Real Time Model Checking

using UPPAAL

Kim G Larsen

BRICS Basic Research in Computer Science

CISS CENTER FOR INDLEJRede SOFTWARE SYSTEMER

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…

BRICS Basic Research in Computer Science

Overview

- UPPAAL: a short look
  - Demo's
  - Architecture
- Train Crossing Example
- UPPAAL Syntax
  - Declarations
  - Expressions
  - Locations and Synchronizations
  - Logical Properties
- UPPAAL Verificaiton Engine
- UPPAAL Verification Options
- UPPAAL Modelling Patterns
- Scheduling using UPPAAL.

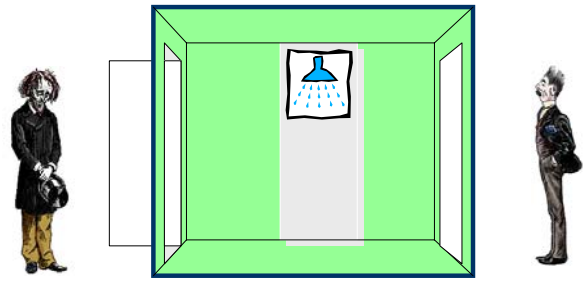
BRICS Basic Research in Computer Science

Druzba

BRICS Basic Research in Computer Science

CISS CENTER FOR INDLEJRede SOFTWARE SYSTEMER

The Druzba MUTEX Problem

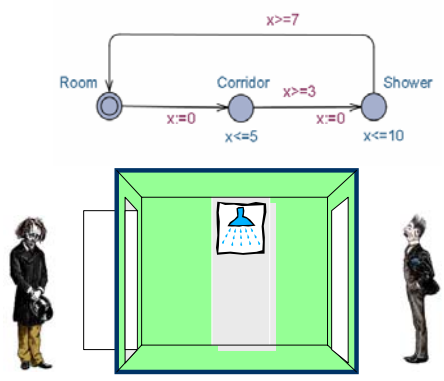


Kim

Gerd

BRICS Basic Research in Computer Science

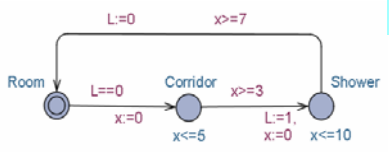
The Druzba MUTEX Problem

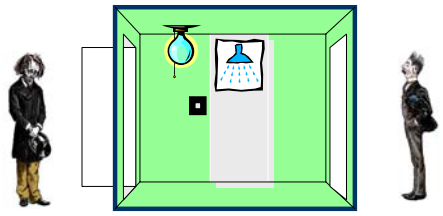


BRICS Basic Research in Computer Science

The Druzba MUTEX Problem

Using the light as semaphor

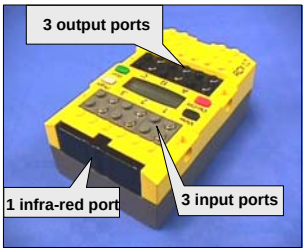




BRICK SORTING

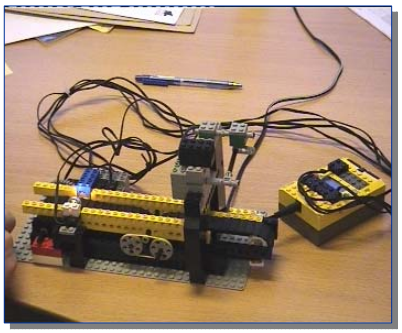
LEGO Mindstorms/RCX

- 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



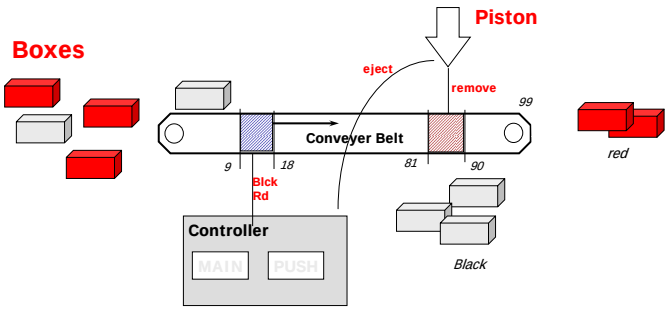
Controller Program
LEGO MINDSTORM

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

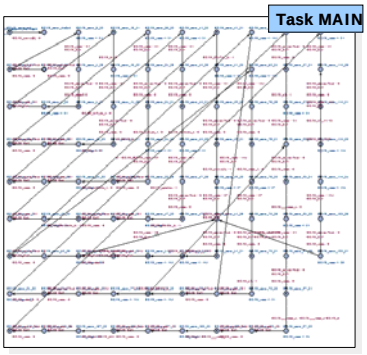
```
int active;
int DELAY;
int LIGHT_LEVEL;
```

```
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

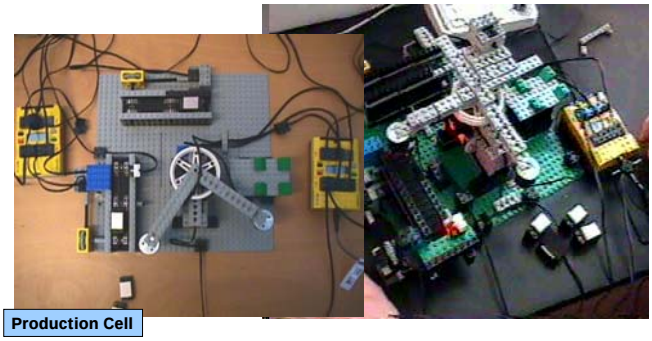
- Model includes Round-Robin Scheduler.
- Compilation of RCX tasks into TA models.
- Presented at ECRTS 2000



BRICS Basic Research in Computer Science

The Production Cell

Course at DTU, Copenhagen



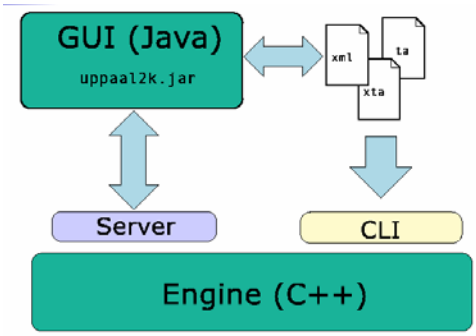
BRICS Basic Research in Computer Science

