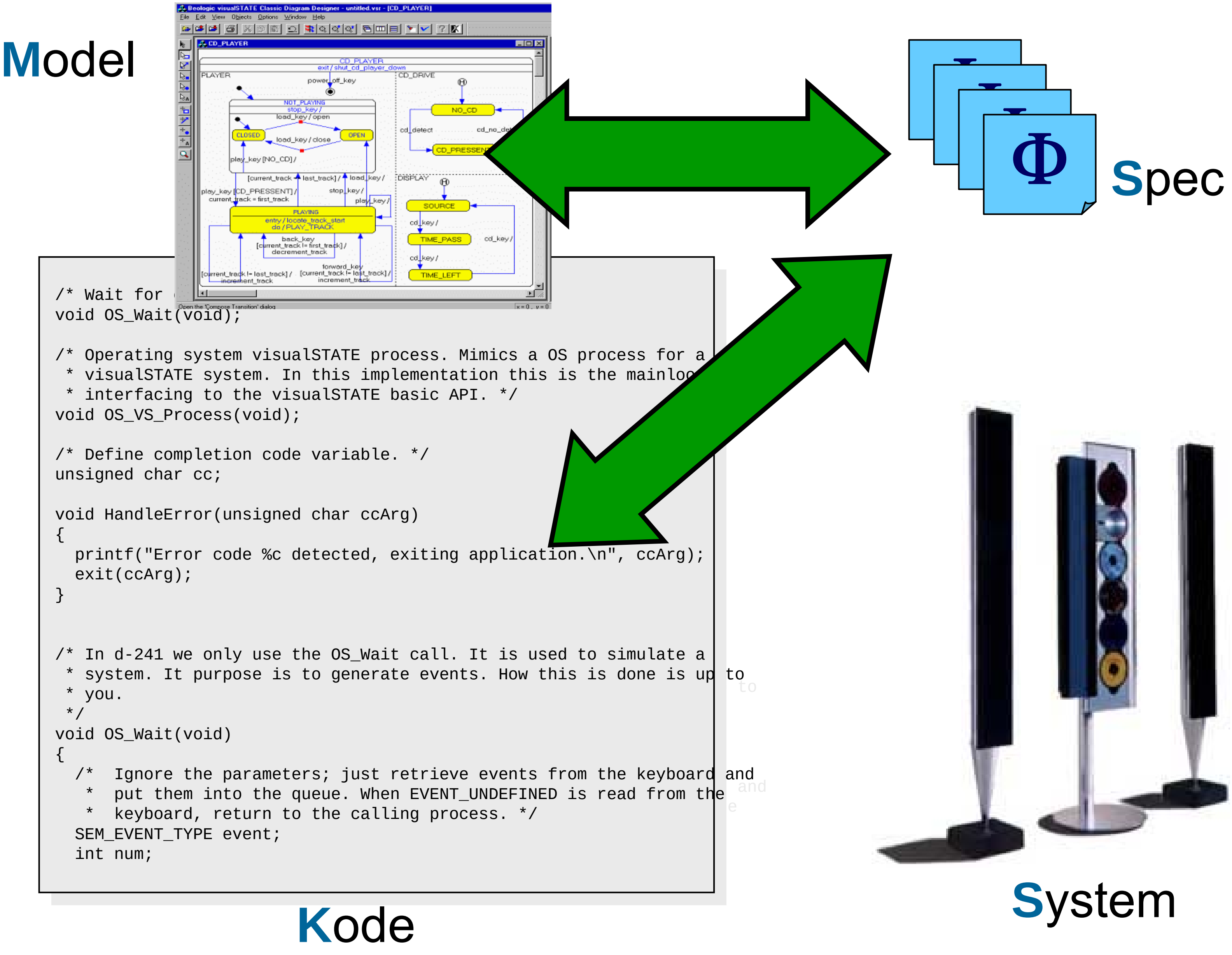
```
/* Wait for
void OS_Wait(void);

/* Operating system visualSTATE process. Mimics a OS process for a
 * visualSTATE system. In this implementation this is the mainloop
 * interfacing to the visualSTATE basic API. */
void OS_VS_Process(void);

/* Define completion code variable. */
unsigned char cc;

void HandleError(unsigned char ccArg)
{
  printf("Error code %c detected, exiting application.\n", ccArg);
  exit(ccArg);
}


/* In d-241 we only use the OS_Wait call. It is used to simulate a
 * system. It purpose is to generate events. How this is done is up to
 * you.
 */
void OS_Wait(void)
{
  /*  Ignore the parameters; just retrieve events from the keyboard and
   *  put them into the queue. When EVENT_UNDEFINED is read from the
   *  keyboard, return to the calling process. */
  SEM_EVENT_TYPE event;
  int num;
```

Kode

System

CISS

Verifikation og Test

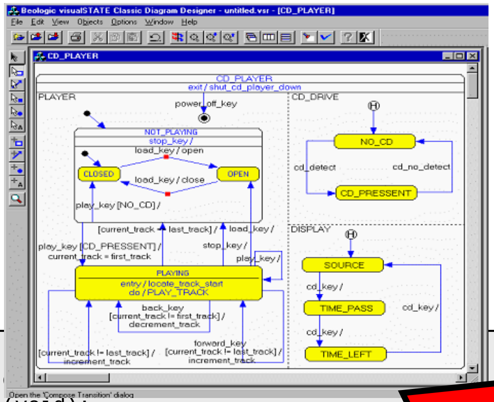


```
/* Wait for
void OS_Wait(void);

/* Operating system visualSTATE process. Mimics a OS process for a
 * visualSTATE system. In this implementation this is the mainloo
 * interfacing to the visualSTATE basic API. */
void OS_VS_Process(void);

/* Define completion code variable. */
unsigned char cc;

void HandleError(unsigned char ccArg)
{
  printf("Error code %c detected, exiting application.\n", ccArg);
  exit(ccArg);
}


/* In d-241 we only use the OS_Wait call. It is used to simulate a
 * system. It purpose is to generate events. How this is done is up to
 * you.
 */
void OS_Wait(void)
{
  /*  Ignore the parameters; just retrieve events from the keyboard and
   *  put them into the queue. When EVENT_UNDEFINED is read from the
   *  keyboard, return to the calling process. */
  SEM_EVENT_TYPE event;
  int num;
```

CISS

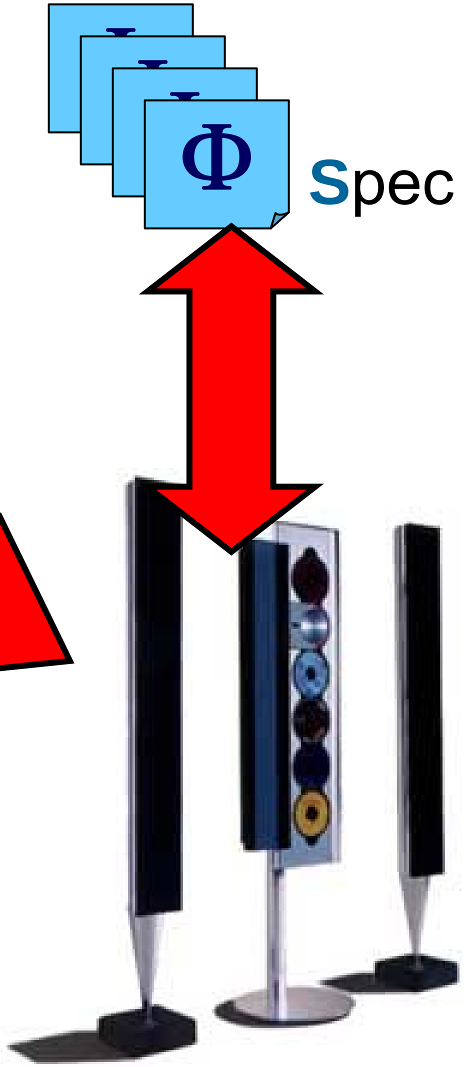
Verifikation og Test

Model



```
/* Wait for
void OS_Wait(void);

/* Operating system visualSTATE process. Mimic                 for a
 * visualSTATE system. In this implementation t                 op
 * interfacing to the visualSTATE basic API. */
void OS_VS_Process(void);

/* Define completion code variable. */
unsigned char cc;

void HandleError(unsigned char ccArg)
{
  printf("Error code %c detected, exiting application.\n", ccArg);
  exit(ccArg);
}


/* In d-241 we only use the OS_Wait call. It is used to simulate a
 * system. It purpose is to generate events. How this is done is up to
 * you.
 */
void OS_Wait(void)
{
  /*  Ignore the parameters; just retrieve events from the keyboard and
   *  put them into the queue. When EVENT_UNDEFINED is read from the
   *  keyboard, return to the calling process. */
  SEM_EVENT_TYPE event;
  int num;
```

Kode



Spec

System

CISS

Modelling Behaviour using State Machines

AALBORG UNIVERSITY DENMARK

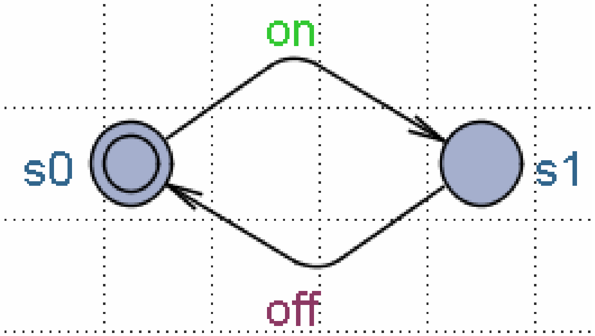
BRICS
Basic Research
in Computer Science

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Modelling processes

- A process is the execution of a sequential program.
- modeled as a finite state machine (LTS)
  - transits from state to state
  - by executing a sequence of *atomic* actions.

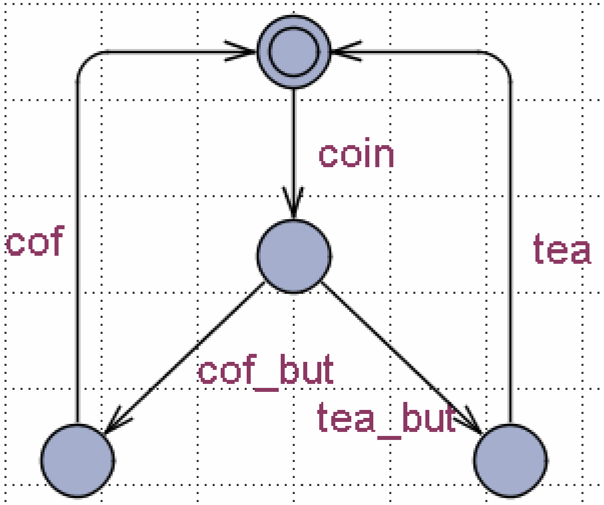
a light switch
LTS



on→off→on→off→on→off→ ……….

a sequence of actions or *trace*

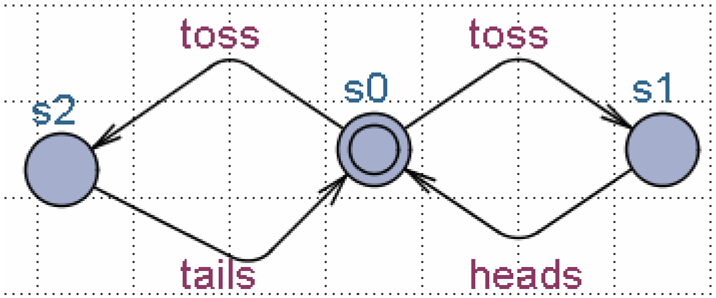
Modelling Choices



- Who or what makes the choice?
- Is there a difference between input and output actions?

Non-deterministic Choice

- Tossing a coin

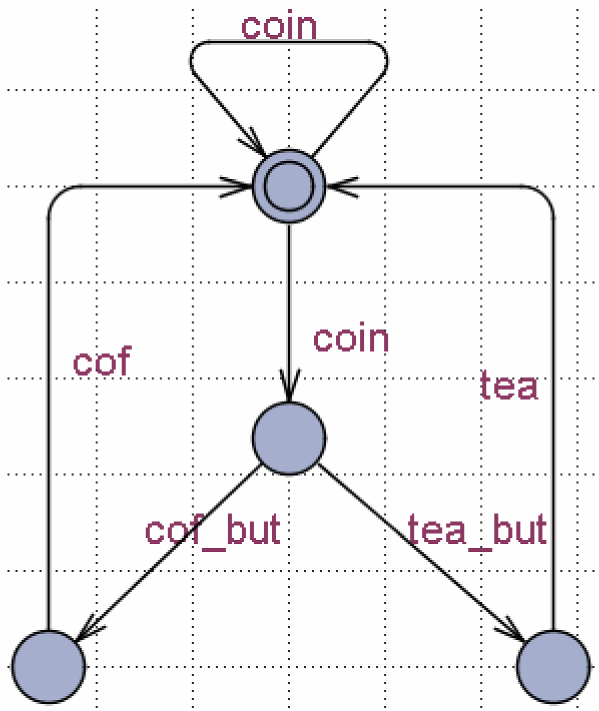


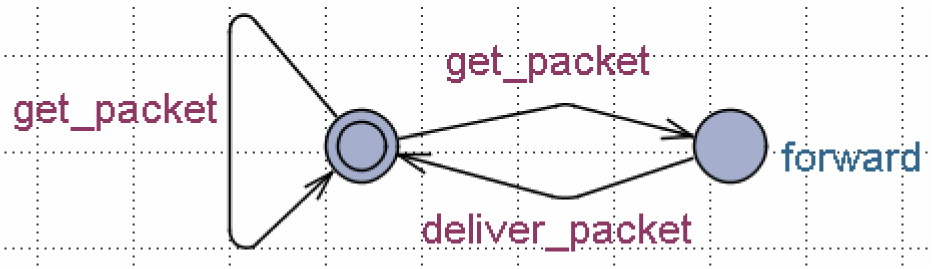
- Possible traces?
  - Both outcomes possible
  - Nothing said about relative frequency
  - If coin is fair, the outcome is 50/50

Non-Deterministic Choice -modelling failure

How do we model an unreliable communication channel which accepts **packets,** and if a failure occurs produces no output, otherwise **delivers** the packet to the receiver?

Use non-determinism...

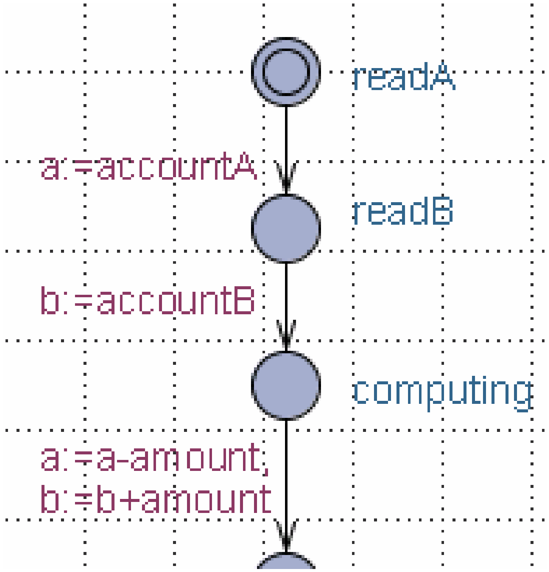


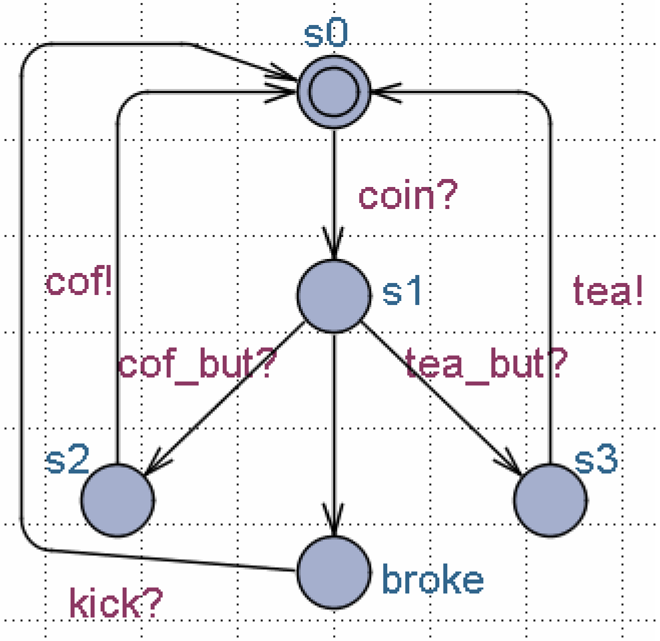


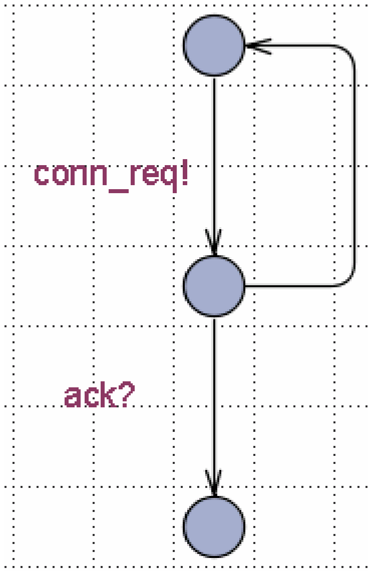
CISS

Internal-Actions

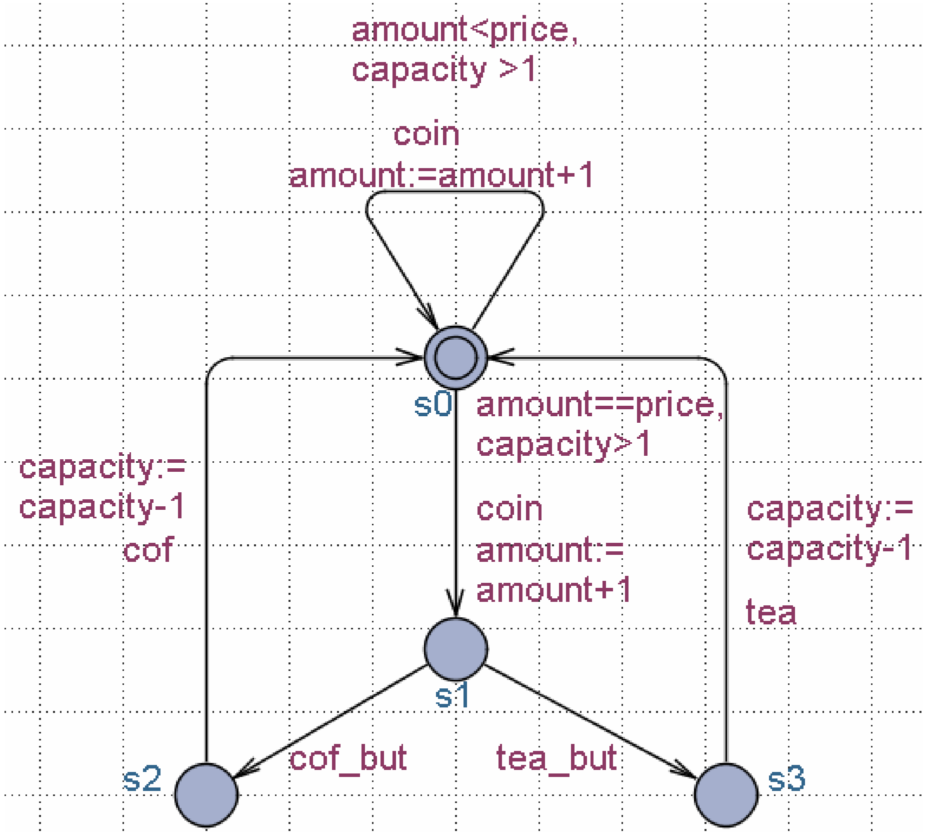
- Spontaneous actions
- Internal actions
- Tau-actions
- Internal transitions can be taken on the initiative of a single machine without communication with others







