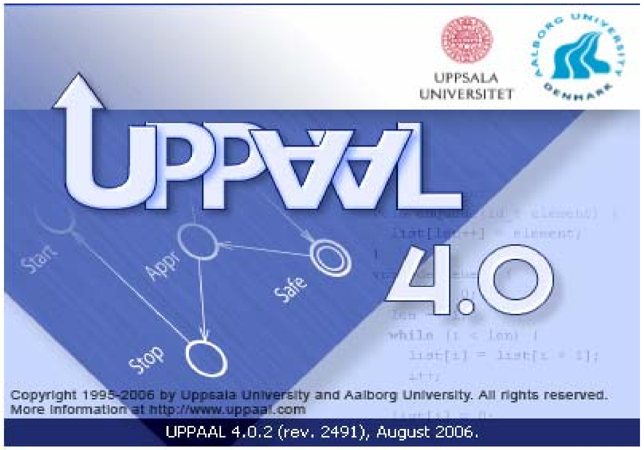


Copyright 1995-2006 by Uppsala University and Aalborg University. All rights reserved.
More information at http://www.uppaal.com

UPPAAL 4.0.2 (rev. 2491), August 2006.

Modeling & Verification

Of Real-Time Systems using UPPAAL

Kim G Larsen

AALBORG UNIVERSITY DENMARK

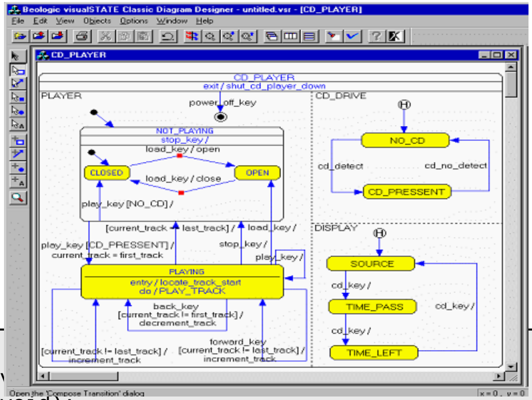
BRICS
Basic Research
in Computer Science

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

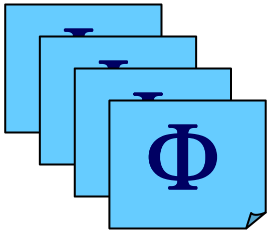
Verifikation og Test

Model



```
/* Wait for ev
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



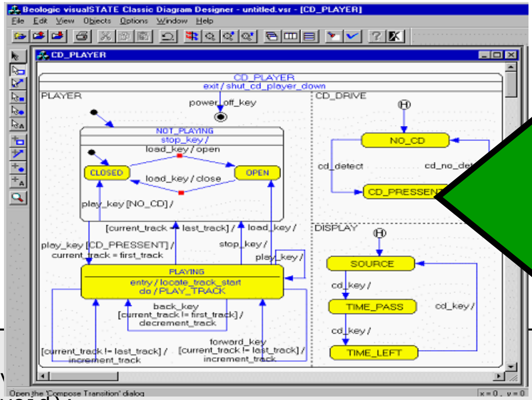
Spec

System

BRI
Basic Research
in Computer Science

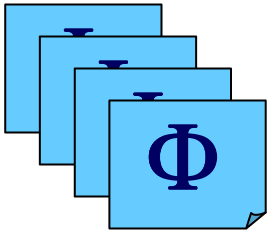
Verifikation og Test

Model



```
/* Wait for ev
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



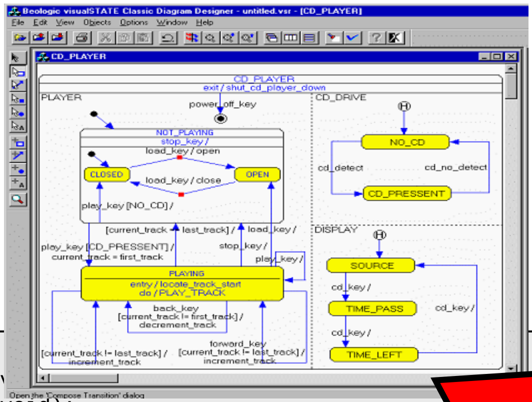
Spec

System

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

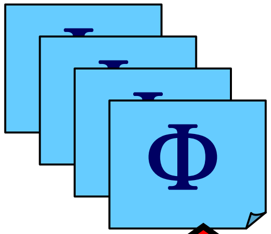
Verifikation og Test

Model



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

/* Operating system visualSTATE process. Mimics a c
 * visualSTATE system. In this implementation this i
 * 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

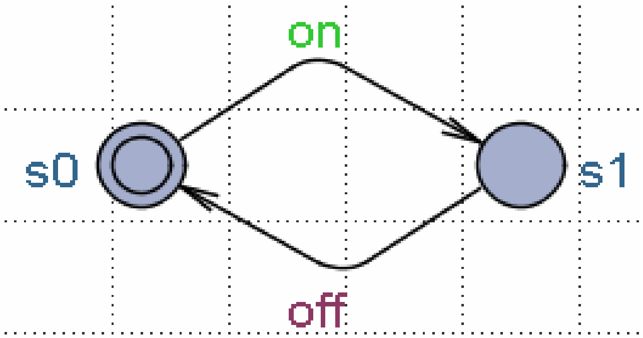
BRI
Basic Research
in Computer Science

Modelling Behaviour using State Machines

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.

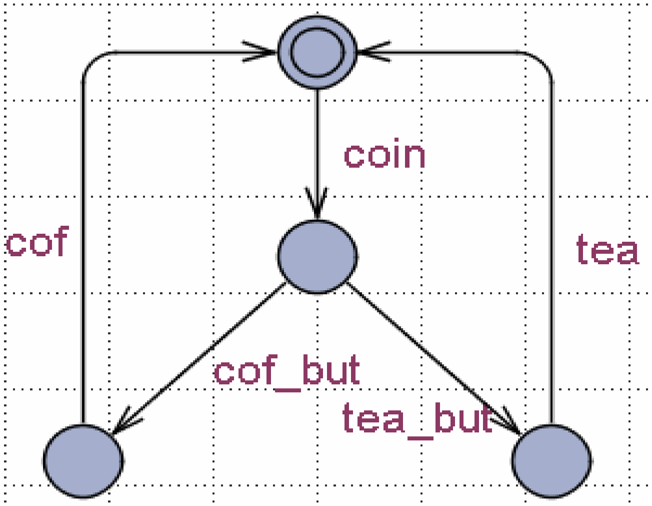
a light switch
LTS



on→off→on→off→on→off→ ……….

a sequence of actions or *trace*

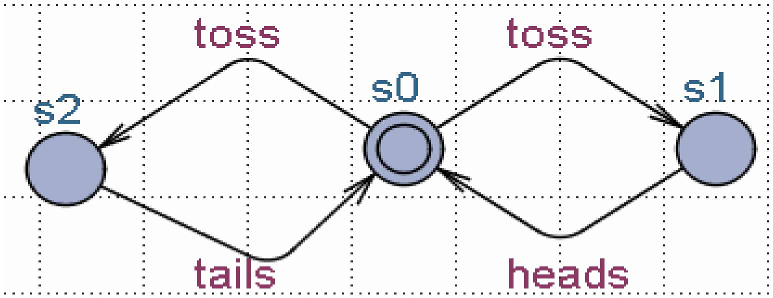
Modelling Choices



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

Non-deterministic Choice

- Tossing a coin

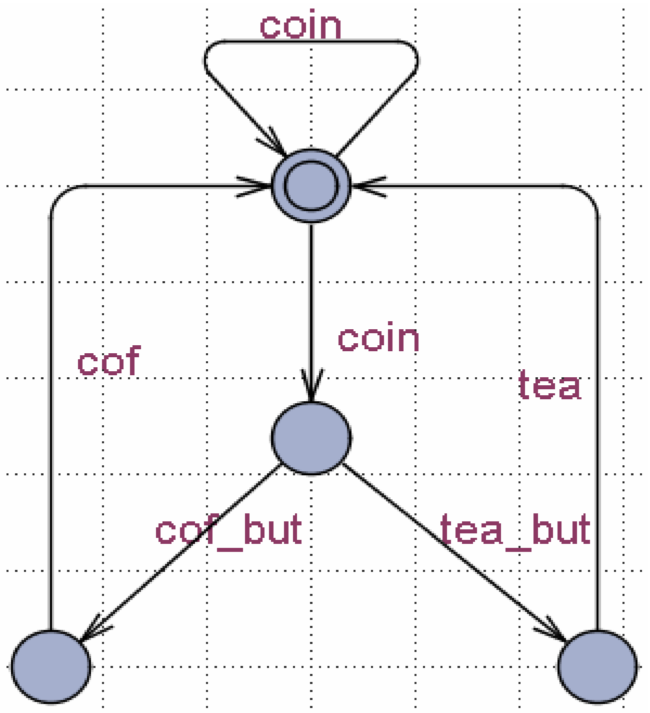


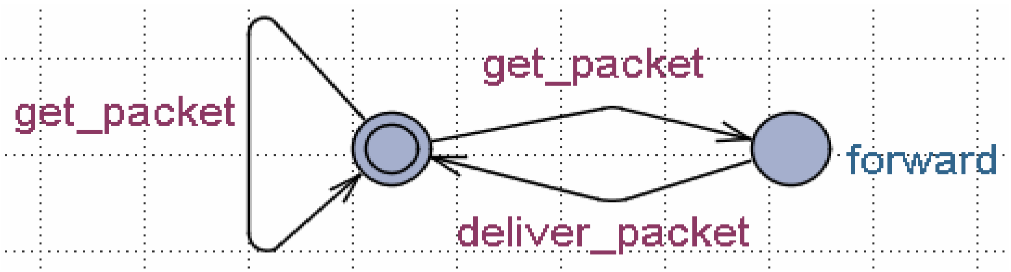
- Possible traces?
  - Both outcomes possible
  - Nothing said about relative frequency
  - If coin is fair, the outcome is 50/50

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?

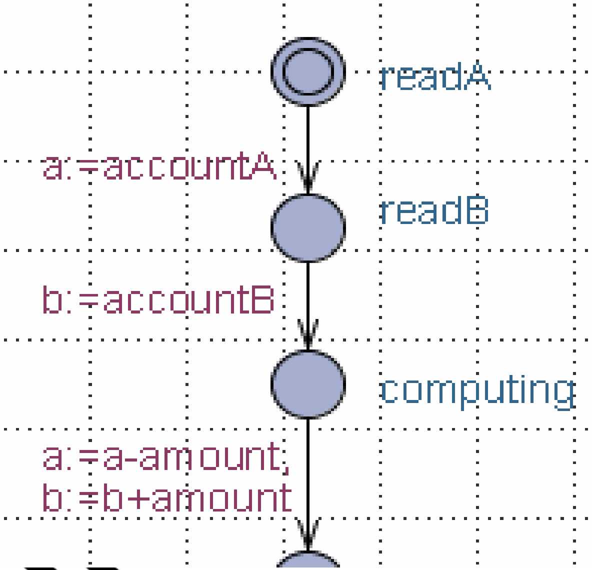
Use non-determinism...

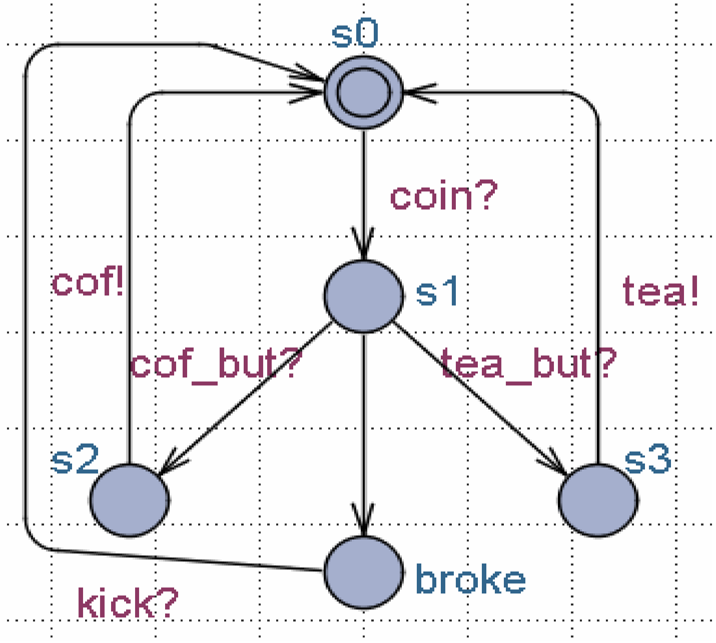


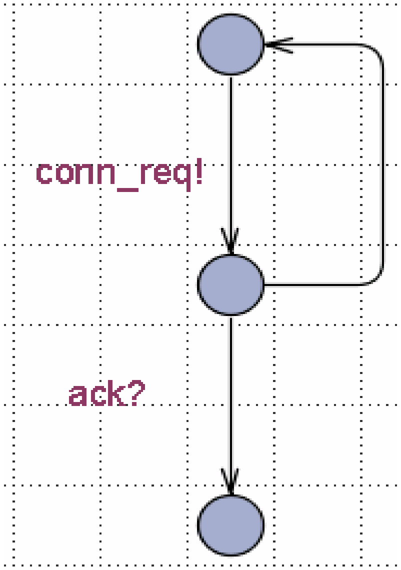


Internal-Actions

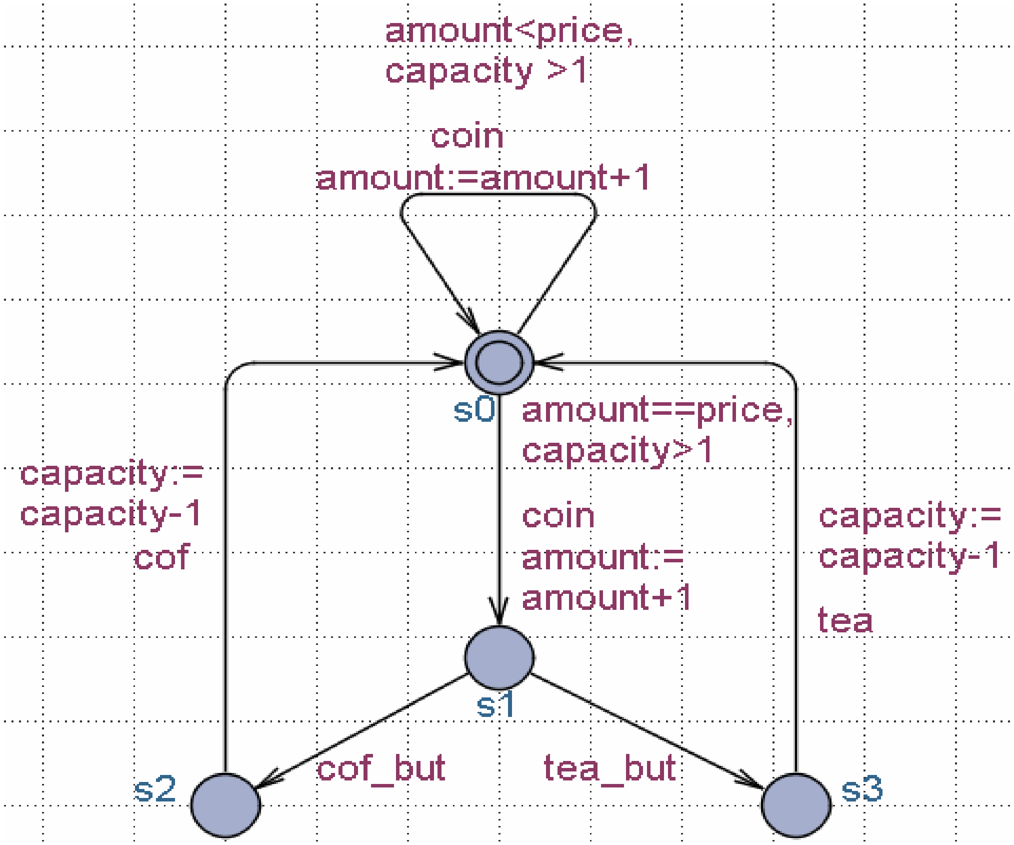
- Spontaneous actions
- Internal actions
- Tau-actions
- Internal transitions can be taken on the initiative of a single machine without communication with others







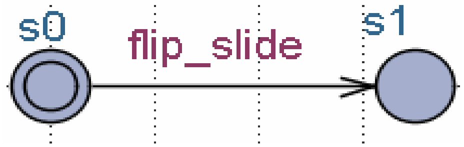
Extended FSM



- 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)

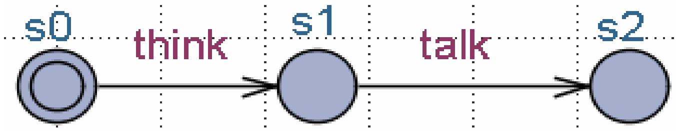
Parallel Composition: interleaving

Flipper



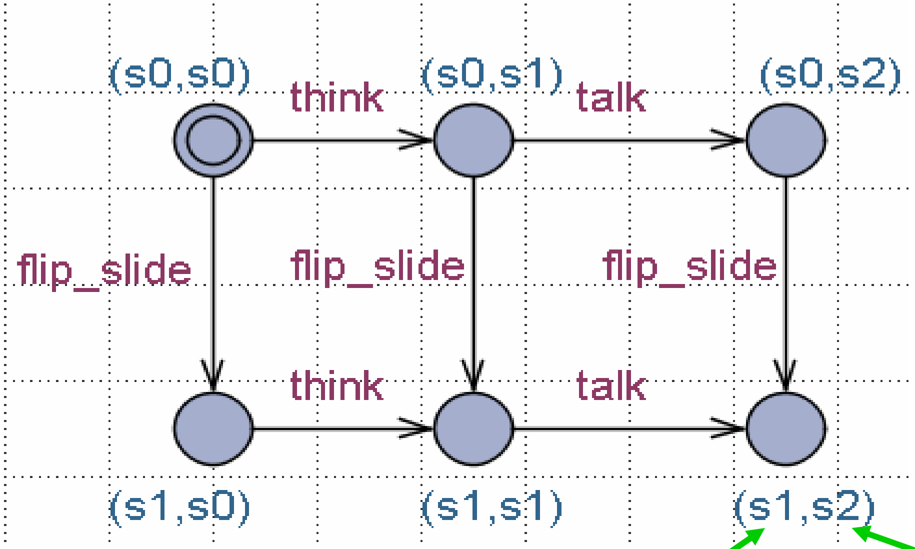
2 states

Speaker



3 states

Lecturer =
Speaker || Flipper



2*3 states

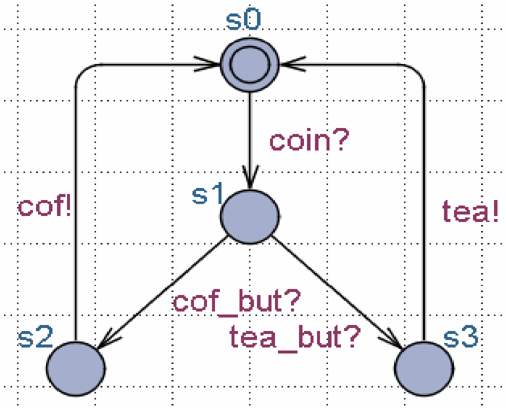
from Flipper

from Speaker

Process Interaction

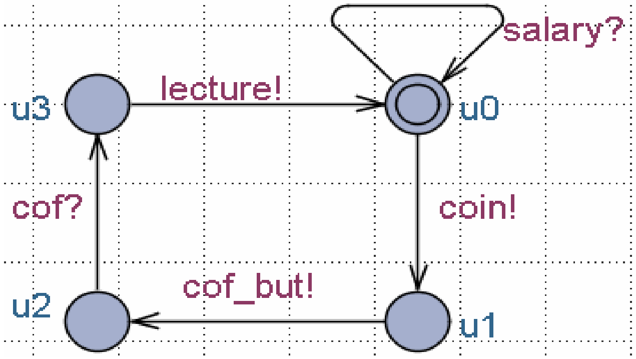
- ! = Output, ? = Input
- Handshake communication
- Two-way

