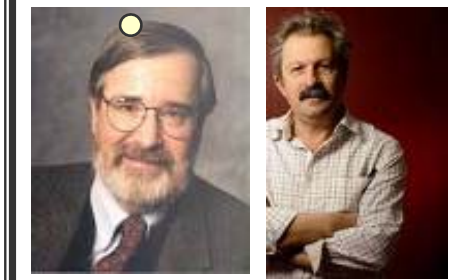
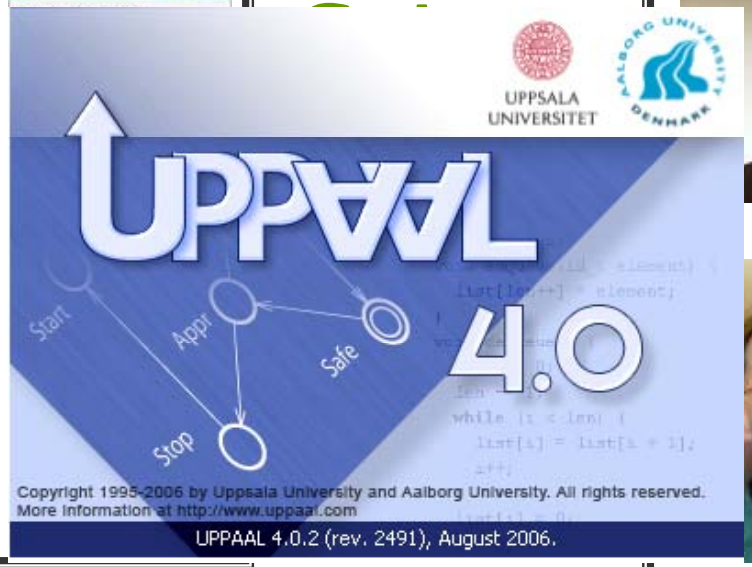
Formal Aspects of Security

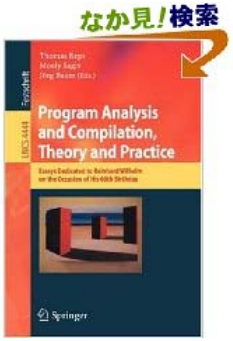
Kim G. Larsen
CISS – Aalborg University
DENMARK

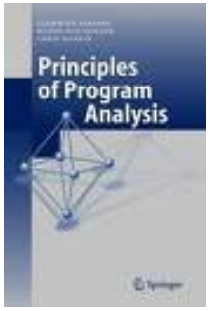
Model checking *is fixpoint iteration without dynamic abstraction and using set union to collect states.*

Abstract Interpretation *is fixpoint iteration with dynamic abstraction using lattice join to combine abstract states.*

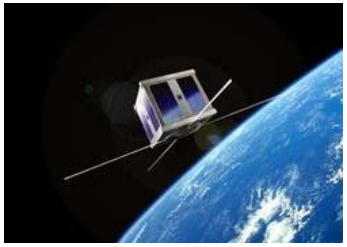








ES are Pervasive



Characteristica:

- Dedicated function
- Complex environment
- SW/HW/Mechanics
- Autonomous
- Ressource constrained
  - : Energy
  - : Bandwidth
  - : Memory
  - : …
- Timing constraints

ES are often Safety Critical

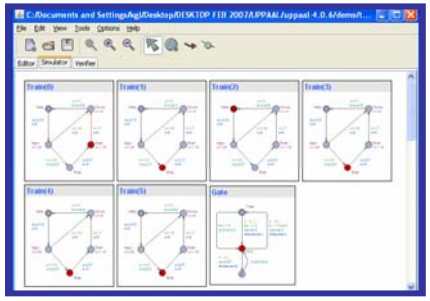
300 horse power
100 processors

How to achieve ES that are:

- correct
- predicable
- dependable
- fault tolerant
- ressource minial
- cheap

Model-Based Development

QUANTITATIVE Model Checking

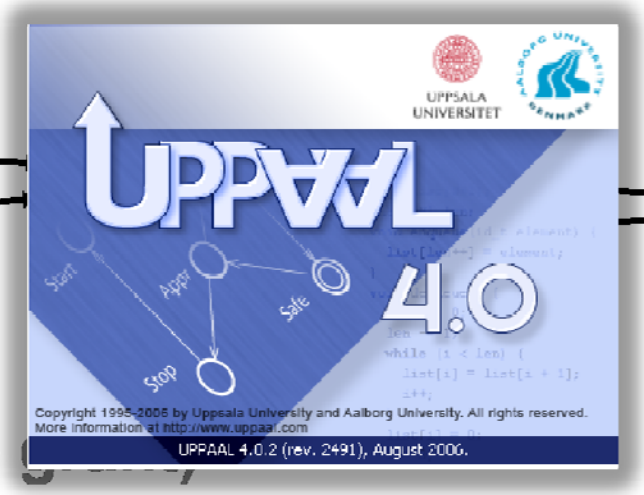


System Description

Time

Cost

Probability



No!
Debugging Information

Yes
Prototypes
Executable Code
Test sequences

Requirement

$A\Box(\ req \Rightarrow A\Diamond\ \ldots)$

$A\Box(\ req \Rightarrow A\Diamond_{t<30s}\ grant)$

$A\Box(\ req \Rightarrow A\Diamond_{t<30s,c<5\$}\ grant)$

$A\Box(\ req \Rightarrow A\Diamond_{t<30s\ ,\ p>0.90}\ grant)$

A simple program

```
int x=100;

Process INC
        do
        :: x<200 --> x:=x+1
        od

Process DEC
        do
        :: x>0 --> x:=x-1
        od

Process RESET
        do
        :: x=200 --> x:=0
        od

( INC || DEC || RESET )
```

Which values may x take ?

Questions/Properties:

E<>(x>100)

E<>(x>200)

A[](x<=200)

E<>(x<0)

A[](x>=0)

Possibly

Always

Formal Aspects of Security

- Formal Aspects
  - Abstract Interpretation & Code
  - Model Checking & Models

- Security
- Safety
- Reliability
- Performance

Overview

- Model Checking in UPPAAL
  - Modeling
  - Requirements
  - Analysis
- Leader Election Protocols
- Performance Analysis

- Compositional Analysis

- Exercise(s)

Timed Automata

UPPAAL (1995-)

@AALborg

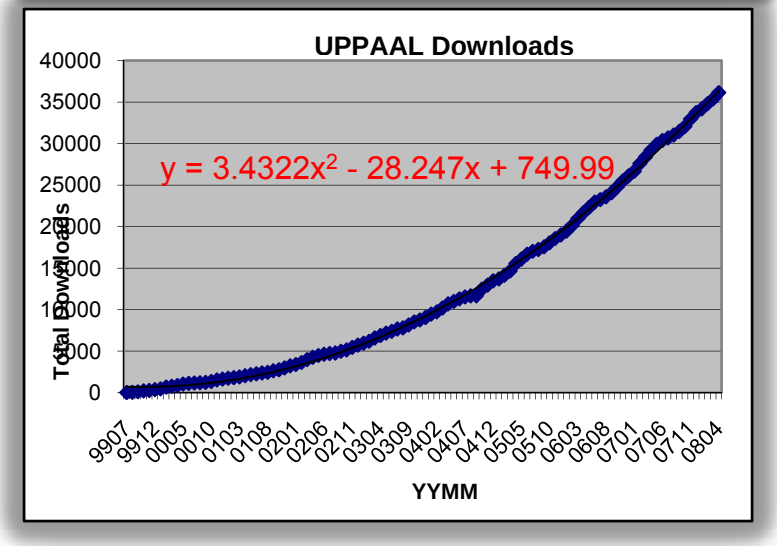
- Kim G Larsen
- Alexandre David
- Gerd Behrman
- Marius Mikucionis
- Jacob I. Rasmussen
- Arne Skou
- Brian Nielsen
- Shuhao Li

@UPPsala

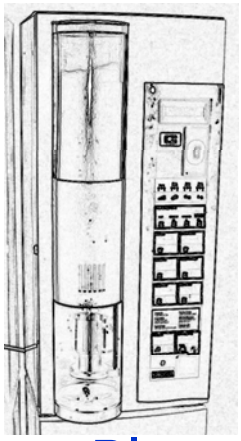
- Wang Yi
- Paul Pettersson
- John Håkansson
- Anders Hessel
- Pavel Krcal
- Leonid Mokrushin
- Shi Xiaochun

@Elsewhere

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

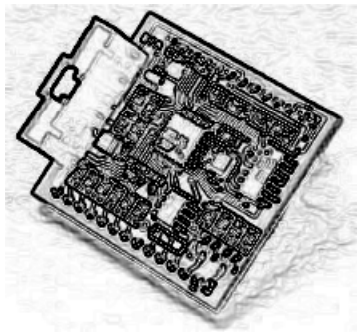


Real Time Systems



sensors

actuators



Plant

Continuous

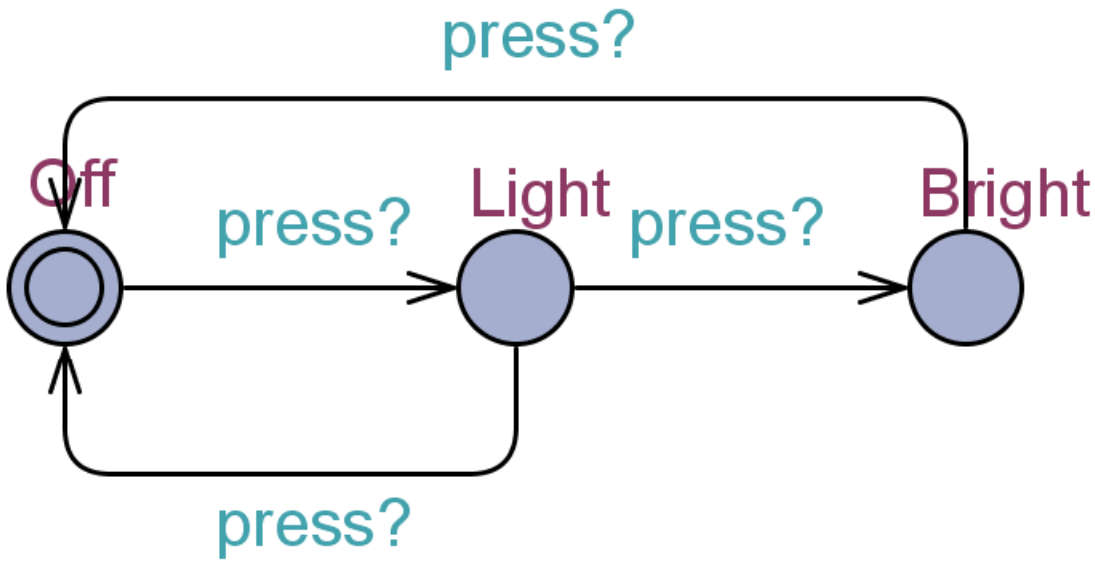
Controller Program

Discrete

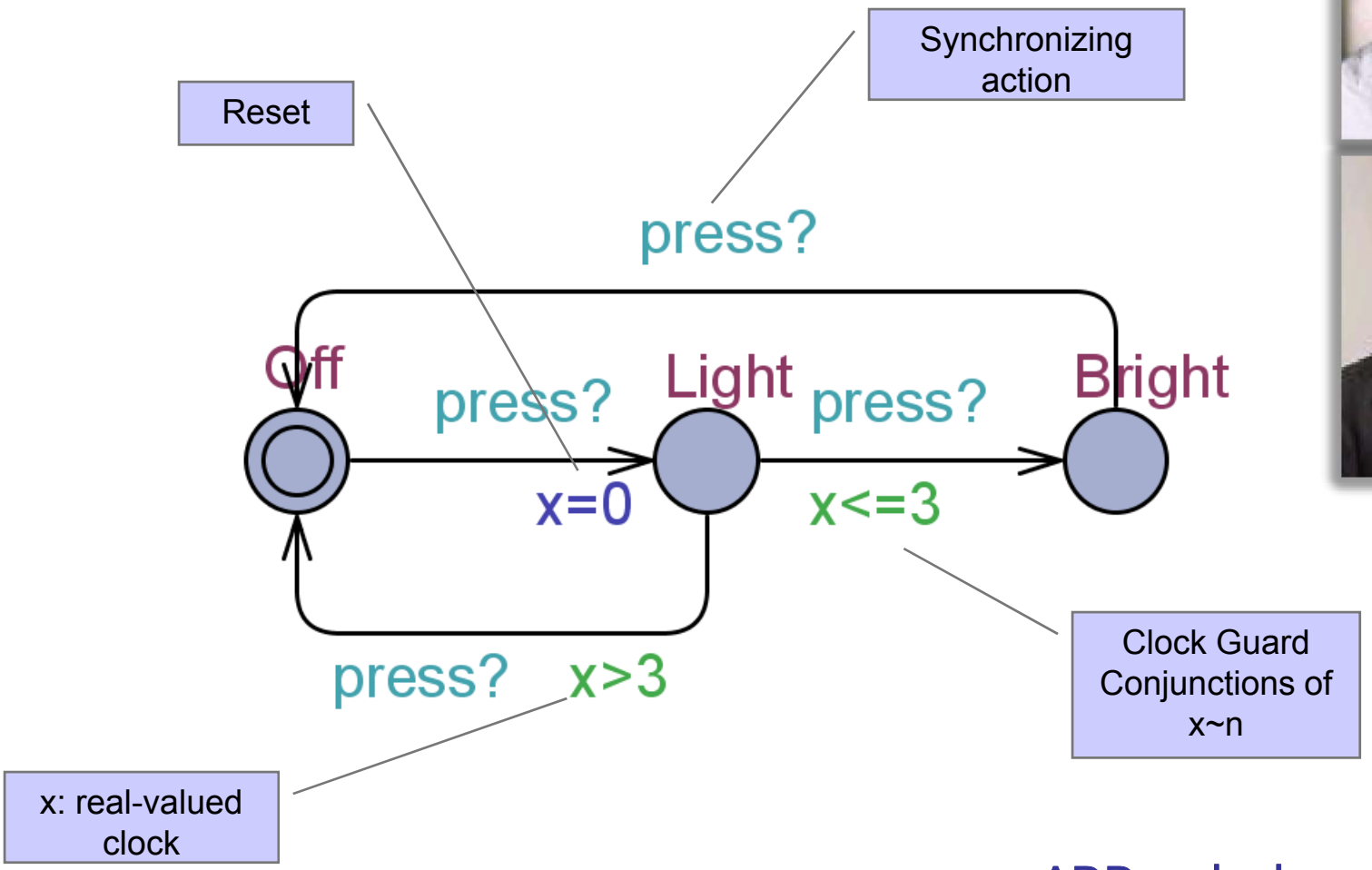
Eg.: Realtime Protocols
Pump Control
Air Bags
Robots
Cruise Control
ABS
CD Players
Production Lines

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

A Dumb Light Controller

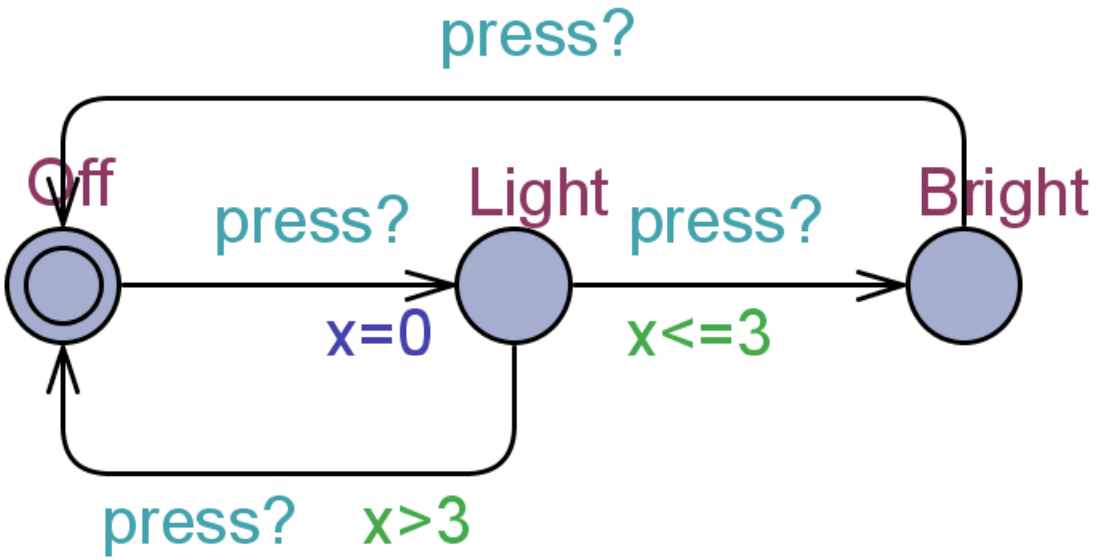


Timed Automata [Alur & Dill'89]



ADD a clock x

A Timed Automata (Semantics)