Extended FSM

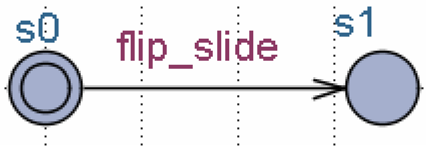


- EFSM = FSM + variables + enabling conditions + assignments
- Transition still atomic
- Can be translated into FSM if variables have bounded domain
- State: control location+variable states: (state,total,capacity)
- (s0,5,10)

SS

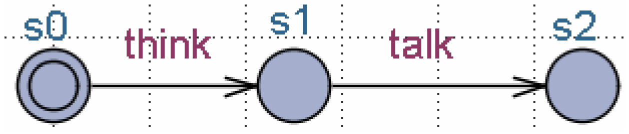
Parallel Composition: interleaving

Flipper



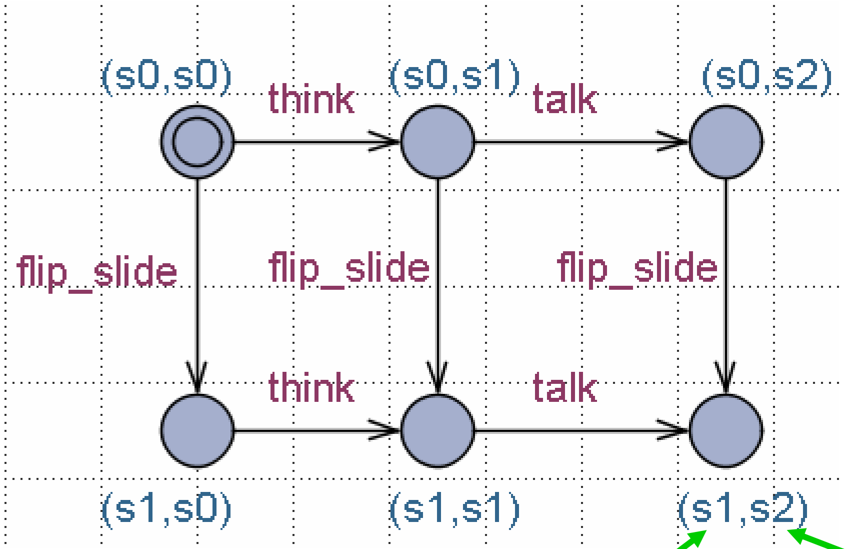
2 states

Speaker



3 states

Lecturer =
Speaker || Flipper



2*3 states

from Flipper

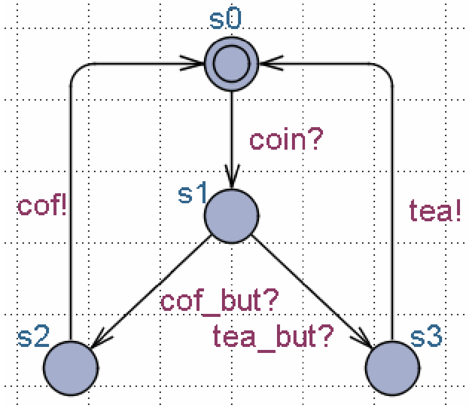
from Speaker

Process Interaction

- ! = Output, ? = Input
- Handshake communication
- Two-way

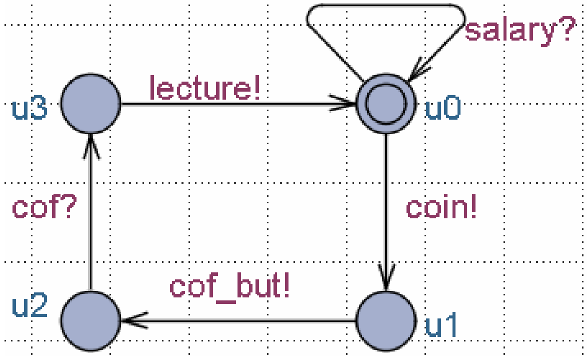
University=
Coffee Machine || Lecturer
•LTS?
•How many states?
•Traces ?

Coffee Machine

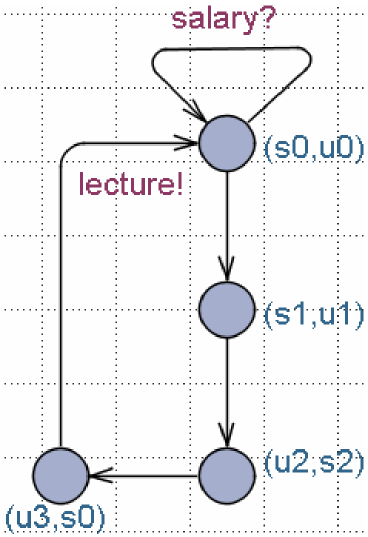


4 states

Lecturer



4 states

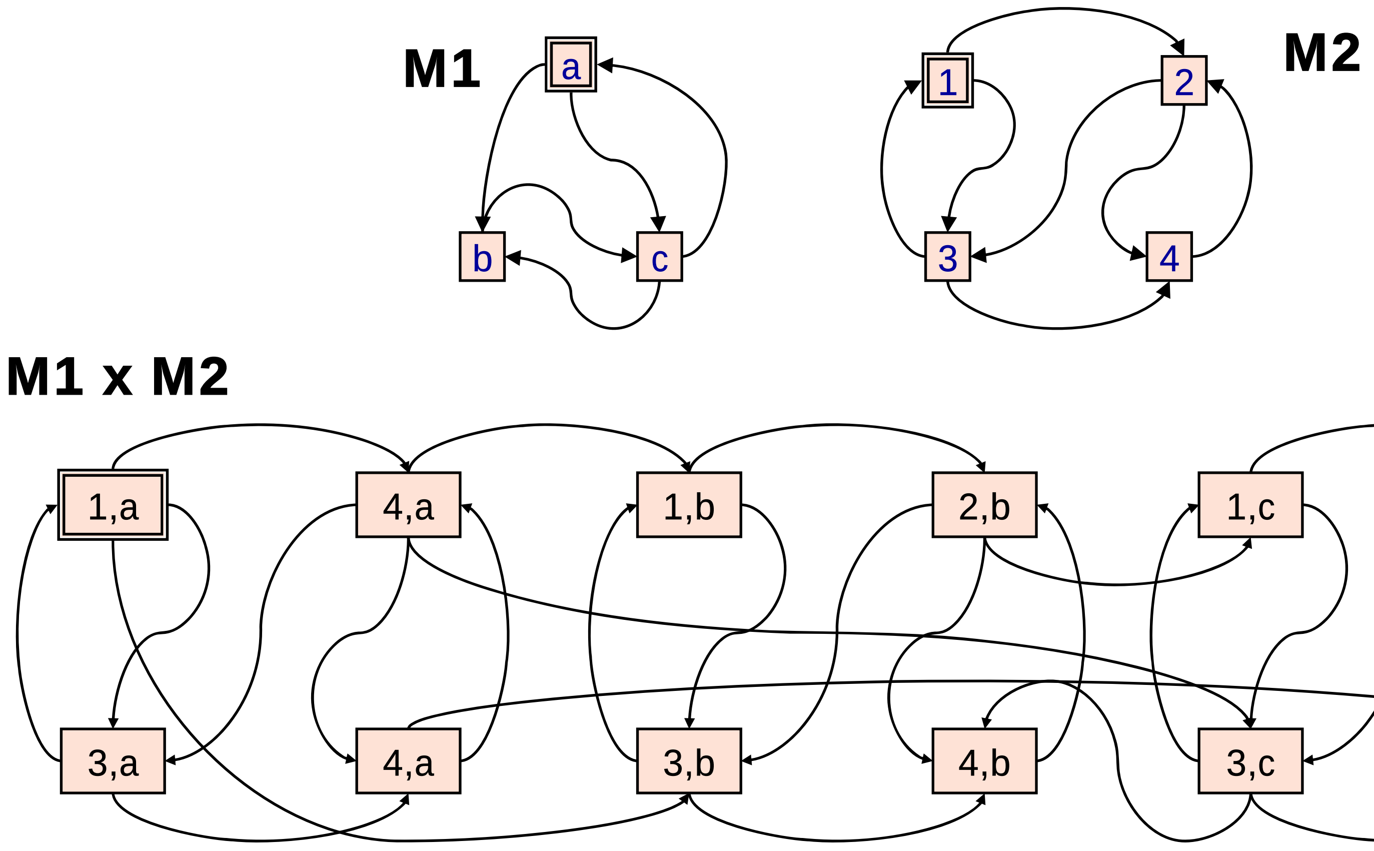


synchronization results in internal actions

4 states:Interaction constrain overall behavior

CISS

Composition



All combinations=
exponential in no of machines

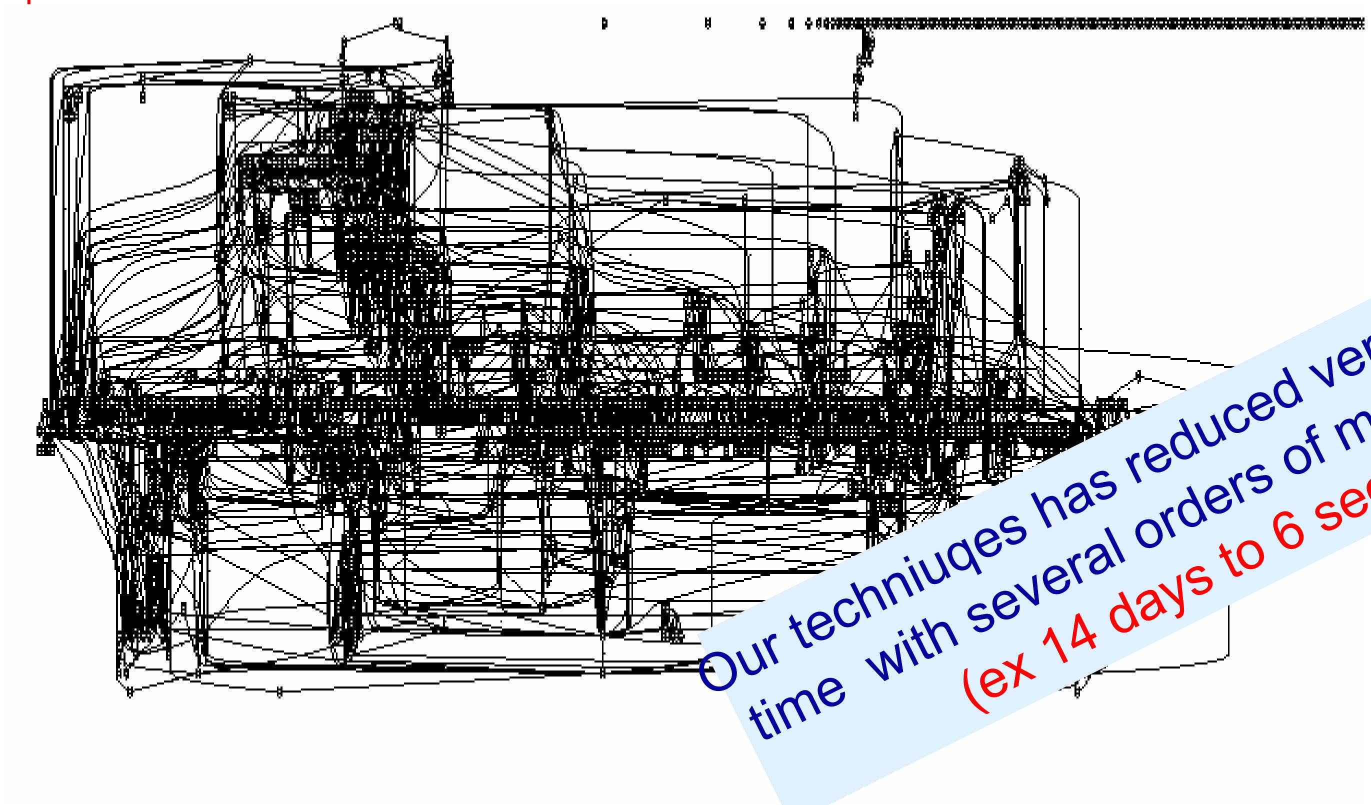
CISS

Train Simulator

VVS
visualSTATE

1421 machines
11102 transitions
2981 inputs
2667 outputs
3204 local states
Declare state sp.: 10^476

BUGS ?

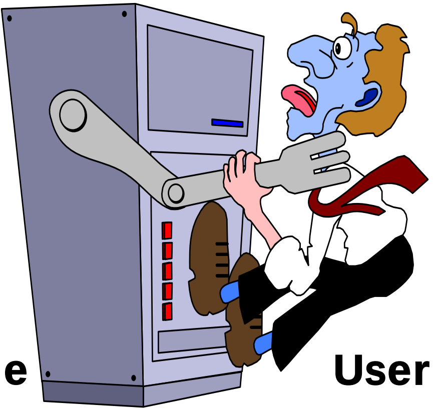


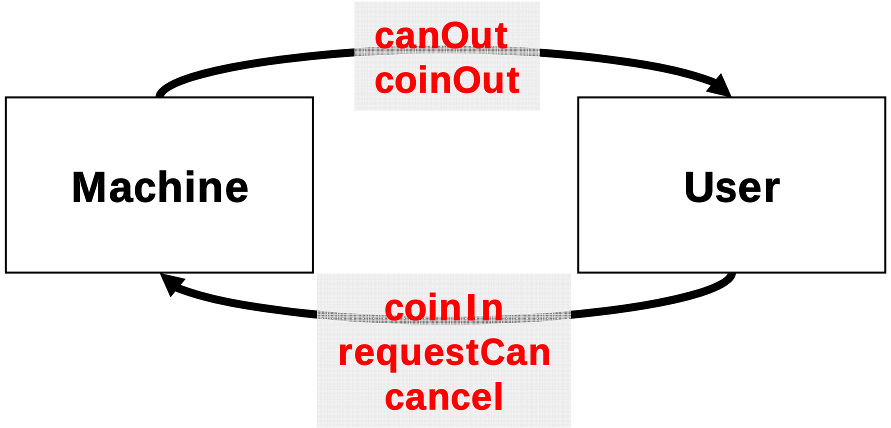
Our techniuqes has reduced verification time with several orders of magnitude (ex 14 days to 6 sec)

CISS

Modelling Exercise

The Vending Machine





Machine

User

canOut
coinOut

coinIn
requestCan
cancel

- Simulate model w
  Random User
- Model Fair User
- Model Non-Thirsty User
- Deadlocks ?

- Cans requested will be delivered ?
- Cancellations are obeyed ?

- What happens if multiple users?

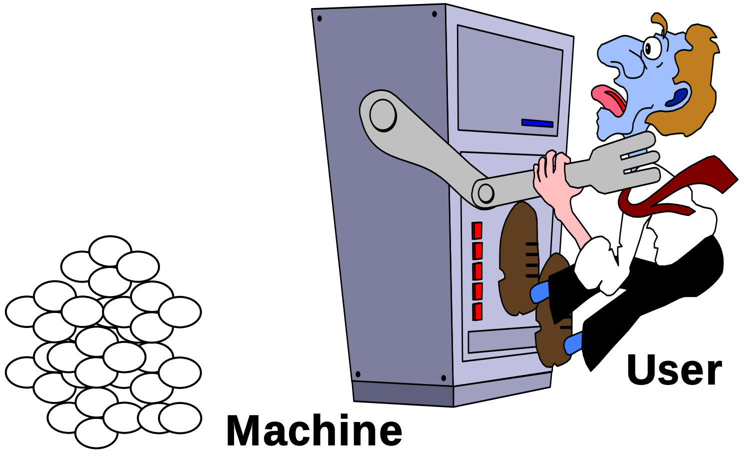
Assumption: 1 can = 1 coin!