Coffee Machine



4 states

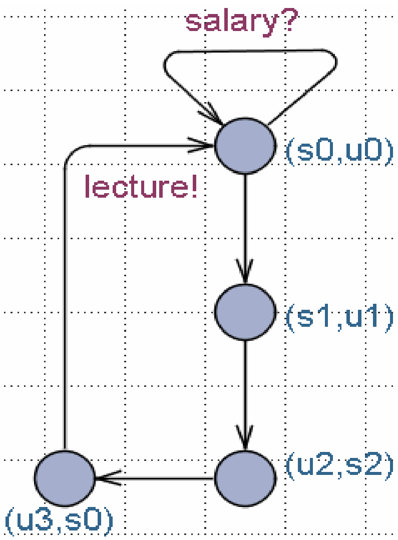
Lecturer



4 states

University=
Coffee Machine || Lecturer

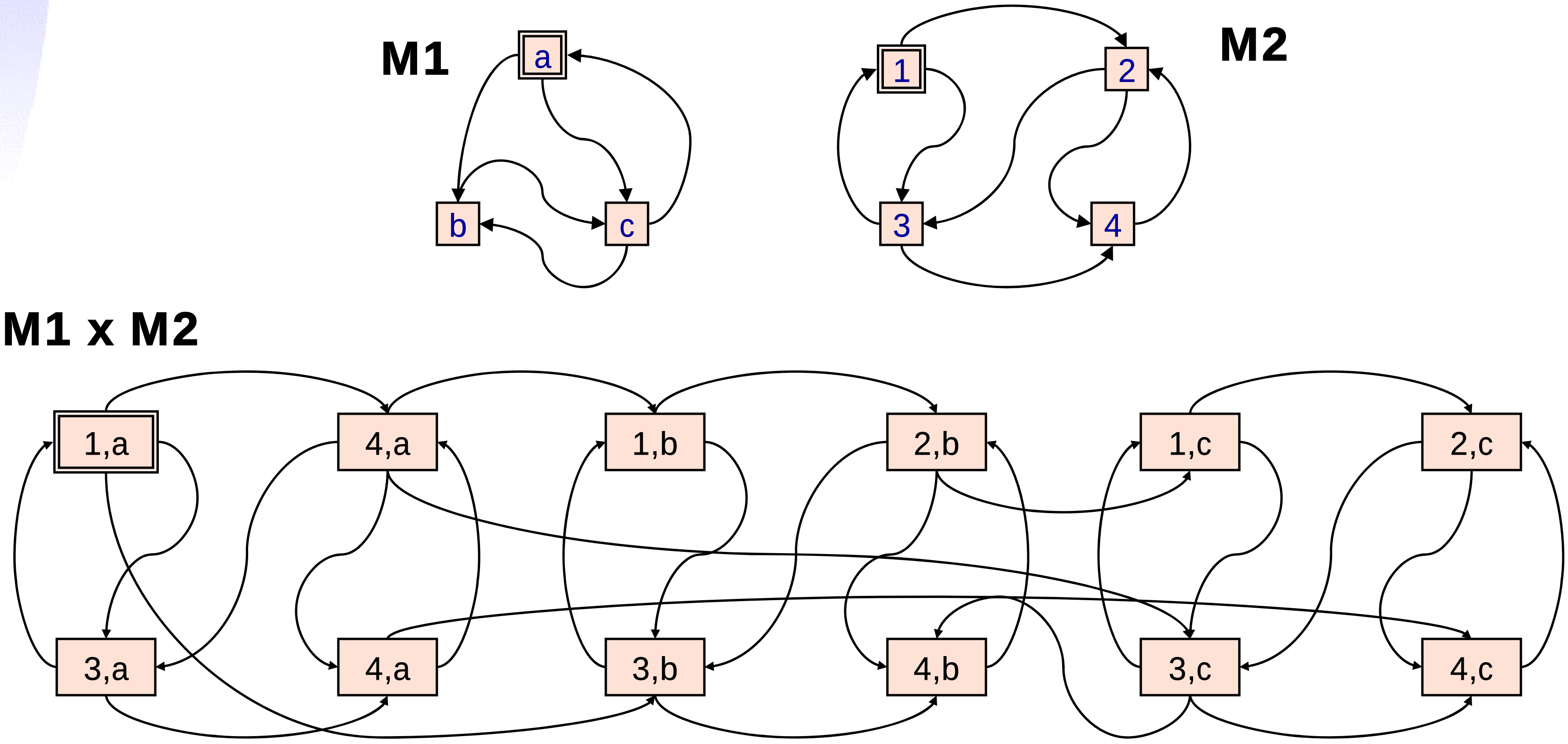
- LTS?
- How many states?
- Traces ?



synchronization results in internal actions

4 states:Interaction constrain overall behavior

Composition



All combinations=
exponential in no of machines

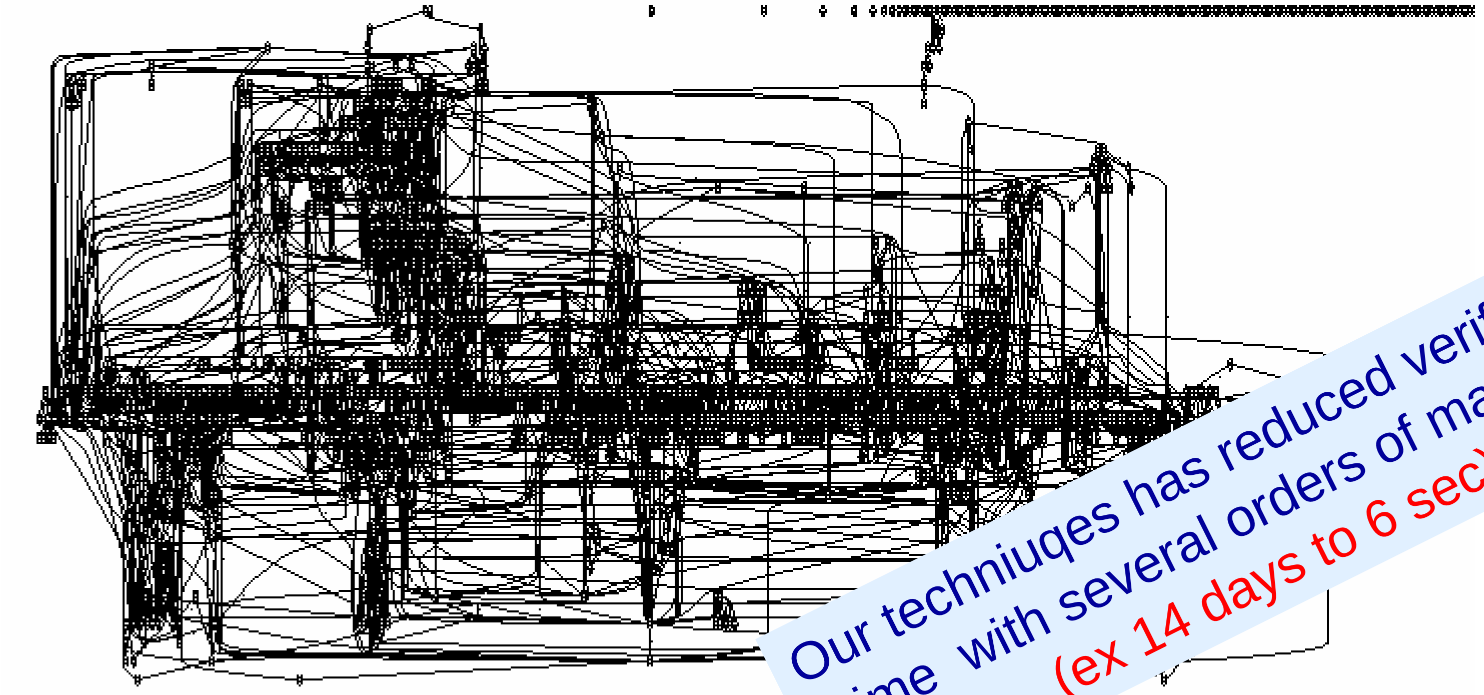
Train Simulator

BRICS
Basic Research
in Computer Science

VVS
visualSTATE

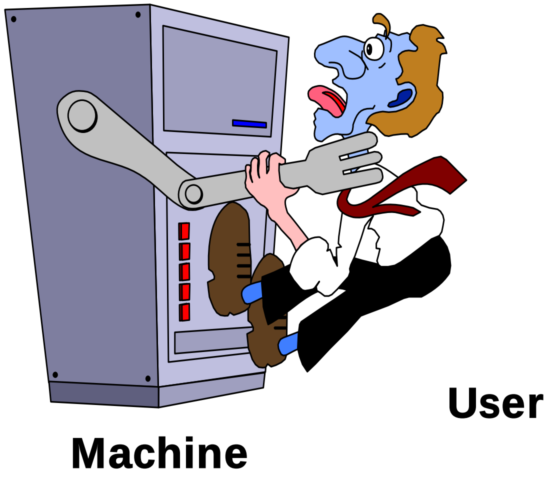
1421 machines
11102 transitions
2981 inputs
2667 outputs
3204 local states
Declare state sp.: 10^476

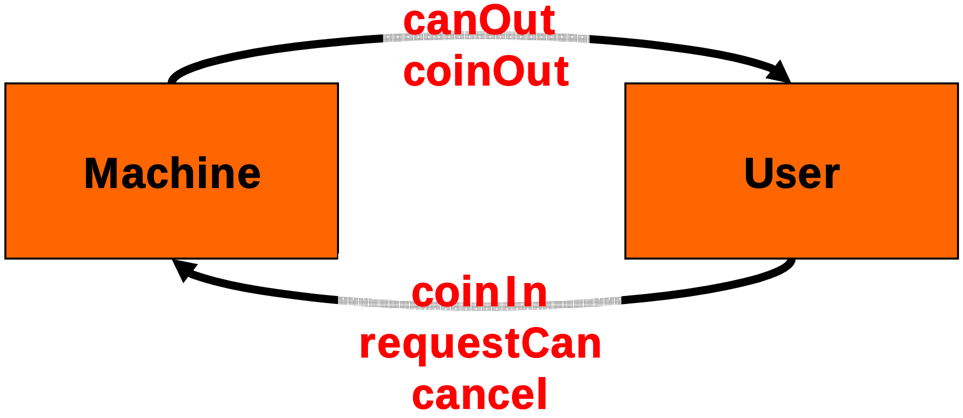
BUGS ?



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

Modelling Exercise
The Vending Machine





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

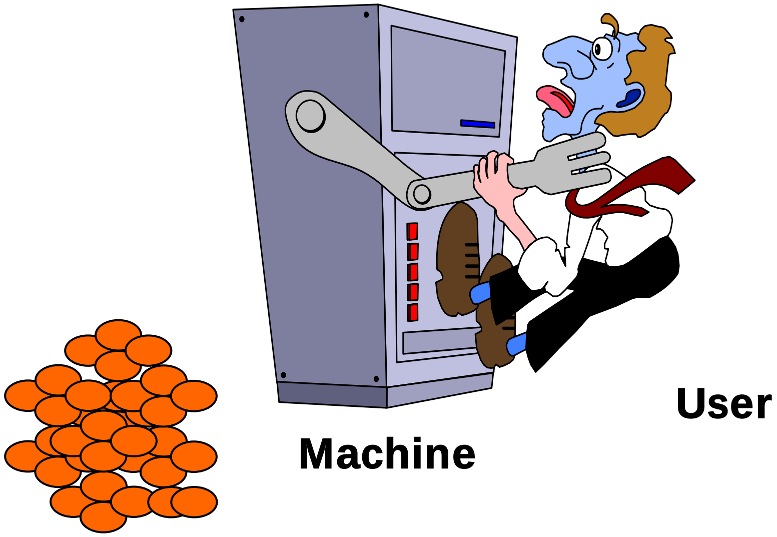
- Cans requested will be delivered ?
- Cancellations are obeyed ?

- What happens if multiple users?

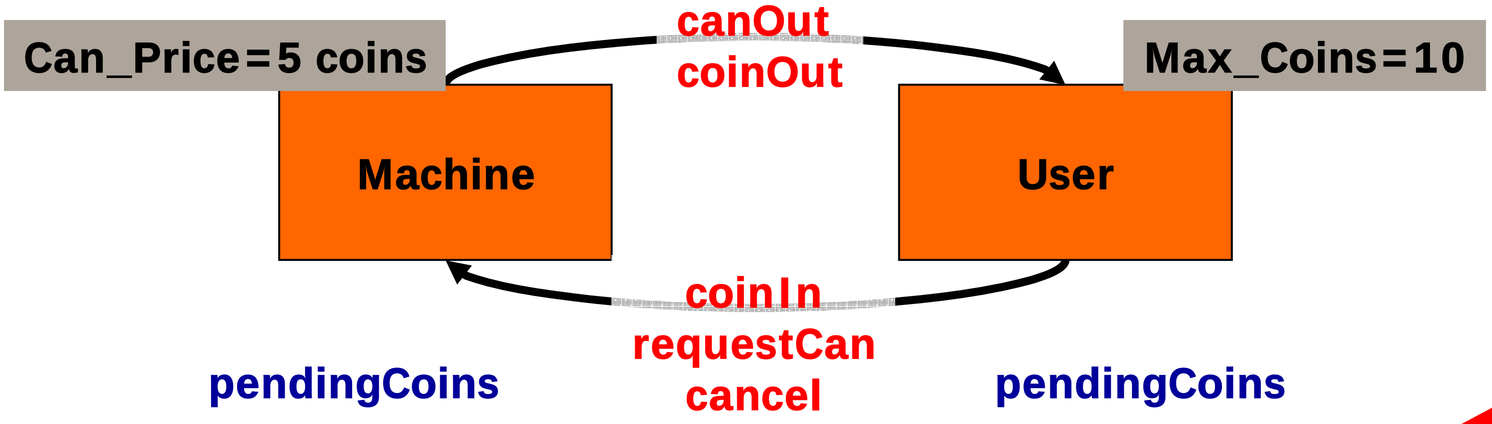
Assumption: 1 can = 1 coin!

Exercise

Modelling Exercise
The Vending Machine



- Extend model of Machine and FairUser
- Do extensive simulation



Can_Price=5 coins

Max_Coins=10

Machine

User

canOut
coinOut

coinIn
requestCan
cancel

pendingCoins

pendingCoins

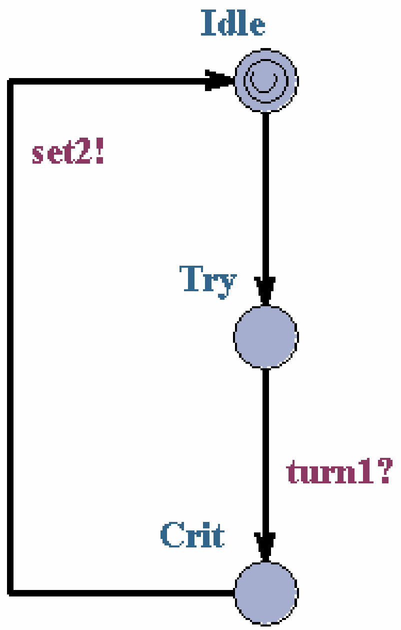
Exercise

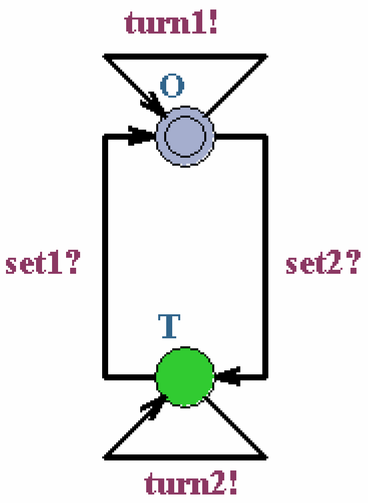
Verification = Model Checking

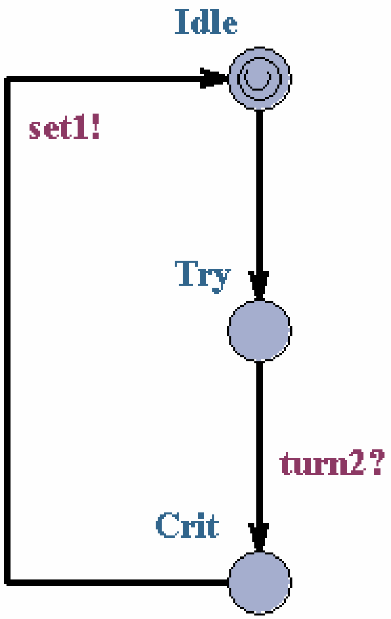
- **Reachability**
- **Generic properties**