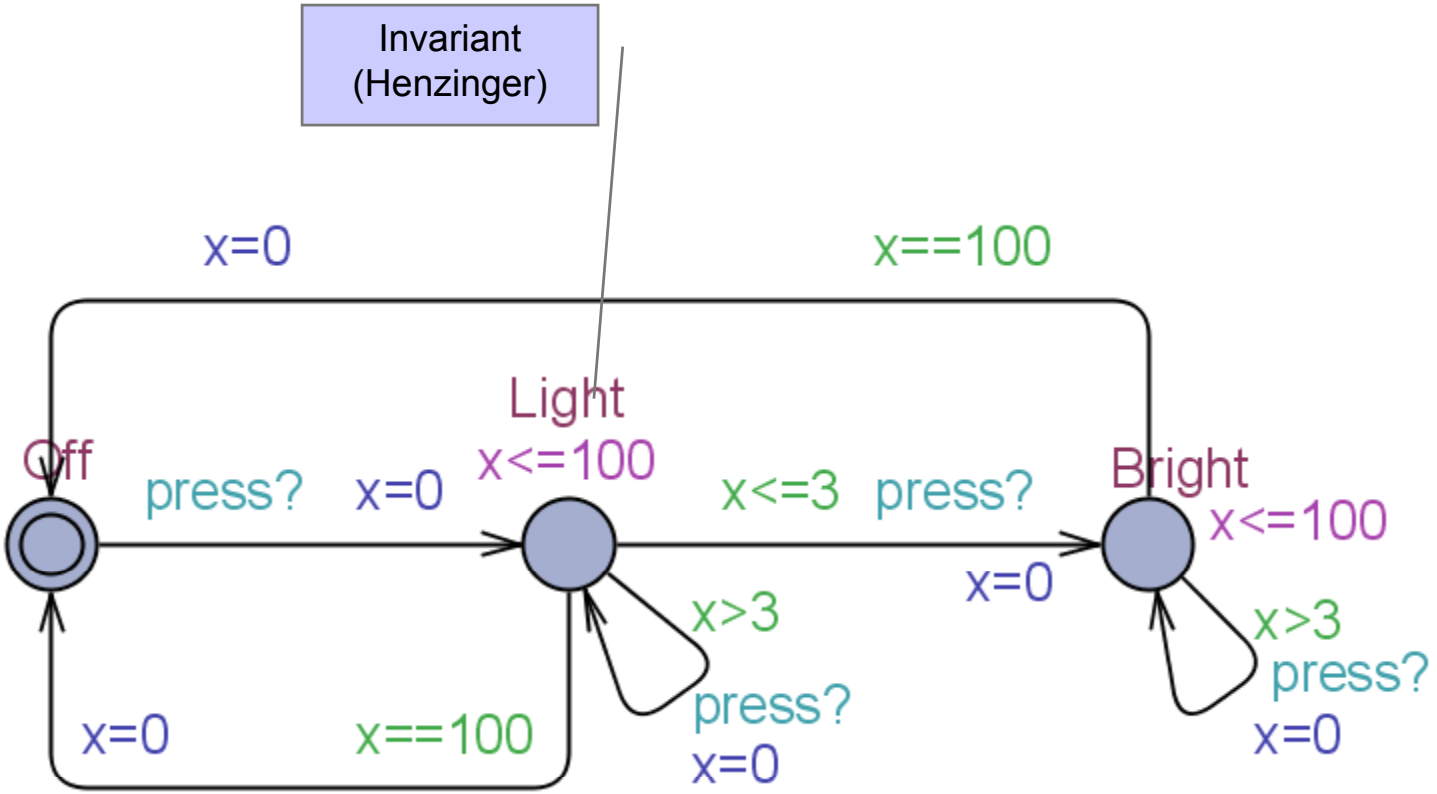


States:
(location , x=v) where v∈**R**

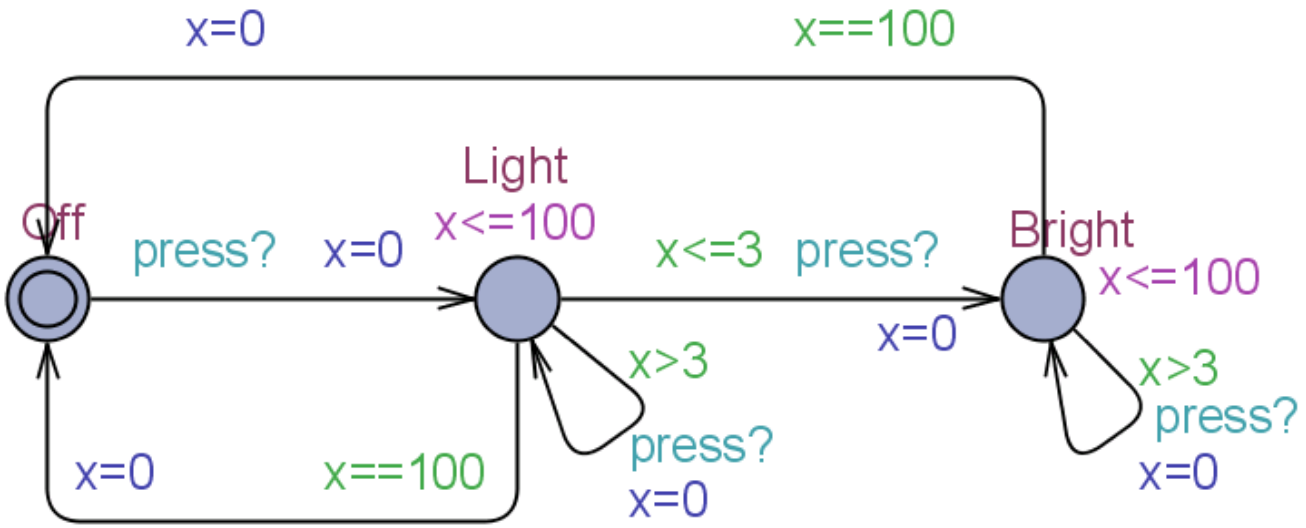
Transitions:

	(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 Controller



Intelligent Light Controller



Transitions:

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

Note:
(Light , x=0) delay 103 →

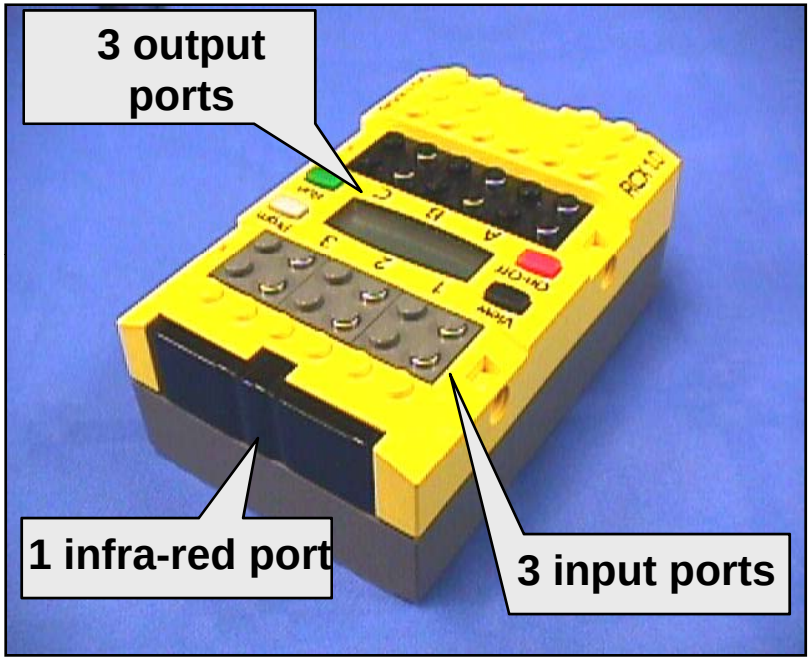
X

Invariants ensures progress

Brick Sorting

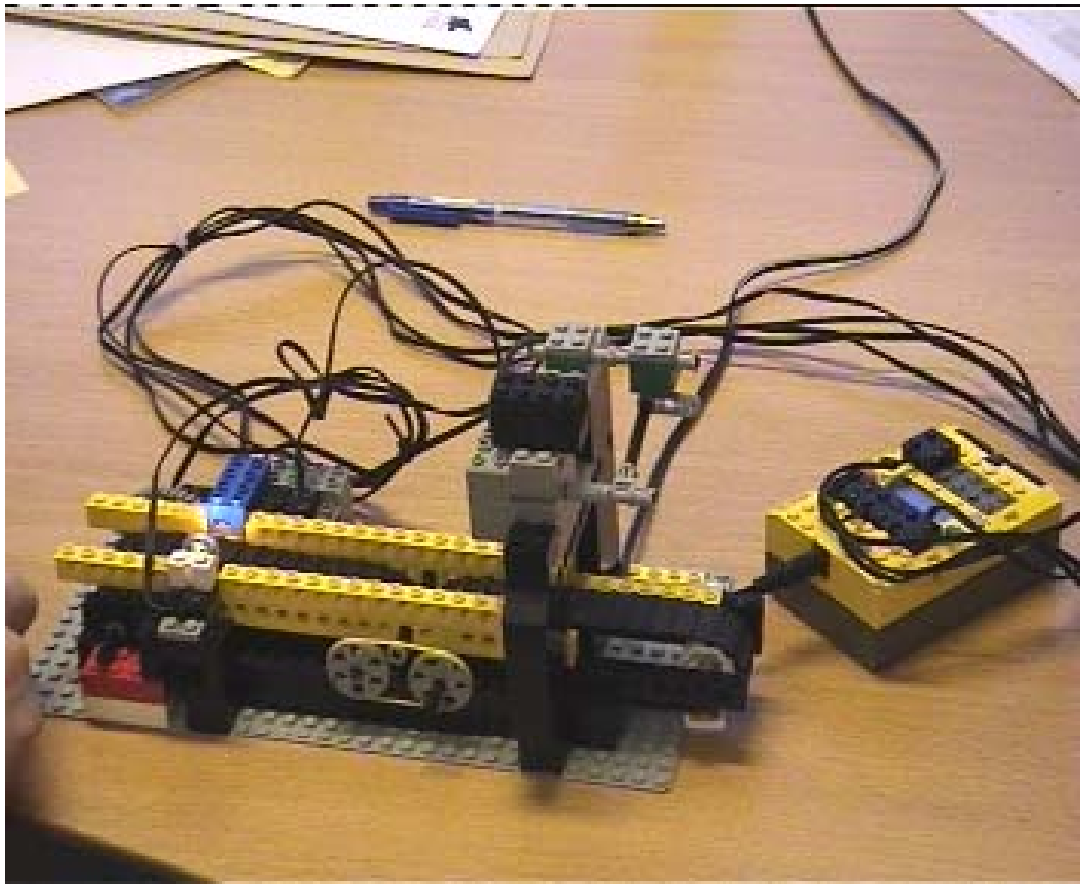
LEGO Mindstorms/RCX

- 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

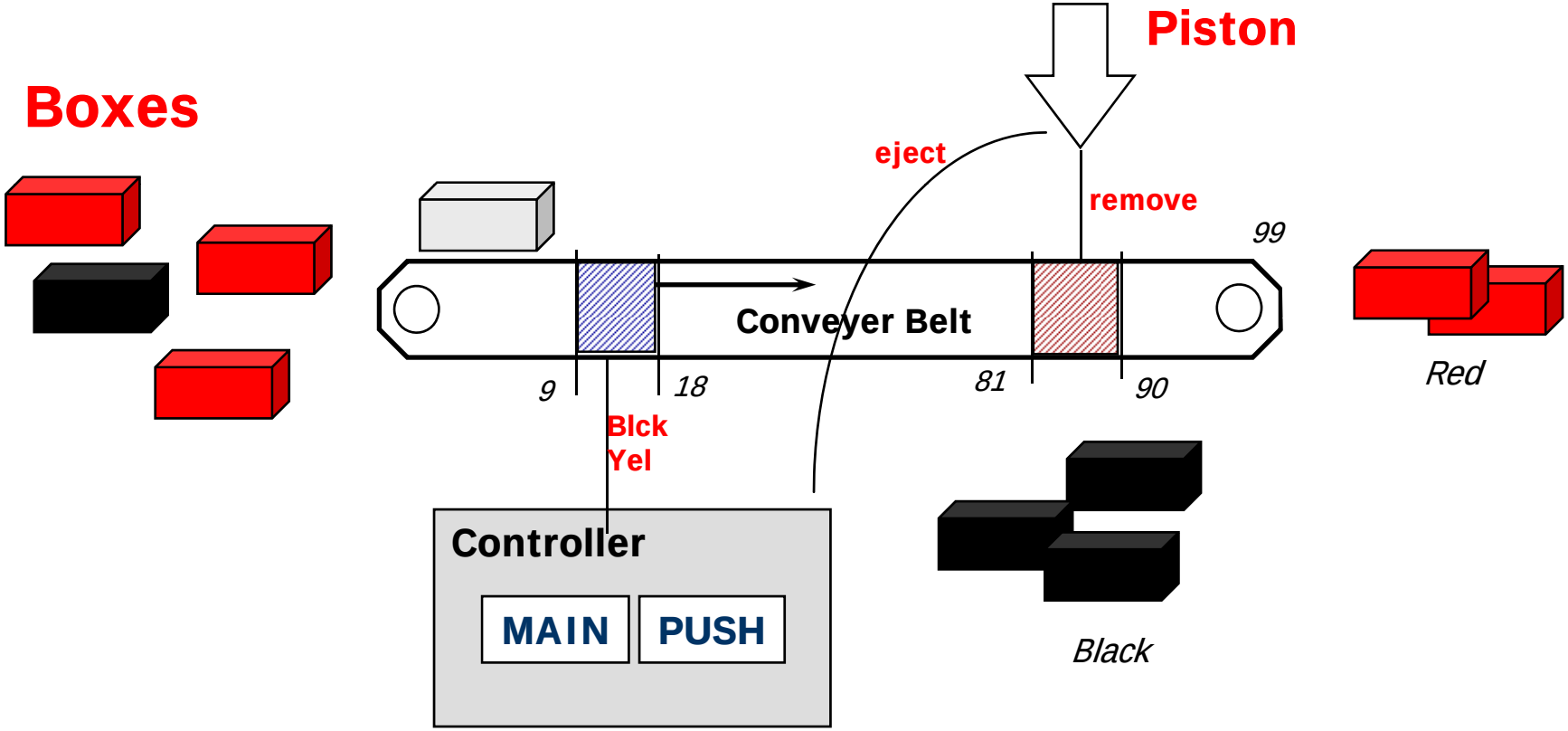


Controller Program
LEGO MINDSTORM

First UPPAAL model

Sorting of Lego Boxes

Ken Tindell



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

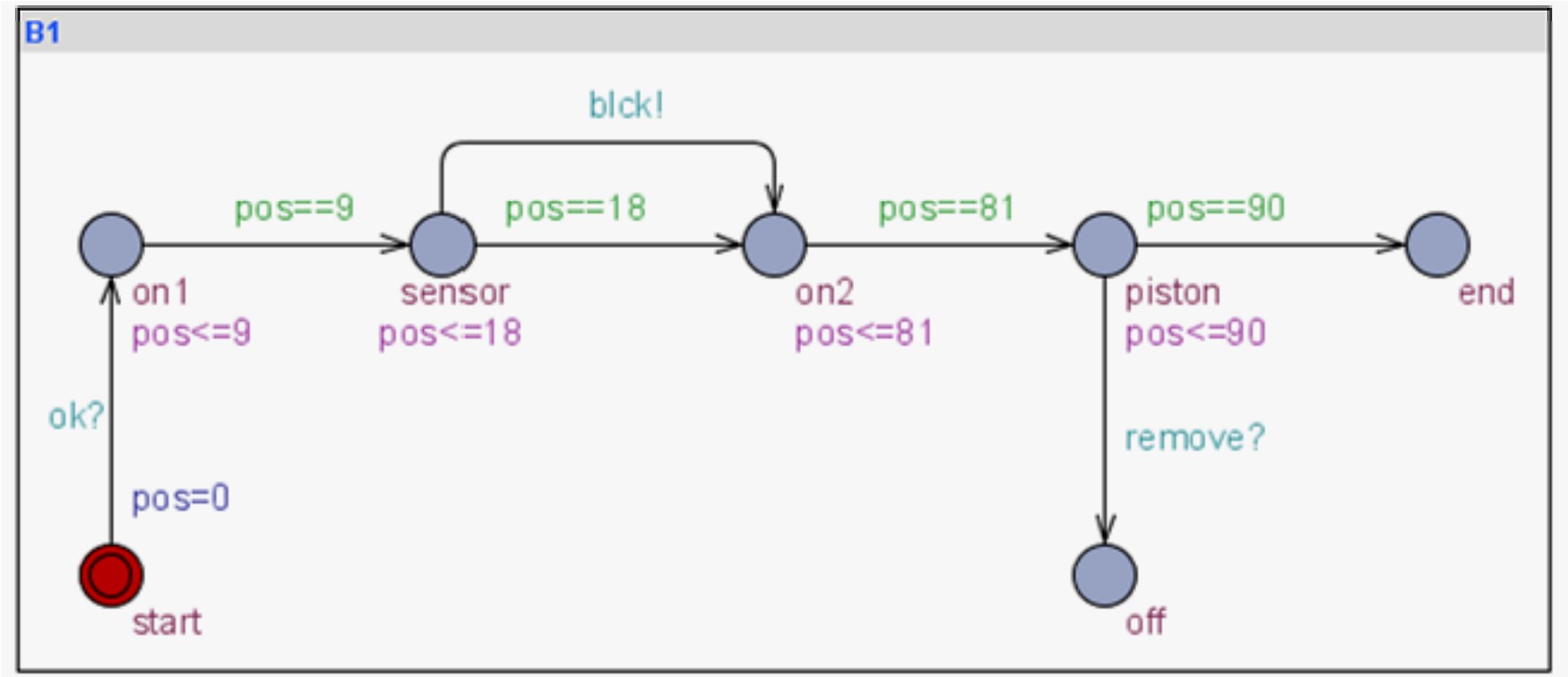
NQC programs

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

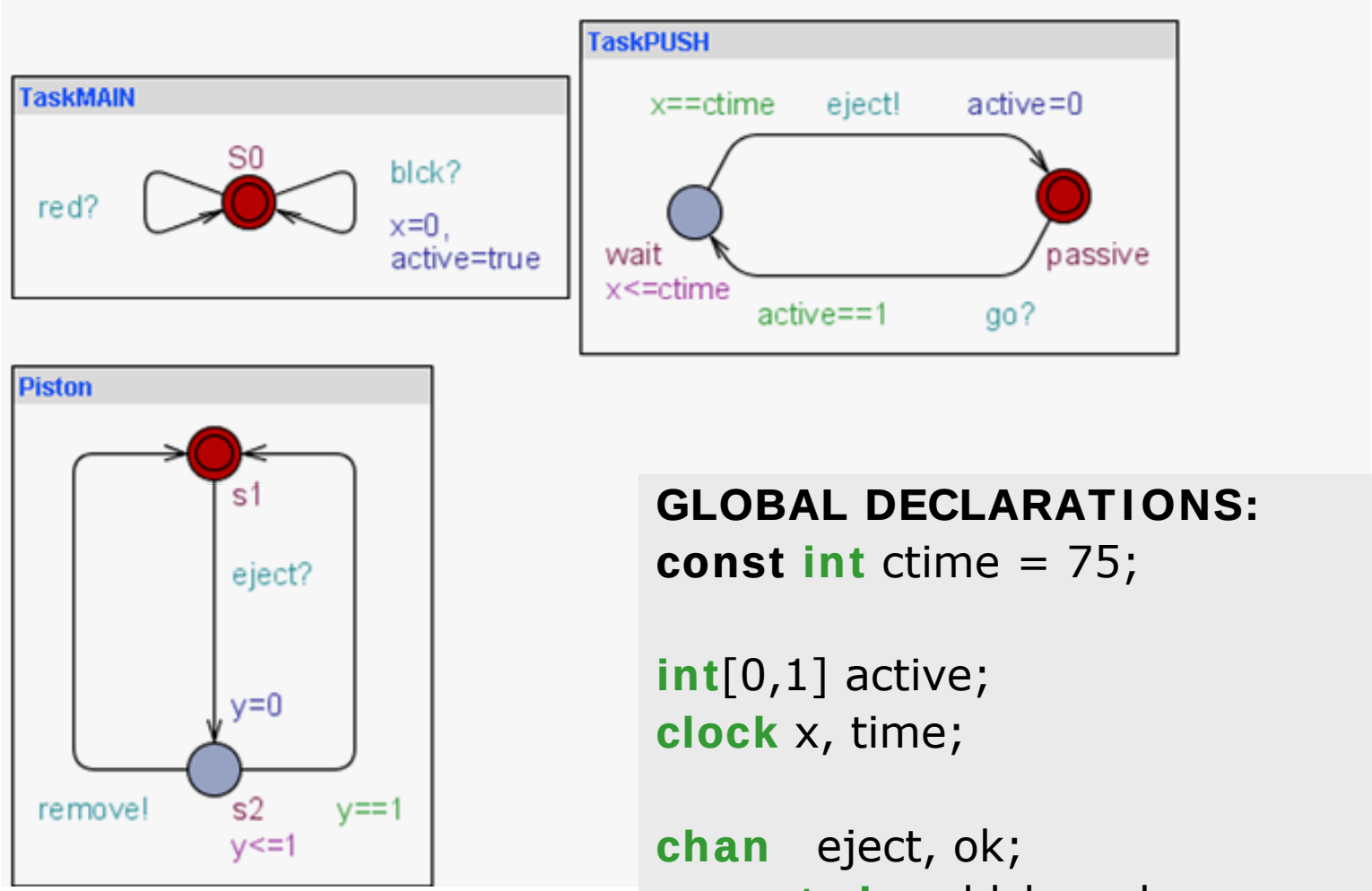
```
task MAIN{
 DELAY=75;
 LIGHT_LEVEL=35;
 active=0;
 Sensor(IN_1, IN_LIGHT);
 Fwd(OUT_A,1);
 Display(1);

 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;
```