Exercise

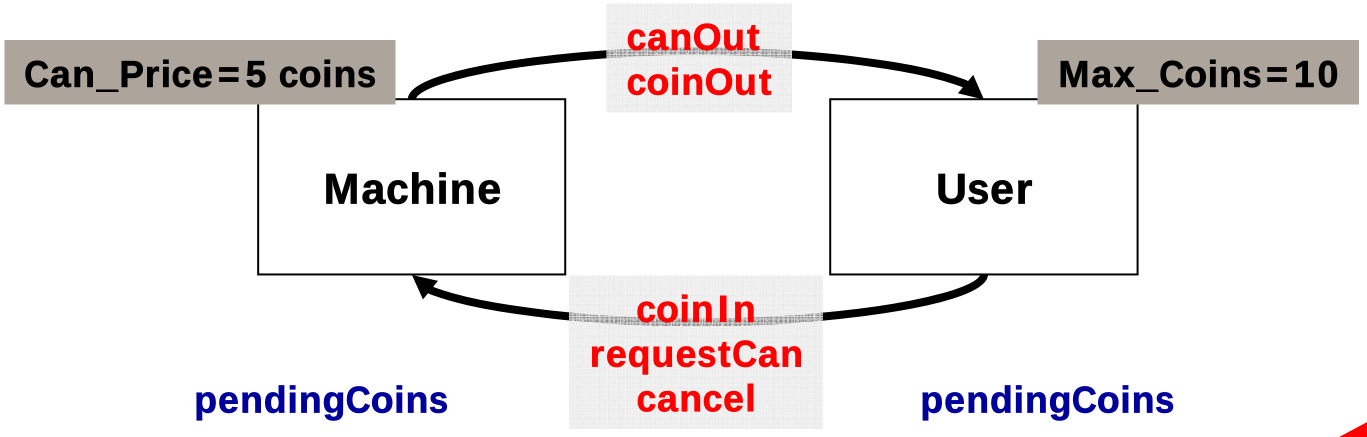
CISS

Modelling Exercise

The Vending Machine



- Extend model of Machine and FairUser
- Do extensive simulation



Can_Price=5 coins

canOut
coinOut

Max_Coins=10

Machine

User

coinIn
requestCan
cancel

pendingCoins

pendingCoins

Exercise

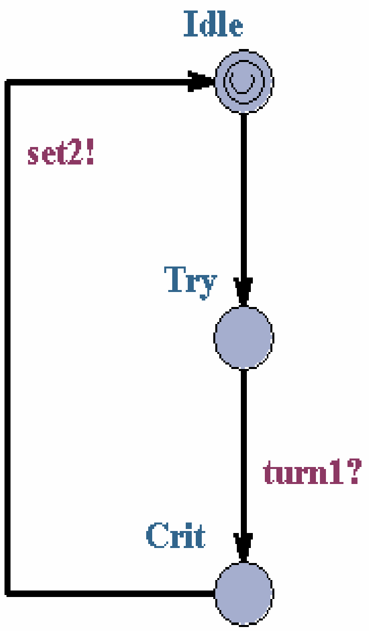
CISS

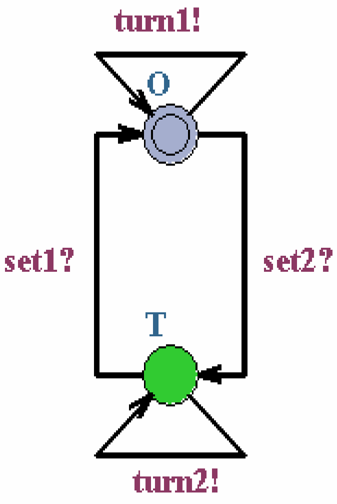
Verification = Model Checking

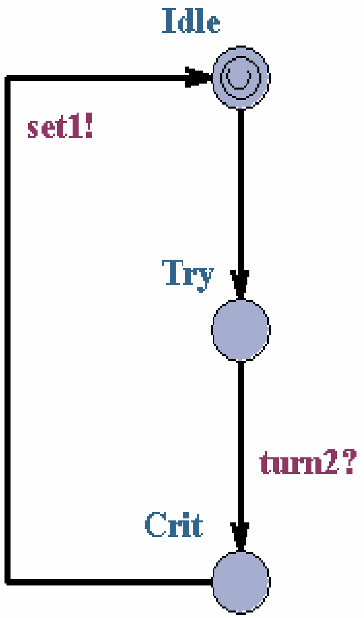
- Reachability
- Generic properties

Mutual Exclusion

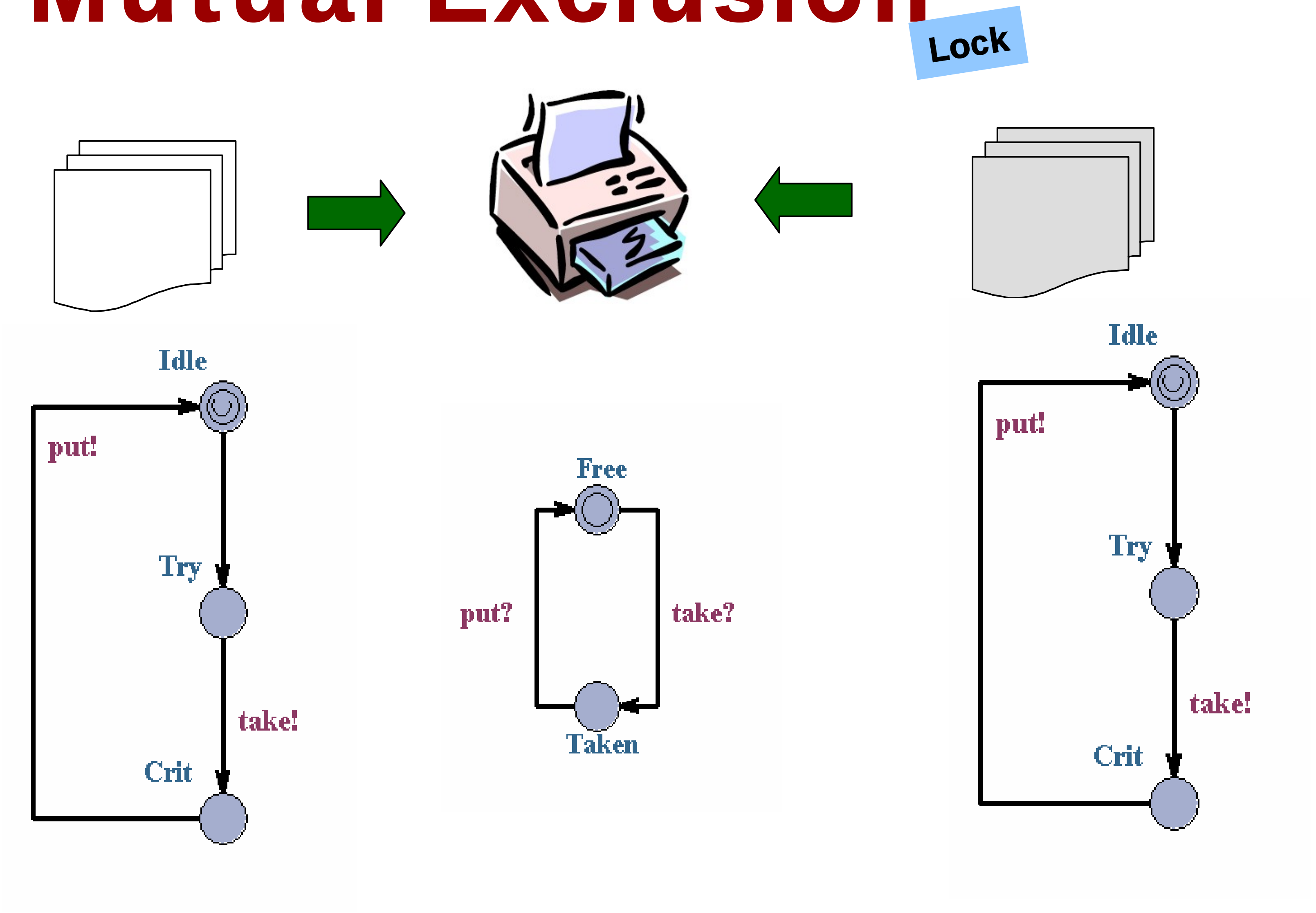
Taking turns



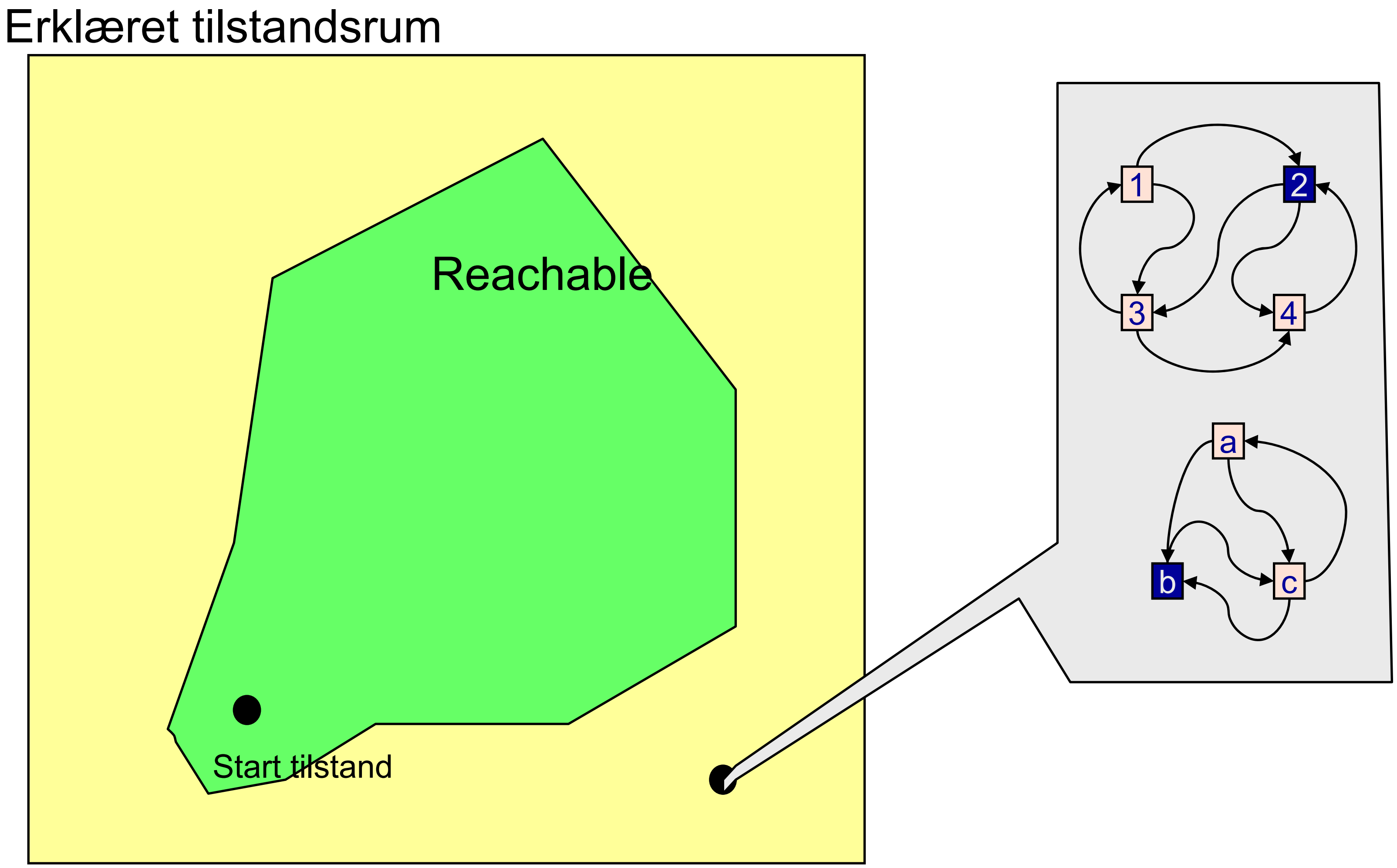




Mutual Exclusion



Udforskning af Tilstandsrum

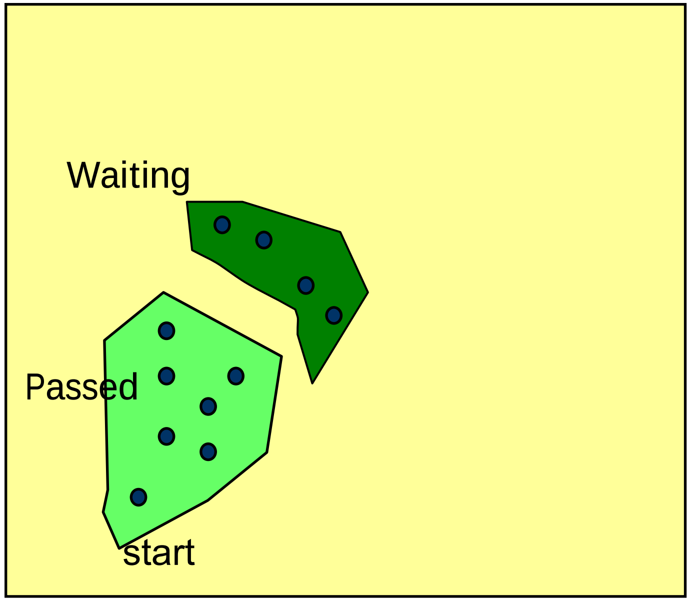


CISS

Udforskning af tilstandrum

Forward Reachability Analysis

Erklæret tilstandsrum



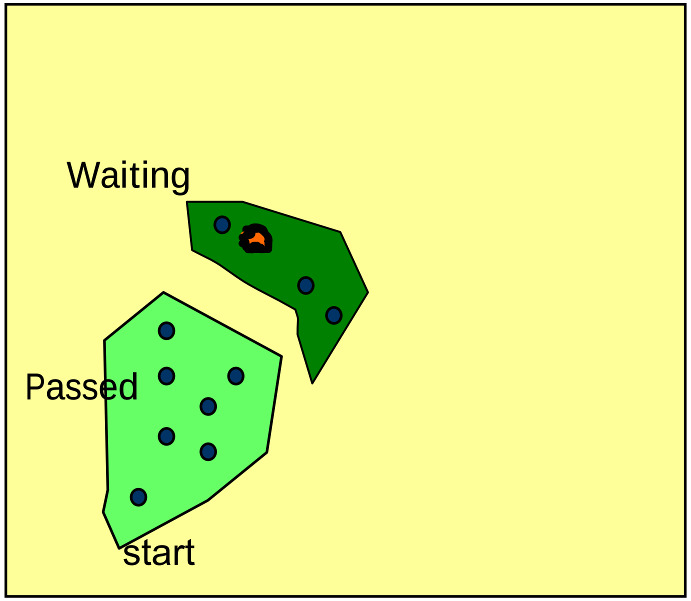
```
Passed:=Ø
Waiting:={$s_0$}
While(Waiting!=Ø)
   {
   select s∈ Waiting
   Waiting:=Waiting\{s}
   if s∉Passed
      whenever (s → t) then
         Waiting:=Waiting ∪ {t}
      Passed:=Passed ∪ {s}
   }
```

CISS

Udforskning af tilstandrum

Forward Reachability Analysis

Erklæret tilstandsrum

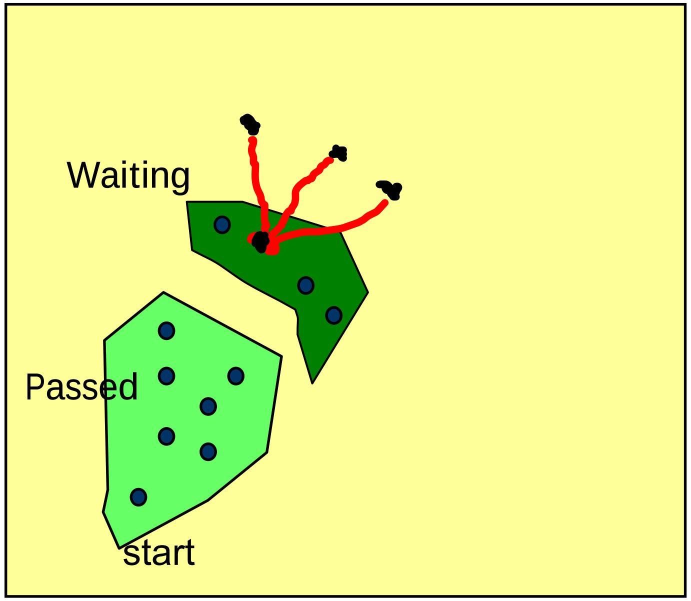


```
Passed:=Ø
Waiting:={$s_0$}
While(Waiting!=Ø)
   {
   select s ∈ Waiting
   Waiting:=Waiting\{s}
   if s∉Passed
      whenever (s → t) then
         Waiting:=Waiting ∪ {t}
      Passed:=Passed ∪ {s}
   }
```

Udforskning af tilstandrum

Forward Reachability Analysis

Erklæret tilstandsrum



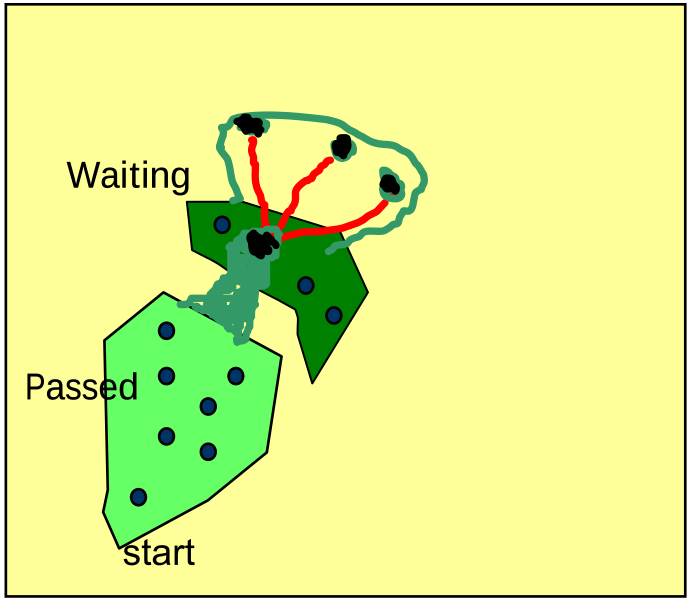
```
Passed:=Ø
Waiting:={$s_0$}
While(Waiting!=Ø)
   {
   select s ∈ Waiting
   Waiting:=Waiting\{s}
   if s∉Passed
      whenever (s → t) then
         Waiting:=Waiting ∪ {t}
      Passed:=Passed ∪ {s}
   }
```