Overview of the UPPAAL Toolkit

BRICS Basic Research in Computer Science

CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

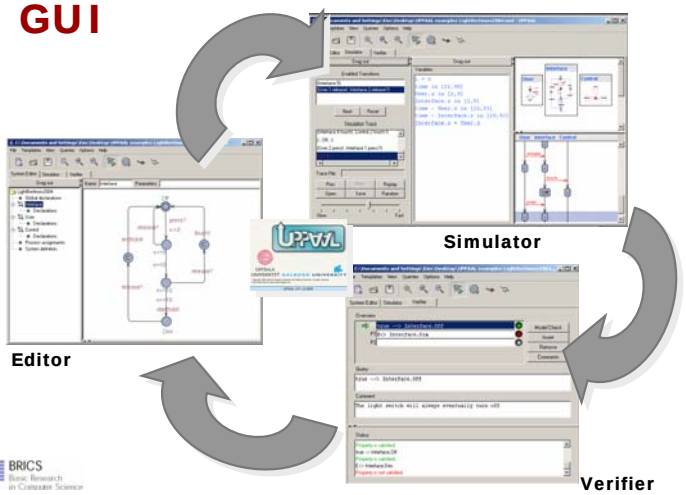
UPPAAL's architecture



Linux, Windows, Solaris, MacOS

BRICS Basic Research in Computer Science

GUI



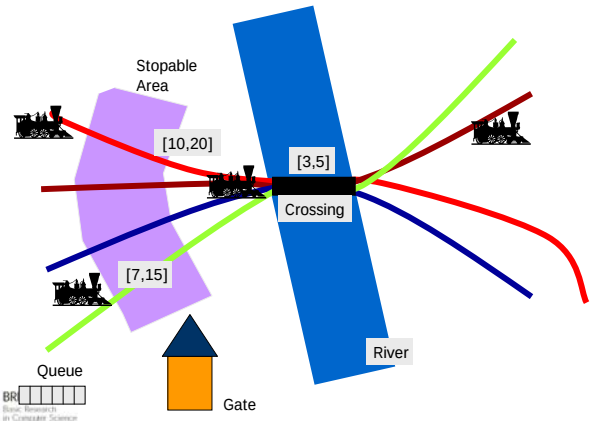
BRICS Basic Research in Computer Science

Train Crossing

BRICS Basic Research in Computer Science

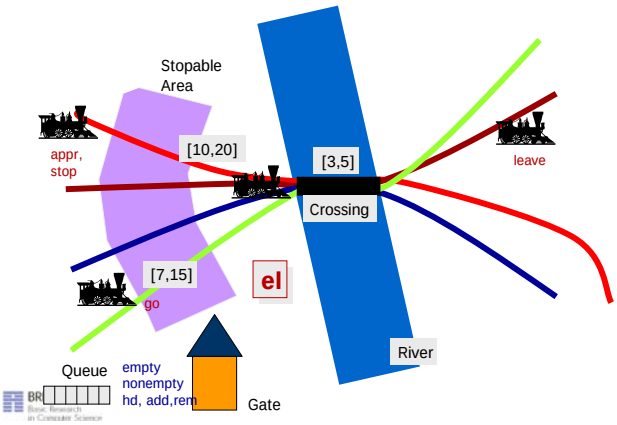
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Train Crossing



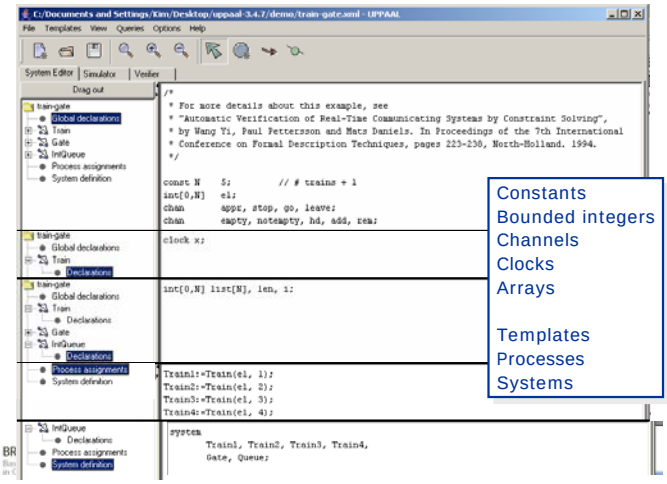
Train Crossing

Communication via channels and shared variable.



Timed Automata in UPPAAL

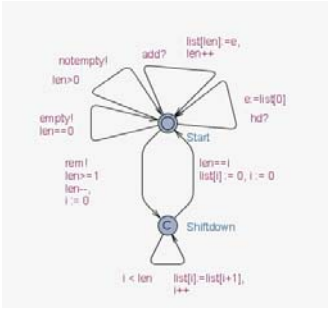
Declarations



Constants
Bounded integers
Channels
Clocks
Arrays

Templates
Processes
Systems

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

Synchronization

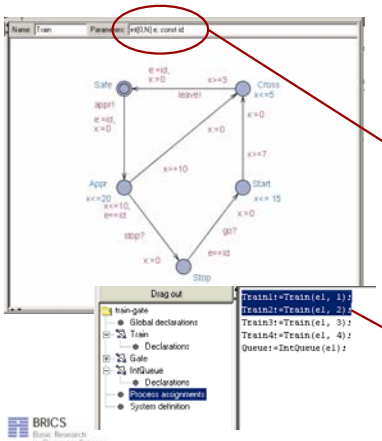
Binary Synchronization

- 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.

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);`

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

Logical Specifications

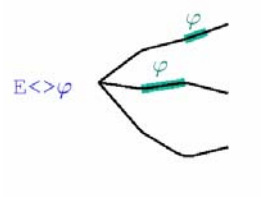
- Validation Properties
  - Possibly: E<> *P*
- Safety Properties
  - Invariant: A[] *P*
  - Pos. Inv.: E[] *P*
- Liveness Properties
  - Eventually: A<> *P*
  - Leadsto: *P* → *Q*
- Bounded Liveness
  - Leads to within: $P \rightarrow_{\leq t} Q$

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).

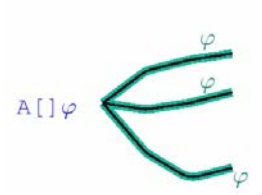
Logical Specifications

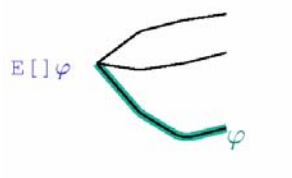
- Validation Properties
  - Possibly: E<> *P*
- Safety Properties
  - Invariant: A[] *P*
  - Pos. Inv.: E[] *P*
- Liveness Properties
  - Eventually: A<> *P*
  - Leadsto: P → *Q*
- Bounded Liveness
  - Leads to within: $P \rightarrow_{\leq t} Q$