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]
- 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]

- Adapting the UPPAAL Model of a Distributed Lift System, 2007
- Analyzing a χ model of a turntable system using Spin, CADP and Uppaal, 2006
- **Designing, Modelling and Verifying a Container Terminal System Using UPPAAL, 2008**
- Model-based system analysis using Chi and Uppaal: An industrial case study, 2008
- Climate Controller for Pig Stables, 2008
- Optimal and Robust Controller for Hydralic Pump, 2009

Case Studies: Protocols

- Philips Audio Protocol [HS'95, CAV'95, RTSS'95, CAV'96]
- Bounded Retransmission Protocol [TACAS'97]
- Bang & Olufsen Audio/Video Protocol [RTSS'97]
- TDMA Protocol [PRFTS'97]
- Lip-Synchronization Protocol [FMICS'97]
- ATM ABR Protocol [CAV'99]
- ABB Fieldbus Protocol [ECRTS'2k]
- IEEE 1394 Firewire Root Contention (2000)
- Distributed Agreement Protocol [Formats05]

Case Studies: Protocols

- Leader Election for Mobile Ad Hoc Networks [Charme05]
- Analysis of a protocol for dynamic configuration of IPv4 link local addresses using Uppaal, 2006
- Formalizing SHIM6, a Proposed Internet Standard in UPPAAL, 2007
- Verifying the distributed real-time network protocol RTnet using Uppaal, 2007
- Analysis of the Zeroconf protocol using UPPAAL, 2009
- Analysis of a Clock Synchronization Protocol for Wireless Sensor Networks, 2009
- Model Checking the FlexRay Physical Layer Protocol, 2010

Using UPPAAL as Back-end

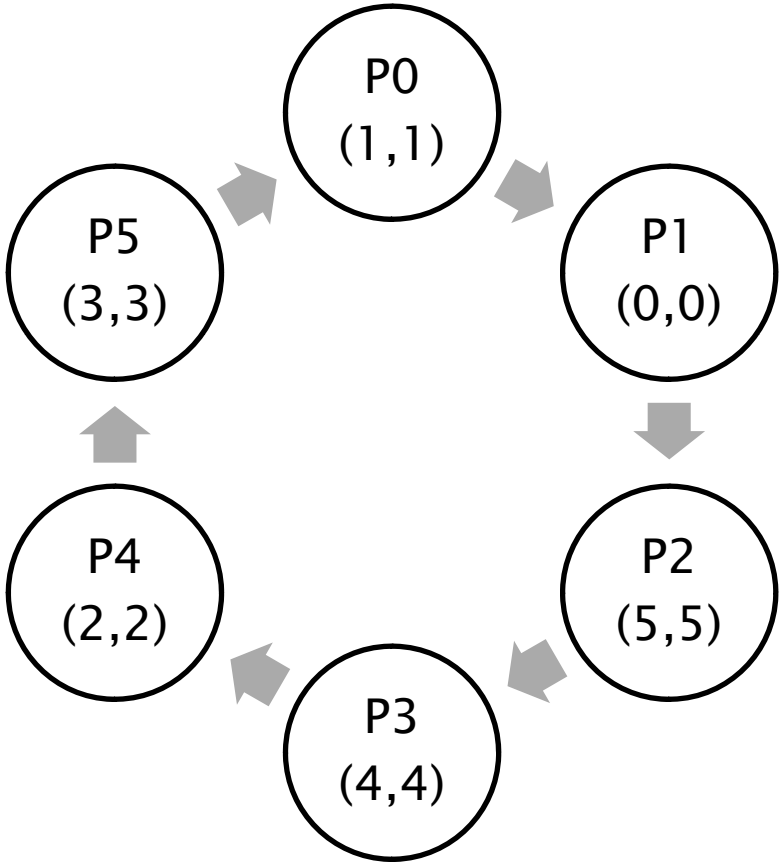
- Vooduu: verification of object-oriented designs using Uppaal, 2004
- Moby/RT: A Tool for Specification and Verification of Real-Time Systems, 2000
- Formalising the ARTS MPSOC Model in UPPAAL, 2007

- Timed automata translator for Uppaal to PVS
- Component-Based Design and Analysis of Embedded Systems with UPPAAL PORT, 2008
- Verification of COMDES-II Systems Using UPPAAL with Model Transformation, 2008
- METAMOC: Modular WCET Analysis Using UPPAAL, 2010.

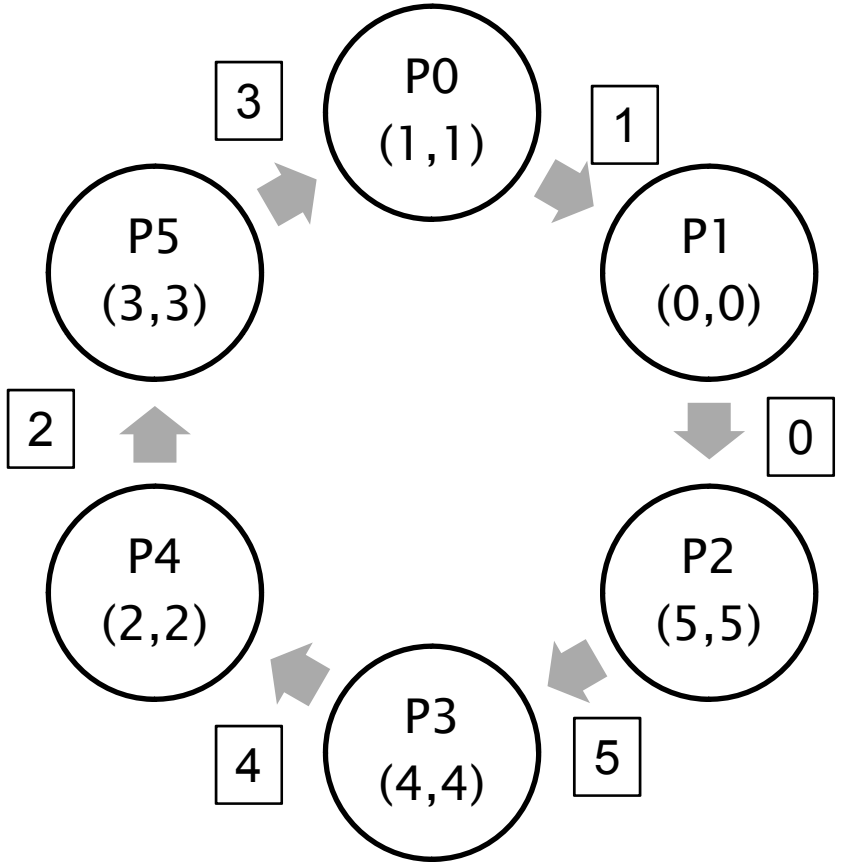
Synchronous Leader Election

Le Lann, Chang, Robert

Ring Topology

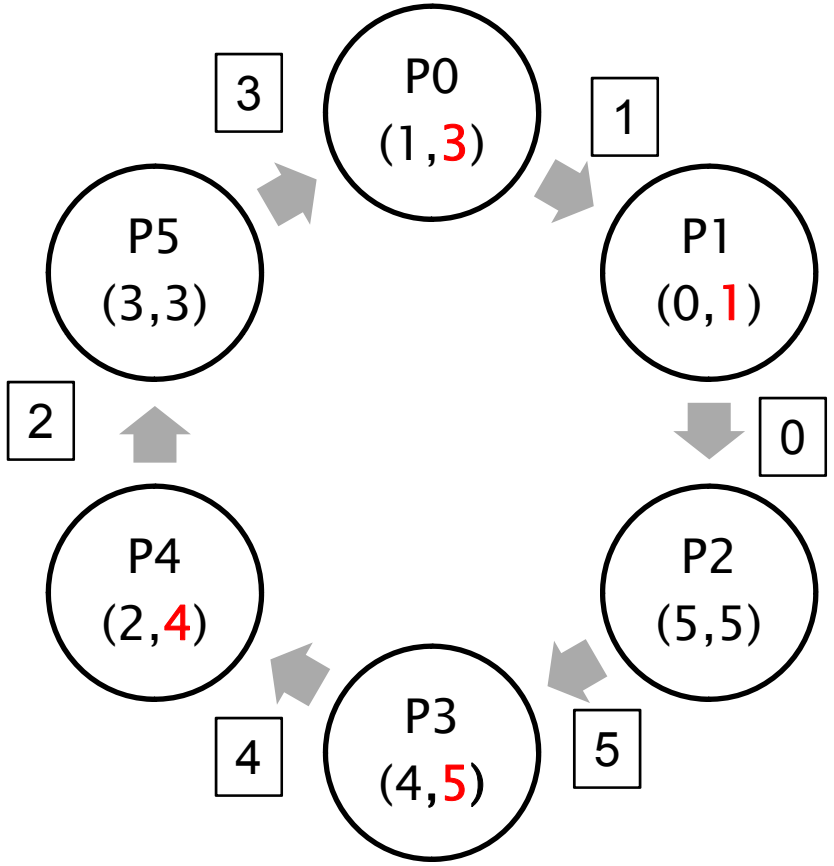


Ring Topology



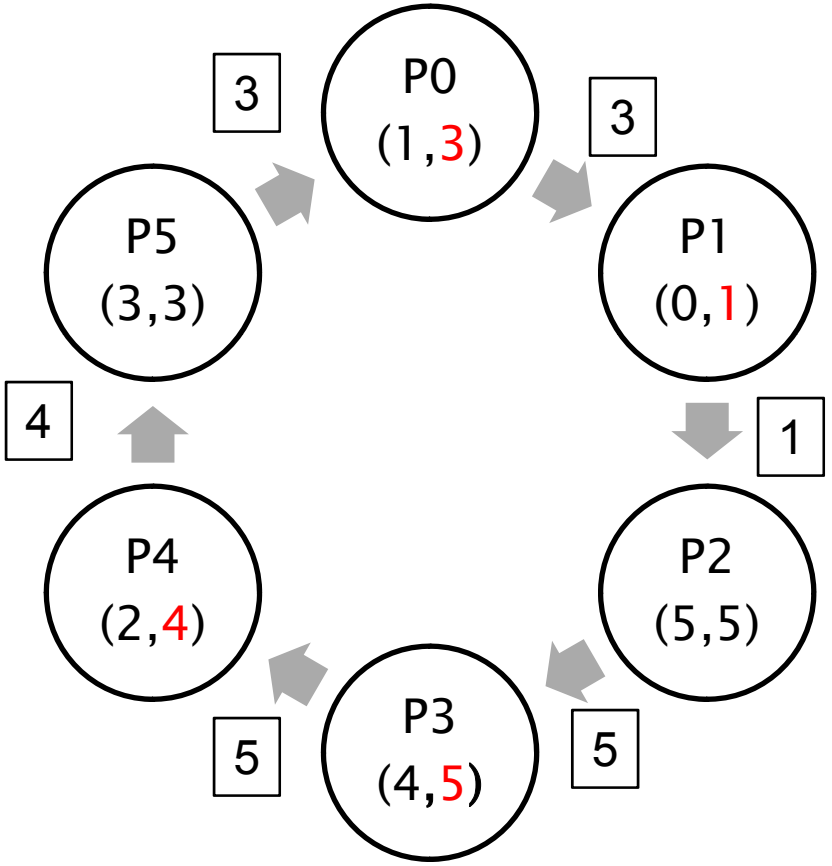
Send

Ring Topology



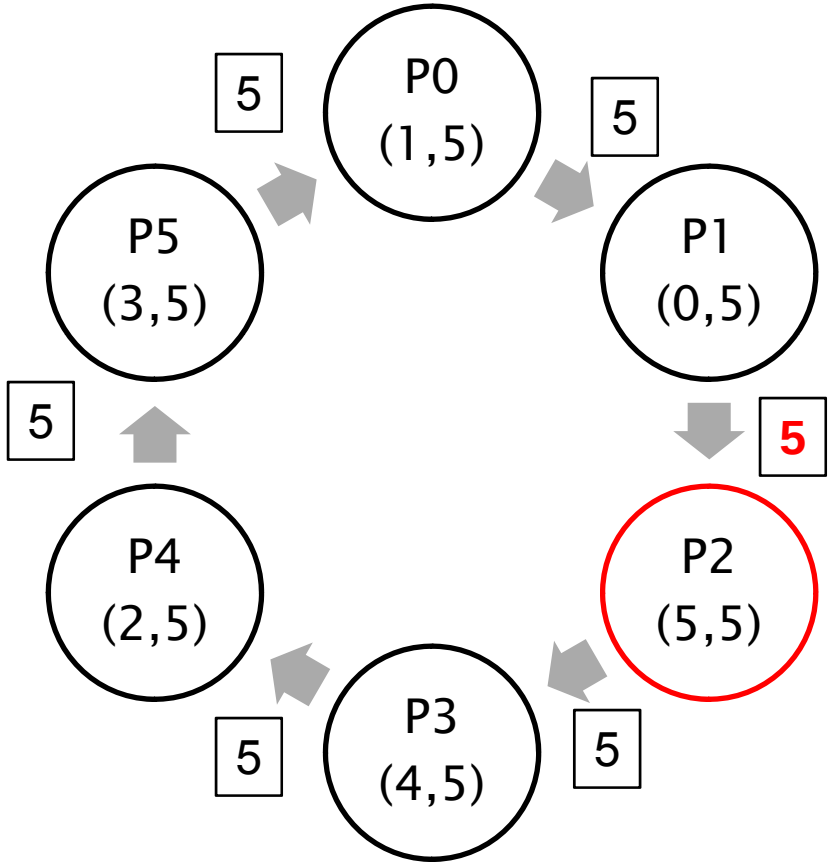
Update

Ring Topology



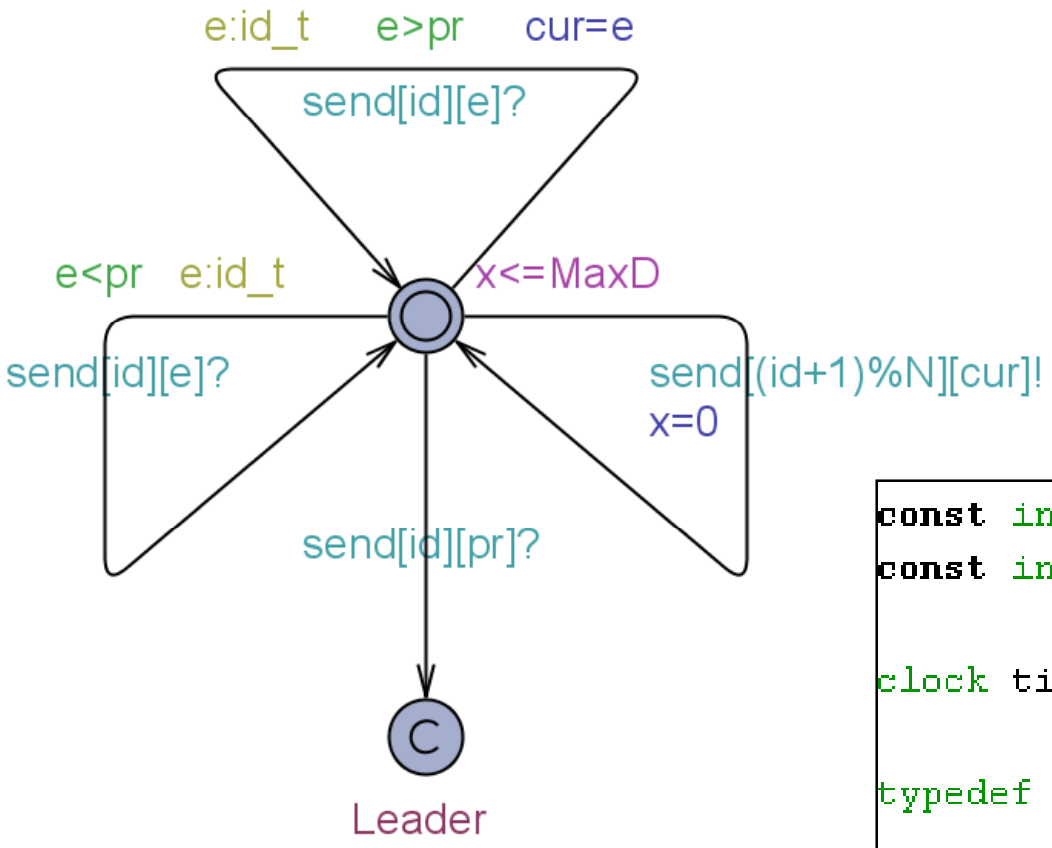
Update

Ring Topology



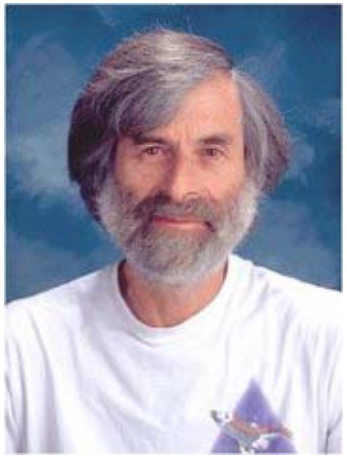
Leader Found

Model of Node (id, pr)



