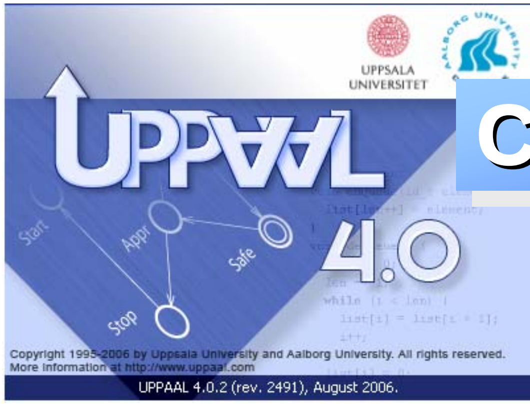


CLASSIC

Modelling, Specification, and Verification

using UPPAAL

Kim Guldstrand Larsen

AALBORG UNIVERSITY DENMARK

BRICS
Basic Research in Computer Science

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Modelling using Finite State Machines

AALBORG UNIVERSITY DENMARK

BRICS
Basic Research in Computer Science

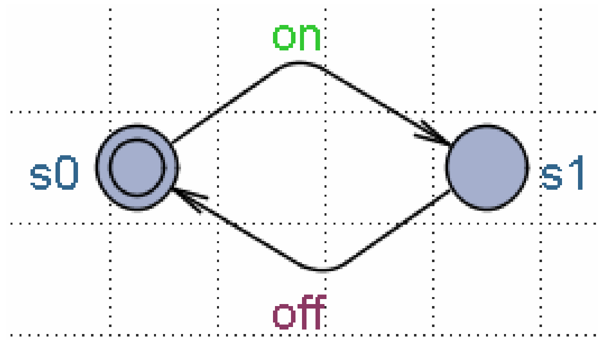
CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Modelling processes

BRICS
Basic Research
in Computer Science

- 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

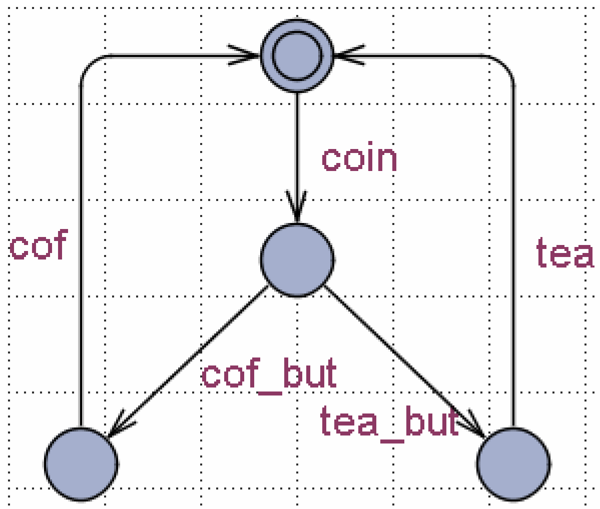


on→off→on→off→on→off→ ....

a sequence of actions or *trace*

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Modelling Choices

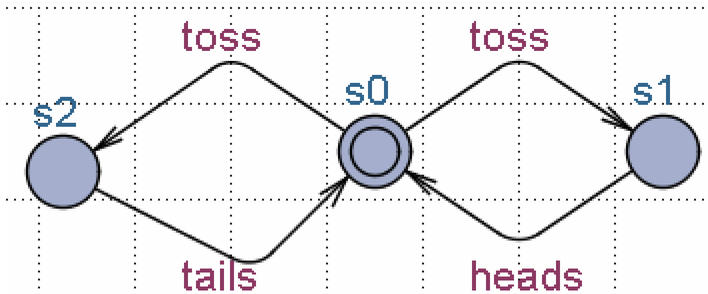


- Who or what makes the choice?
- Is there a difference between input and output actions?

Non-deterministic Choice

BRICS
Basic Research
in Computer Science

- Tossing a coin

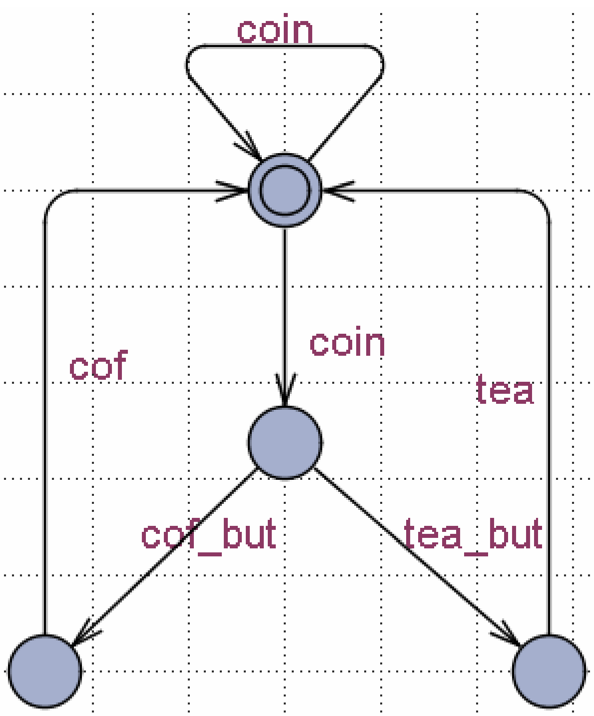


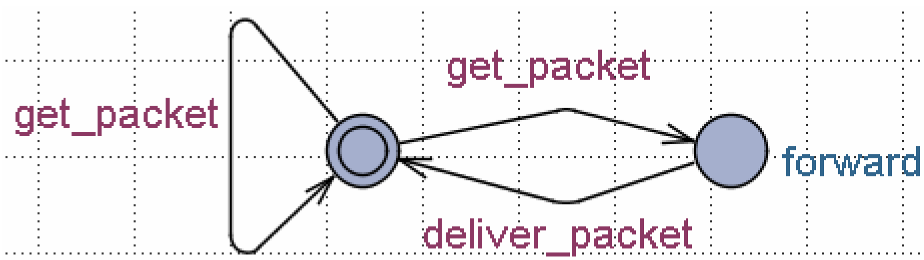
- Possible traces?
  - Both outcomes possible
  - Nothing said about relative frequency
  - If coin is fair, the outcome is 50/50

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?

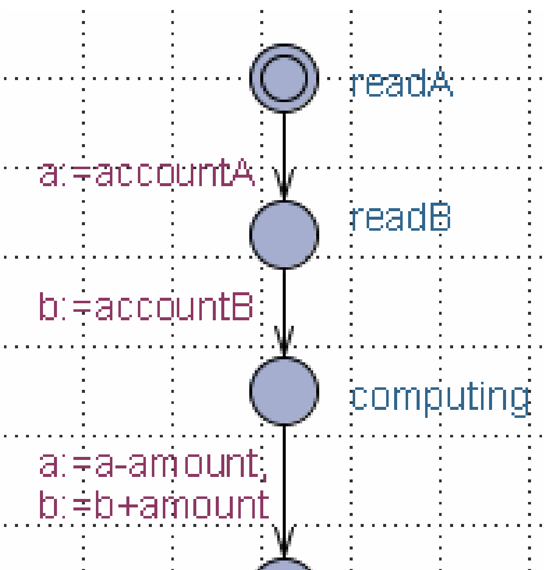
Use non-determinism...

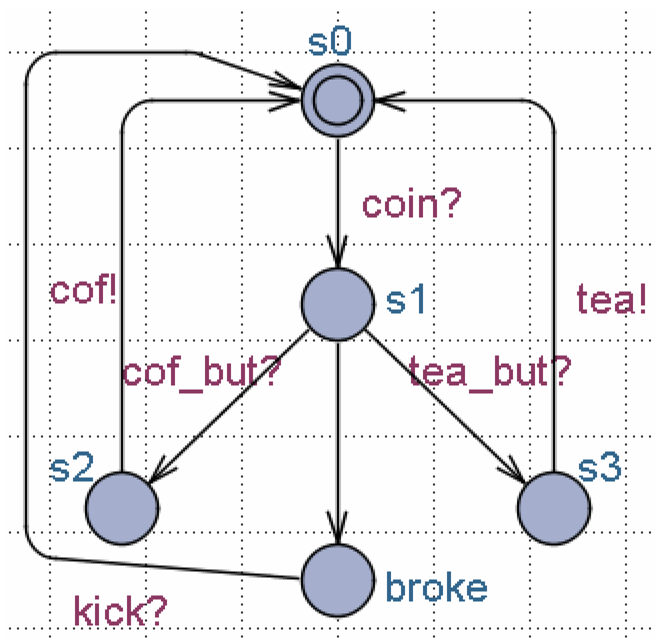


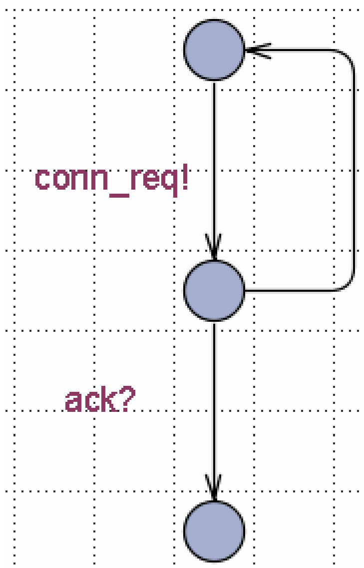


Internal-Actions

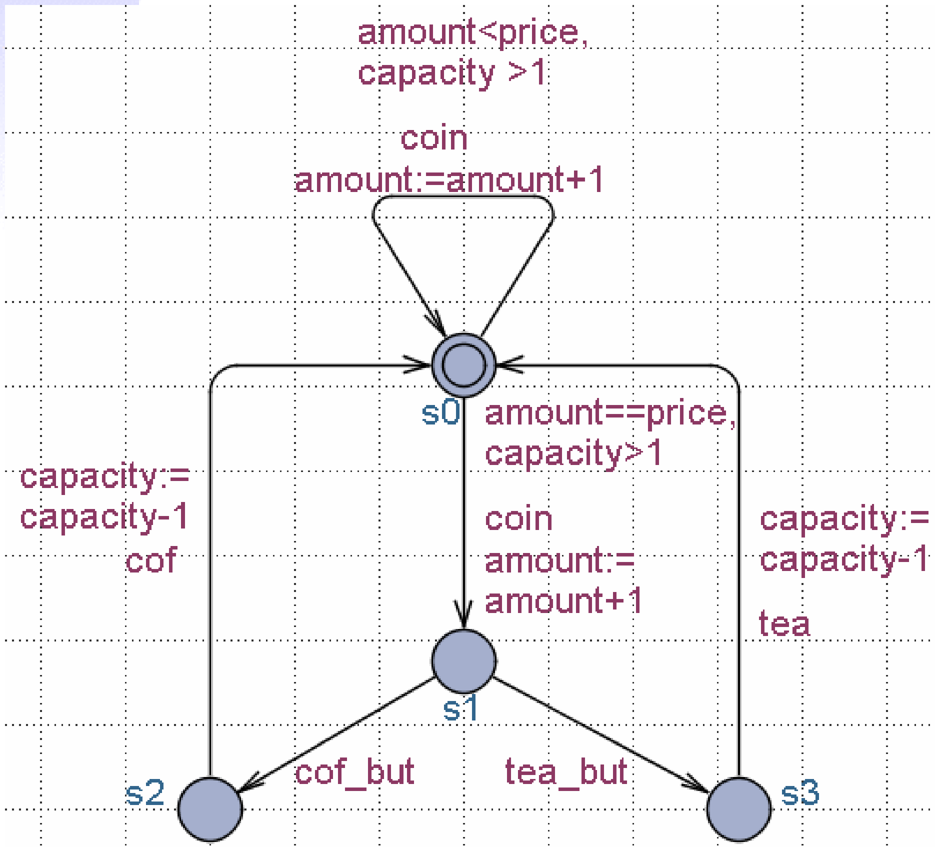
- Spontaneous actions
- Internal actions
- Tau-actions
- Internal transitions can be taken on the initiative of a single machine without communication with others







Extended FSM



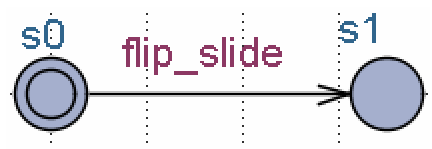
- **EFSM** =
  FSM +
  variables +
  enabling conditions +
  assignments
- Transition still atomic
- Can be translated into FSM if variables have bounded domain
- State: control location
  + variable values:

  (state,amount,capacity)
- (s0,5,10)

Parallel Composition: interleaving

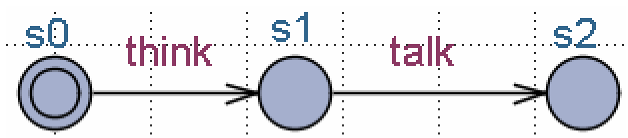
BRICS
Basic Research
in Computer Science

Flipper



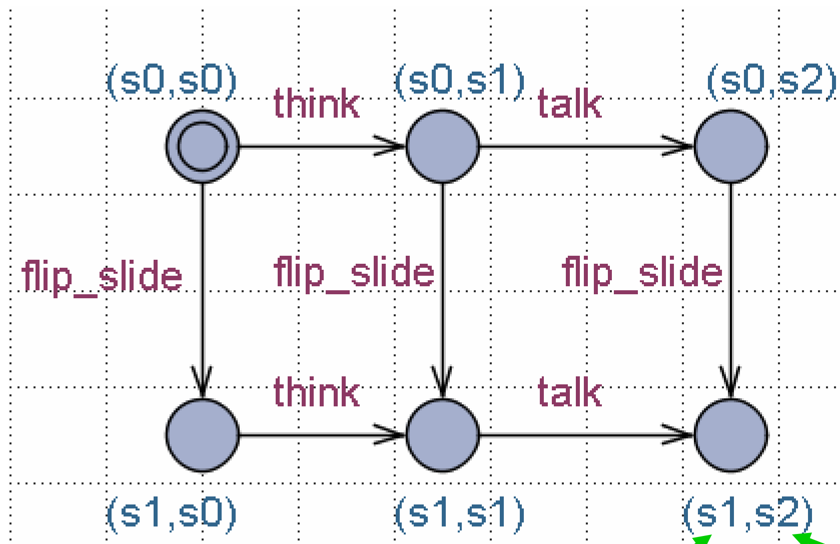
2 states

Speaker



3 states

Lecturer =
Speaker || Flipper



2*3 states

from Flipper

from Speaker

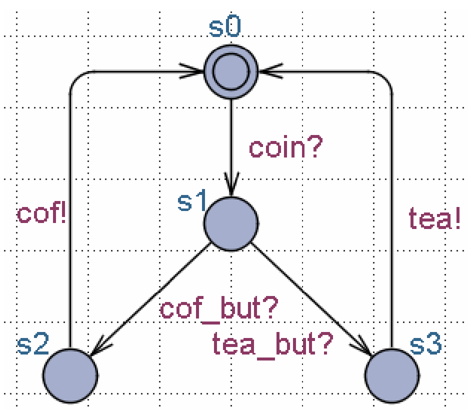
CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Process Interaction

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

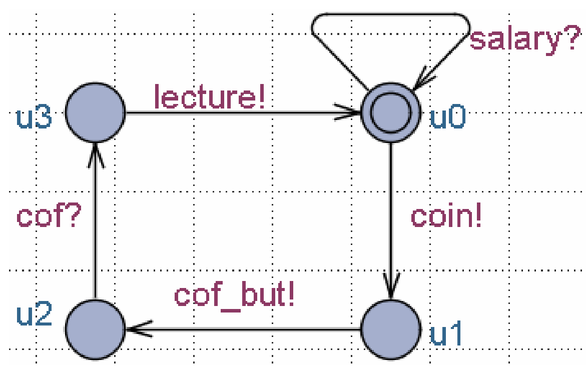
University=
Coffee Machine || Lecturer
•LTS?
•How many states?
•Traces ?

Coffee Machine

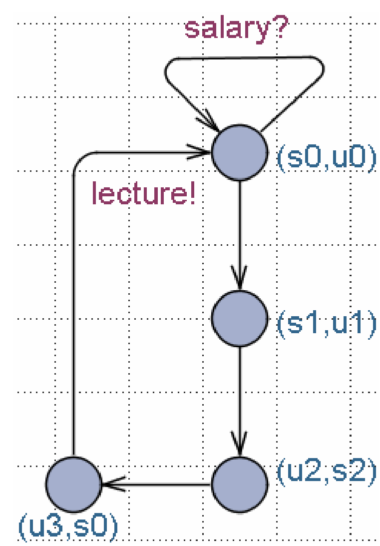


4 states

Lecturer

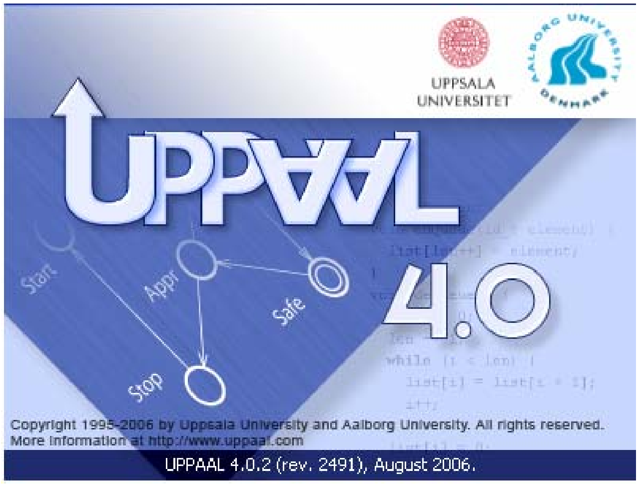


4 states



synchronization results in internal actions

4 states:Interaction constrain overall behavior



Adding Time

DaNES
Danish Network for Intelligent Embedded Systems

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Collaborators

@UPPsala

UPPSALA UNIVERSITET

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

@AALborg

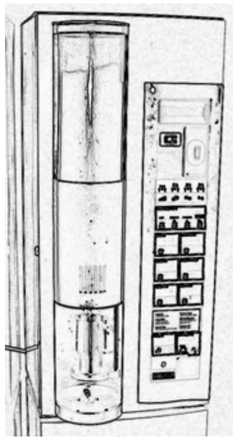
AALBORG UNIVERSITY DENMARK

- Kim G Larsen
- Gerd Behrman
- Arne Skou
- Brian Nielsen
- Alexandre David
- Jacob I. Rasmussen
- Marius Mikucionis
- Thomas Chatain

@Elsewhere

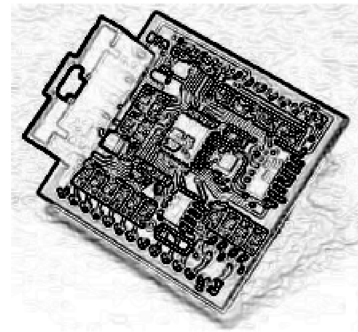
- Emmanuel Fleury, Didier Lime, Johan Bengtsson, Fredrik Larsson, Kåre J Kristoffersen, Tobias Amnell, Thomas Hune, Oliver Möller, Elena Fersman, Carsten Weise, David Griffioen, Ansgar Fehnker, Frits Vandraager, Theo Ruys, Pedro D'Argenio, J-P Katoen, Jan Tretmans, Judi Romijn, Ed Brinksma, Martijn Hendriks, 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

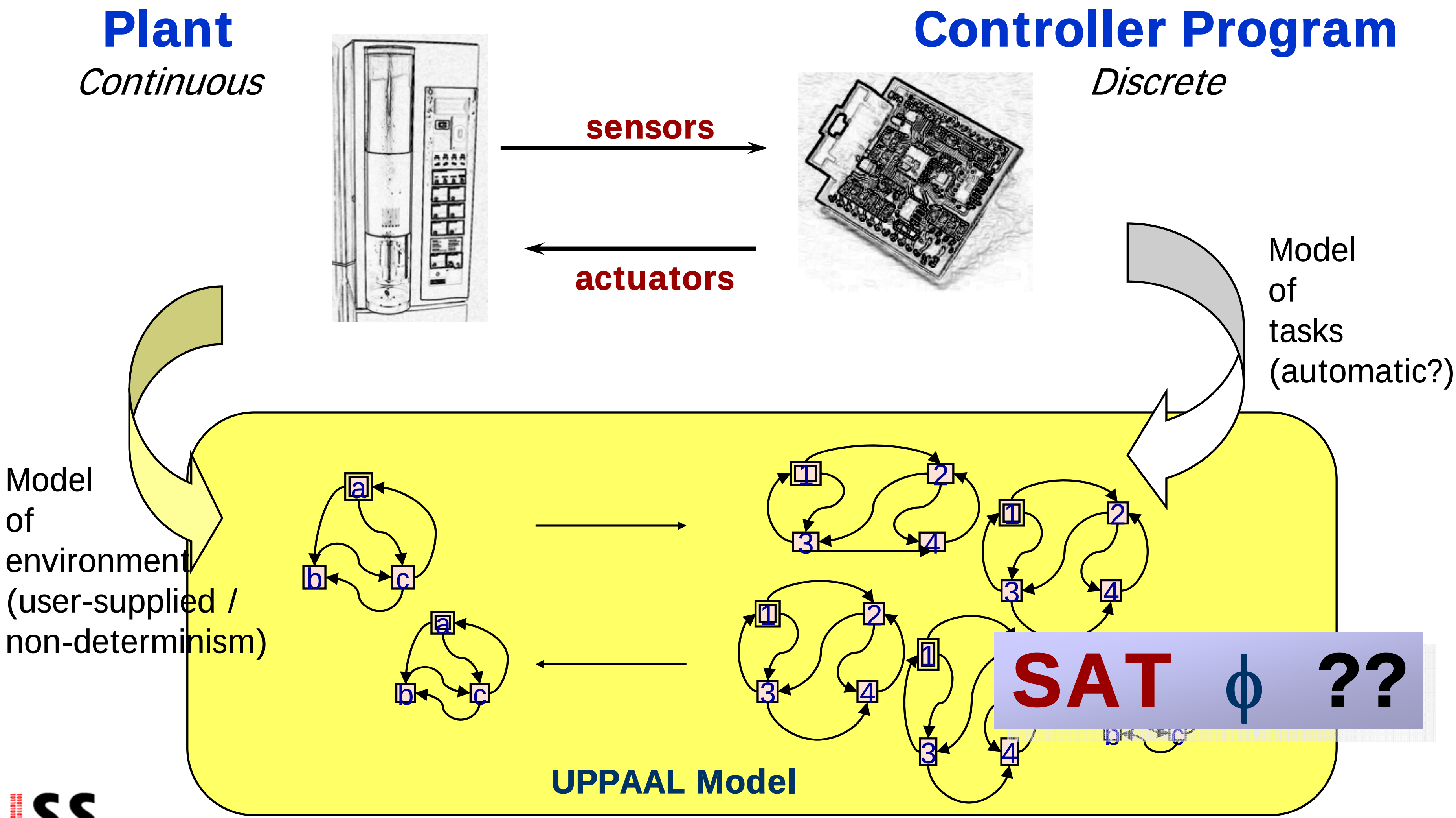
Discrete

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

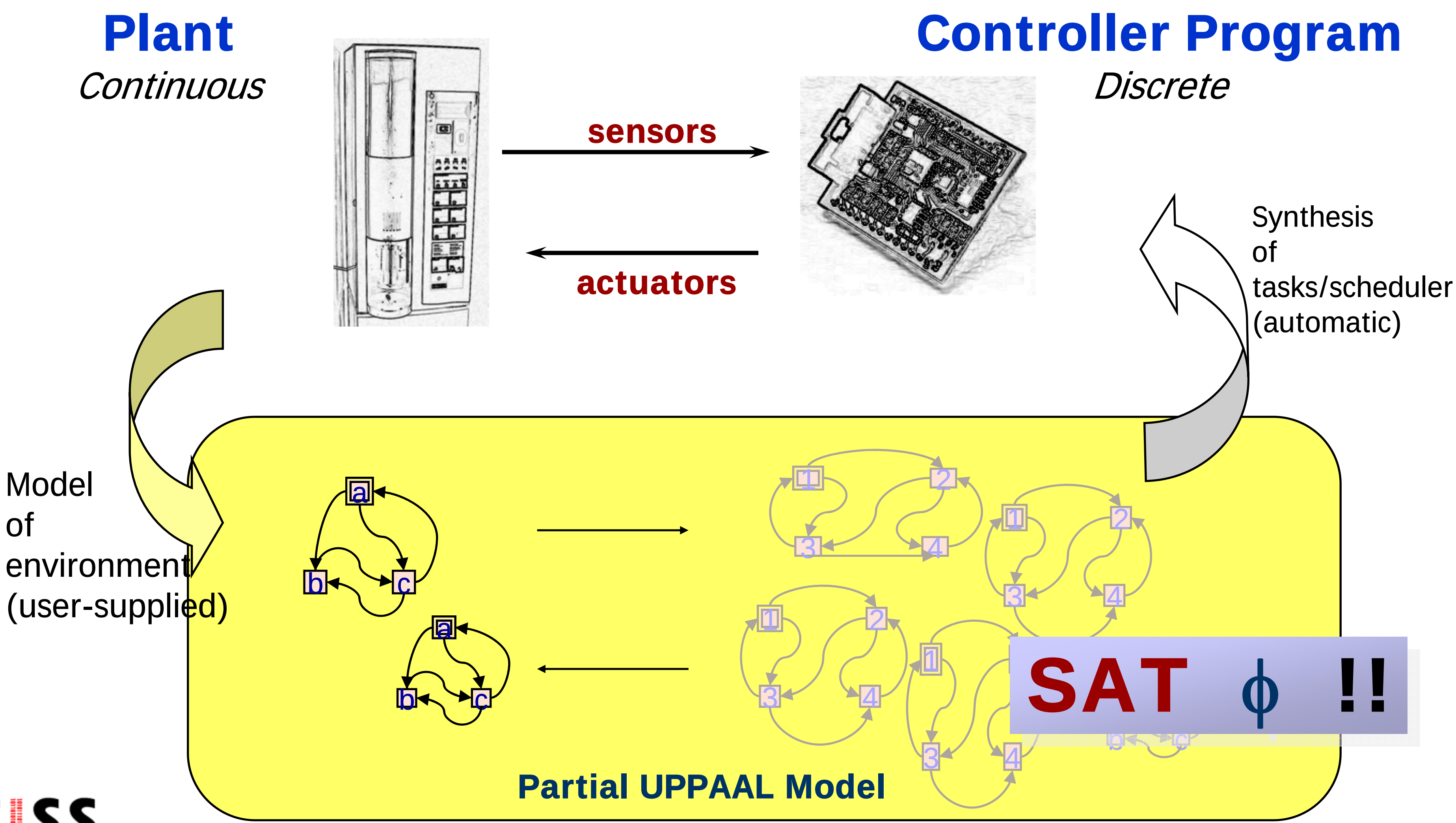
> **Real Time System**
> A system where correctness not only depends on the logical order of events but also on their **timing**!!