Logical Specifications

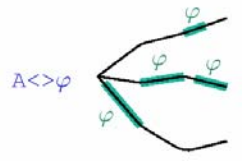
- Validation Properties
  - Possibly: E<> *P*
- Safety Properties
  - Invariant: A[] *P*
  - Pos. Inv.: E[] *P*
- Liveness Properties
  - Eventually: A<> *P*
  - Leadsto: P → *Q*
- Bounded Liveness
  - Leads to within: $P \rightarrow_{\leq t} Q$

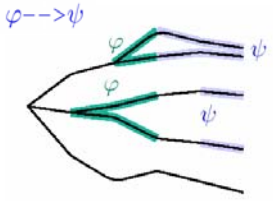




Logical Specifications

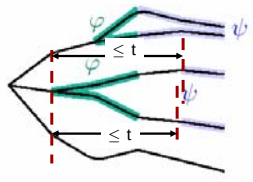
- Validation Properties
  - Possibly: E<> *P*
- Safety Properties
  - Invariant: A[] *P*
  - Pos. Inv.: E[] *P*
- Liveness Properties
  - Eventually: A<> *P*
  - Leadsto: P → *Q*
- Bounded Liveness
  - Leads to within: $P \rightarrow_{\leq t} Q$





Logical Specifications

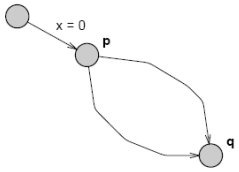
- Validation Properties
  - Possibly: E<> *P*
- Safety Properties
  - Invariant: A[] *P*
  - Pos. Inv.: E[] *P*
- Liveness Properties
  - Eventually: A<> *P*
  - Leadsto: P → *Q*
- Bounded Liveness
  - Leads to within: $P \rightarrow_{\leq t} Q$



Bounded Liveness

We can reduce p $\dashrightarrow_{\leq t}$ q to an unbounded liveness property:

- Add a clock x and reset it whenever p becomes true.
- Check `p --> (q and x <= t)`.

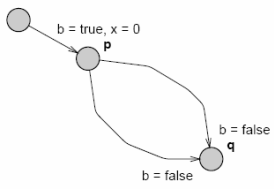


Care must be taken that x is not reset several times before q becomes true.

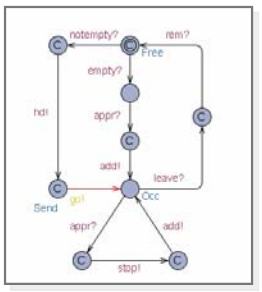
Bounded Liveness

We can reduce p $\dashrightarrow_{\leq t}$ q to a reachability property:

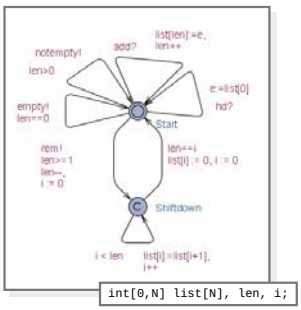
- Add a clock x and reset it whenever p becomes true.
- Add a boolean b, set it to true when p starts to hold and to false when p ceases to hold.
- Check `A[] (b implies x <= t)`.



UPPAAL

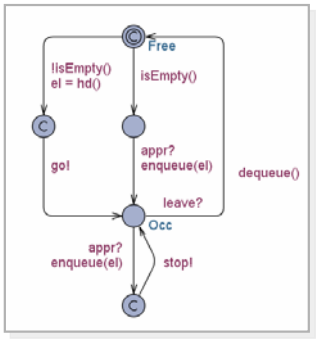


Gate Template



IntQueue

UPPAAL with C-Code *(U-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];
}
```

