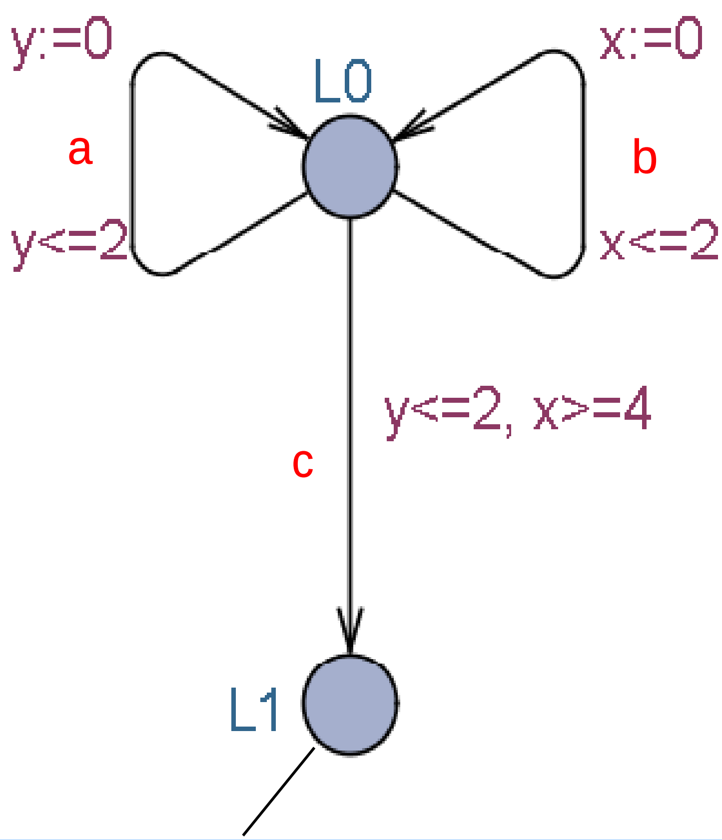
Decidability and Symbolic Verification

Kim G. Larsen
Aalborg University, DENMARK

Overview

- Decidability
  - Region Construction
  - Reachability & Bisimulation Checking
- Symbolic Verification
  - On-the-fly Exploration
  - Zones and Difference Bounded Matrices (DBM)
  - Clock Difference Diagrams (CDD)
- Verification Options

Reachability ?



Reachable from initial state (L0,x=0,y=0) ?

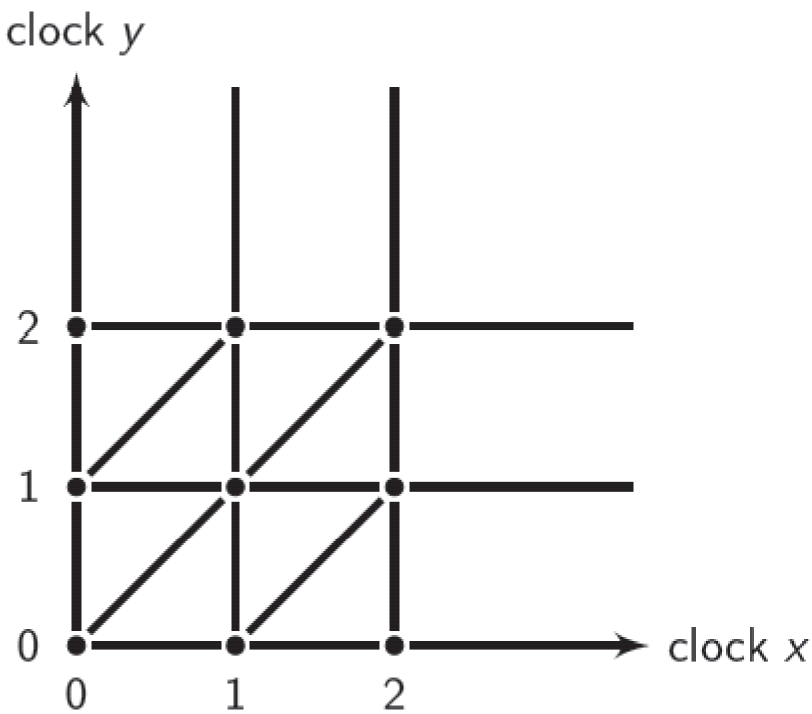
OBSTACLE:
Uncountably infinite
state space

$$L \times \mathbb{R}^C$$

locations

clock-valuations

The Region Abstraction



- "compatibility" between regions and constraints
- "compatibility" between regions and time elapsing