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

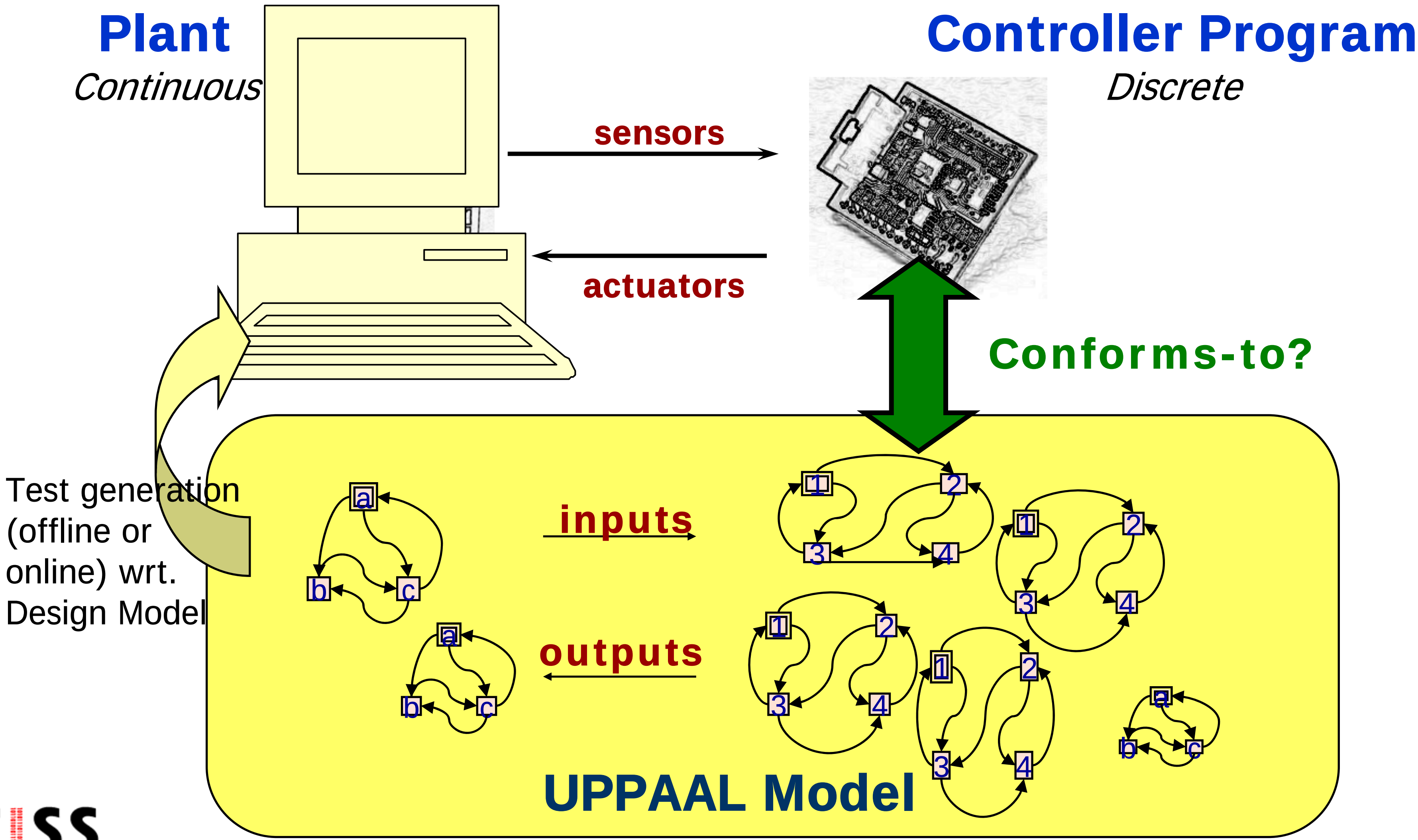
Real Time Model Checking



Real Time Control Synthesis

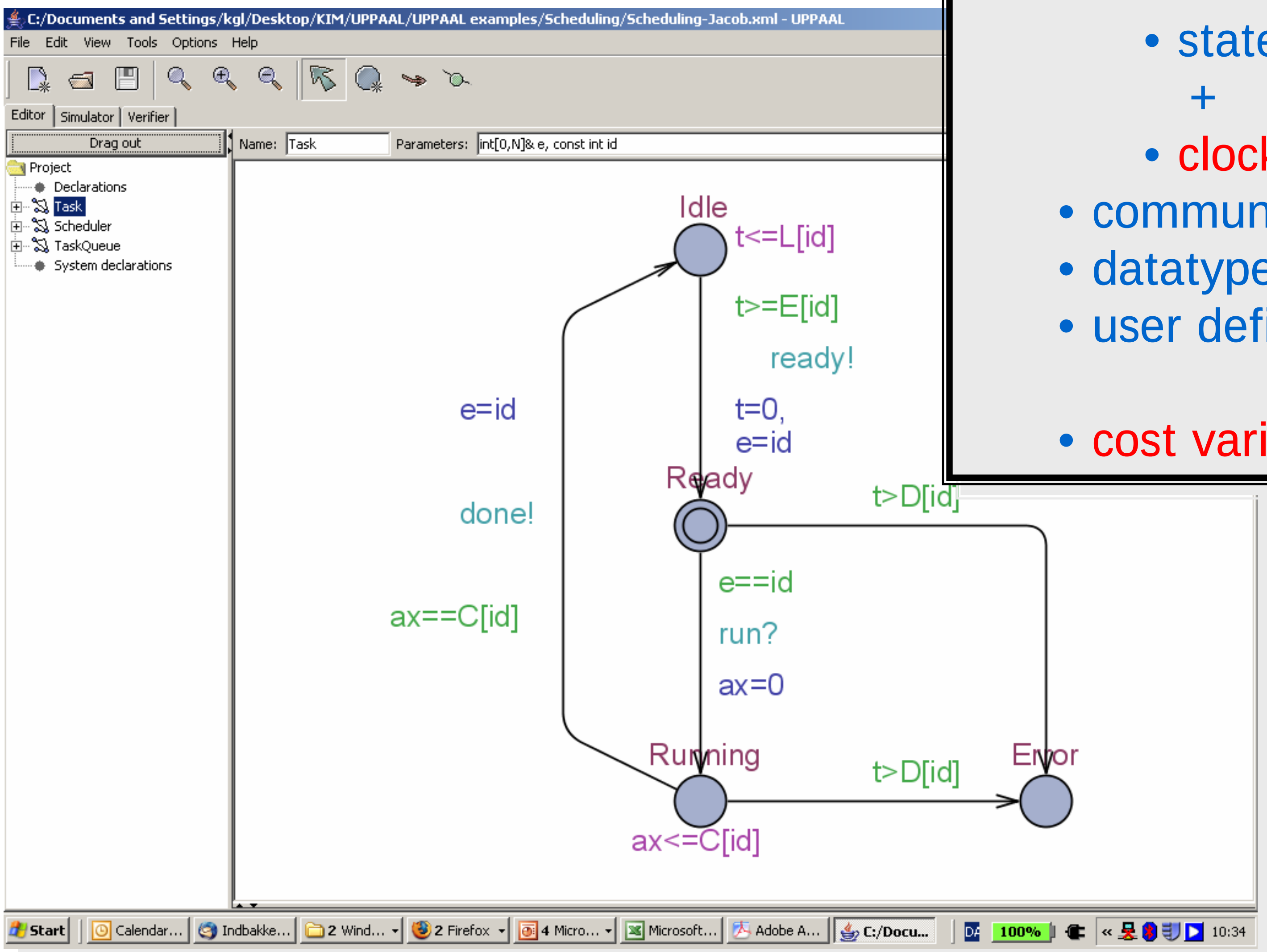


Real-time Model-Based Testing



CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

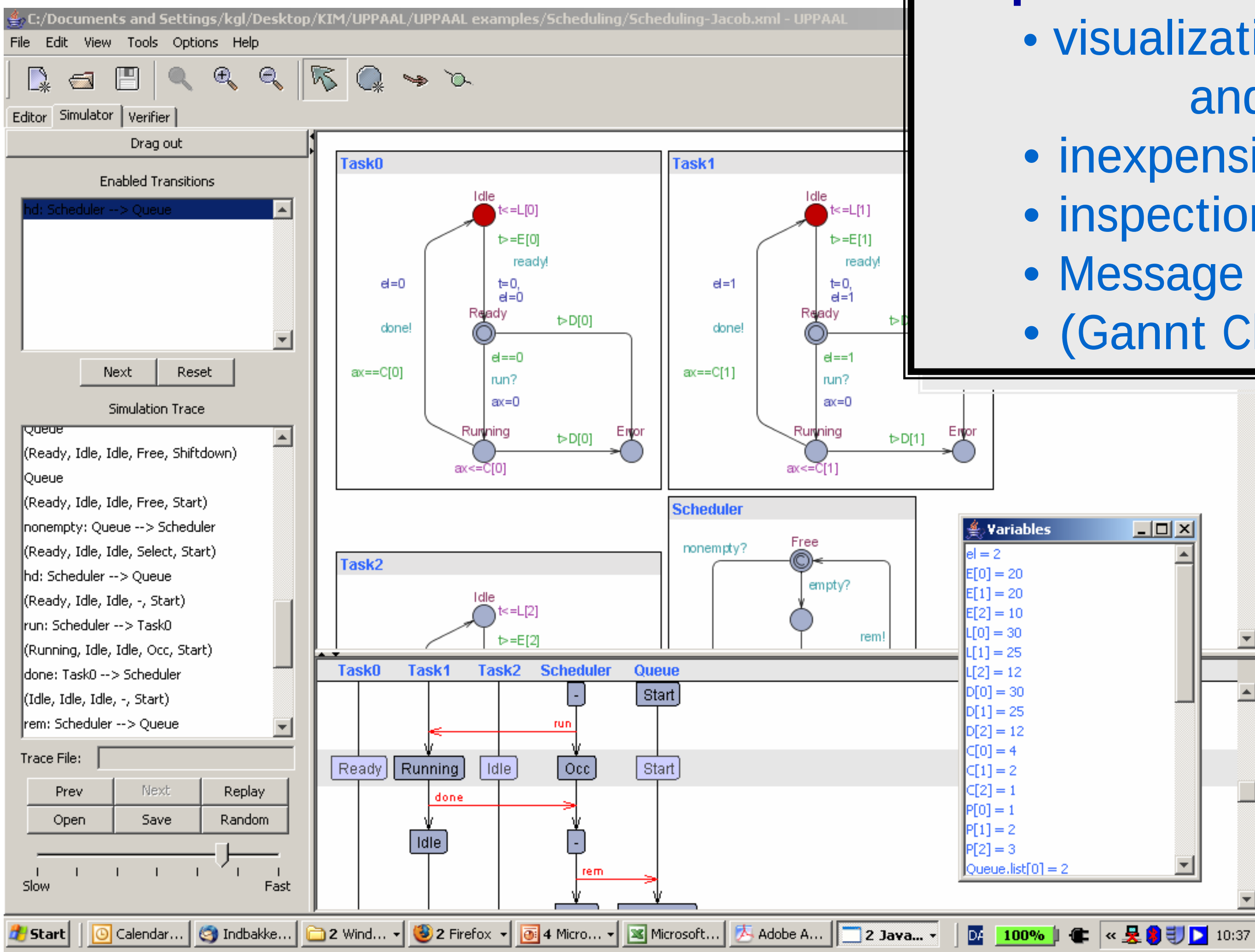
UPPAAL



Graphical Design Tool

- timed automata =
  - state machines
  - \+
  - clocks
- communication
- datatypes
- user defined functions
- cost variable

UPPAAL

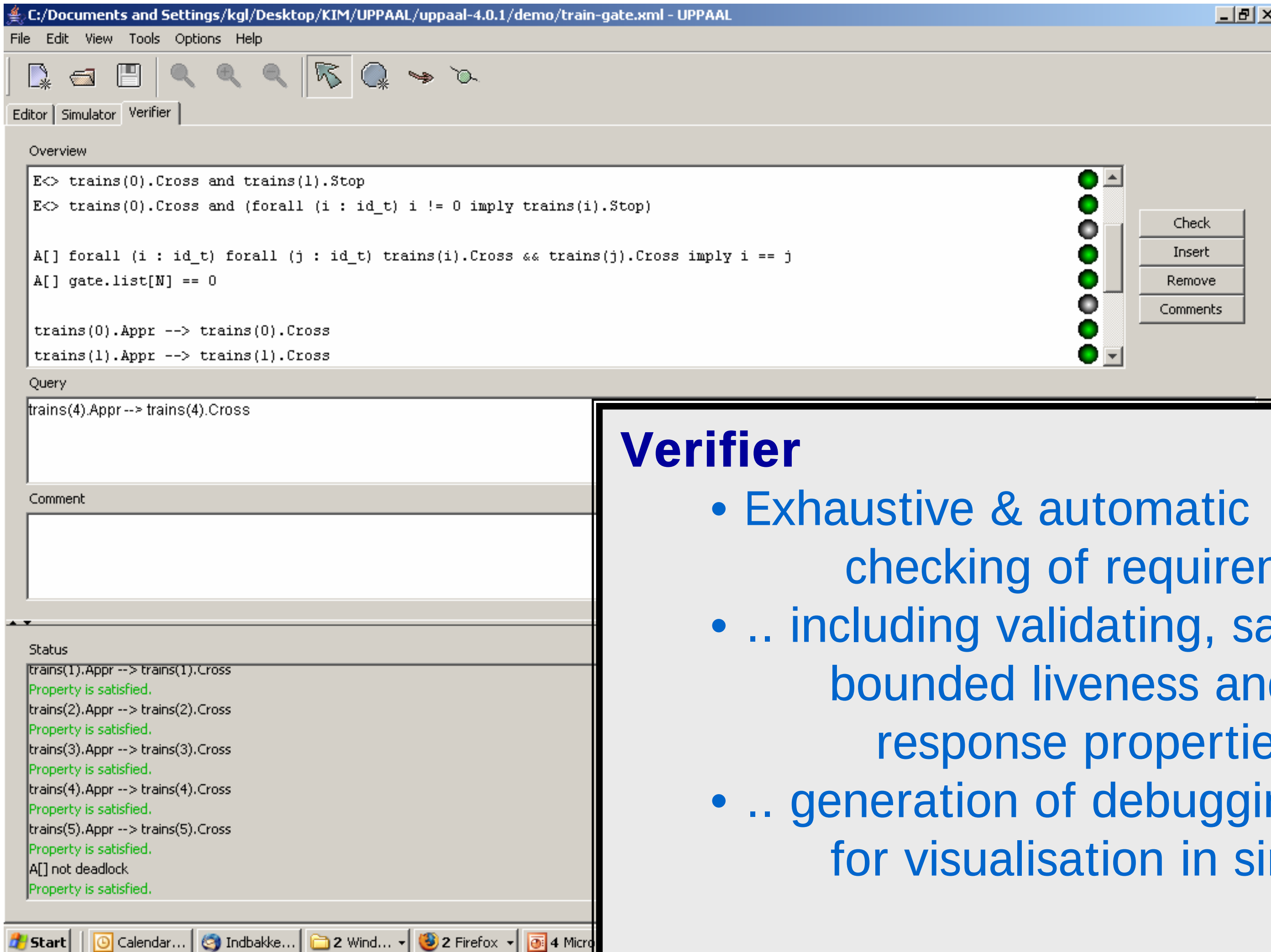


Graphical Simulator

- visualization
  and recording
- inexpensive fault detection
- inspection of error traces
- Message Sequence Charts
- (Gannt Charts)

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

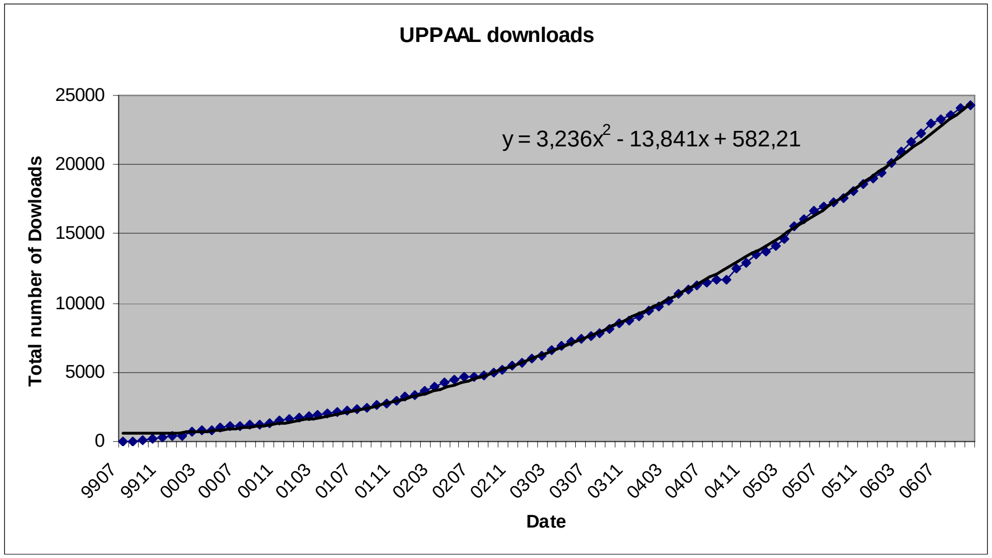
UPPAAL

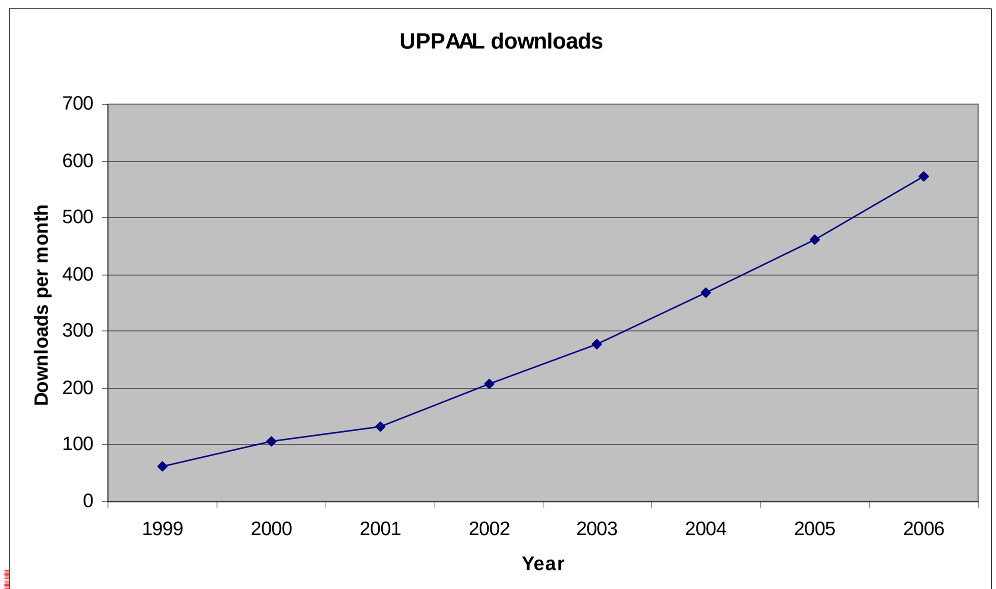


Verifier

- Exhaustive & automatic checking of requirements
- .. including validating, safety, liveness, bounded liveness and response properties
- .. generation of debugging information for visualisation in simulator.
- Optimal scheduling for cost models

“Impact





Google:

UPPAAL:	134.000
SPIN Verifier:	242.000
nuSMV:	57.700

> 1.500
Google Scholar Citations
(Rhapsody/Esterel < 3.500)

CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Impact

Academic Courses @

DTU, MCI, IT-U (DK)

Chalmers, Linköping,Lund, Chalmers,

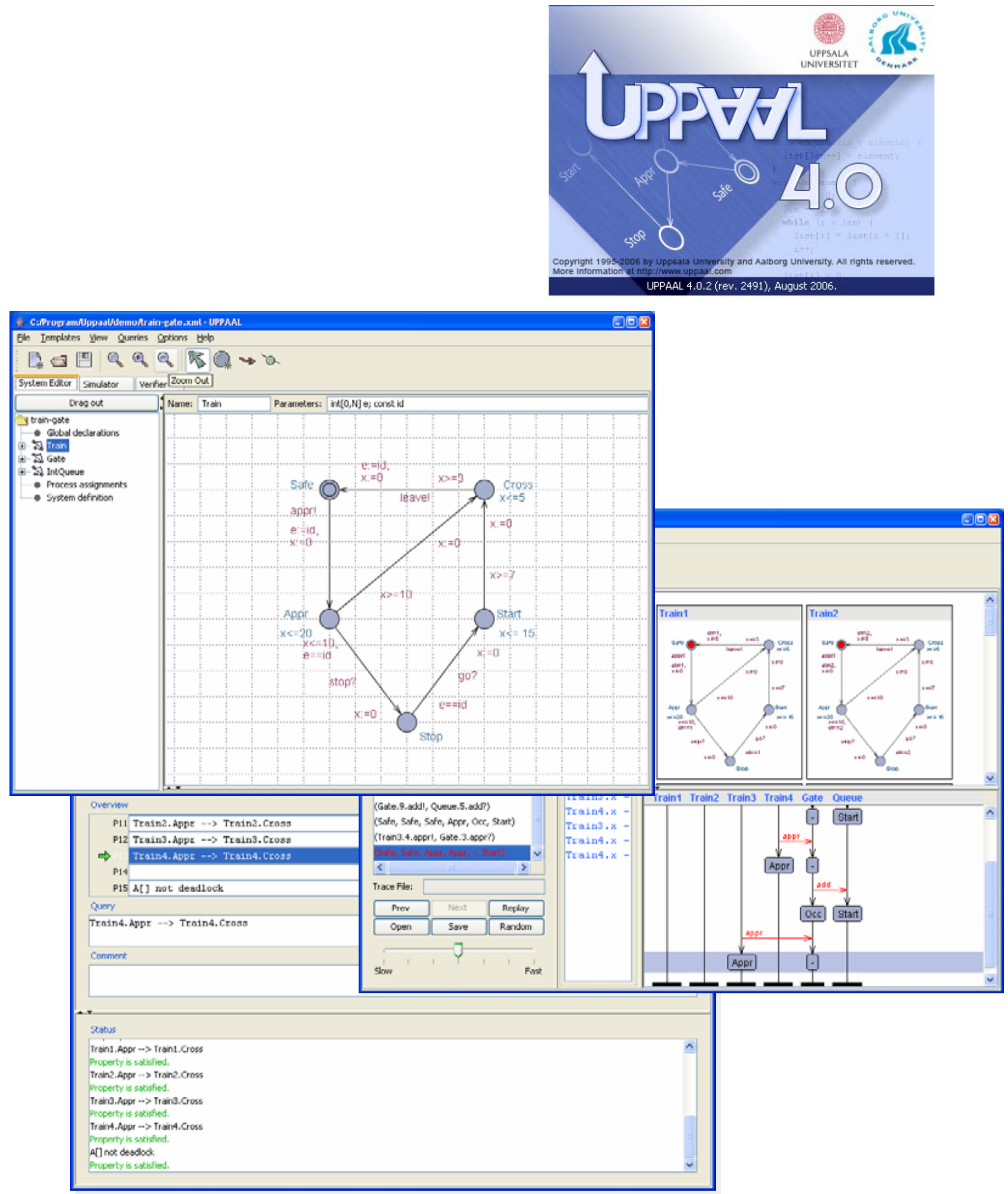
Mälardalarn (S)

Nijmegen, Twente, CWI (NL)

Upenn, Northumbria(US)

Braunschweig, Oldenborg, Marktoberdorf (D)

Tsinghua, Shanghai, ISS, NUS (Asia)

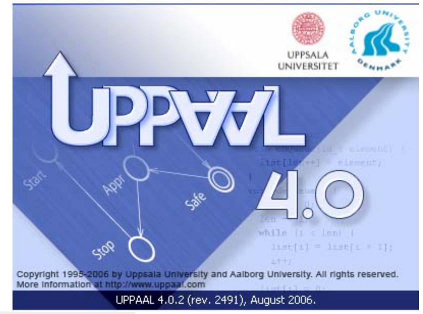


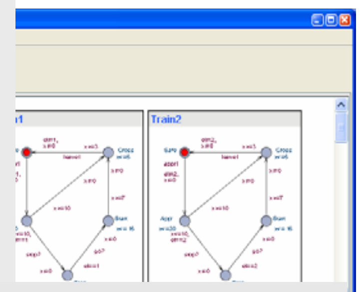
CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

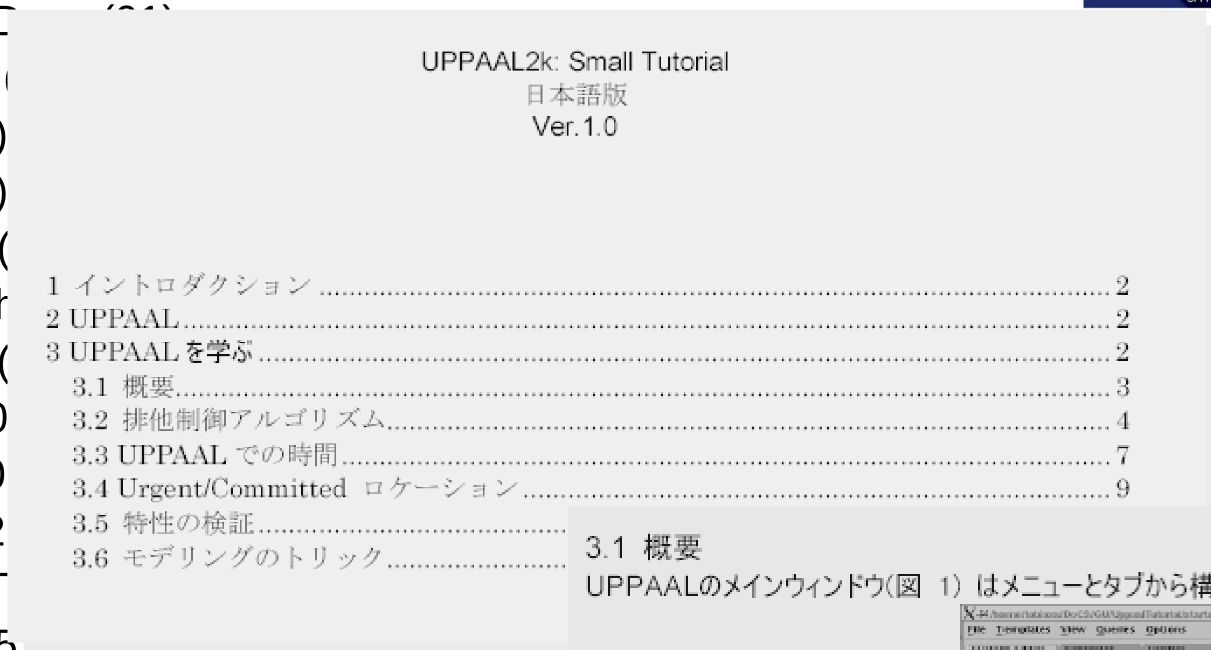
Impact

Tutorials Given @

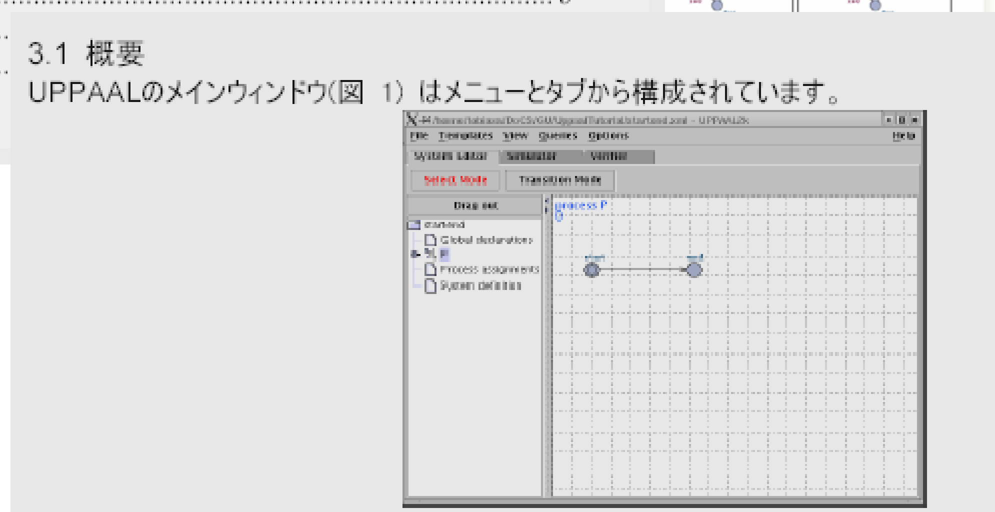
Estonian School (01)
IPA Fall D... (01)
FTRTFT (
CPN (02)
SFM (02)
MOVEP (
DISC Sch
MOVEP (
PRISE (0
PDMC (0
ARTIST2
EMSOFT
RTSS (05)
TECS week (06)
TAROT (06)
ARTS (06)
GLOBAN (06)
ARTIST ASIAN SCH (07)







UPPAAL2k: Small Tutorial
日本語版
Ver.1.0



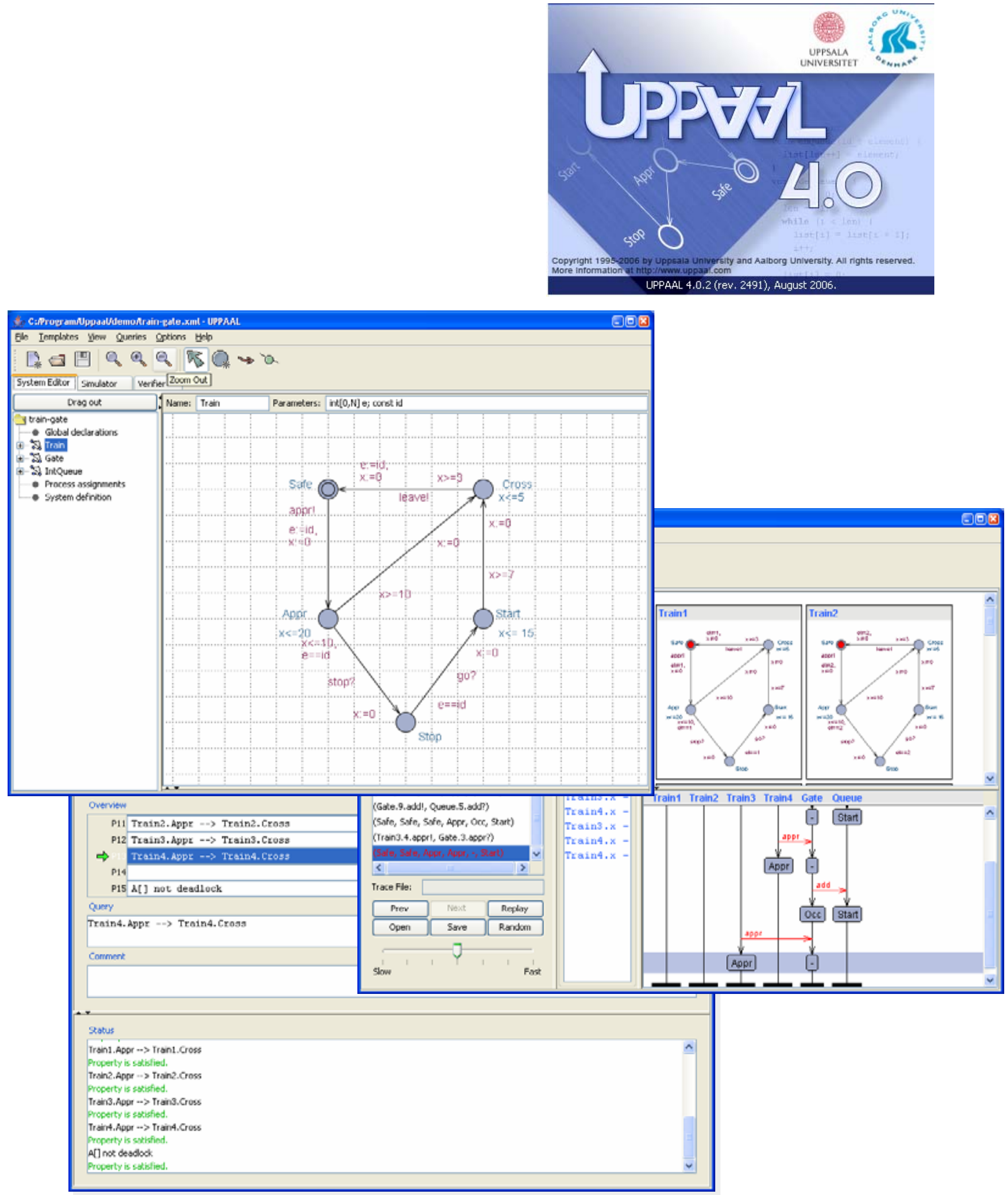
3.1 概要
UPPAALのメインウィンドウ(図 1) はメニューとタブから構成されています。

図 1: UPPAALの画面

Impact

Company Downloads

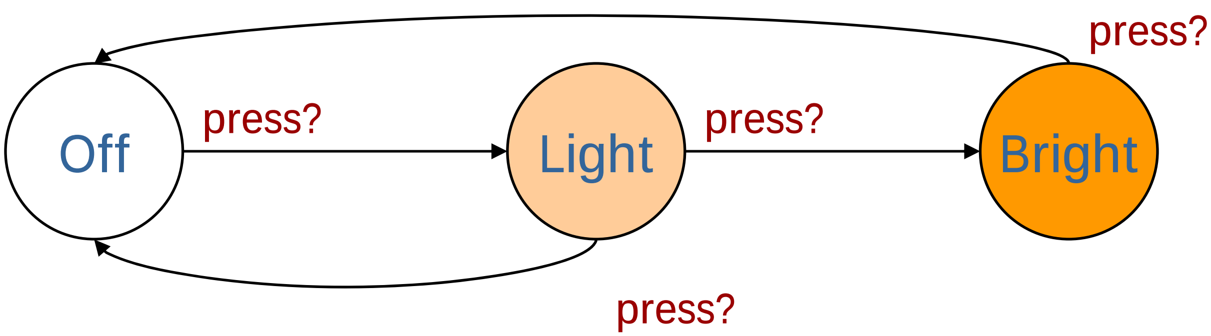
Mecel
Jet
Symantec
SRI
Relogic
Realwork
NASA
Verified Systems
Microsoft
ABB
Airbus
PSA
Saab
Siemens
Volvo
Lucent Technologies