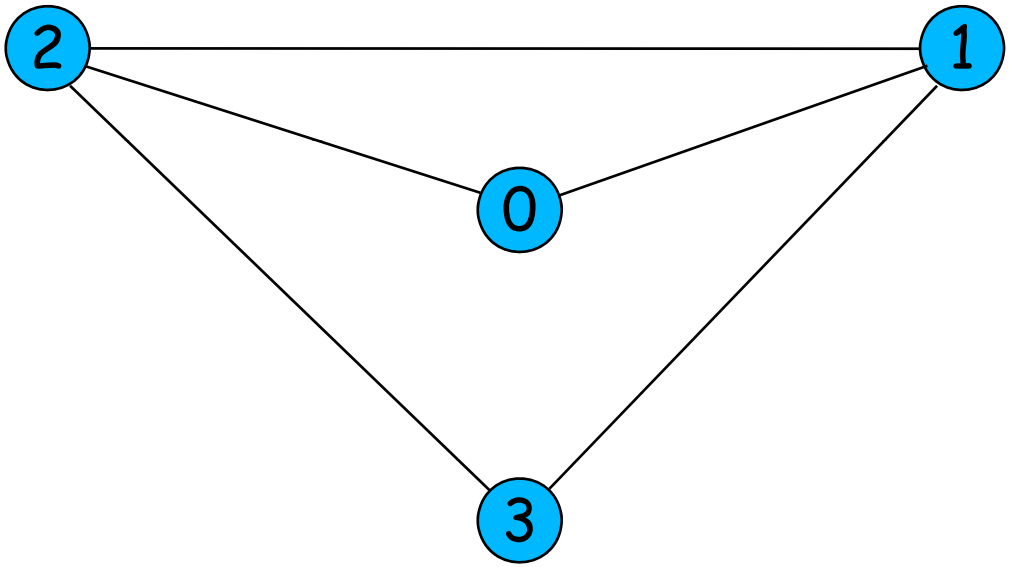

```
const int N = 6;
const int MaxD = 2;

clock time;

typedef int[0,N-1] id_t;

broadcast chan send[N][N];
```

