Real Time Model Checking

using UPPAAL

Kim G Larsen

Collaborators

@UPPsala

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

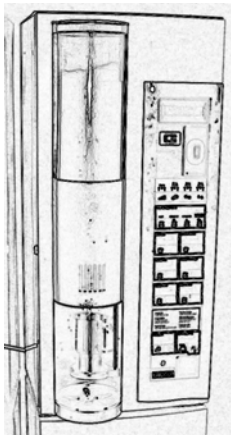
@AALborg

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

@Elsewhere

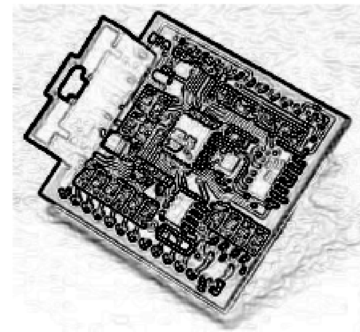
- Emmanuel Fleury, Didier Lime, Johan Bengtsson, Fredrik Larsson, Kåre J Kristoffersen, Tobias Amnell, Thomas Hune, Oliver Möller, Elena Fersman, Carsten Weise, David Griffioen, Ansgar Fehnker, Frits Vandraager, Theo Ruys, Pedro D'Argenio, J-P Katoen, Jan Tretmans, Judi Romijn, Ed Brinksma, Martijn Hendriks,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

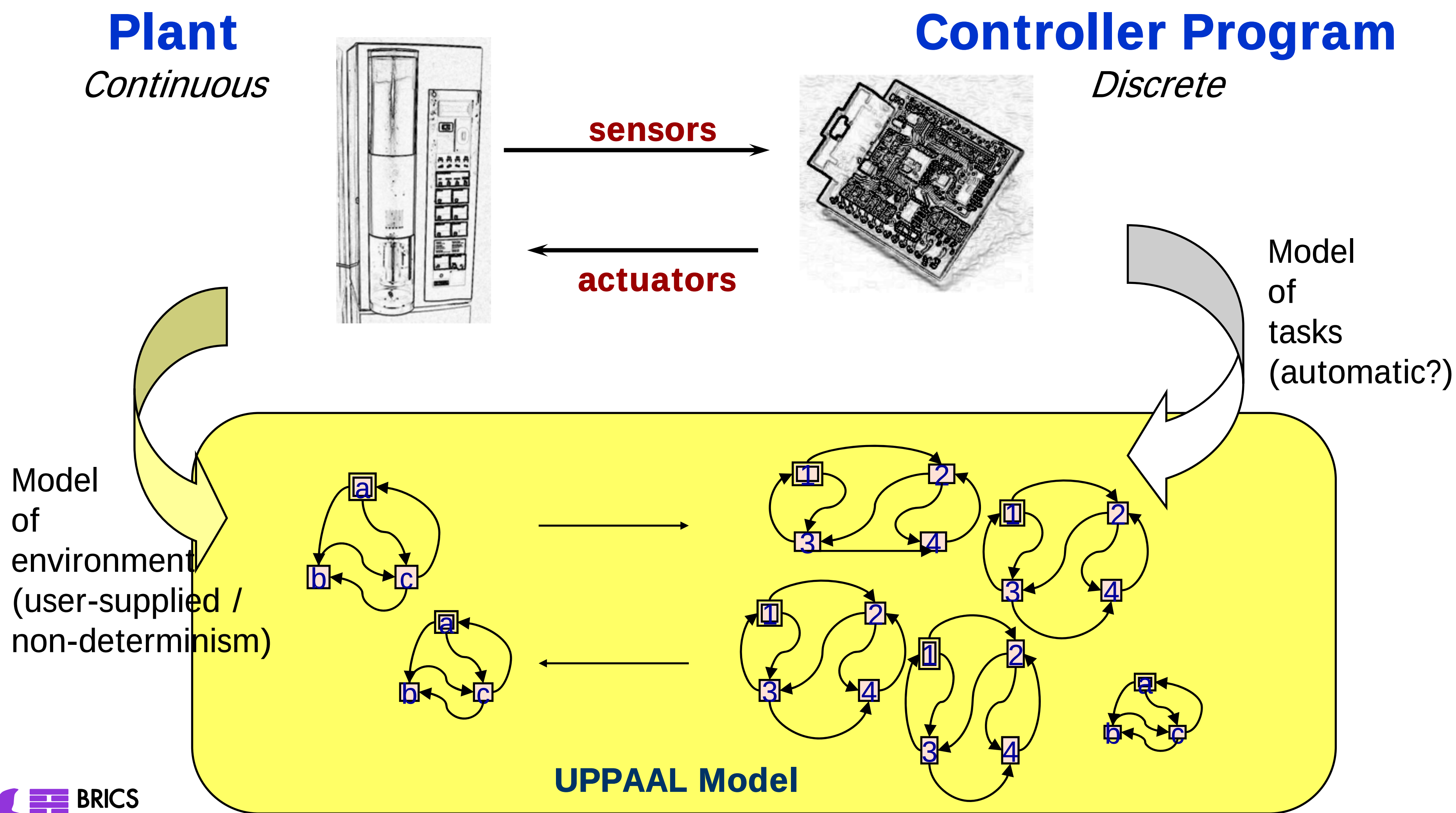
Controller Program

Discrete

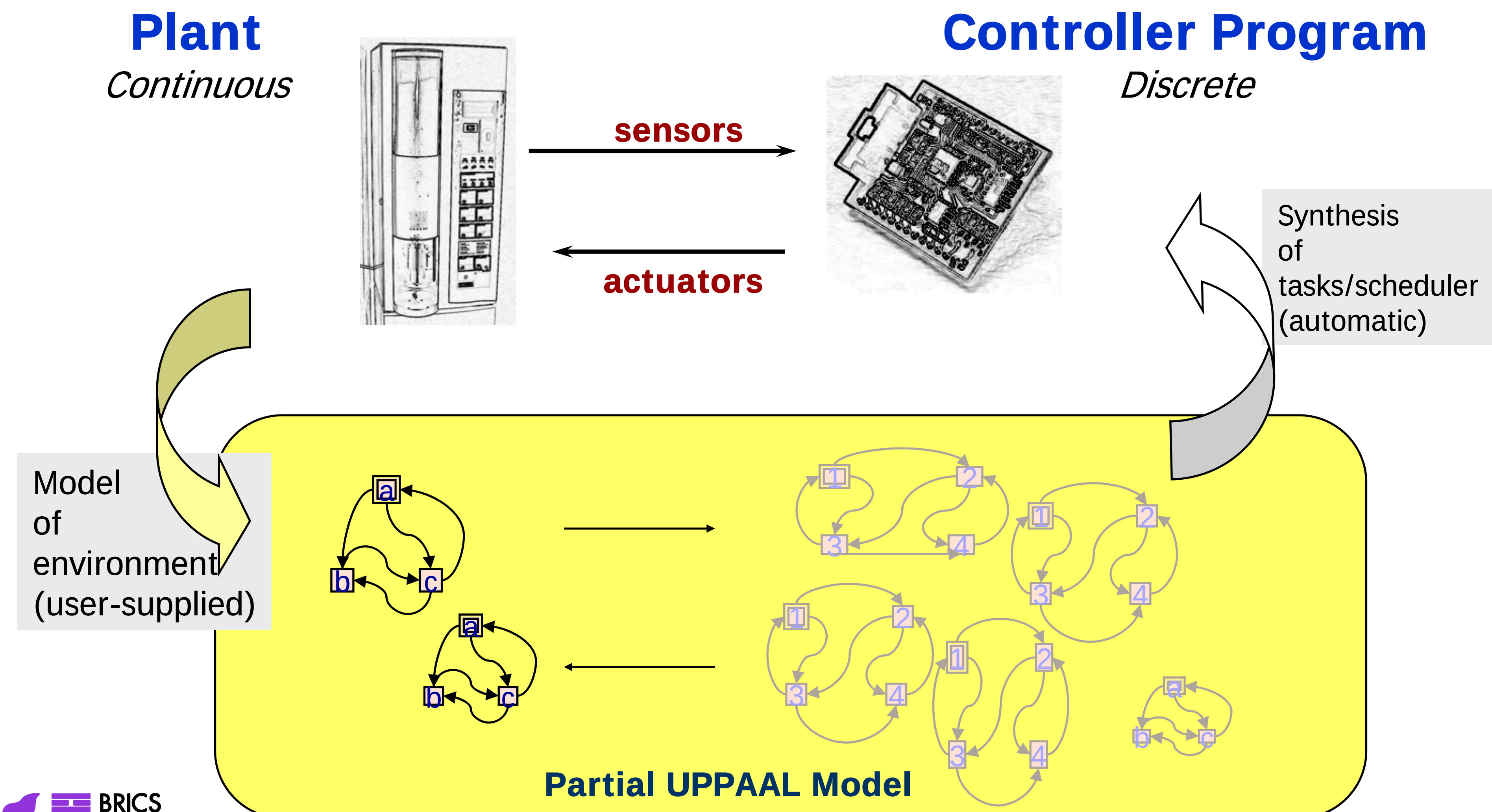
Eg.: Realtime Protocols
Pump Control
Air Bags
Robots
Cruise Control
ABS
CD Players
Production Lines

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

Real Time Model Checking

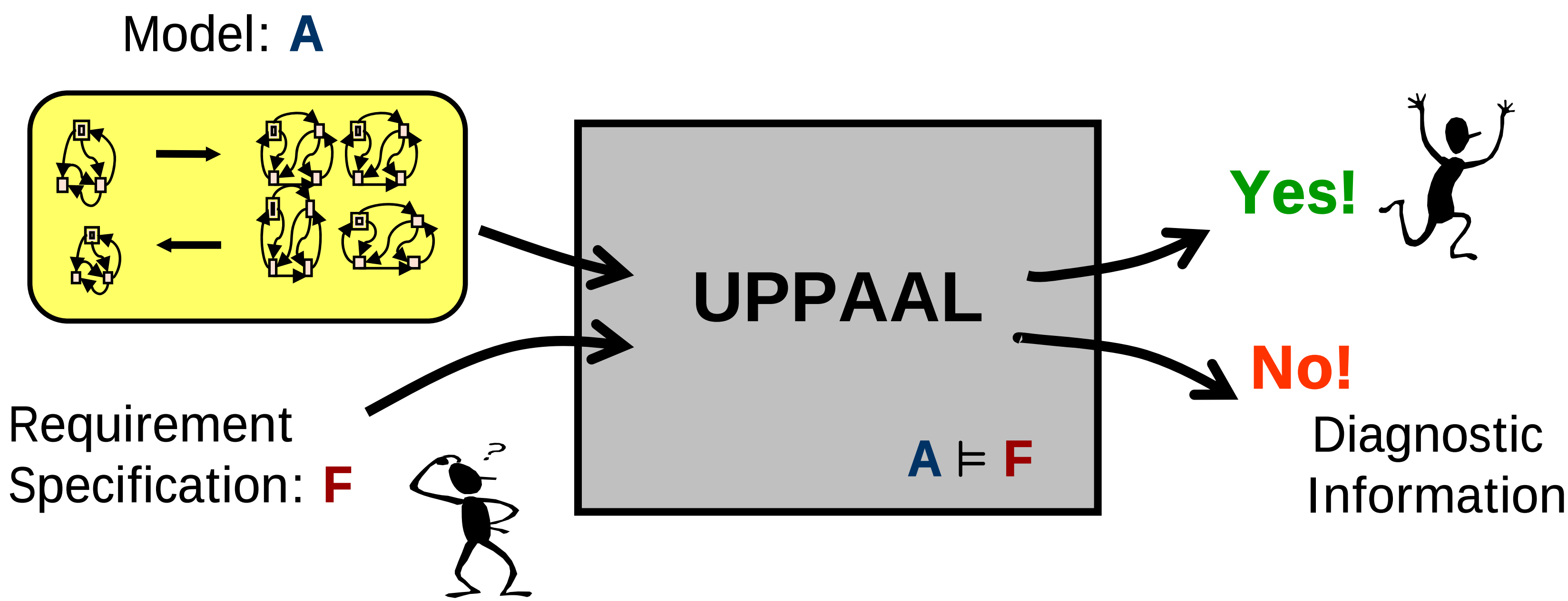


Real Time Control Synthesis



BRICS Basic Research in Computer Science

Model-Checking

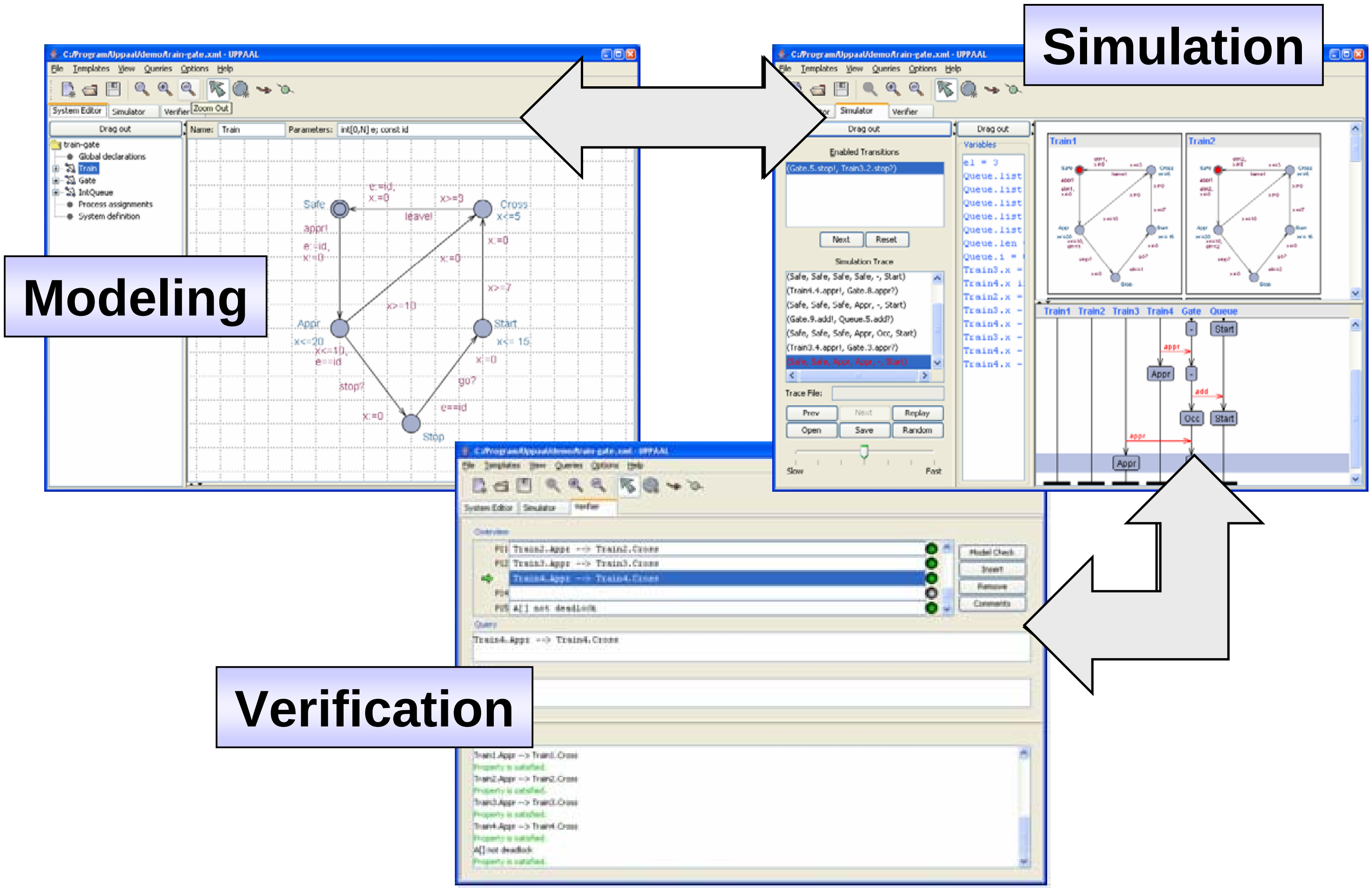


A – Model: Network of Timed Automata

F – Requirement: **T**CTL formula, e.g.:

- *Invariant*: something bad will never happen
- *Liveness*: something good will eventually happen
- *Bounded Liveness*: something good will happen before some upper time-bound T.