Timed Automata

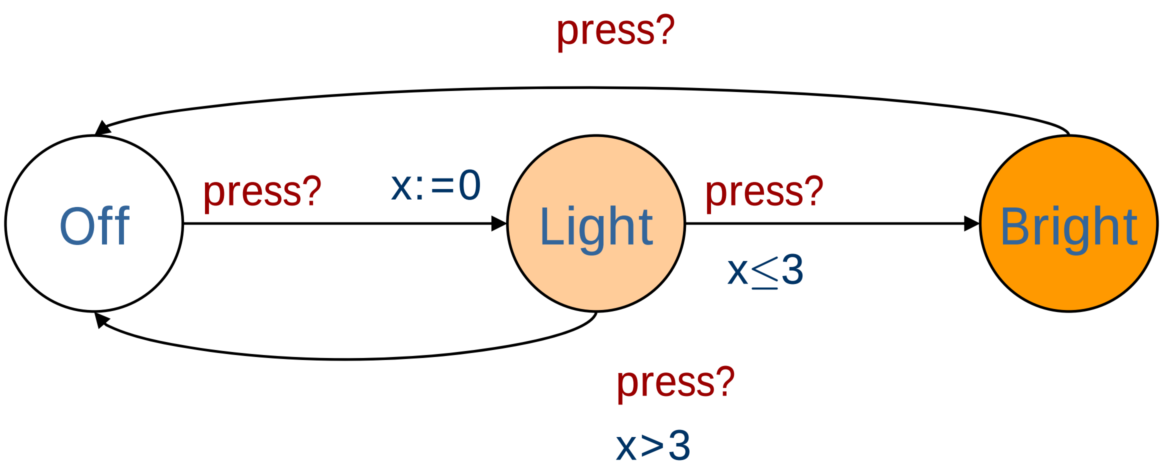
Alur & Dill 1989

Dumb Light Control



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

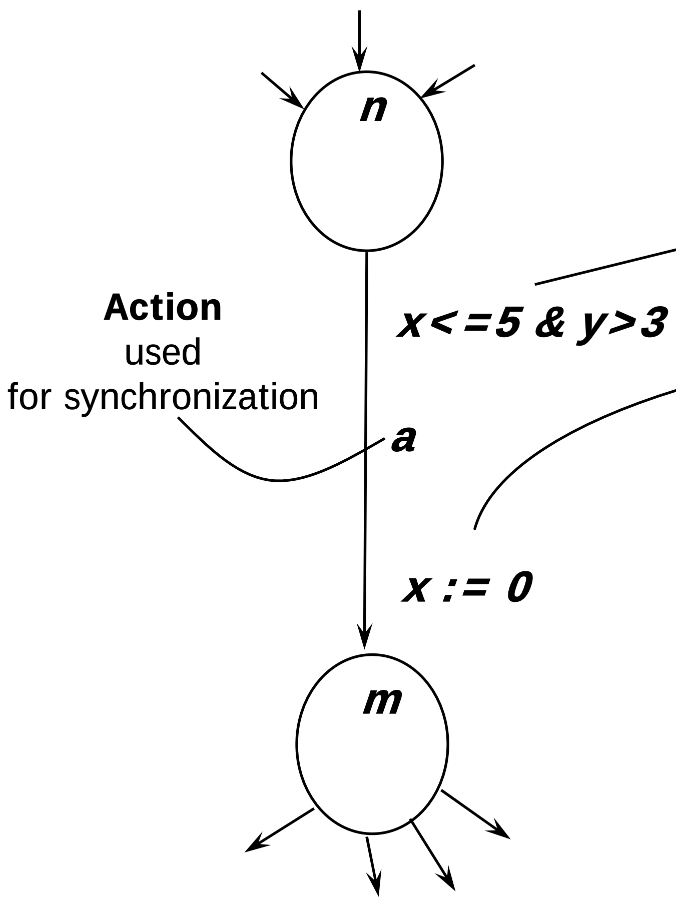
Dumb Light Control *Alur & Dill 1990*



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

Timed Automata *review*

Alur & Dill 1990



Clocks:

Guard
Boolean combination of **integer bounds** on **clocks**

Reset
Action performed on clocks

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

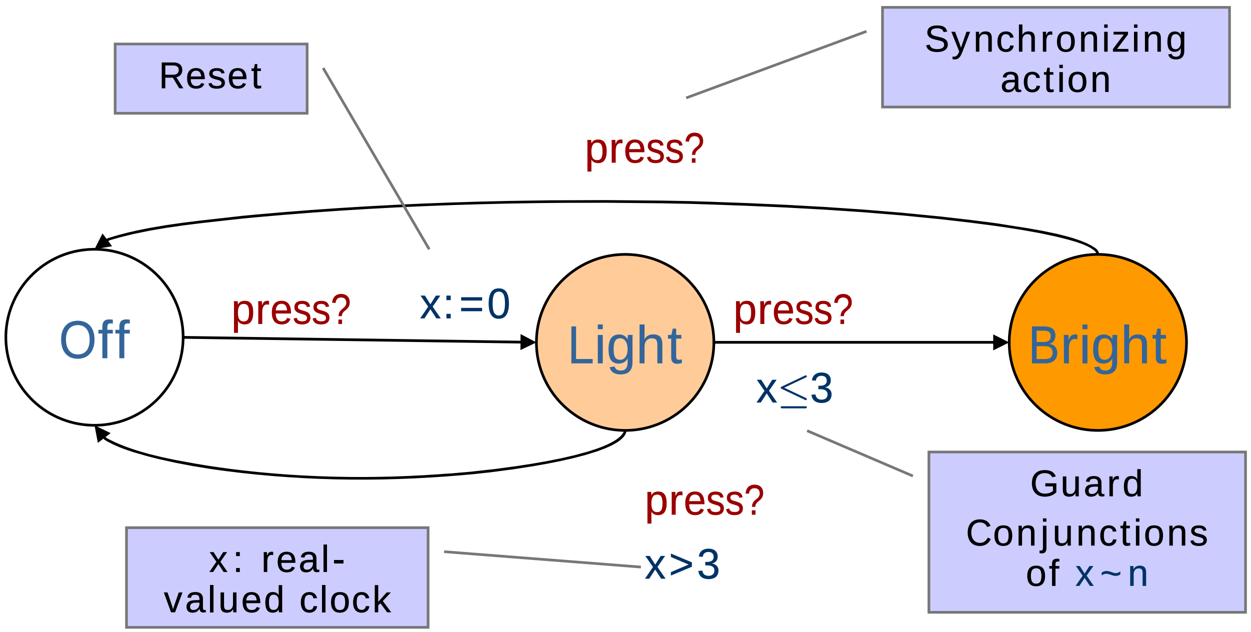
Transitions

Discrete Trans
(***n*** , ***x***=2.4 , ***y***=3.1415) $\xrightarrow{a}$ (***m*** , ***x***=0 , ***y***=3.1415)

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

Timed Automata

Alur & Dill 1990

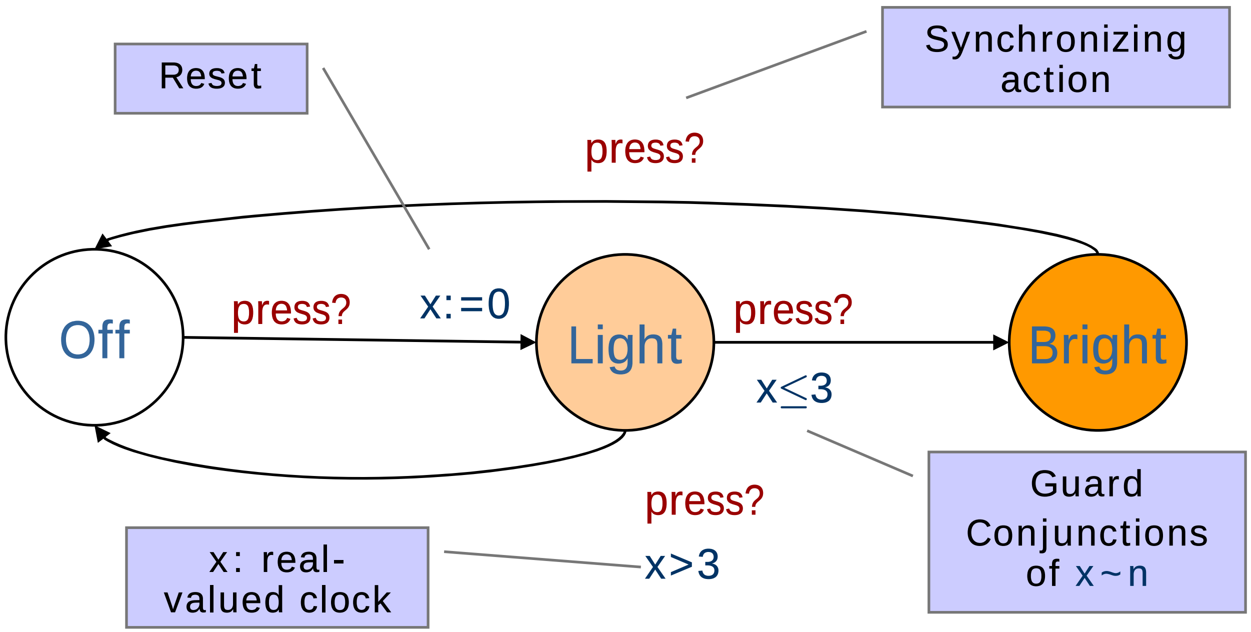


States:
(location , x=v) where v∈**R**

Transitions:
(Off , x=0)

Timed Automata

Alur & Dill 1990

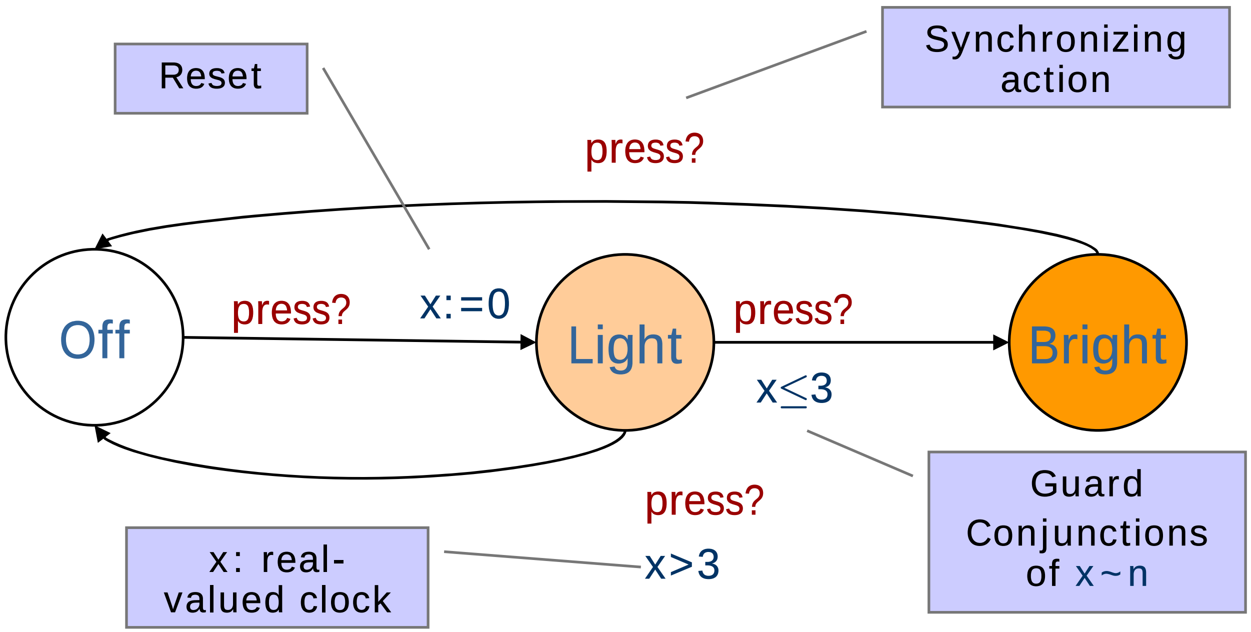


States:
(location , x=v) where $v \in \mathbf{R}$

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

Timed Automata

Alur & Dill 1990



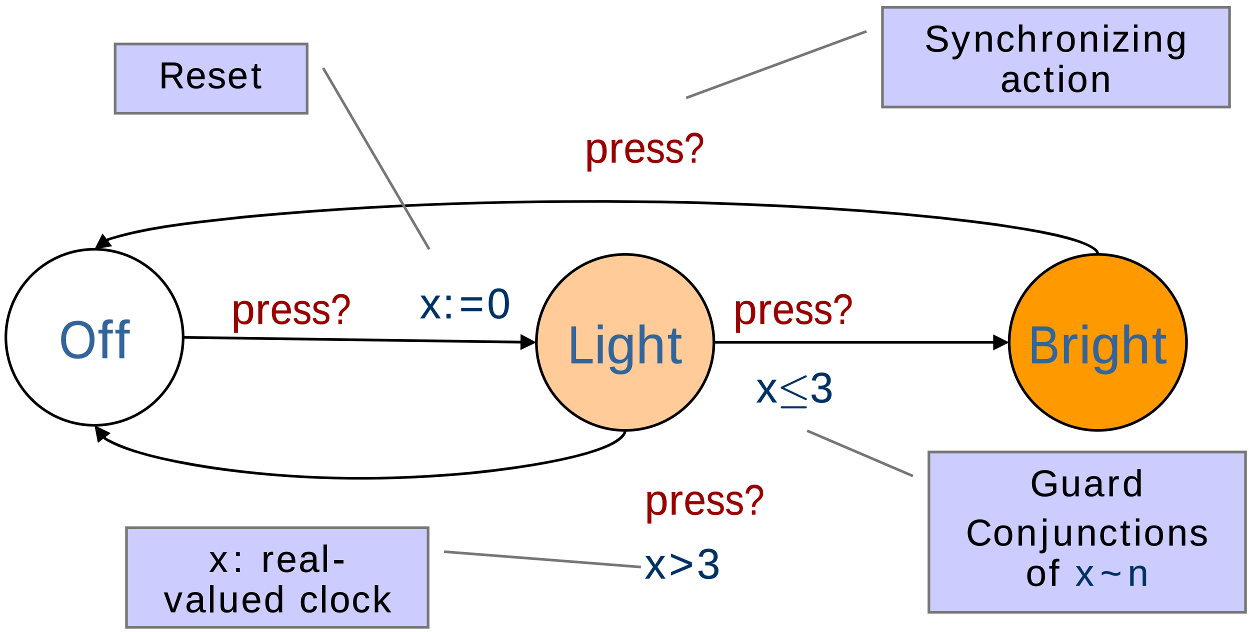
States:
(location , x=v) where v∈**R**

Transitions:

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

Timed Automata

Alur & Dill 1990



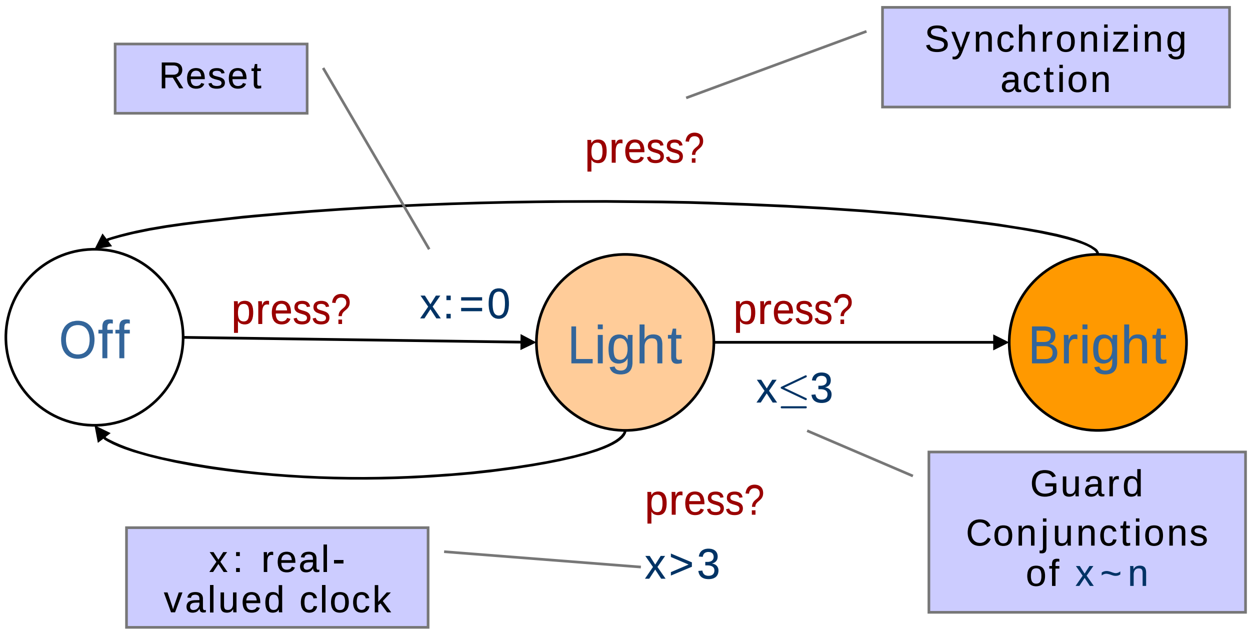
States:
(location , x=v) where $v \in \mathbf{R}$

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



States:

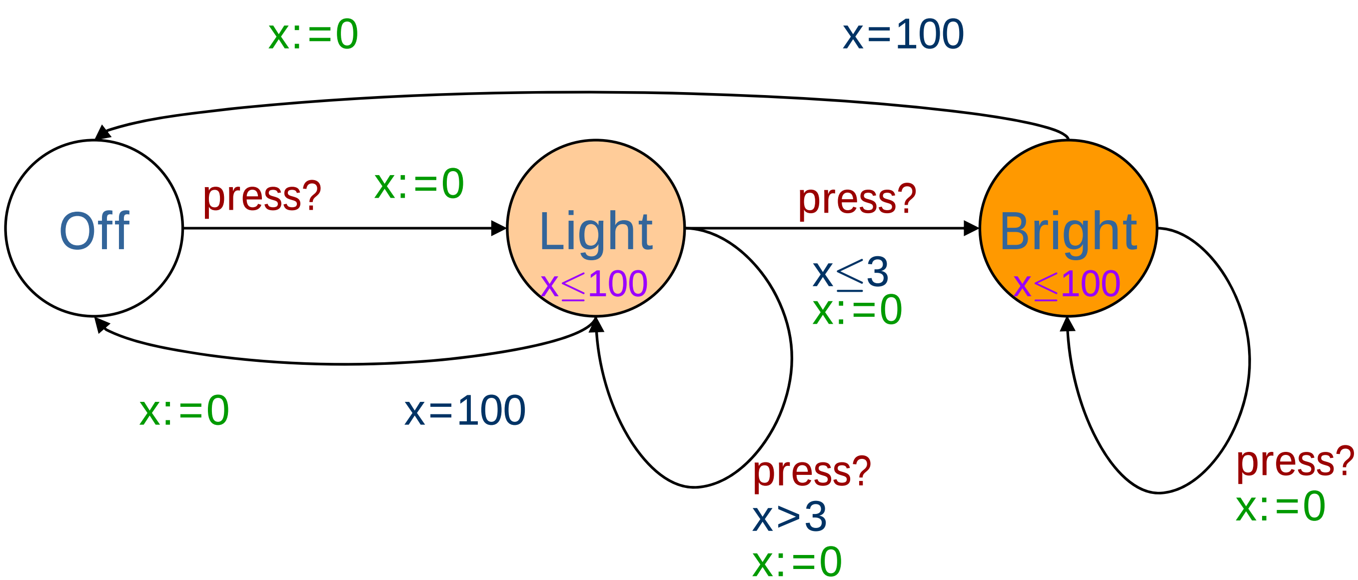
(location , x=v) where v∈**R**

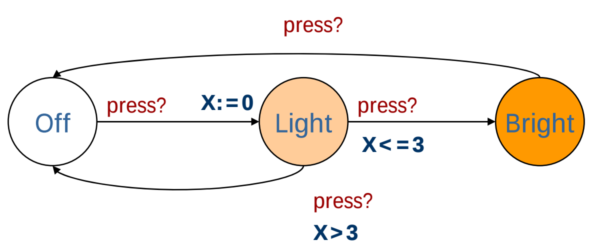
Transitions:

	(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

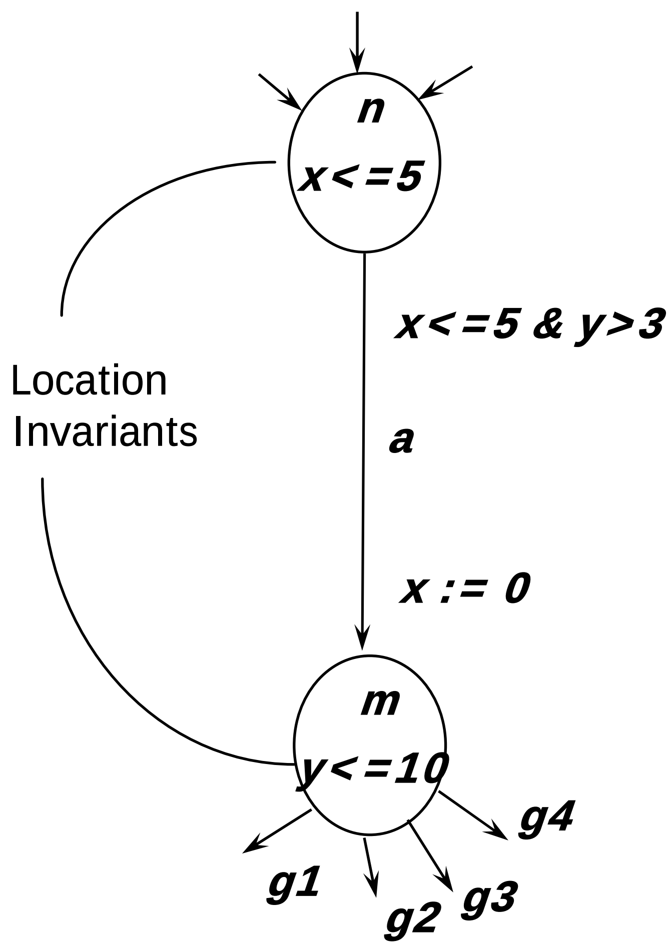
Using Invariants





Timed Automata *review*

Invariants



Clocks: *x, y*

Transitions

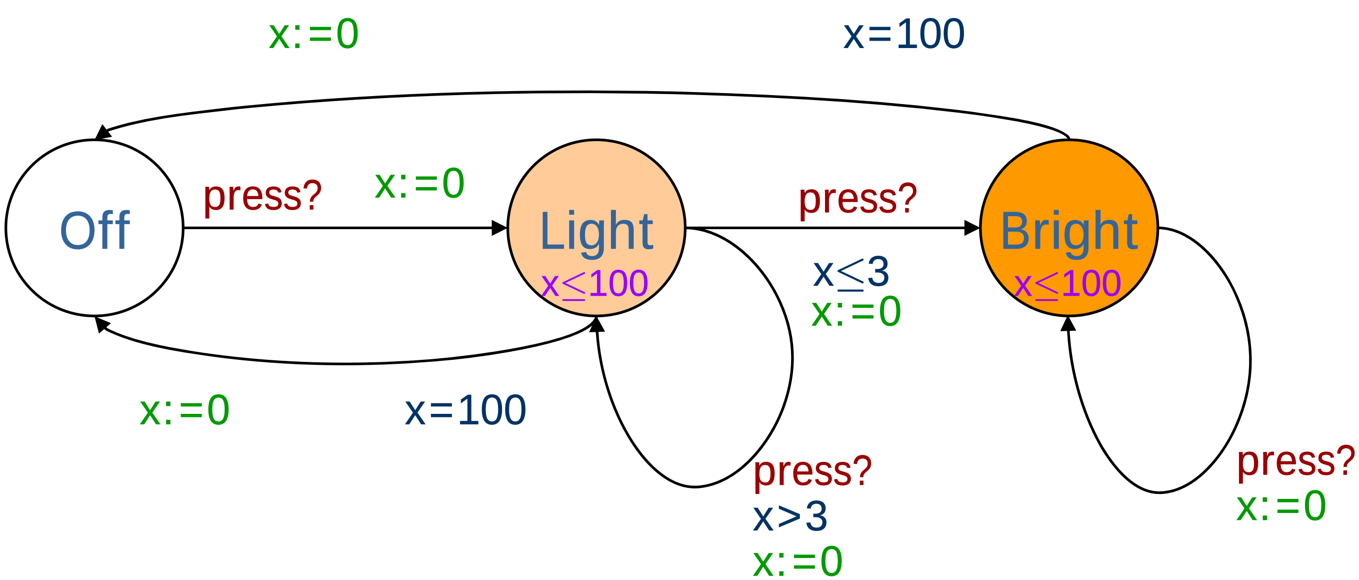
(***n*** , ***x***=2.4 , ***y***=3.1415) $\xrightarrow{\cancel{e(3.2)}}$

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

Invariants ensure progress!!

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)

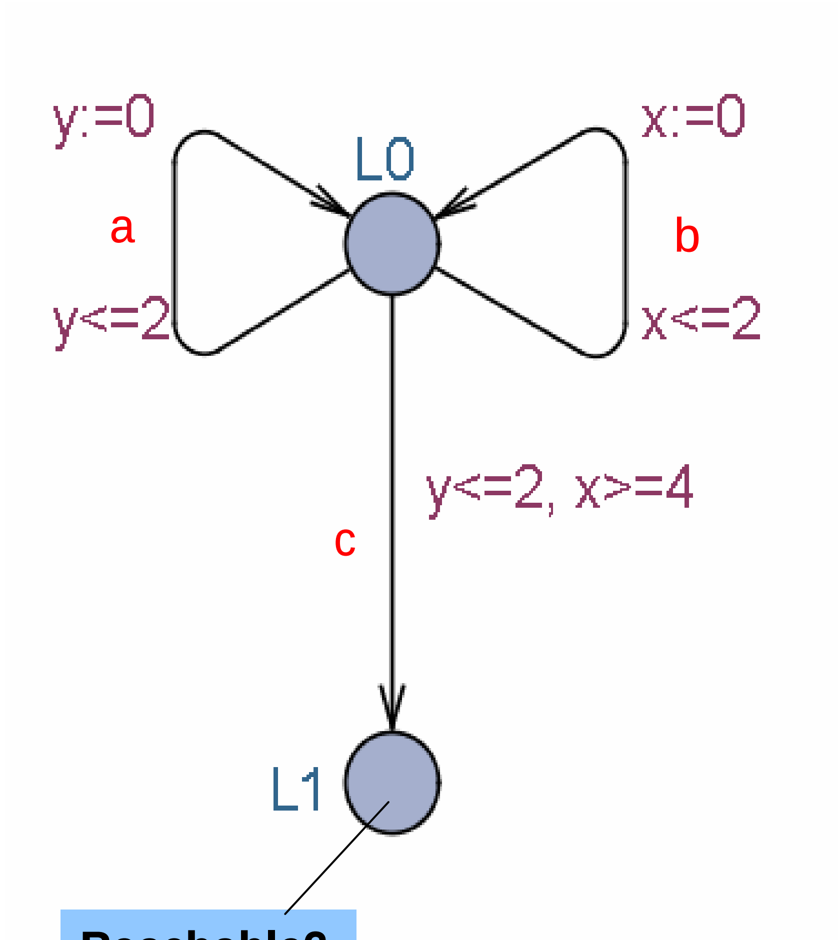
Note: X

(Light , x=0) delay 103 →

Invariants ensures progress

Example

With two clocks

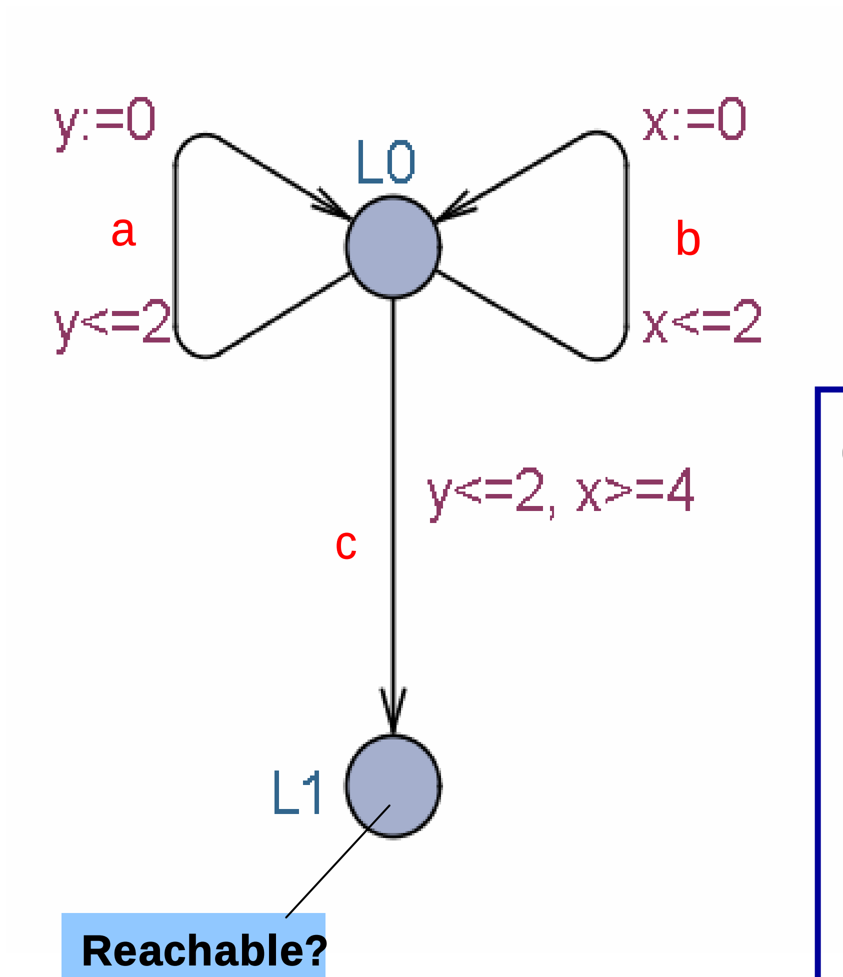


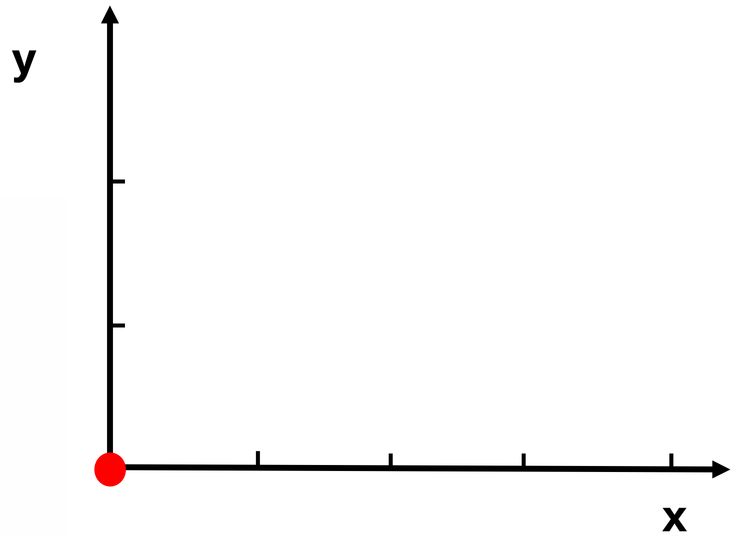
Reachable?