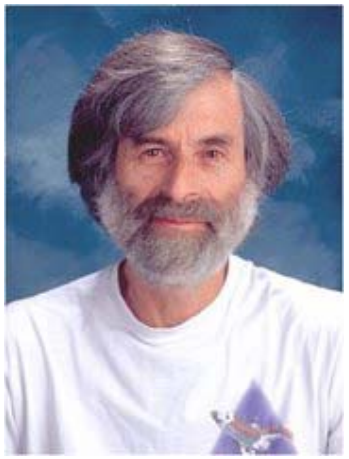
Asynchronous Leader Election



Protocol analysed in UPPAAL by
Leslie Lamport
CHARME'05

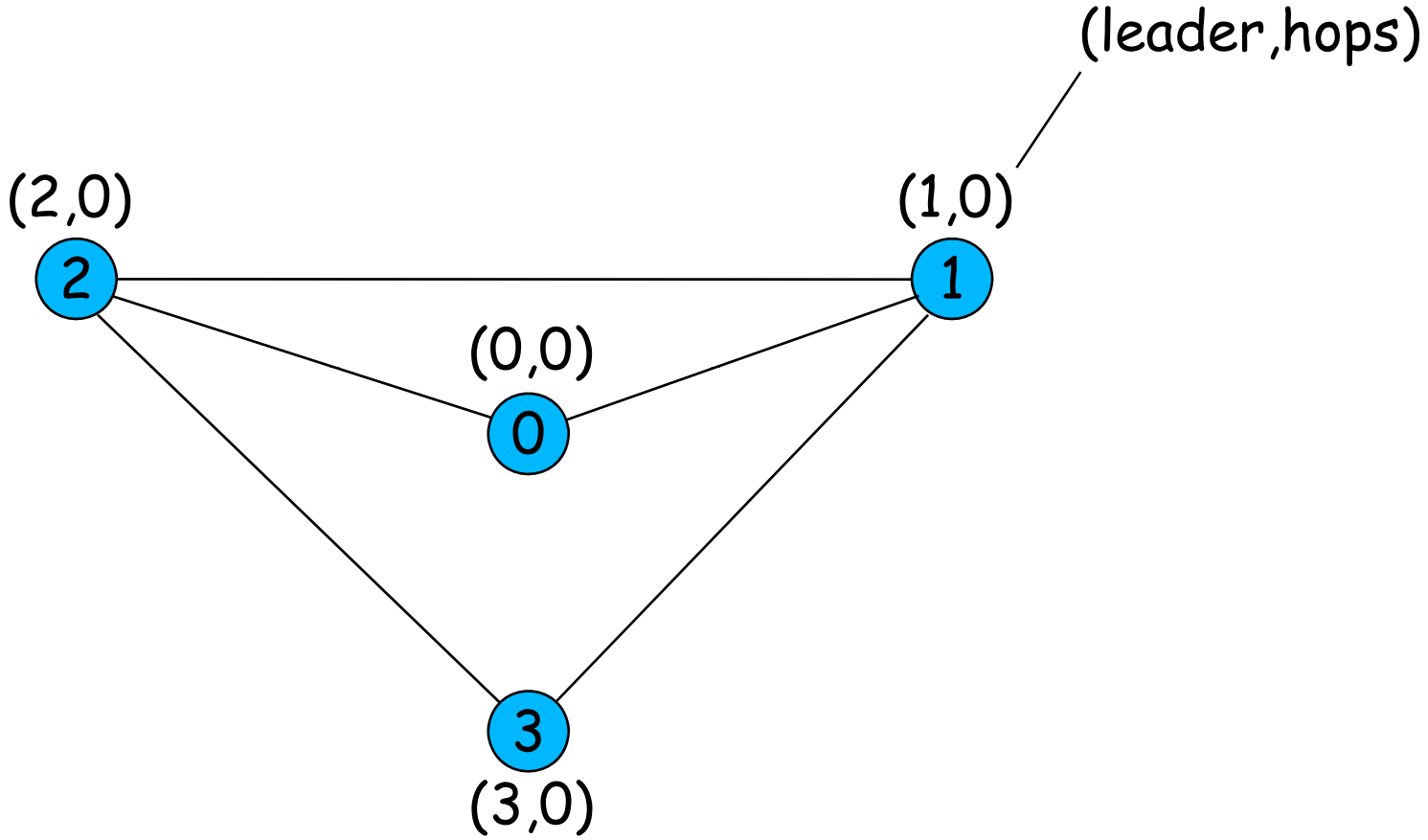
Leader Election



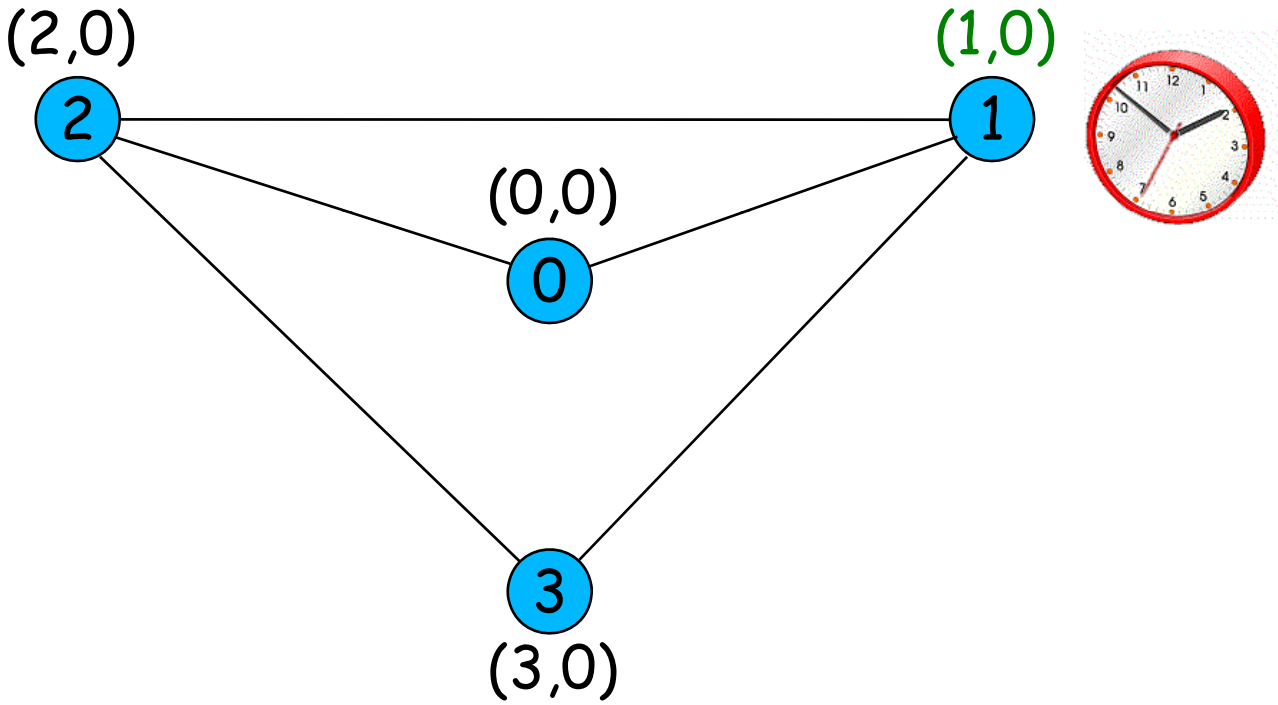


Protocol by
Leslie Lamport

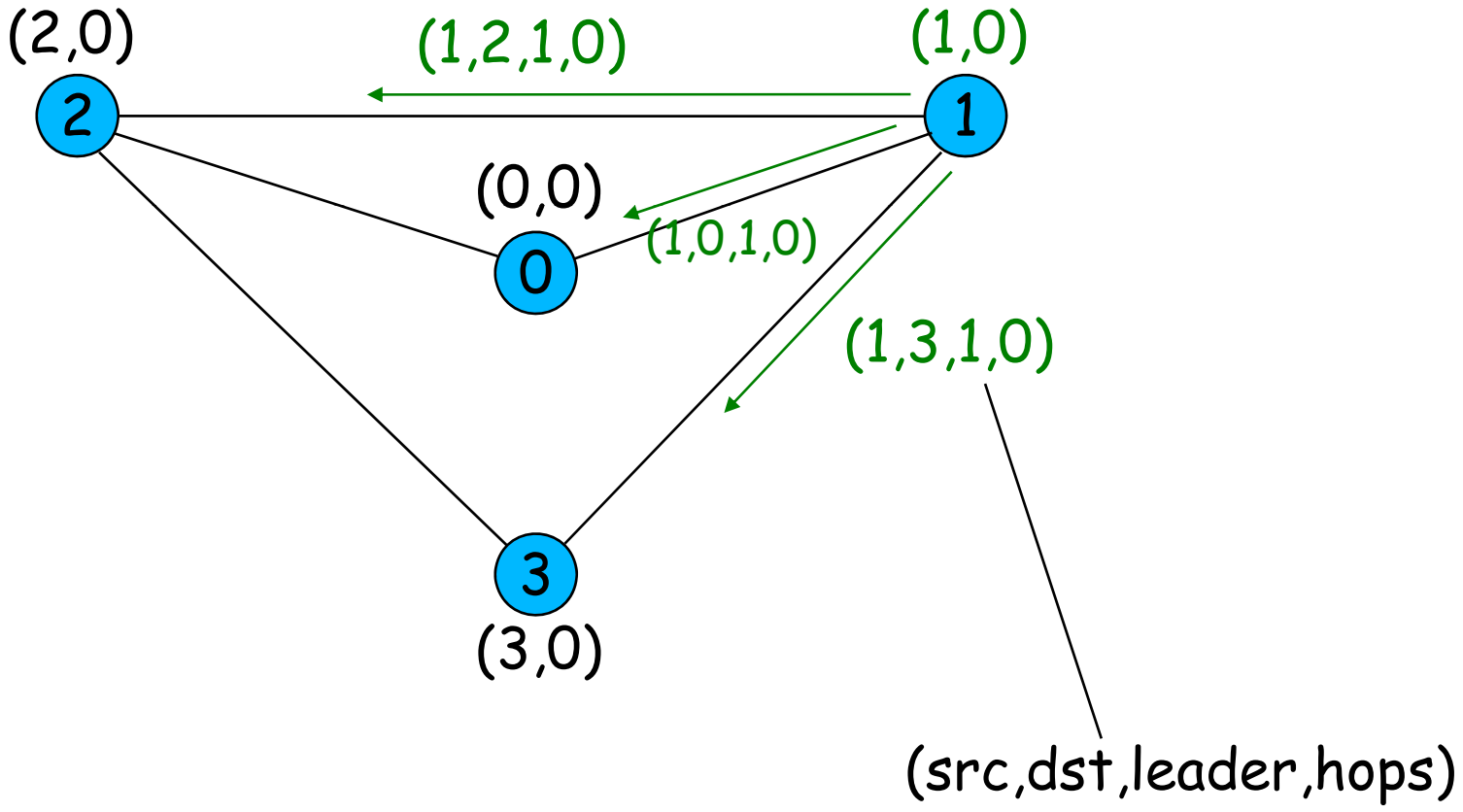
Leader Election



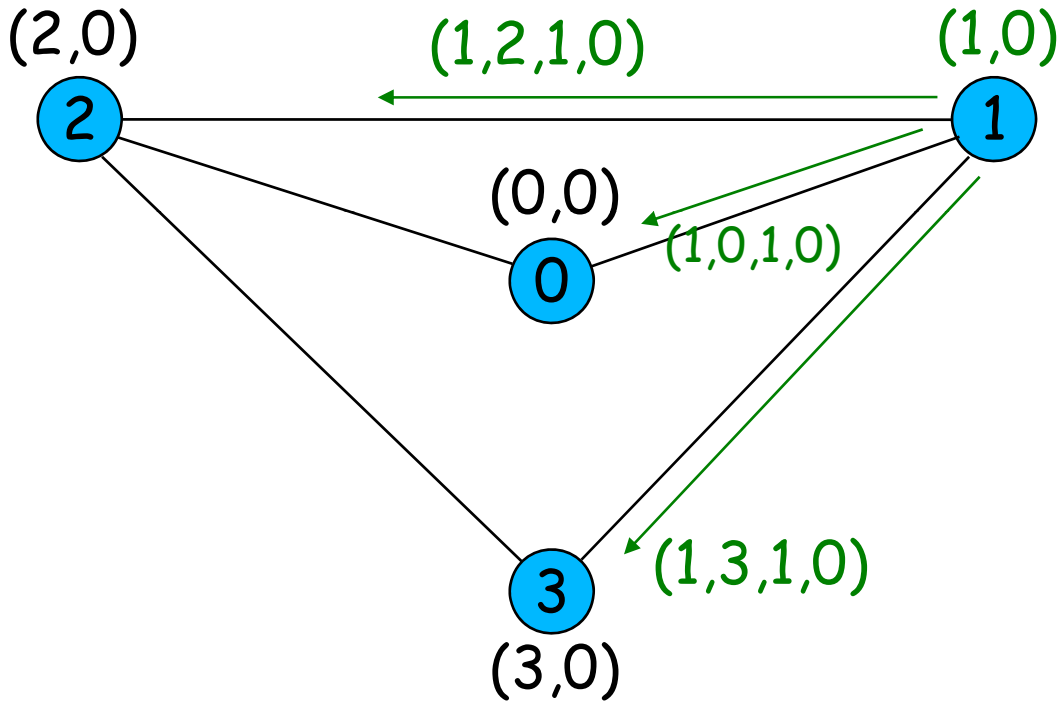
Timeout



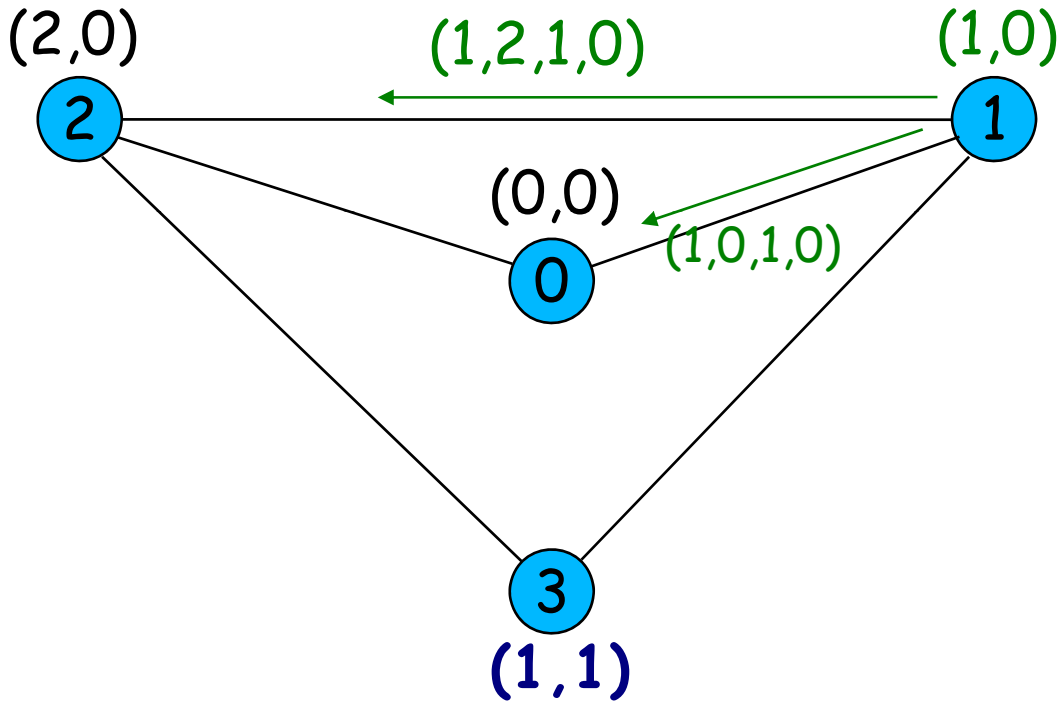
Flooding



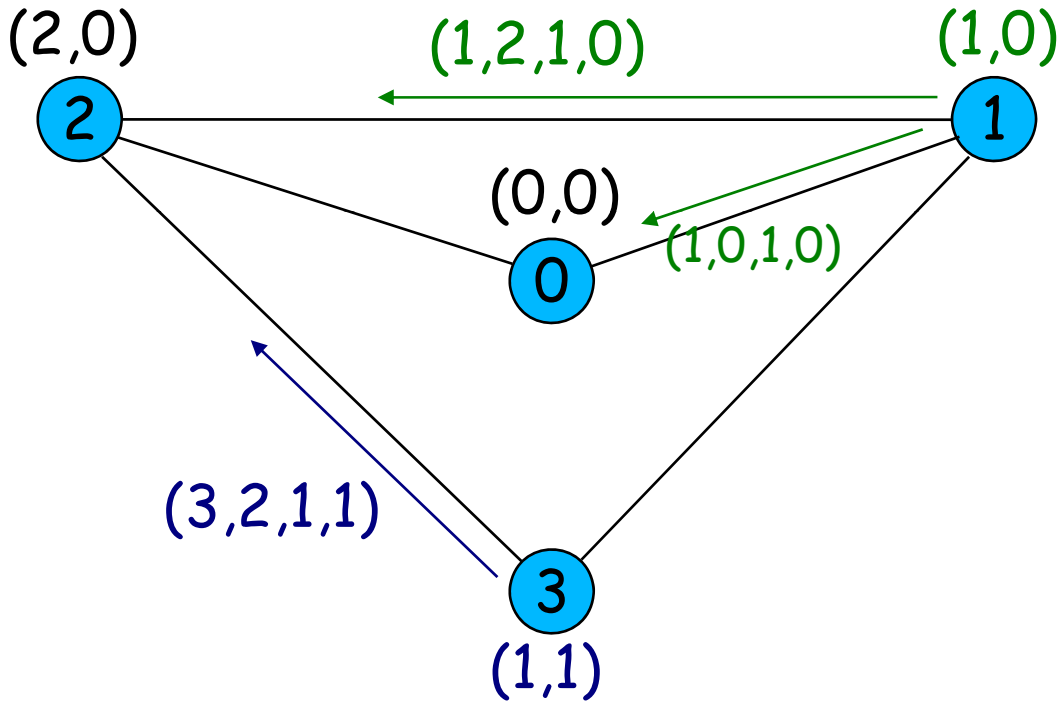
Flooding



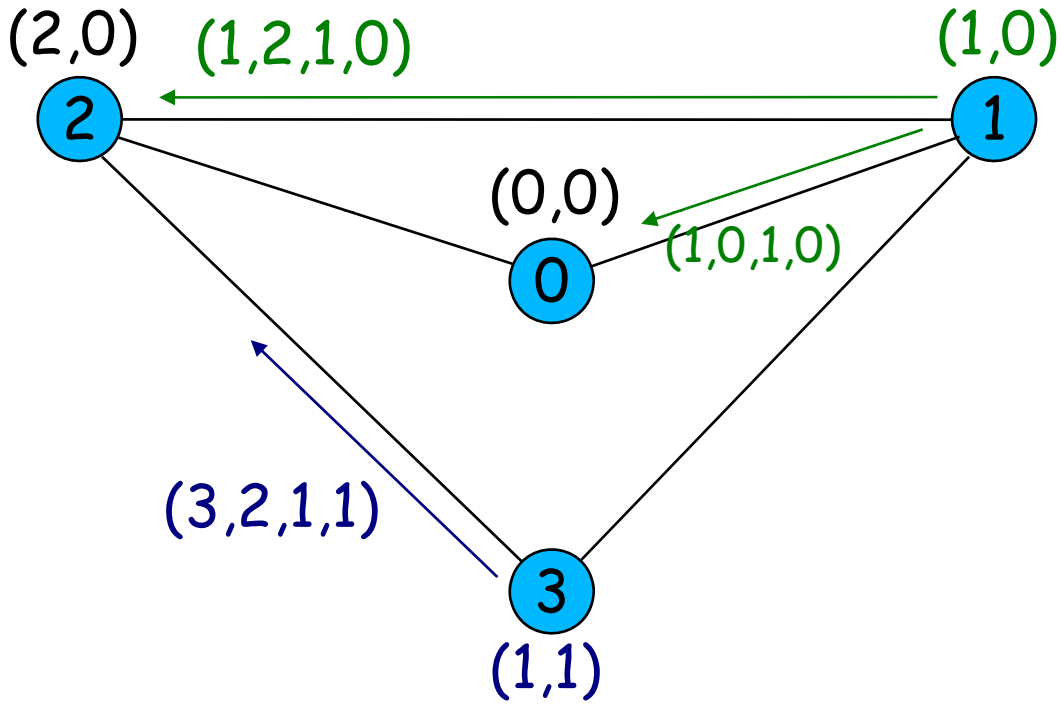
Flooding



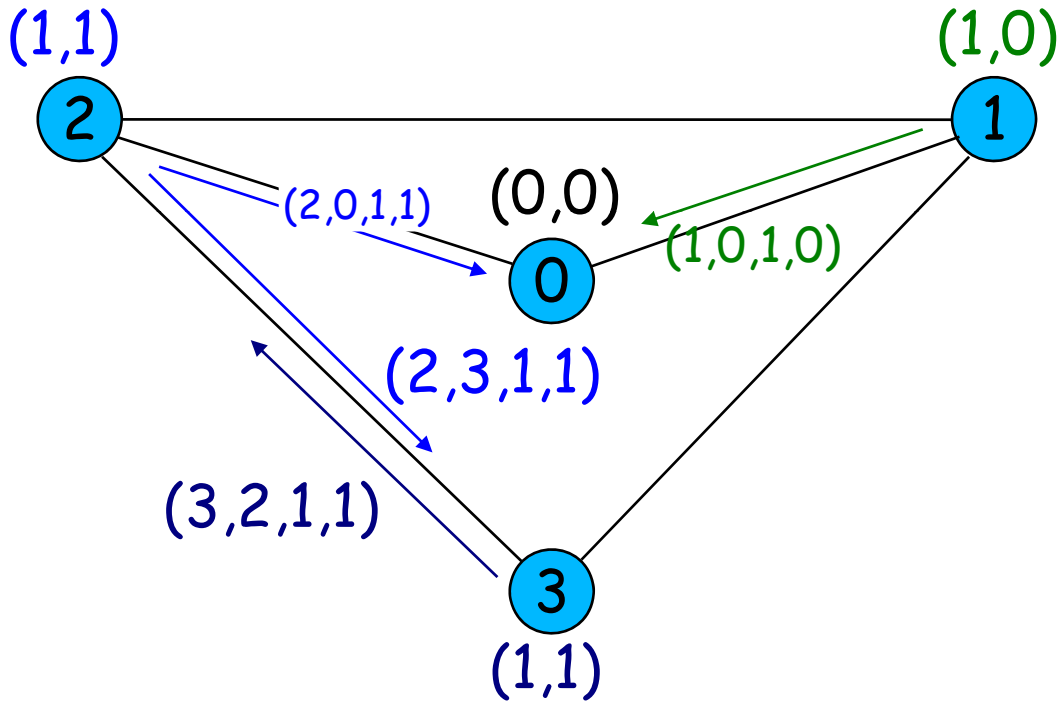
Forwarding



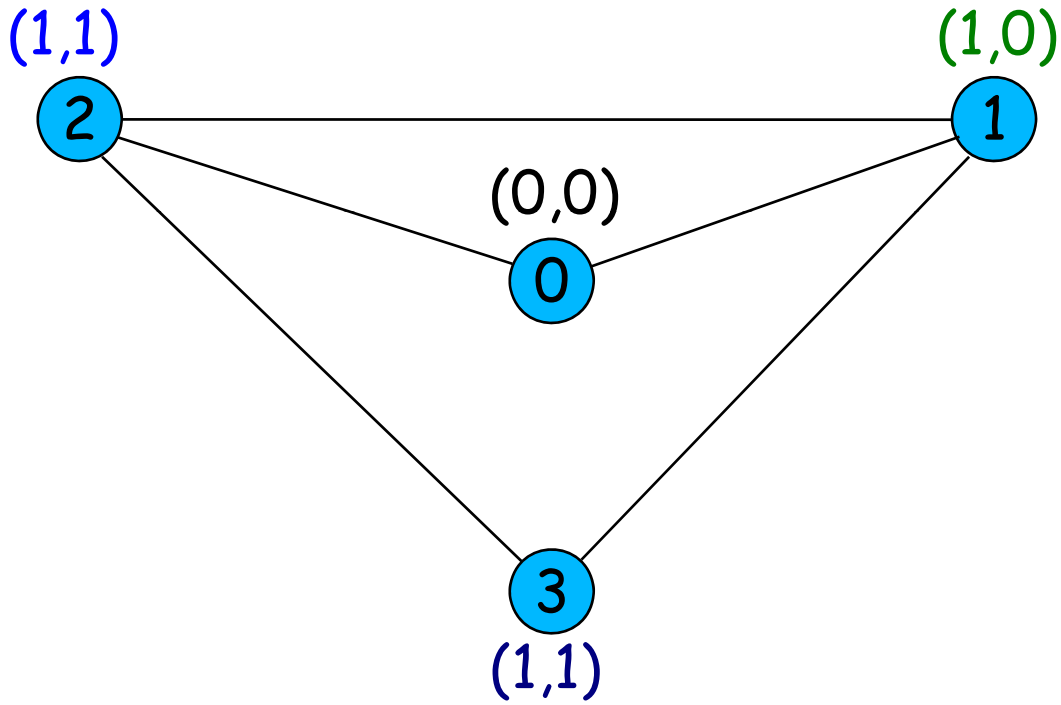
Forwarding



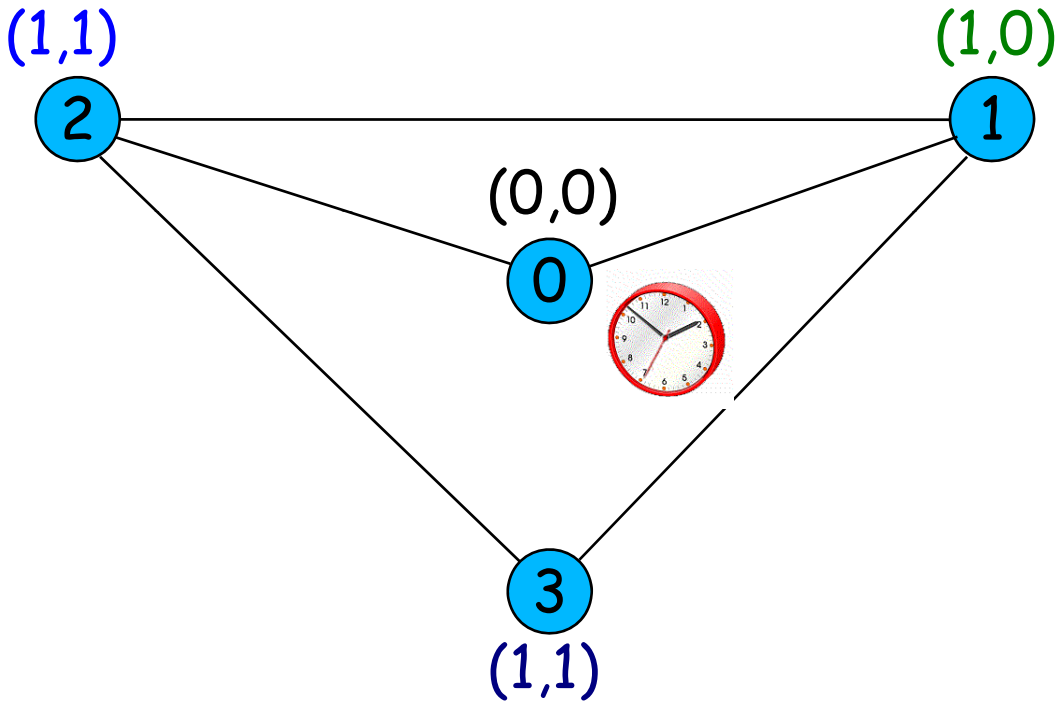
Forwarding



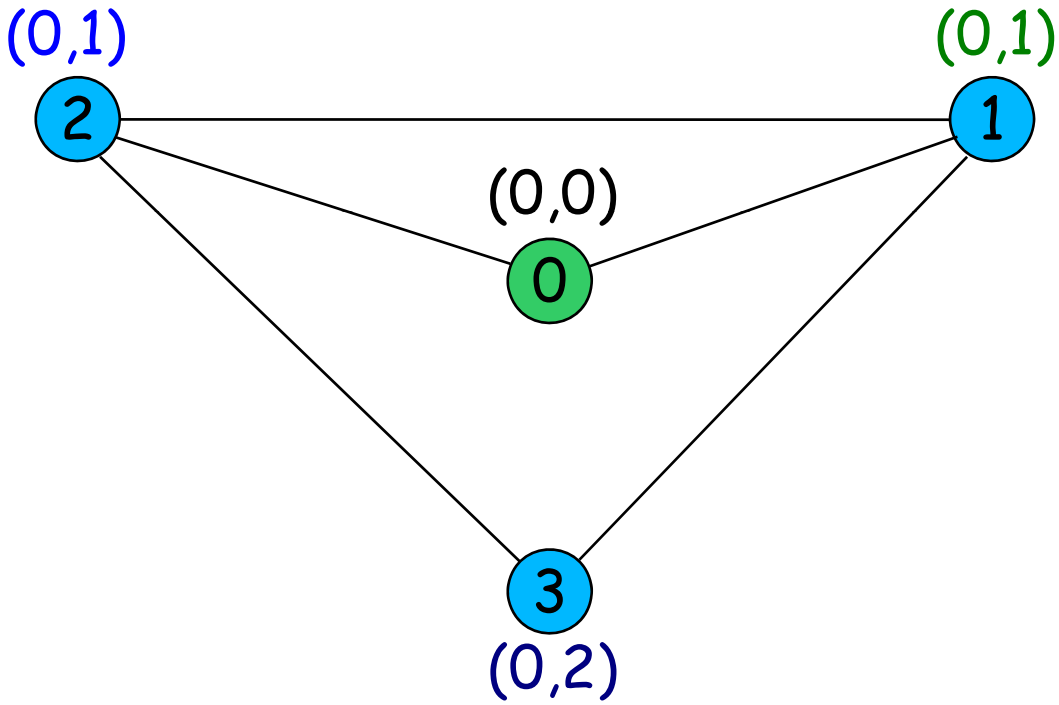
Leader Election



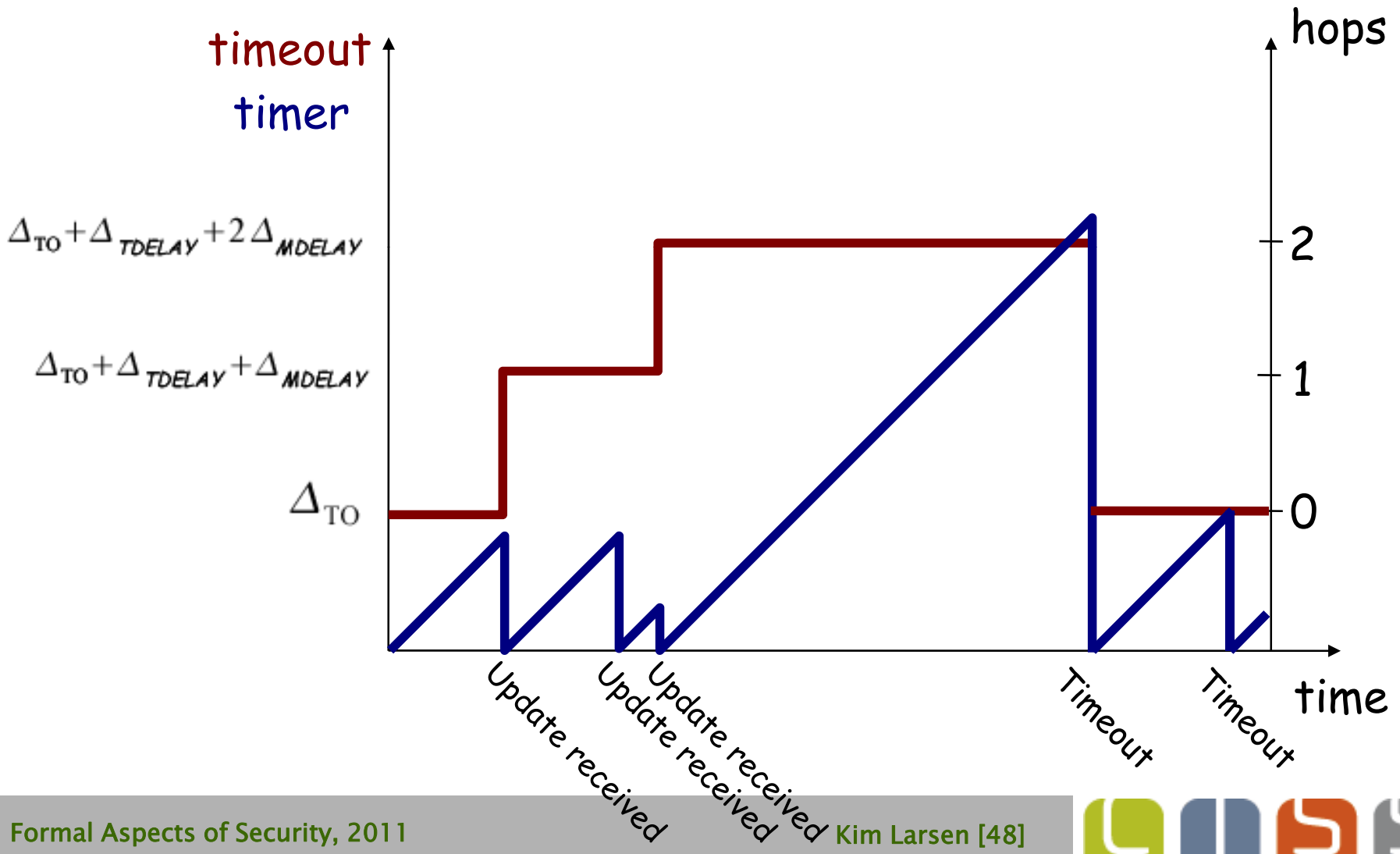
Leader Election



Leader Election



Variable timeout



Leader Election

Claim to be verified

Correct leader is known at a node *i* after

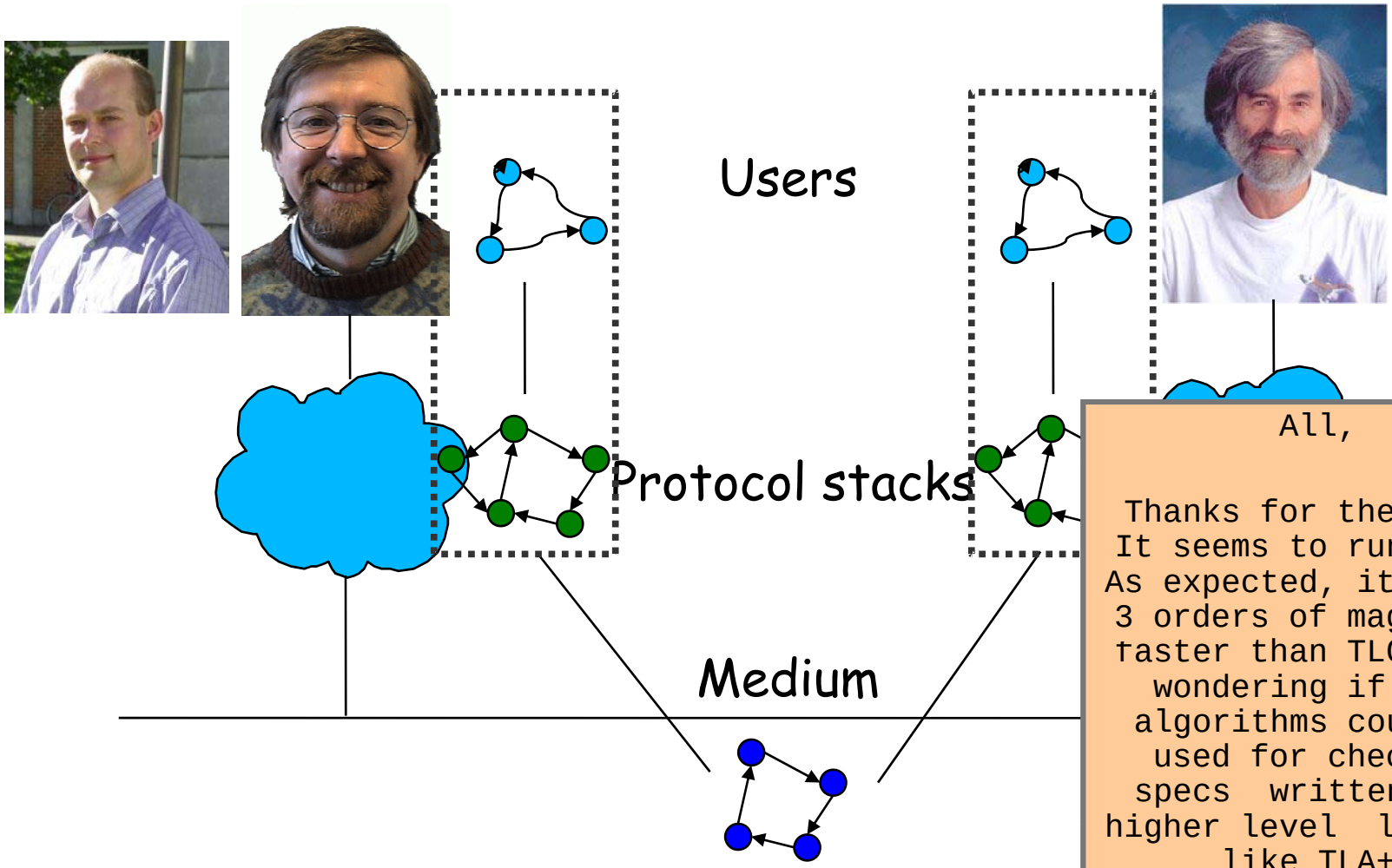
$$t(i) = \Delta_{TO} + \Delta_{TDELAY} + d_i \cdot \Delta_{MDELAY}$$