CISS

Udforskning af tilstandrum

Forward Reachability Analysis

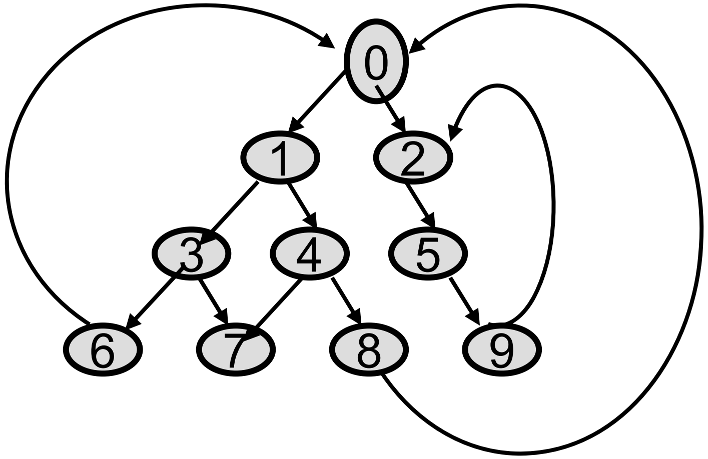
Erklæret tilstandsrum



```
Passed:=Ø
Waiting:={s_0}
While(Waiting!=Ø)
   {
   select s ∈ Waiting
   Waiting:=Waiting\{s}
   if s∉Passed
      whenever (s → t) then
         Waiting:=Waiting ∪ {t}
      Passed:=Passed ∪ {s}
   }
```

Udforskning af tilstandrum

Forward Reachability Analysis

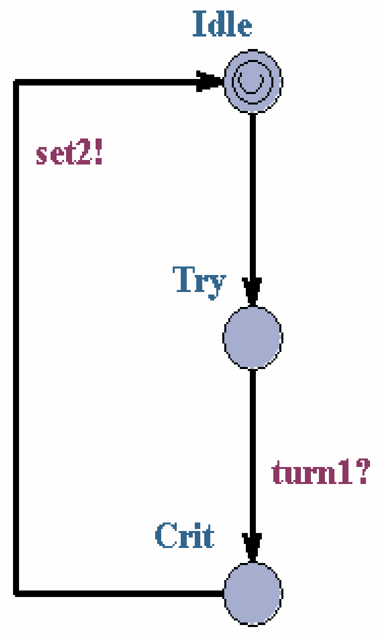


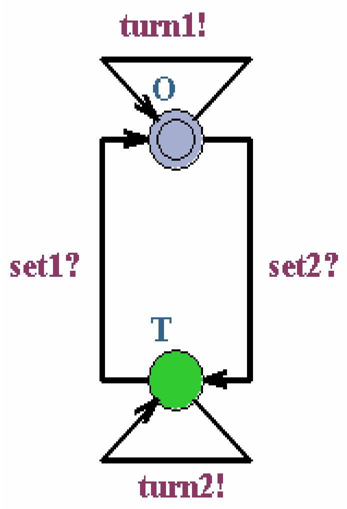
```
Passed:=Ø
Waiting:={s0}
While(Waiting!=Ø)
   {
   select s ∈ Waiting
   Waiting:=Waiting\{s}
   if s∉Passed
       whenever (s → t) then
          Waiting:=Waiting ∪ {t}
       Passed:=Passed ∪ {s}
   }
```

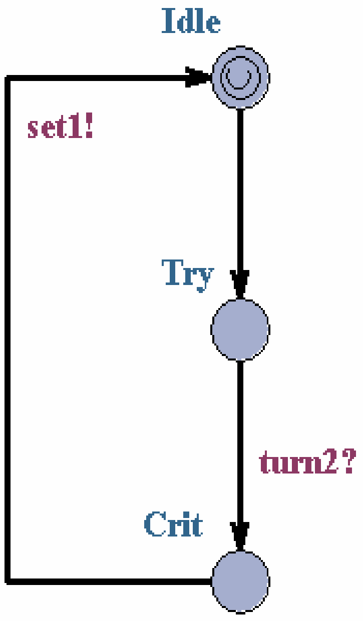
Depth-first search: organize Waiting as a **Stack**
Order: 0 1 3 6 7 4 8 2 5 9