BRICS Basic Research in Computer Science

Example

With two clocks

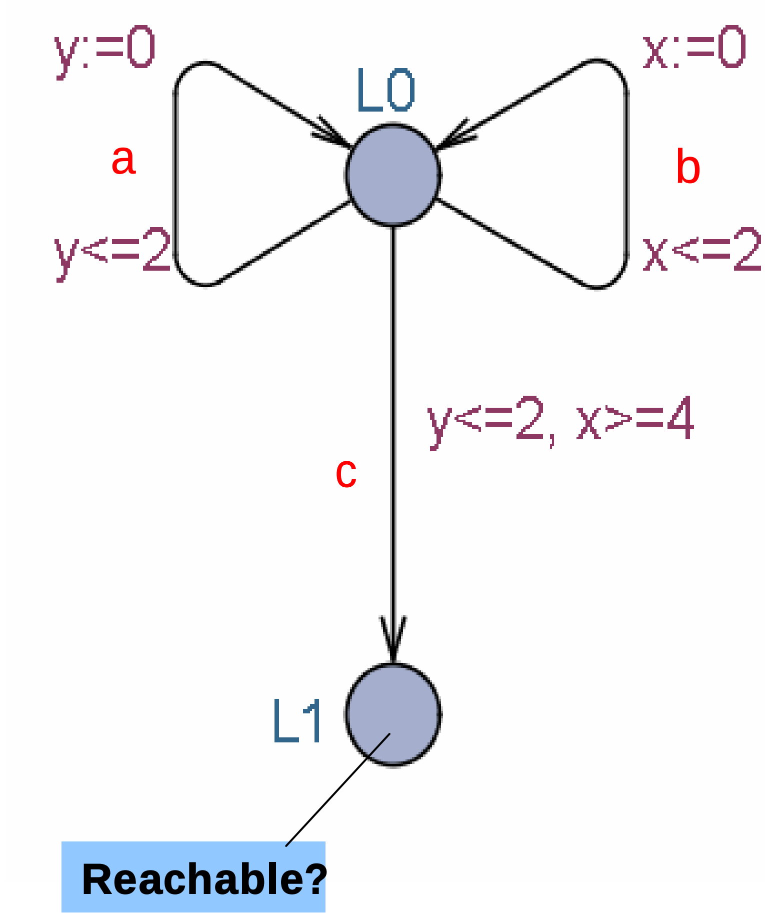


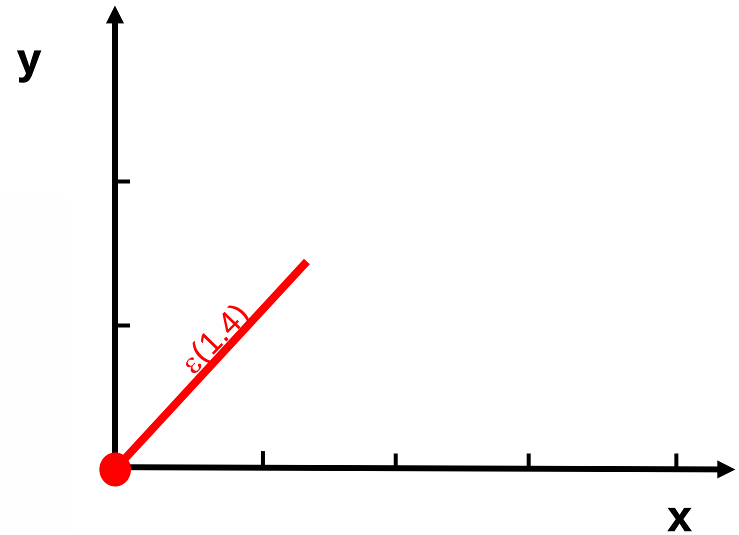


(L0,x=0,y=0)

Example

With two clocks





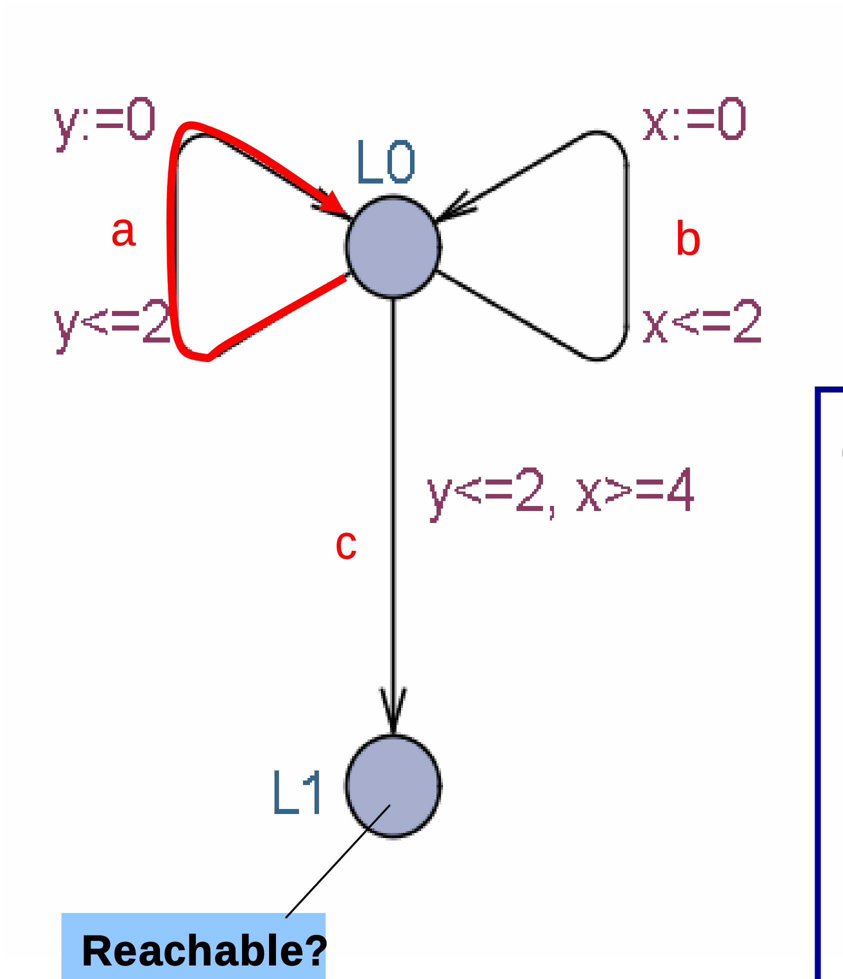
(L0,x=0,y=0)

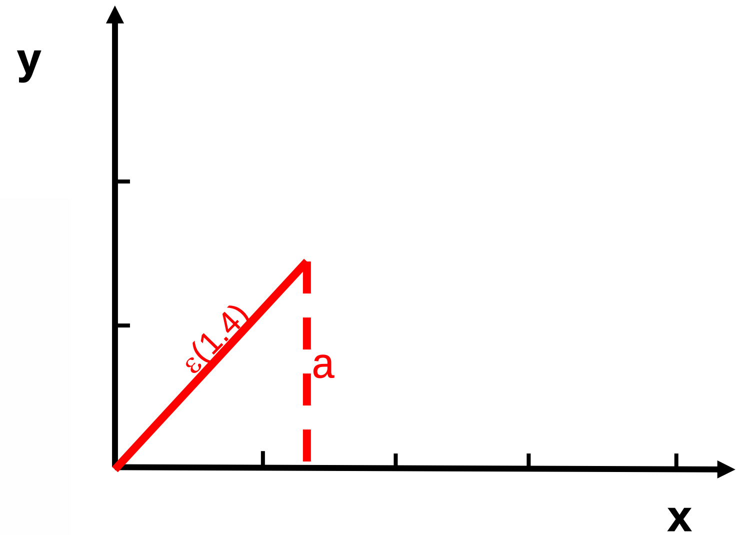
$\rightarrow_{\varepsilon(1.4)}$

(L0,x=1.4,y=1.4)

Example

With two clocks





(L0,x=0,y=0)

$\rightarrow_{\varepsilon(1.4)}$

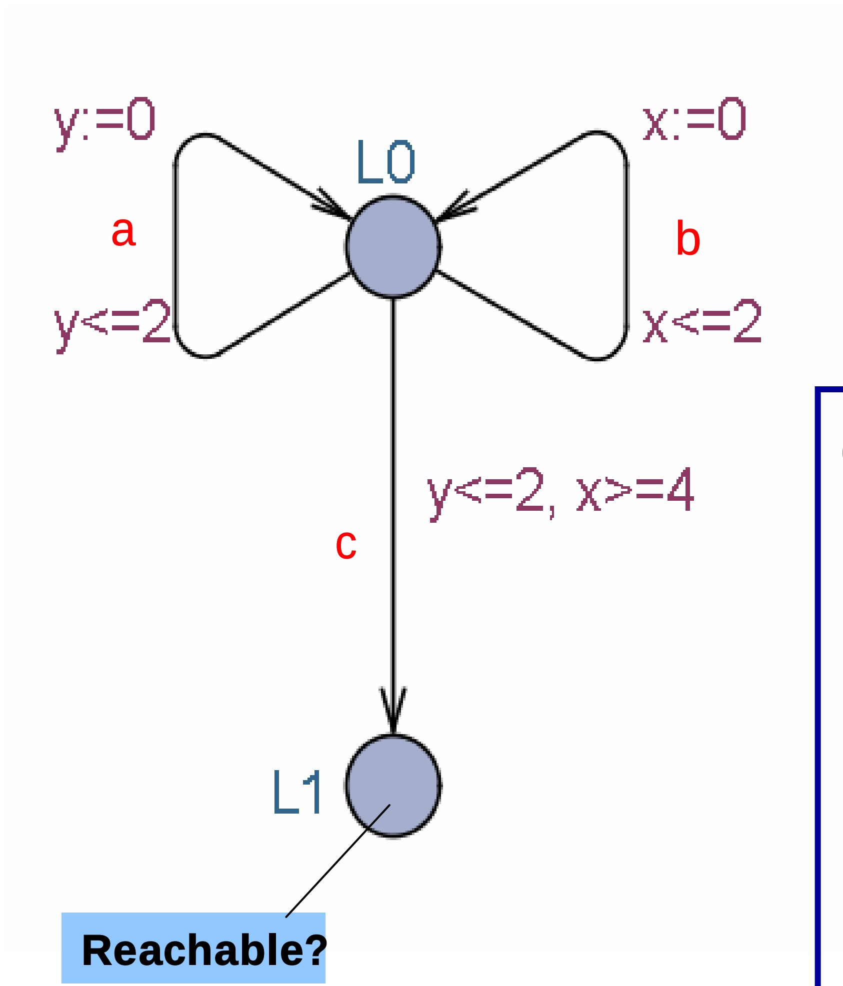
(L0,x=1.4,y=1.4)

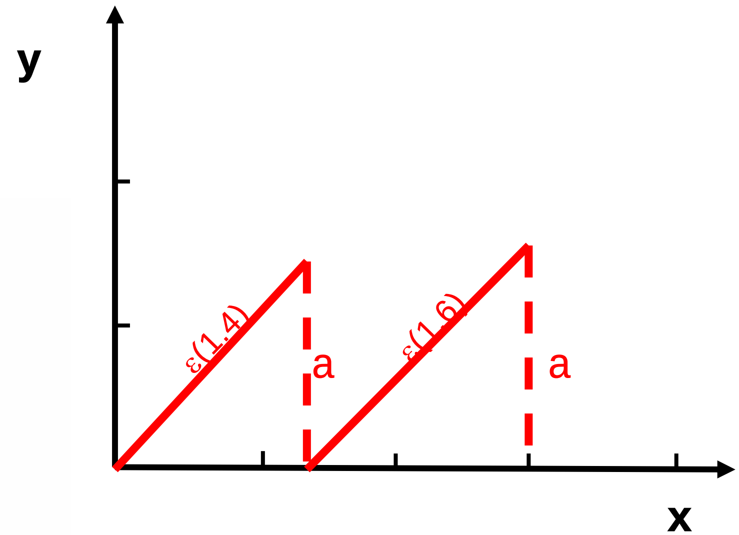
$\rightarrow_{a}$

(L0,x=1.4,y=0)

Example

With two clocks





(L0,x=0,y=0)

$\rightarrow_{\varepsilon(1.4)}$

(L0,x=1.4,y=1.4)

$\rightarrow_{a}$

(L0,x=1.4,y=0)

$\rightarrow_{\varepsilon(1.6)}$

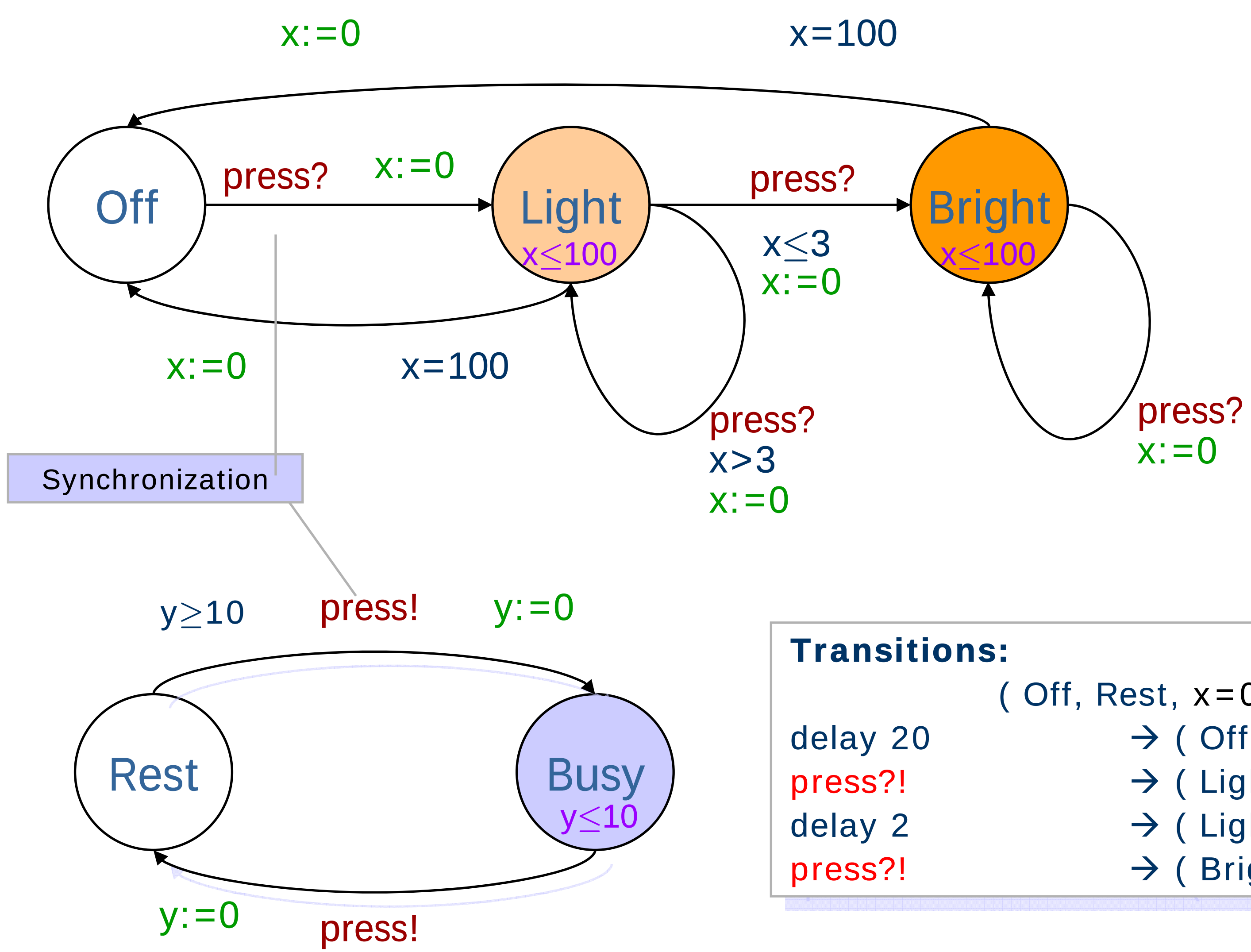
(L0,x=3.0,y=1.6)

$\rightarrow_{a}$

(L0,x=3.0,y=0)

BRICS Basic Research in Computer Science

Networks Light Controller & User



Transitions:

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

delay 20 → (Off, Rest, x=20, y=20)

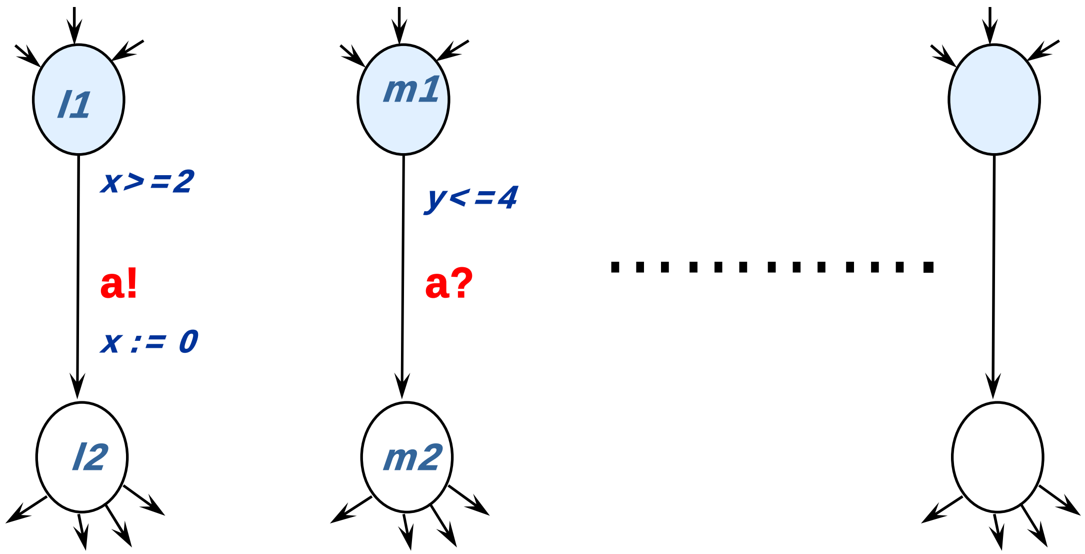
press?! → (Light, Busy, x=0, y=0)

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, …..)*

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

If **a** URGENT CHANNEL

BRICS Basic Research in Computer Science

Timed Automata
Formally

AALBORG UNIVERSITY
DENMARK

BRICS
Basic Research
in Computer Science

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Constraints

Definition

Let X be a set of clock variables. The set $\mathcal{B}(X)$ of *clock constraints* ϕ is given by the grammar:

$$\phi \ ::= \ x \leq c \mid c \leq x \mid x < c \mid c < x \mid \phi_1 \wedge \phi_2$$

where $c \in \mathbb{N}$ (or $\mathbb{Q}$).

Clock Valuations and Notation

Definition

The set of *clock valuations*, $\mathbb{R}^C$ is the set of functions $C \longrightarrow \mathbb{R}_{\geq 0}$ ranged over by $u, v, w, \ldots$.

Notation

Let $u \in \mathbb{R}^C$, $r \subseteq C$, $d \in \mathbb{R}_{\geq 0}$, and $g \in \mathcal{B}(X)$ then:

- $u + d \in \mathbb{R}^C$ is defined by $(u + d)(x) = u(x) + d$ for any clock x
- $u[r] \in \mathbb{R}^C$ is defined by $u[r](x) = 0$ when $x \in r$ and $u[r](x) = u(x)$ for $x \notin r$.
- $u \models g$ denotes that g is satisfied by u.

Timed Automata

Definition

A timed automaton A over clocks C and actions Act is a tuple (L, l_0, E, I), where:

- L is a finite set of locations
- $l_0 \in L$ is the initial location
- $E \subseteq L \times \mathcal{B}(X) \times Act \times \mathcal{P}(C) \times L$ is the set of edges
- $I : L \longrightarrow \mathcal{B}(X)$ assigns to each location an invariant

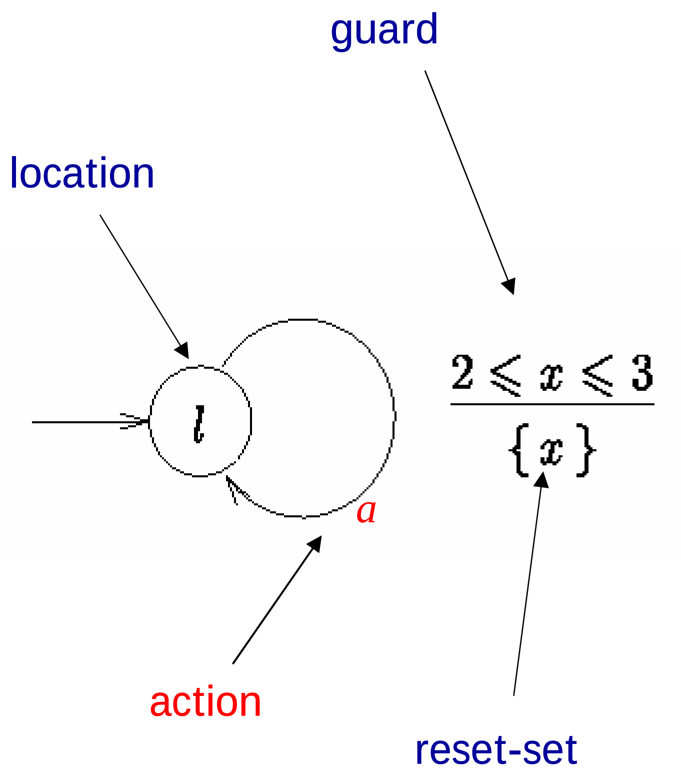
Semantics

Definition
The semantics of a timed automaton A is a labelled transition system with state space $L \times \mathbb{R}^C$ with initial state (l_0, u_0)* and with the following transitions:

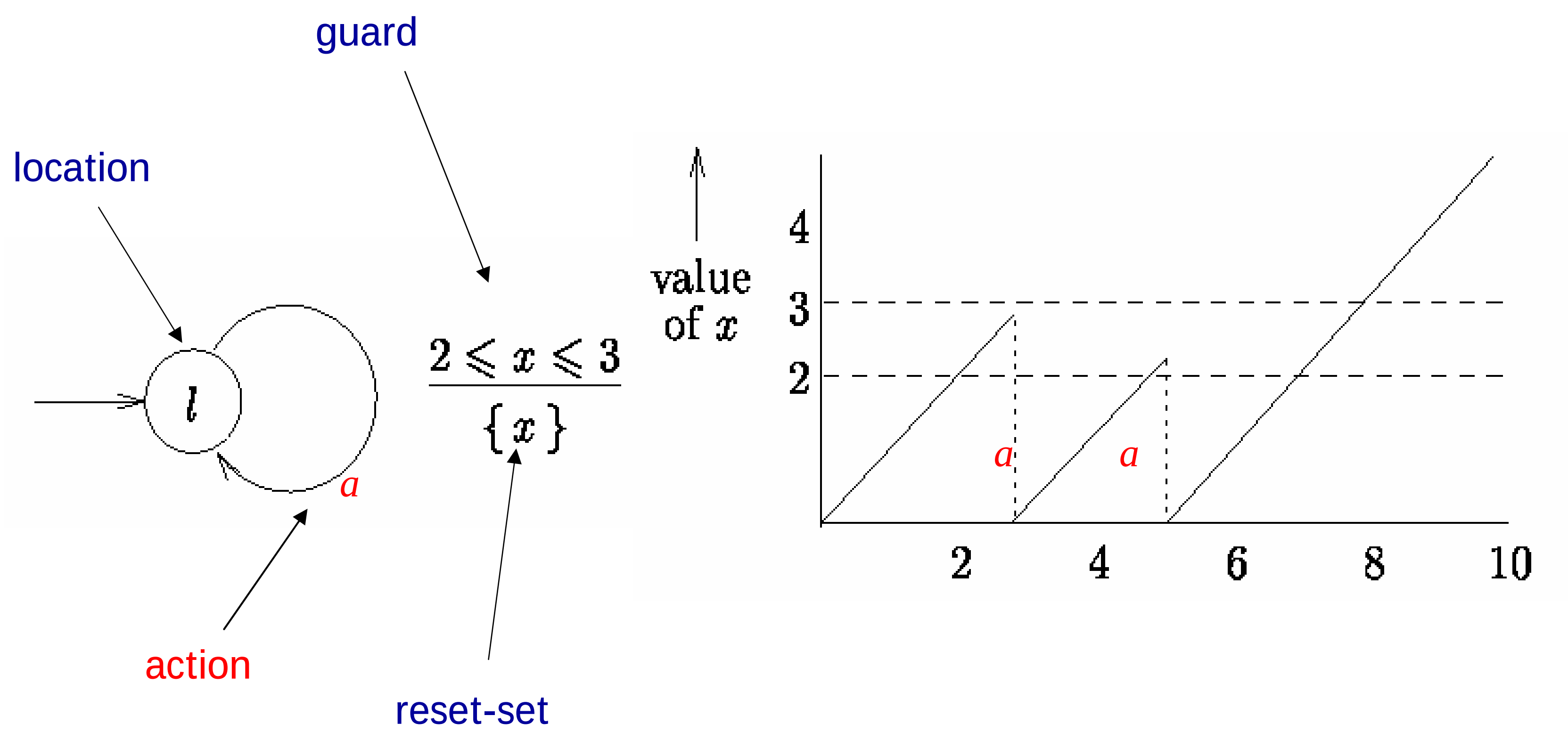
- $(l, u) \xrightarrow{\epsilon(d)} (l, u + d)$ iff $u \in I(l)$ and $u + d \in I(l)$,
- $(l, u) \xrightarrow{a} (l', u')$ iff there exists $(l, g, a, r, l') \in E$ such that
  - $u \models g$,
  - $u' = u[r]$, and
  - $u' \in I(l')$