UPPAAL Tool



Timed Automata

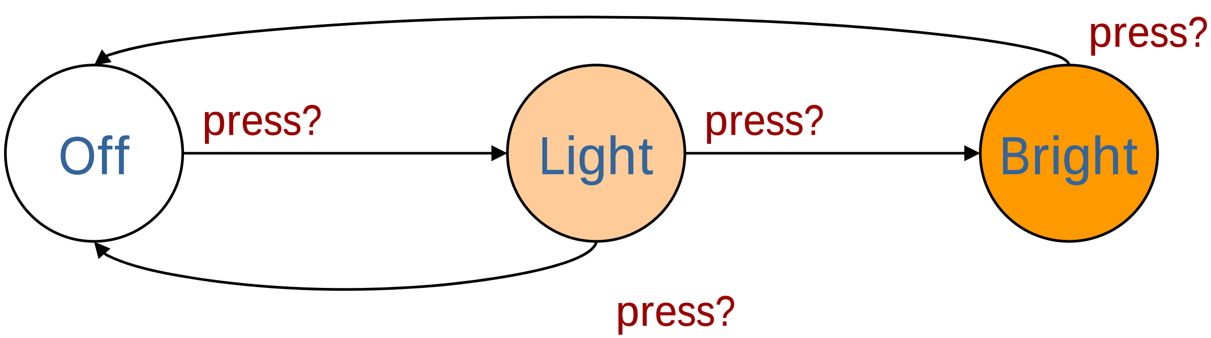
Alur & Dill 1989

BRICS
Basic Research
in Computer Science

CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

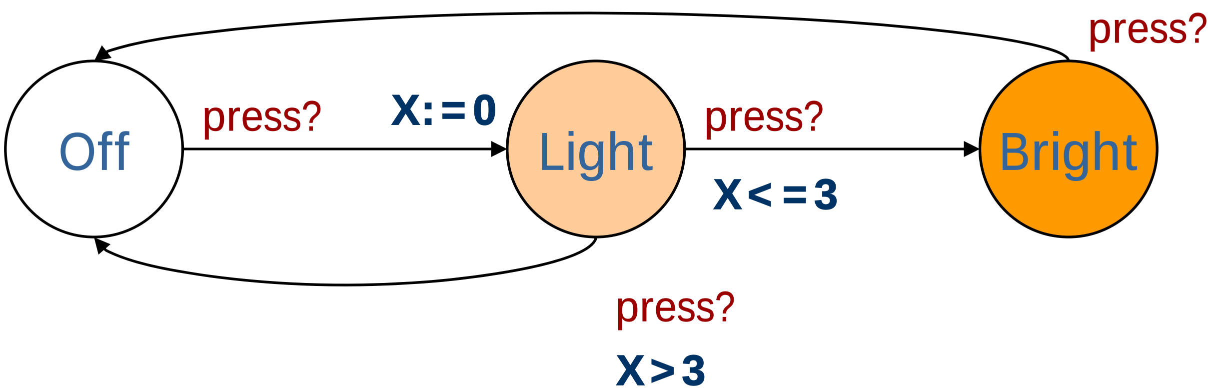
AALBORG UNIVERSITY DENMARK

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

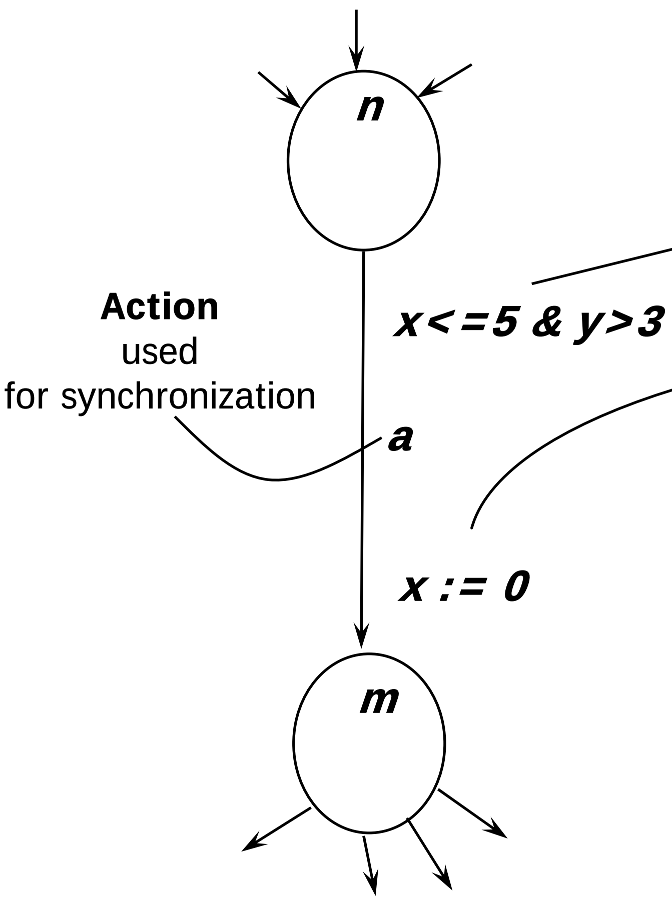


Solution: Add real-valued clock **x**

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

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

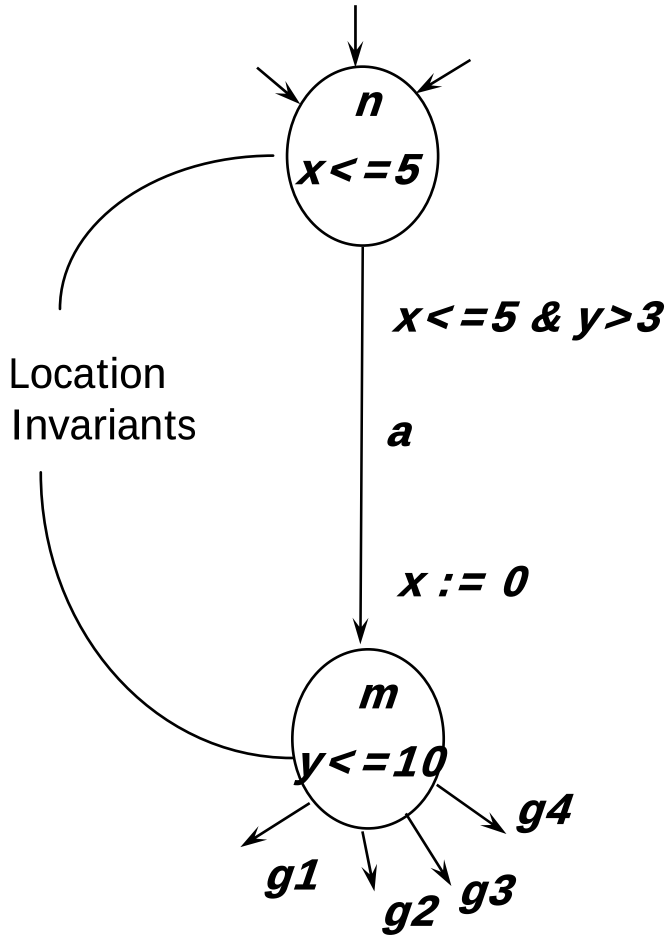
Discrete Trans
$(n , x{=}2.4 , y{=}3.1415) \xrightarrow{a} (m , x{=}0 , y{=}3.1415)$

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

BRICS
Basic Research
in Computer Science

Timed Automata *review*

Invariants



Clocks: ***x, y***

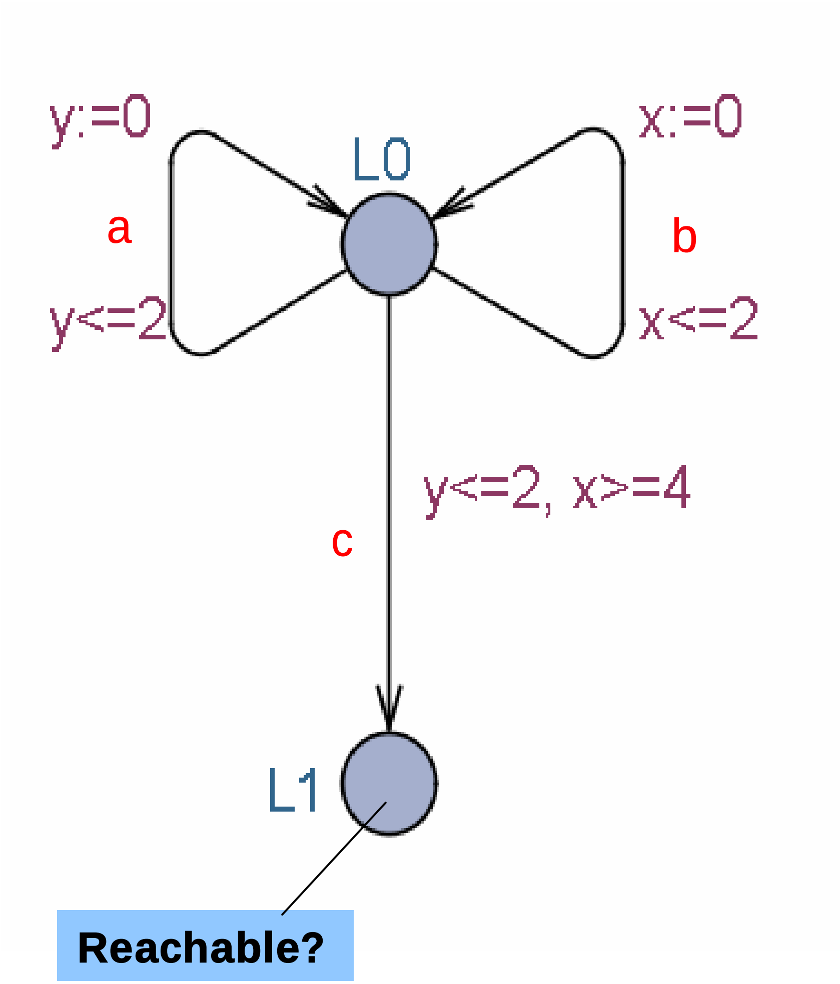
Transitions

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

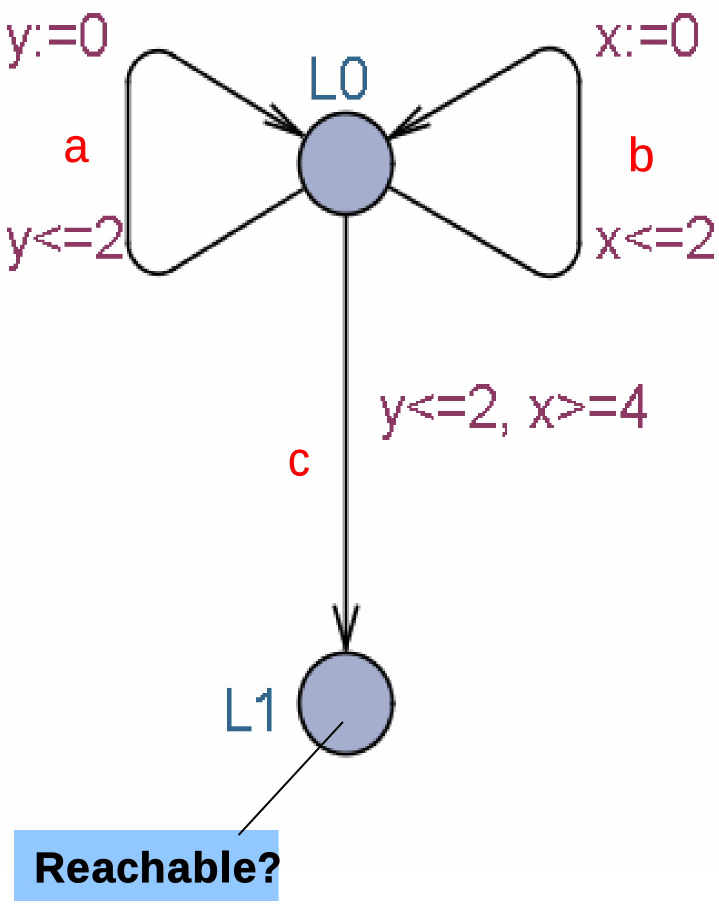
(***n*** , ***x***=2.4 , ***y***=3.1415) $\xrightarrow{e(1.1)}$ (***n*** , ***x***=3.5 , ***y***=4.2415)

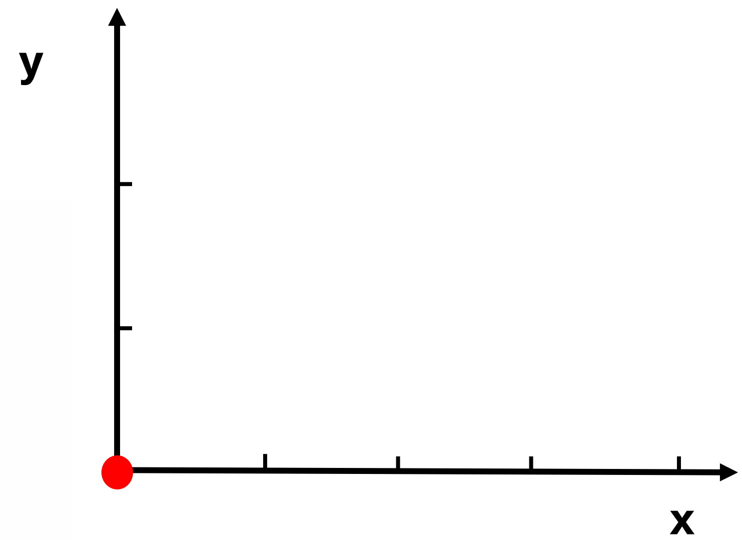
Invariants ensure progress!!