Breadth-first search: organize Waiting as a **Queue**
Order: 0 1 2 3 4 5 6 7 8 9

Gensidig Udelukkelse



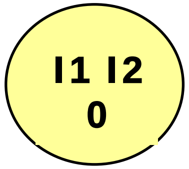


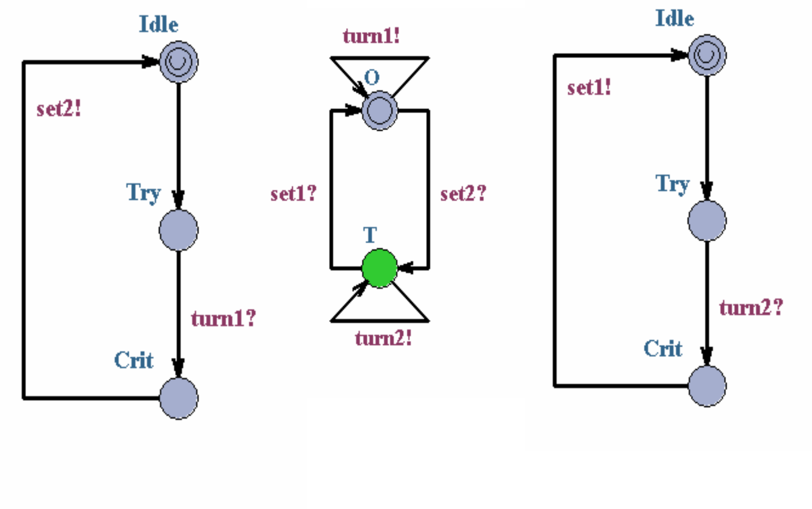


Turn

Gensidig udelukkelse

Forward Reachability

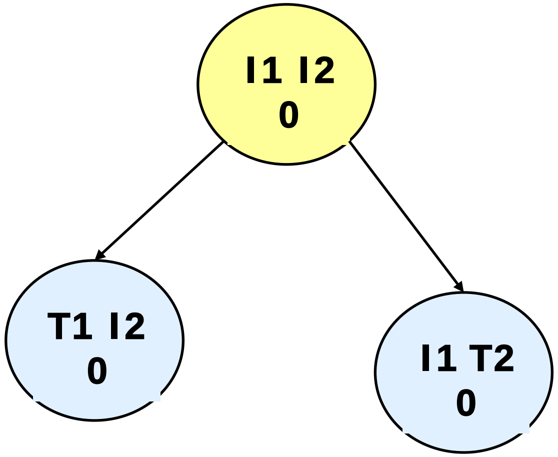


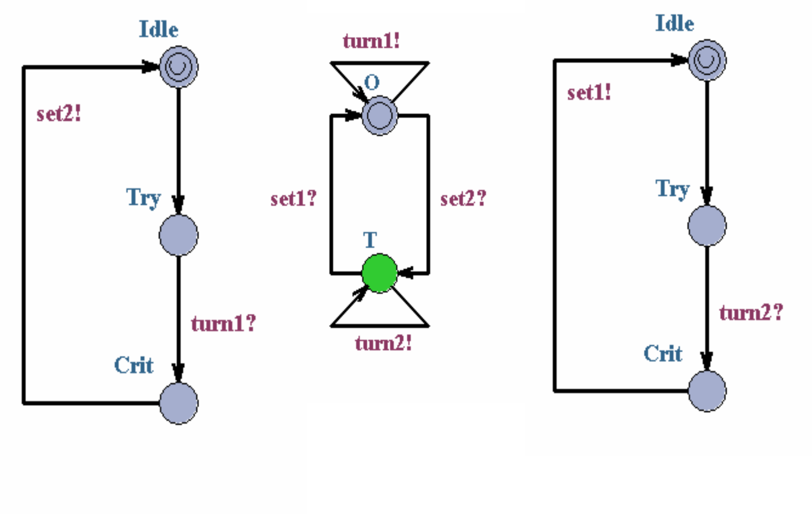


Turn

Gensidig udelukkelse

Forward Reachability

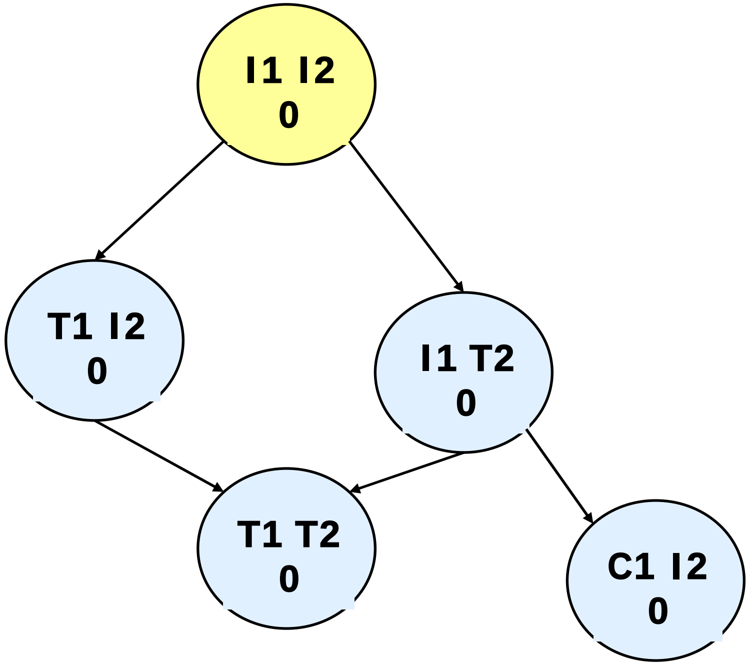




Turn

Gensidig udelukkelse

Forward Reachability



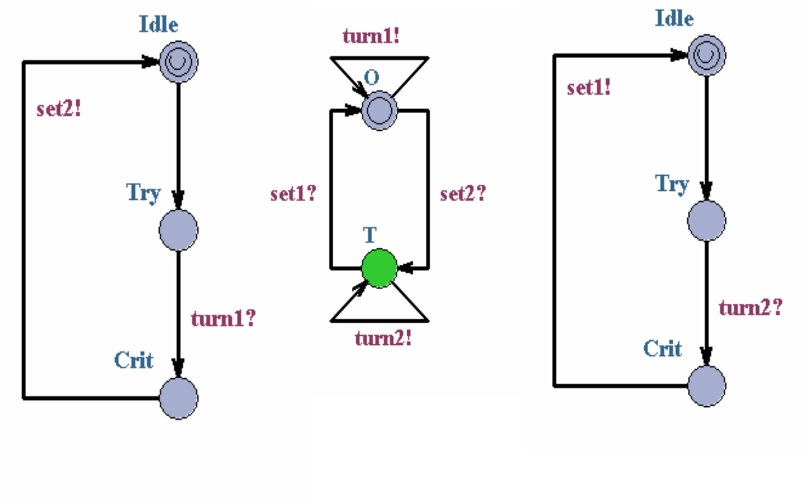
I1 I2
0

T1 I2
0

I1 T2
0

T1 T2
0

C1 I2
0

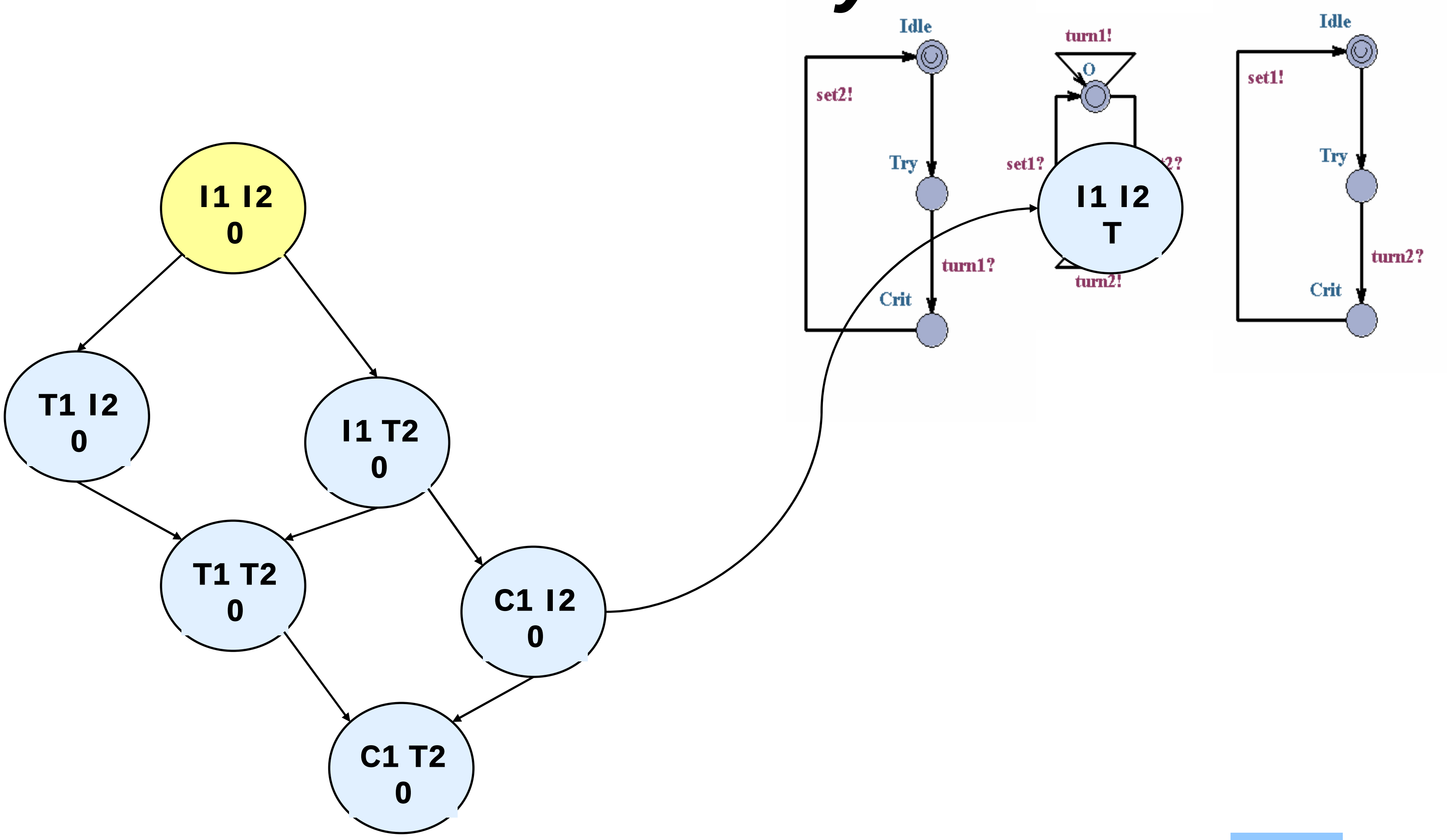


Turn

CISS

Gensidig udelukkelse

Forward Reachability

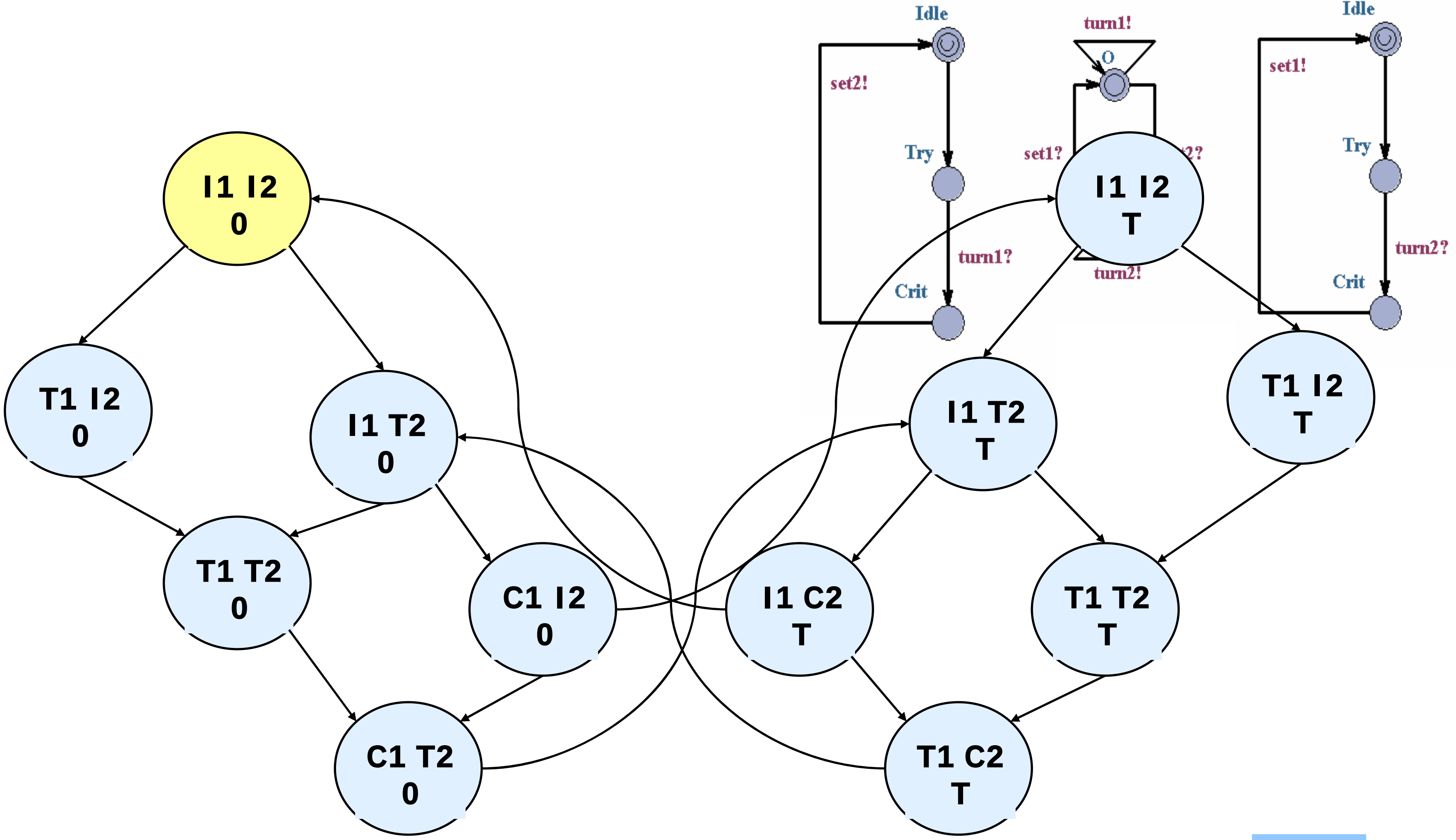


Turn

CISS

Gensidig udelukkelse

Forward Reachability

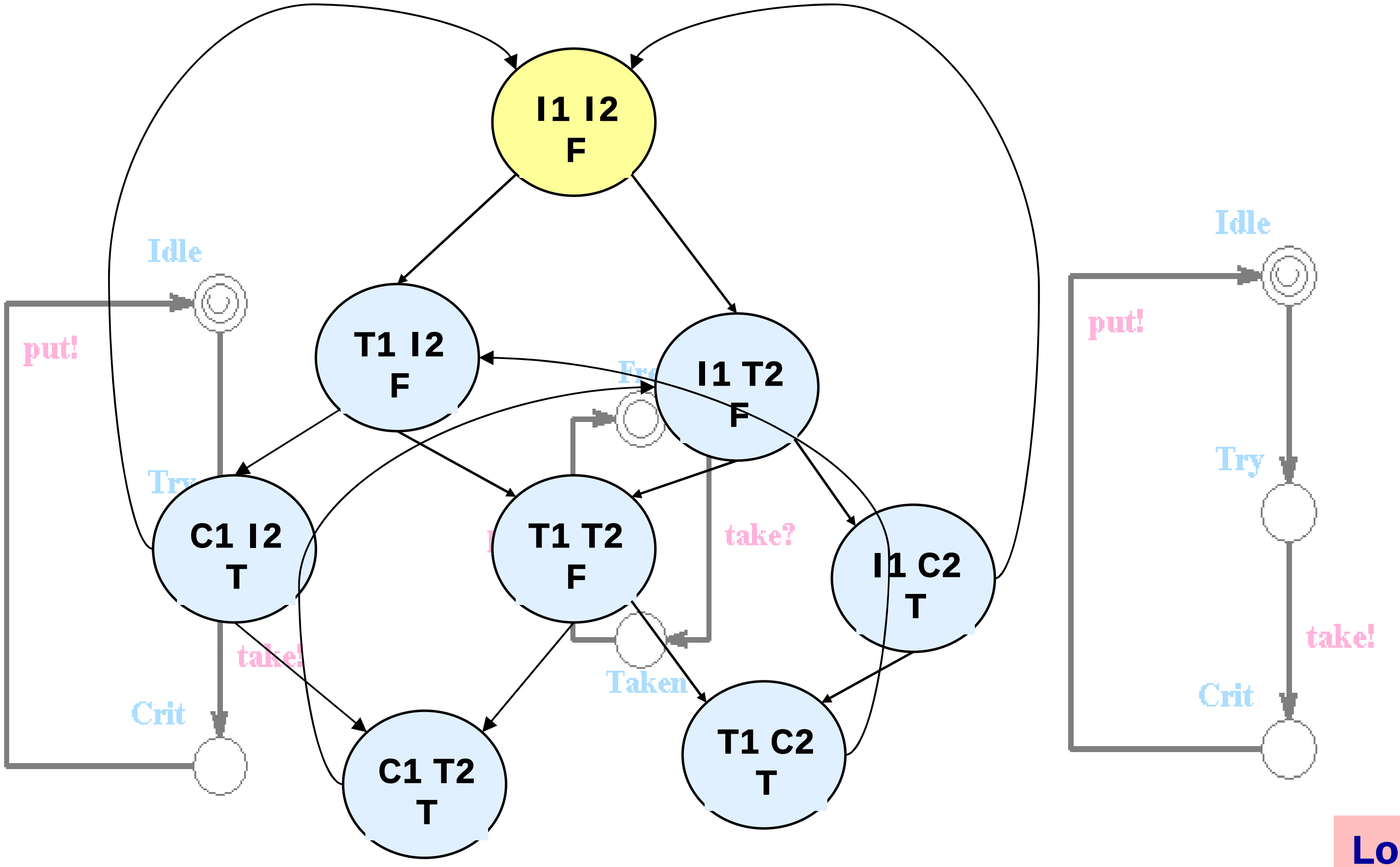


Turn

CISS

Gensidig udelukkelse

Forward Reachability



Lock

Generiske egenskaber

- Non-determinisme
- Tilstande der ikke aktiveres
- Transitioner der ikke bruges
- Input der ikke processeres
- Output der ikke genereres
- Lokal deadlock
- System deadlock

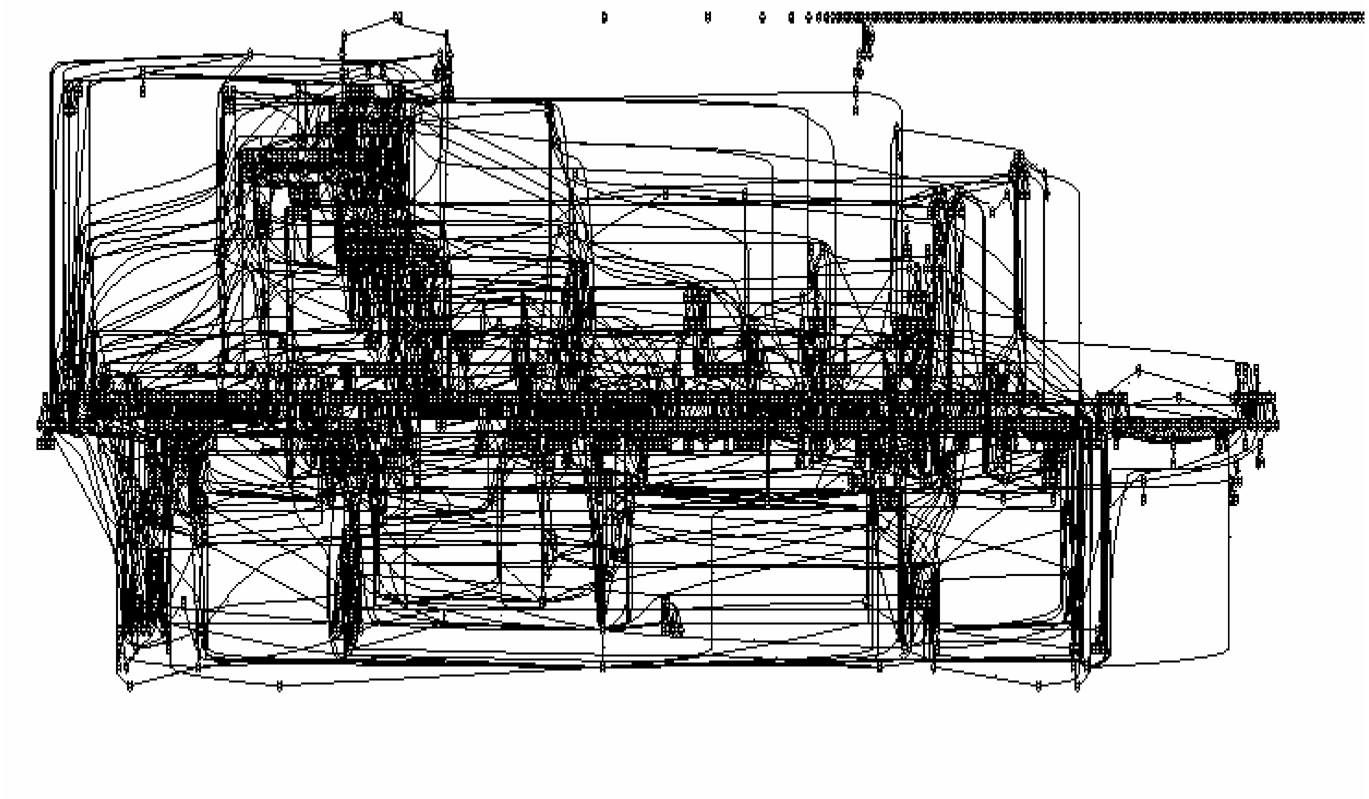
Kan alle reduceres til REACHABILITY

Train Simulator

VVS
visualSTATE

1421 machines
11102 transitions
2981 inputs
2667 outputs
3204 local states
Declare state sp.: 10^476

BUGS ?



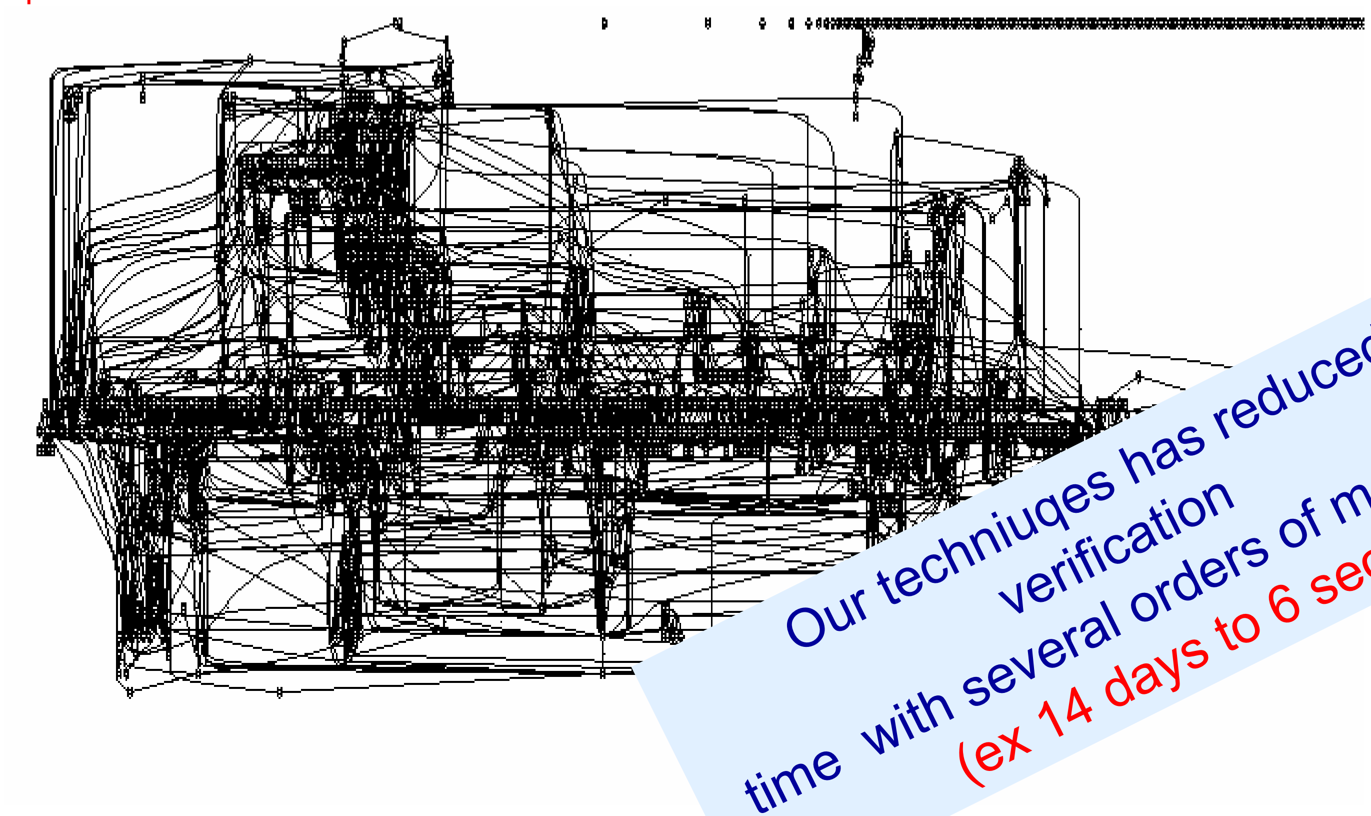
CISS

Train Simulator

VVS
visualSTATE

1421 machines
11102 transitions
2981 inputs
2667 outputs
3204 local states
Declare state sp.: 10^476

BUGS ?



Our techniuqes has reduced verification time with several orders of magnitude
(ex 14 days to 6 sec)

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.

AALBORG UNIVERSITY DENMARK

BRICS
Basic Research
in Computer Science

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Collaborators

@AALborg

- Kim G Larsen
- Gerd Behrman
- Arne Skou
- Brian Nielsen
- Alexandre David
- Jacob Illum Rasmussen
- Marius Mikucionis

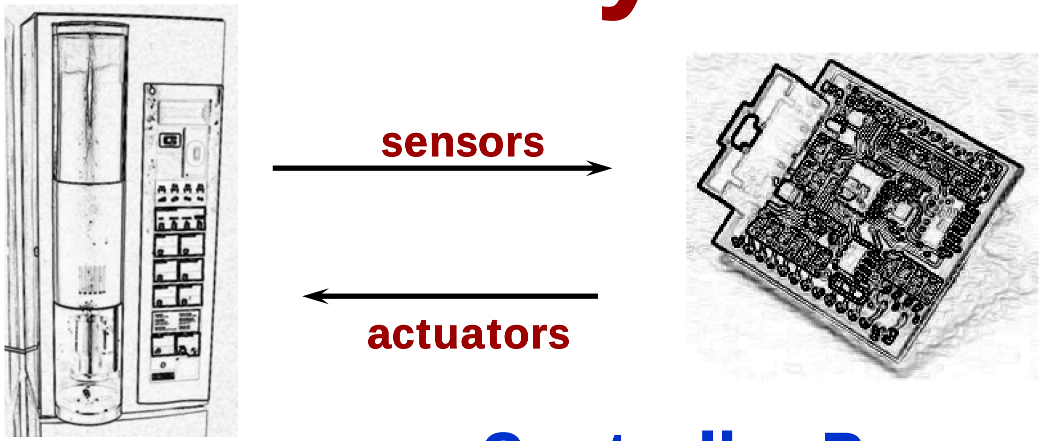
@UPPsala

- Wang Yi
- Paul Pettersson
- John Håkansson
- Anders Hessel
- Pavel Krcal
- Leonid Mokrushin
- Shi Xiaochun

@Elsewhere

- Emmanuel Fleury, Didier Lime, Johan Bengtsson, Fredrik Larsson, Kåre J Kristoffersen, Tobias Amnell, Thomas Hune, Oliver Möller, Elena Fersman, Carsten Weise, David Griffioen, Ansgar Fehnker, Frits Vandraager, Theo Ruys, Pedro D'Argenio, J-P Katoen, Jan Tretmans, Judi Romijn, Ed Brinksma, Martijn Hendriks, Klaus Havelund, Franck Cassez, Magnus Lindahl, Francois Laroussinie, Patricia Bouyer, Augusto Burgueno, H. Bowmann, D. Latella, M. Massink, G. Faconti, Kristina Lundqvist, Lars Asplund, Justin Pearson...

Real Time Systems



sensors

actuators

Plant

Continuous

Controller Program

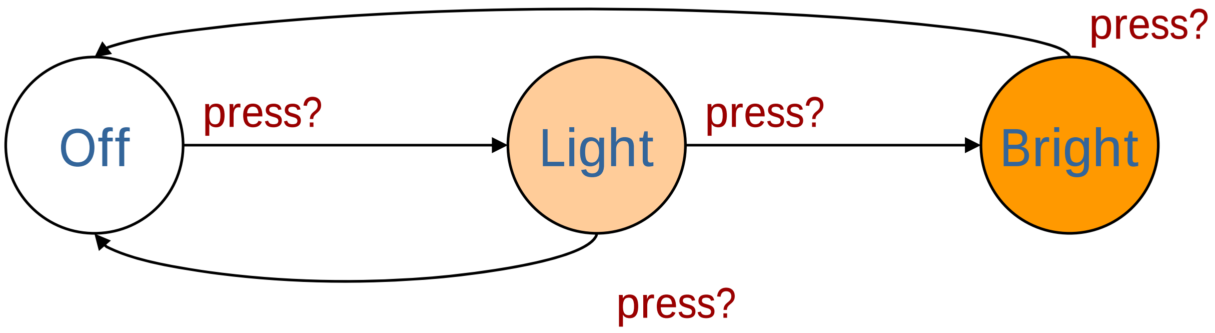
Discrete

Eg.: Realtime Protocols
Pump Control
Air Bags
Robots
Cruise Control
ABS
CD Players
Production Lines

Real Time System

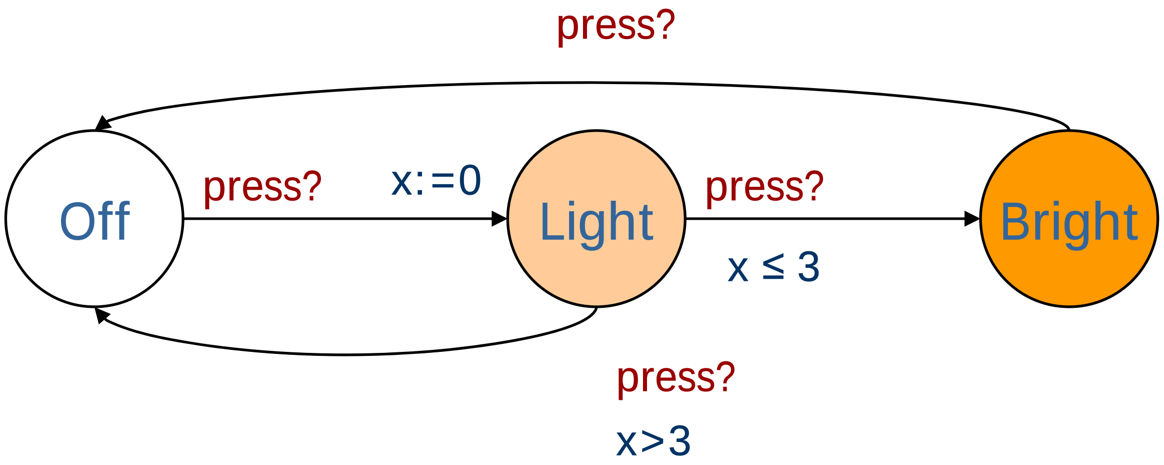
A system where correctness not only depends on the logical order of events but also on their **timing**!!