*$u_0(x) = 0$ for all $x \in C$

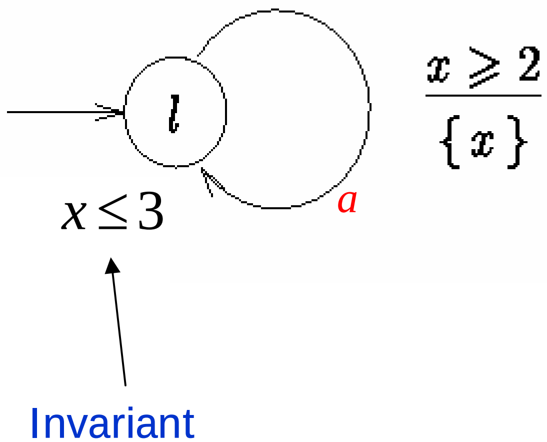
Timed Automata: Example



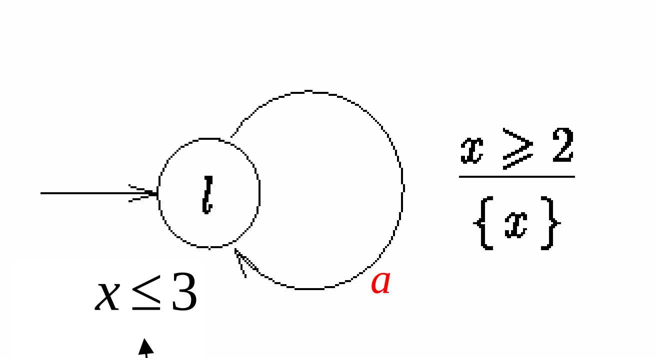
Timed Automata: Example



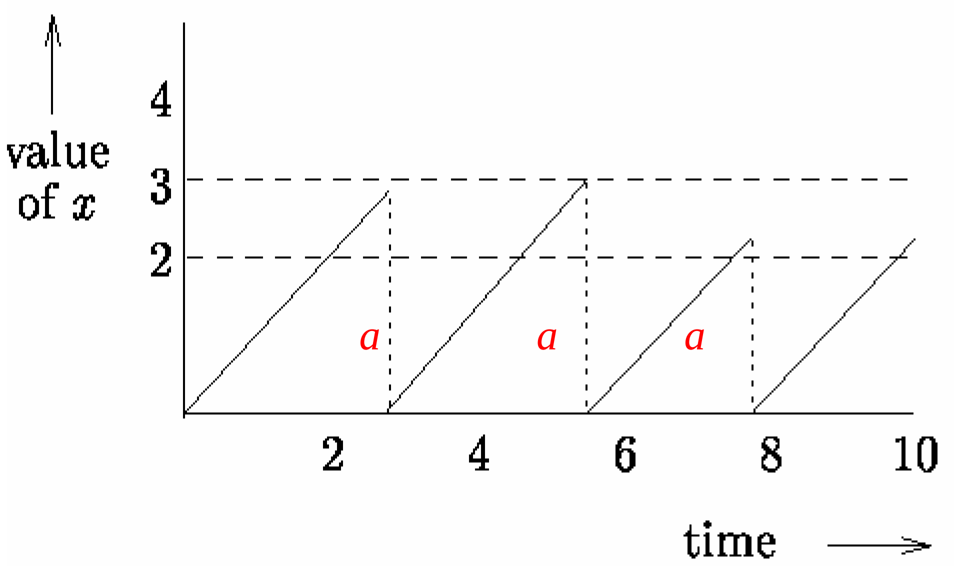
Timed Automata: Example



Timed Automata: Example



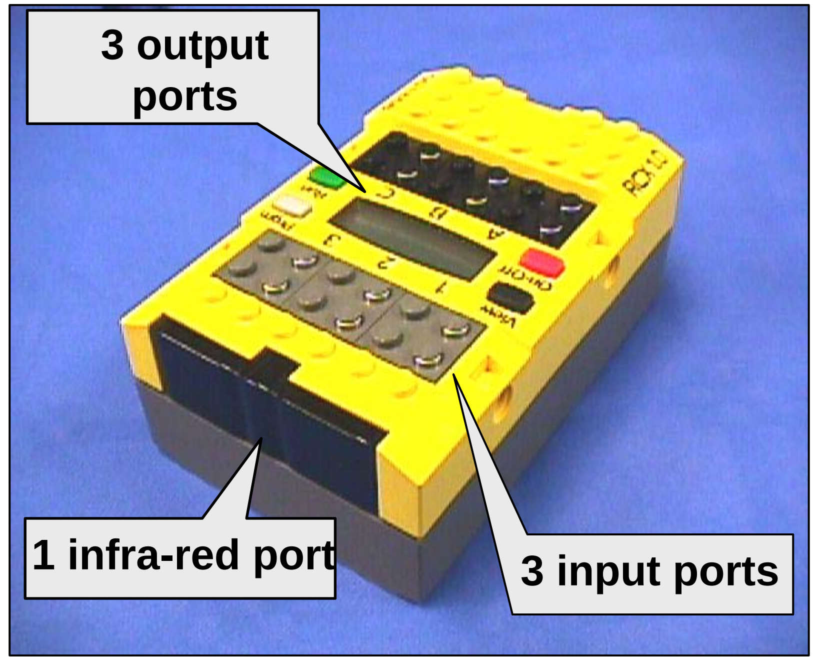
Invariant



Brick Sorting

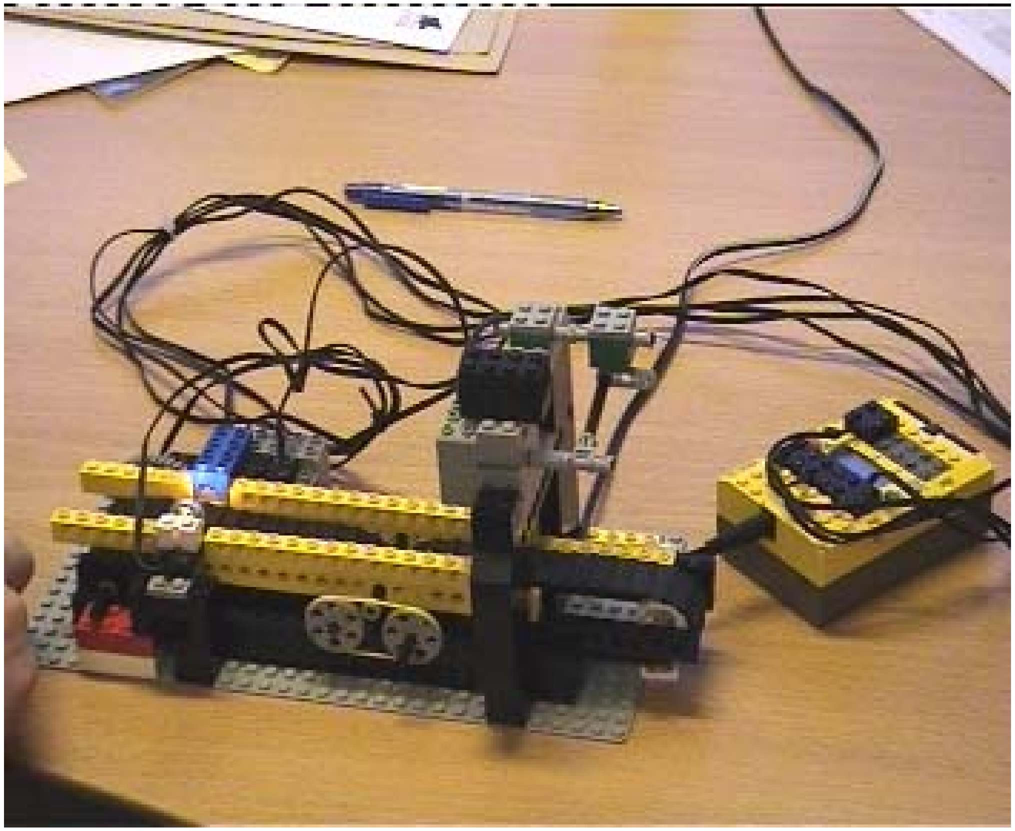
LEGO Mindstorms/RCX

- 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 Real Timed System

The Plant
Conveyor Belt
&
Bricks

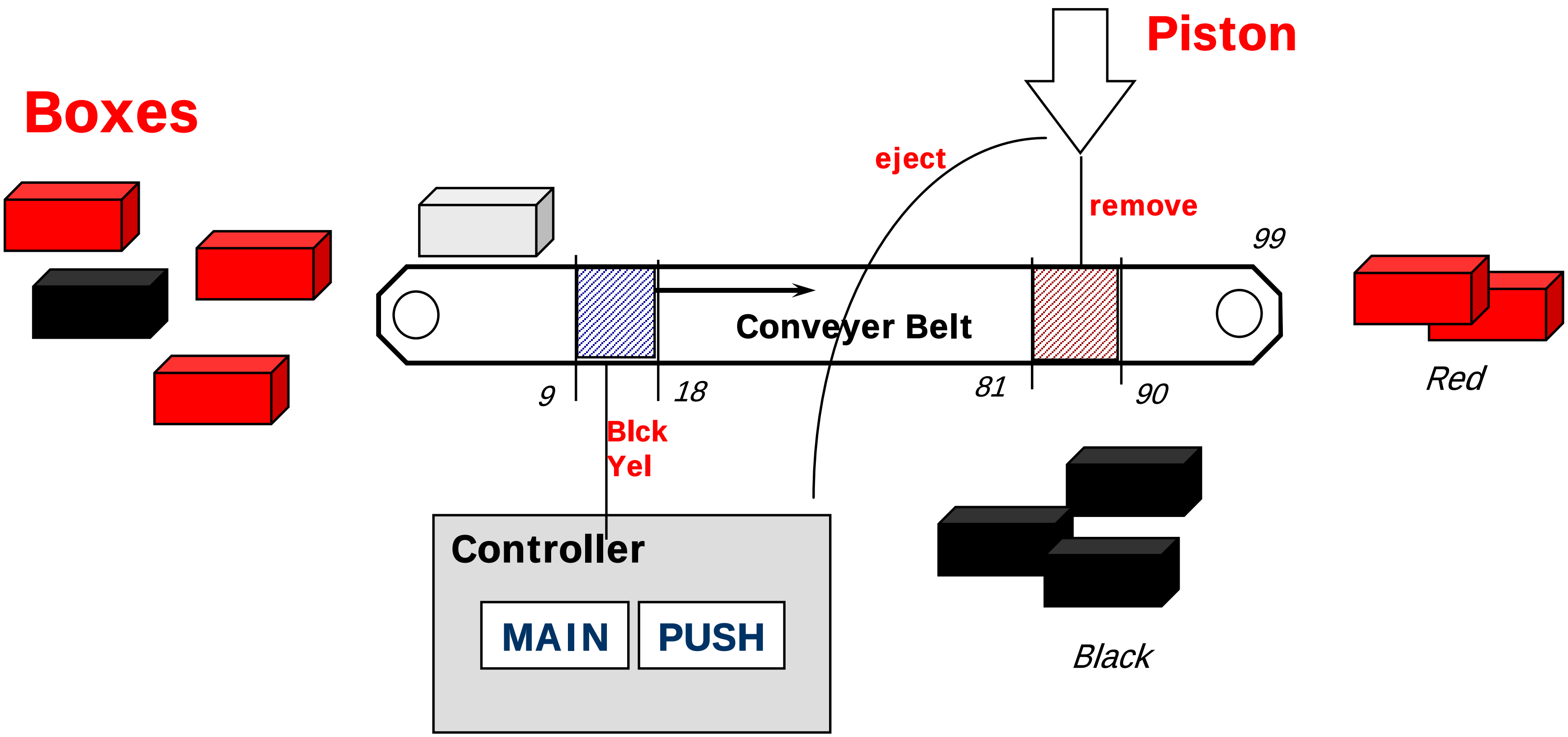


Controller Program
LEGO MINDSTORM

First UPPAAL model

Sorting of Lego Boxes

Ken Tindell



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

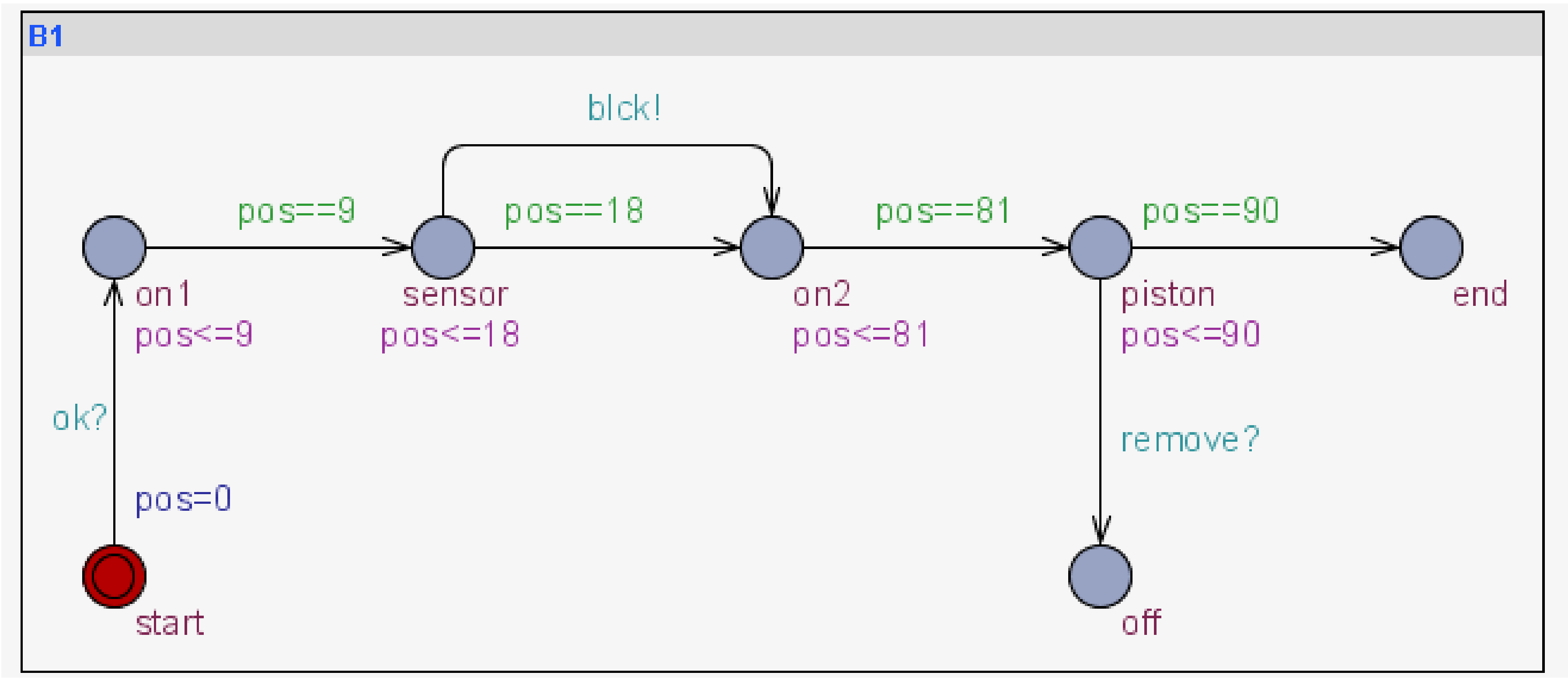
NQC programs

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

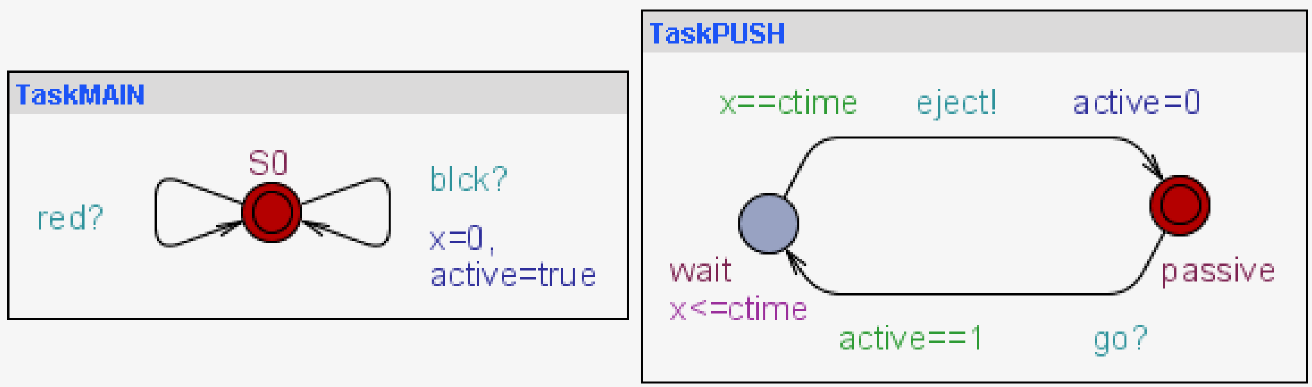
```
task MAIN{
 DELAY=75;
 LIGHT_LEVEL=35;
 active=0;
 Sensor(IN_1, IN_LIGHT);
 Fwd(OUT_A,1);
 Display(1);

 start PUSH;

 while(true){

wait(IN_1<=LIGHT_LEVEL);
   ClearTimer(1);
   active=1;
   PlaySound(1);

wait(IN_1>LIGHT_LEVEL);
  }
}
```

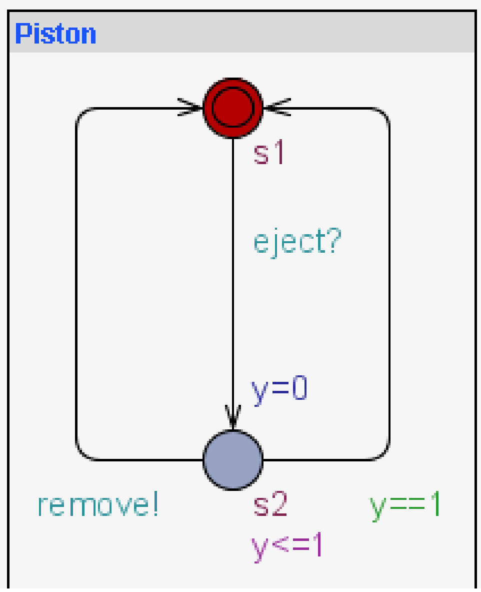
```
task PUSH{
  while(true){
    wait(Timer(1)>DELAY && active==1);
    active=0;
    Rev(OUT_C,1);
    Sleep(8);
    Fwd(OUT_C,1);
    Sleep(12);
    Off(OUT_C);
  }
}
```

A Black Brick



Control Tasks & Piston





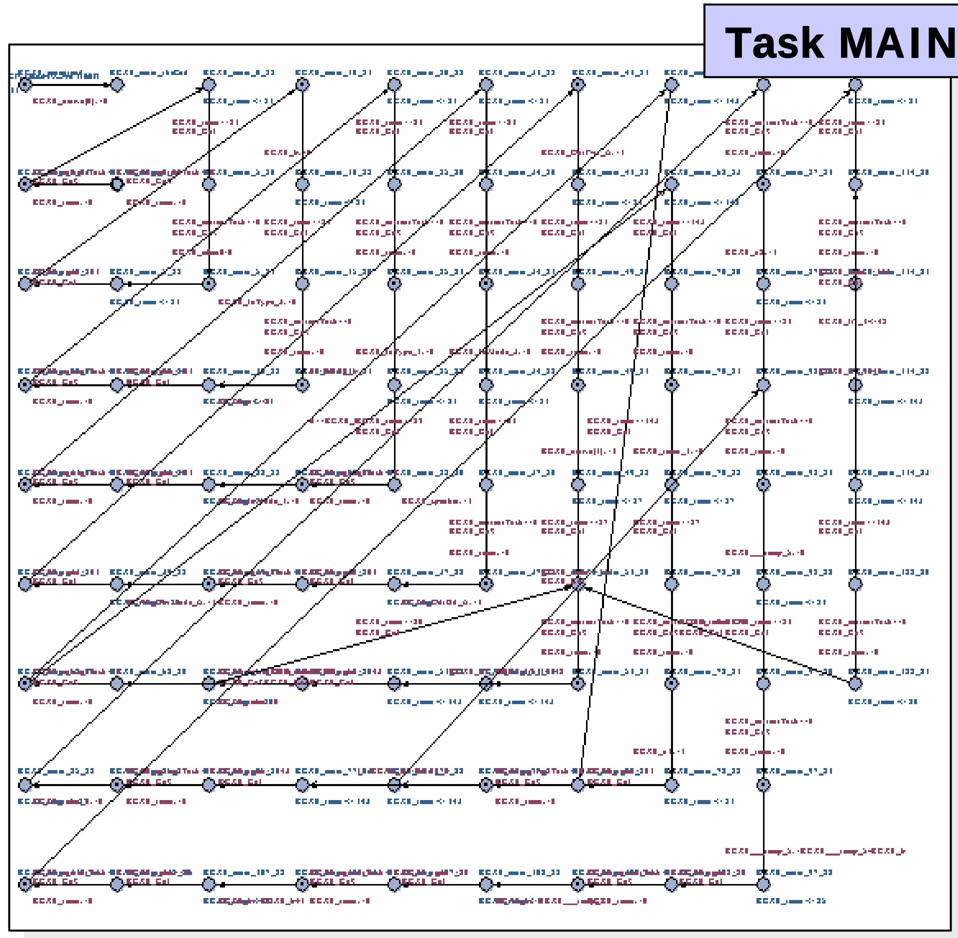
GLOBAL DECLARATIONS:

```
const int ctime = 75;

int[0,1] active;
clock x, time;

chan   eject, ok;
urgent chan blck, red, remove, go;
```

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

From RCX to UPPAAL – and back

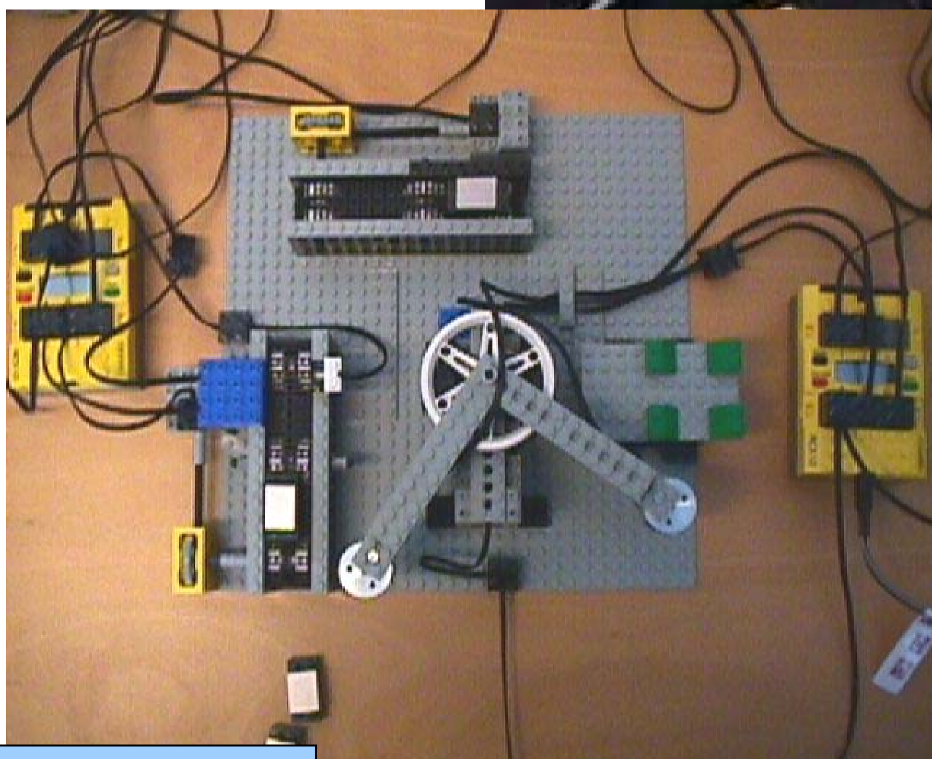
- Model includes Round-Robin Scheduler.
- Compilation of RCX tasks into TA models.
- Presented at ECRTS 2000 in Stockholm.
- From UPPAAL to RCX: Martijn Hendriks.

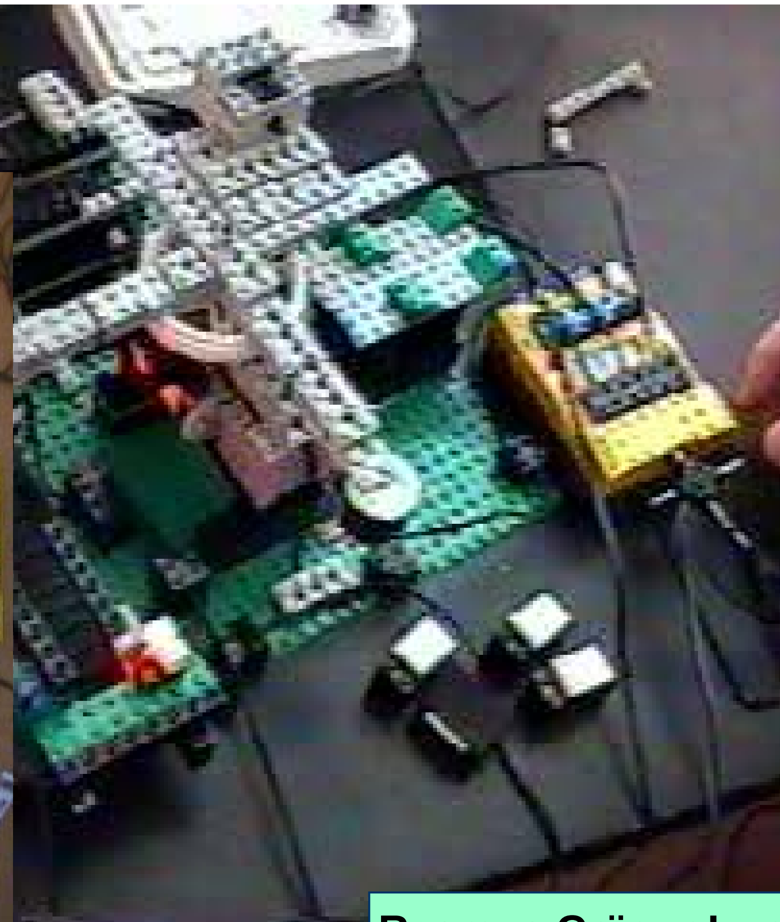


BRICS
Basic Research
in Computer Science

The Production Cell in LEGO

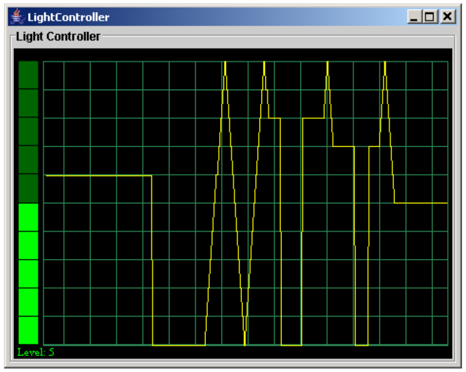
Course at DTU, Copenhagen





Production Cell

Rasmus Crüger Lund
Simon Tune Riemanni



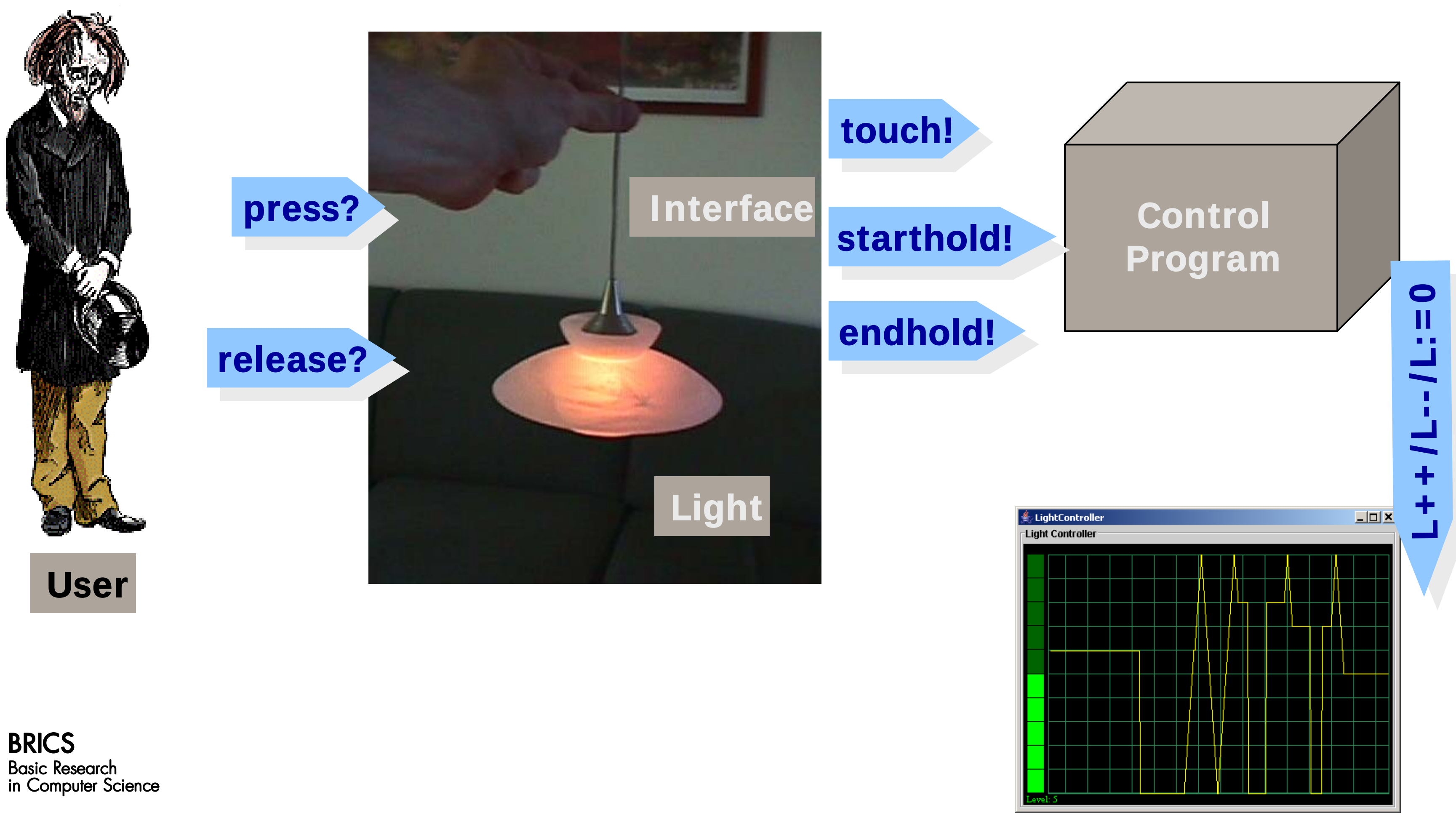
Light Control Interface

AALBORG UNIVERSITY DENMARK

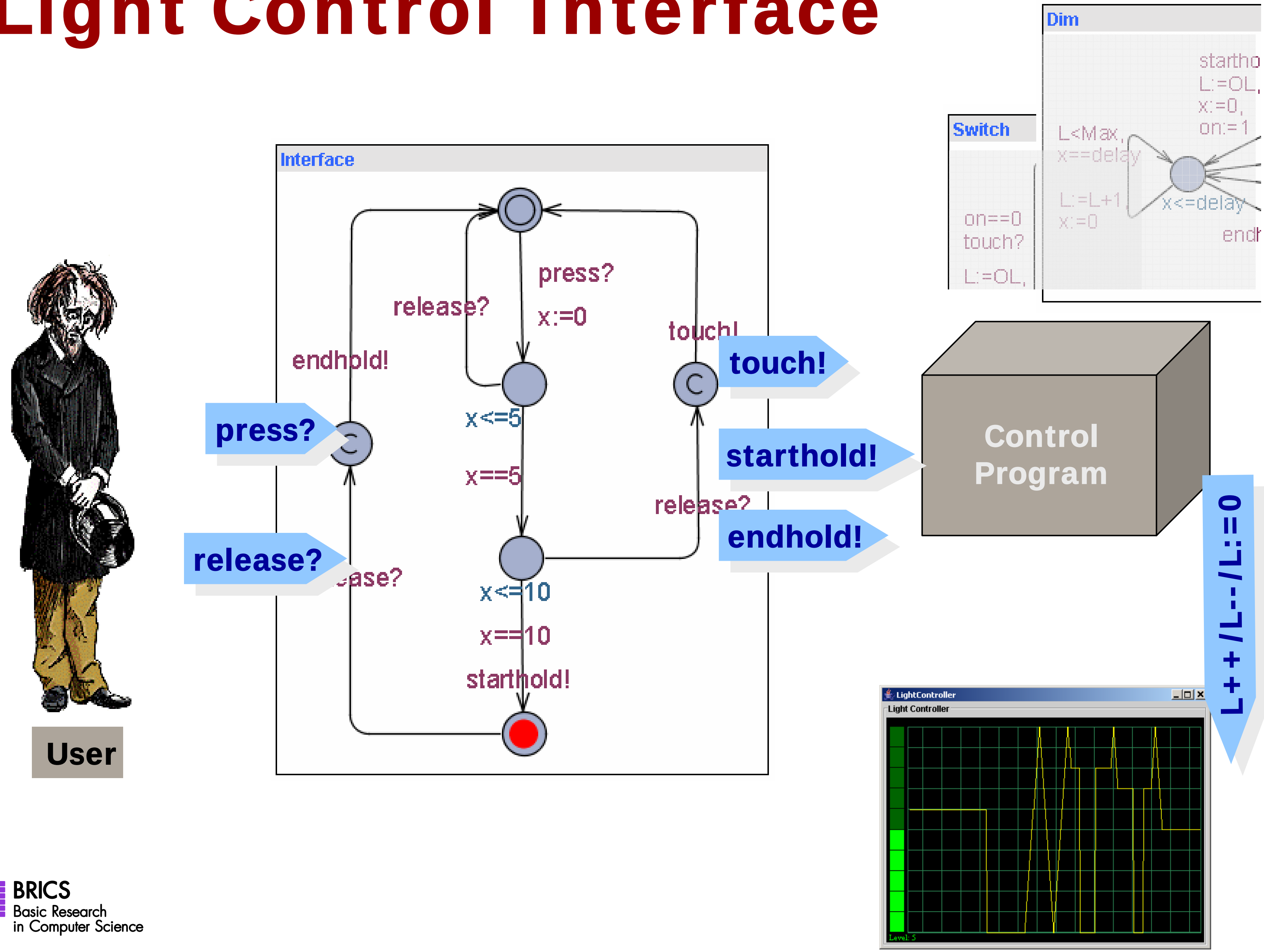
BRICS
Basic Research
in Computer Science

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Light Control Interface



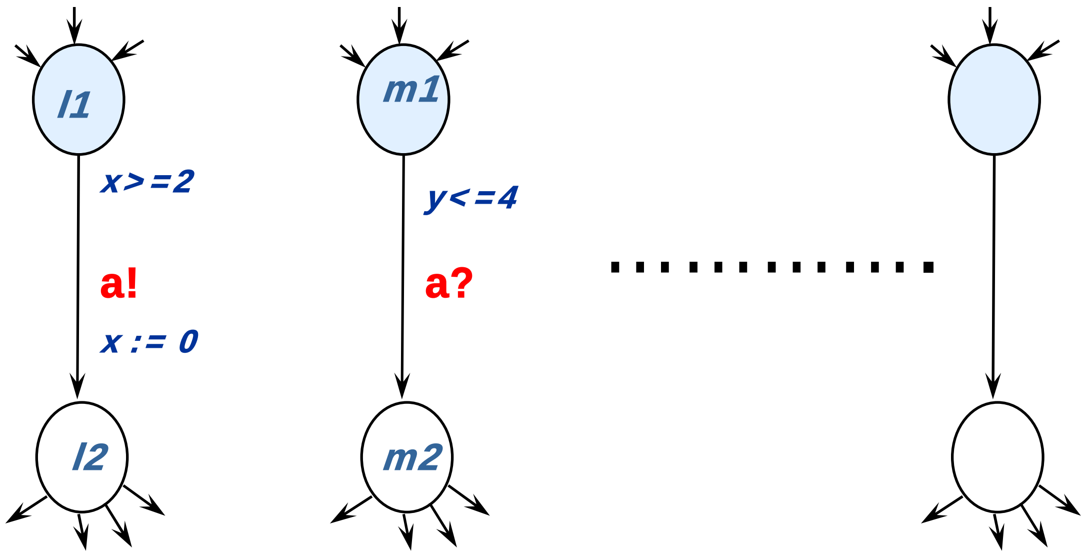
Light Control Interface



BRICS
Basic Research
in Computer Science

Networks of Timed Automata

(a'la CCS)



Two-way synchronization on *complementary* actions.