$\rightsquigarrow$ an equivalence of finite index
a time-abstract bisimulation

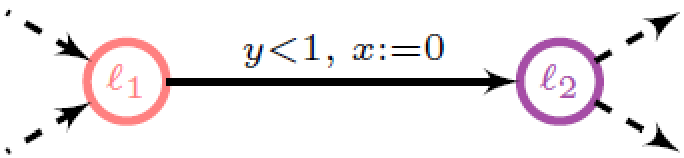
Time Abstracted Bisimulation

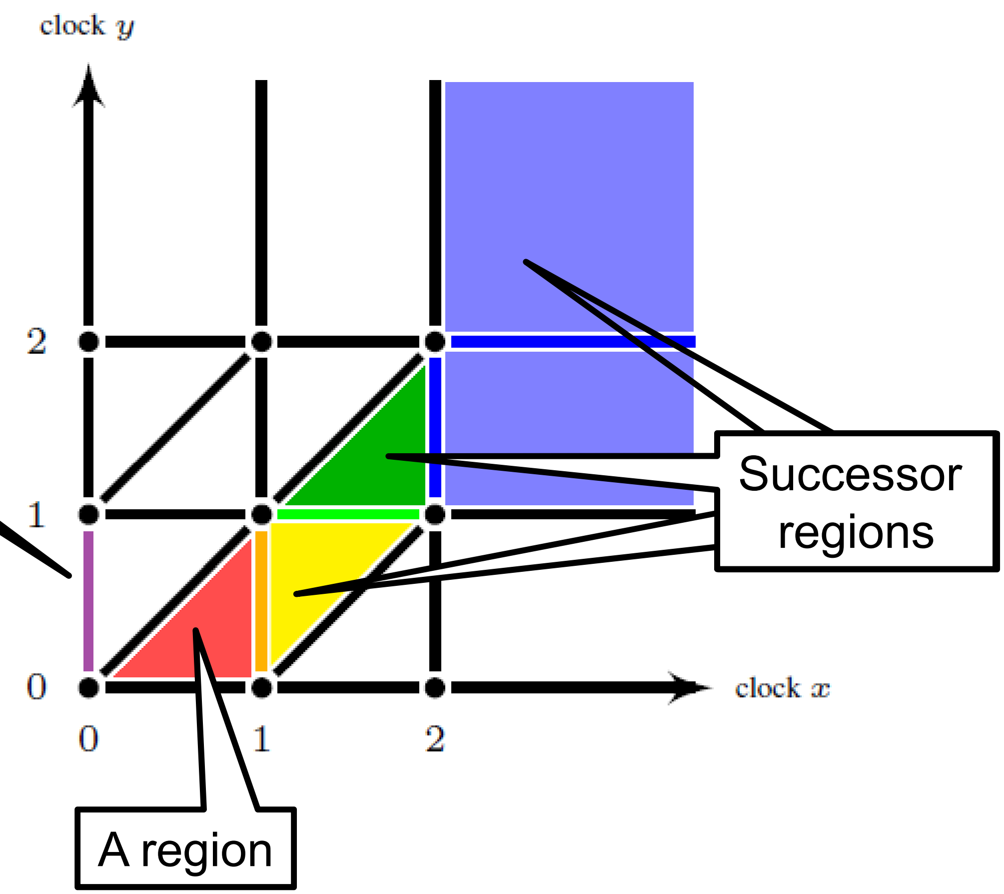
This is a relation between • and • such that:

$$\begin{tikzcd} \forall & \bullet \arrow[r, "a"] \arrow[d, dash, dashed] & \bullet \arrow[d, dash, dashed] \\ \exists & \bullet \arrow[r, "a"] & \bullet \end{tikzcd} \qquad \begin{tikzcd} \forall d > 0 & \bullet \arrow[r, "\delta(d)"] \arrow[d, dash, dashed] & \bullet \arrow[d, dash, dashed] \\ \exists d' > 0 & \bullet \arrow[r, "\delta(d')"] & \bullet \end{tikzcd}$$

… and vice-versa (swap • and •).