Mutual Exclusion

Token

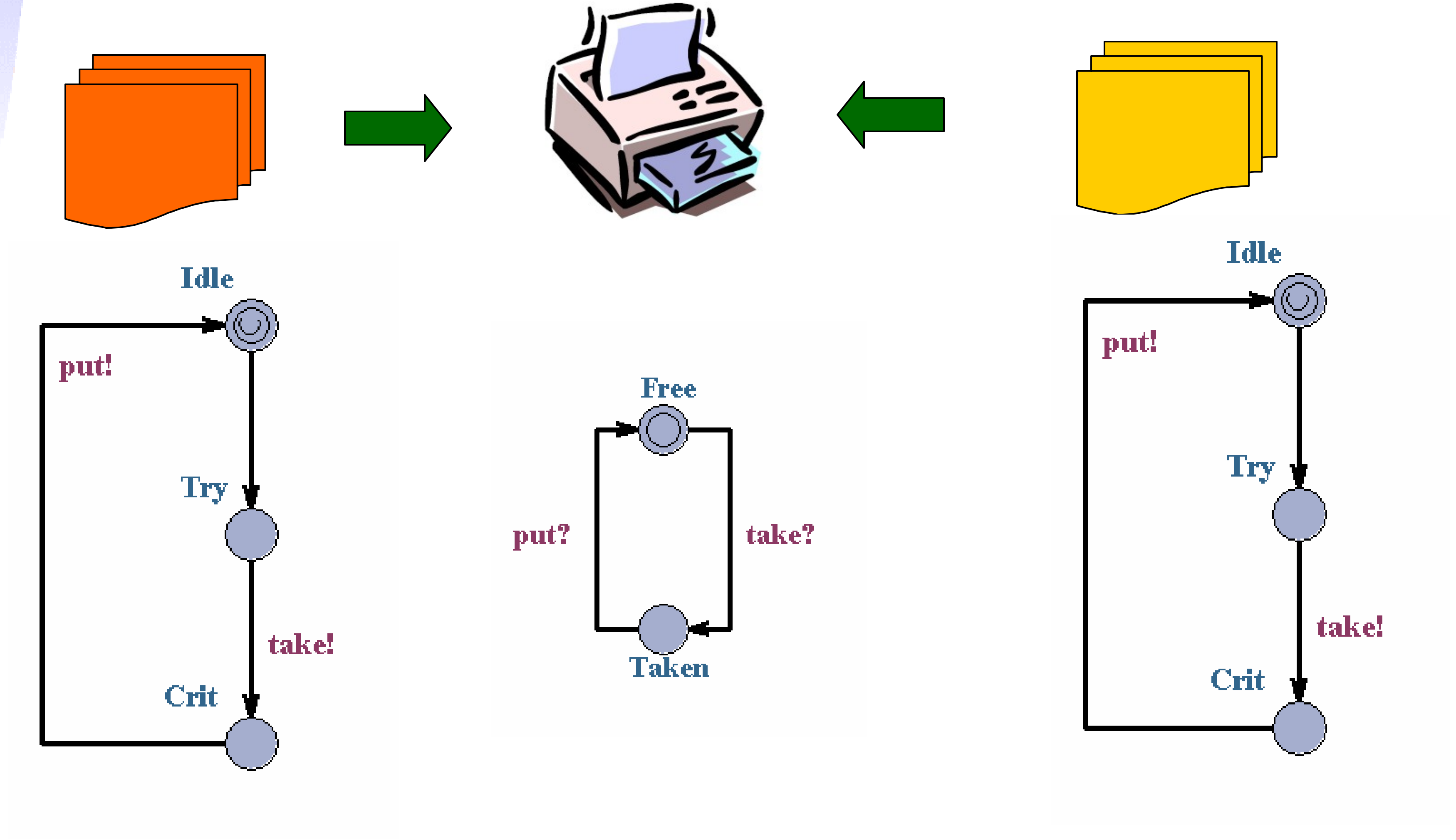






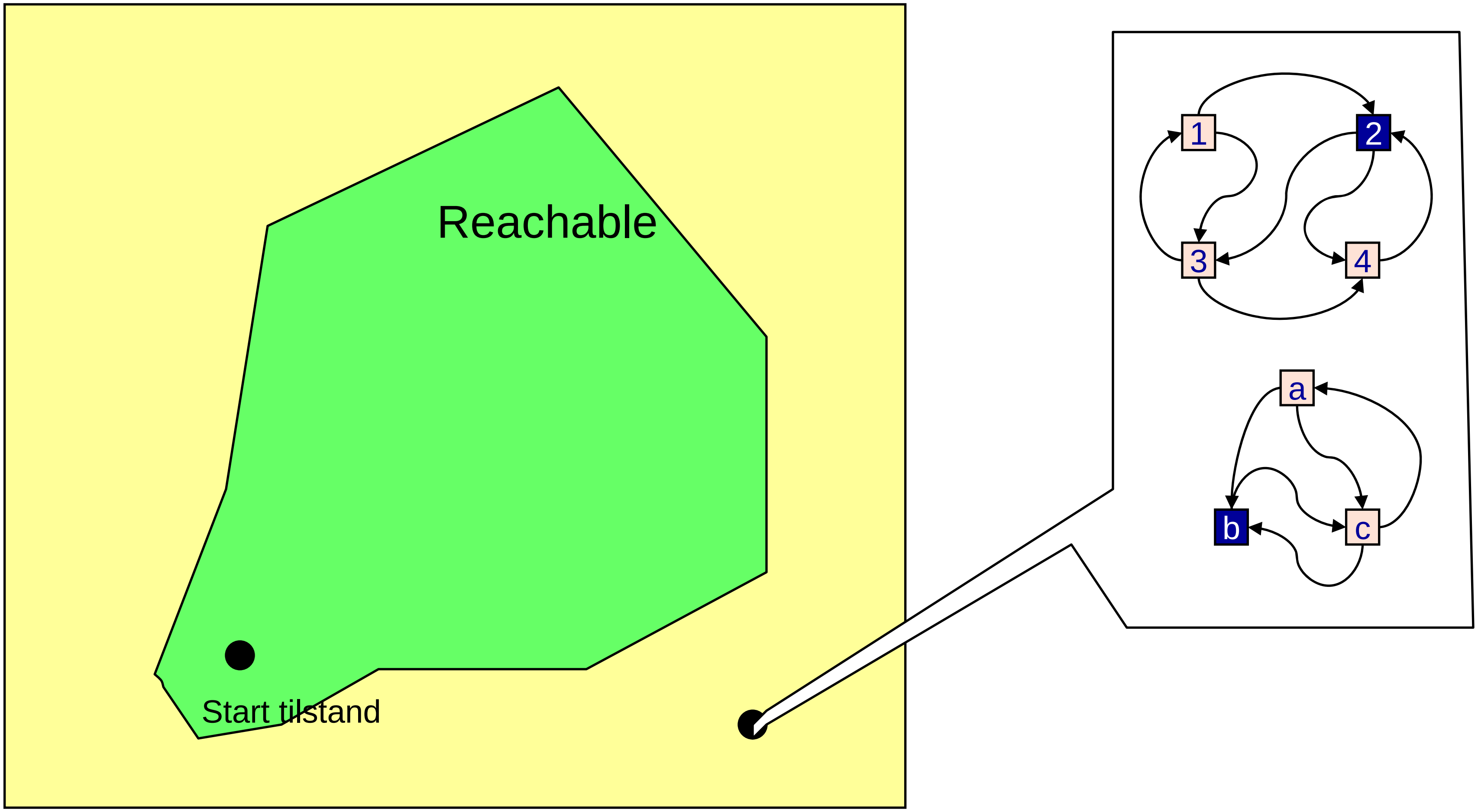
Mutual Exclusion

Semafor



Udforskning af Tilstandsrum

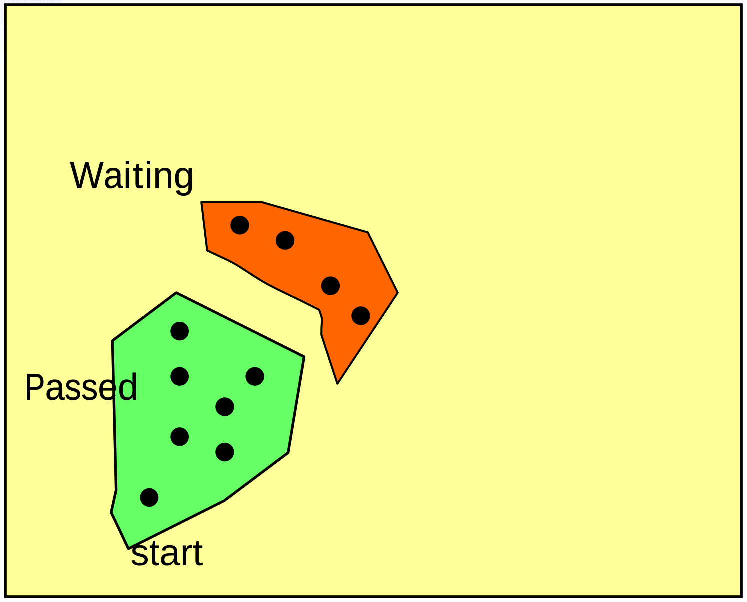
Erklæret tilstandsrum



Udforskning af tilstandrum

Forward Reachability Analysis

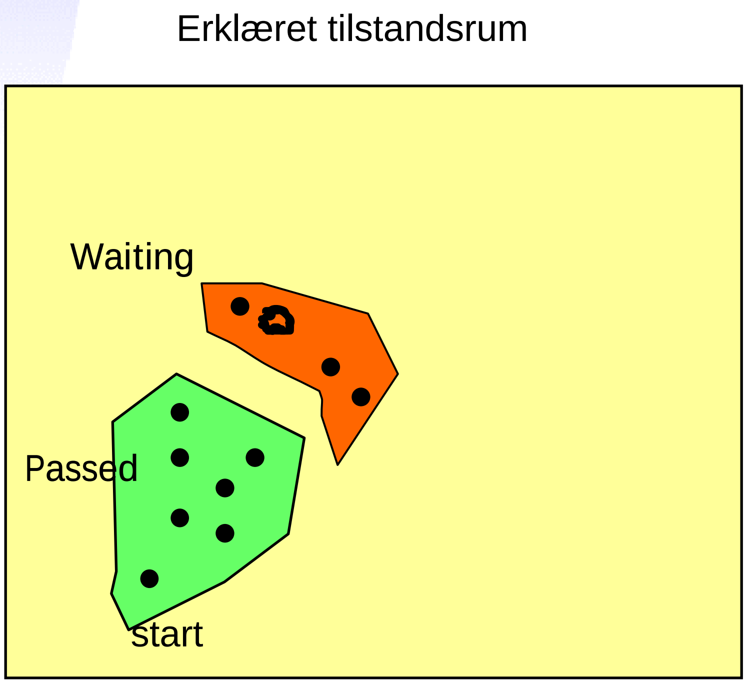
Erklæret tilstandsrum



```
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



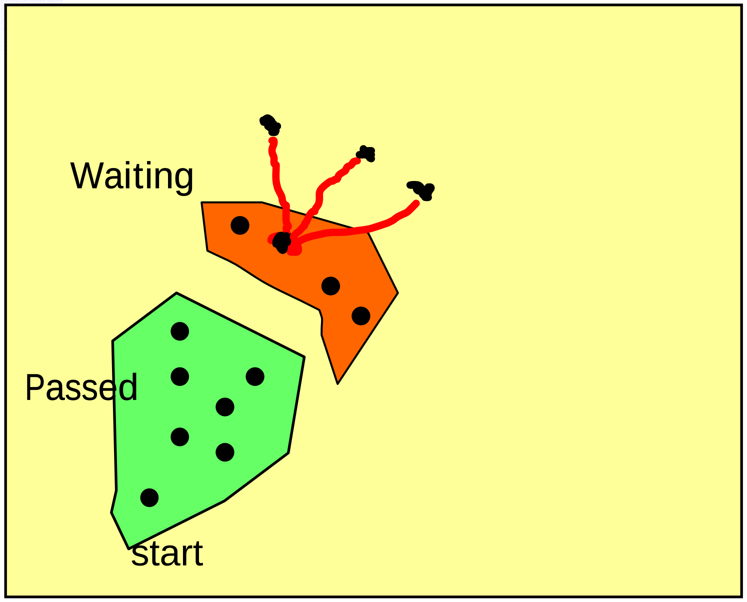
```
Passed:=Ø
Waiting:={$s_0$}
While(Waiting!=Ø)
  {
  select s∈ Waiting
  Waiting:=Waiting\{s}
  if s∉Passed
     whenever (s → t) then
        Waiting:=Waiting ∪ {t}
     Passed:=Passed ∪ {s}
  }
```

BRICS Basic Research in Computer Science

Udforskning af tilstandrum

Forward Reachability Analysis

Erklæret tilstandsrum



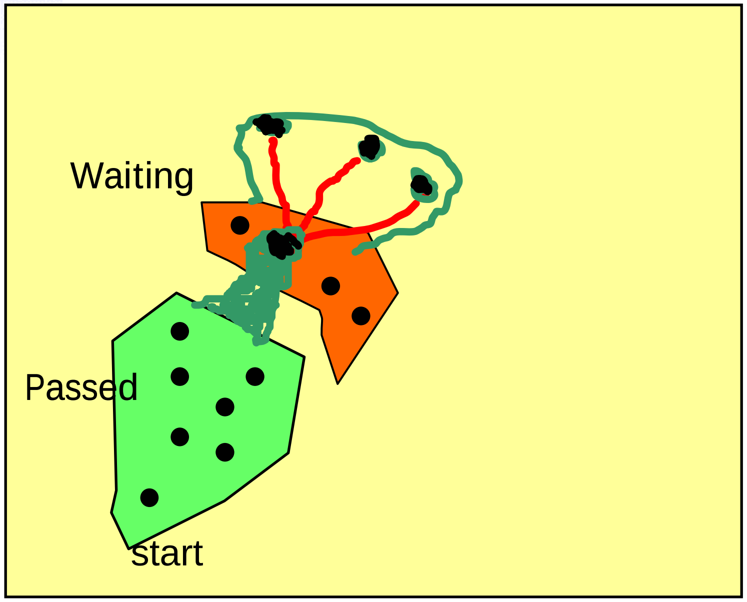
```
Passed:=Ø
Waiting:={s0}
While(Waiting!=Ø)
  {
  select s∈ Waiting
  Waiting:=Waiting\{s}
  if s∉Passed
     whenever (s → t) then
        Waiting:=Waiting ∪ {t}
     Passed:=Passed ∪ {s}
  }
```

CISS CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Udforskning af tilstandrum

Forward Reachability Analysis

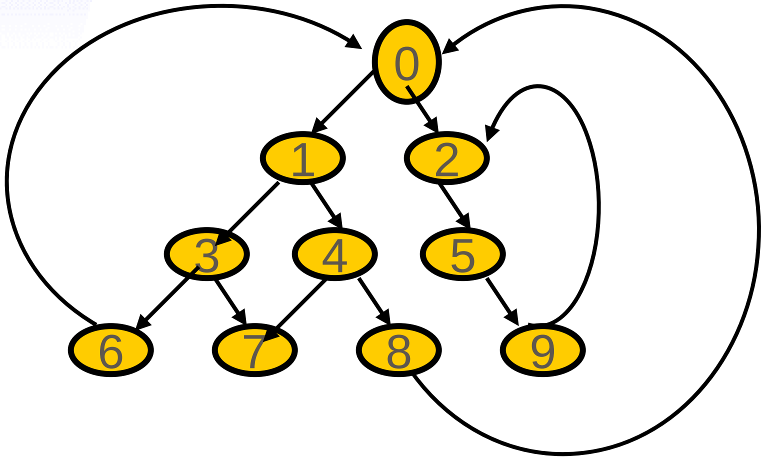
Erklæret tilstandsrum