Closed Systems!

Example transitions

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

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

If **a** URGENT CHANNEL

Network Semantics

$$T_1 \|_X T_2 = (S_1 \times S_2, \rightarrow, s_0^1 \|_X s_0^2) \quad \text{where}$$

$$\frac{s_1 \xrightarrow{\mu}_1 s_1'}{s_1 \|_X s_2 \xrightarrow{\mu} s_1' \|_X s_2} \qquad \frac{s_2 \xrightarrow{\mu}_2 s_2'}{s_1 \|_X s_2 \xrightarrow{\mu} s_1 \|_X s_2'}$$

$$\frac{s_1 \xrightarrow{a!}_1 s_1' \qquad s_2 \xrightarrow{a?}_2 s_2'}{s_1 \|_X s_2 \xrightarrow{\tau} s_1' \|_X s_2'}$$

$$\frac{s_1 \xrightarrow{e(d)}_1 s_1' \qquad s_2 \xrightarrow{e(d)}_2 s_2'}{s_1 \|_X s_2 \xrightarrow{e(d)} s_1' \|_X s_2'}$$

Network Semantics

(URGENT synchronization)

+ Urgent synchronization

$$T_1 \|_X T_2 = (S_1 \times S_2, \rightarrow, s_0^1 \|_X s_0^2)$$ where

$$\frac{s_1 \xrightarrow{\mu}_1 s_1'}{s_1 \|_X s_2 \xrightarrow{\mu} s_1' \|_X s_2} \qquad \frac{s_2 \xrightarrow{\mu}_2 s_2'}{s_1 \|_X s_2 \xrightarrow{\mu} s_1 \|_X s_2'}$$

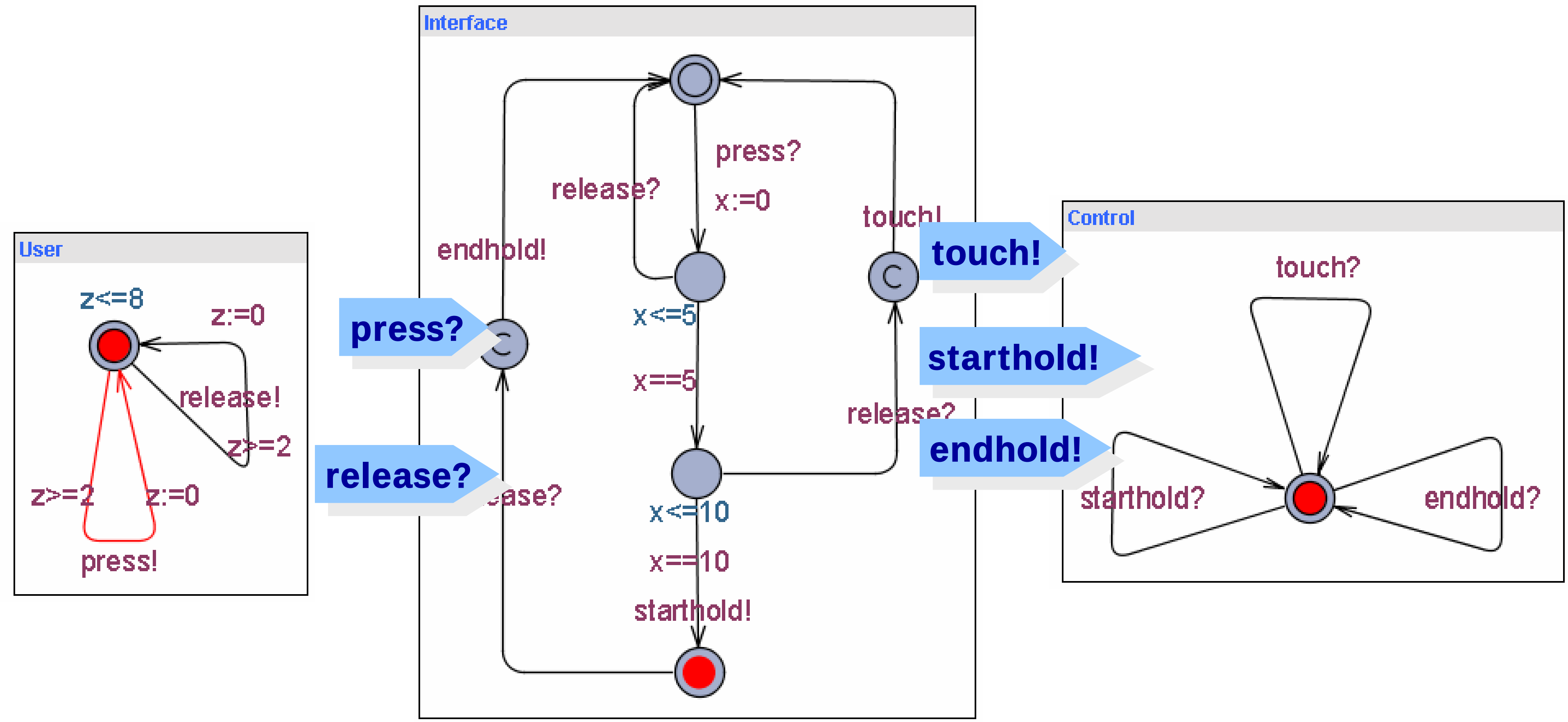
$$\frac{s_1 \xrightarrow{a!}_1 s_1' \qquad s_2 \xrightarrow{a?}_2 s_2'}{s_1 \|_X s_2 \xrightarrow{\tau} s_1' \|_X s_2'}$$

$$\frac{s_1 \xrightarrow{e(d)}_1 s_1' \qquad s_2 \xrightarrow{e(d)}_2 s_2'}{s_1 \|_X s_2 \xrightarrow{e(d)} s_1' \|_X s_2'}$$

$\forall d' < d, \forall u \in \mathrm{UAct}:$

$\neg\, (s_1 \xrightarrow{e(d')} \xrightarrow{u?} \wedge\ s_2 \xrightarrow{e(d')} \xrightarrow{u!})$