A model checking problem

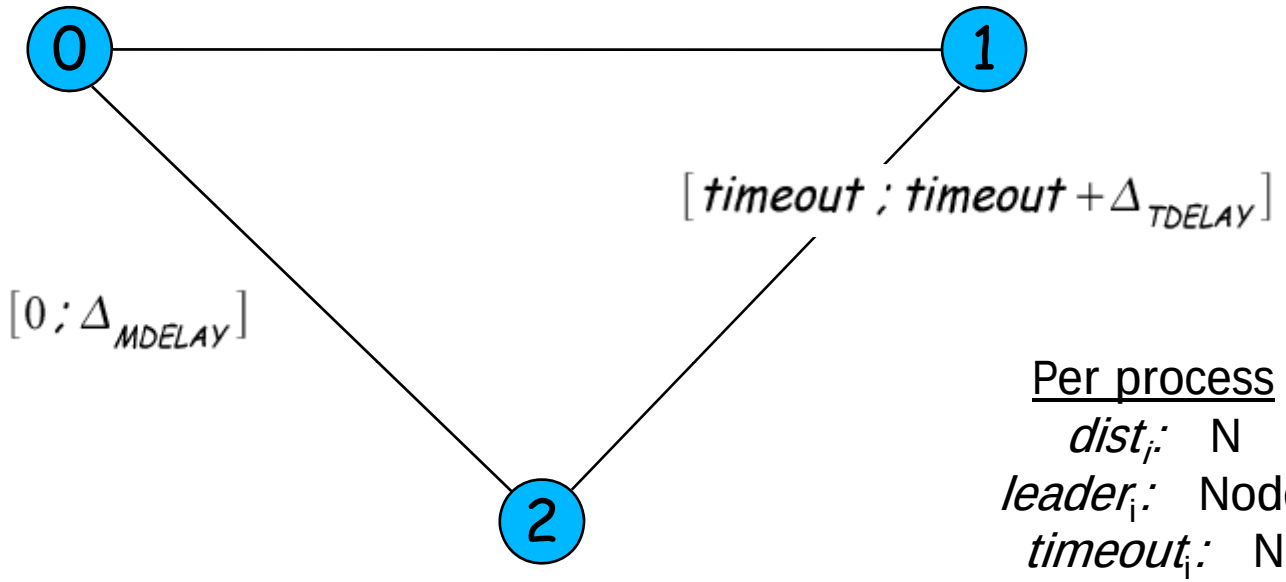
$$IMP \models \Box_{>t(i)}\ l(i)=L(i)$$

for all i.

Modelling (RT) protocols



Modelling the election protocol



Per process

$dist_i$: N

$leader_i$: Node

$timeout_i$: N

Static

$Topology : Node \times Node \rightarrow B$

Message

src: Node

dst: Node

leader: Node

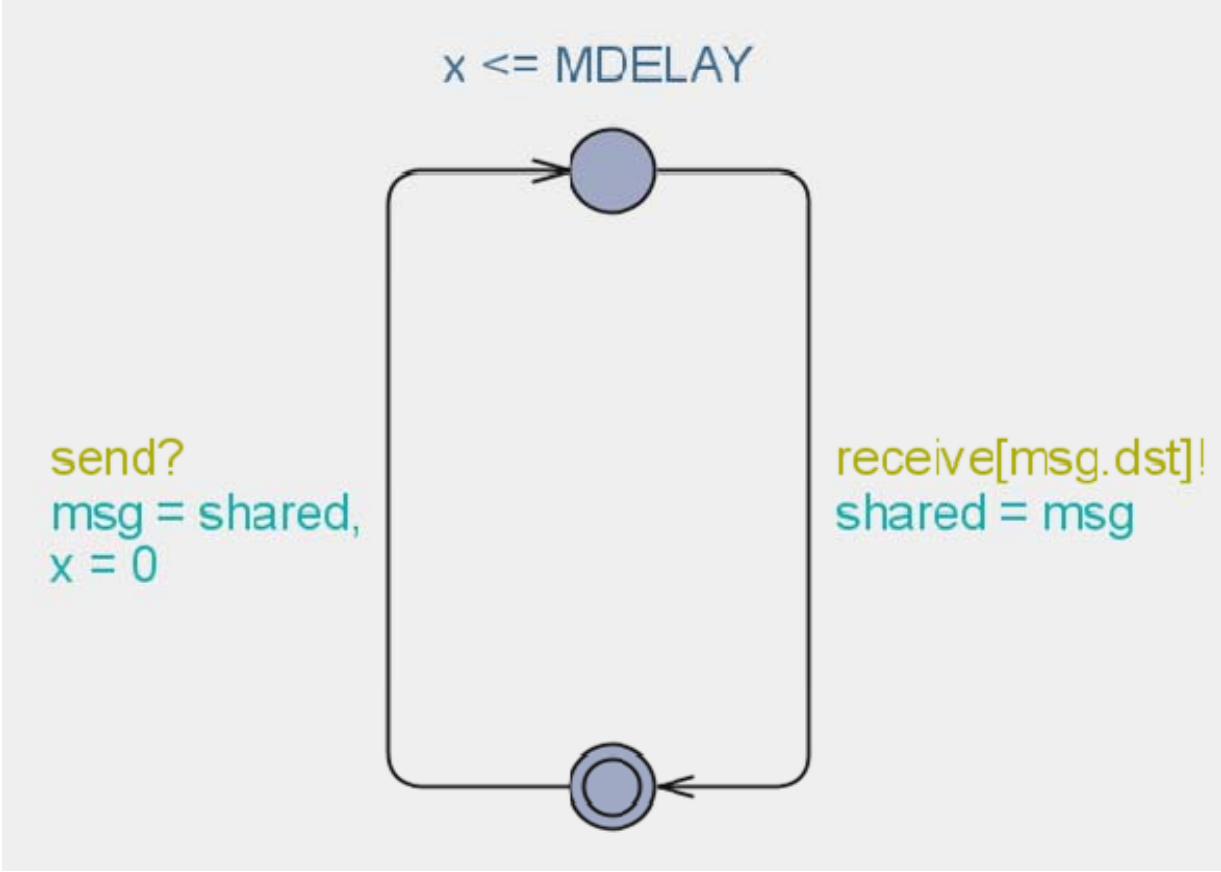
hopss: N

Global Declaration

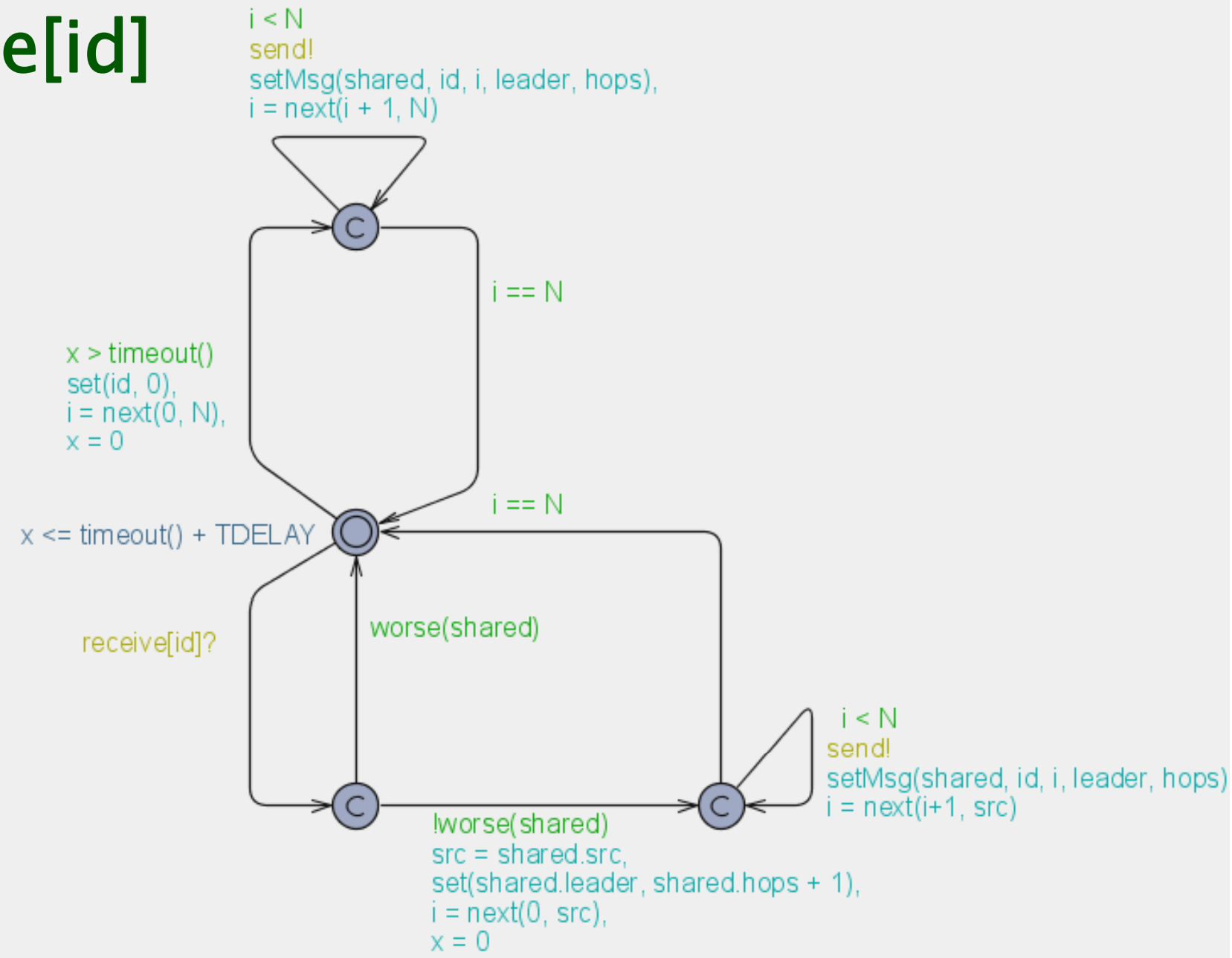
```
void setMsg(msg_t &msg, id_t src, id_t dst, id_t leader, int[0,N] hops)
{
  msg.src = src;
  msg.dst = dst;
  msg.leader = leader;
  msg.hops = hops;
}

chan send;
chan receive[N];
msg_t shared;

const int link[N][N] = {
  { 0,1,1 },
  { 1,0,1 },
  { 1,1,0 }
};
```

```
const int N = 3;
const int MDELAY = 3;
const int TDELAY = 5;
const int TO = 10;

typedef int[0,N-1] id_t;
typedef struct
{
  id_t src;
  id_t dst;
  id_t leader;
  int[0,N] hops;
} msg_t;
```

Message



Node[id]

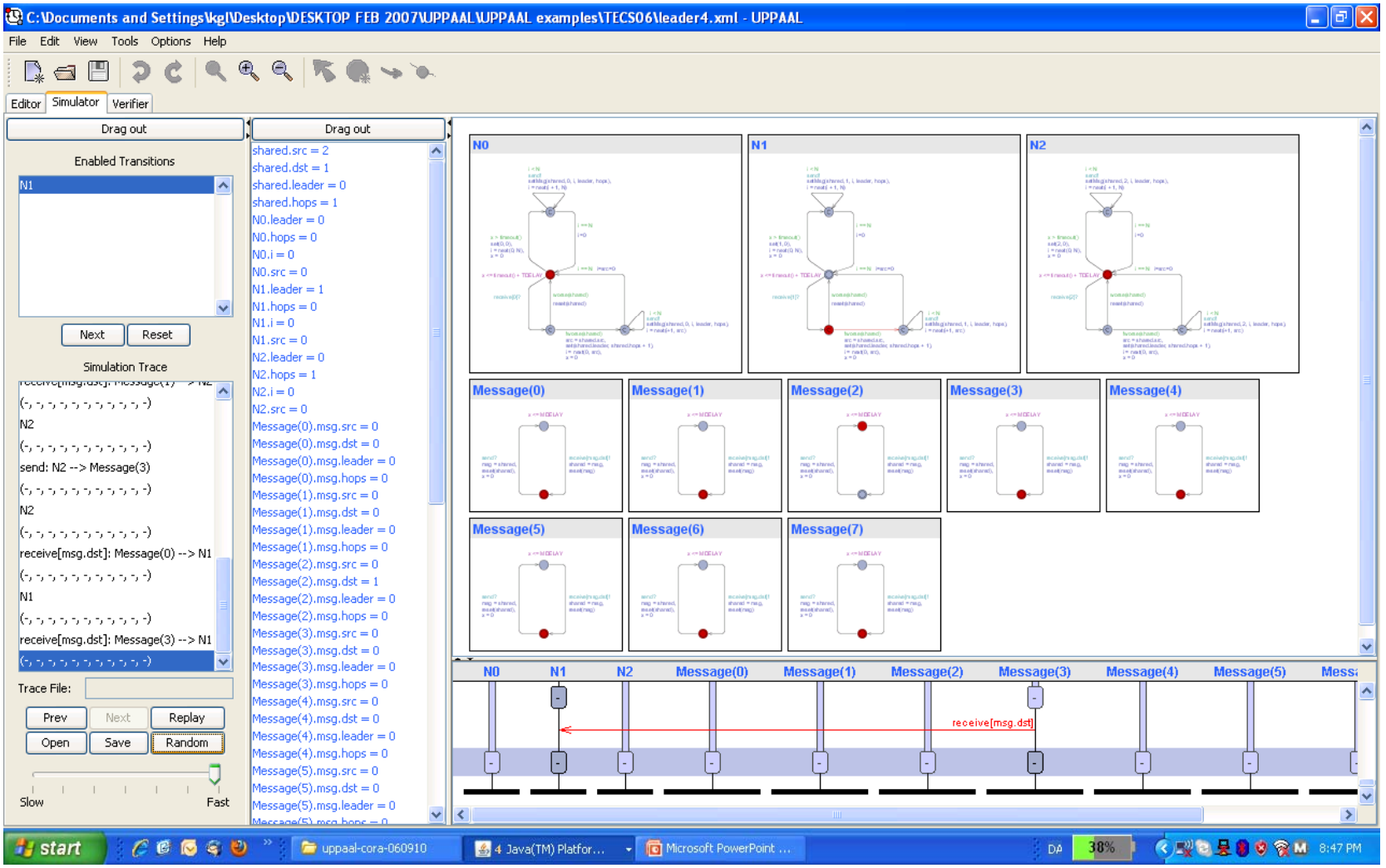

i < N
send!
setMsg(shared, id, i, leader, hops),
i = next(i + 1, N)
i == N
x > timeout()
set(id, 0),
i = next(0, N),
x = 0
i == N
x <= timeout() + TDELAY
receive[id]?
worse(shared)
i < N
send!
setMsg(shared, id, i, leader, hops)
i = next(i+1, src)
!worse(shared)
src = shared.src,
set(shared.leader, shared.hops + 1),
i = next(0, src),
x = 0

Local Declarations (Node[id])

```
id_t leader = id;
int[0,N] hops;
clock x;
int[0,N] i;
id_t src;

void set(id_t l, int[0,N] h)
{
  leader = l;
  hops = h;
}

int[0,N] next(int[0,N] i,int[0,N] src)
{
  while (i  < N && (!link[id][i] || i == src))
  {
    i++;
  }
  return i;
}
```

```
int[0,1000] timeout()
{
  if (hops > 0)
    return TO + TDELAY + hops * MDELAY;
  return TO;
}

bool worse(const msg_t &msg)
{
  return msg.leader > leader || msg.leader
       == leader && msg.hops > hops;
}
```