To come in next release

Gate Declaration

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, Memory Management for Radar (2001)

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)

UPPAAL Verification Engine

Overview

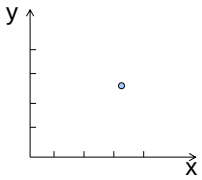
- Zones and DBMs
- Minimal Constraint Form
- Clock Difference Diagrams

- Distributed UPPAAL [CAV2000, STTT2004]
- Unification & Sharing [FTRTFT2002, SPIN2003]
- Acceleration [FORMATS2002]
- Static Guard Analysis [TACAS2003,TACAS2004]
- Storage-Strategies [CAV2003]

Zones

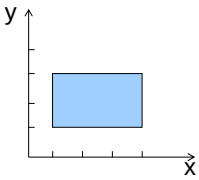
From infinite to finite

State
(n, x=3.2, y=2.5)

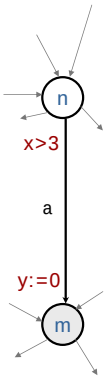


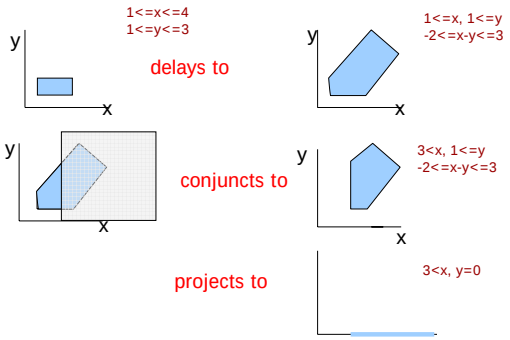
Symbolic state (set)
(n, 1≤x≤4, 1≤y≤ 3)