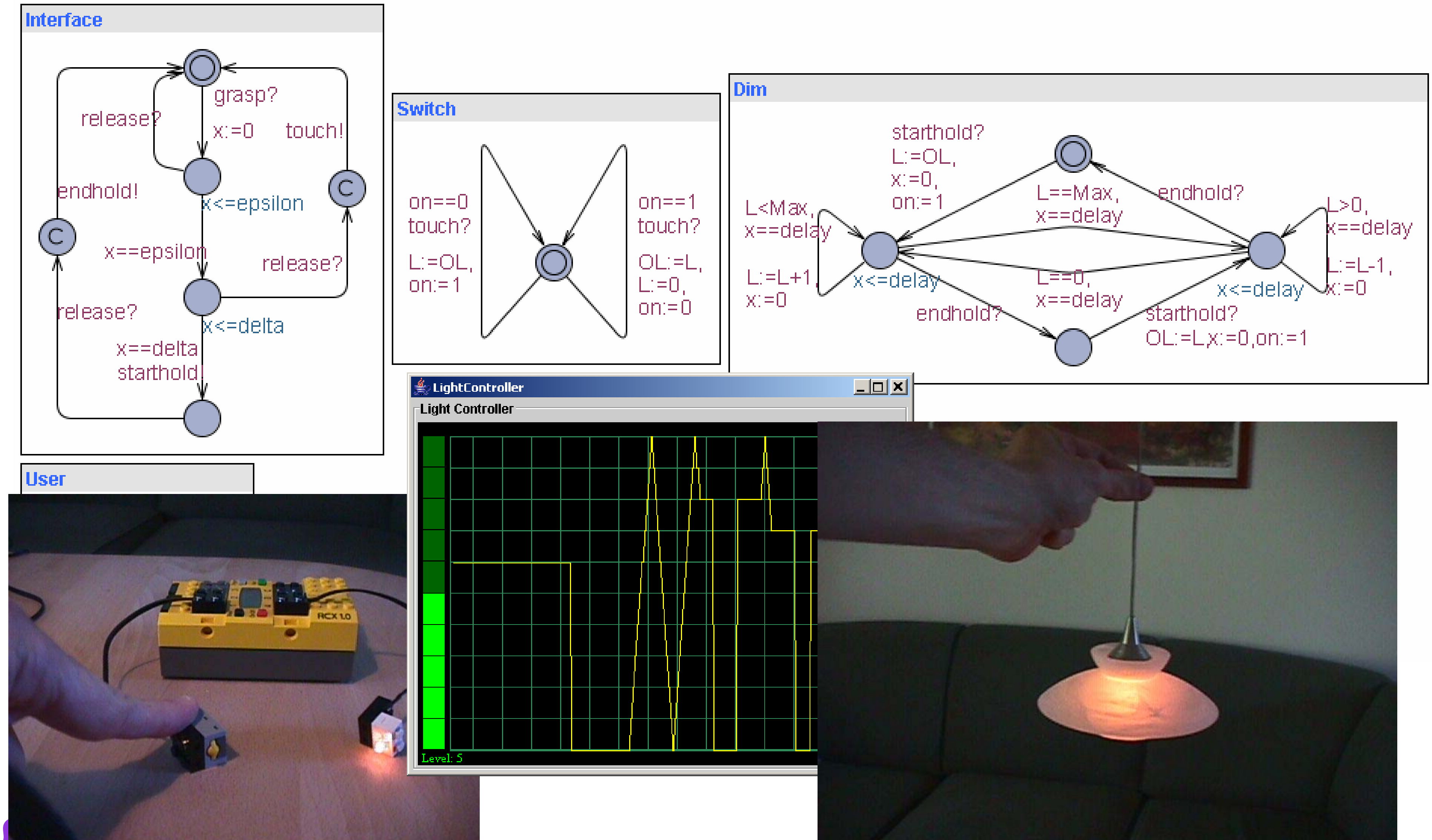
Light Control Network



BRICS
Basic Research
in Computer Science

Validation

Light Controller



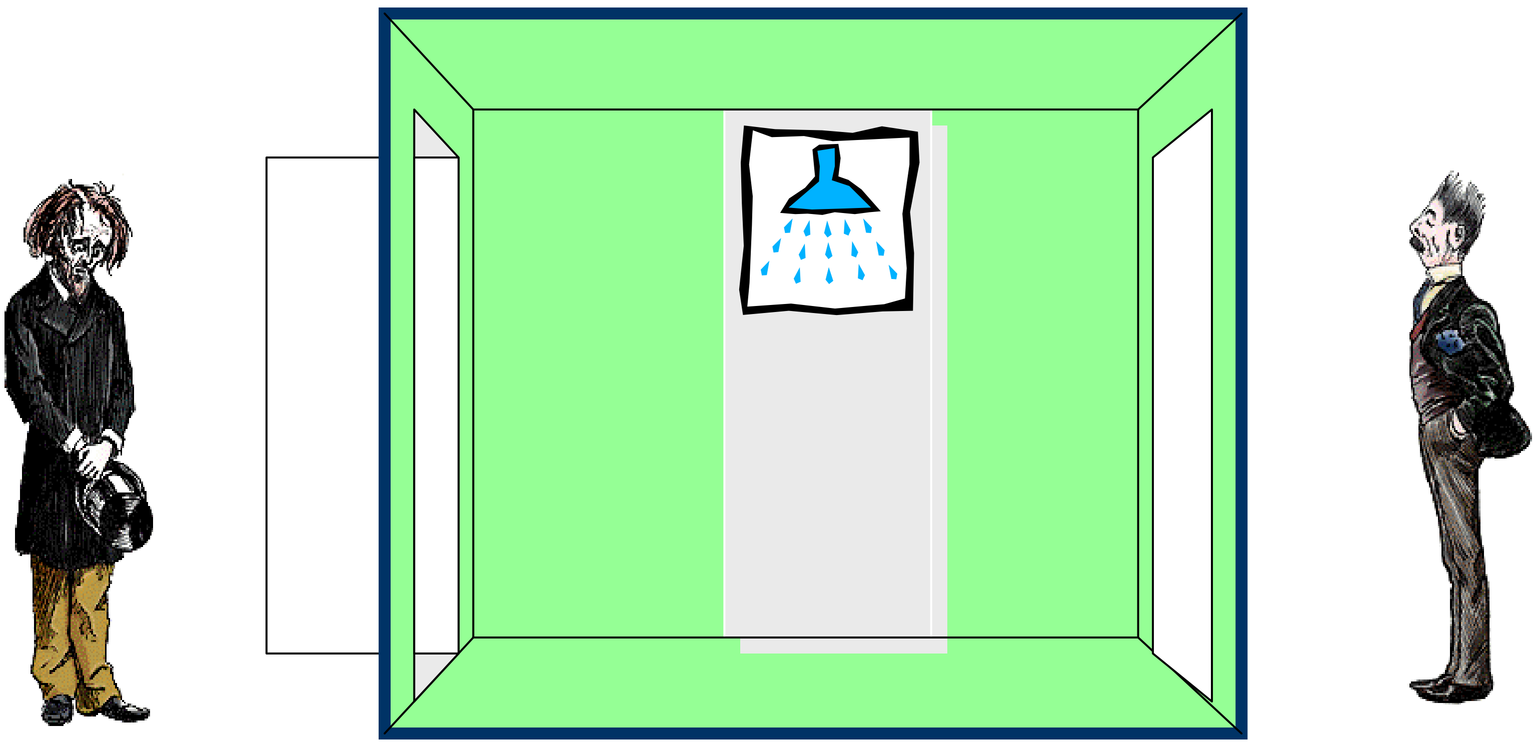
Druzba:
The Shower Problem

AALBORG UNIVERSITY DENMARK

BRICS
Basic Research in Computer Science

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

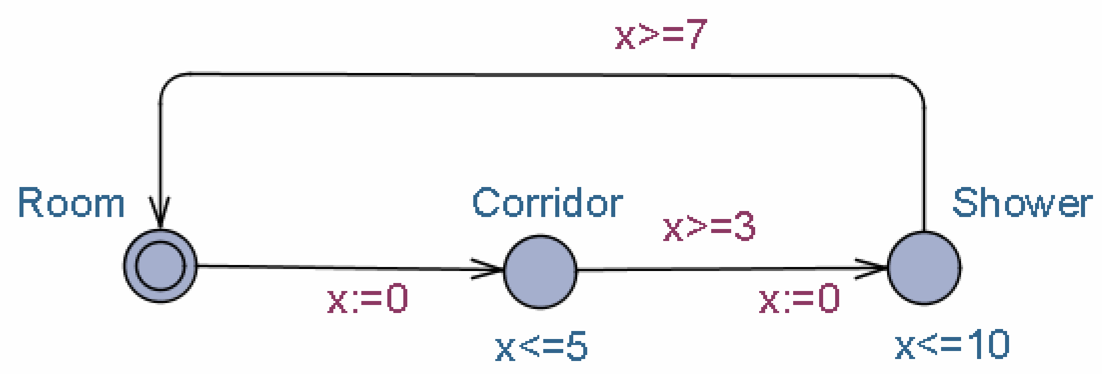
The Druzba MUTEX Problem

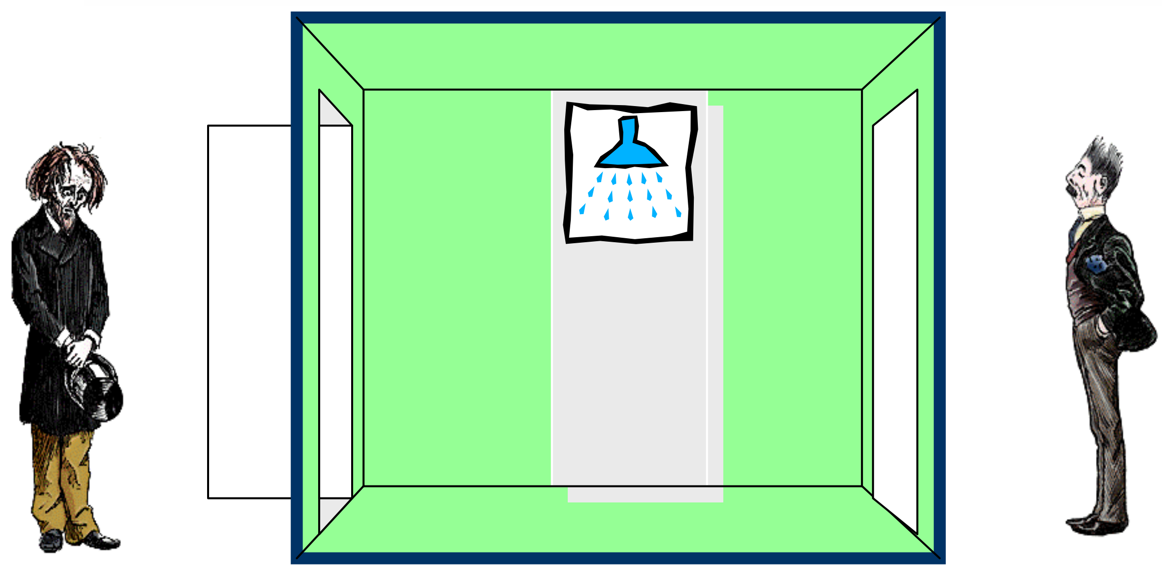


Kim

Gerd

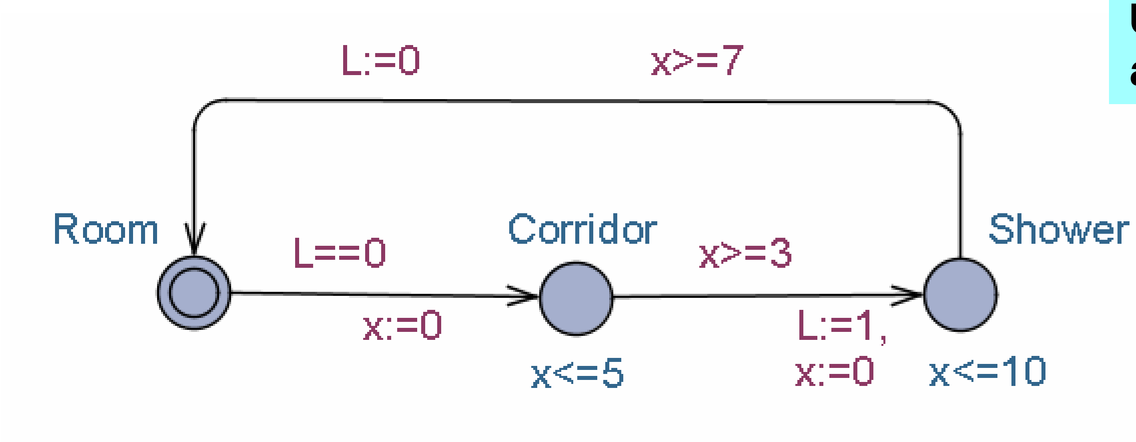
The Druzba MUTEX Problem

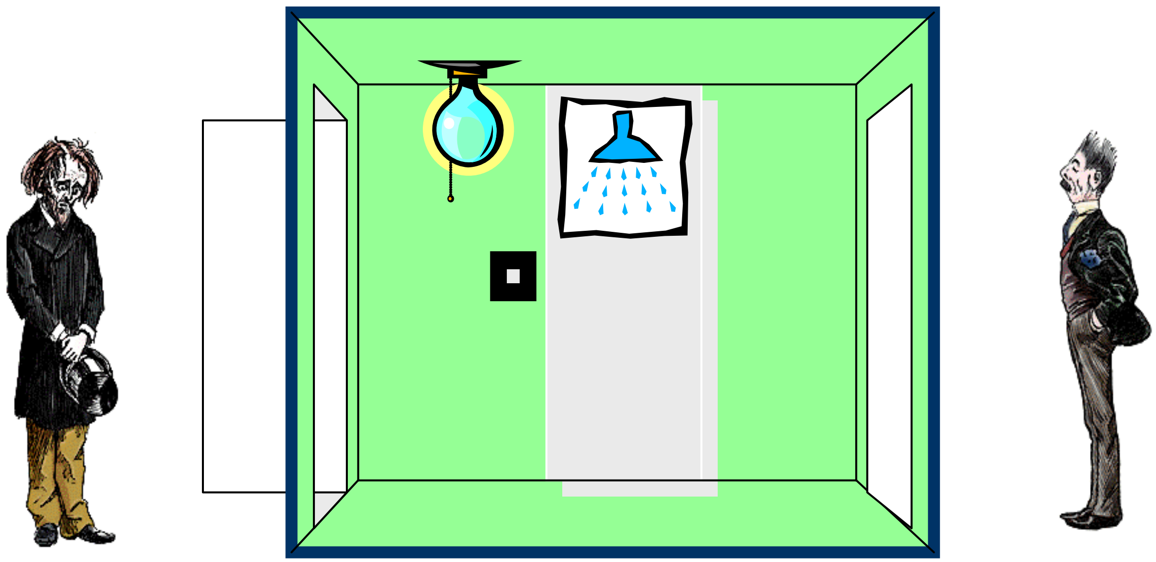




The Druzba MUTEX Problem

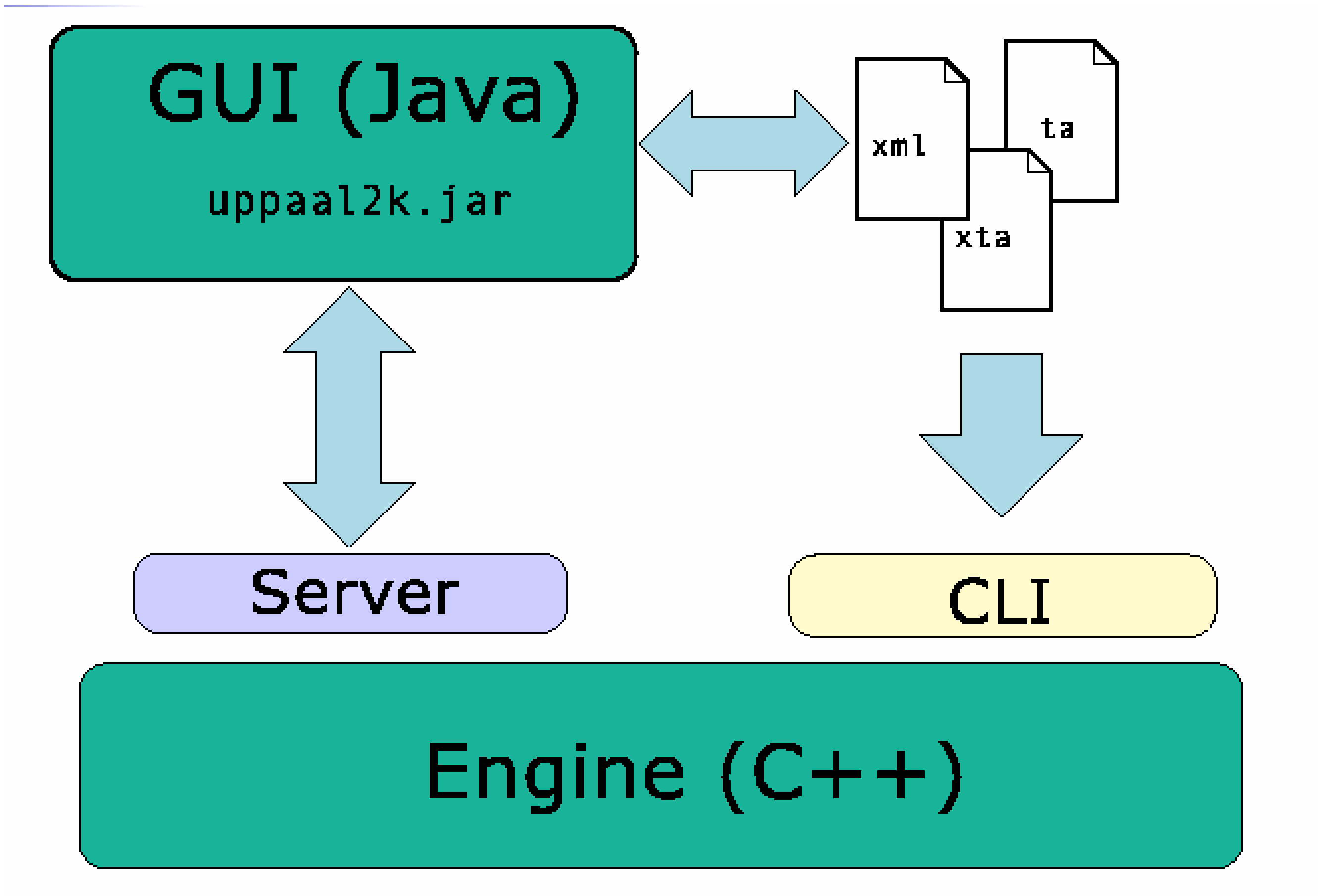
Using the light as semaphor





Overview of the UPPAAL Toolkit

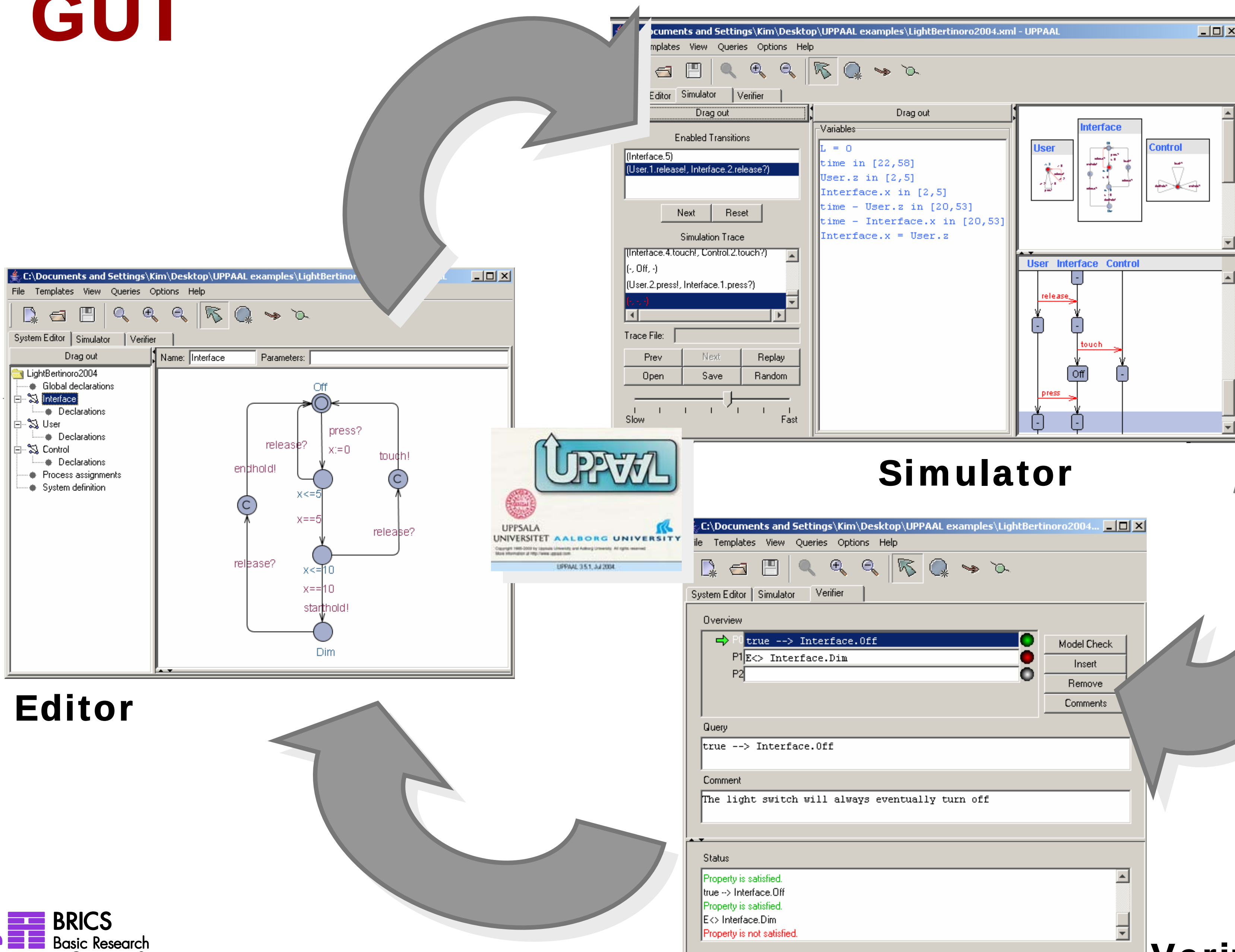
UPPAAL's architecture



Linux, Windows, Solaris, MacOS

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

GUI



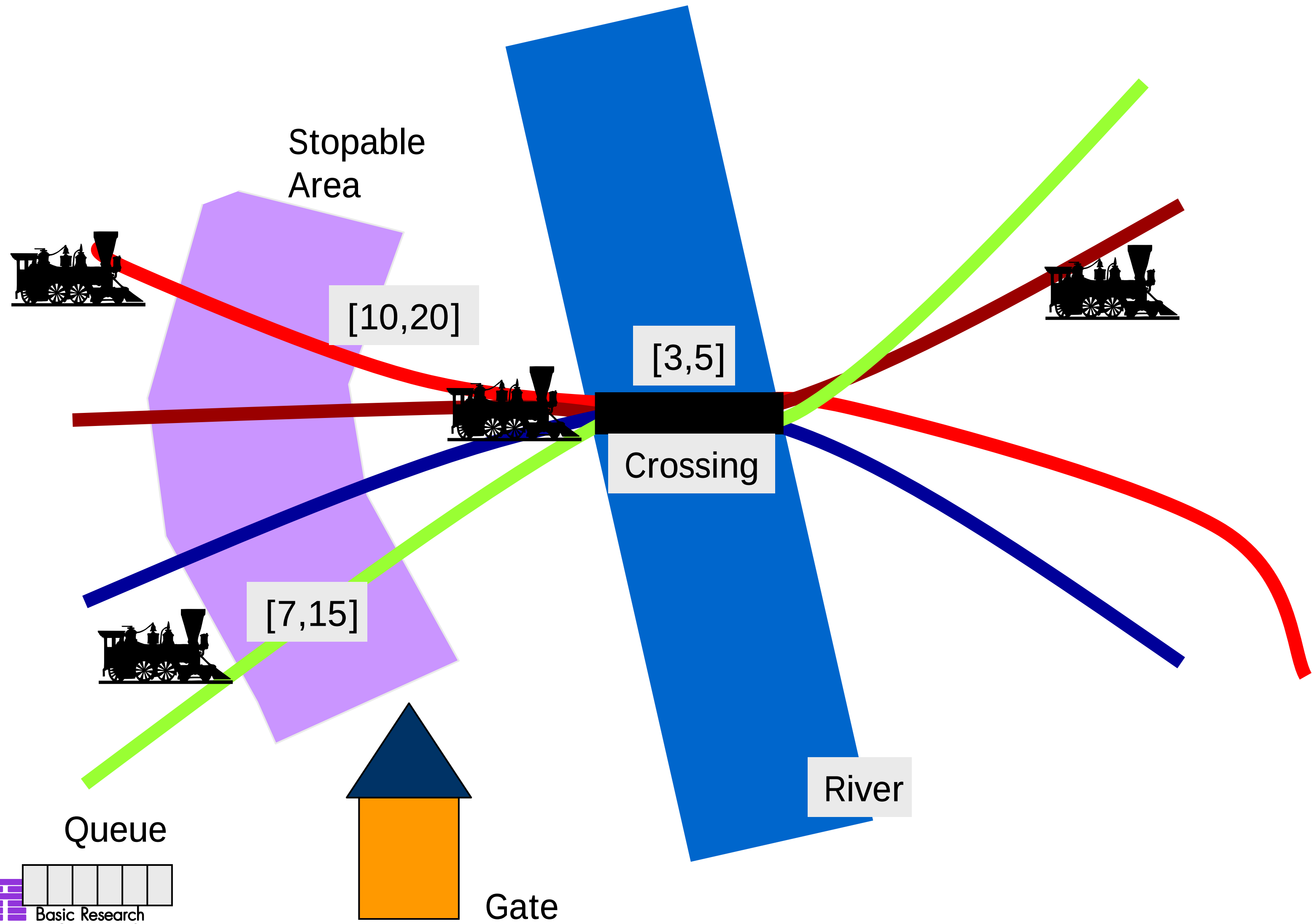
Simulator

Editor

Verifier

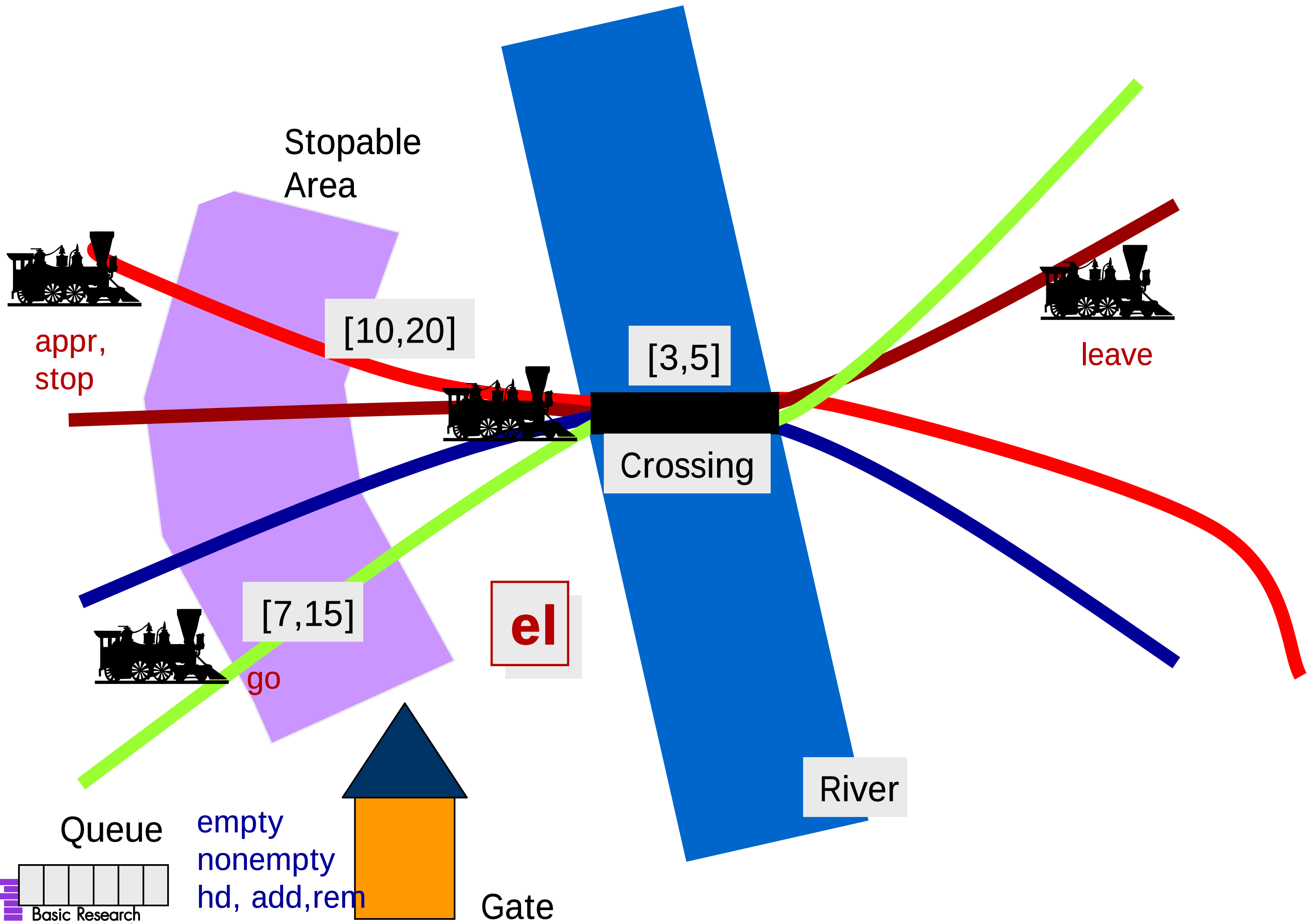
BRICS
Basic Research
in Computer Science

Train Crossing



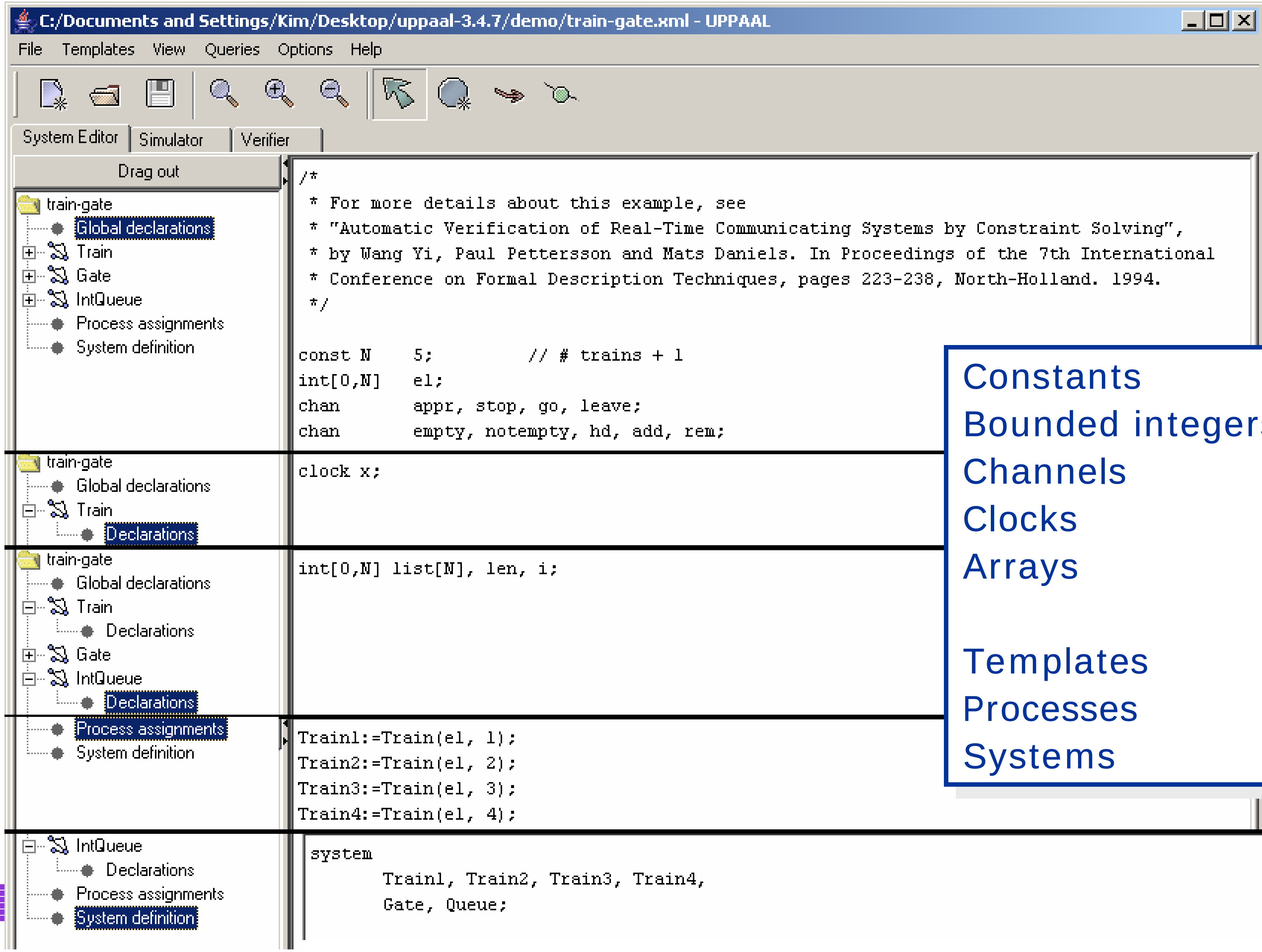
Train Crossing

Communication via channels and shared variable.



Timed Automata
in UPPAAL

Declarations



Constants
Bounded integers
Channels
Clocks
Arrays

Templates
Processes
Systems

Declarations in UPPAAL

- The syntax used for declarations in UPPAAL is similar to the syntax used in the C programming language.

- **Clocks**:
  - Syntax:

```
clock x1, …, xn ;
```

  - Example:
  - `clock x, y;` **Declares two clocks: x and y.**

Declarations in UPPAAL (cont.)

- **Data variables**
  - Syntax:

```
– int n1, … ;
– int[l,u] n1, … ;
– int n1[m], … ;
```

Integer with "default" domain.

Integer with domain "l" to "u".

Integer array w. elements n1[0] to n1[m-1].

  - Example:
  - `int a, b;`
  - `int[0,1] a, b[5][6];`

Declarations in UPPAAL (cont.)

- **Actions** (or channels):
  - Syntax:

```
chan a, … ;
urgent chan b, … ;
```

Ordinary channels.

Urgent actions (see later)

  - Example:
  - **`chan a, b;`**
  - **`urgent chan c;`**

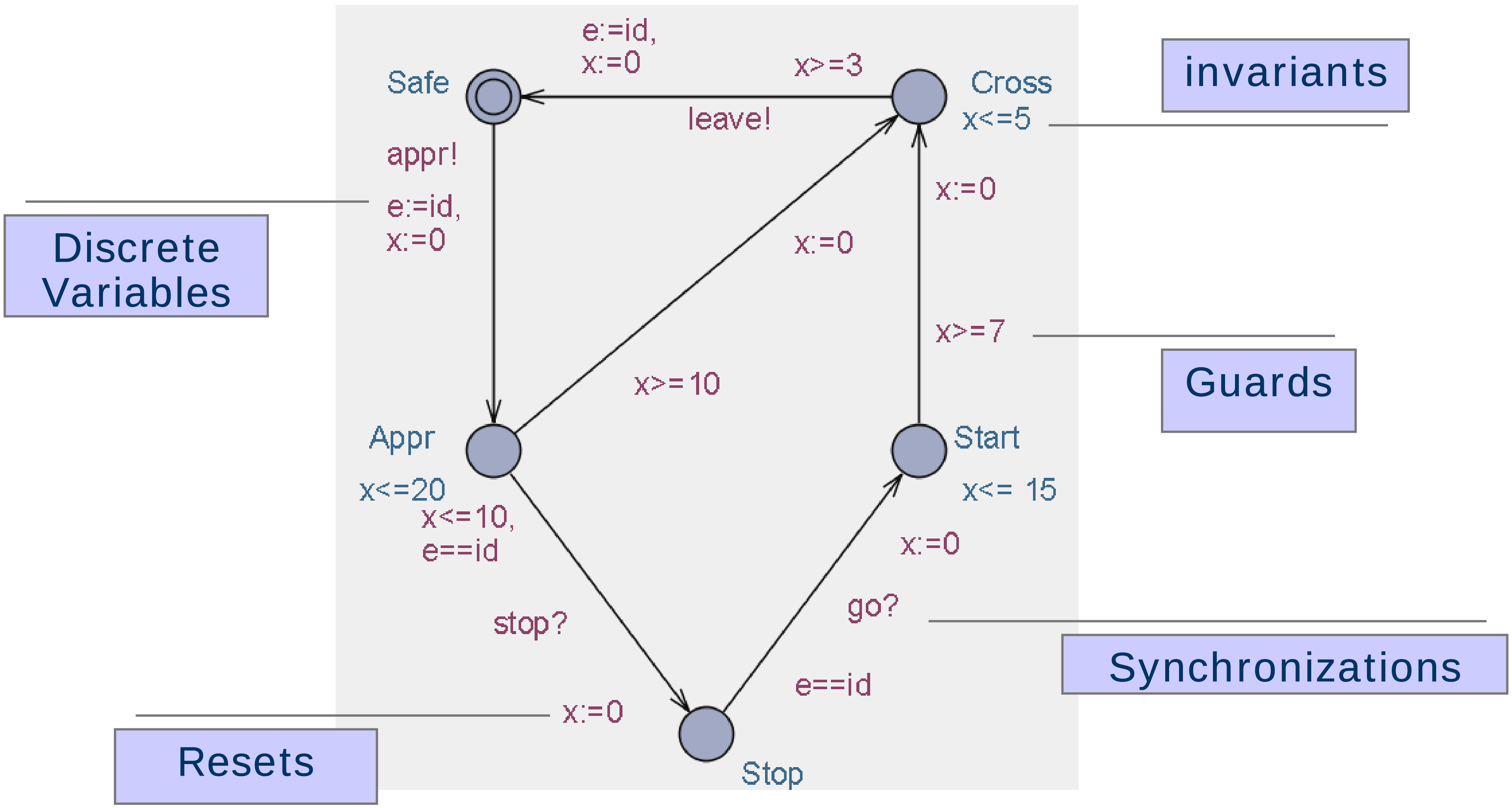
Declarations UPPAAL (cont.)

- **Constants**
  - Syntax:

```
– const int c1 = n1;
```

  - Example:
  - `const int[0,1] YES = 1;`
  - `const bool NO = false;`

Timed Automata in UPPAAL

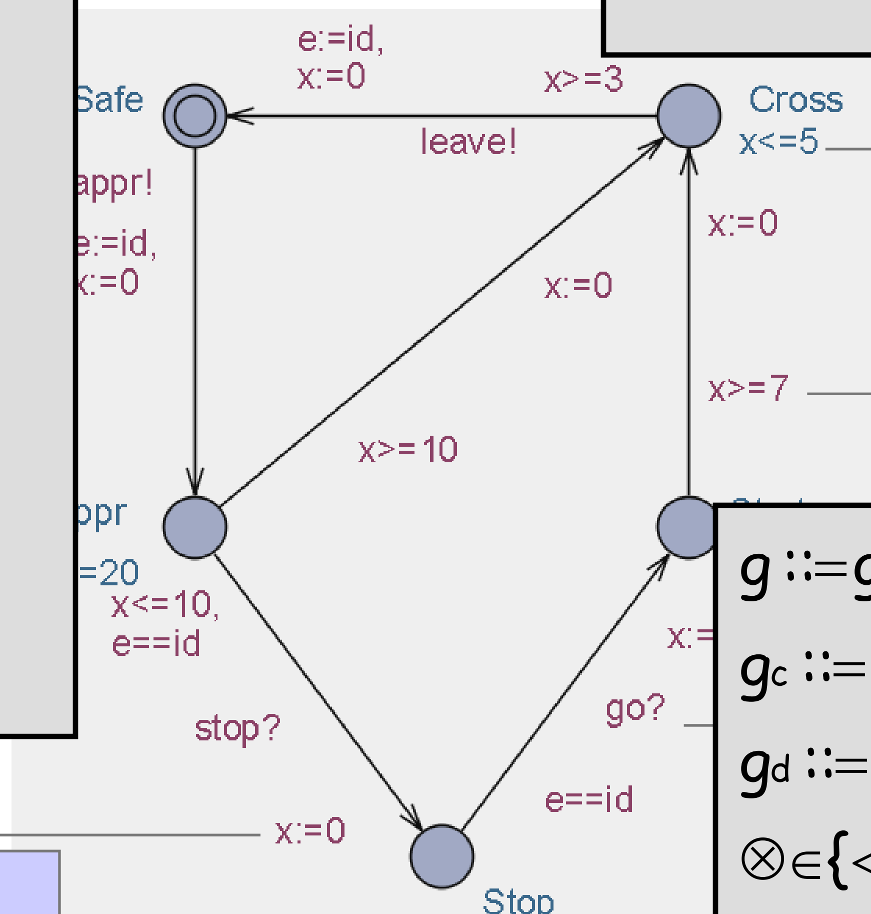


Timed Automata in UPPAAL

$$
\begin{aligned}
i &:= Expr \\
Expr &::= i \mid i[Expr] \mid \\
&\quad n \mid -Expr \mid \\
&\quad Expr + Expr \mid \\
&\quad Expr - Expr \mid \\
&\quad Expr * Expr \mid \\
&\quad Expr / Expr \mid \\
&\quad (g_d ? Expr : Expr)
\end{aligned}
$$

$$inv ::= x < Expr \mid x <= Expr \mid inv, inv$$

invariants



Guards

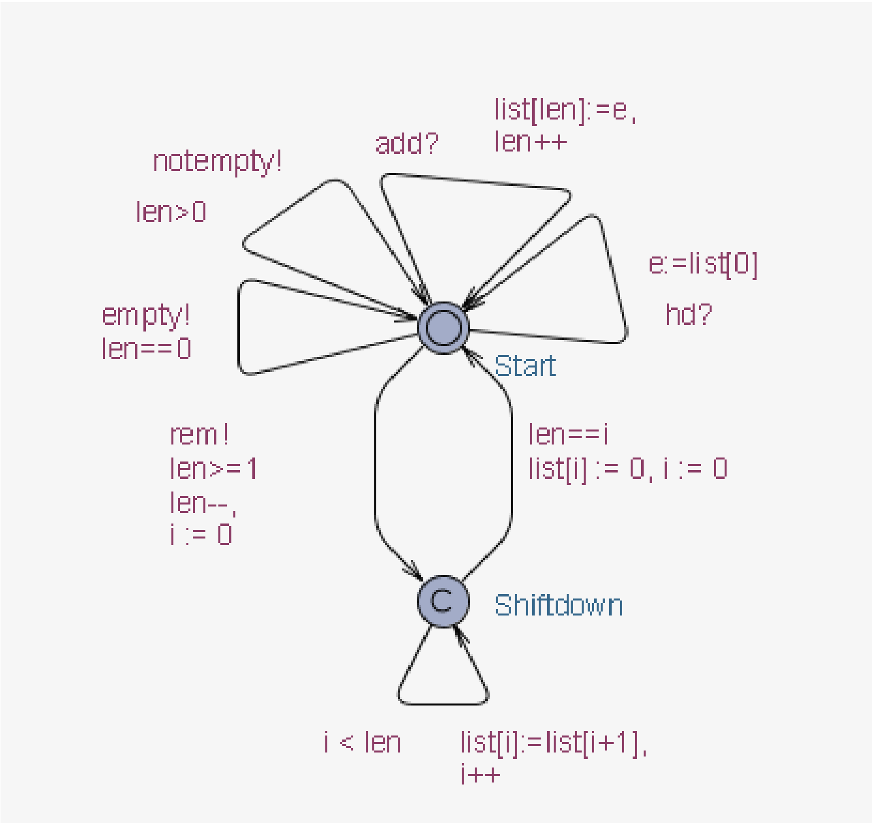
$$
\begin{aligned}
g &::= g_c \mid g_d \mid g, g \\
g_c &::= x \otimes Expr \mid x \otimes y + Expr \\
g_d &::= Expr\ op\ Expr \\
\otimes &\in \{<, <=, ==, >=, >\} \\
op &\in \{<, <=, ==, >=, >, !=\}
\end{aligned}
$$

Resets

$$x := Expr$$

BRICS
Basic Research
in Computer Science

Expressions



used in
- guards,
- invariants,
- assignments,
- synchronizations
- properties,

Expressions

```
Expression
   ::= ID
   |   NAT
   |   Expression '[' Expression ']'
   |   '(' Expression ')'
   |   Expression '++' | '++' Expression
   |   Expression '--' | '--' Expression
   |   Expression AssignOp Expression
   |   UnaryOp Expression
   |   Expression BinOp Expression
   |   Expression '?' Expression ':' Expression
   |   ID '.' ID
```

Operators

Unary

```
'-' | '!' | 'not'
```

Binary

```
'<'   | '<=' | '==' | '!=' | '>=' | '>'
'+'   | '-'  | '*'  | '/'  | '%'  | '&'
'|'   | '^'  | '<<' | '>>' | '&&' | '||'
'and' | 'or' | 'imply'
```

Assignment

```
':=' | '+=' | '-=' | '*=' | '/=' | '%='
'|=' | '&=' | '^=' | '<<=' | '>>='
```

Guards, Invariants, Assignments

Guards:

- It is side-effect free, type correct, and evaluates to boolean
- Only clock variables, integer variables, constants are referenced (or arrays of such)
- Clocks and differences are only compared to integer expressions
- Guards over clocks are essentially conjunctions (I.e. disjunctions are only allowed over integer conditions)

Assignments

- It has a side effect and is type correct
- Only clock variable, integer variables and constants are referenced (or arrays of such)
- Only integer are assigned to clocks

Invariants

- It forms conjunctions of conditions of the form x<e or x<=e where x is a clock reference and e evaluates to an integer

BRICS Basic Research in Computer Science

Synchronization

Binary Synchronization

- Declared like:
  `chan a, b, c[3];`
- If a is channel then:
  - a! = Emmision
  - a? = Reception

- Two edges in different processes can synchronize if one is emitting and the other is receiving on the same channel.

Broadcast Synchronization

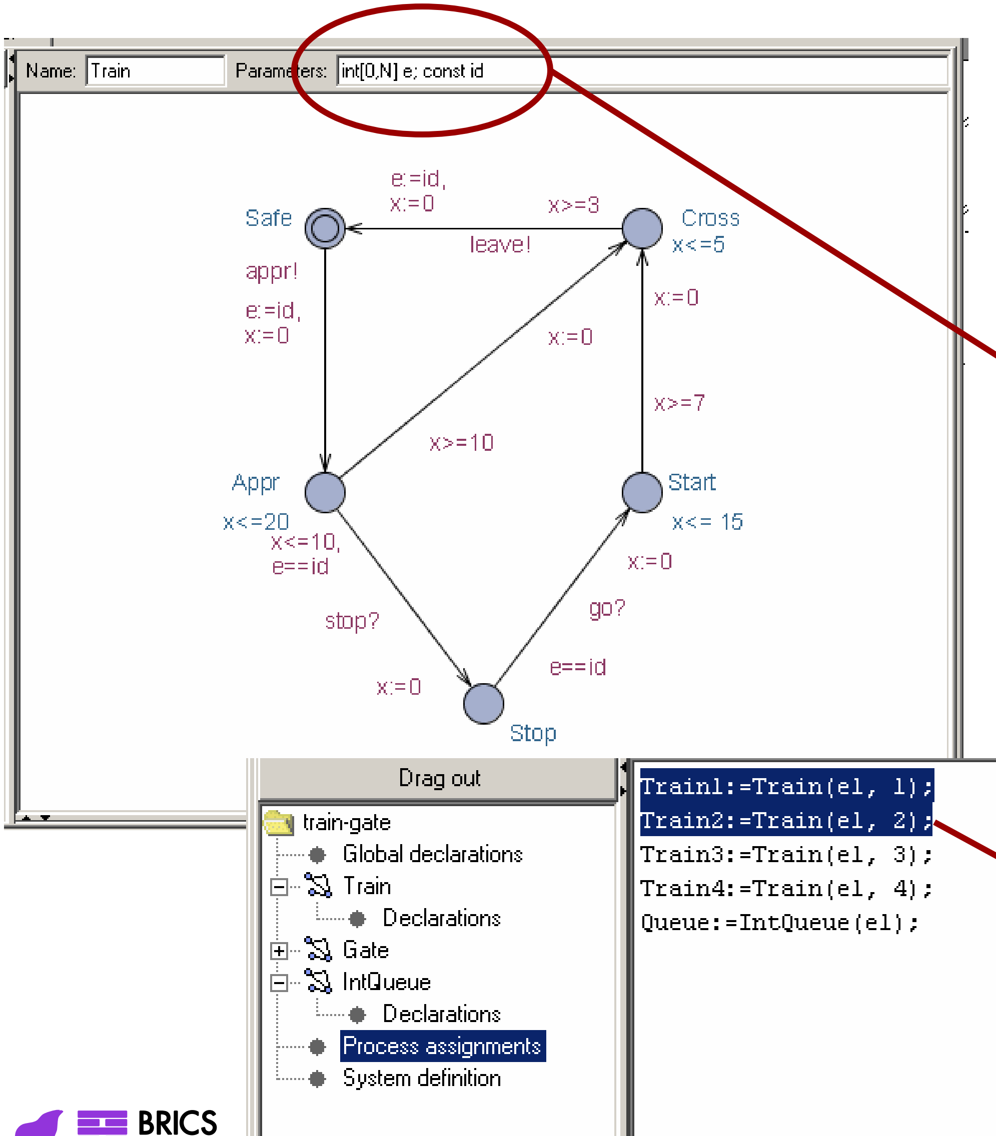
- Declared like
  `broadcast chan a, b, c[2];`
- If a is a broadcast channel:
  - a! = Emmision of broadcast
  - a? = Reception of broadcast

- A set of edges in different processes can synchronize if one is emitting and the others are receiving on the same b.c. channle. A process can always emit.
  Receivers MUST synchronize if they can.
  No blocking.

More on Types

- Multi dimensional arrays
  - e.g. `int b[4][2];`
- Array initialiser:
  - e.g. `int b[4] := { 1, 2, 3, 4 };`
- Arrays of channels, clocks, constants.
  - e.g.
  - `chan a[3];`
  - `clock c[3];`
  - `const k[3] { 1, 2, 3 };`
- Broadcast channels.
  - e.g. `broadcast chan a;`

Templates



- Templates may be parameterised:
  - `int v; const min; const max`
  - `int[0,N] e; const id`
- Templates are instantiated to form processes:
  - `P:= A(i,1,5);`
  - `Q:= A(j,0,4);`
  - `Train1:=Train(el, 1);`
  - `Train2:=Train(el, 2);`

Extensions

Select statement

- models a non-deterministic choise
- `x : int[0,42]`

Types

- Record types
- Type declarations
- Meta variables:

  not stored with state

  `meta int x;`

Forall / Exists expressions

- `forall (x:int[0,42]) expr`
  true if expr is true for *all* values in [0,42] of x
- `exists (x:int[0,4]) expr`
  true if expr is true for *some* values in [0,42] of x

Example:

```
forall
(x:int[0,4])array[x];
```

Urgency & Commitment

Urgent Channels

- No delay if the synchronization edges can be taken !