Example



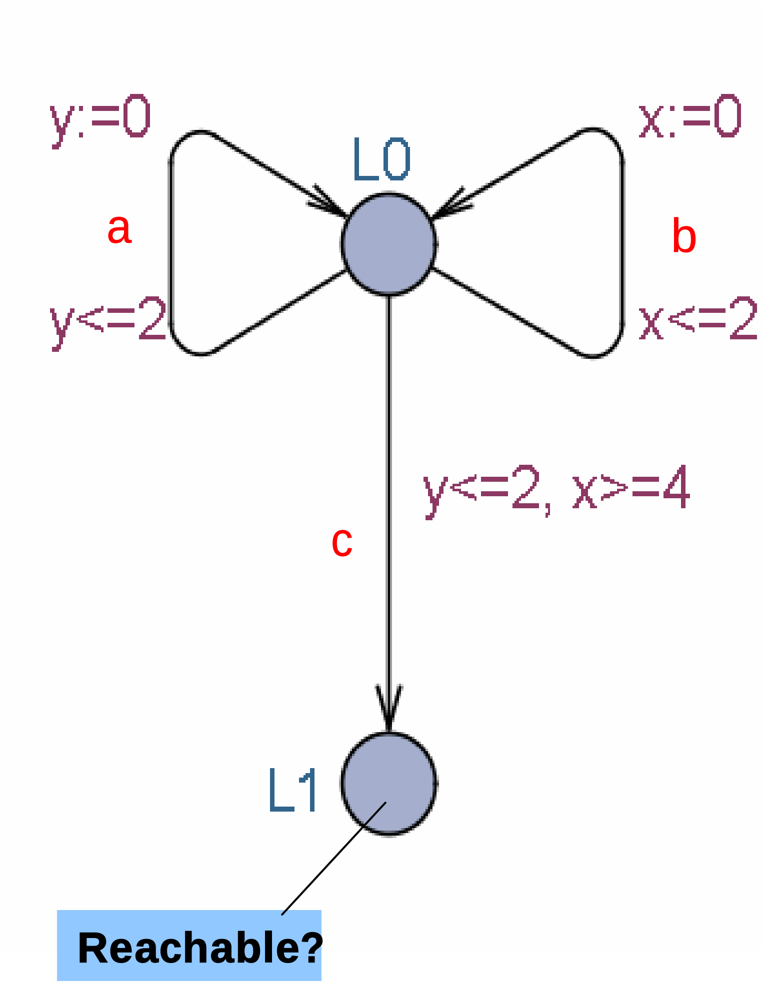
Example

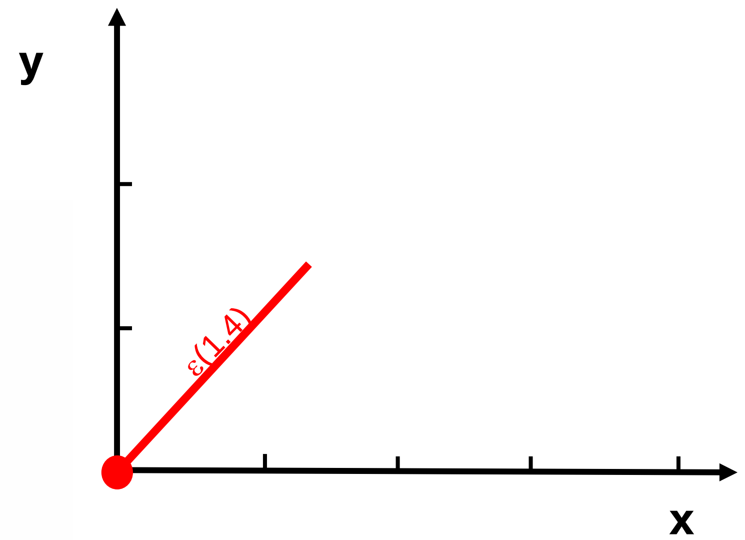




(L0,x=0,y=0)

Example



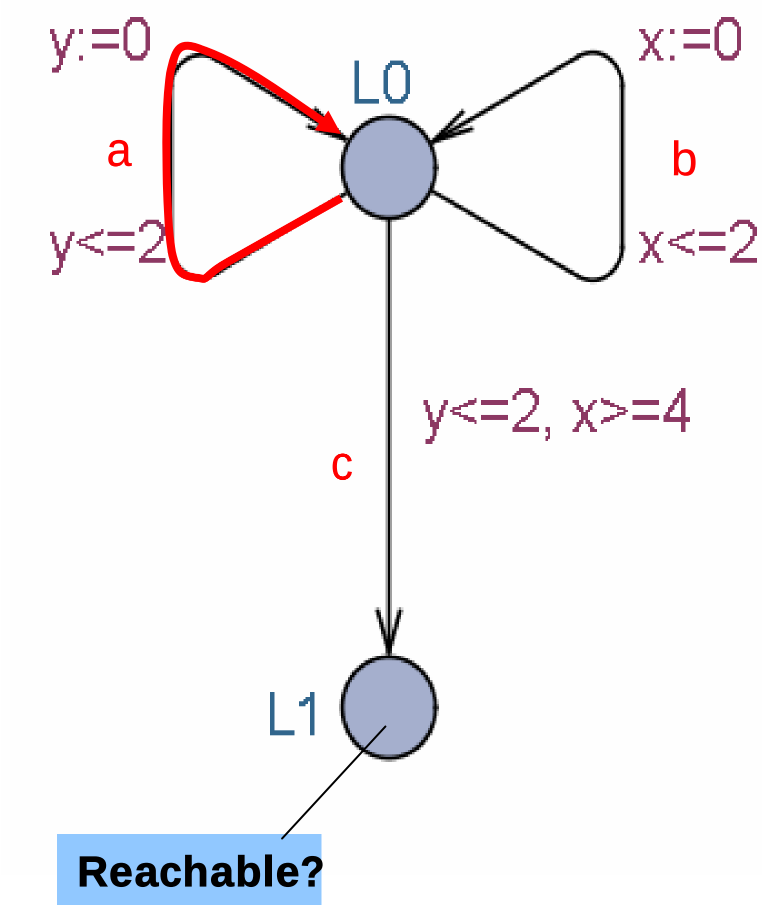


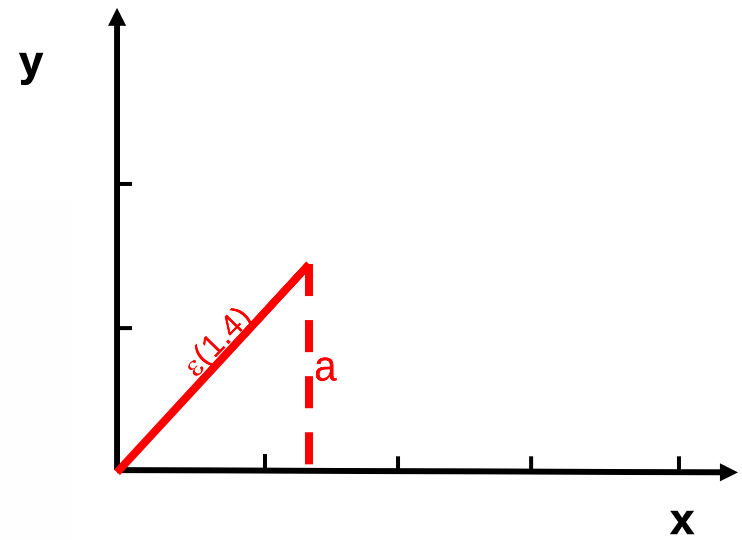
(L0,x=0,y=0)

$\rightarrow_{\varepsilon(1.4)}$

(L0,x=1.4,y=1.4)

Example





(L0,x=0,y=0)

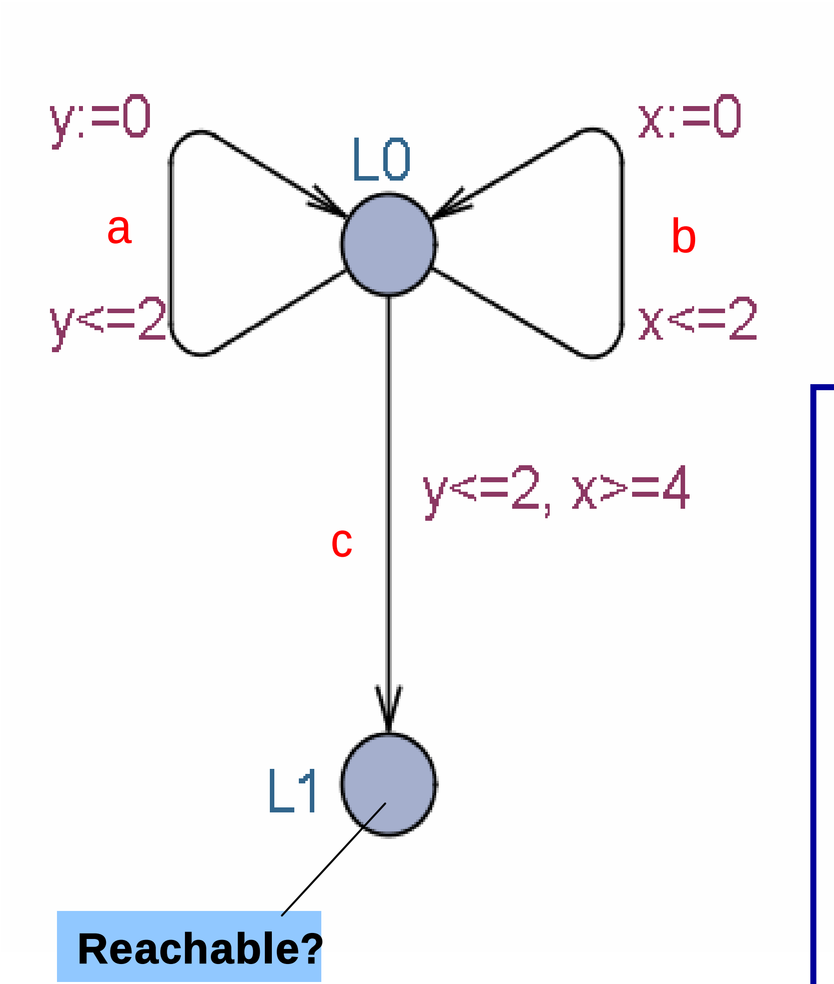
$\rightarrow_{\varepsilon(1.4)}$

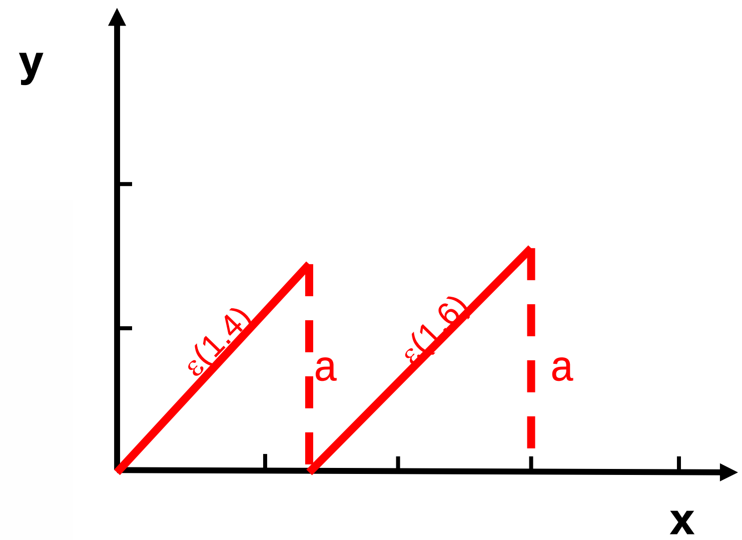
(L0,x=1.4,y=1.4)

$\rightarrow_{a}$

(L0,x=1.4,y=0)

Example





(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)

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

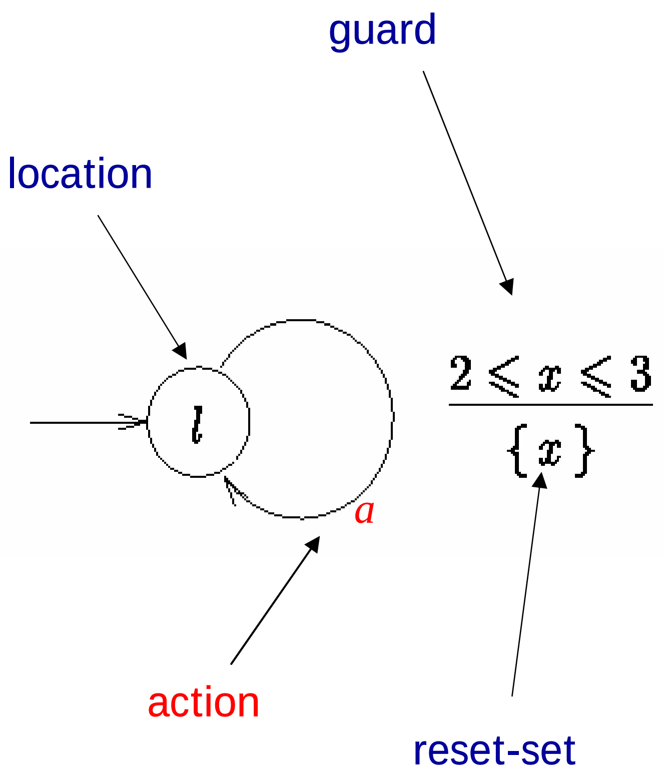
Semantics

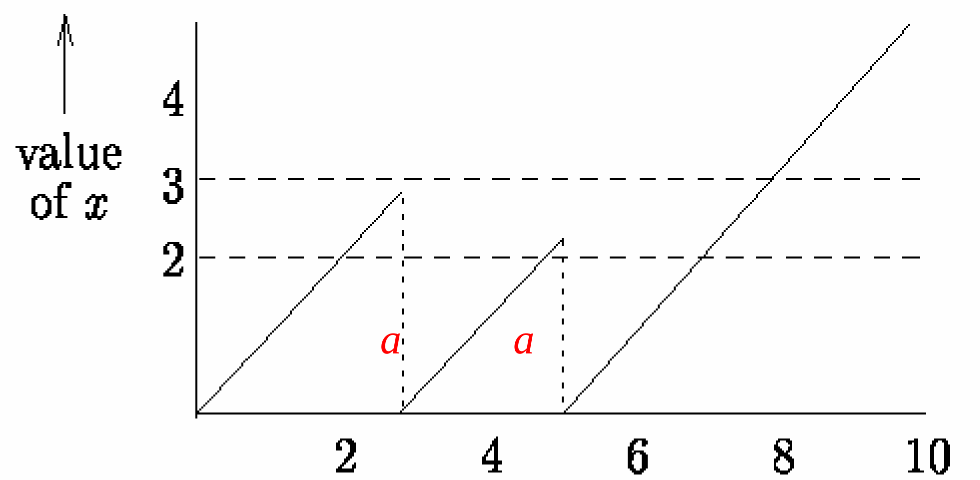
Definition
The semantics of a timed automaton A is a labelled transition system with state space $L \times \mathbb{R}^C$ with initial state (l_0, u_0)* and with the following transitions:

- $(l, u) \xrightarrow{\epsilon(d)} (l, u + d)$ 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
  - $u' \in I(l')$