Dumb Light Control



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

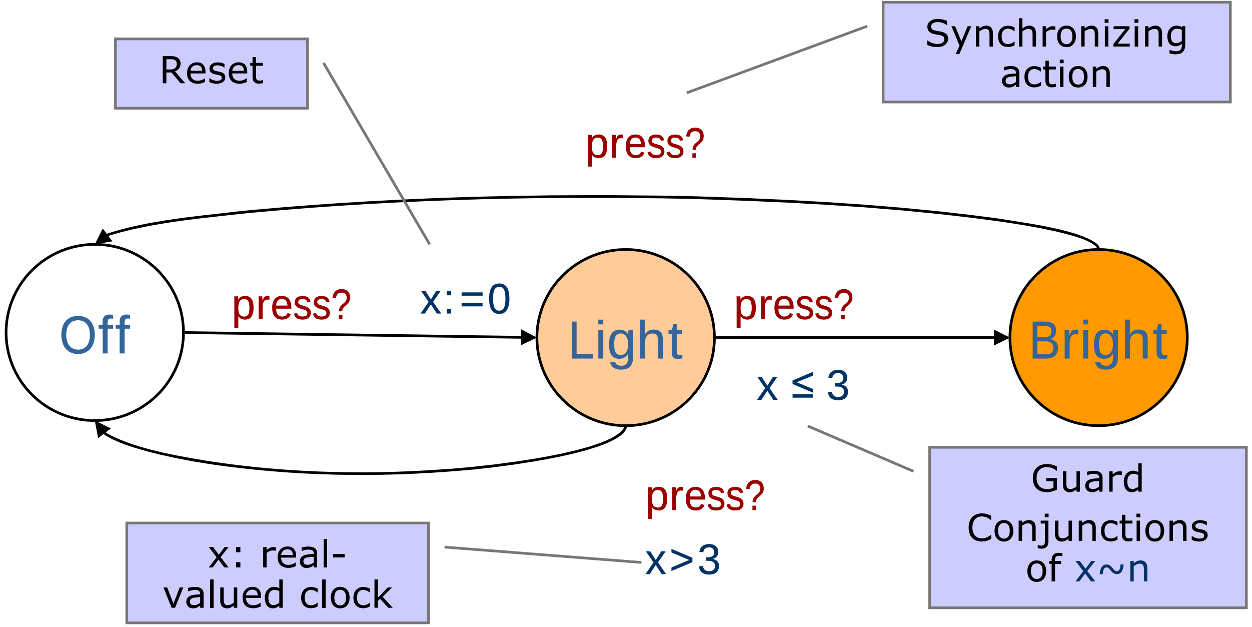
Dumb Light Control *Alur & Dill 1990*



Solution: Add real-valued clock **x**

Timed Automata

Alur & Dill 1990



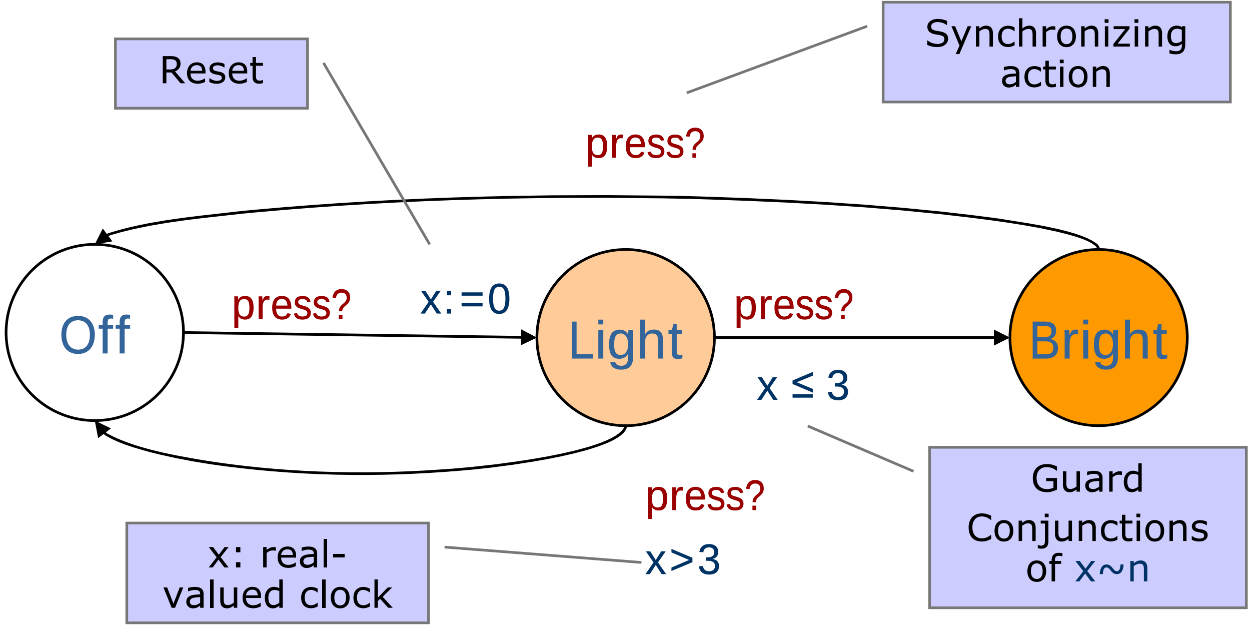
States:

(location , x=v) where v ∈ **R**

Transitions:

(Off , x=0)

Timed Automata *Alur & Dill 1990*



States:

(location , x=v) where v ∈ **R**

Transitions:

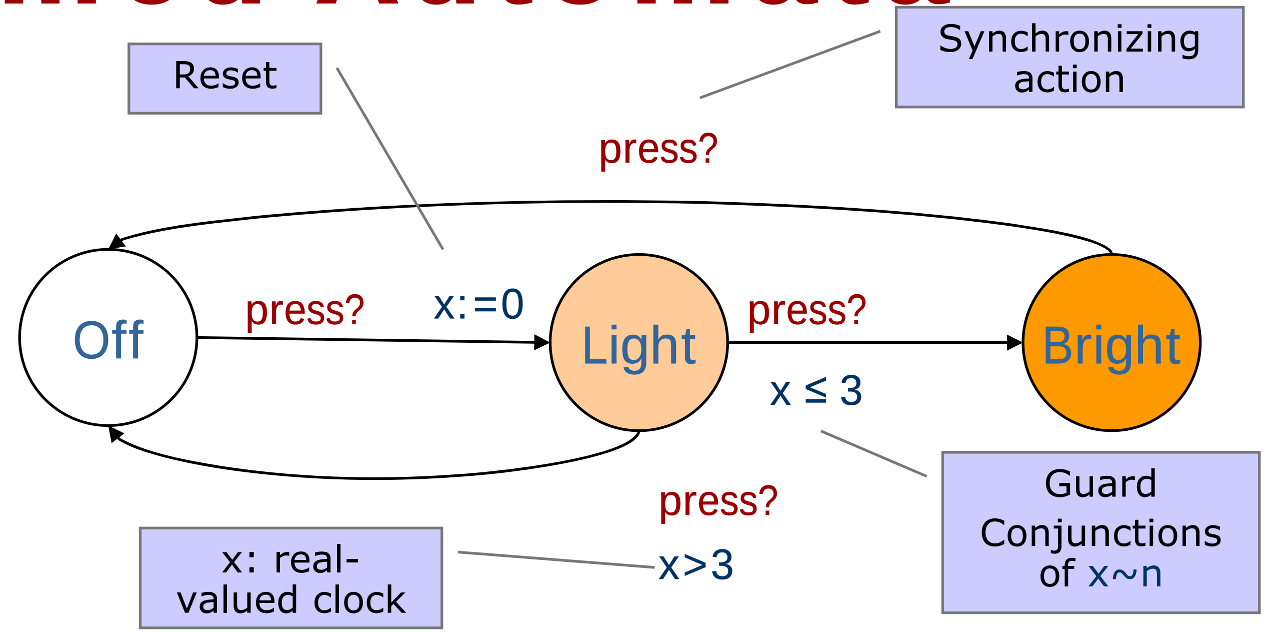
(Off , x=0)

delay 4.32 → (Off , x=4.32)

CISS

Timed Automata

Alur & Dill 1990



States:

(location , x=v) where v ∈ **R**

Transitions:

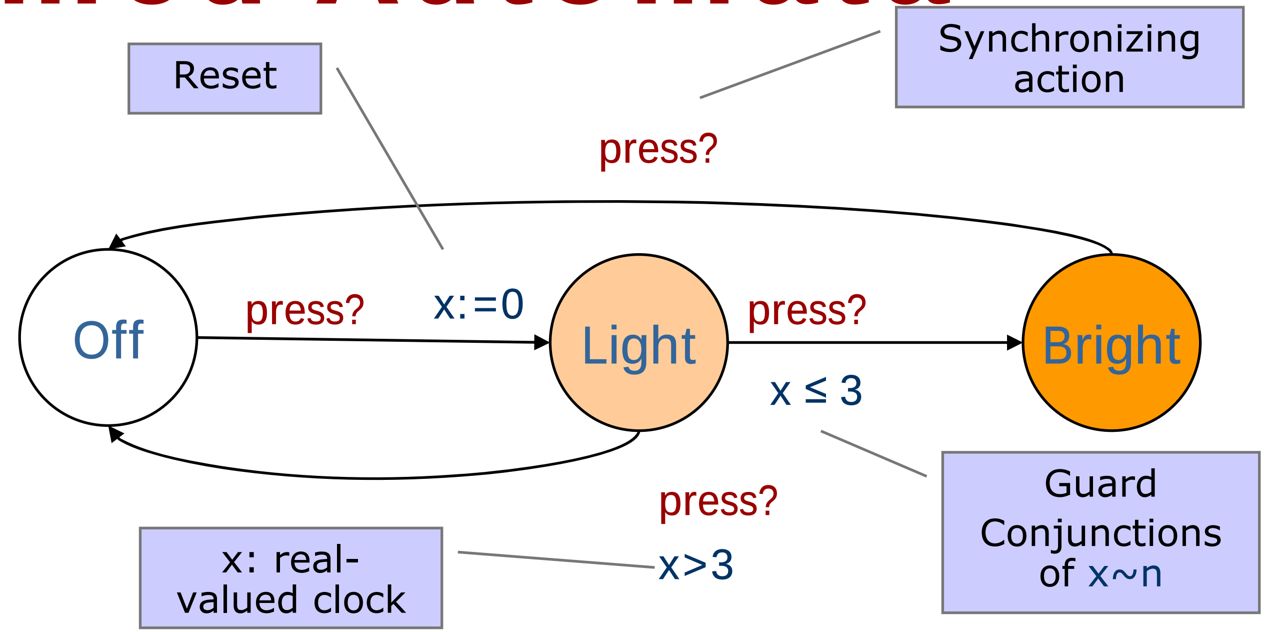
(Off , x=0)

delay 4.32 → (Off , x=4.32)

press? → (Light , x=0)

Timed Automata

Alur & Dill 1990



Reset

Synchronizing action

Guard
Conjunctions of x~n

x: real-valued clock

States:

(location , x=v) where v $\in$ **R**

Transitions:

(Off , x=0)

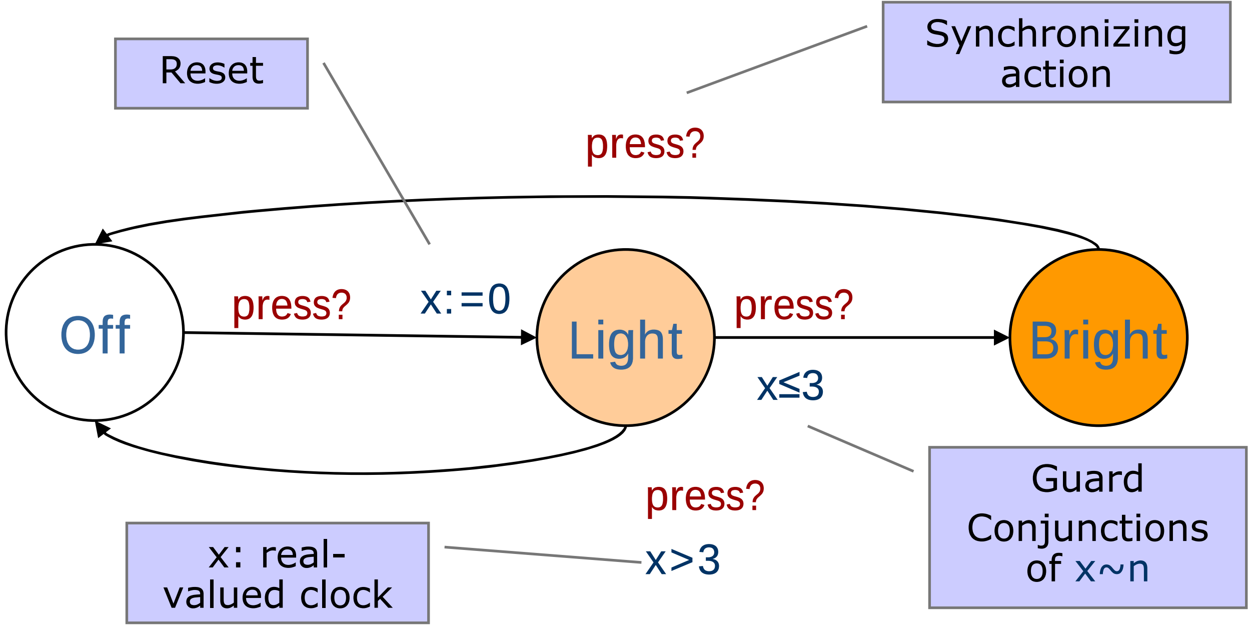
delay 4.32 → (Off , x=4.32)

press? → (Light , x=0)

delay 2.51 → (Light , x=2.51)

Timed Automata

Alur & Dill 1990



Reset

Synchronizing action

Guard Conjunctions of x~n

x: real-valued clock

States:

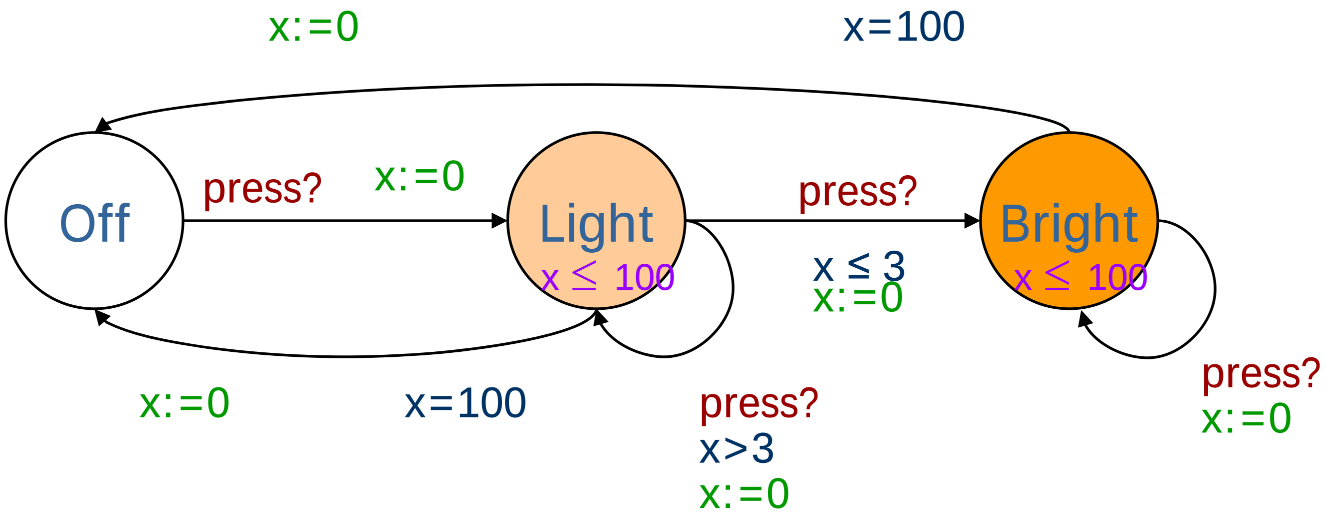
(location , x=v) where v ∈ **R**

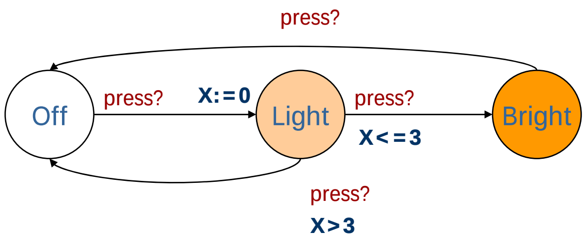
Transitions:

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

Intelligent Light Control

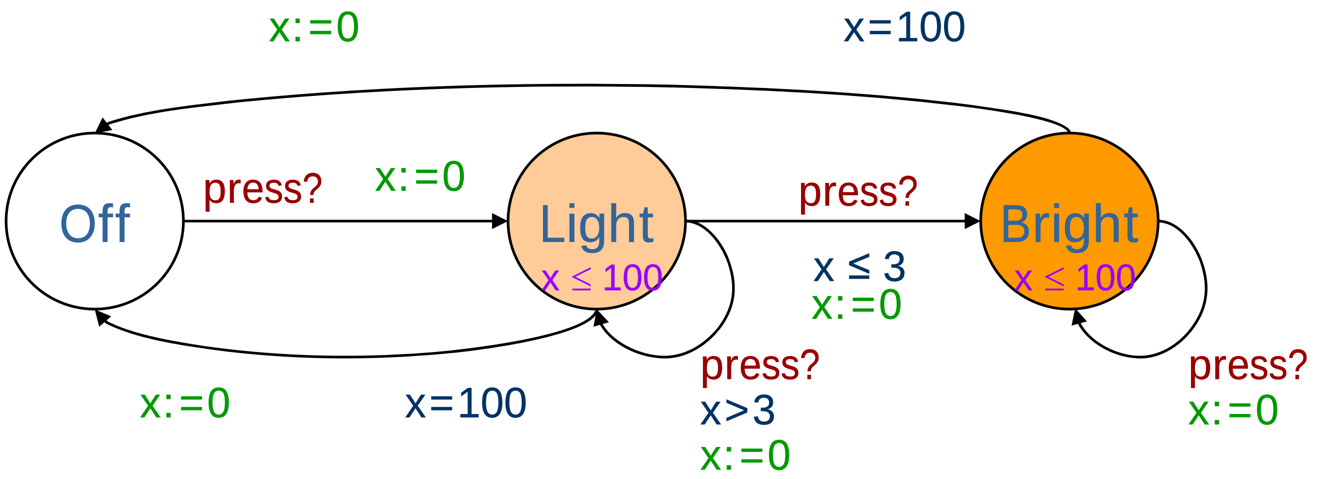
Using Invariants





Intelligent Light Control

Using Invariants



Transitions:

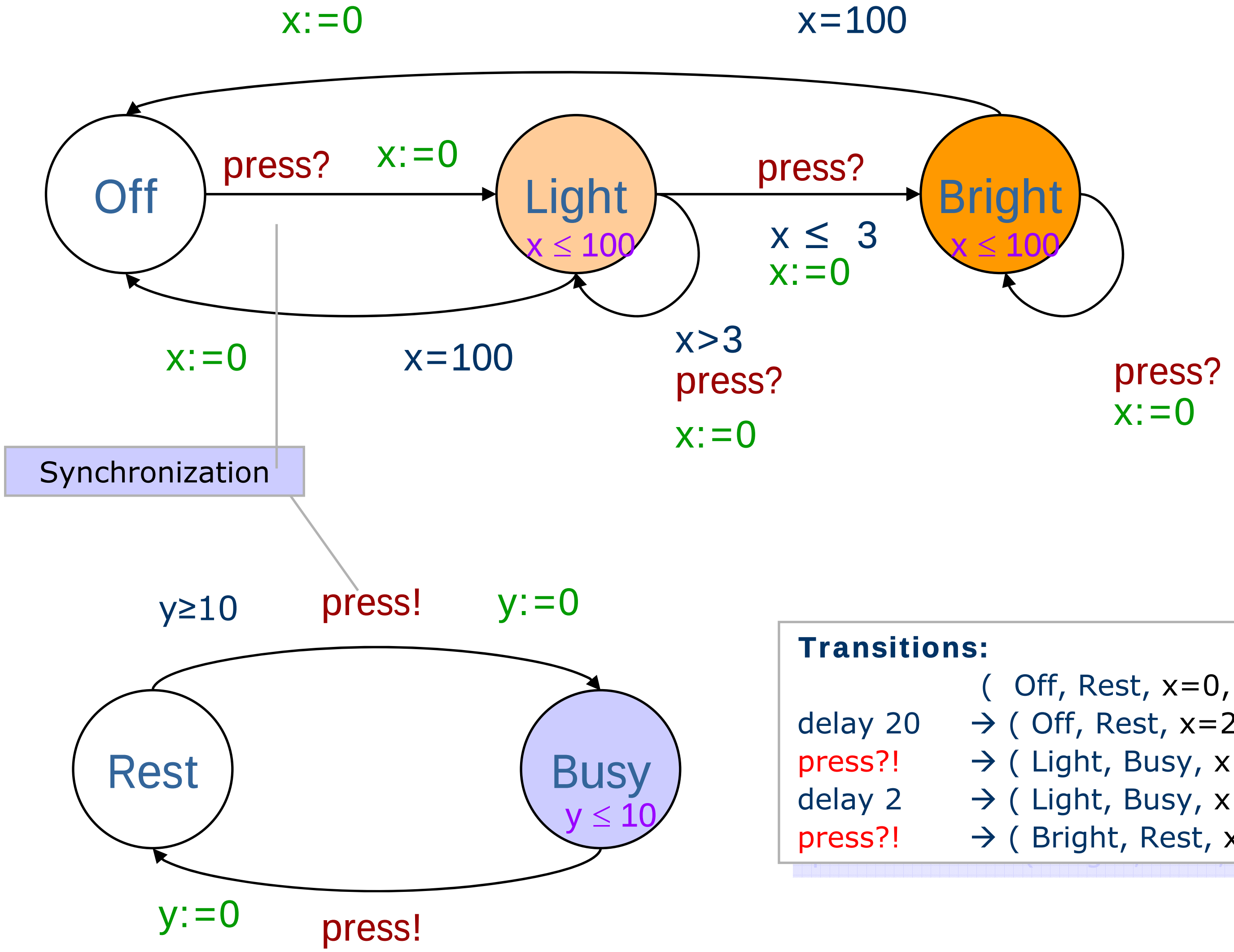
	(Off , x=0)
delay 4.32	→ (Off , x=4.32)
press?	→ (Light , x=0)
delay 4.51	→ (Light , x=4.51)
press?	→ (Light , x=0)
delay 100	→ (Light , x=100)
τ	→ (Off , x=0)

Note:
(Light , x=0) delay 103 → X

Invariants ensures progress

CISS

Light Controller || User



Transitions:

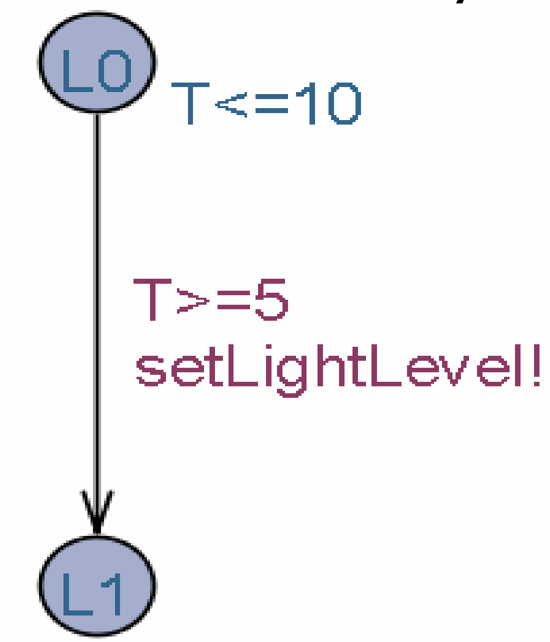
	(Off, Rest, x=0, y=0)
delay 20	→ (Off, Rest, x=20, y=20)
press?!	→ (Light, Busy, x=0, y=0)
delay 2	→ (Light, Busy, x=2, y=2)
press?!	→ (Bright, Rest, x=0, y=0)

CISS

Timing Uncertainty

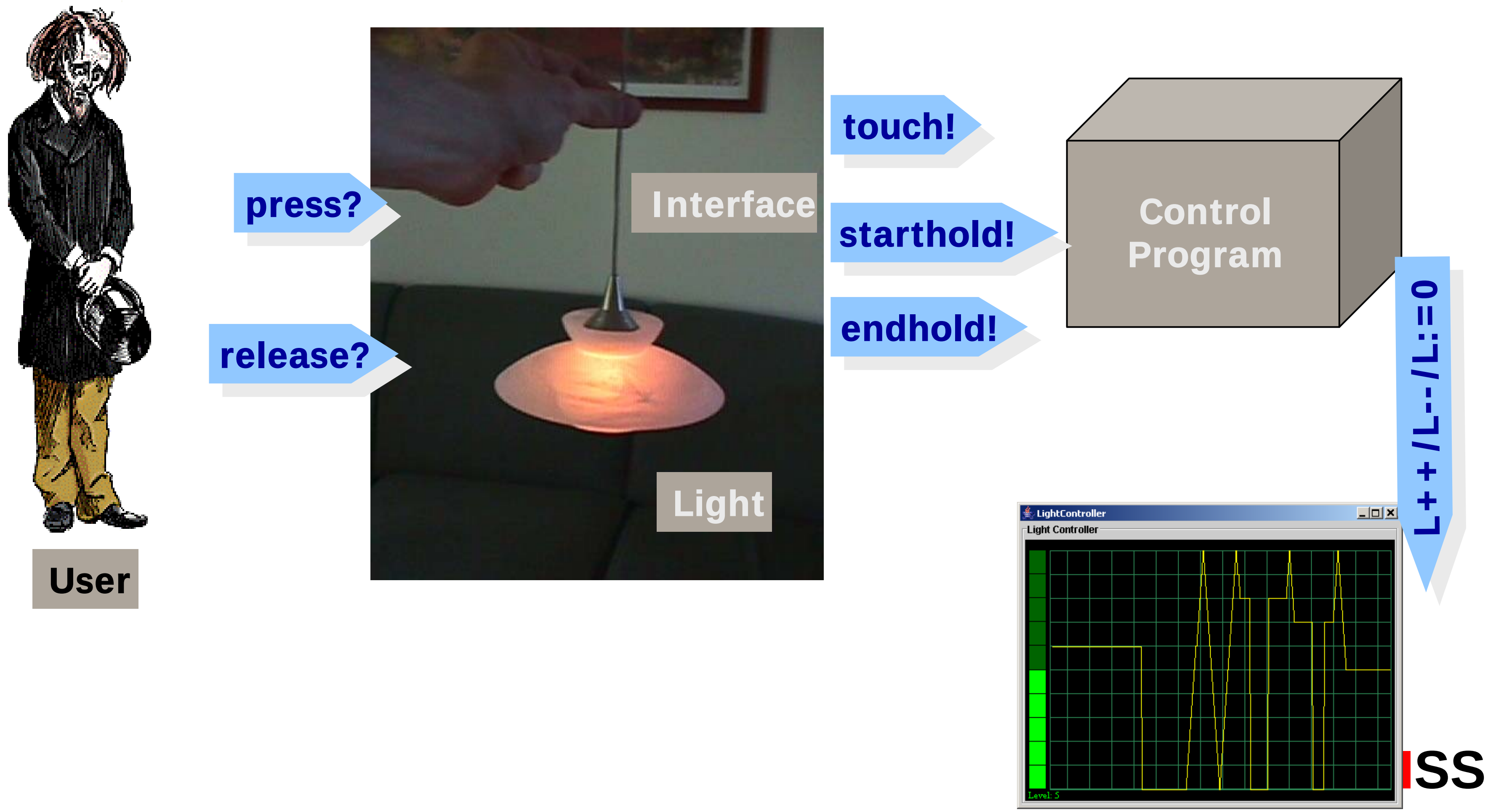
- Unpredictable or variable
  - response time,
  - computation time
  - transmission time etc:

- Initially T=0



LightLevel must be adjusted
between 5 and 10

Light Control Interface



User

press?

release?

Interface

Light

touch!

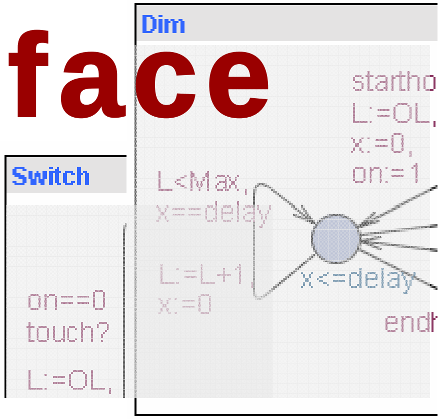
starthold!

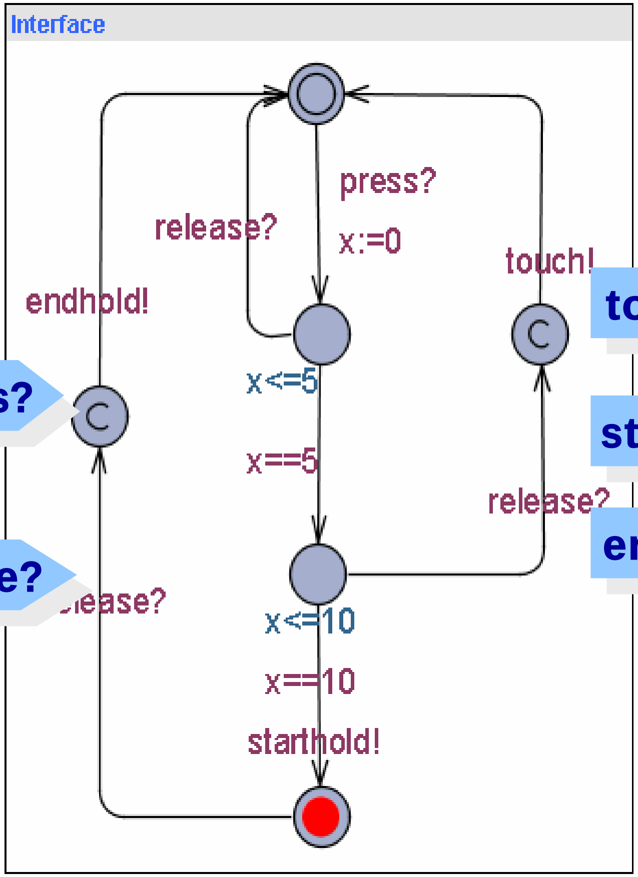
endhold!

Control Program

L++/L--/L:=0

Light Control Interface





User

press?

release?

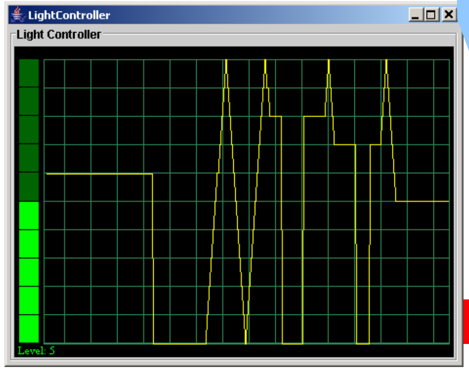
touch!

starthold!

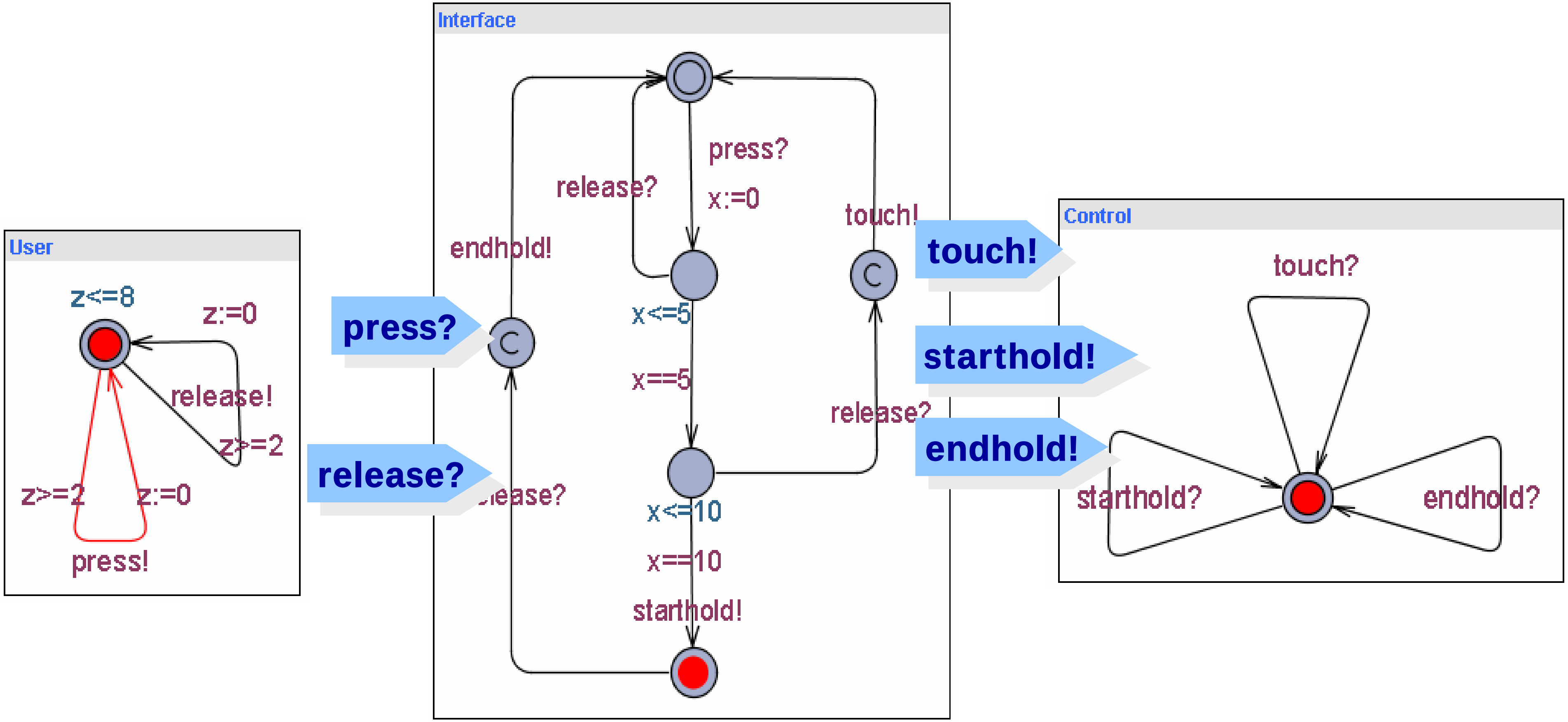
endhold!