```
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}
  }
```

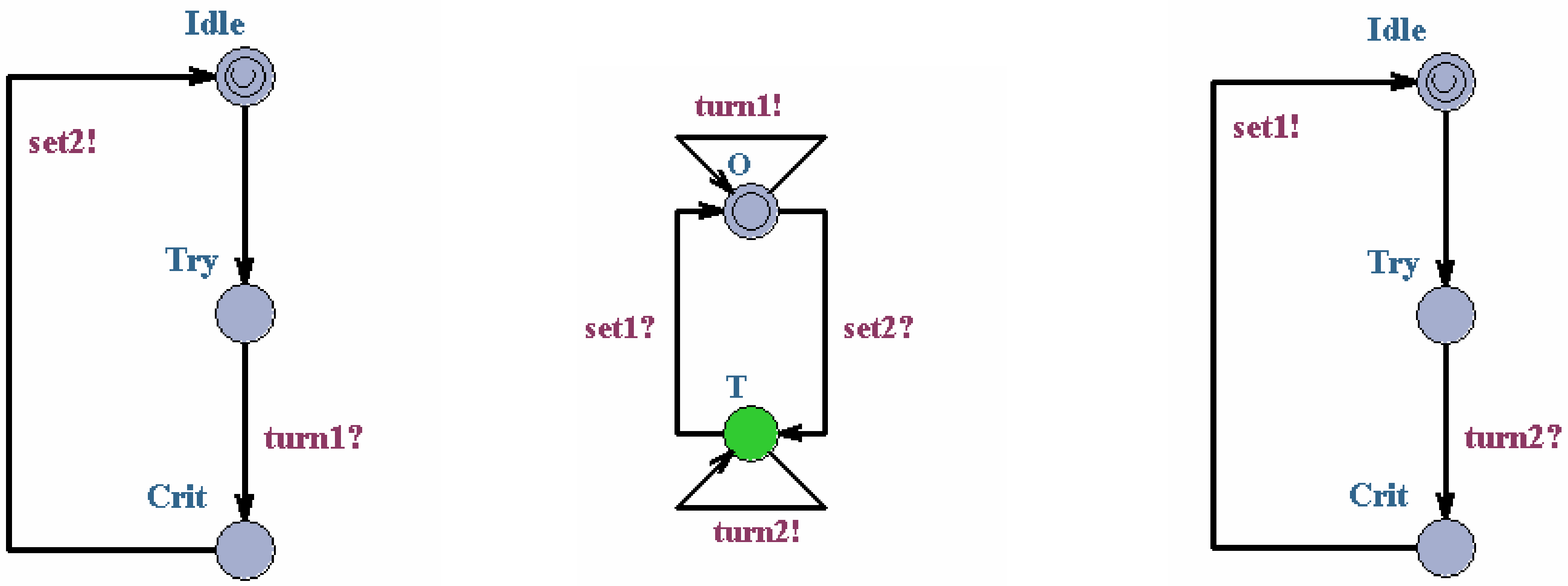
Depth-first search: organize Waiting as a **Stack**

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

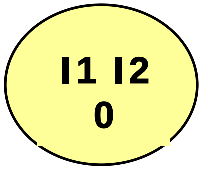
Gensidig Udelukkelse

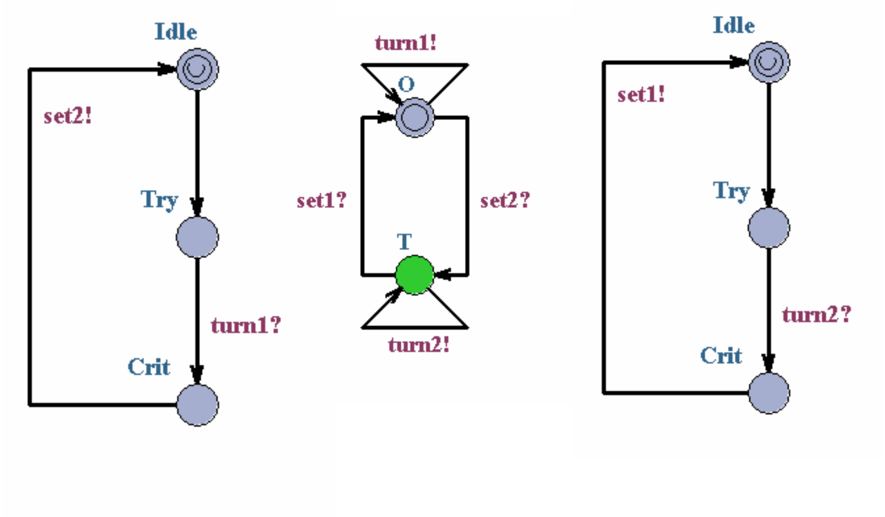


Token

Gensidig udelukkelse

Forward Reachability

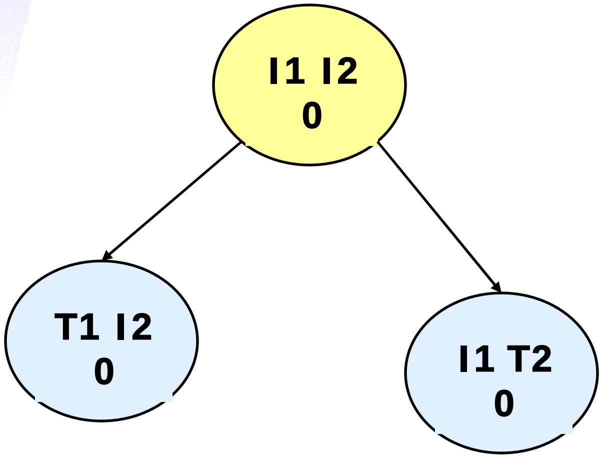


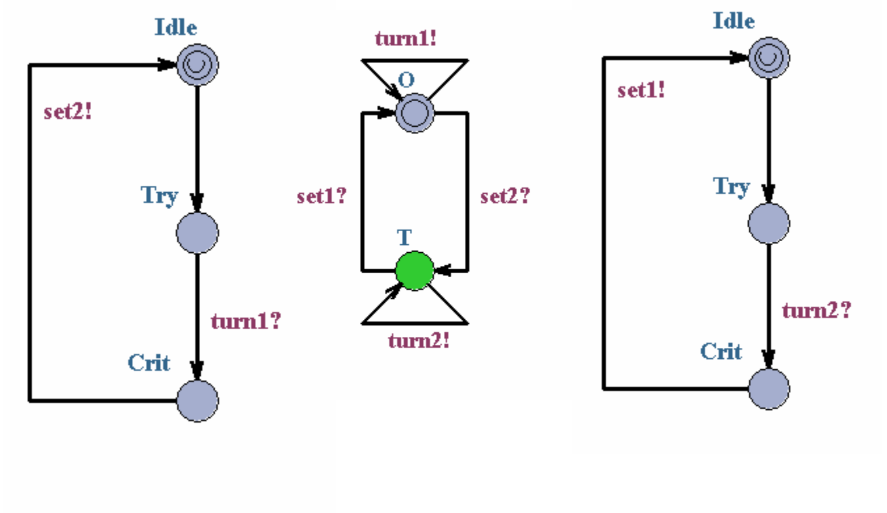


Token

Gensidig udelukkelse

Forward Reachability

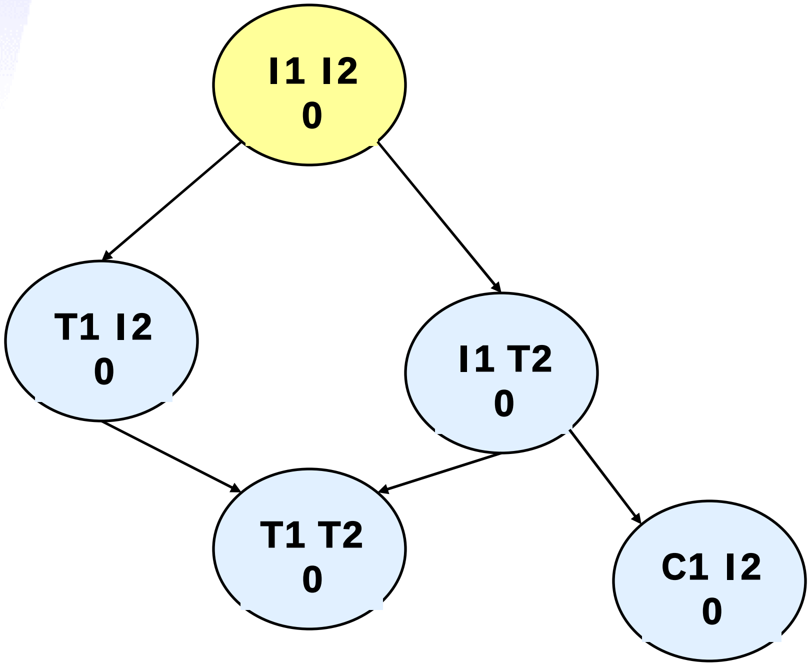


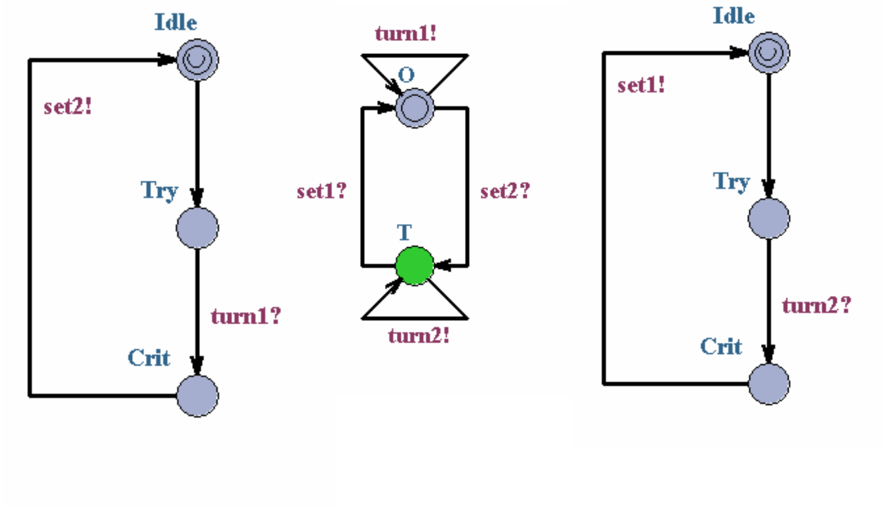


Token

Gensidig udelukkelse

Forward Reachability

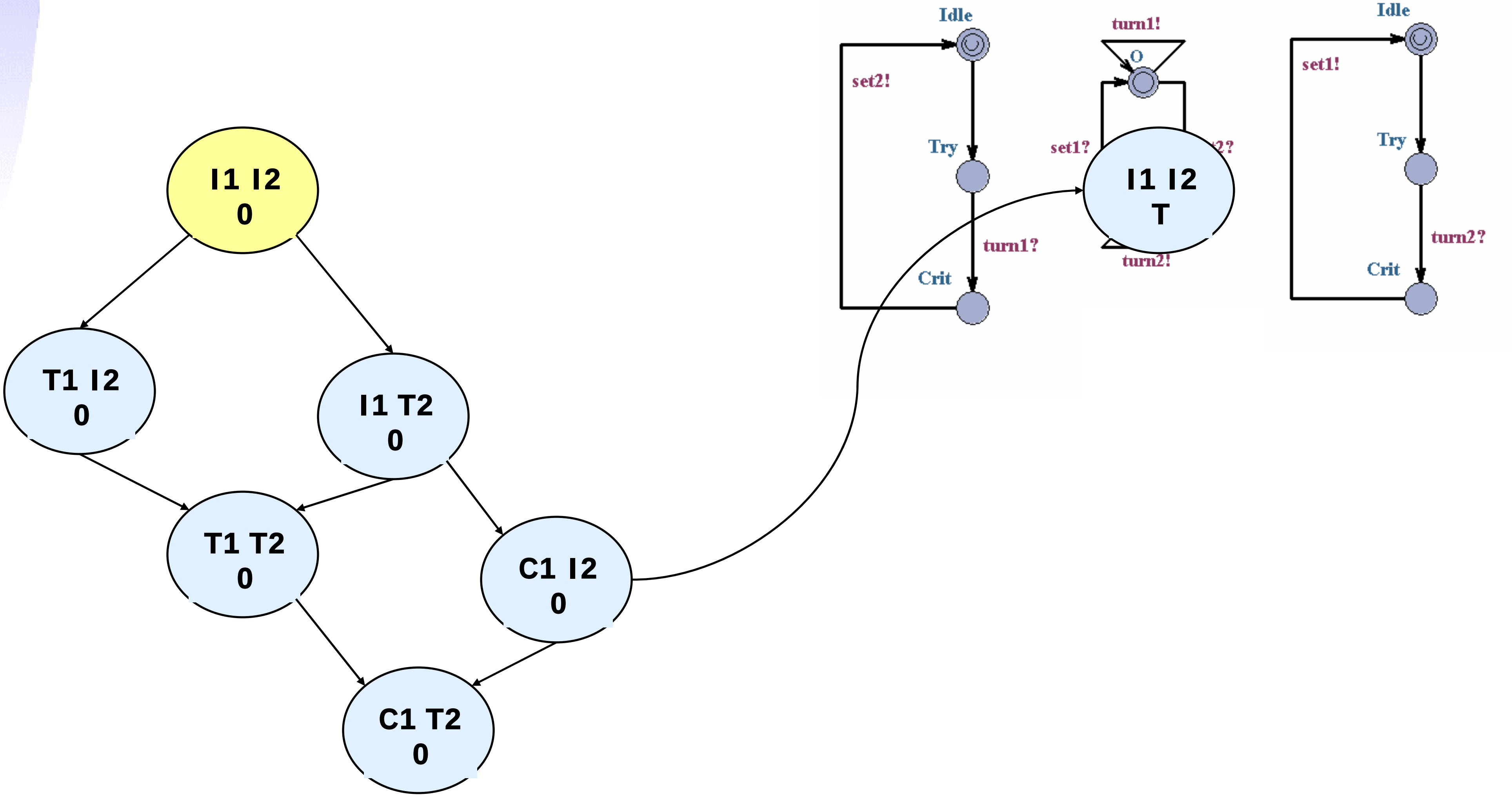




Token

Gensidig udelukkelse

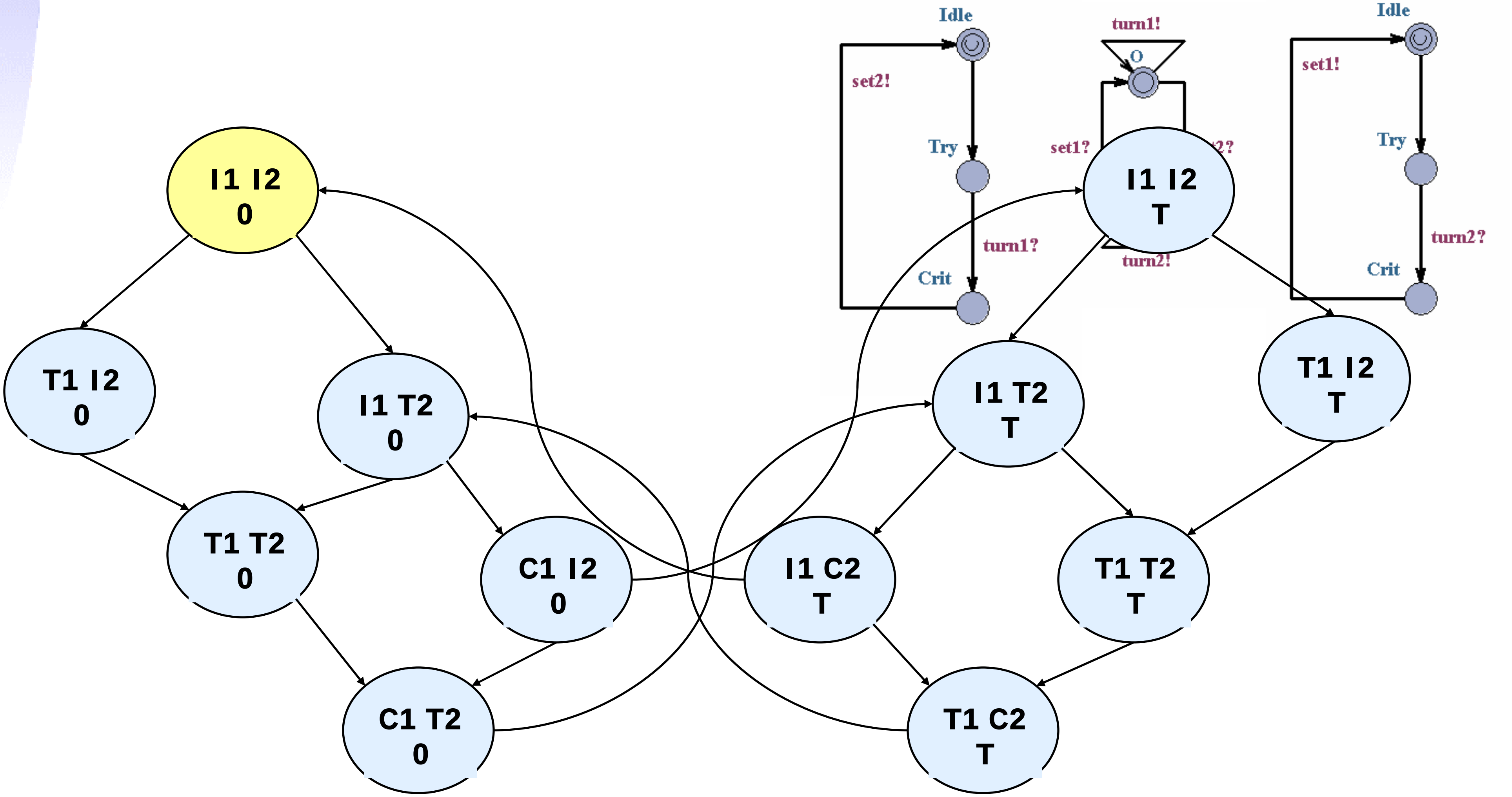
Forward Reachability



Token

Gensidig udelukkelse

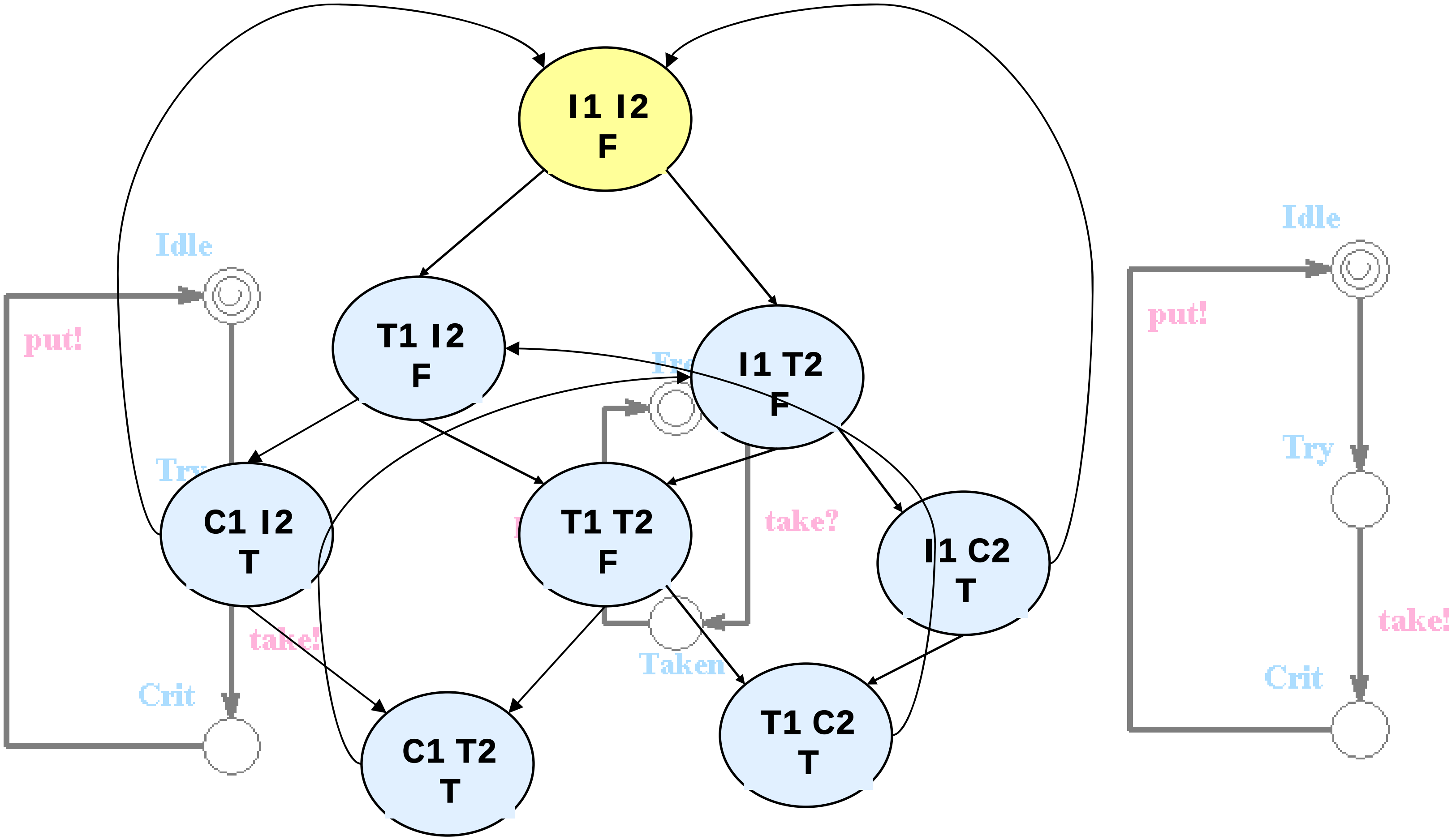
Forward Reachability



Token

Gensidig udelukkelse

Forward Reachability



Semafor

Generiske egenskaber

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

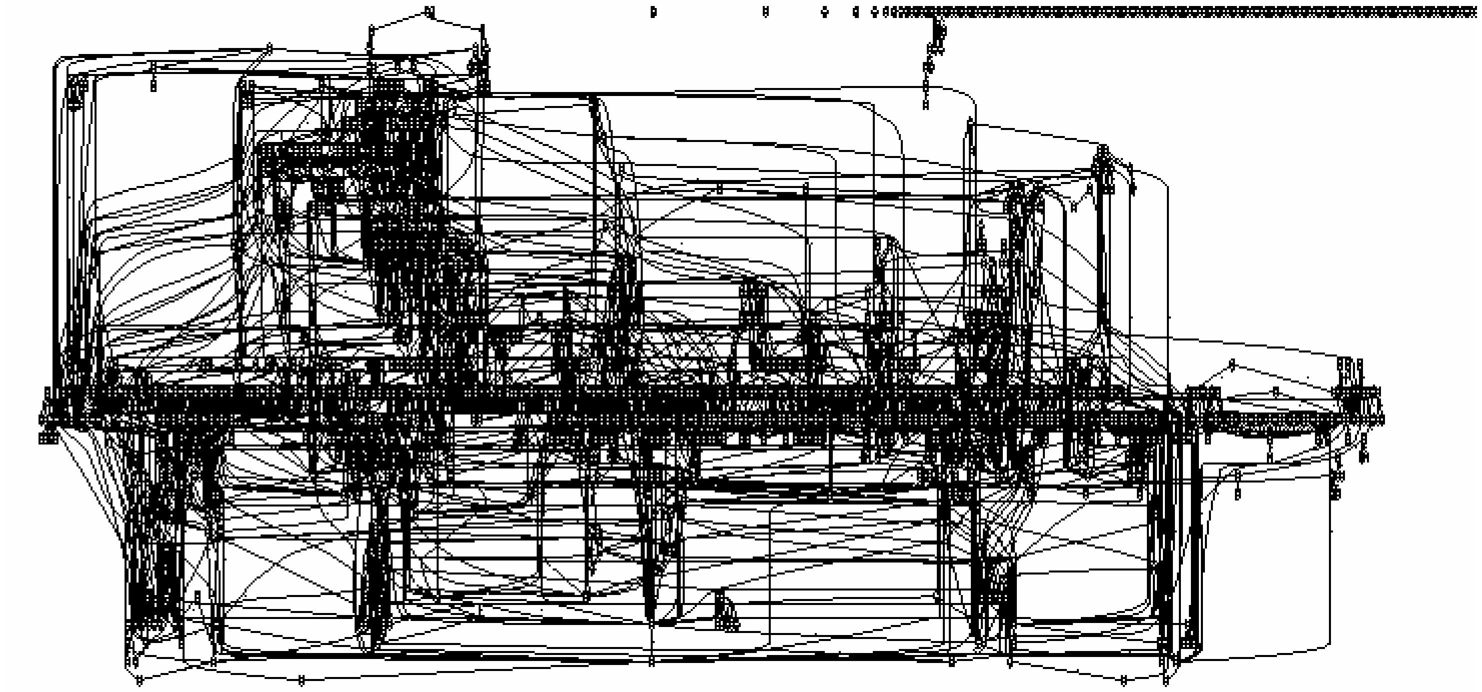
Kan alle reduceres til REACHABILITY

Train Simulator

VVS
visualSTATE

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

BUGS ?



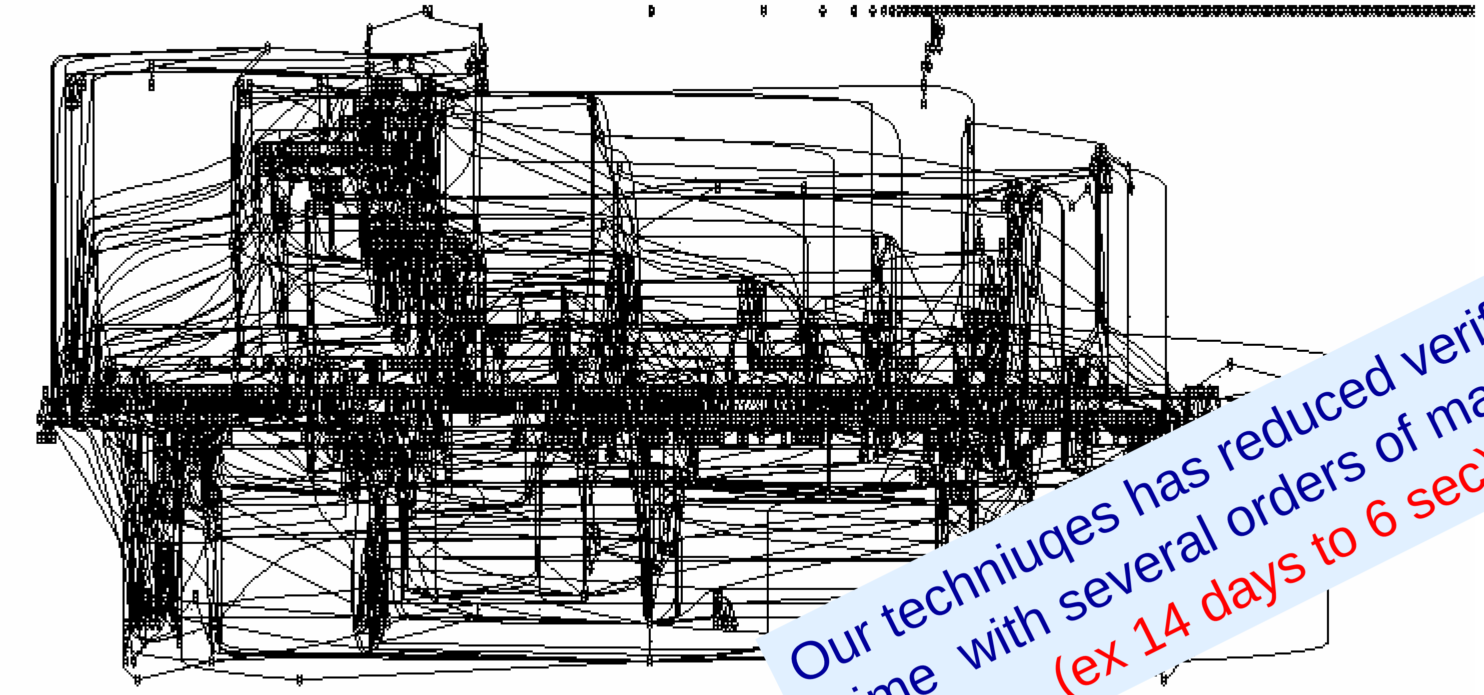
BRICS
Basic Research
in Computer Science

VVS
visualSTATE

Train Simulator

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)

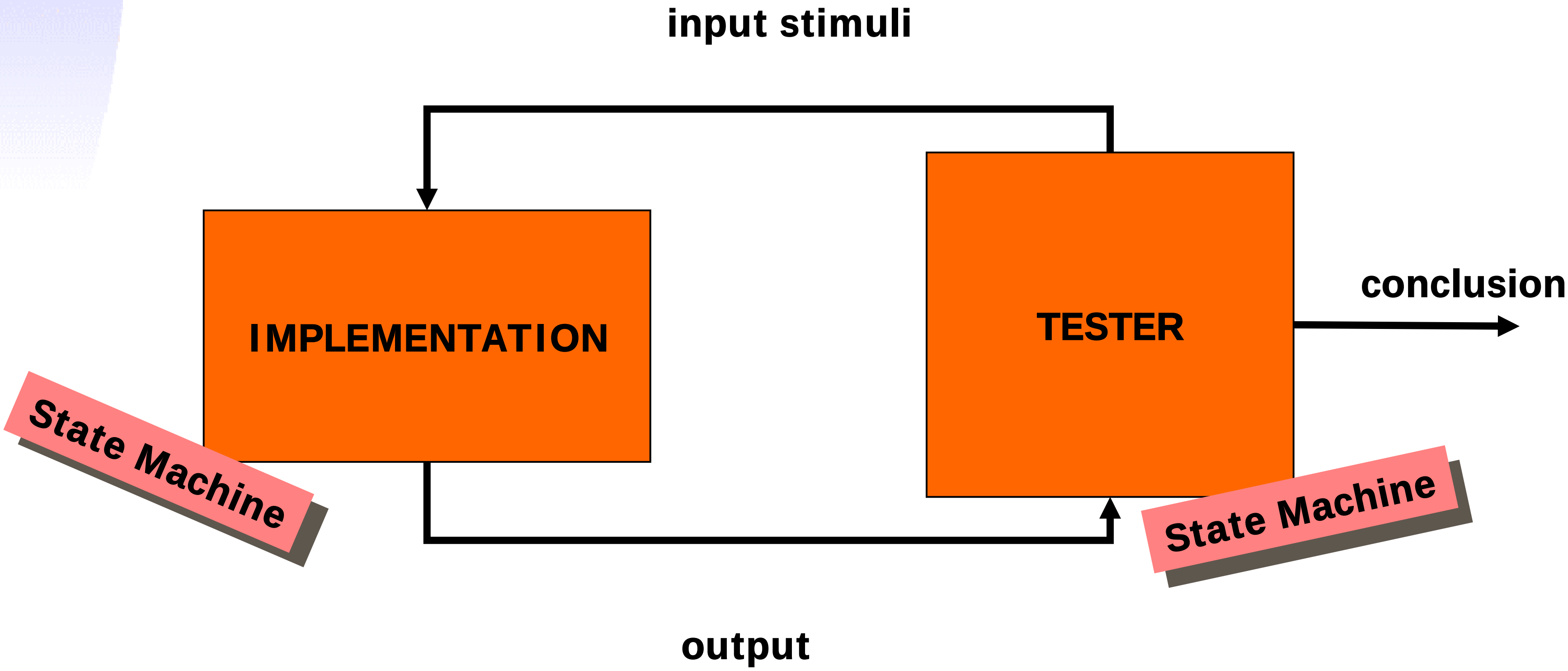
CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Modelbased Testing

Motivation

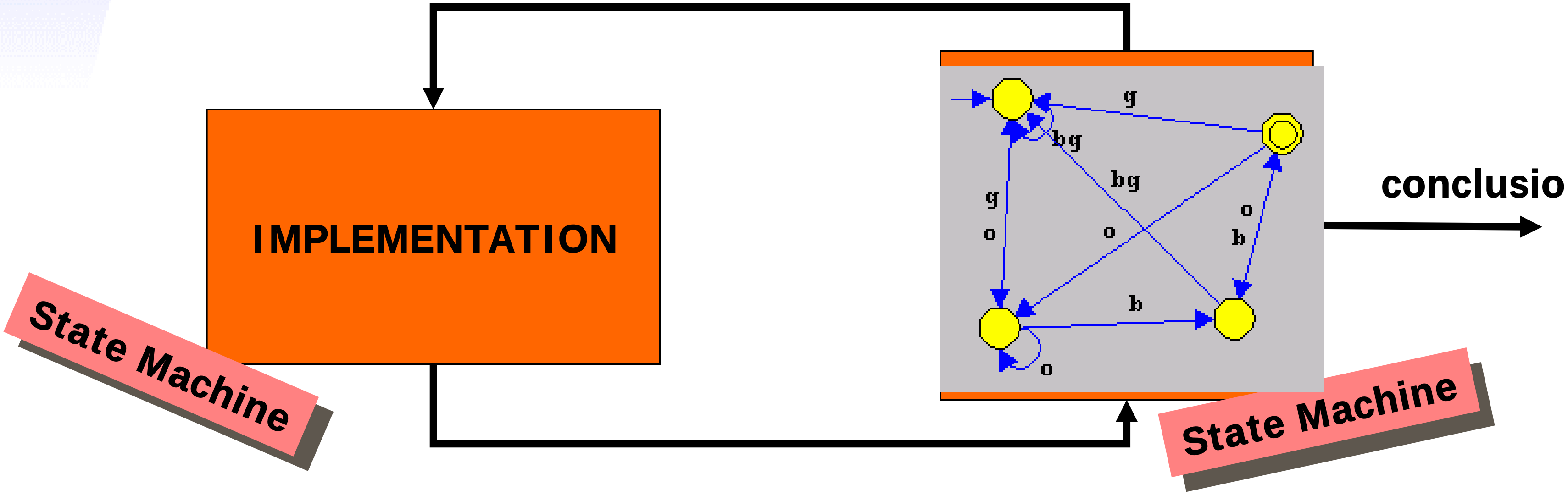
- Testing = sample executions of system compared with requirements
- Testing may identify errors but can not be used to exclude their presence.
- Testing is the de-facto used method of validation
- 30-40% of the entire development process is concerned with testing.