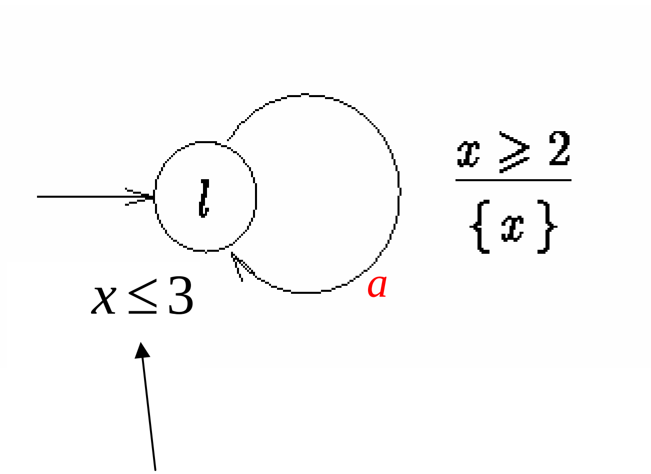
*$u_0(x) = 0$ for all $x \in C$

Timed Automata: Example

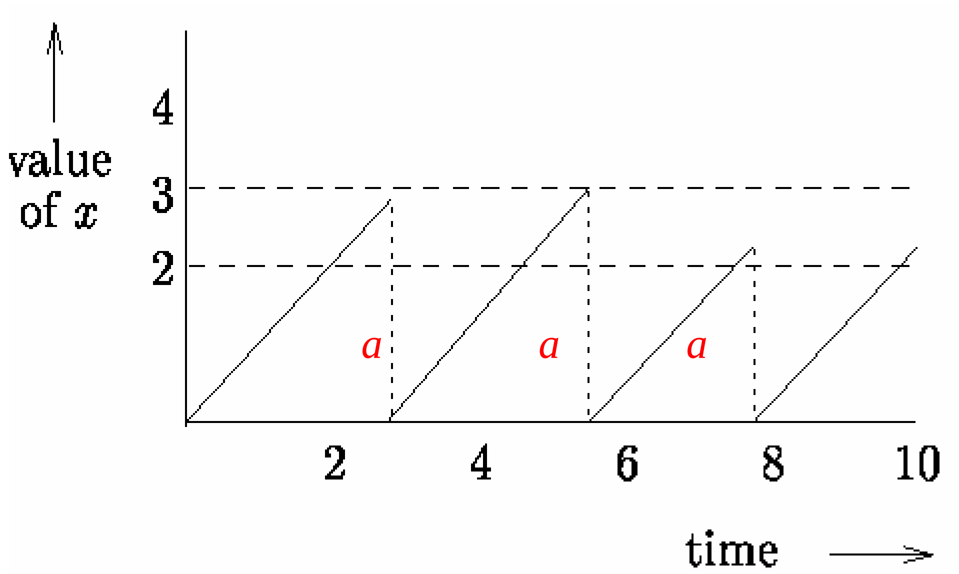




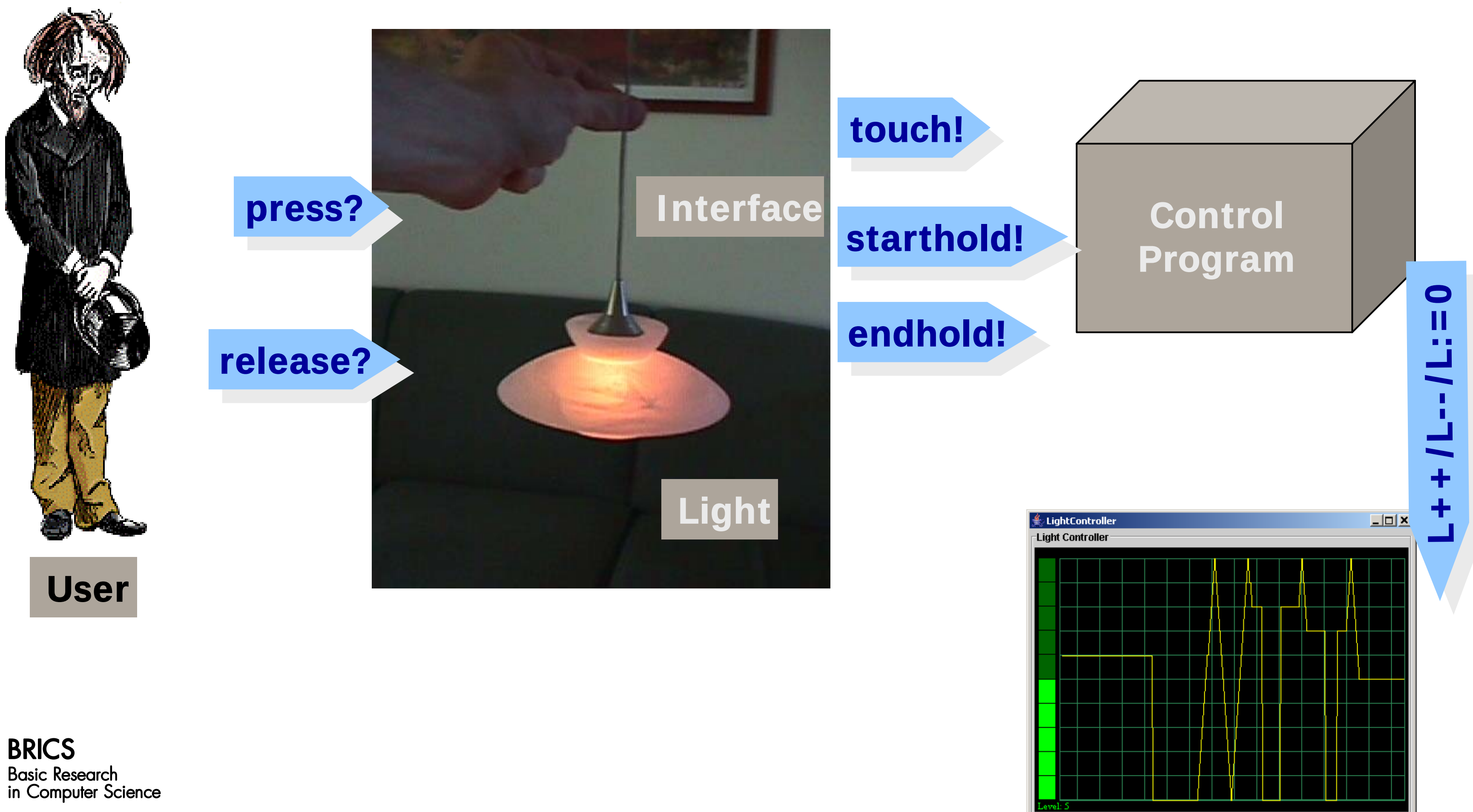
Timed Automata: Example



Invariant

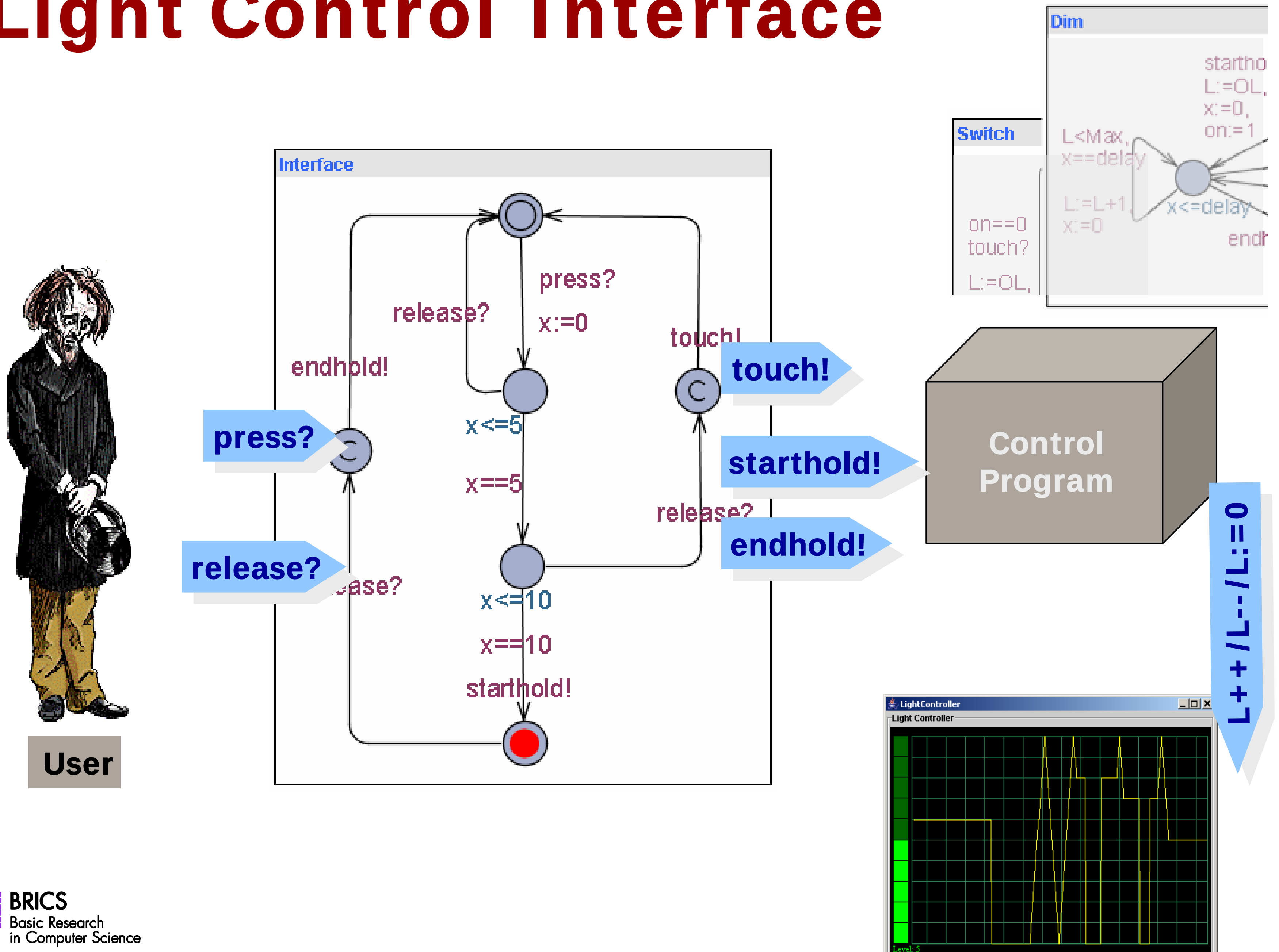


Light Control Interface



CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Light Control Interface

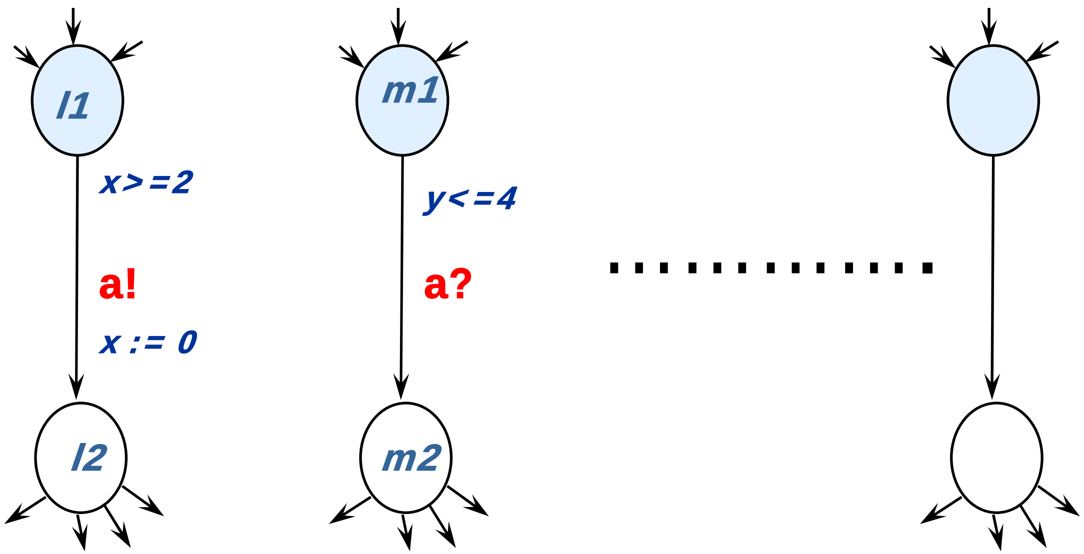


BRICS
Basic Research
in Computer Science

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

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, …..)*

~~0.2~~ (***l1***,***m1***,………,*x=2.2, y=3.7, …..)*

If **a** URGENT CHANNEL

BRICS
Basic Research
in Computer Science

Network Semantics

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

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

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

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

BRICS Basic Research in Computer Science

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

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

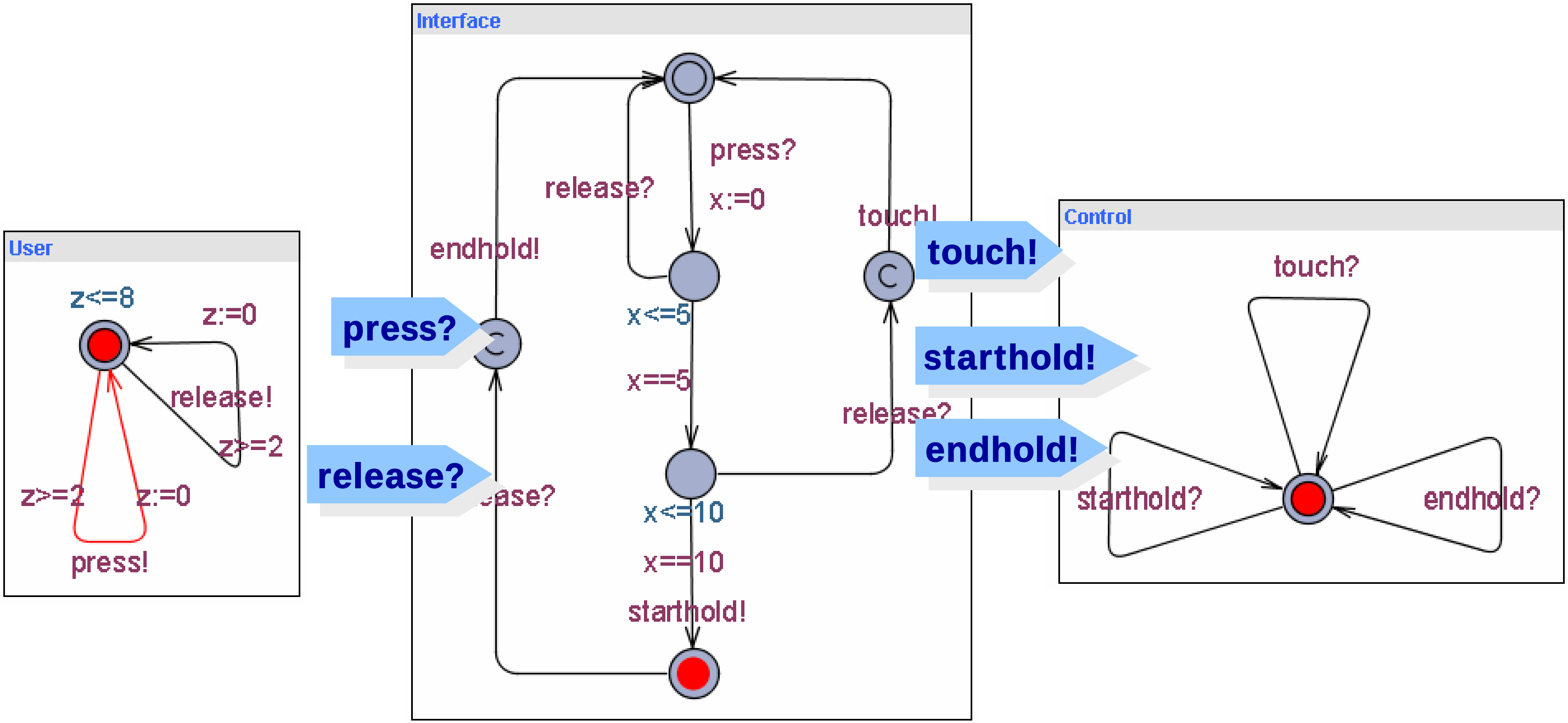
$$\frac{s_1 \xrightarrow{a!}_1 s_1' \qquad s_2 \xrightarrow{a?}_2 s_2'}{s_1 \|_X s_2 \xrightarrow{\tau} s_1' \|_X s_2'}$$

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

$\forall d' < d,\ \forall u \in$ UAct:

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

Light Control Network

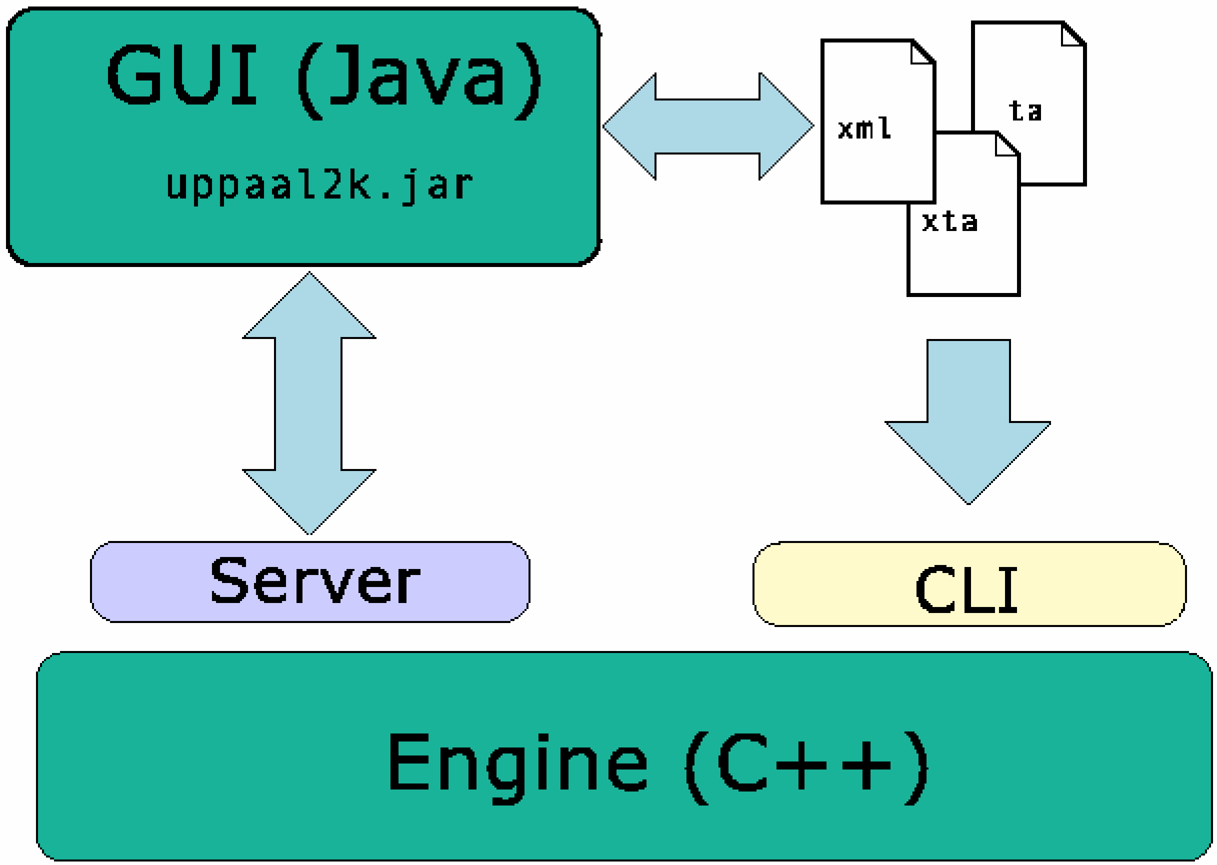


Overview of the UPPAAL Toolkit

BRICS
Basic Research
in Computer Science

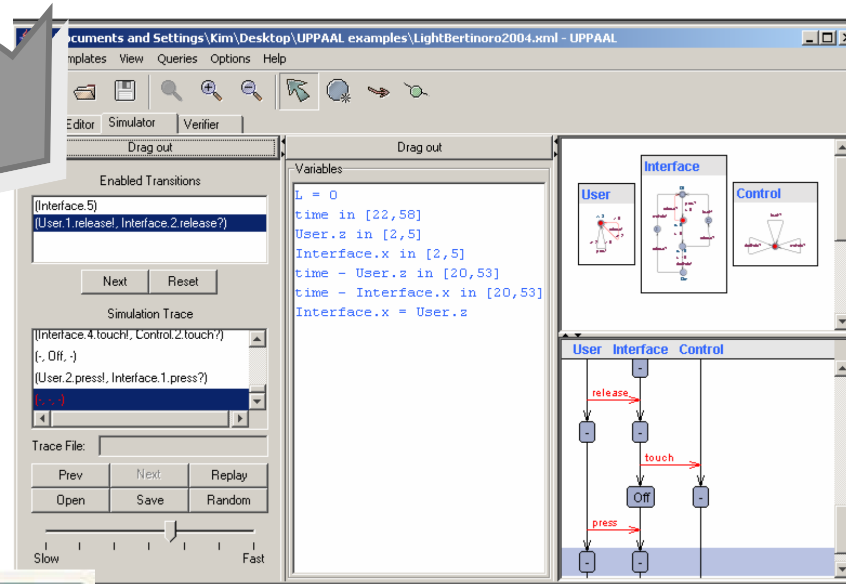
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

UPPAAL's architecture

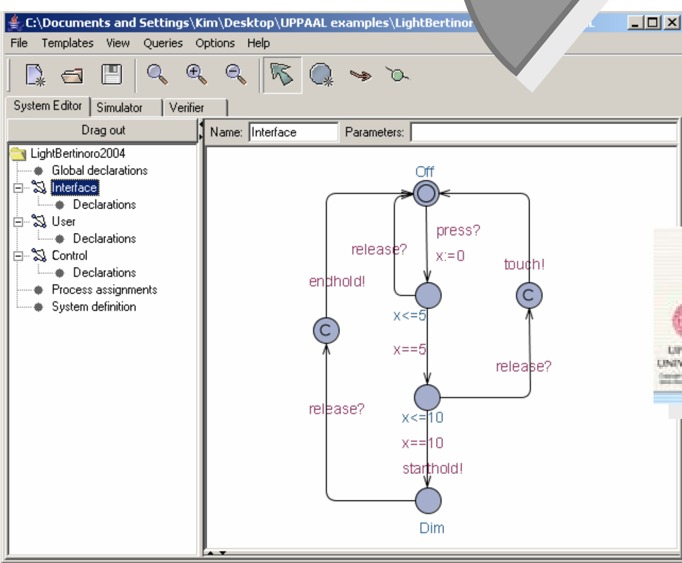


Linux, Windows, Solaris, MacOS

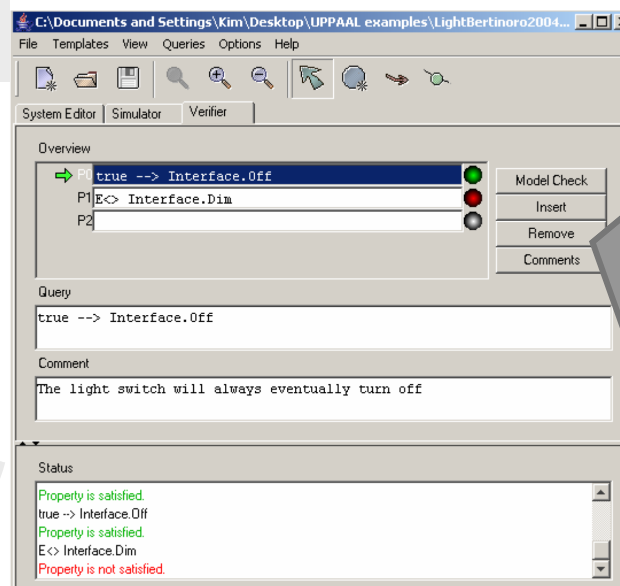
GUI



Simulator

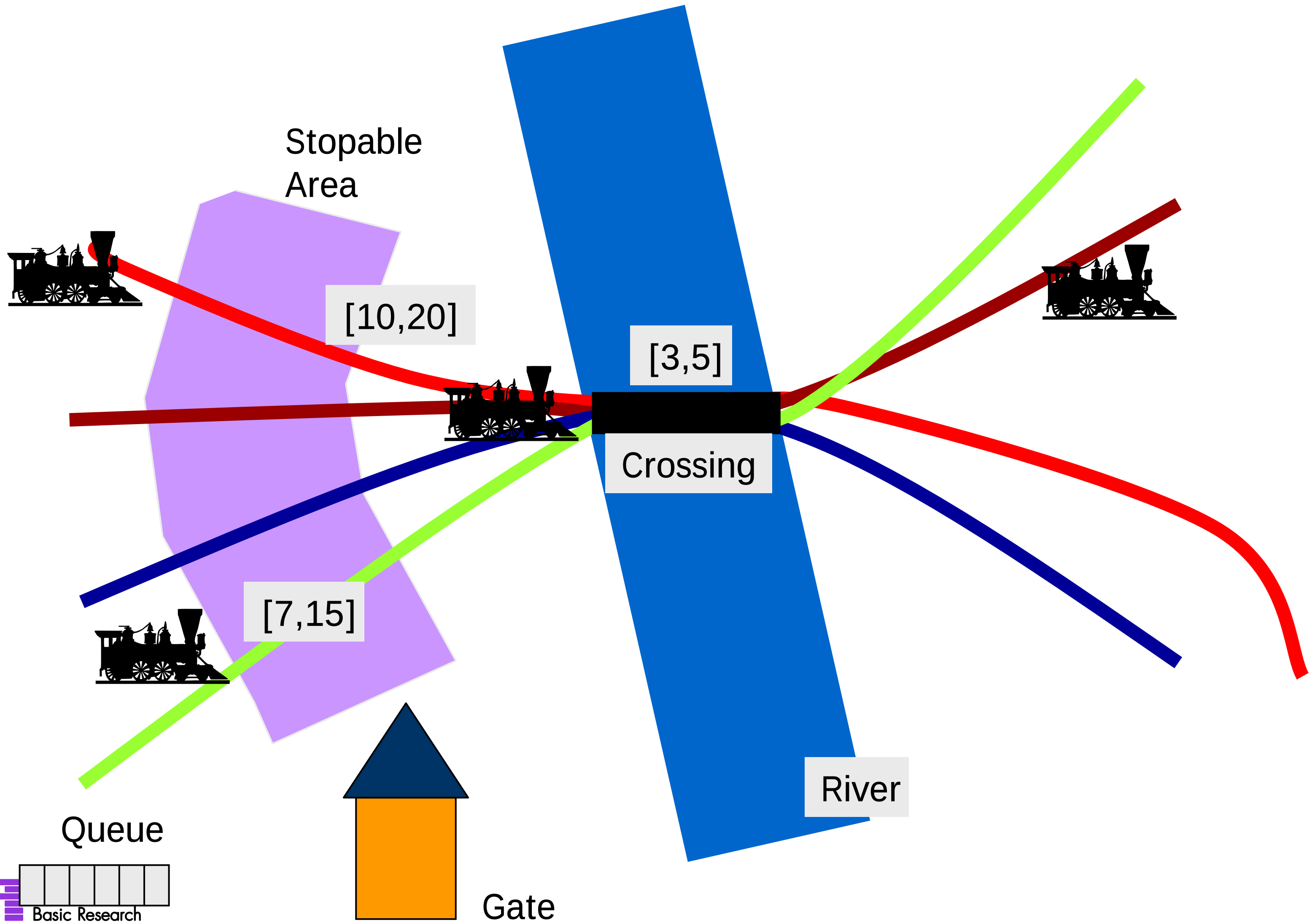


Editor



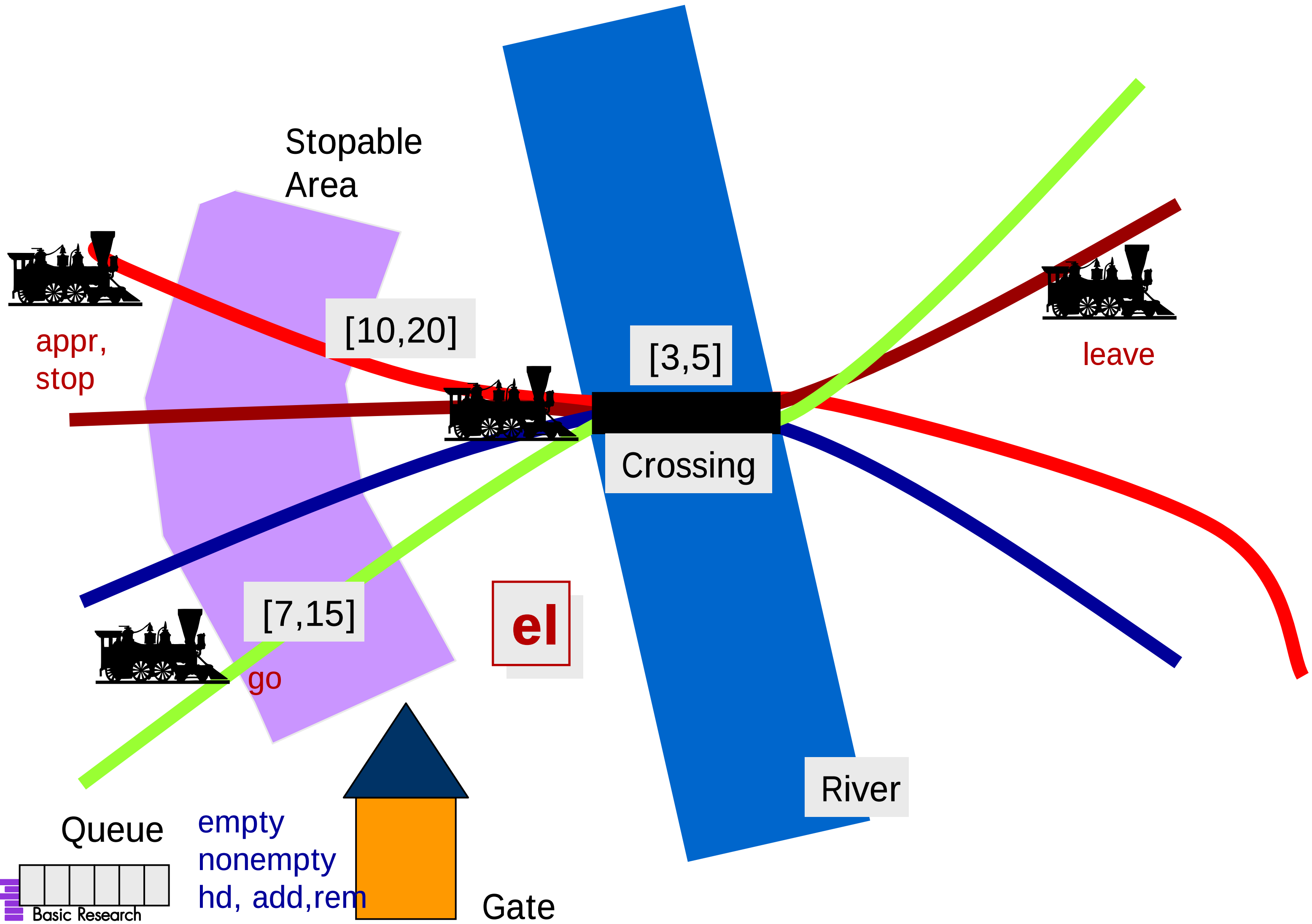
Verifier

Train Crossing



Train Crossing

Communication via channels and shared variable.