Zone:
conjunction of
x-y<=n, x<=>n



Symbolic Transitions

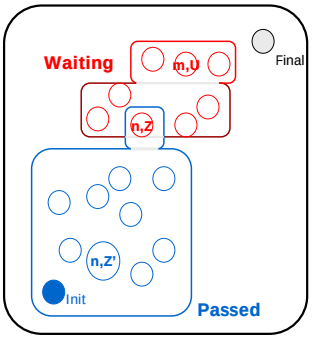




Thus (n,1<=x<=4,1<=y<=3) =a => (m,3<x, y=0)

Forward Rechability

Init -> Final ?

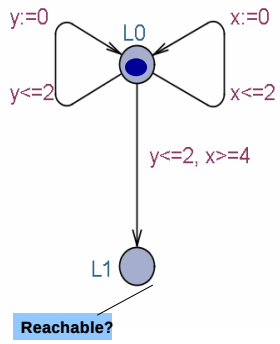


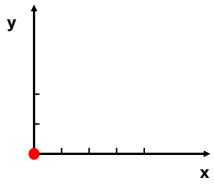
INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

REPEAT
- pick (n,Z) in **Waiting**
- **if** for some Z' ⊇ Z
(n,Z') in **Passed** **then** **STOP**
- **else** /explore/ add
{ (m,U) : (n,Z) => (m,U) }
to **Waiting**;
Add (n,Z) to **Passed**

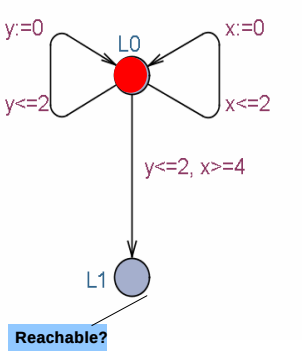
UNTIL **Waiting** = Ø
or
Final is in **Waiting**