Control Program

L++/L--/L:=0



Light Control Network



Broad-casts

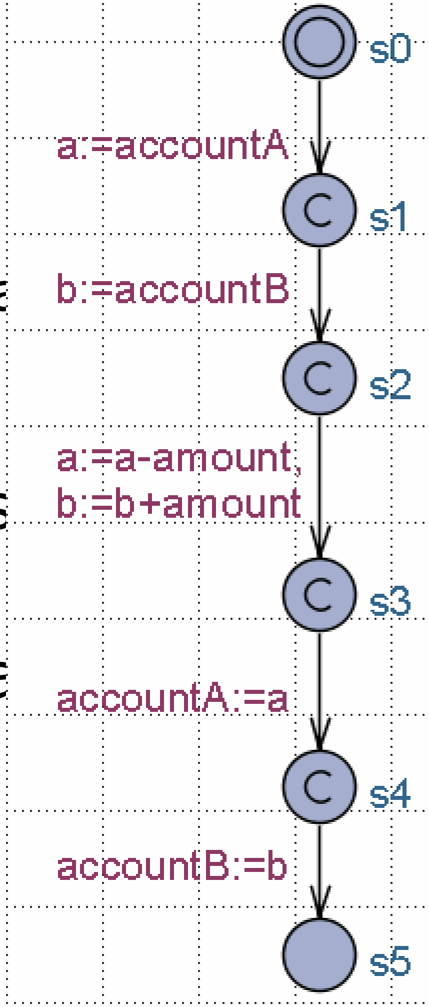
- chan coin, cof, cofBut;
- broadcast chan join;
  - sending: output join!
  - every automaton that listens to join moves
  - ie. every automaton with enabled “join?” transition moves in one step
  - may be zero!

Comitted Locations

- Locations marked **C**
  - ***No delay*** in committed location.
  - Next transition must involve automata in ***committed location.***

- Handy to model atomic sequences
- The use of committed locations <u>reduces</u> the number of states in a model, <u>and</u> allows for more space and time efficient analysis.

- S0 to s5 executed atomically



CISS

Urgent Channels and Locations

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

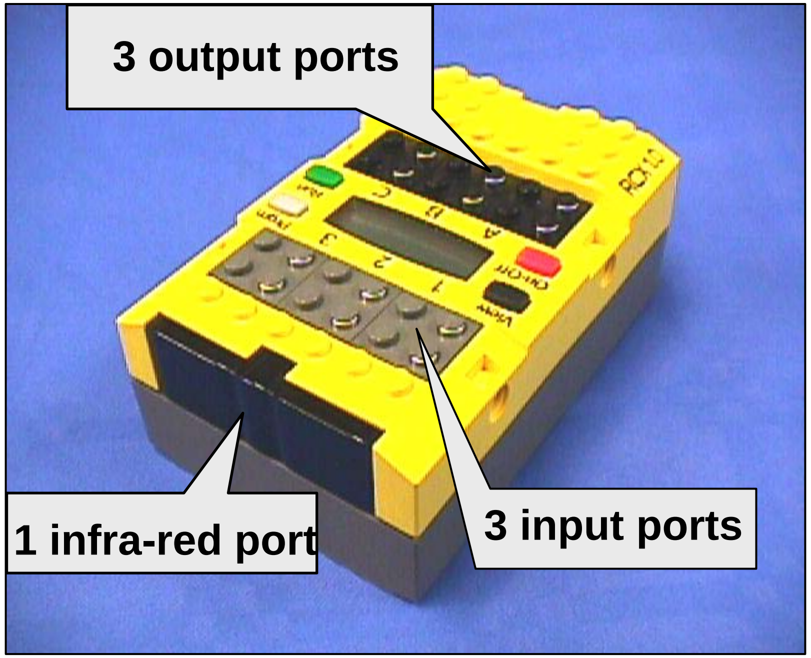
Other Uppaal features

- Bounded domain
  - Int [1..4] a;
- C-like data-structures and user defined functions in declaration section
  - structs, arrays, and typedef
- **`select a:T`** construct
- `Forall`, `exists` in expr
- Scalar sets (for giving unique ID’s)
- Process and channel **priorities**
- Value passing (emulation)

BRICK SORTING

LEGO Mindstorms/RCX

- Sensors: temperature, light, rotation, pressure.
- Actuators: motors, lamps,
- Virtual machine:
  - 10 tasks, 4 timers, 16 integers.



- Several Programming Languages:
  - NotQuiteC, Mindstorm, Robotics, legOS, etc.

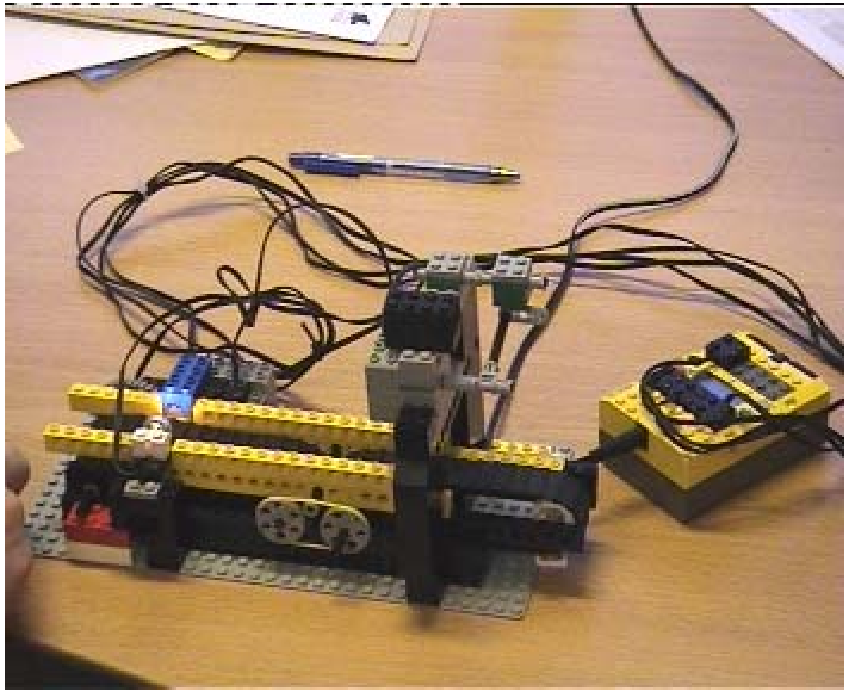
A Real Timed System

The Plant

Conveyor Belt

&

Bricks



Controller Program

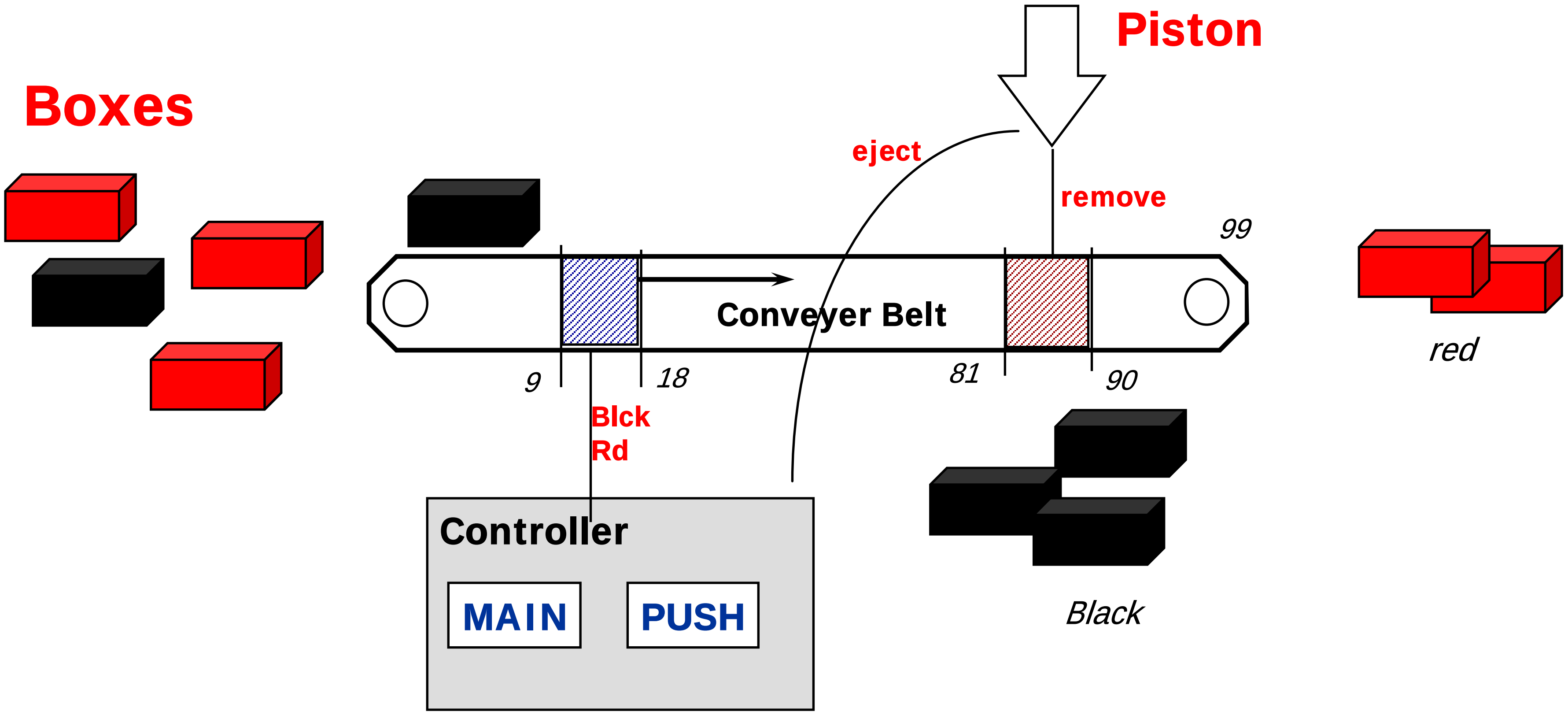
LEGO MINDSTORM

What is suppose to happen?

First UPPAAL model

Sorting of Lego Boxes

Ken Tindell



Exercise: Design **Controller** so that only black boxes are being pushed out

NQC programs

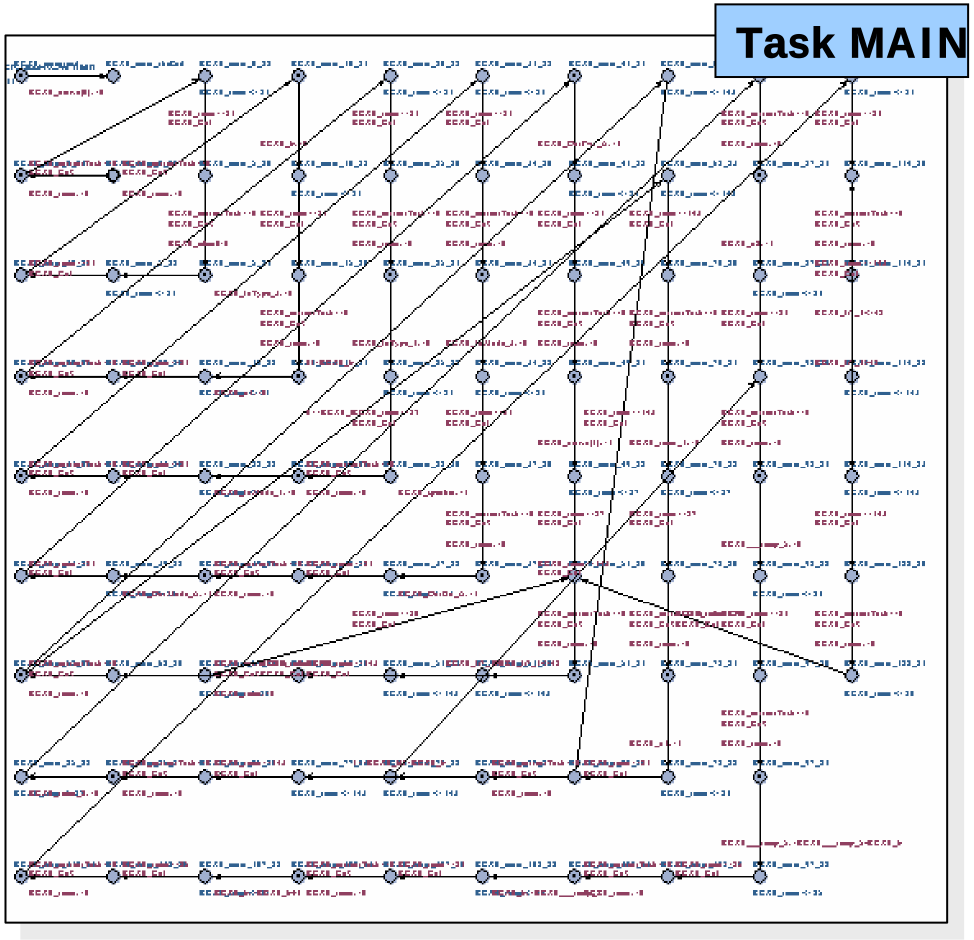
```
int active;
int DELAY;
int LIGHT_LEVEL;
```

```
task MAIN{
 DELAY=75;
 LIGHT_LEVEL=35;
 active=0;
 Sensor(IN_1, IN_LIGHT);
 Fwd(OUT_A,1);
 Display(1);

 start PUSH;

 while(true){
   wait(IN_1<=LIGHT_LEVEL);
   ClearTimer(1);
   active=1;
   PlaySound(1);
   wait(IN_1>LIGHT_LEVEL);
  }
}
```

```
task PUSH{
  while(true){
    wait(Timer(1)>DELAY && active==1);
    active=0;
    Rev(OUT_C,1);
    Sleep(8);
    Fwd(OUT_C,1);
    Sleep(12);
    Off(OUT_C);
  }
}
```

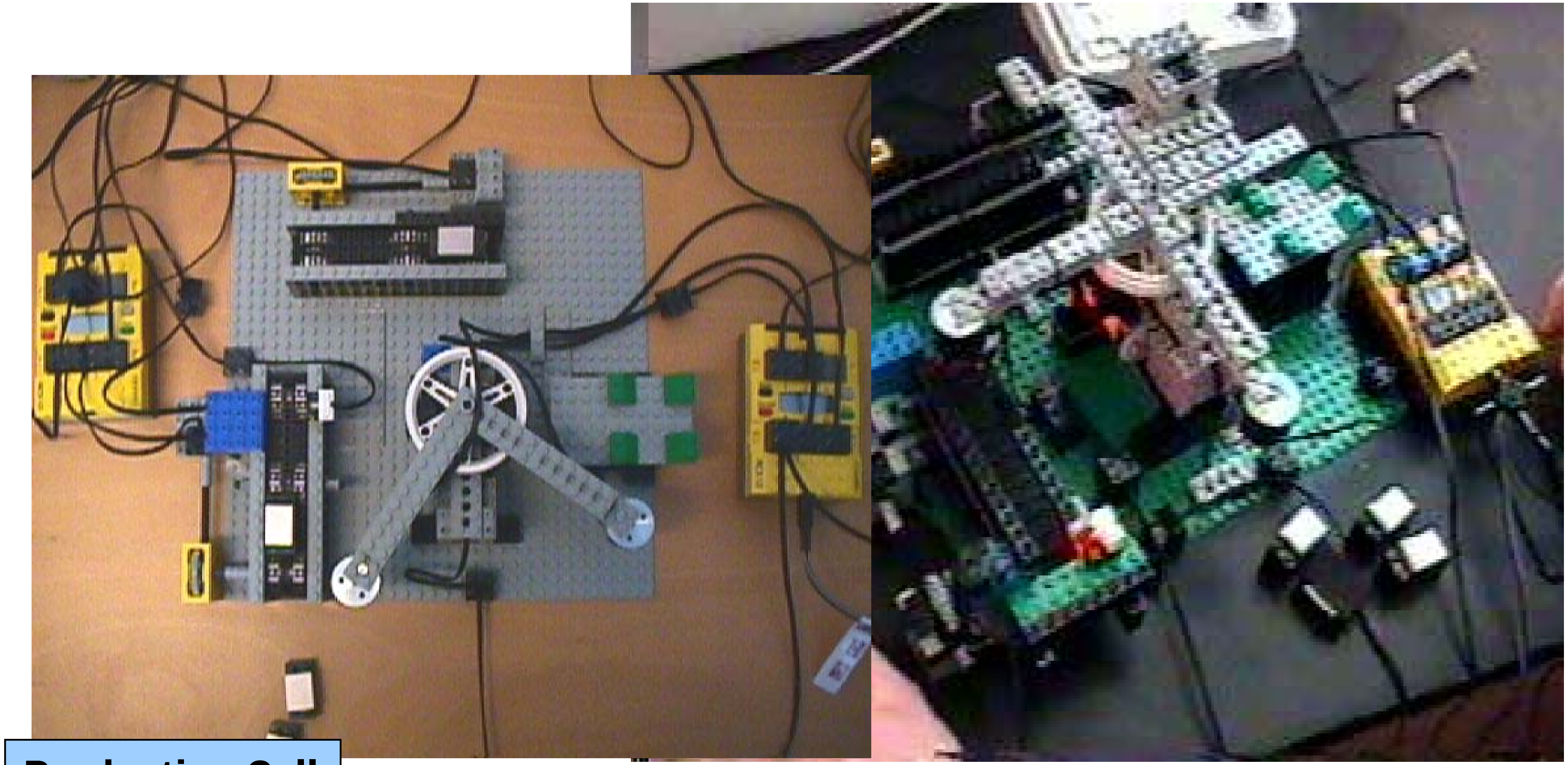
From RCX to UPPAAL

- Model includes Round-Robin Scheduler.
- Compilation of RCX tasks into TA models.
- Presented at ECRTS 2000



The Production Cell

Course at DTU, Copenhagen



Production Cell