Basic Research
in Computer Science

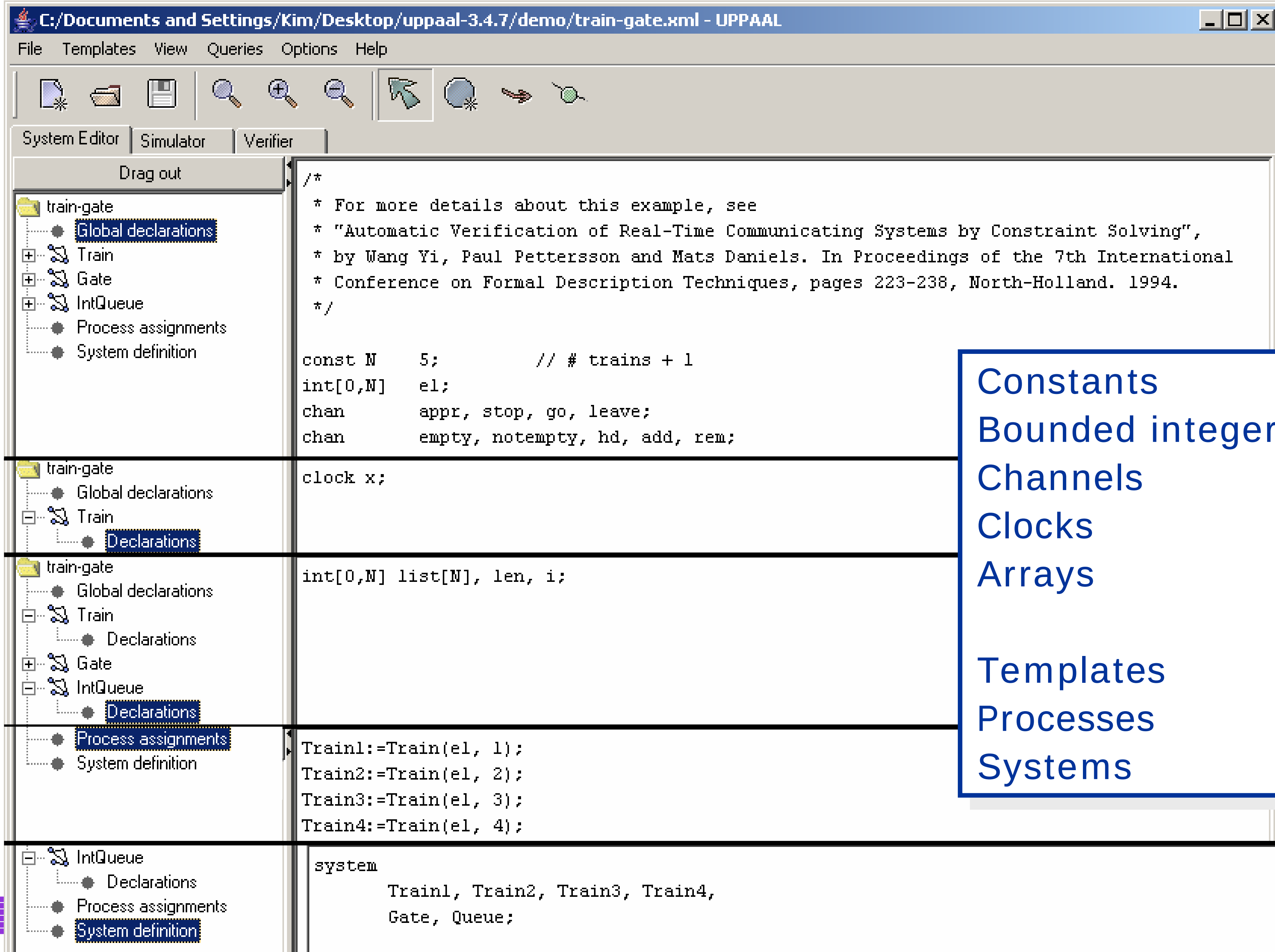
Timed Automata in UPPAAL

AALBORG UNIVERSITY DENMARK

BRICS Basic Research in Computer Science

CISS CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

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.**

BRICS
Basic Research
in Computer Science

Declarations in UPPAAL (cont.)

- **Data variables**
  - Syntax:

– `int n1, … ;`	**Integer with "default" domain.**
– `int[l,u] n1, … ;`	**Integer with domain "l" to "u".**
– `int n1[m], … ;`	**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;`

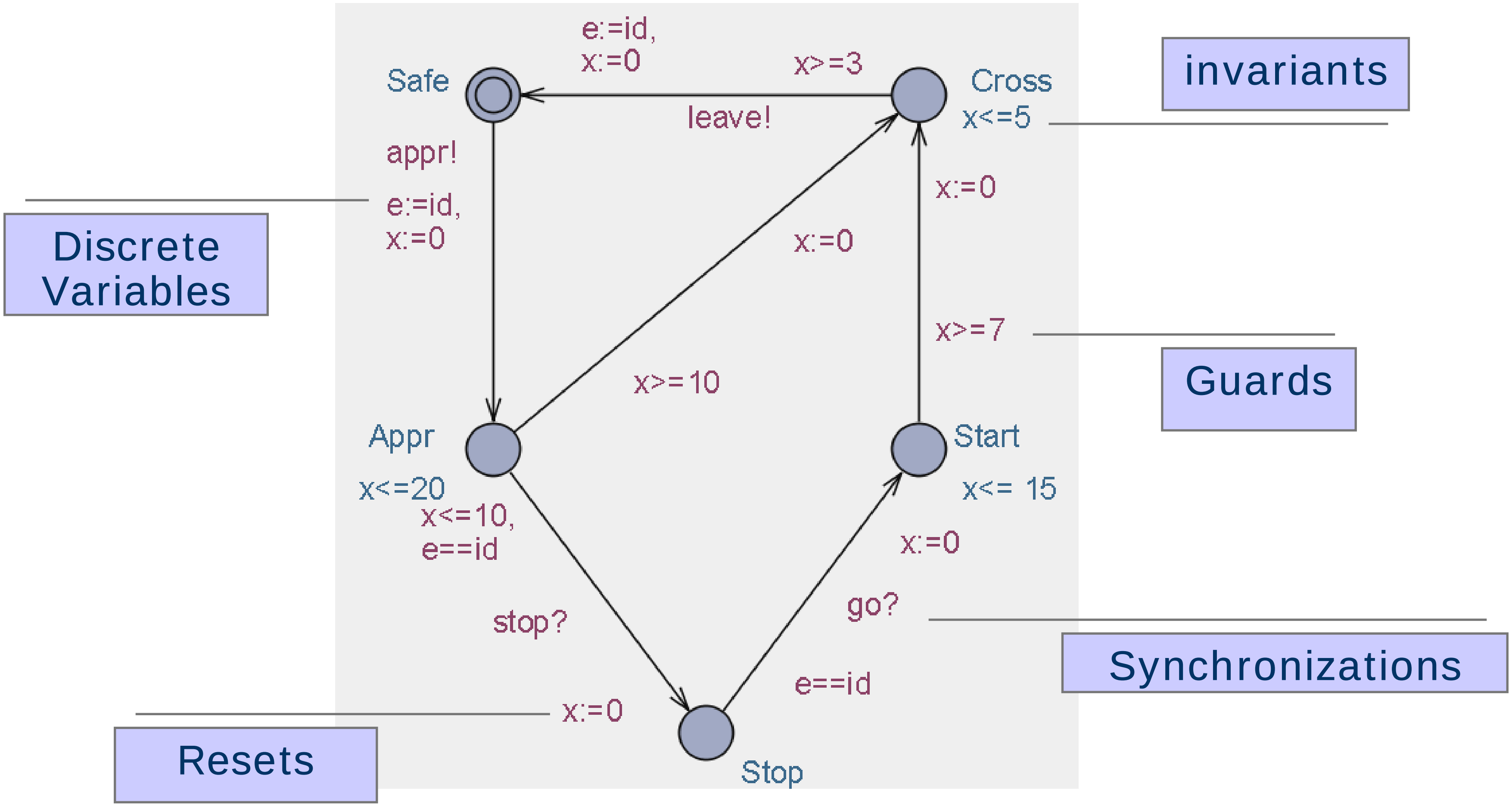
Declarations UPPAAL (cont.)