Black Box Testing



Black Box Testing

input stimuli



IMPLEMENTATION

State Machine

State Machine

conclusion

output

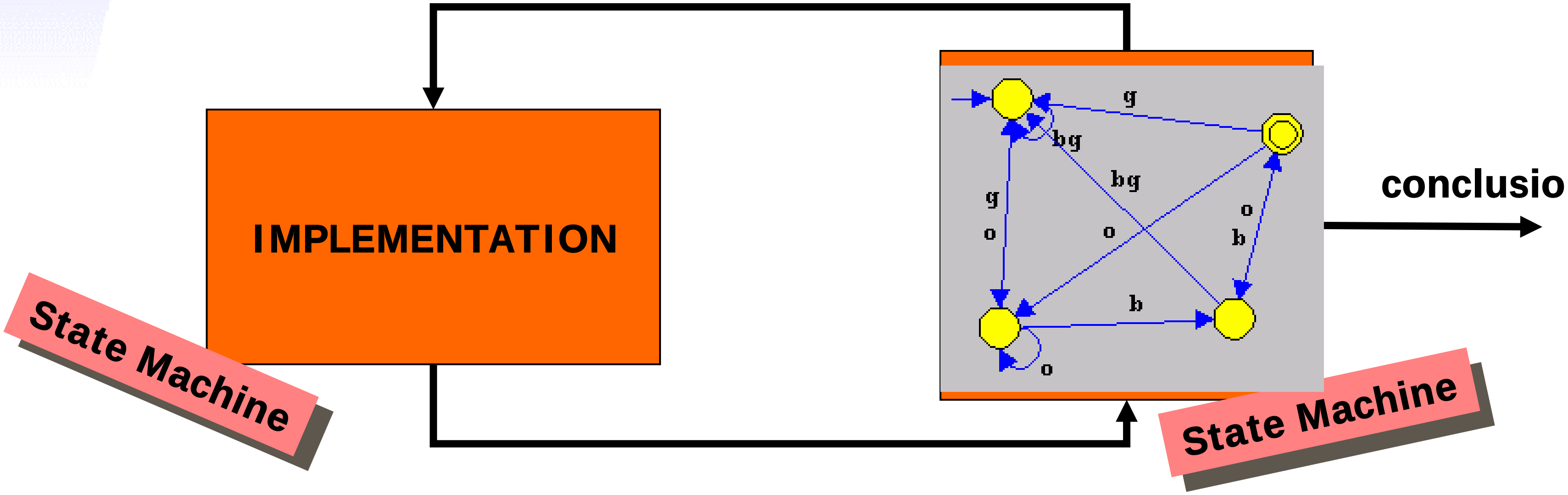
closed/open

TEST	EXPECTED OUTPUT
gogoobb	closed
gooobo	open
ggggggggg	closed
ooooggobo	open
…….	….

Black Box Testing

MOORE's Theorem:
Hvis IMP antages at have *m* tilstande og SPEC har *n* tilstande da er det nok at teste mht alle sekvenser af lgd *n+m-1*

input stimuli



IMPLEMENTATION

State Machine

conclusion

State Machine

output

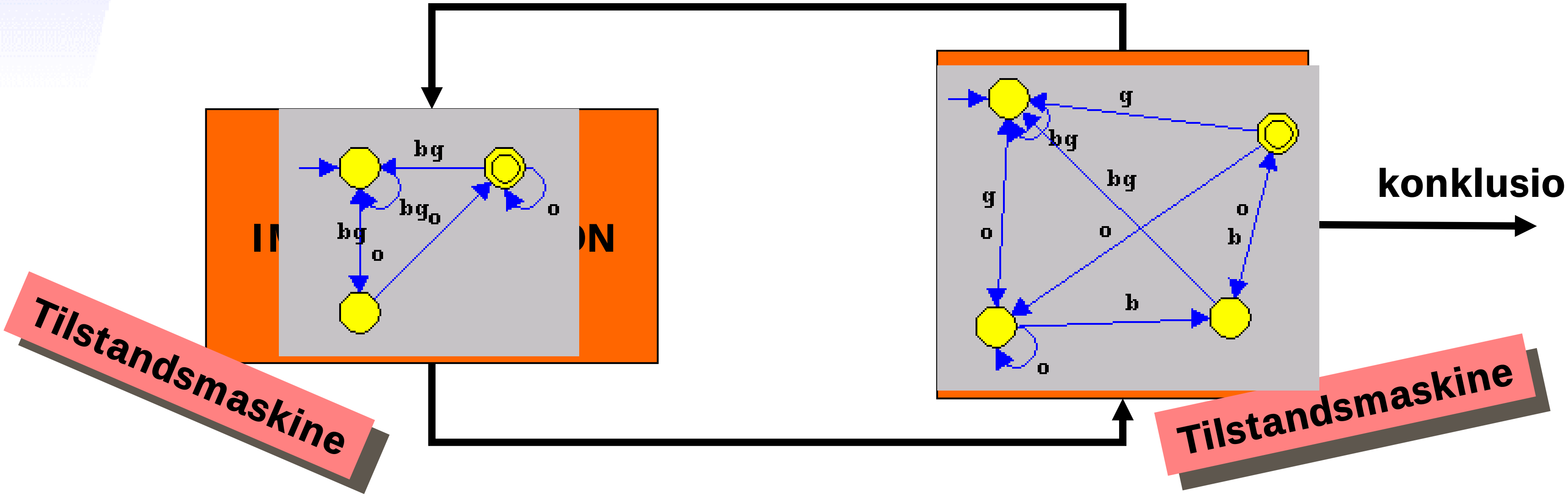
closed/open

TEST	EXPECTED OUTPUT
gogoobb	closed
gooobo	open
ggggggggg	closed
ooooggobo	open
…….	….

Black Box Testing

MOORE's Theorem:
Hvis IMP antages at have *m* tilstande og SPEC har *n* tilstande da er det nok at teste mht alle sekvenser af lgd *n+m-1*

input stimuli



Tilstandsmaskine

Tilstandsmaskine

konklusion

output

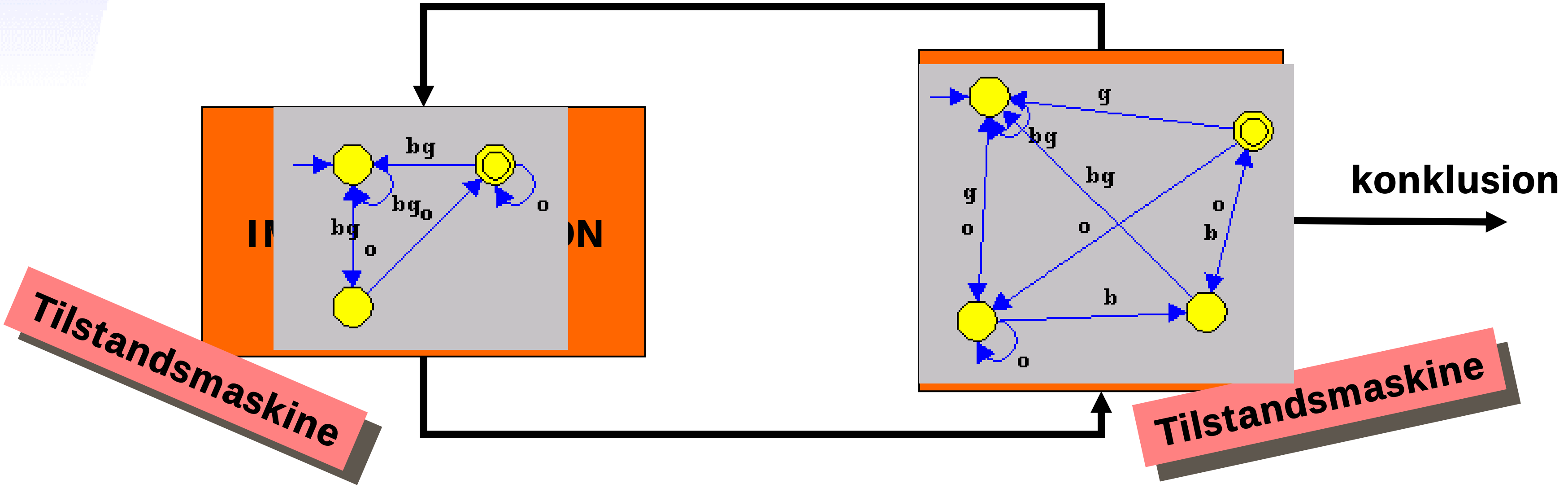
closed/open

TEST	EXPECTED OUTPUT
ggggobo	open (closed)
gggggoo	closed (open)
.....	...
.....	...
....	....

Black Box Testing

MOORE's Theorem:
Hvis IMP antages at have *m* tilstande og SPEC har *n* tilstande da er det nok at teste mht alle sekvenser af lgd *n+m-1*

input stimuli



Tilstandsmaskine

Tilstandsmaskine

konklusion

output

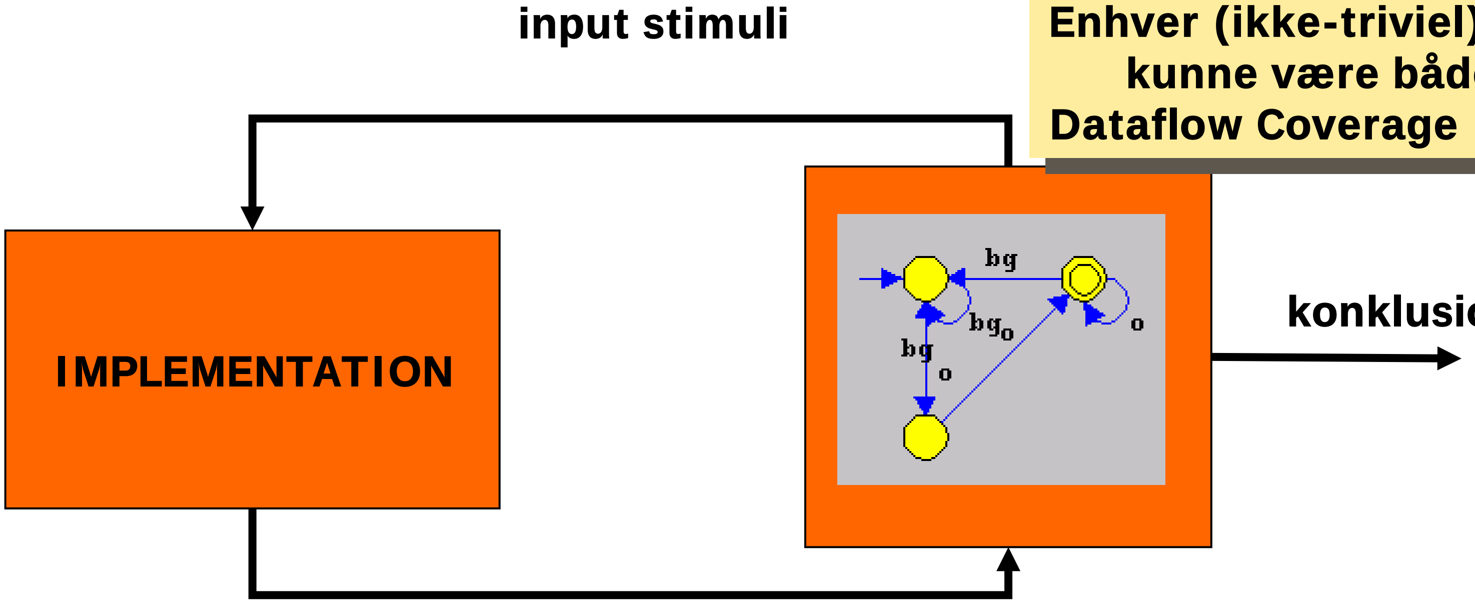
closed/open

Problem:
Antal test er ASTRONOMISK:
$k^{(n+m-1)}$
hvor *k* er antal symboler

TEST	EXPECTED OUTPUT
ggggobo	open (closed)
gggggoo	closed (open)
.....	...
.....	...
....	....

Black Box Testing

Control Flow Coverage
Enhver transition skal fyres
Enhver (lokal) tilstand skal nås
Enhver (ikke-triviel) guard skal kunne være både sand/falsk
Dataflow Coverage ...



input stimuli

IMPLEMENTATION

konklusion

output

closed/open