Regions – From Infinite to Finite



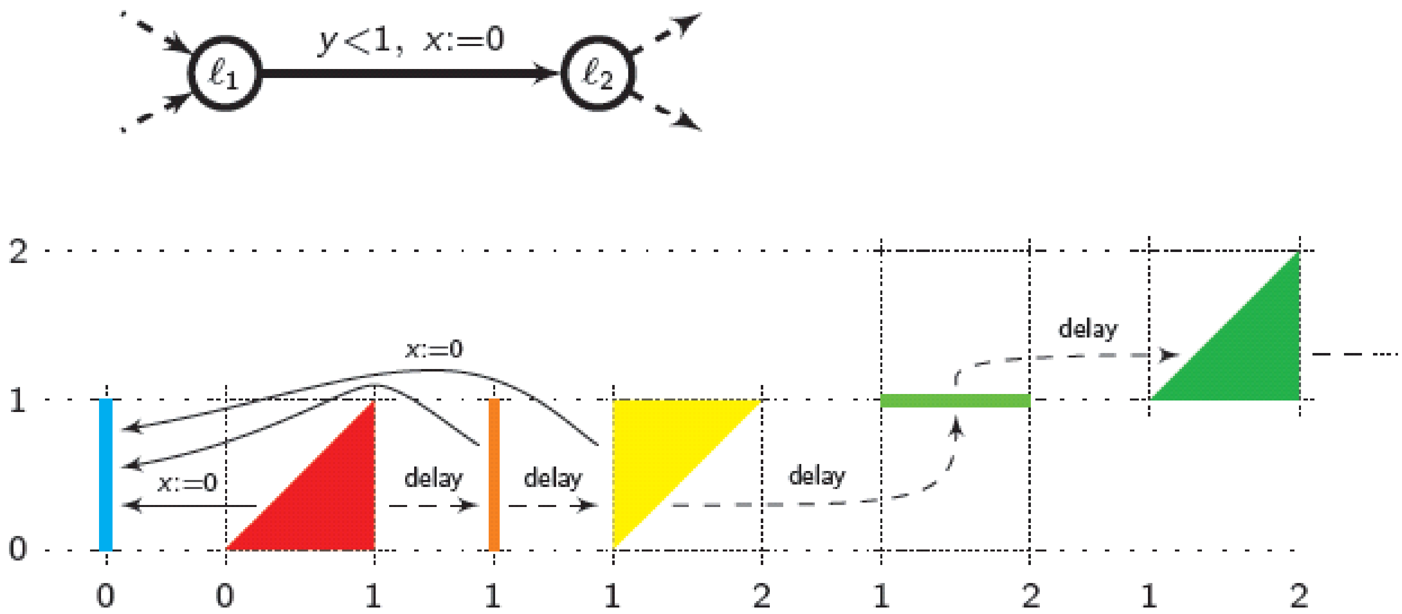


THM [AD90]
Reachability is decidable (and PSPACE-complete) for timed automata

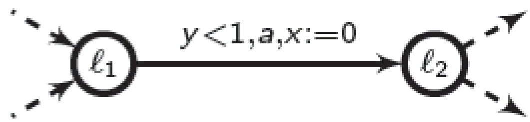
THM [CY90]
Time-optimal reachability is decidable (and PSPACE-complete) for timed automata

Region Graph

It "mimicks" the behaviours of the clocks.