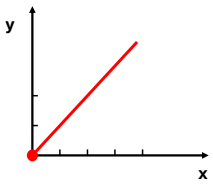
Symbolic Exploration





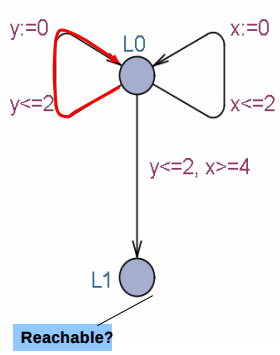
Symbolic Exploration

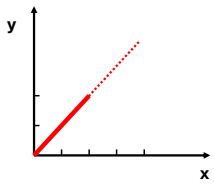




Delay

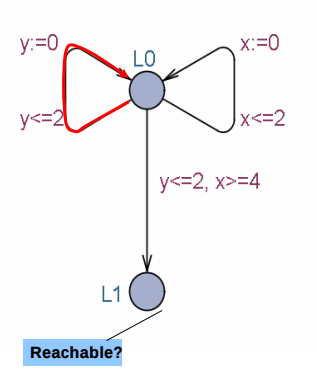
Symbolic Exploration

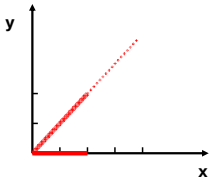




Left

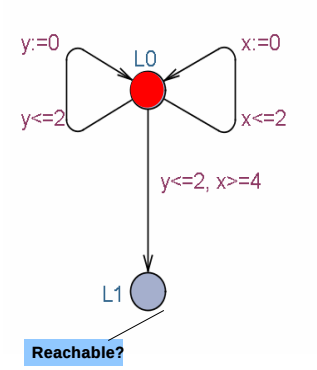
Symbolic Exploration

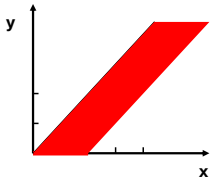




Left

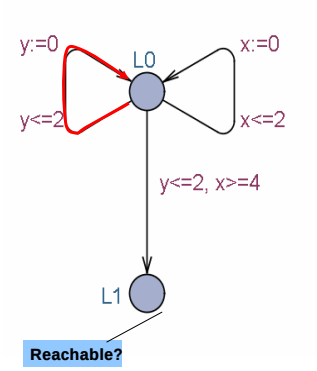
Symbolic Exploration

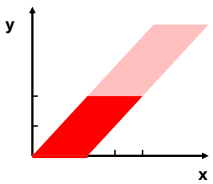




Delay

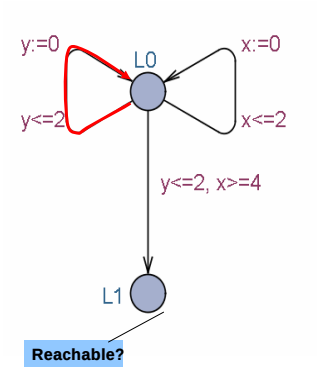
Symbolic Exploration

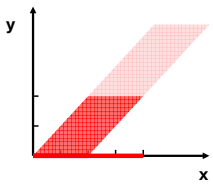




Left

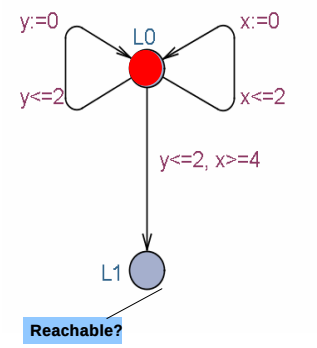
Symbolic Exploration

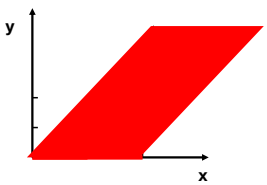




Left

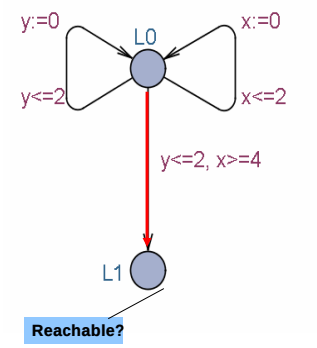
Symbolic Exploration

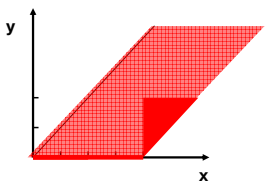




Delay

Symbolic Exploration





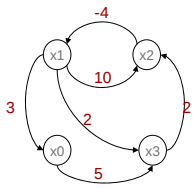
Down

Canonical Datastructures for Zones

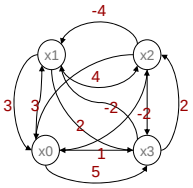
Minimal Constraint Form

RTSS 1997

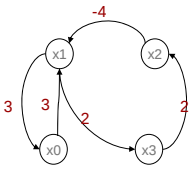
x1-x2<=4
x2-x1<=10
x3-x1<=2
x2-x3<=2
x0-x1<=3
x3-x0<=5



Shortest Path Closure O(n^3)

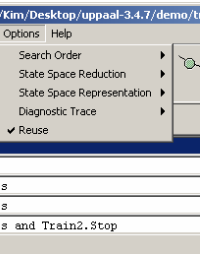


Shortest Path Reduction O(n^3)



Space worst O(n^2)
practice O(n)

Verification Options



Search Order
- Depth First
- Breadth First

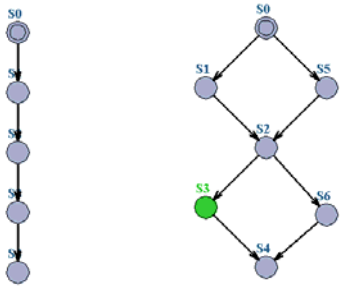
State Space Reduction
- None
- Conservative
- Aggressive

State Space Representation
- DBM
- Compact Form
- Under Approximation
- Over Approximation

Diagnostic Trace
- Some
- Shortest
- Fastest

Verification Options

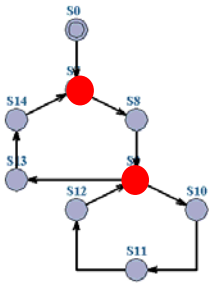
State Space Reduction



However,
Passed list useful for efficiency

No Cycles: **Passed** list not needed for *termination*

State Space Reduction



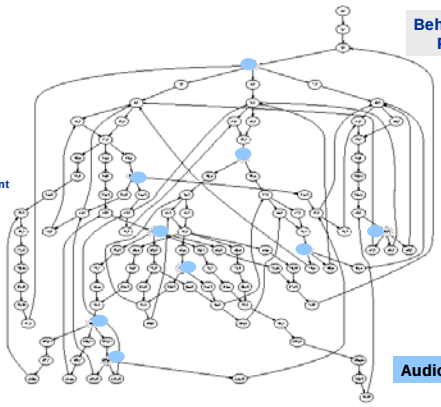
Cycles:
Only symbolic states involving loop-entry points need to be saved on **Passed** list