Problem:
Coverage kun af specifikation – implementation behøver kun at være dækket ganske lidt!

UPPAAL Tron

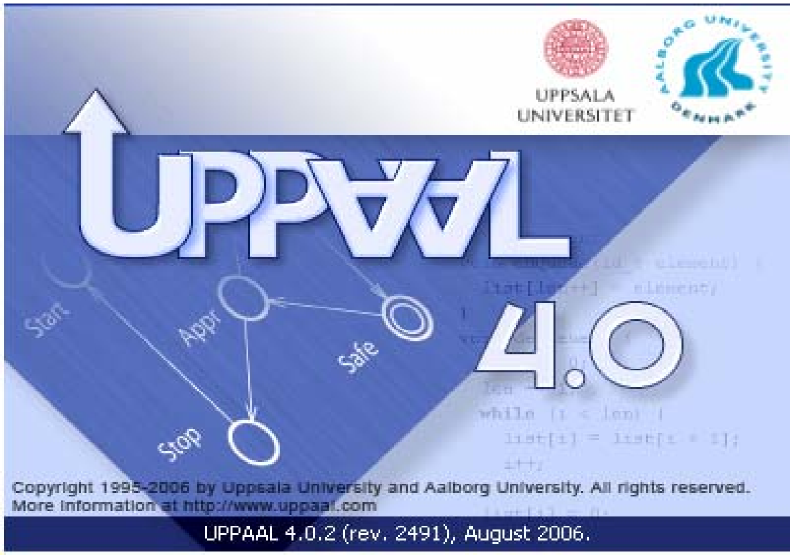
Løsning:
Brug specifikation automata til at (randomiseret) stimulering og løbende check konsistens af implementations reaktion

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.

Collaborators

@UPPsala

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

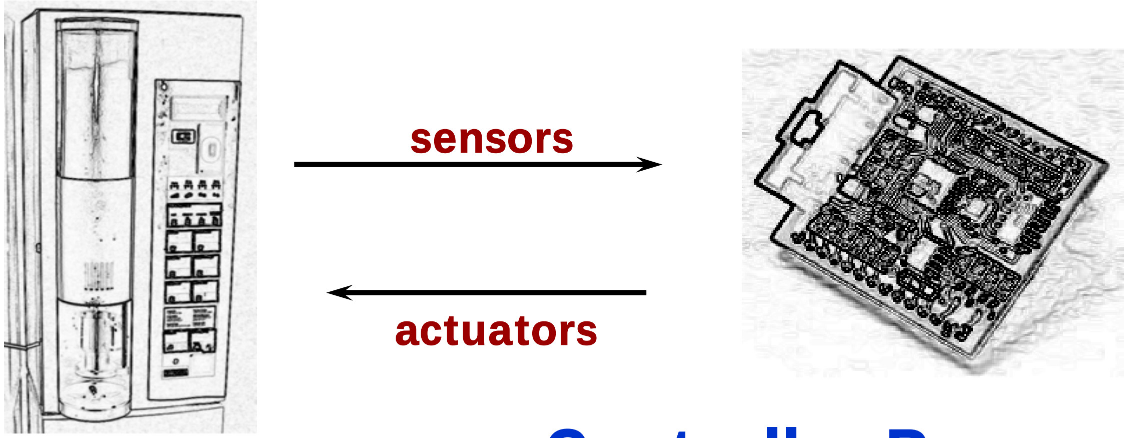
@AALborg

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

@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

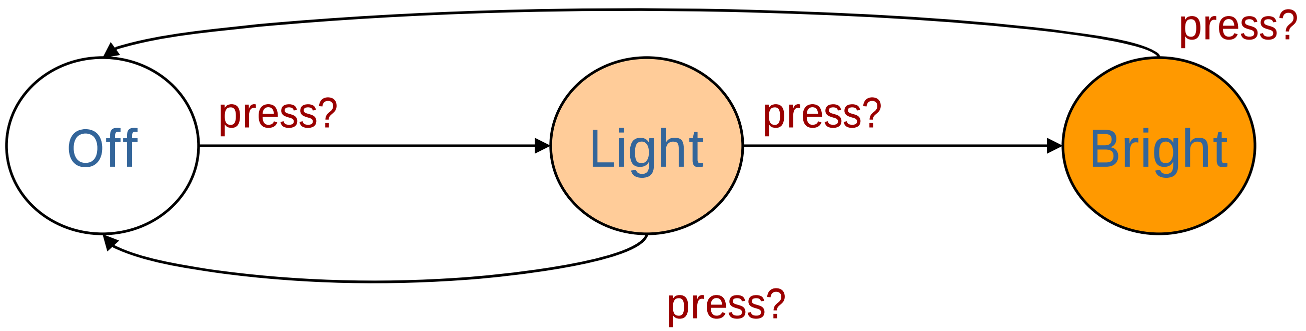
Discrete

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**!!

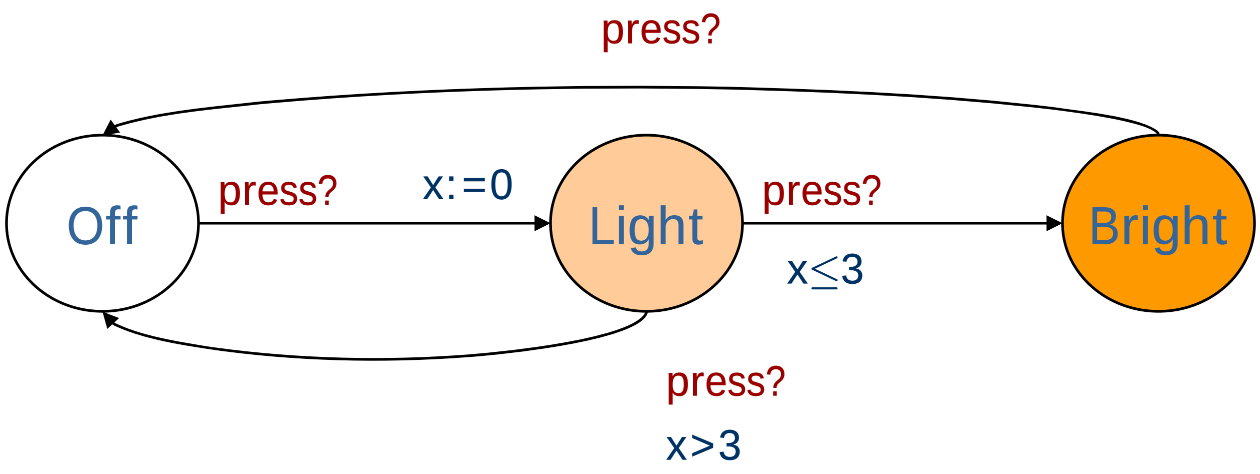
Dumb Light Control



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

Dumb Light Control

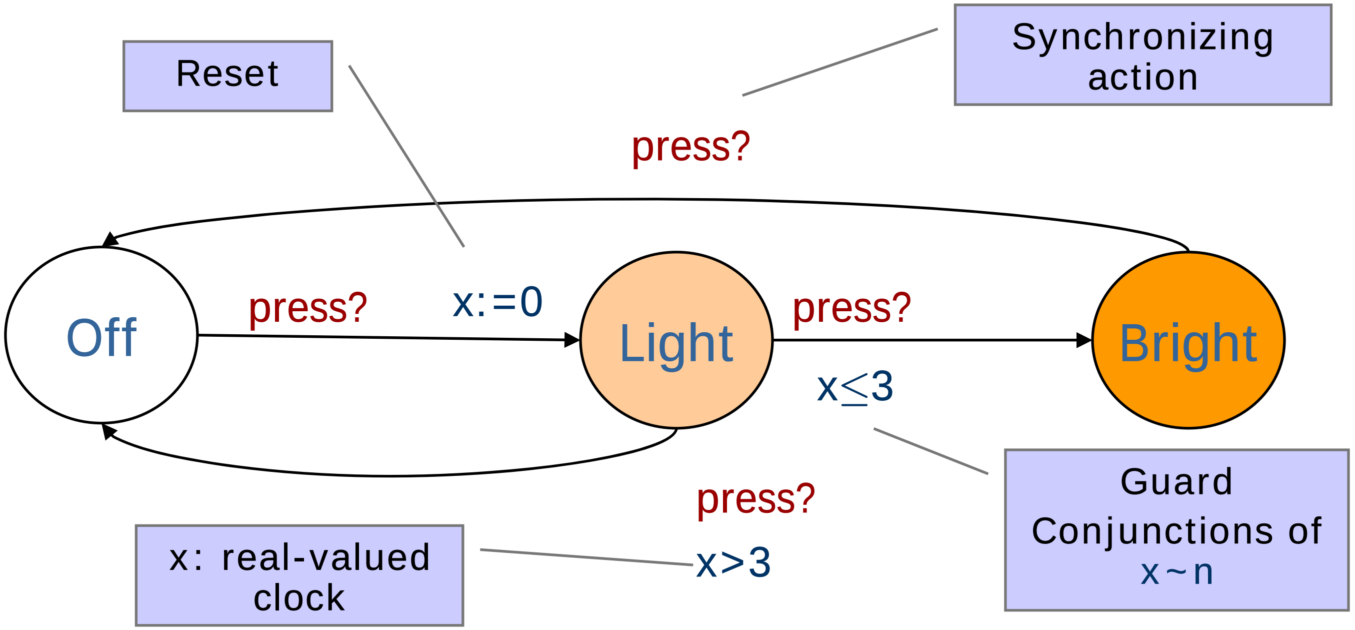
Alur & Dill 1990



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

Timed Automata

Alur & Dill 1990



Reset

Synchronizing action

Guard
Conjunctions of x~n

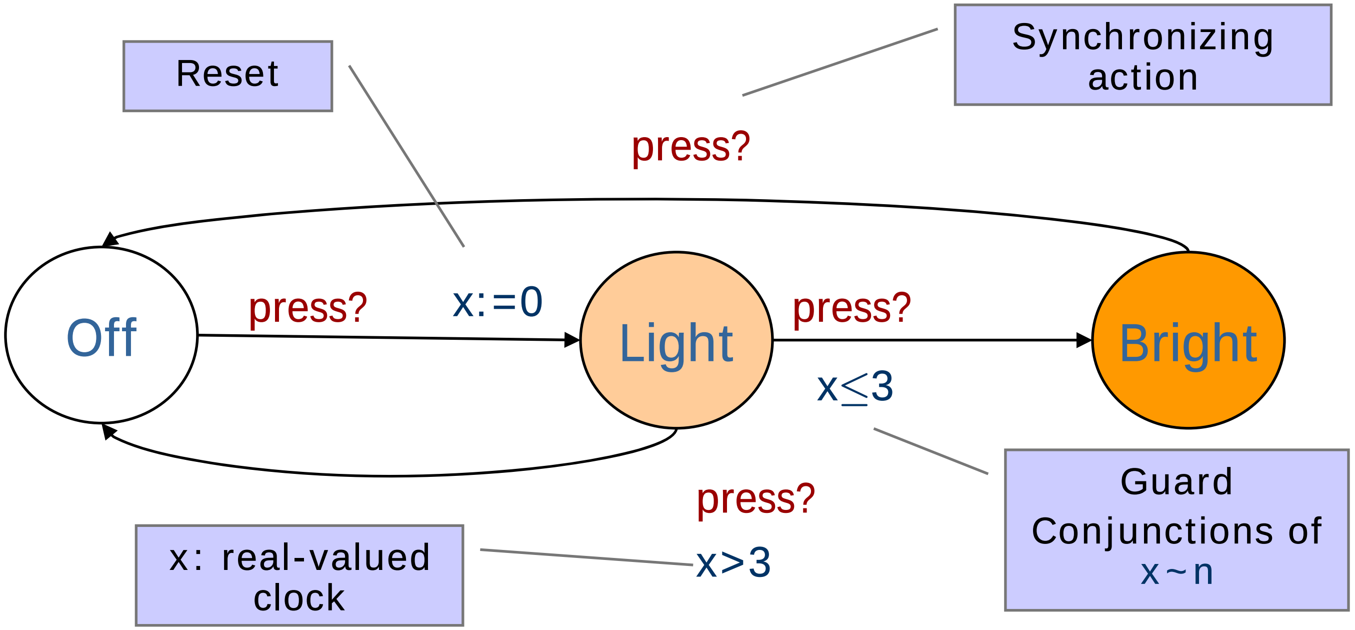
x: real-valued clock

States:
(location , x=v) where v∈**R**

Transitions:
(Off , x=0)

Timed Automata

Alur & Dill 1990



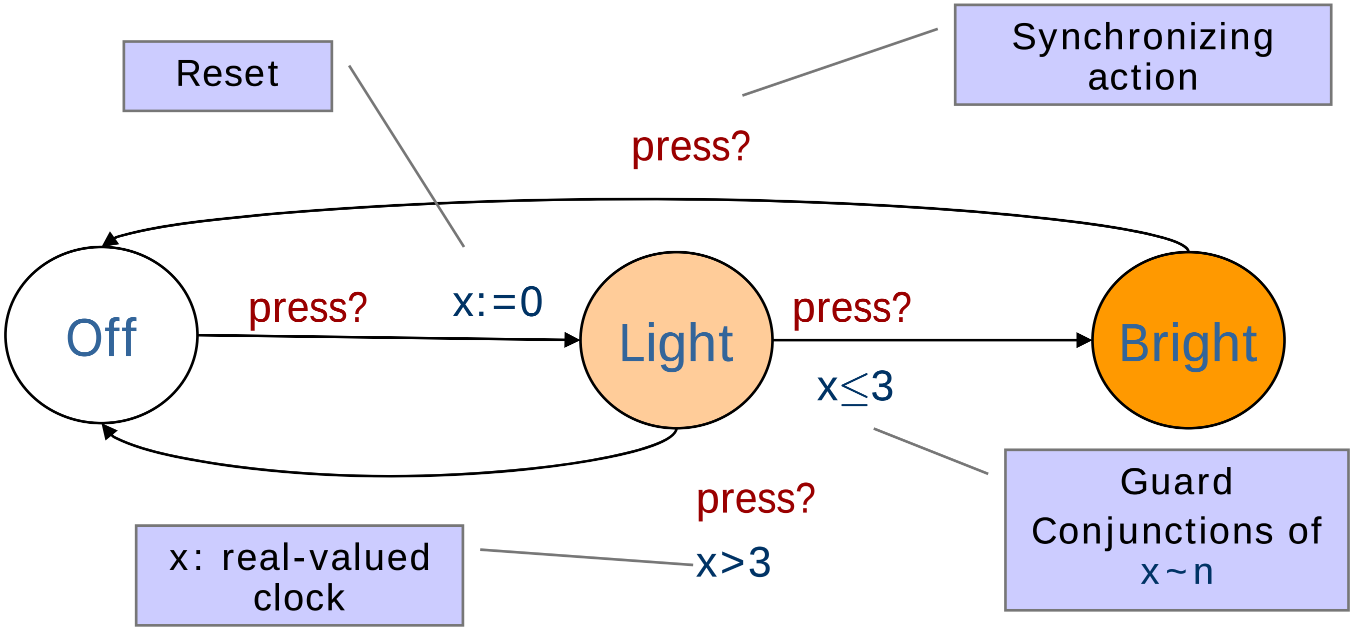
States:
(location , x=v) where v∈**R**

Transitions:

(Off , x=0)
delay 4.32 → (Off , x=4.32)

Timed Automata

Alur & Dill 1990



States:
(location , x=v) where v∈**R**

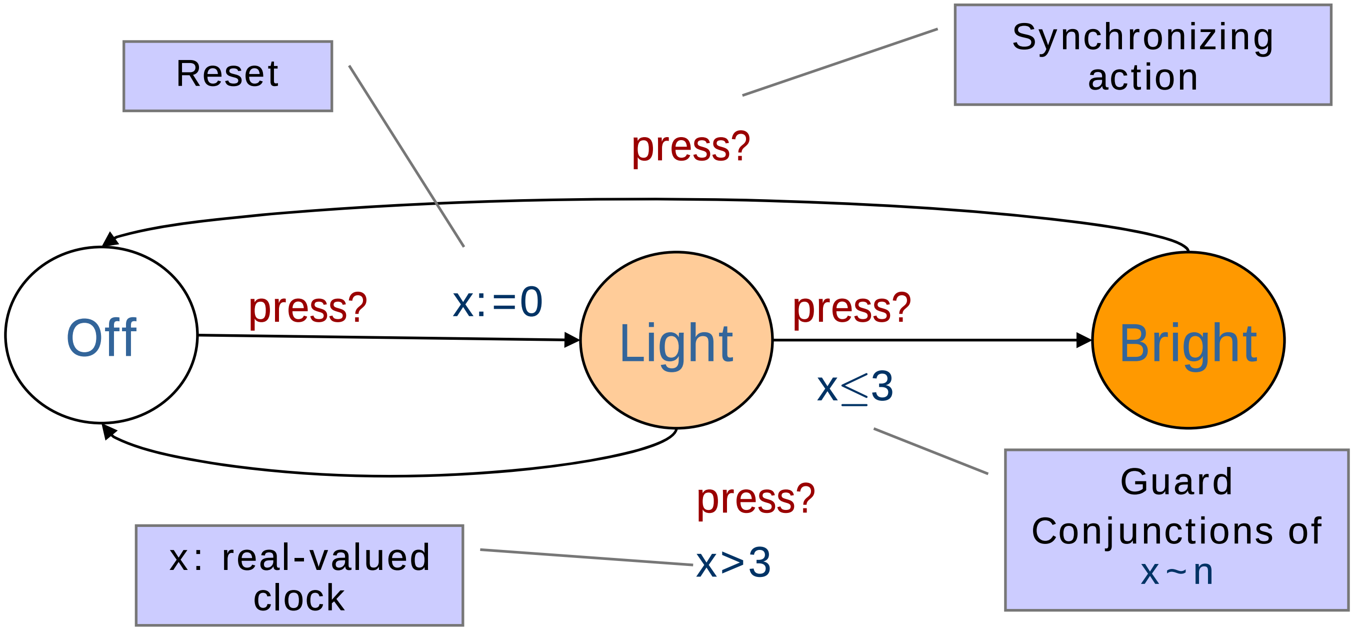
Transitions:

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

BRICS
Basic Research
in Computer Science

Timed Automata

Alur & Dill 1990



States:
(location , x=v) where v∈**R**

Transitions:

(Off , x=0)

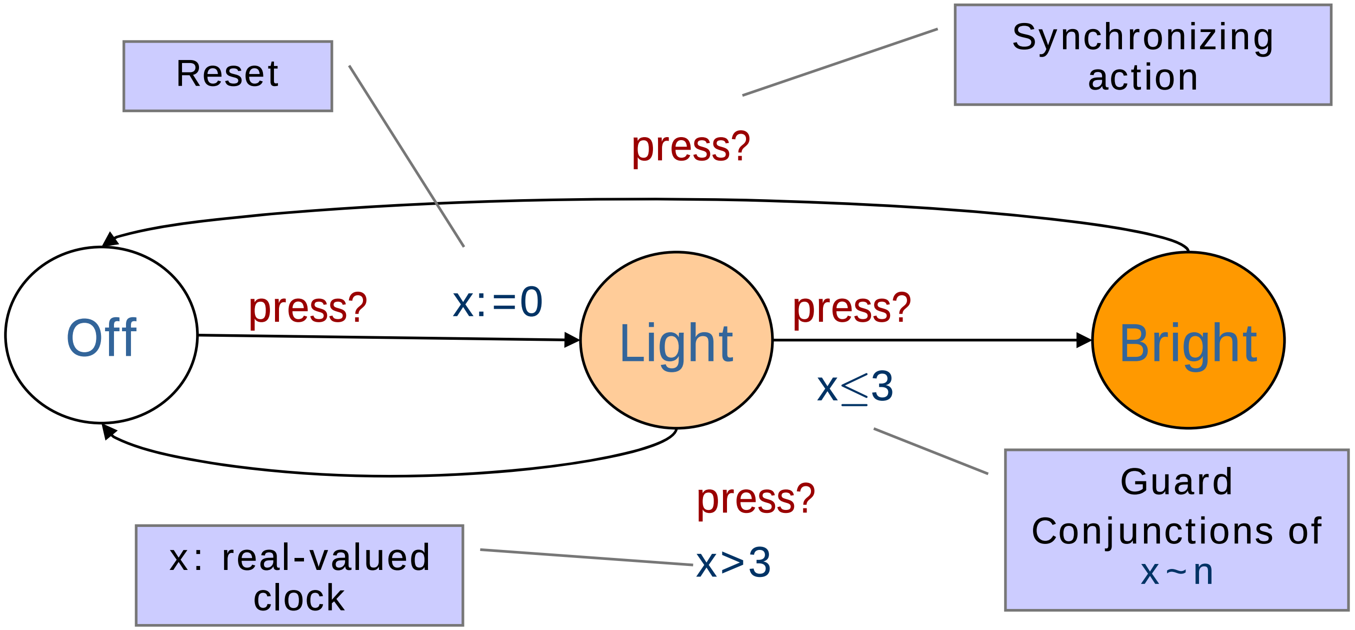
delay 4.32 → (Off , x=4.32)

press? → (Light , x=0)