- **Constants**
  - Syntax:

```
– const int c1 = n1;
```

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

Timed Automata in UPPAAL

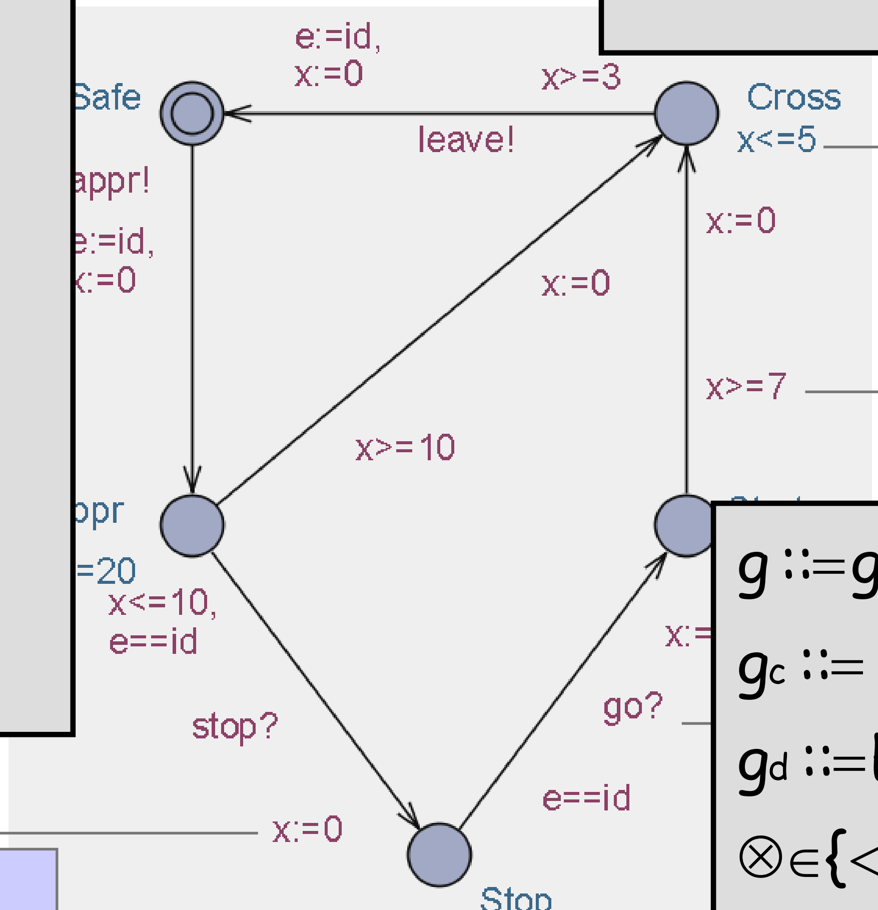


Timed Automata in UPPAAL

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

invariants

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



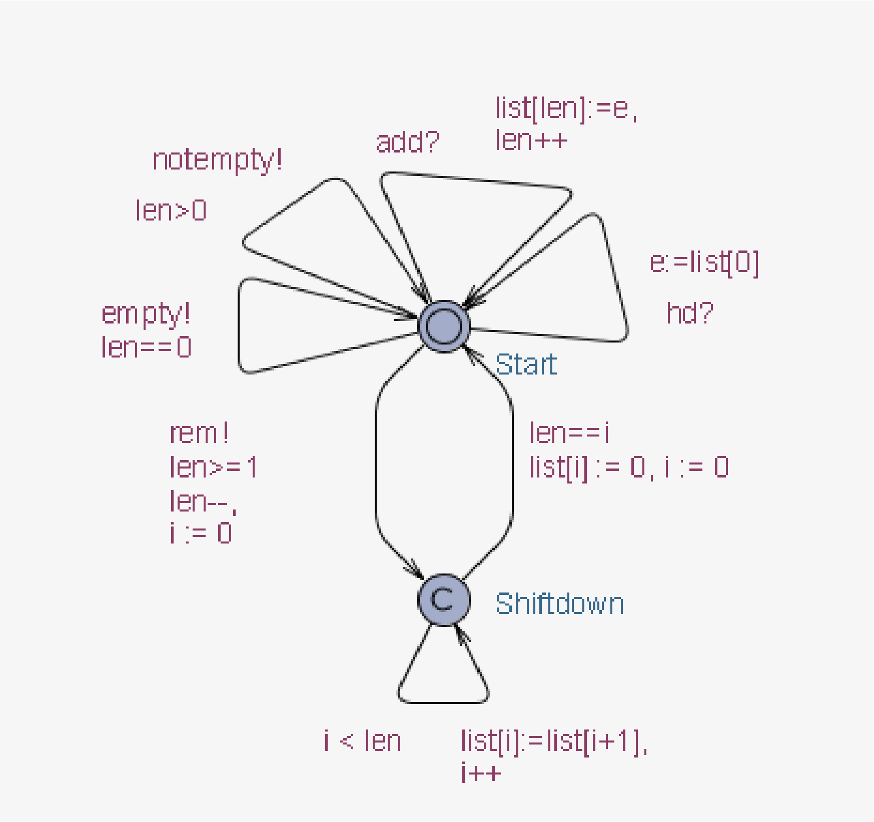
Guards

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

Resets

$$x := Expr$$

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
```

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

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

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

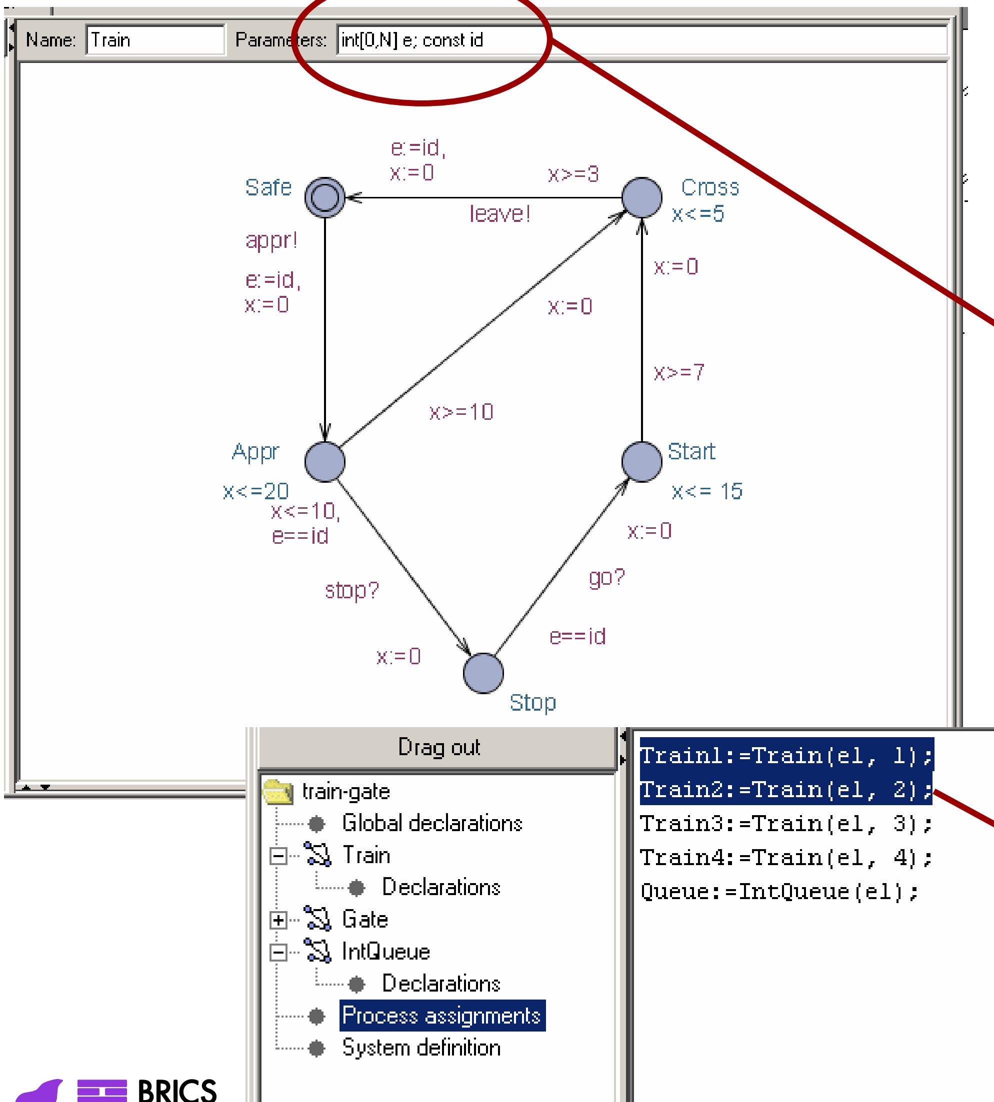
- Declared like
  `broadcast chan a, b, c[2];`
- If a is a broadcast channel:
  - a! = Emmision of broadcast
  - a? = Reception of broadcast

- 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

TCTL:
Timed Computational Tree Logic

TCTL = CTL + Time

$$\phi ::= p \mid \alpha \mid \neg\phi \mid \phi \vee \phi \mid z \text{ in } \phi \mid \mathrm{E}[\phi \,\mathrm{U}\, \phi] \mid \mathrm{A}[\phi \,\mathrm{U}\, \phi]$$

$p \in AP$, *automic propositions*,

$z \in D$, *formula clocks*,

α – *constraints over formula clocks and automata clock*

$z \text{ in } \phi$ – *"freeze operator" introduces new formula clock z*

E[ϕ U ϕ], A[ϕ U ϕ] - *like in CTL*

No EX ϕ

Derived Operators

$$\boxed{\mathsf{A}[\phi\, \mathsf{U}_{\leqslant 7}\, \psi]} \quad = \quad z \text{ in } \mathsf{A}[(\phi \wedge z \leqslant 7)\, \mathsf{U}\, \psi].$$

Along any path ϕ holds continuously until within 7 time units ψ becomes valid.

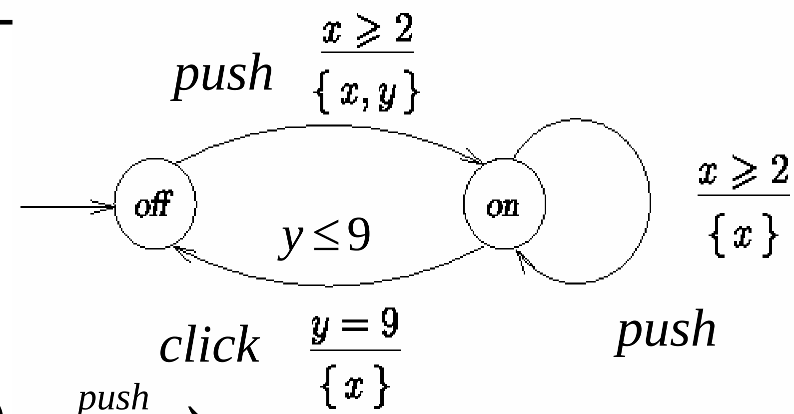
$$\boxed{\mathsf{EF}_{<5}\, \phi} \quad = \quad z \text{ in } \mathsf{EF}\,(z < 5 \wedge \phi)$$

The property ϕ becomes valid within 5 time units.

Paths

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

Example:



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

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

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

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

Elapsed time in path

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

Example:

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

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

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

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

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

TCTL Semantics

$s, w \models p$ iff $p \in \mathit{Label}(s)$

$s, w \models \alpha$ iff $v \cup w \models \alpha$

$s, w \models \neg \phi$ iff $\neg (s, w \models \phi)$

$s, w \models \phi \vee \psi$ iff $(s, w \models \phi) \vee (s, w \models \psi)$

$s, w \models z \text{ in } \phi$ iff $s, \text{reset } z \text{ in } w \models \phi$

$s, w \models \mathsf{E}[\phi \, \mathsf{U} \, \psi]$ iff $\exists \sigma \in P^{\infty}_{\mathcal{M}}(s). \exists (i, d) \in \mathit{Pos}(\sigma).$
$(\sigma(i, d), w{+}\Delta(\sigma, i) \models \psi \;\wedge$
$(\forall (j, d') \ll (i, d). \sigma(j, d'), w{+}\Delta(\sigma, j) \models \phi \vee \psi))$

$s, w \models \mathsf{A}[\phi \, \mathsf{U} \, \psi]$ iff $\forall \sigma \in P^{\infty}_{\mathcal{M}}(s). \exists (i, d) \in \mathit{Pos}(\sigma).$
$((\sigma(i, d), w{+}\Delta(\sigma, i)) \models \psi \;\wedge$
$(\forall (j, d') \ll (i, d). (\sigma(j, d'), w{+}\Delta(\sigma, j)) \models \phi \vee \psi)).$

s - location

w - formula clock valuation

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

$Pos(\sigma)$ - positions in σ

$\Delta(\sigma,i)$ - elapsed time

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

Timeliness Properties

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

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

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

receive(m) occurs exactly 11 time units after *send(m)*

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