- No clock guard allowed.
- Guards on data-variables.

- Declarations:
  `urgent chan a, b, c[3];`

Urgent Locations

- No delay – time is freezed!
- May reduce number of clocks!

Committed Locations

- No delay.
- Next transition MUST involve edge in one of the processes in committed location
- May reduce considerably state space

Queries : Specification Language

Logical Specifications

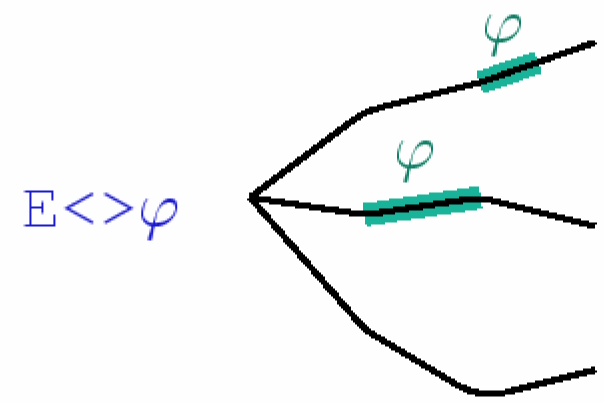
- Validation Properties
  - Possibly: E<> *P*
- Safety Properties
  - Invariant: A[] *P*
  - Pos. Inv.: E[] *P*
- Liveness Properties
  - Eventually: A<> *P*
  - Leadsto: $P \rightarrow Q$
- Bounded Liveness
  - Leads to within: $P \rightarrow_{\leq t} Q$

The expressions *P* and *Q* must be type safe, side effect free, and evaluate to a boolean.

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

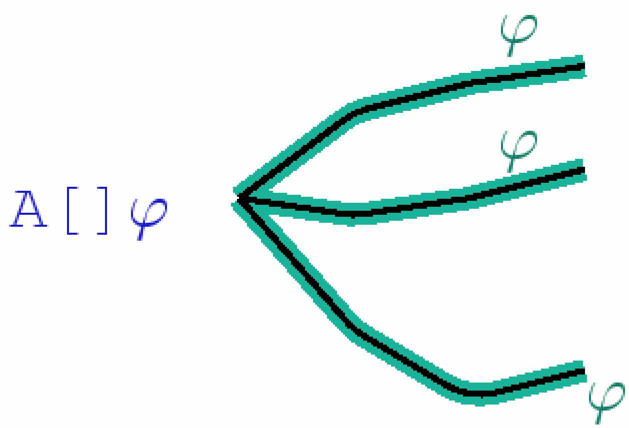
Logical Specifications

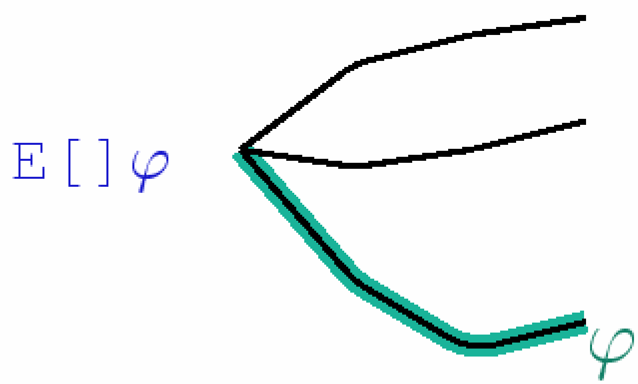
- Validation Properties
  - Possibly: E<> *P*
- Safety Properties
  - Invariant: A[] *P*
  - Pos. Inv.: E[] *P*
- Liveness Properties
  - Eventually: A<> *P*
  - Leadsto: P → *Q*
- Bounded Liveness
  - Leads to within: *P* $\rightarrow_{\leq t}$ *Q*



Logical Specifications

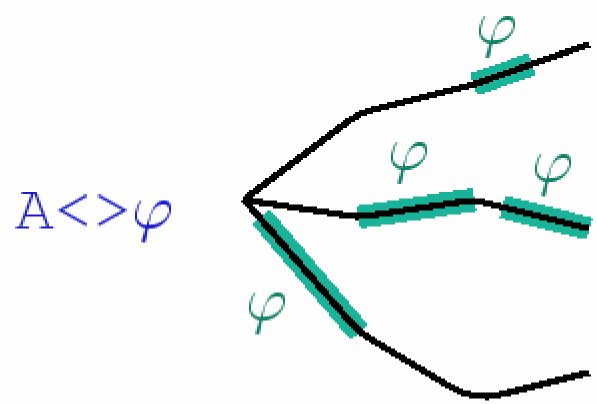
- Validation Properties
  - Possibly: E<> *P*
- Safety Properties
  - Invariant: A[] *P*
  - Pos. Inv.: E[] *P*
- Liveness Properties
  - Eventually: A<> *P*
  - Leadsto: P → *Q*
- Bounded Liveness
  - Leads to within: *P* $\rightarrow_{\leq t}$ *Q*

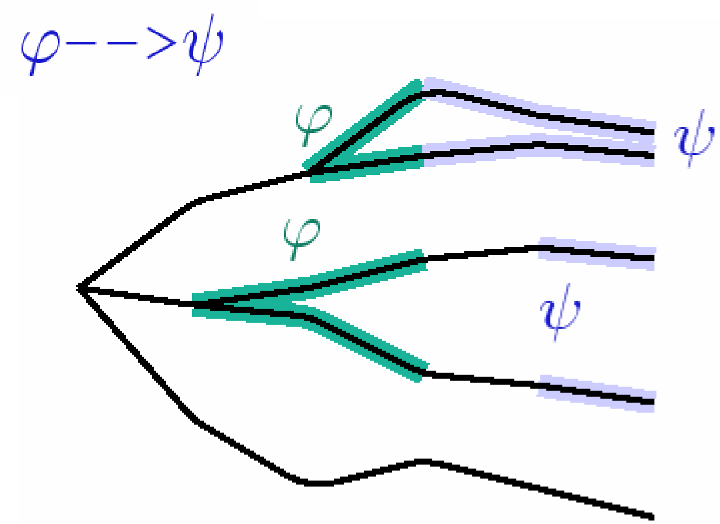




Logical Specifications

- Validation Properties
  - Possibly: E<> *P*
- Safety Properties
  - Invariant: A[] *P*
  - Pos. Inv.: E[] *P*
- Liveness Properties
  - Eventually: A<> *P*
  - Leadsto: P → *Q*
- Bounded Liveness
  - Leads to within: $P \rightarrow_{\leq t} Q$

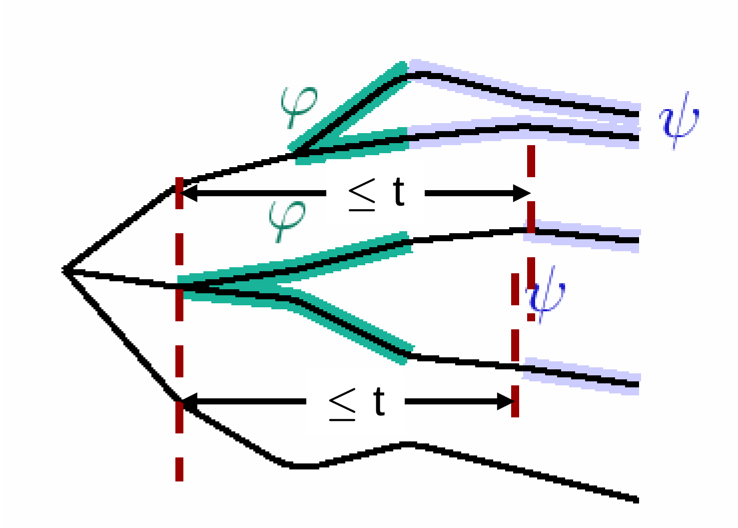




BRICS Basic Research in Computer Science

Logical Specifications

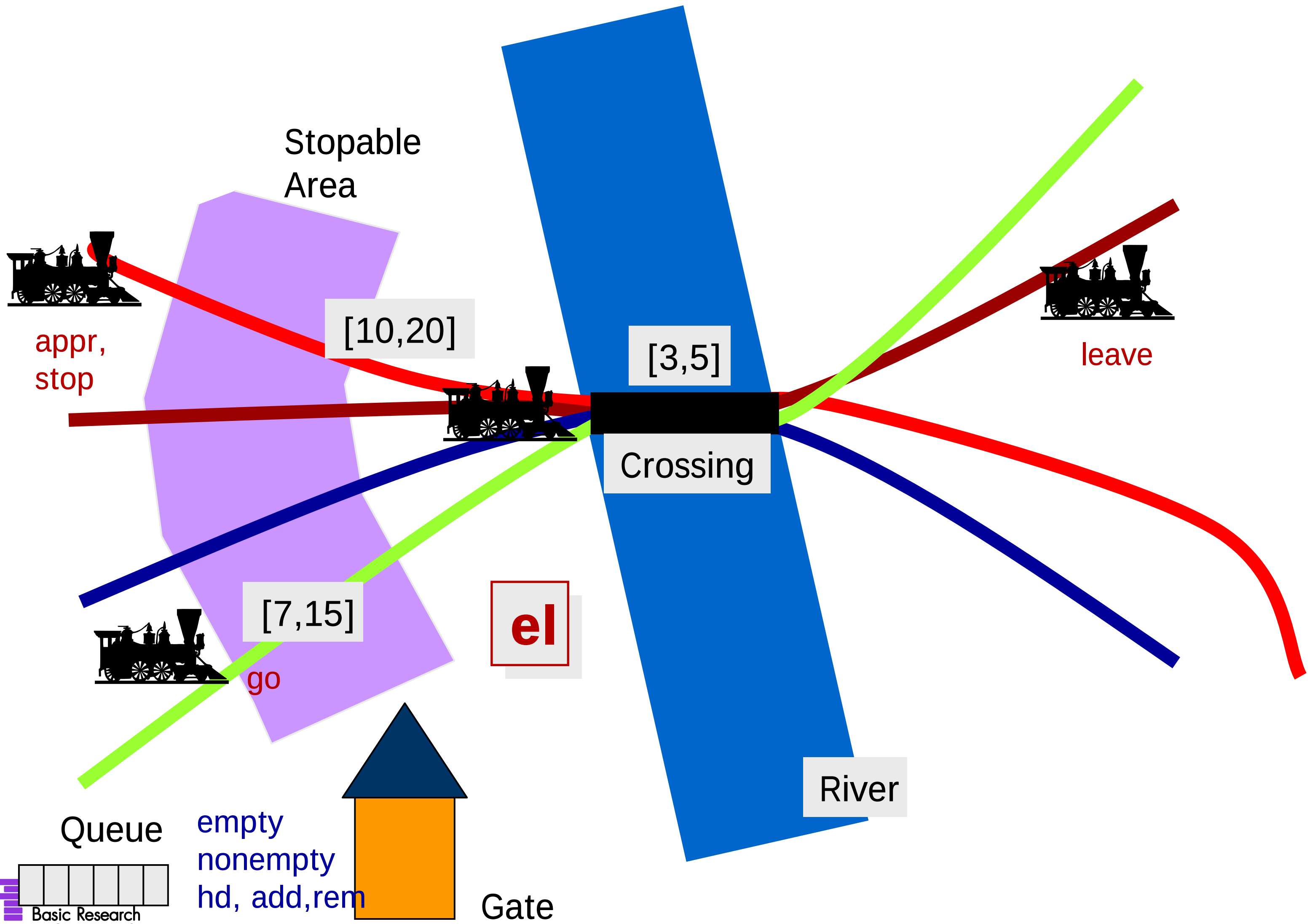
- Validation Properties
  - Possibly: E<> P
- Safety Properties
  - Invariant: A[] P
  - Pos. Inv.: E[] P
- Liveness Properties
  - Eventually: A<> P
  - Leadsto: $P \rightarrow Q$
- Bounded Liveness
  - Leads to within: $P \rightarrow_{\leq t} Q$



BRICS Basic Research in Computer Science

Train Crossing

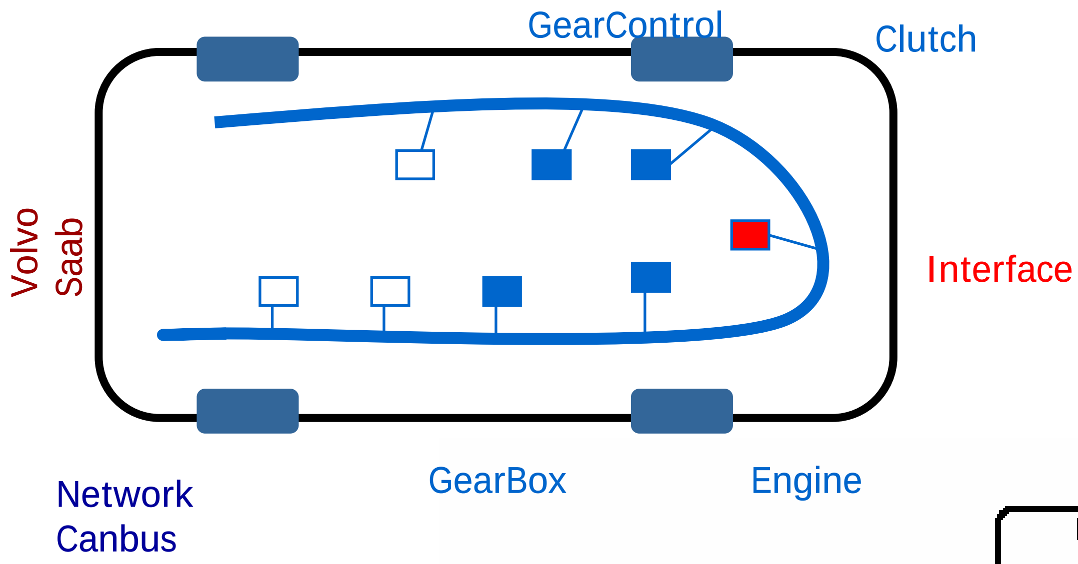
Communication via channels and shared variable.

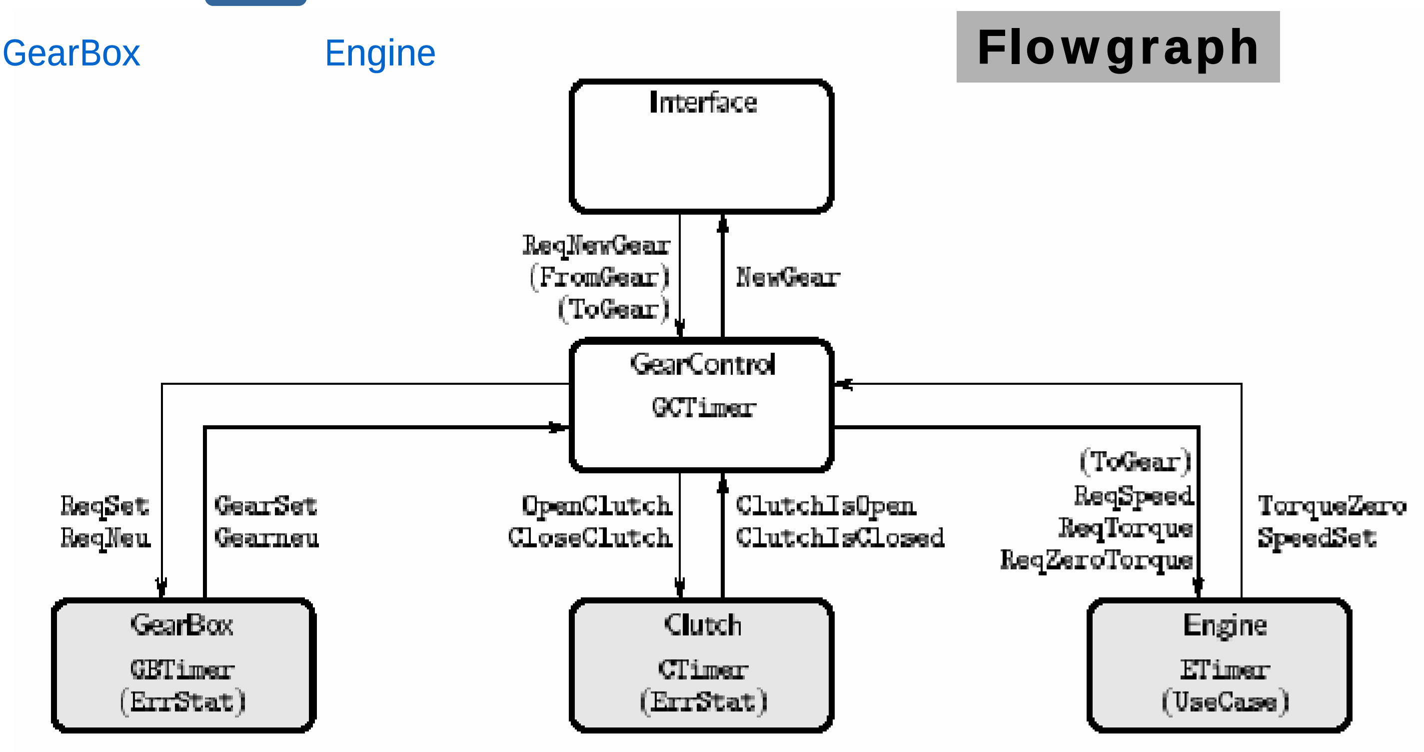


Gear Controller

with MECEL AB

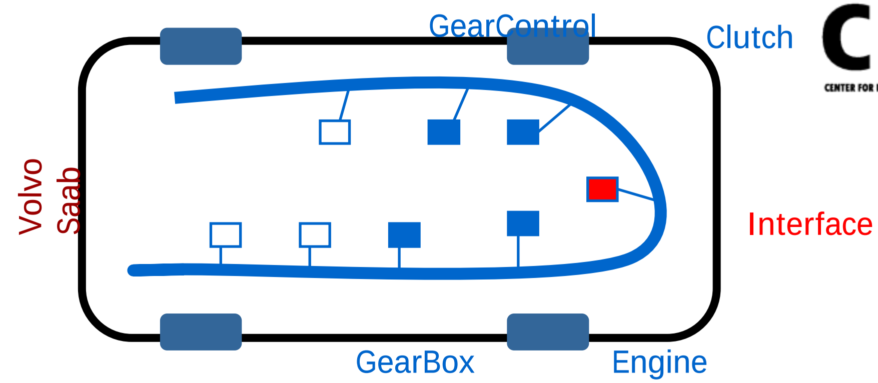
Lindahl, Pettersson, Yi 1998





Gear Controller

with MECEL AB



Requirements

$$\mathsf{GearControl@Initiate} \rightsquigarrow_{\leq 1500} ((\mathrm{ErrStat} = 0) \Rightarrow \mathsf{GearControl@GearChanged}) \tag{1}$$

$$\mathsf{GearControl@Initiate} \rightsquigarrow_{\leq 1000} ((\mathrm{ErrStat} = 0 \wedge \mathrm{UseCase} = 0) \Rightarrow \mathsf{GearControl@GearChanged}) \tag{2}$$

$$\mathsf{Clutch@ErrorClose} \rightsquigarrow_{\leq 200} \mathsf{GearControl@CCloseError} \tag{3}$$

$$\mathsf{Clutch@ErrorOpen} \rightsquigarrow_{\leq 200} \mathsf{GearControl@COpenError} \tag{4}$$

$$\mathsf{GearBox@ErrorIdle} \rightsquigarrow_{\leq 350} \mathsf{GearControl@GSetError} \tag{5}$$

$$\mathsf{GearBox@ErrorNeu} \rightsquigarrow_{\leq 200} \mathsf{GearControl@GNeuError} \tag{6}$$

$$Inv\ (\mathsf{GearControl@CCloseError} \Rightarrow \mathsf{Clutch@ErrorClose}) \tag{7}$$

$$Inv\ (\mathsf{GearControl@COpenError} \Rightarrow \mathsf{Clutch@ErrorOpen}) \tag{8}$$

$$Inv\ (\mathsf{GearControl@GSetError} \Rightarrow \mathsf{GearBox@ErrorIdle}) \tag{9}$$

$$Inv\ (\mathsf{GearControl@GNeuError} \Rightarrow \mathsf{GearBox@ErrorNeu}) \tag{10}$$

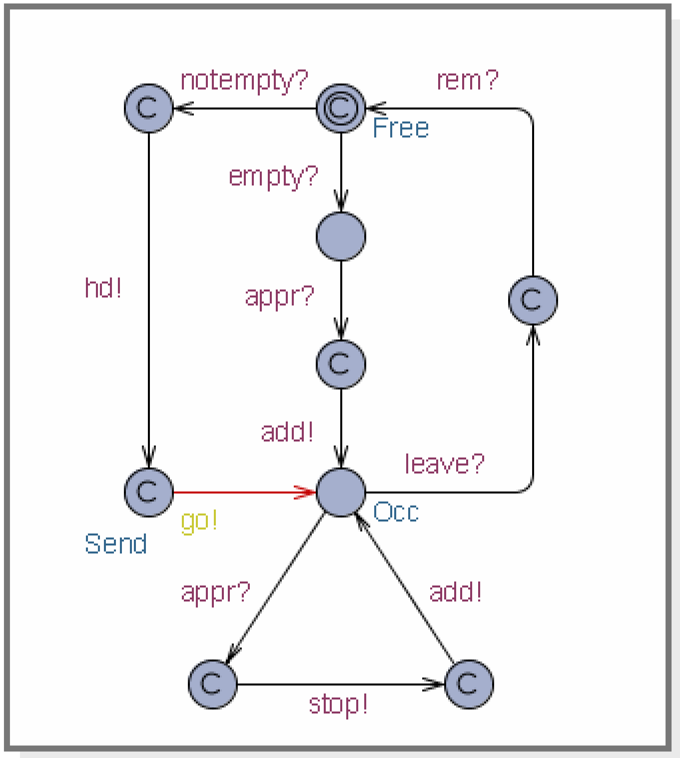
$$Inv\ (\mathsf{Engine@ErrorSpeed} \Rightarrow \mathrm{ErrStat} \neq 0) \tag{11}$$

$$Inv\ (\mathsf{Engine@Torque} \Rightarrow \mathsf{Clutch@Closed}) \tag{12}$$

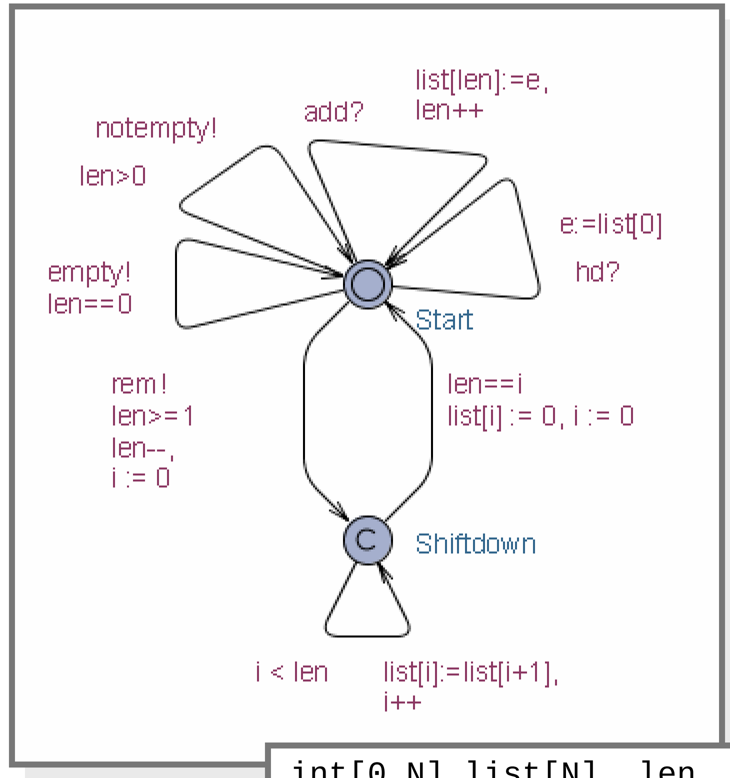
$$\bigwedge_{i \in \{R, N, 1, \ldots, 5\}} Poss\ (\mathsf{Gear@Gear}_i) \tag{13}$$

$$\bigwedge_{i \in \{R, 1, \ldots, 5\}} Inv\ ((\mathsf{GearControl@Gear} \wedge \mathsf{Gear@Gear}_i) \Rightarrow \mathsf{Engine@Torque}) \tag{14}$$

UPPAAL 3.4



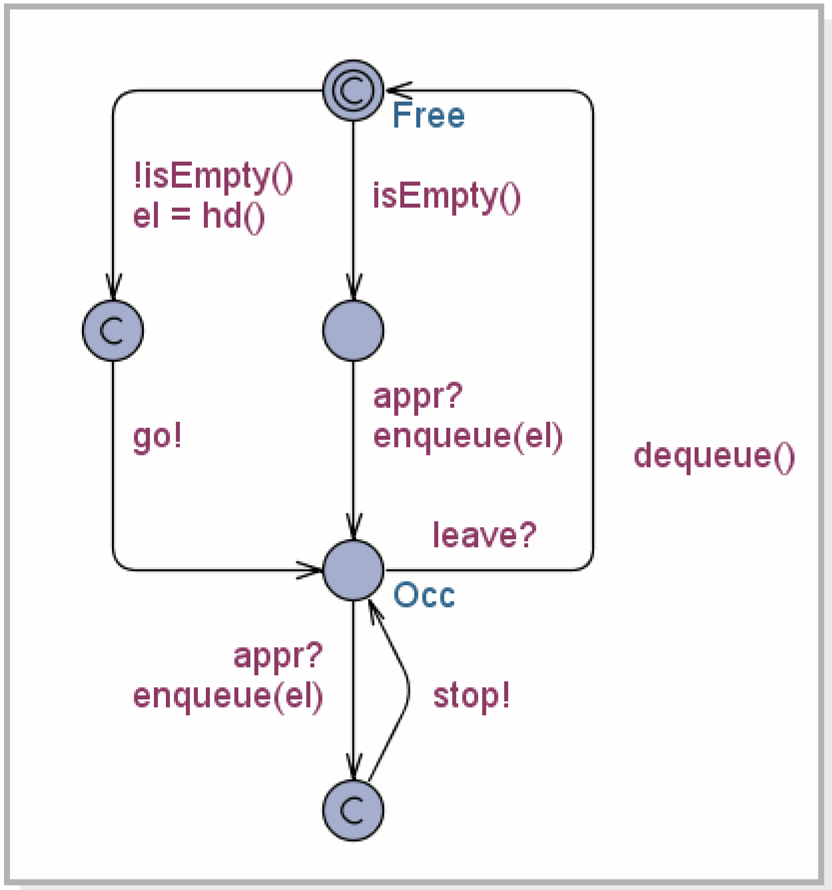
Gate Template




```
int[0,N] list[N], len, i;
```

IntQueue

UPPAAL 3.6 (3.5) with C-Code



Gate Template

```
int[0,N] list[N], len;

void enqueue(int[0,N] element)
{
        list[len++] = element;
}

void dequeue()
{
        int i = 0;
        len -= 1;
        while (i < len)
        {
                list[i] = list[i + 1];
                i++;
        }
        list[i] = 0;
        i = 0;
}

bool isEmpty()
{
  return len == 0;
}

int[0,N] hd()
{
  return list[0];
}
```

Gate Declaration

BRICS
Basic Research
in Computer Science

Case-Studies: Controllers

- Gearbox Controller [TACAS'98]
- Bang & Olufsen Power Controller [RTPS'99,FTRTFT'2k]
- SIDMAR Steel Production Plant [RTCSA'99, DSVV'2k]
- Real-Time RCX Control-Programs [ECRTS'2k]
- Experimental Batch Plant (2000)
- RCX Production Cell (2000)
- Terma, Verification of Memory Management for Radar (2001)
- Scheduling Lacquer Production (2005)
- Memory Arbiter Synthesis and Verification for a Radar Memory Interface Card [NJC'05]

Case Studies: Protocols

- Philips Audio Protocol [HS'95, CAV'95, RTSS'95, CAV'96]
- Collision-Avoidance Protocol [SPIN'95]
- Bounded Retransmission Protocol [TACAS'97]
- Bang & Olufsen Audio/Video Protocol [RTSS'97]
- TDMA Protocol [PRFTS'97]
- Lip-Synchronization Protocol [FMICS'97]
- Multimedia Streams [DSVIS'98]
- ATM ABR Protocol [CAV'99]
- ABB Fieldbus Protocol [ECRTS'2k]
- IEEE 1394 Firewire Root Contention (2000)
- Distributed Agreement Protocol [Formats05]
- Leader Election for Mobile Ad Hoc Networks [Charme05]

www.uppaal.com

RELATED SITES: TIMES | UPPAAL CORA | UPPAAL TRON

UPPAAL

Home

Home | About | Documentation | Download | Examples | Bugs

UPPAAL is an integrated tool environment for modeling, validation and verification of real-time systems modeled as networks of timed automata, extended with data types (bounded integers, arrays, etc.).

The tool is developed in collaboration between the Department of Information Technology at Uppsala University, Sweden and the Department of Computer Science at Aalborg University in Denmark.

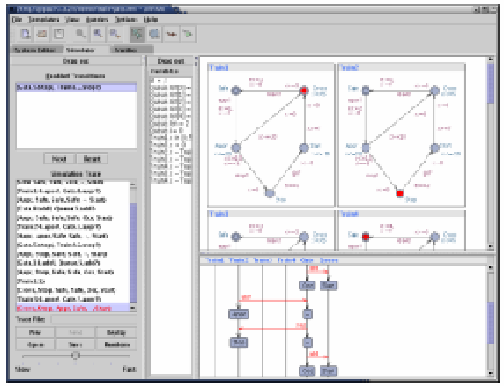


Figure 1: UPPAAL on screen.

Download

The current official release is UPPAAL 3.4.11 (Jun 23, 2005). A release of UPPAAL **3.6 alpha 3** (dec 20, 2005) is also available. For more information about UPPAAL version 3.4, we refer to this press release.

UPPSALA UNIVERSITET

AALBORG UNIVERSITY

License

The UPPAAL tool is **free** for non-profit applications. For information about commercial licenses, please email sales(at)uppaal(dot)com.

To find out more about UPPAAL, read this short introduction. Further information may be found at this web site in the pages About, Documentation, Download, and Examples.

Mailing Lists

UPPAAL has an open discussion forum group at Yahoo!Groups intended for users of the tool. To join or post to the forum, please refer to the information at the discussion forum page. Bugs should be reported using the bug tracking system. To email the development team directly, please use uppaal(at)list(dot)it(dot)uu(dot)se.

BRICS Basic Research in Computer Science