delay 2.51 → (Light , x=2.51)

Timed Automata

Alur & Dill 1990



Reset

Synchronizing action

Guard
Conjunctions of
x~n

x: real-valued clock

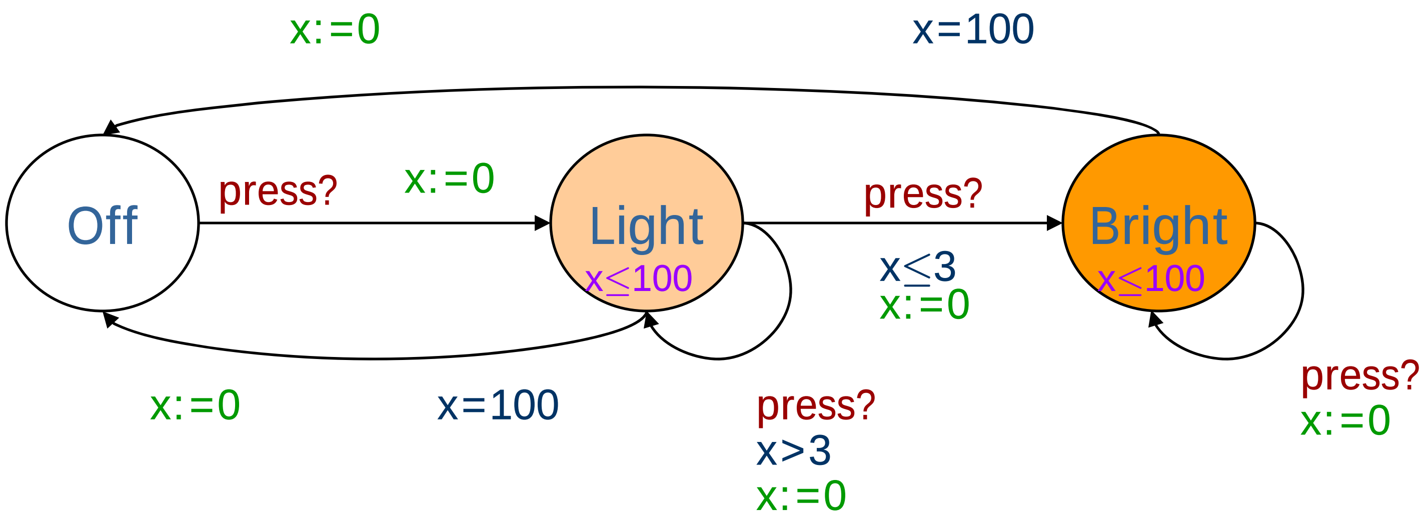
States:
(location , x=v) where v∈**R**

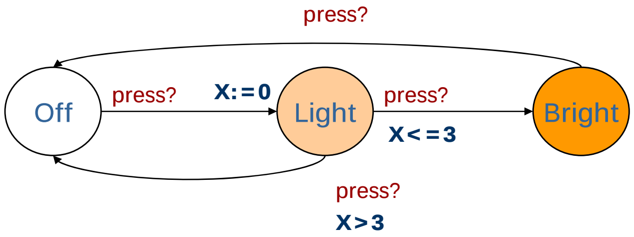
Transitions:

	(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

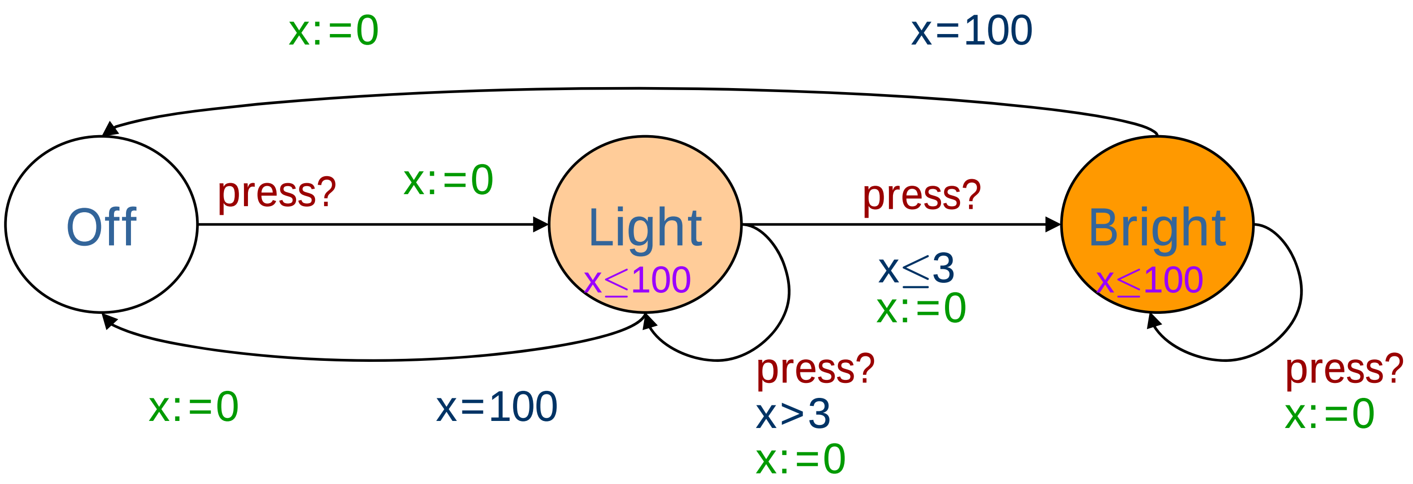
Using Invariants





Intelligent Light Control

Using Invariants



Transitions:

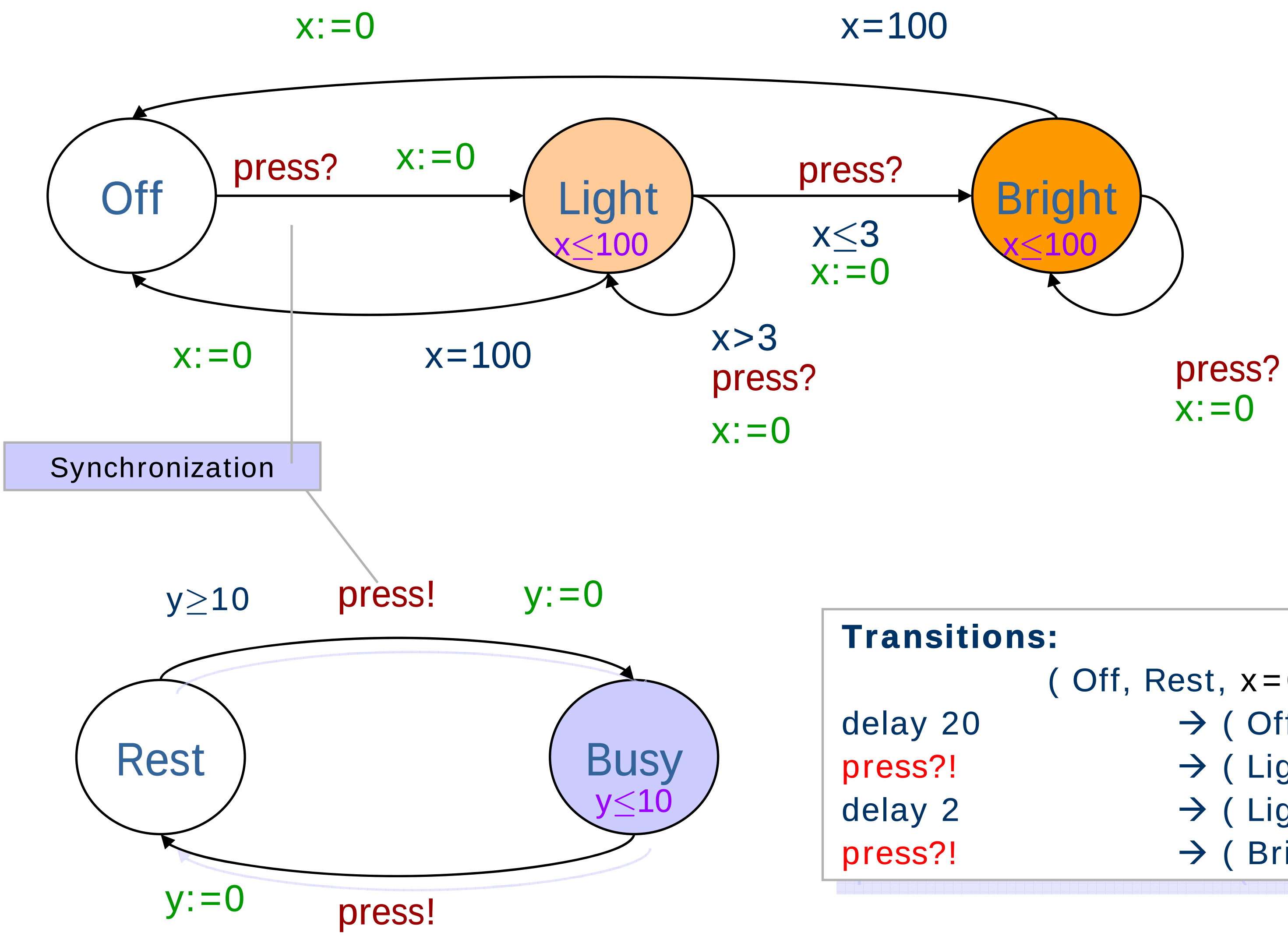
	(Off , x=0)
delay 4.32	→ (Off , x=4.32)
press?	→ (Light , x=0)
delay 4.51	→ (Light , x=4.51)
press?	→ (Light , x=0)
delay 100	→ (Light , x=100)
τ	→ (Off , x=0)

Note: X

(Light , x=0) delay 103 →

Invariants ensures progress

Networks Light Controller & User

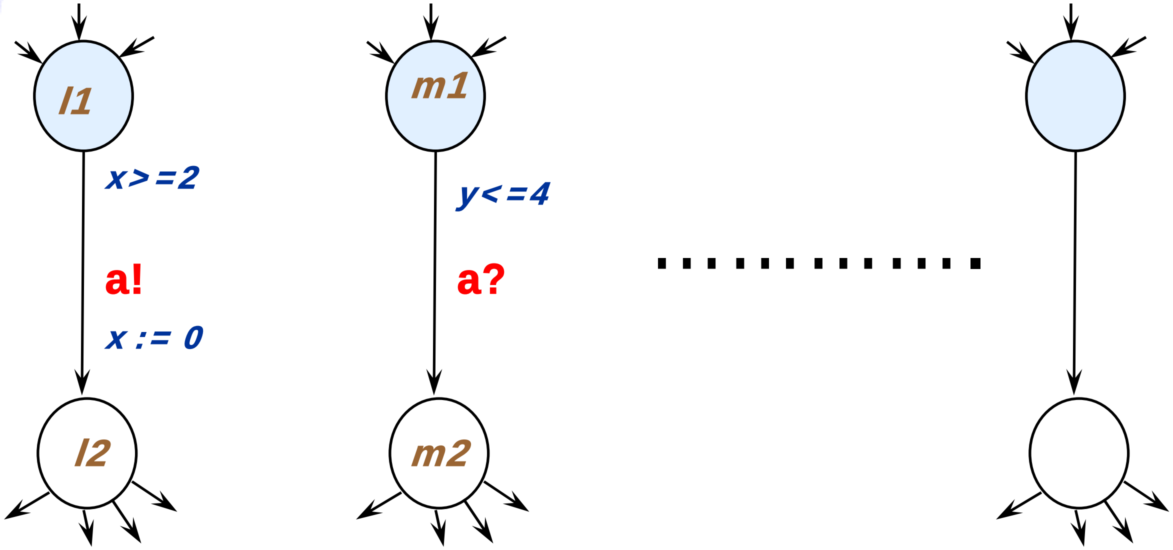


Transitions:

	(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!

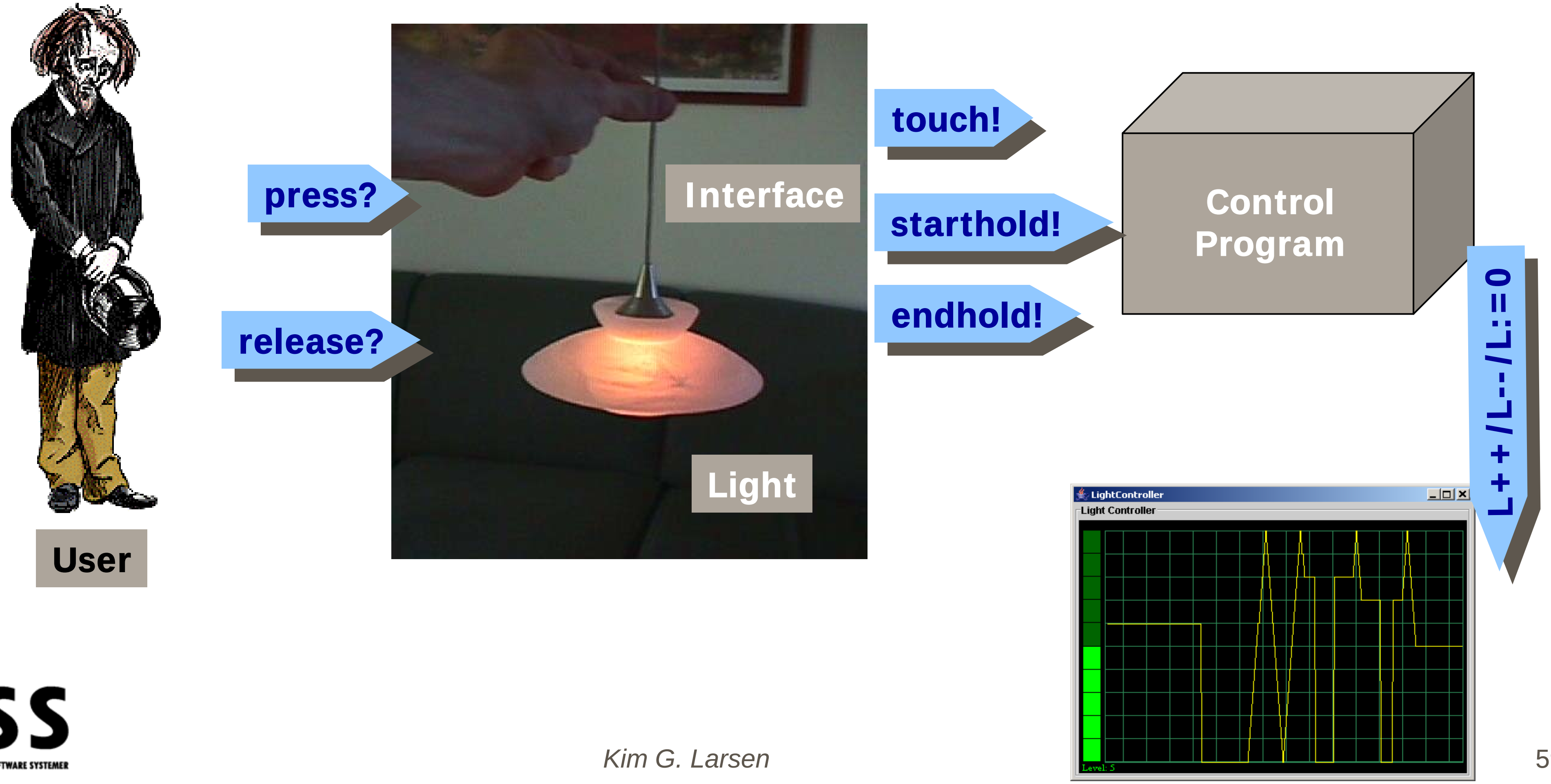
Example transitions

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

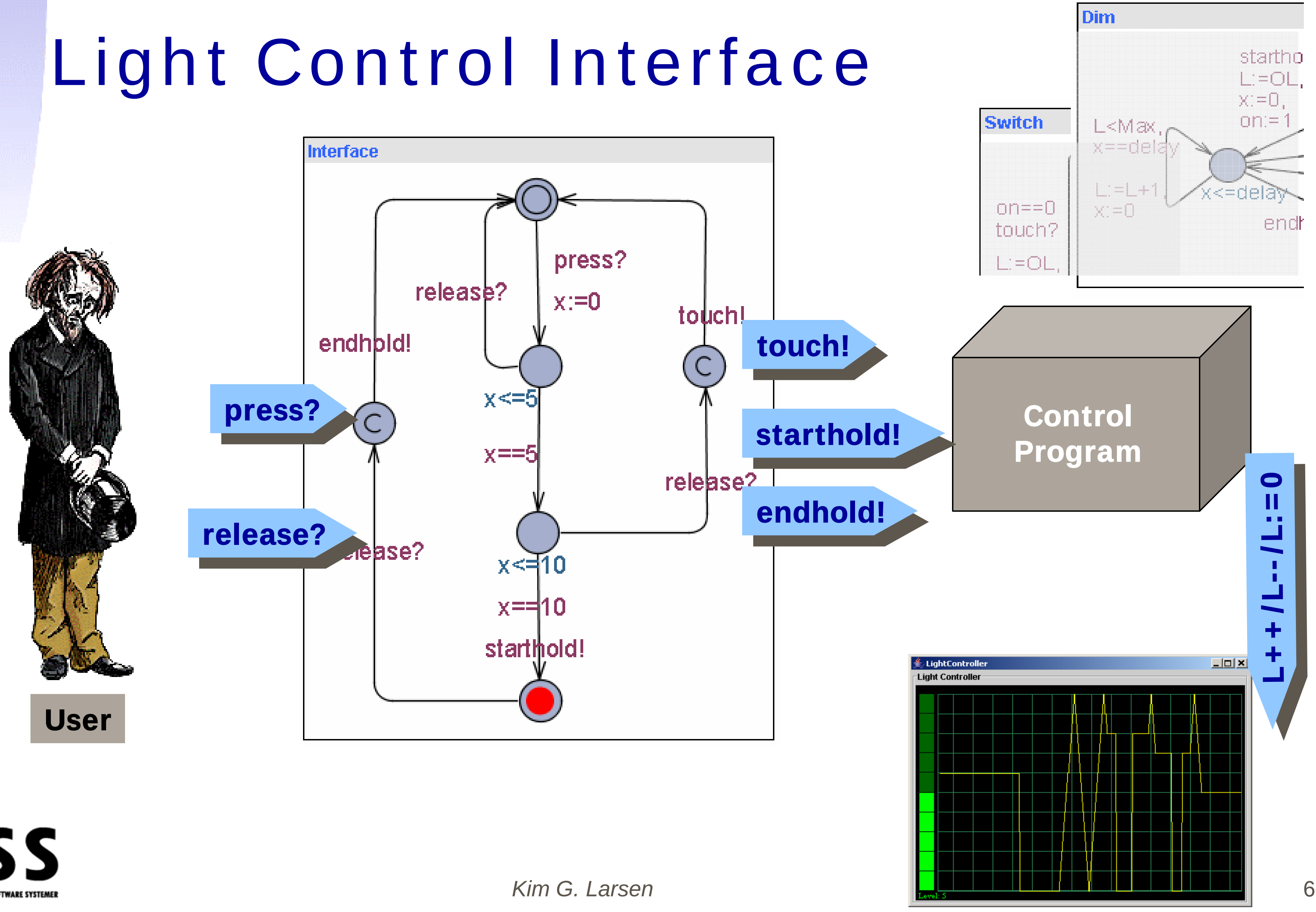
$\cancel{\xrightarrow{0.2}}$ *(**l1**,**m1**,………,x=2.2, y=3.7, …..)*

If **a** URGENT CHANNEL

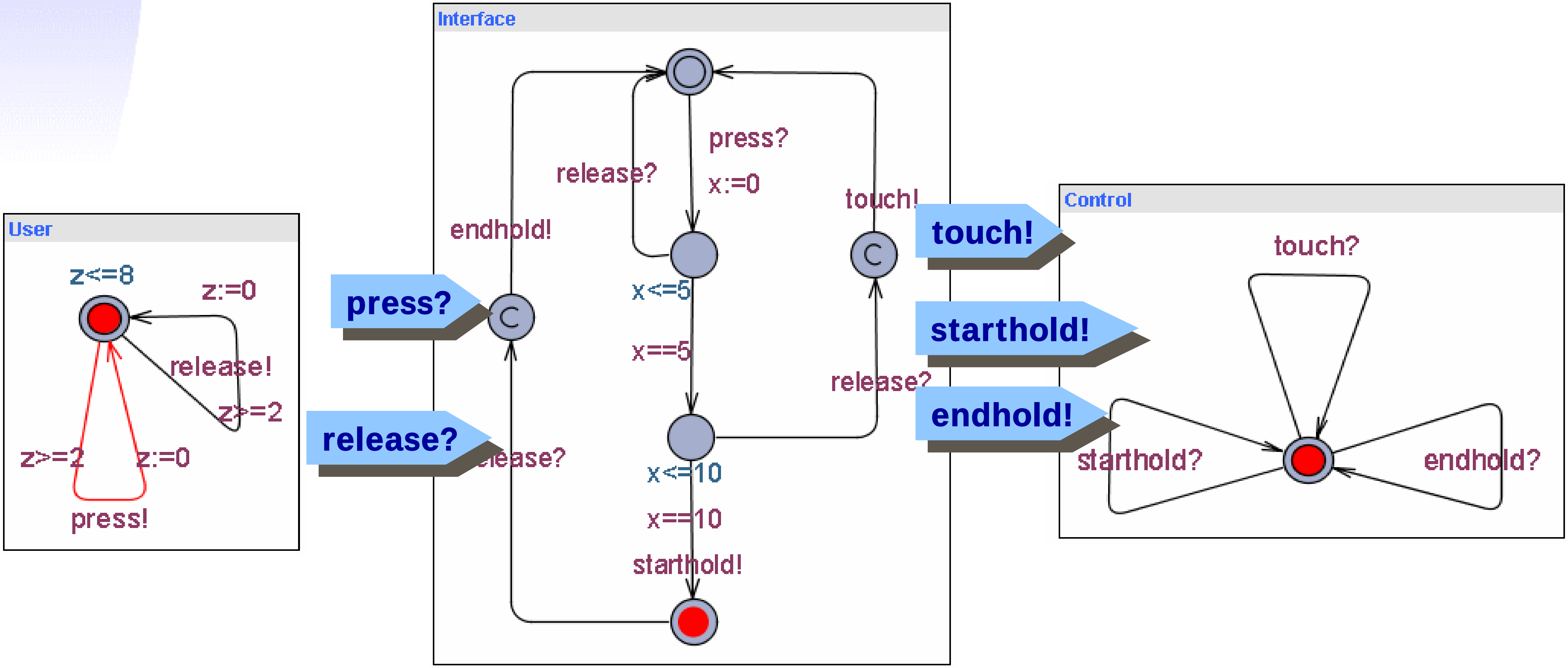
Light Control Interface



Light Control Interface



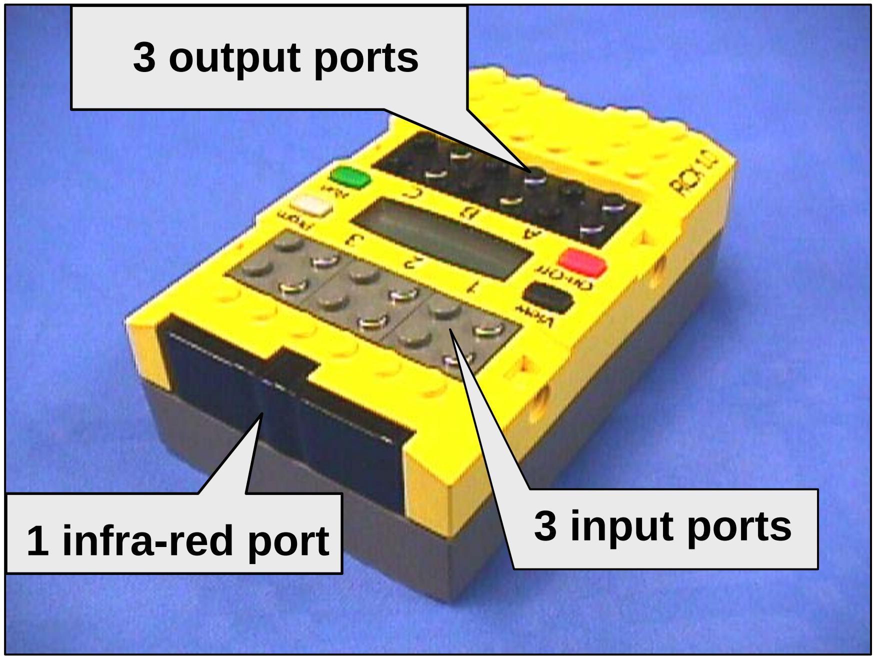
Light Control Network



BRICK SORTING

LEGO Mindstorms/RCX

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

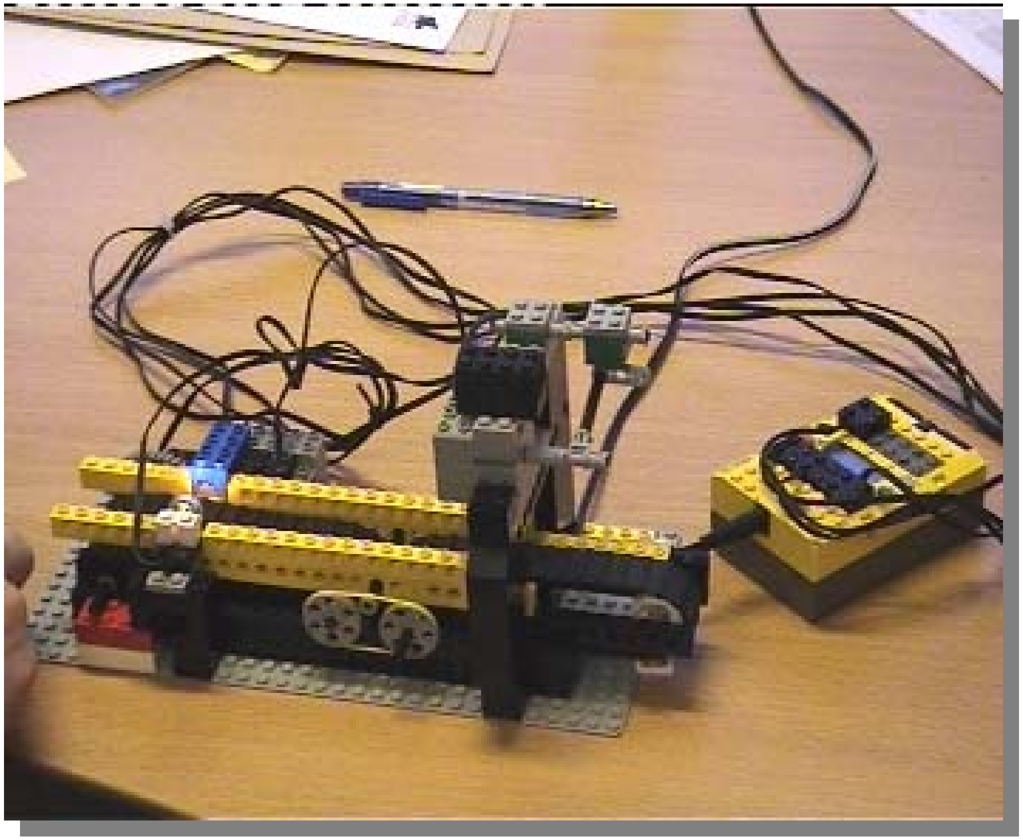


- Several Programming Languages:
  - NotQuiteC, Mindstorm, Robotics, legOS, etc.

A Real Timed System

The Plant

Conveyor Belt
&
Bricks



Controller Program

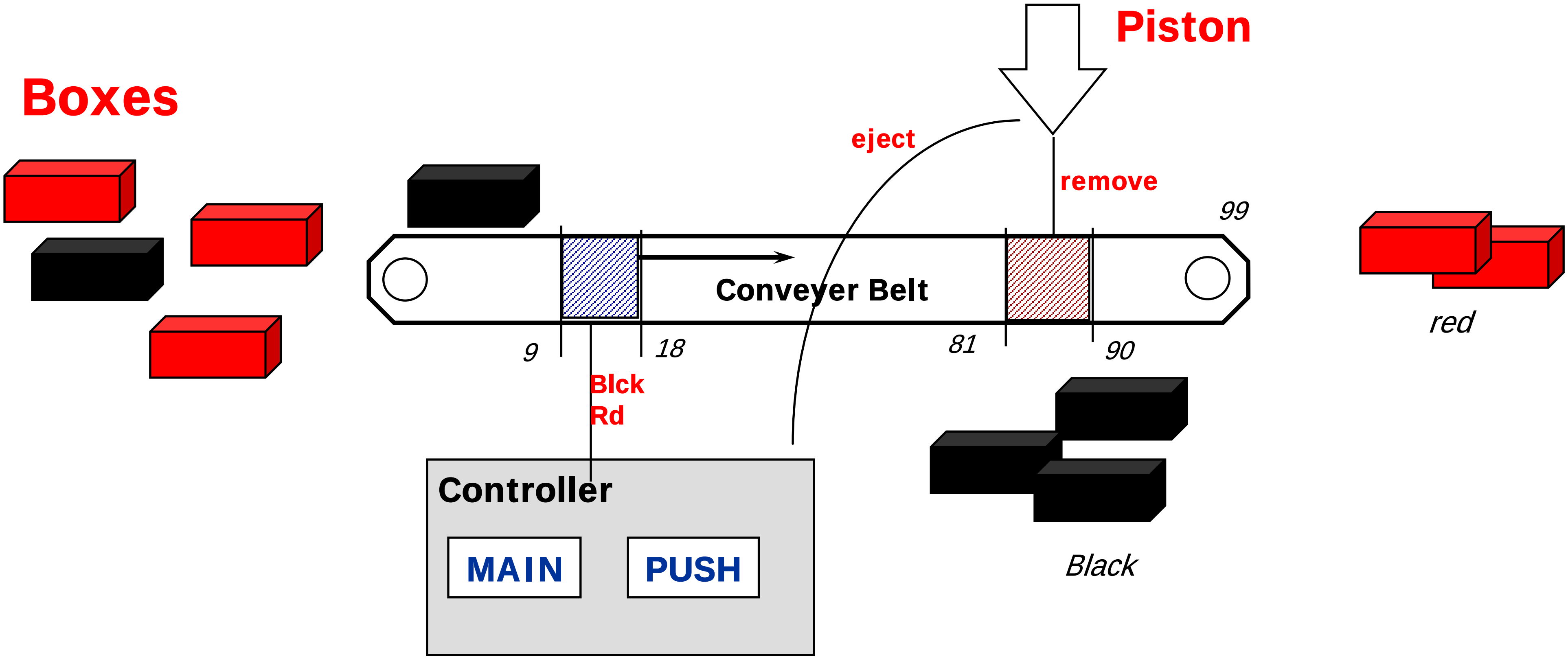
LEGO MINDSTORM

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

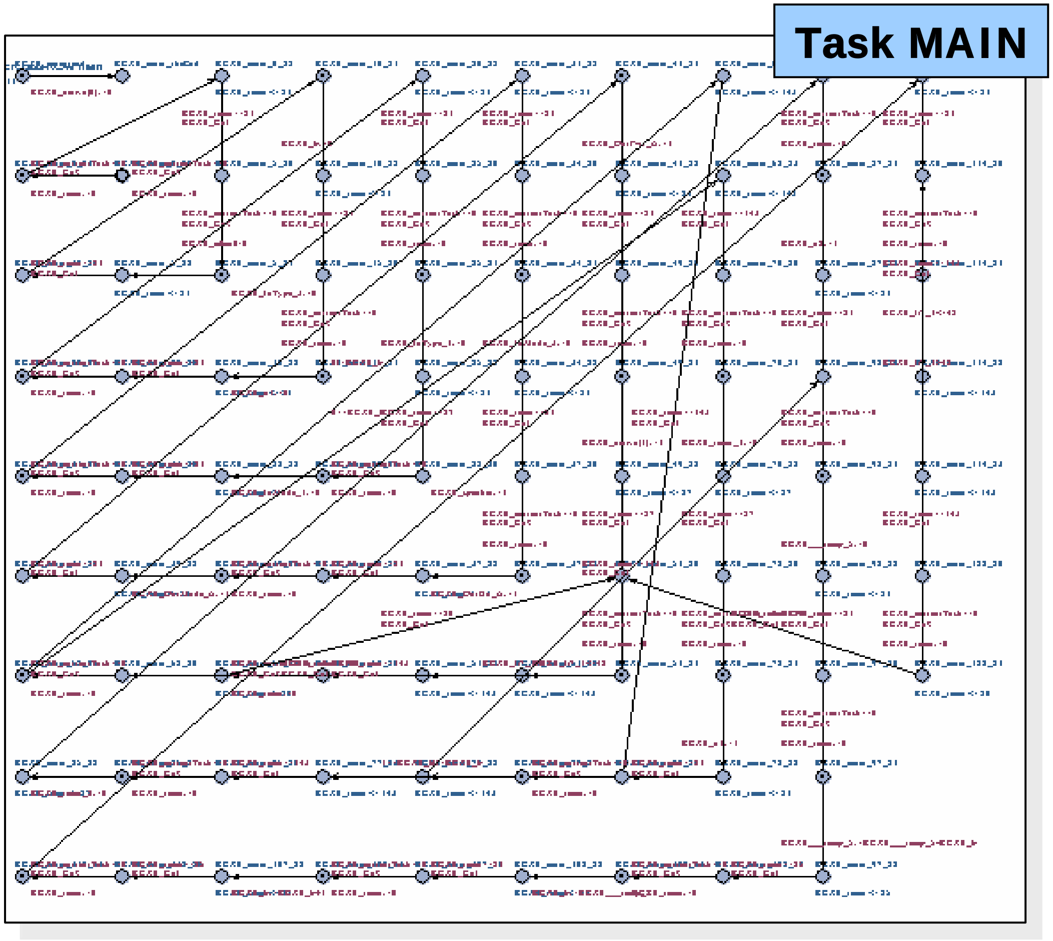
```
int active;
int DELAY;
int LIGHT_LEVEL;
```