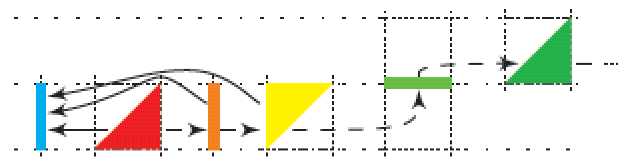


Region Automaton = Finite Bisimulation Quotiont



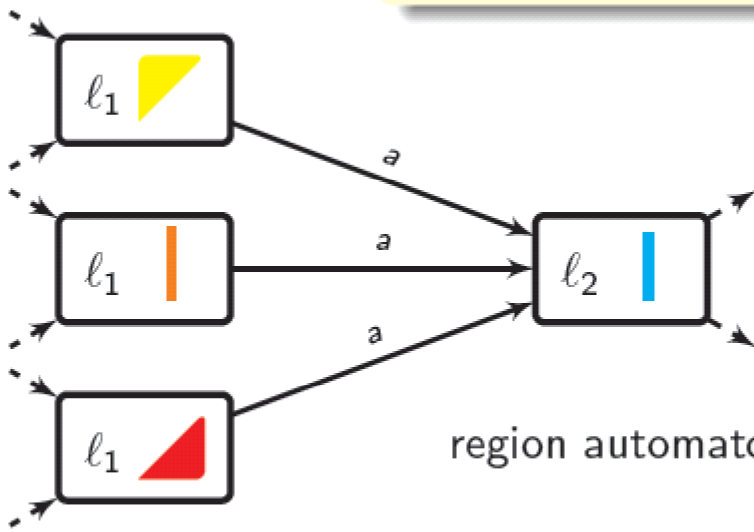
timed automaton

$\otimes$



region graph

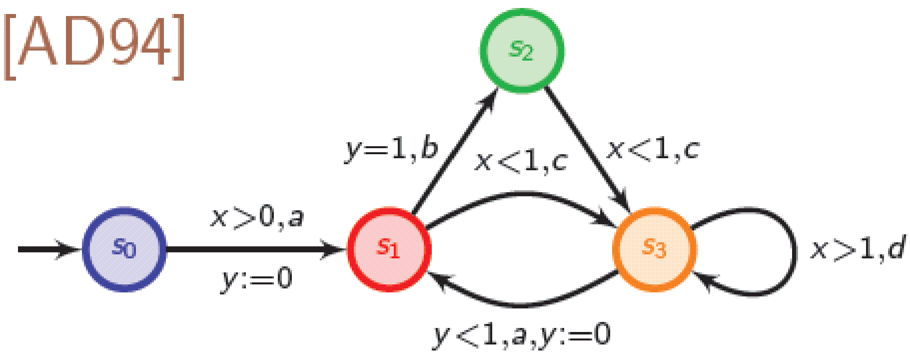
$\mathcal{L}(\text{reg. aut.}) = \text{UNTIME}(\mathcal{L}(\text{timed aut.}))$