Demo

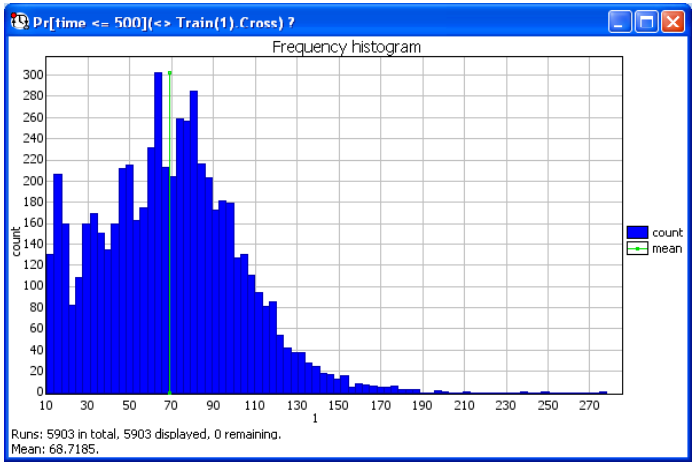

C:\Documents and Settings\kgl\Desktop\DESKTOP FEB 2007\UPPAAL\UPPAAL examples\TECS06\leader4.xml - UPPAAL
File Edit View Tools Options Help
Editor Simulator Verifier
Drag out
Enabled Transitions
N1
Next Reset
Simulation Trace
(-, -, -, -, -, -, -, -, -, -)
N2
(-, -, -, -, -, -, -, -, -, -)
send: N2 --> Message(3)
(-, -, -, -, -, -, -, -, -, -)
N2
(-, -, -, -, -, -, -, -, -, -)
receive[msg.dst]: Message(0) --> N1
(-, -, -, -, -, -, -, -, -, -)
N1
(-, -, -, -, -, -, -, -, -, -)
receive[msg.dst]: Message(3) --> N1
(-, -, -, -, -, -, -, -, -, -)
Trace File:
Prev Next Replay
Open Save Random
Slow Fast
Drag out
shared.src = 2
shared.dst = 1
shared.leader = 0
shared.hops = 1
N0.leader = 0
N0.hops = 0
N0.i = 0
N0.src = 0
N1.leader = 1
N1.hops = 0
N1.i = 0
N1.src = 0
N2.leader = 0
N2.hops = 1
N2.i = 0
N2.src = 0
Message(0).msg.src = 0
Message(0).msg.dst = 0
Message(0).msg.leader = 0
Message(0).msg.hops = 0
Message(1).msg.src = 0
Message(1).msg.dst = 0
Message(1).msg.leader = 0
Message(1).msg.hops = 0
Message(2).msg.src = 0
Message(2).msg.dst = 1
Message(2).msg.leader = 0
Message(2).msg.hops = 0
Message(3).msg.src = 0
Message(3).msg.dst = 0
Message(3).msg.leader = 0
Message(3).msg.hops = 0
Message(4).msg.src = 0
Message(4).msg.dst = 0
Message(4).msg.leader = 0
Message(4).msg.hops = 0
Message(5).msg.src = 0
Message(5).msg.dst = 0
Message(5).msg.leader = 0
N0
N1
N2
Message(0)
Message(1)
Message(2)
Message(3)
Message(4)
Message(5)
Message(6)
Message(7)
N0 N1 N2 Message(0) Message(1) Message(2) Message(3) Message(4) Message(5)
receive[msg.dst]
start
uppaal-cora-060910
4 Java(TM) Platfor...
Microsoft PowerPoint ...
DA
38%
8:47 PM

Optimisations

- **Reducing** the number of active variables
  - If variable is never used until next reset, then the value does not matter.

- **Symmetry** of message processes
  - The message processes are symmetric: It does not matter which is used to transfer a message.

Performance Analysis
using
Statistical Model Checking

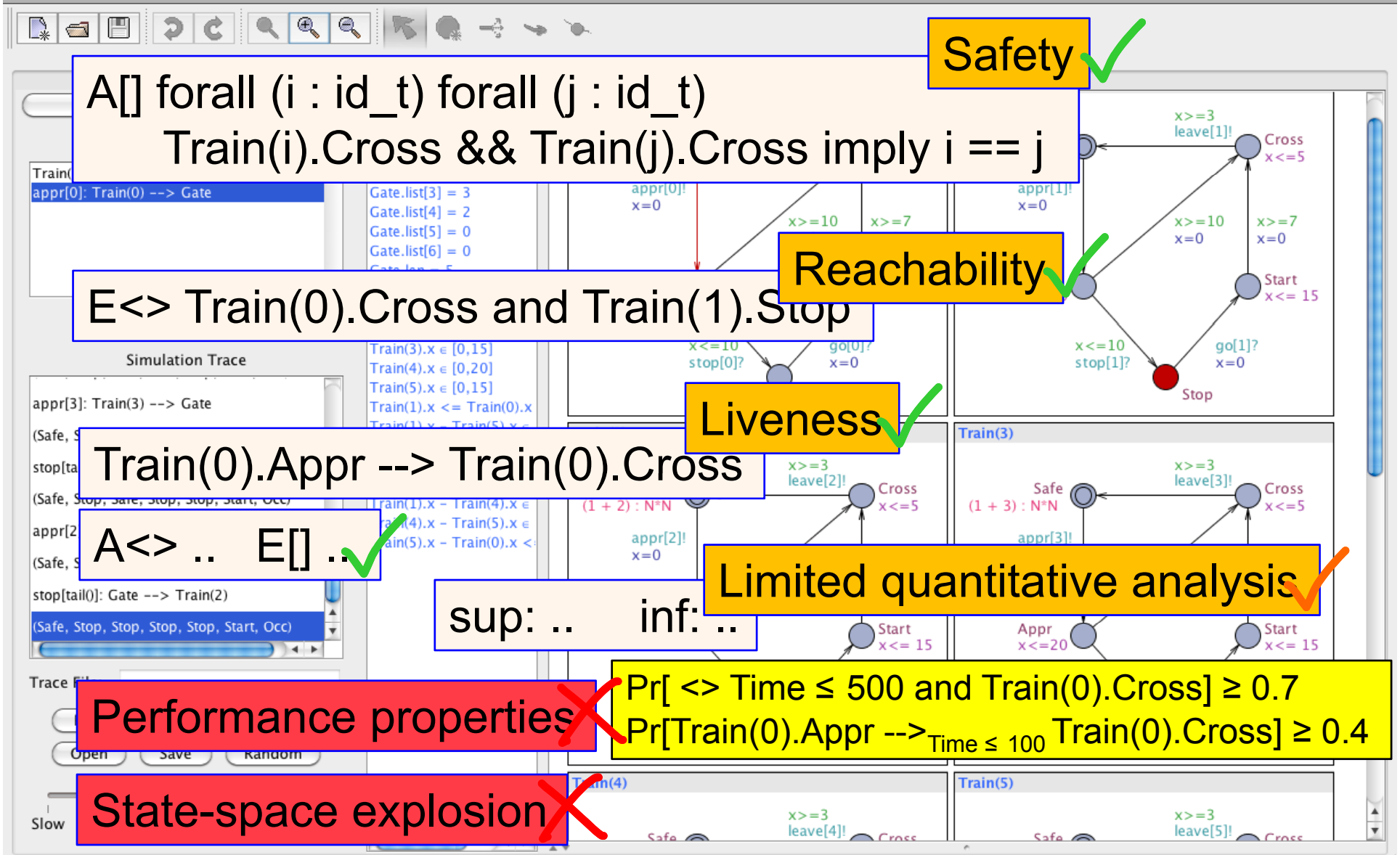


Collaborators:

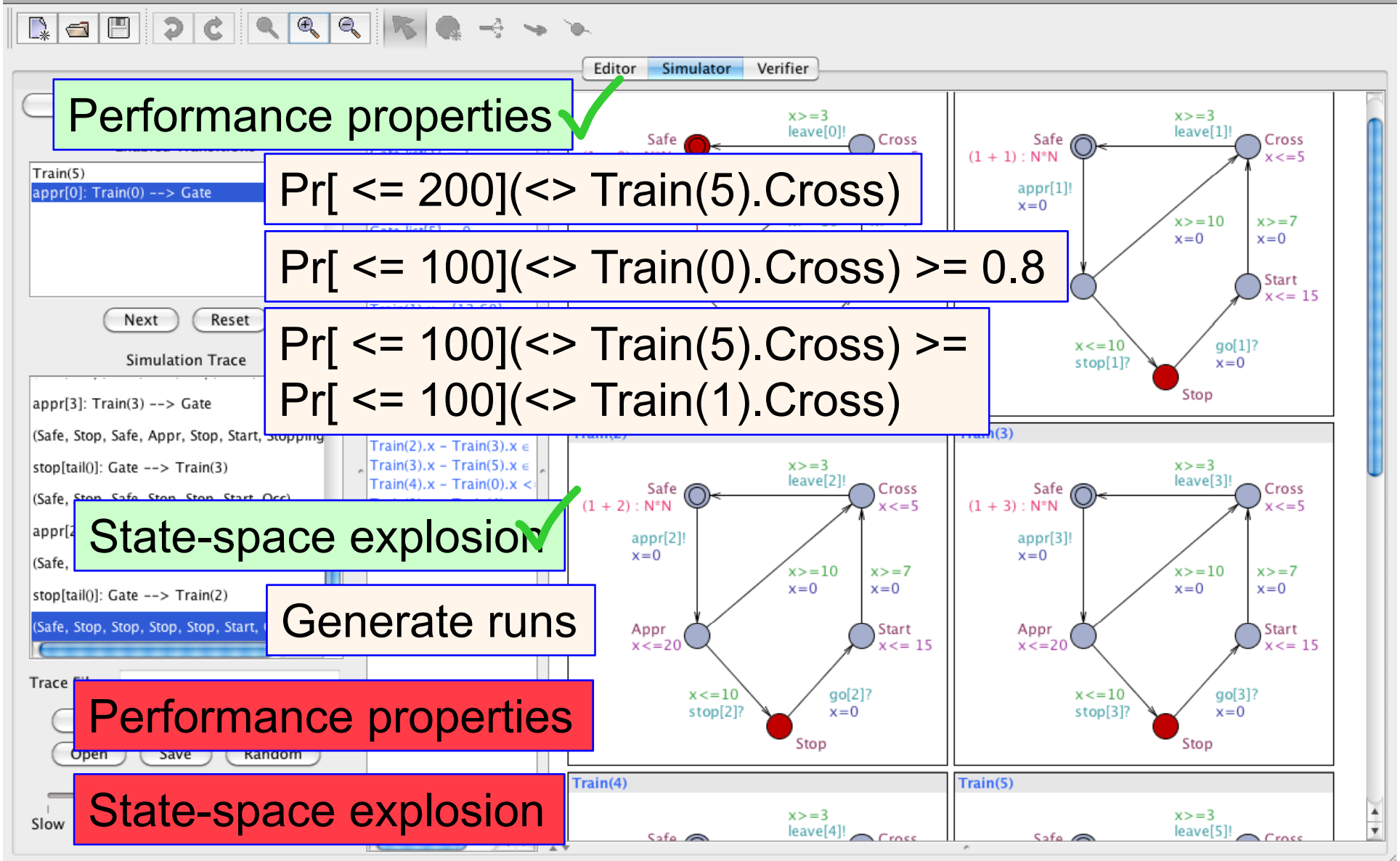
Peter Bulychev, Alexandre David
Axel Legay, Marius Mikucionis
Wang Zheng
Jonas van Vliet, Danny Poulsen

CAV 2011, PDMC 2011,
FORMATS 2011

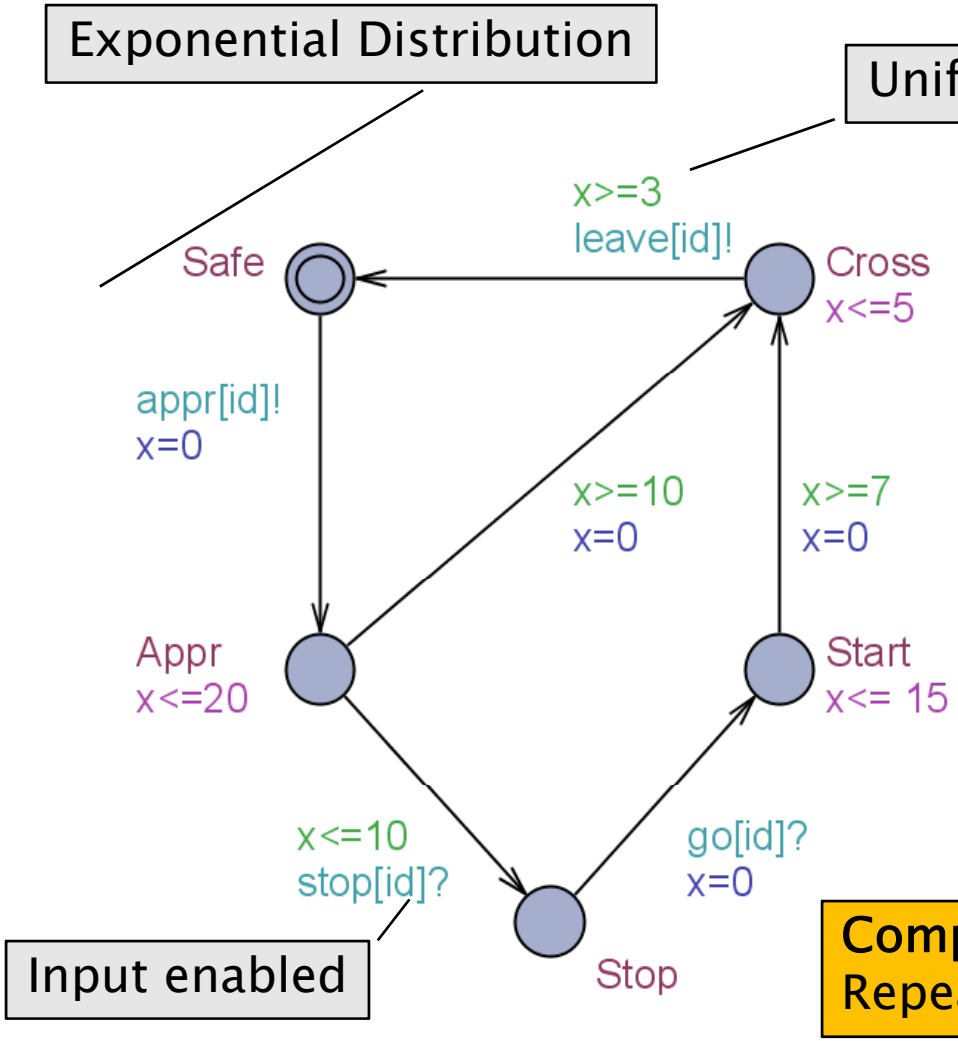
UPPAAL

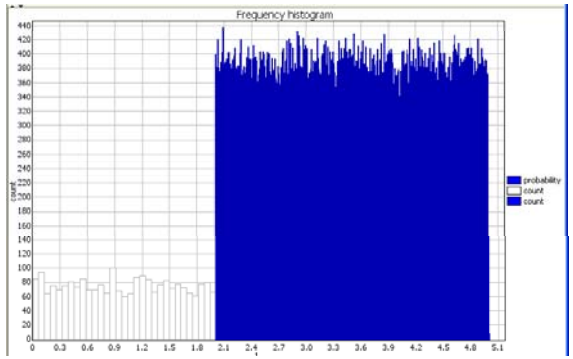

A[] forall (i : id_t) forall (j : id_t)
Train(i).Cross && Train(j).Cross imply i == j
Safety
E<> Train(0).Cross and Train(1).Stop
Reachability
Train(0).Appr --> Train(0).Cross
Liveness
A<> .. E[] ..
sup: .. inf: ..
Limited quantitative analysis
Performance properties
Pr[<> Time ≤ 500 and Train(0).Cross] ≥ 0.7
Pr[Train(0).Appr -->Time ≤ 100 Train(0).Cross] ≥ 0.4
State-space explosion
Simulation Trace
appr[0]: Train(0) --> Gate
appr[3]: Train(3) --> Gate
stop[tail()]: Gate --> Train(2)
(Safe, Stop, Stop, Stop, Stop, Start, Occ)
Open
Save
Random
Slow
Train(3)
Train(4)
Train(5)

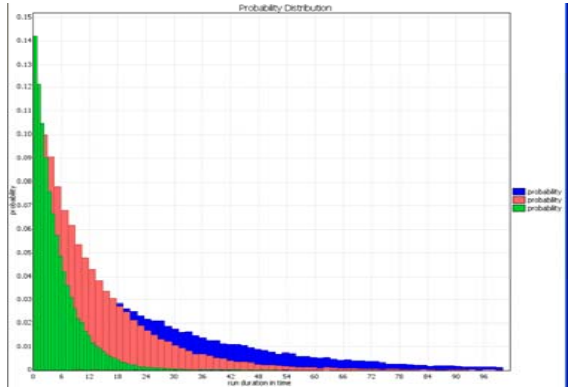
UPPAAL SMC


Editor
Simulator
Verifier
Performance properties
Pr[<= 200](<> Train(5).Cross)
Pr[<= 100](<> Train(0).Cross) >= 0.8
Pr[<= 100](<> Train(5).Cross) >=
Pr[<= 100](<> Train(1).Cross)
State-space explosion
Generate runs
Performance properties
State-space explosion
Train(5)
appr[0]: Train(0) --> Gate
Next
Reset
Simulation Trace
appr[3]: Train(3) --> Gate
stop[tail()]: Gate --> Train(3)
stop[tail()]: Gate --> Train(2)
Open
Save
Random
Slow
Safe
Cross
Appr
Start
Stop
leave[0]!
leave[1]!
leave[2]!
leave[3]!
leave[4]!
leave[5]!
appr[1]!
appr[2]!
appr[3]!
stop[1]?
stop[2]?
stop[3]?
go[1]?
go[2]?
go[3]?
Train(4)
Train(5)