To Store or Not To Store

Behrmann, Larsen, Pelanek 2003

117 states$_{total}$
→
81 states$_{entrypoint}$
→
9 states

Time OH less than 10 %



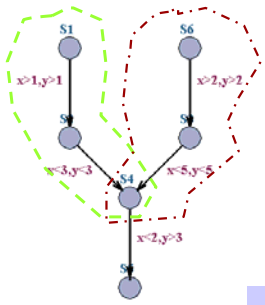
Audio Protocol

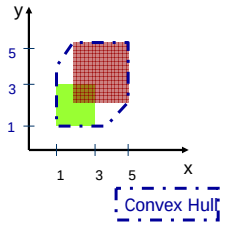
To Store or Not to Store

Behrmann, Larsen, Pelanek 2003

	entry points	covering set	successors	random $p = 0.1$	distance $k = 10$	combination $k = 3$
Fischer	27.1%	42.1%	47.9%	53.7%	67.6%	56.9%
3,077	1.00	1.66	1.00	4.51	2.76	6.57
BRP	70.5%	16.5%	19.8%	18.3%	15.8%	7.6%
6,060	1.01	1.20	1.03	1.78	1.34	1.68
Token Ring	33.0%	10.3%	20.7%	17.2%	17.5%	16.8%
15,103	1.16	1.46	1.03	1.63	1.43	7.40
Train-gate	71.1%	27.4%	24.2%	31.8%	24.2%	19.8%
16,666	1.22	1.55	1.68	2.90	2.11	5.08
Dacapo	29.4%	24.3%	24.9%	12.2%	12.7%	7.0%
30,502	1.07	1.08	1.07	1.21	1.16	1.26
CSMA	94.0%	75.9%	81.2%	105.9%	114.9%	120.3%
47,857	1.06	2.62	1.40	7.66	2.83	6.82
BOCDP	25.2%	22.5%	6.5%	10.2%	9.3%	4.5%
203,557	1.00	1.01	1.08	1.02	1.01	1.09
BOPDP	14.7%	13.2%	42.1%	15.2%	11%	4.3%
1,013,072	2.40	1.33	1.02	1.52	1.14	1.74
Buscoupler	53.2%	13.6%	40.5%	31.7%	24.6%	14.3%
3,595,108	1.29	2.48	1.18	3.17	2.13	8.73

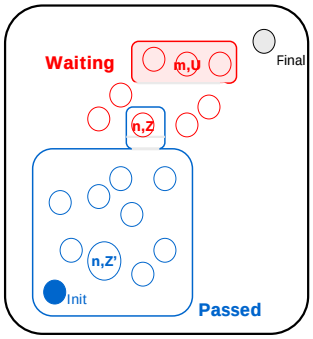
Over-approximation
Convex Hull



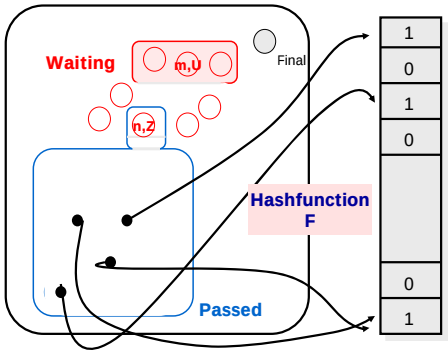


TACAS04: An **EXACT** method performing as well as Convex Hull has been developed based on abstractions taking max constants into account distinguishing between clocks, locations and $\leq$ & $\geq$

Under-approximation
Bitstate Hashing



Under-approximation
Bitstate Hashing



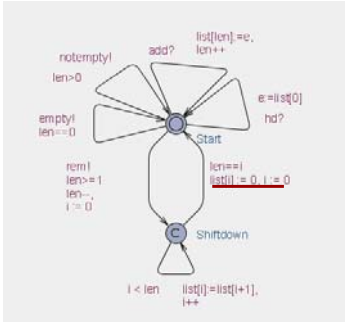
Passed= Bitarray

UPPAAL 8 Mbits

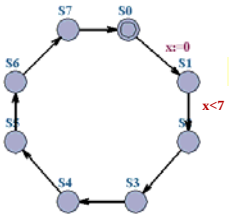
Modelling Patterns

Variable Reduction

- Reduce size of state space by explicitely resetting variables when they are not used!
- Automatically performed for clock variables (active clock reduction)