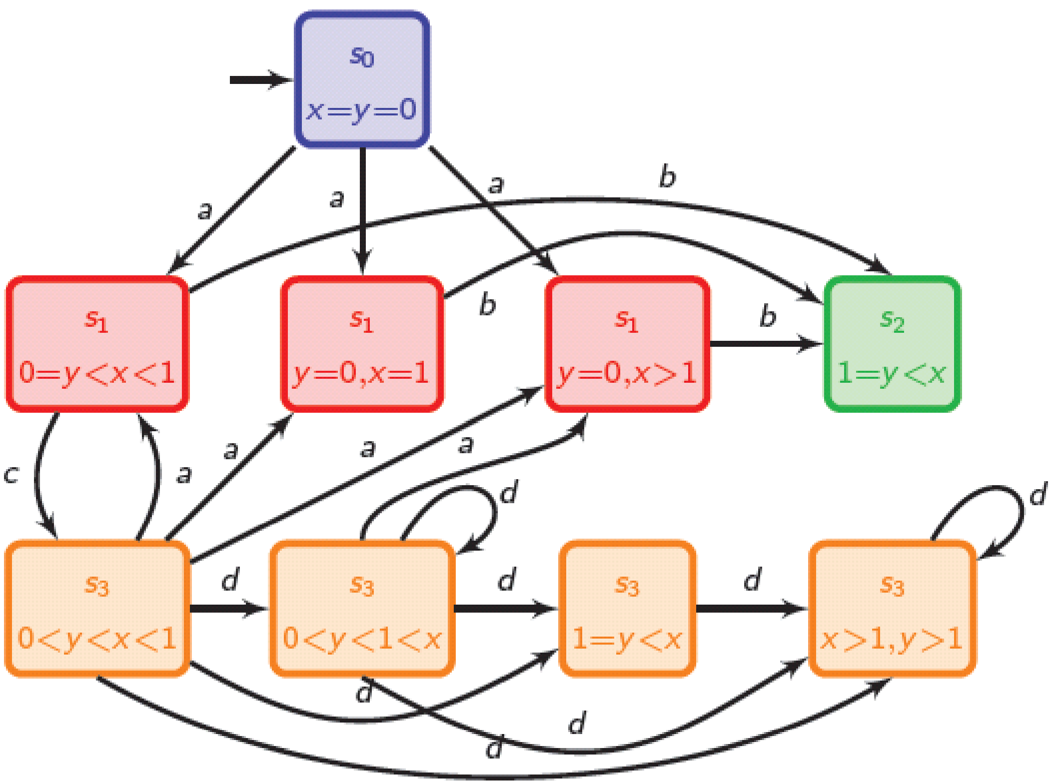


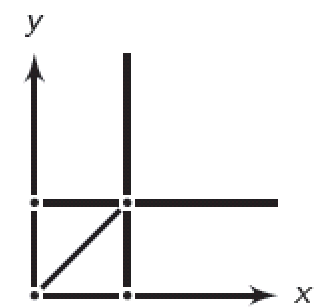
region automaton

An Example

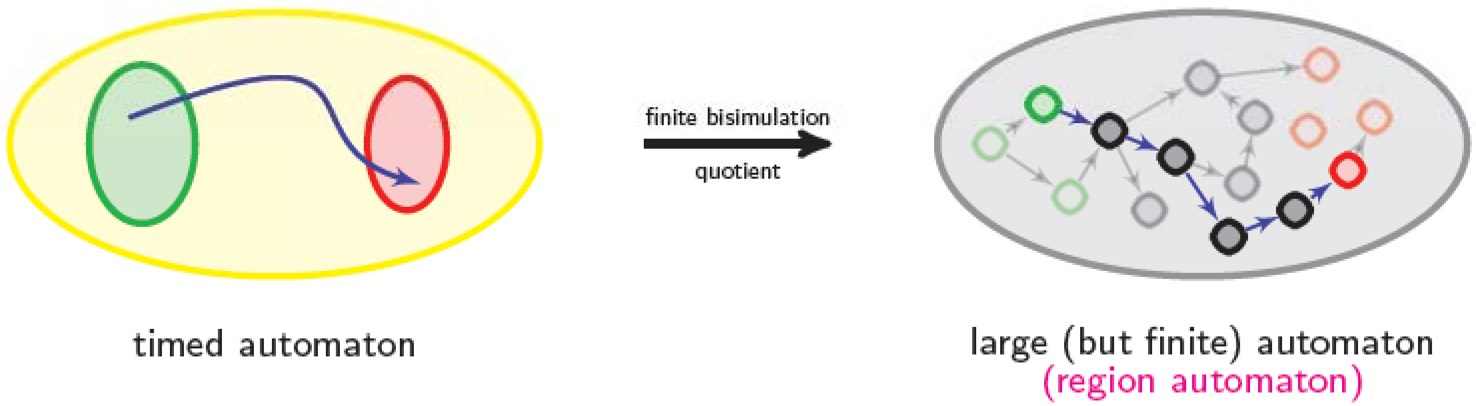
[AD94]