putbox occurs periodically (exactly) every 25 time units

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

UPPAAL
Specification Language

Aalborg University Denmark

BRICS
Basic Research in Computer Science

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

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: $P \rightarrow_{\leq t} Q$

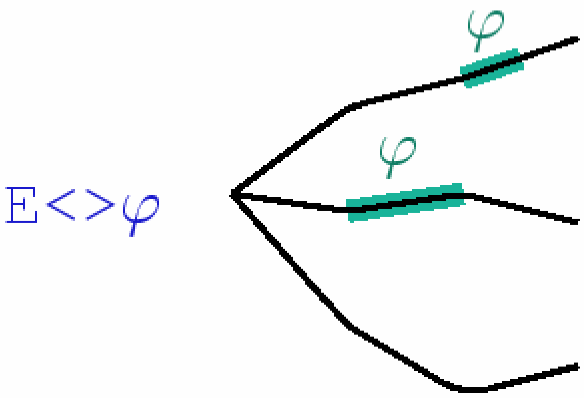
The expressions P and Q must be type safe, side effect free, and evaluate to a boolean.

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

BRICS Basic Research in Computer Science

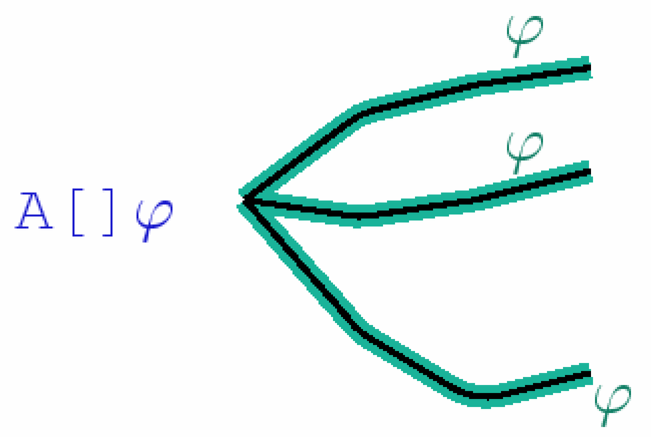
Logical Specifications

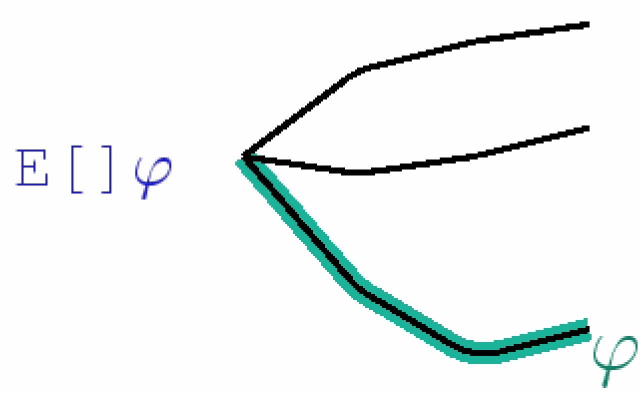
- Validation Properties
  - Possibly: E<> P
- Safety Properties
  - Invariant: A[] P
  - Pos. Inv.: E[] P
- Liveness Properties
  - Eventually: A<> P
  - Leadsto: P → Q
- Bounded Liveness
  - Leads to within: $P \rightarrow_{\leq t} Q$



Logical Specifications

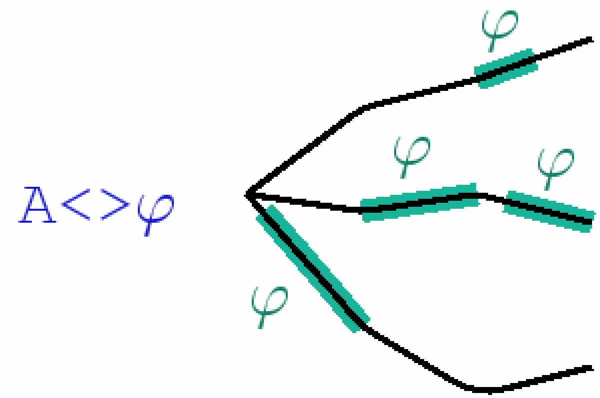
- Validation Properties
  - Possibly: E<> *P*
- Safety Properties
  - Invariant: A[] *P*
  - Pos. Inv.: E[] *P*
- Liveness Properties
  - Eventually: A<> *P*
  - Leadsto: P → *Q*
- Bounded Liveness
  - Leads to within: $P \rightarrow_{\leq t} Q$

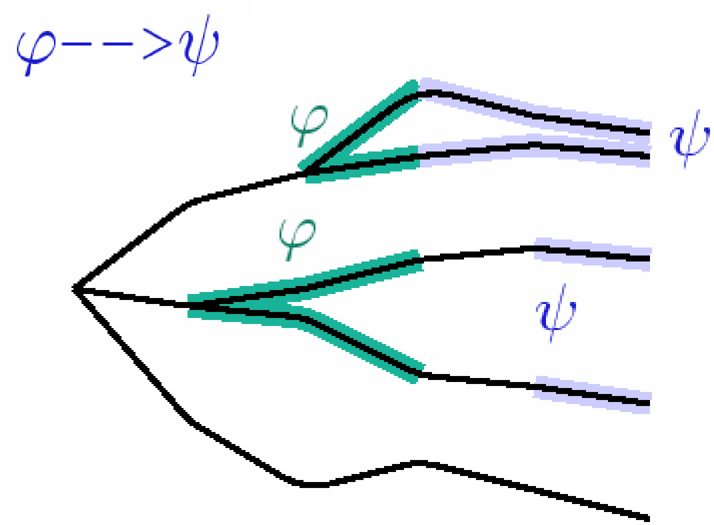




Logical Specifications

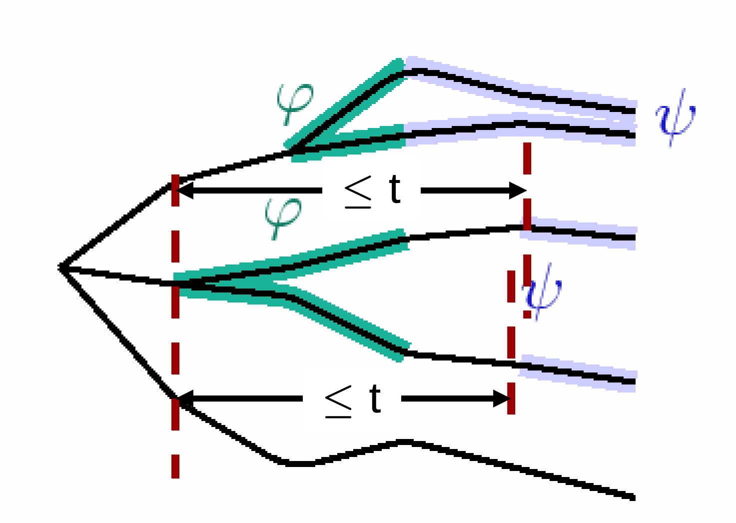
- Validation Properties
  - Possibly: E<> *P*
- Safety Properties
  - Invariant: A[] *P*
  - Pos. Inv.: E[] *P*
- Liveness Properties
  - Eventually: A**<>** *P*
  - Leadsto: P → *Q*
- Bounded Liveness
  - Leads to within: $P \to_{\leq t} Q$





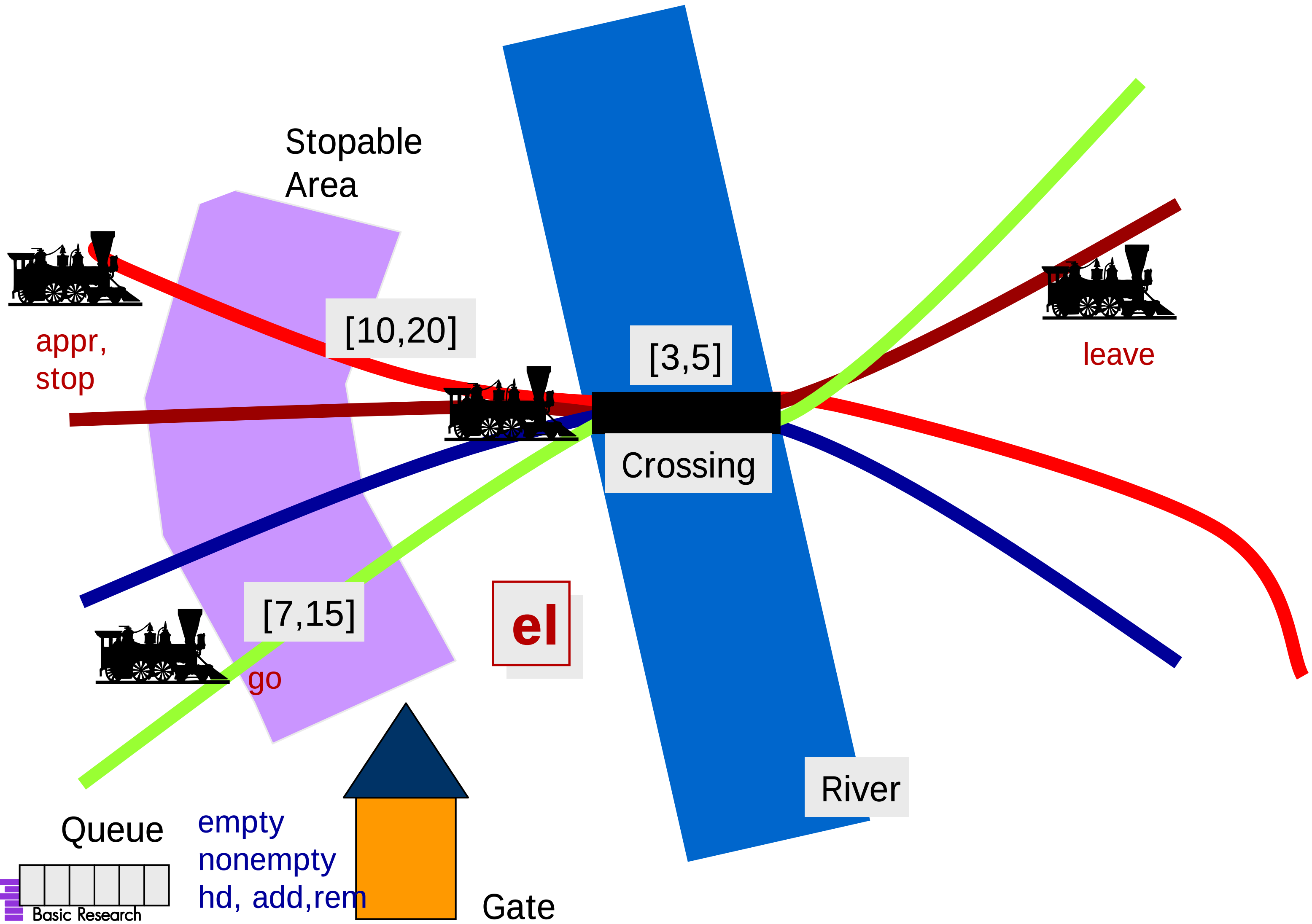
Logical Specifications

- 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: $P \rightarrow_{\leq t} Q$



Train Crossing

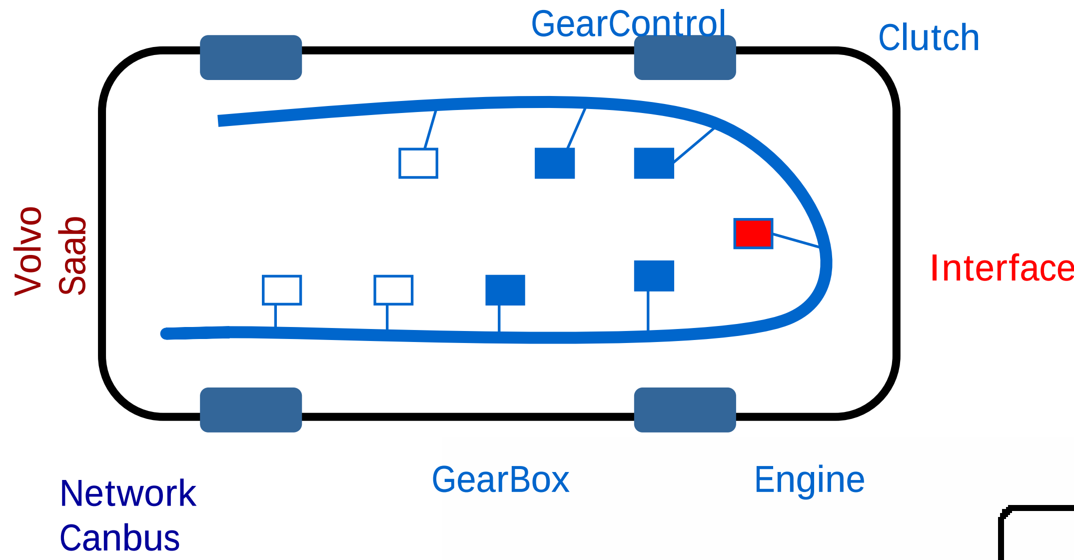
Communication via channels and shared variable.



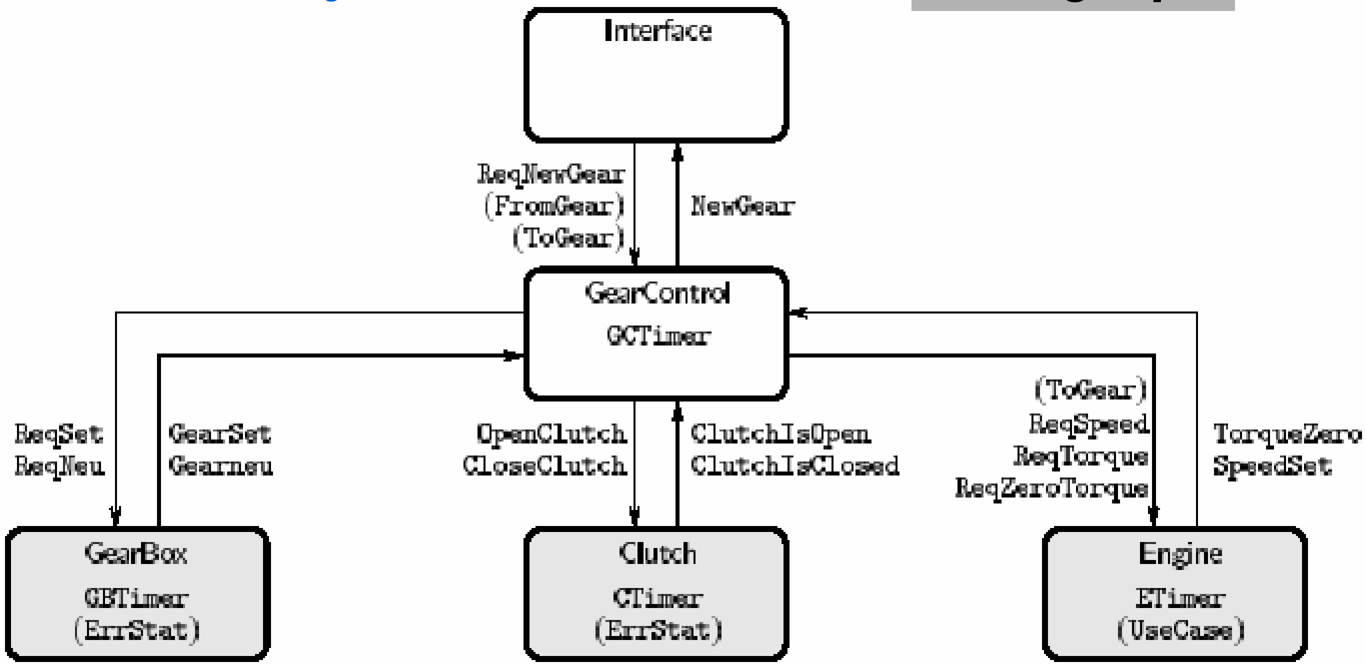
Gear Controller

with MECEL AB

Lindahl, Pettersson, Yi 1998

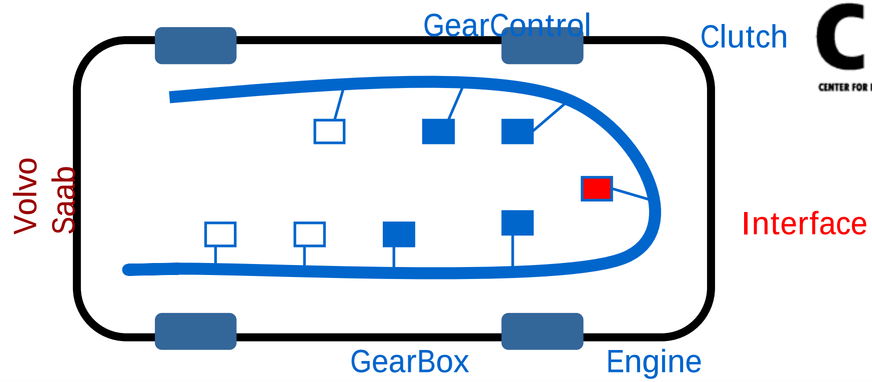


Flowgraph



Gear Controller

with MECEL AB



Requirements

$$\text{GearControl@Initiate} \leadsto_{\leq 1500} (\,(\,\mathtt{ErrStat} = 0\,) \Rightarrow \text{GearControl@GearChanged}\,) \quad (1)$$

$$\text{GearControl@Initiate} \leadsto_{\leq 1000} (\,(\,\mathtt{ErrStat} = 0 \wedge \mathtt{UseCase} = 0\,) \Rightarrow \text{GearControl@GearChanged}\,) \quad (2)$$

$$\text{Clutch@ErrorClose} \leadsto_{\leq 200} \text{GearControl@CCloseError} \quad (3)$$

$$\text{Clutch@ErrorOpen} \leadsto_{\leq 200} \text{GearControl@COpenError} \quad (4)$$

$$\text{GearBox@ErrorIdle} \leadsto_{\leq 350} \text{GearControl@GSetError} \quad (5)$$

$$\text{GearBox@ErrorNeu} \leadsto_{\leq 200} \text{GearControl@GNeuError} \quad (6)$$

$$Inv\,(\,\text{GearControl@CCloseError} \Rightarrow \text{Clutch@ErrorClose}\,) \quad (7)$$

$$Inv\,(\,\text{GearControl@COpenError} \Rightarrow \text{Clutch@ErrorOpen}\,) \quad (8)$$

$$Inv\,(\,\text{GearControl@GSetError} \Rightarrow \text{GearBox@ErrorIdle}\,) \quad (9)$$

$$Inv\,(\,\text{GearControl@GNeuError} \Rightarrow \text{GearBox@ErrorNeu}\,) \quad (10)$$

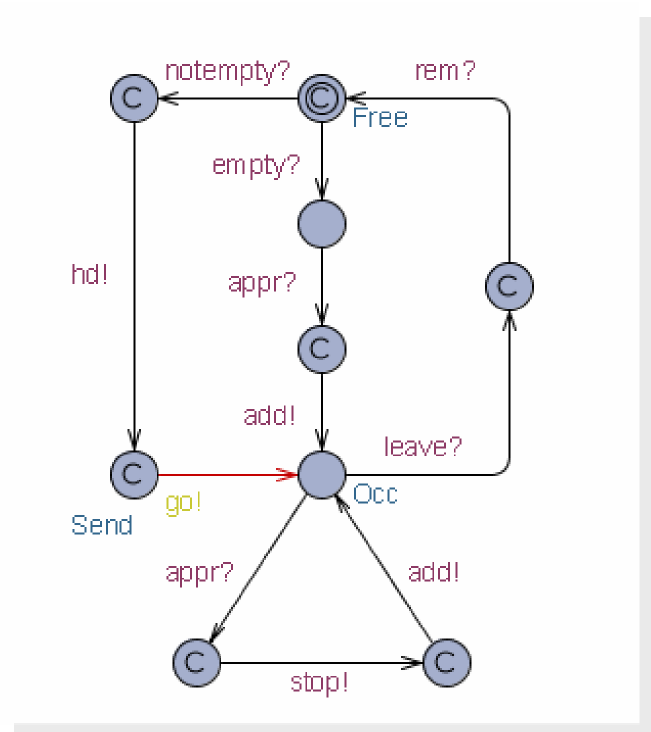
$$Inv\,(\,\text{Engine@ErrorSpeed} \Rightarrow \mathtt{ErrStat} \neq 0\,) \quad (11)$$

$$Inv\,(\,\text{Engine@Torque} \Rightarrow \text{Clutch@Closed}\,) \quad (12)$$

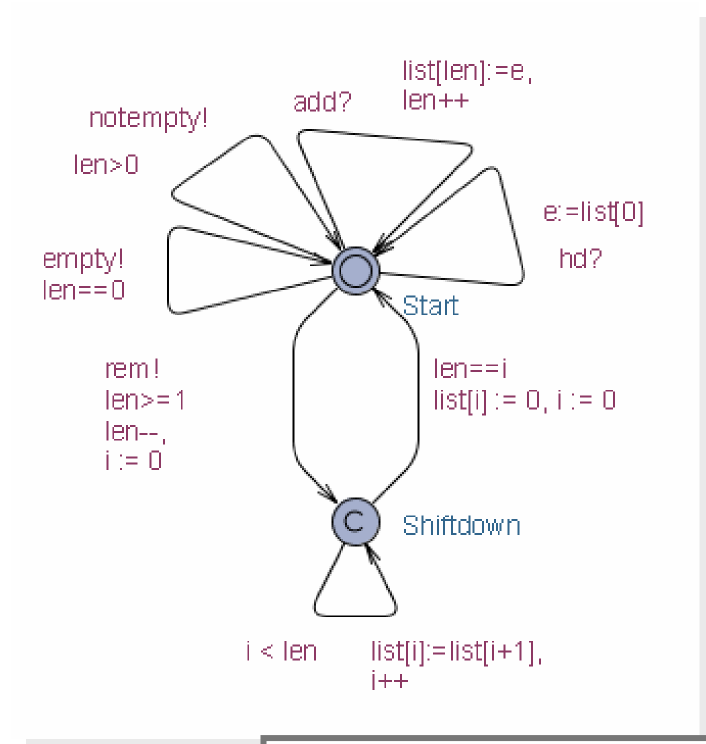
$$\bigwedge_{i \in \{R,N,1,\ldots,5\}} Poss\,(\,\text{Gear@Gear}_i\,) \quad (13)$$

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

UPPAAL 3.4



Gate Template

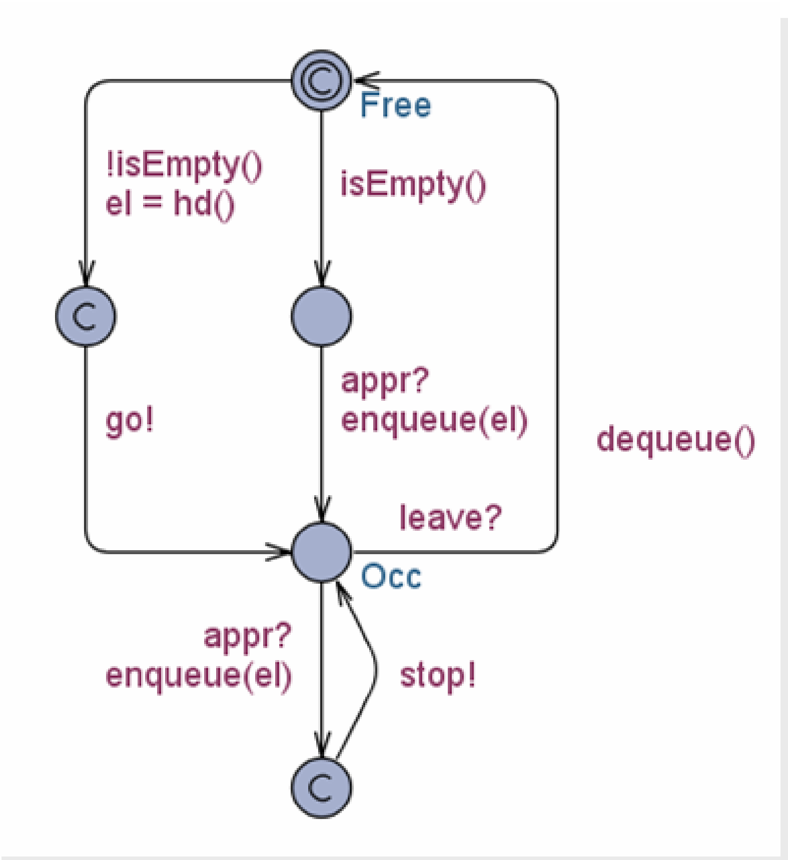



```
int[0,N] list[N], len, i;
```

IntQueue

BRICS
Basic Research
in Computer Science

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]

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

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.

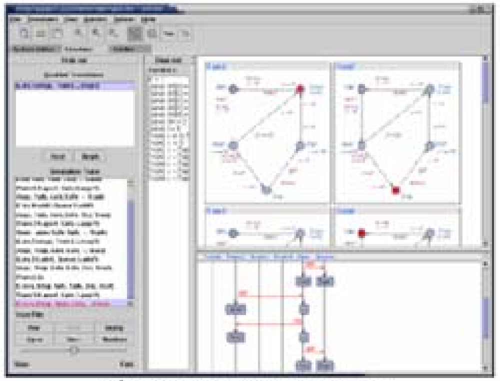


Figure 1: UPPAAL on screen.

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