```
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);
  }
}
```

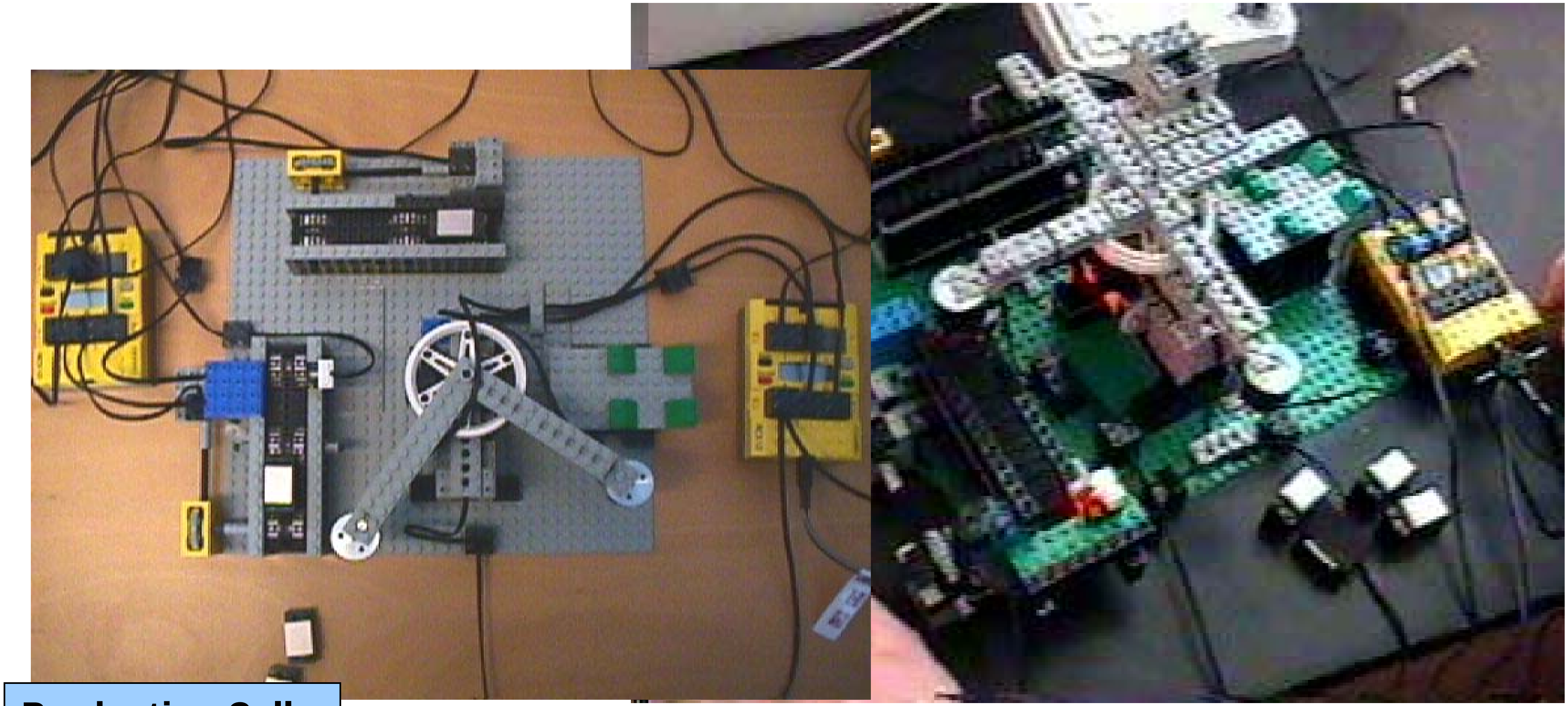
From RCX to UPPAAL

- 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