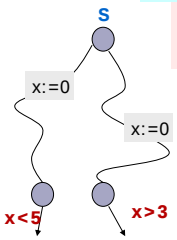
Variable Reduction



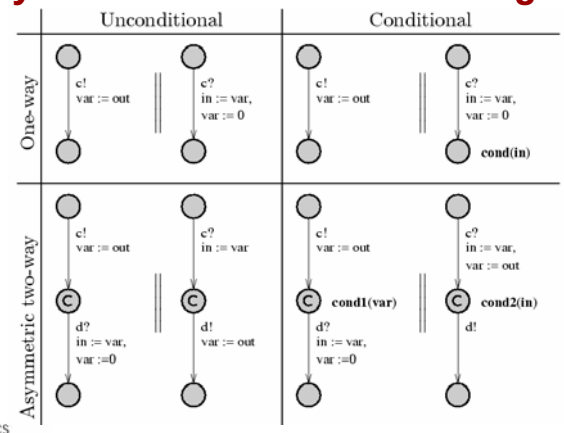
x is only ***active*** in location **S1**

Definition

x is ***inactive*** at **S** if on all path from **S**, x is always reset before being tested.

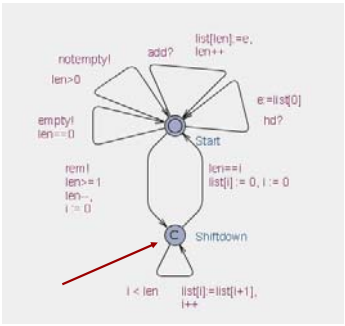


Synchronous Value Passing



Atomicity

- To allow encoding of control structure (for- or while-loops, conditionals, etc.) without erroneous interleaving
- To allow encoding of multicasting.
- Heavy use of committed locations.



AMETIST
advanced methods for timed systems

Optimal Real Time Planning & Scheduling

with Gerd Behrmann, Ed Brinksma, Ansgar Fehnker, Thomas Hune, Paul Pettersson, Judi Romijn, Frits Vaandrager, Patricia Bouyer, Franck Cassez, Emmanuel Fleury, Arne Skou, Jacob Rasmussen, K. Subramani

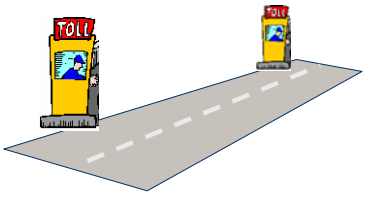
AALBORG UNIVERSITY DENMARK

BRICS Basic Research in Computer Science

CISS CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

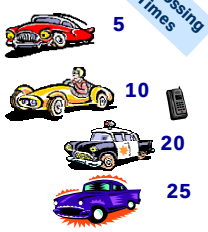
Real Time Scheduling

- Only 1 "BroBizz"
- Cheat is possible (drive close to car with "Bizz")



UNSAFE

Crossing Times



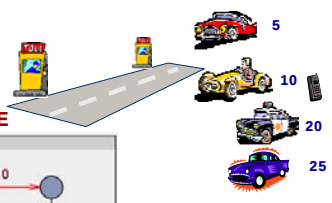
SAFE

CAN THEY MAKE IT TO SAFE WITHIN 70 MINUTES ???

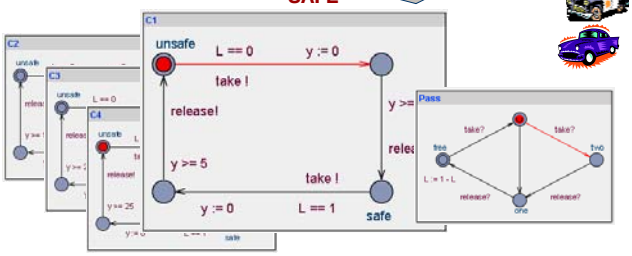
Real Time Scheduling

Solve Scheduling Problem using UPPAAL

UNSAFE



SAFE



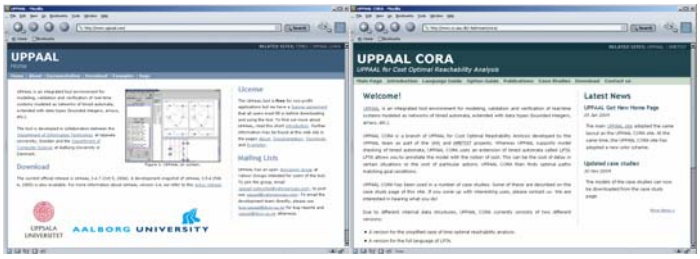
Rush Hour

OBJECTIVE: Get your CAR out

EEF Summerschool on Concurrency, Kapellerput

BRICS Basic Research in Computer Science

Further Information



www.uppaal.com **www.cs.auc.dk/~behrmann/cora**

BRICS Basic Research in Computer Science