Stochastic Semantics of TA



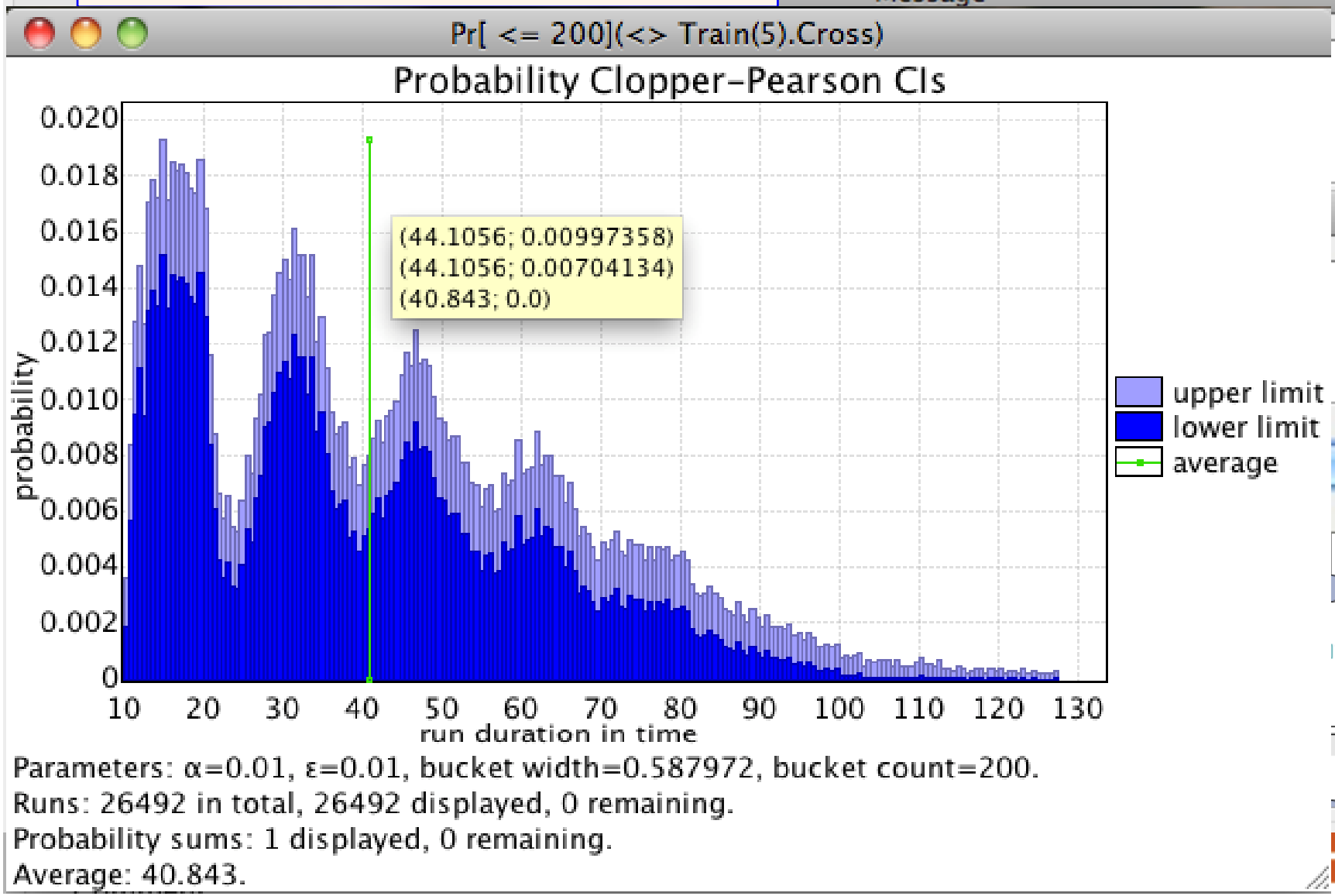




Composition =
Repeated races between components

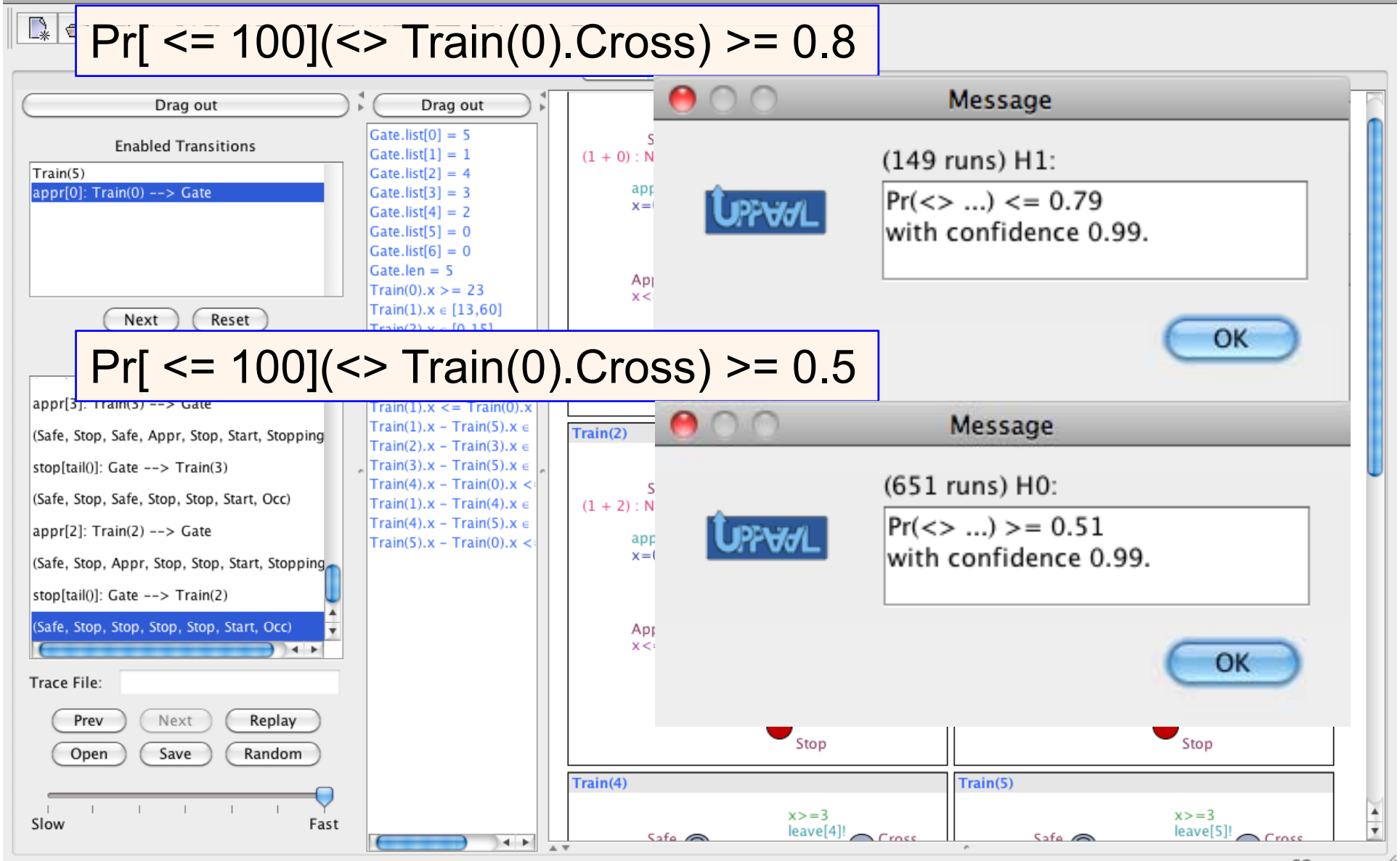
Queries in UPPAAL SMC

Pr[<= 200](<> Train(5).Cross)



Queries in UPPAAL SMC

Pr[<= 100](<> Train(0).Cross) >= 0.8

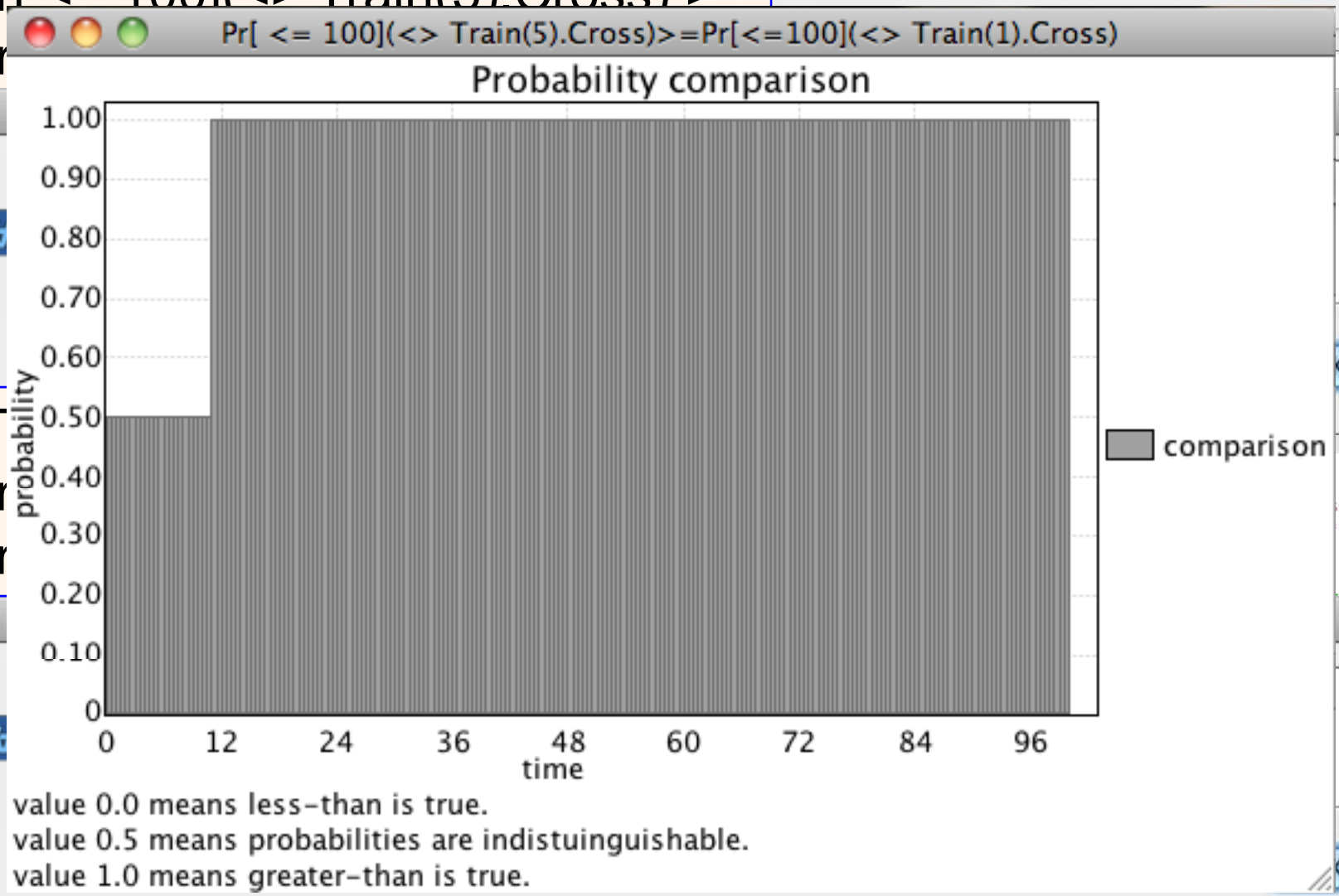


Pr[<= 100](<> Train(0).Cross) >= 0.5

Queries in UPPAAL SMC

Pr[<= 100](<> Train(5).Cross) >=

Pr

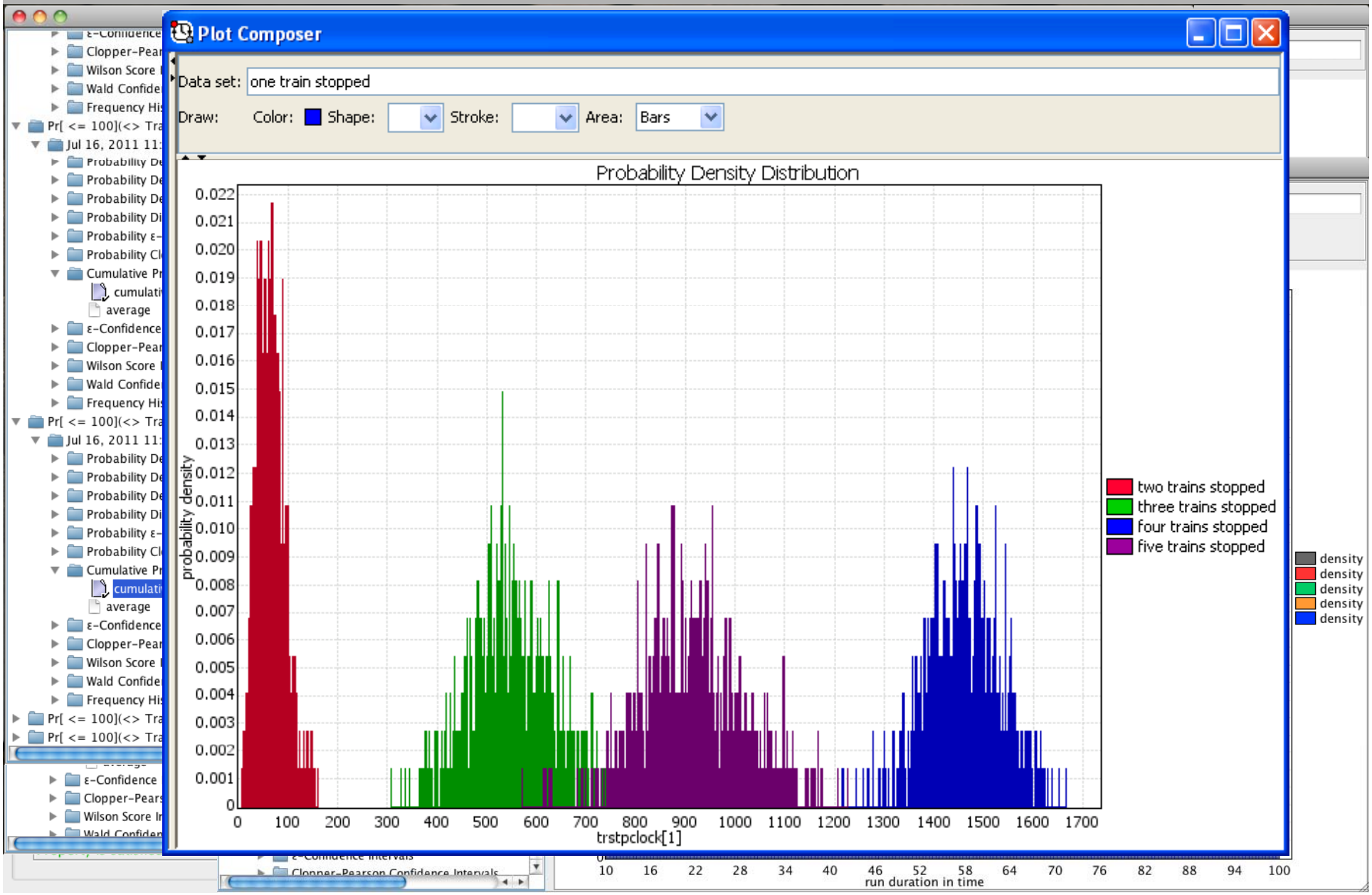


∀

Pr

Pr

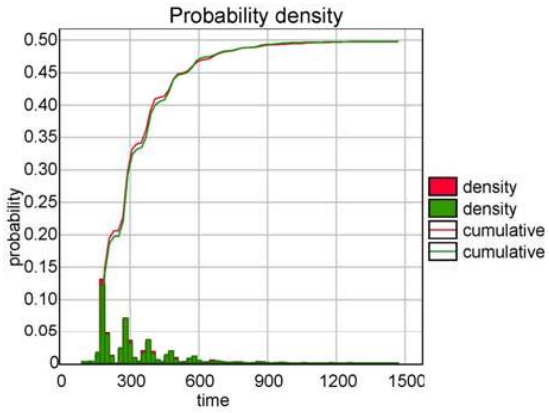
Analysis Tool: Plot Composer



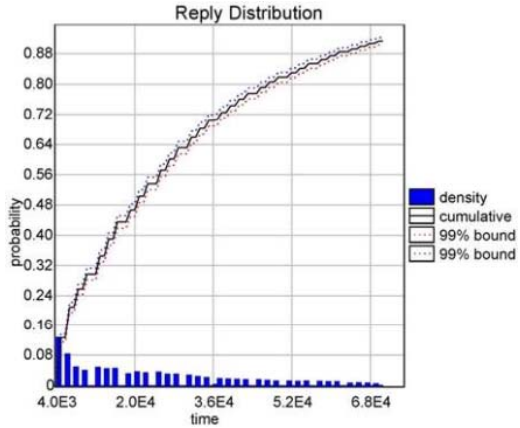
SMC in UPPAAL 4.1.4

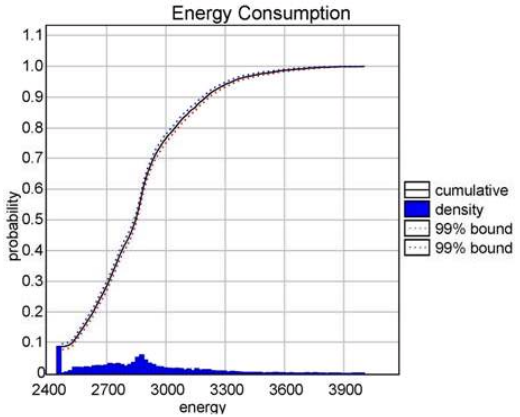
- Constant Slope Timed Automata
  - Clocks may have different (integer) slope in different locations.
  - Branching edges with discrete probabilities (weights).
  - Beyond Priced TA, Energy TA. Equal LHA in (non-stochastic) expressive power.
  - Beyond DTMC, beyond CTMC (with multiple rewards)
- All features of UPPAAL supported
  - User defined functions and types
  - Expressions in guards, invariants, clock-rates, delay-rates (rationals), and weights.
- New GUI for plot-composing and exporting.
- Distributed SMC, 64bits.

Case Studies

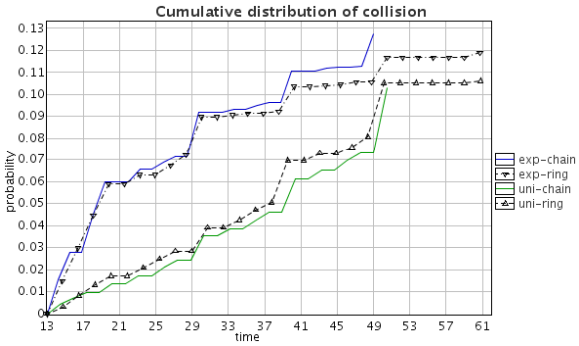


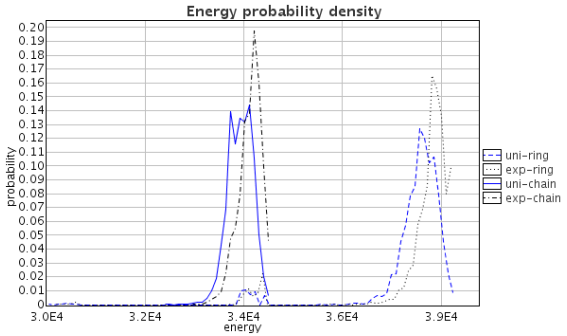
FIREWIRE



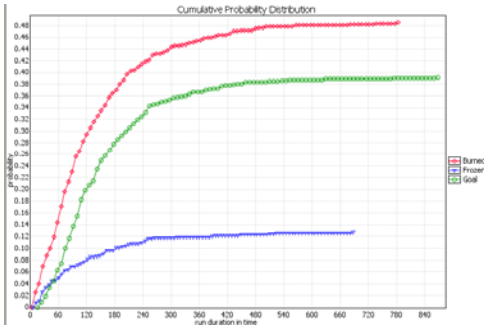


BLUETOOTH





10 node LMAC



100 x 100 ROBOT

The Hammer Game

Alex

Axel