Region Automaton



LARGE: exponential in the number of clocks and in the constants (if encoded in binary). The number of regions is

$$\prod_{x \in X} (2M_x + 2) \cdot |X!| \cdot 2^{|X|}$$

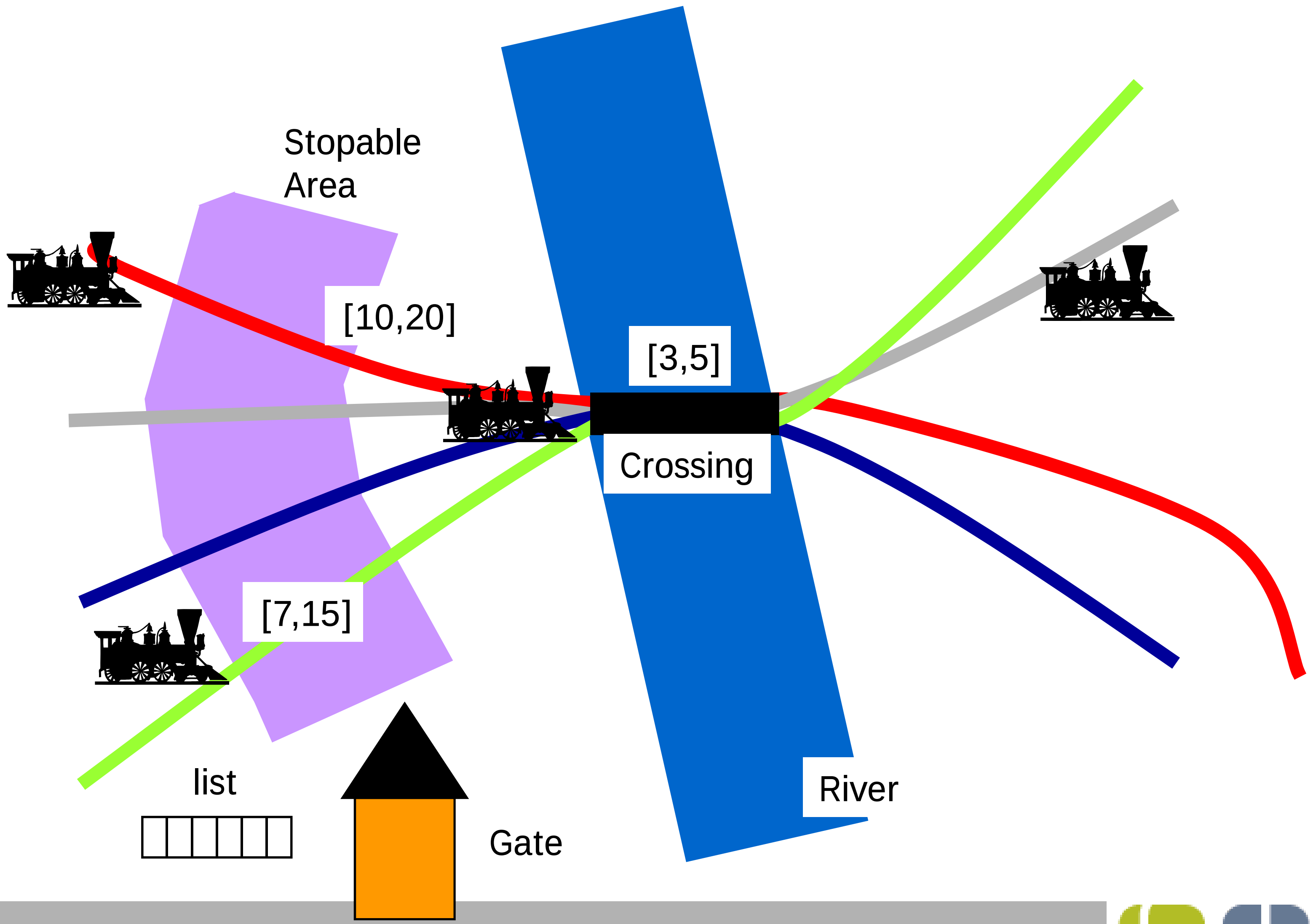
Fundamental Results

- Reachability ☺ PSPACE-c
- Model-checking
  - TCTL ☺ PSPACE-c ; MTL ☹ UNDECIDABLE ; MITL ☺ EXPSPACE-c
- Bisimulation, Simulation
  - Timed ☺ EXPTIME-c ; Untimed ☺
- Trace-inclusion
  - Timed ☹ UNDECIDABLE ; Untimed ☺ PSPACE-c

UPPAAL

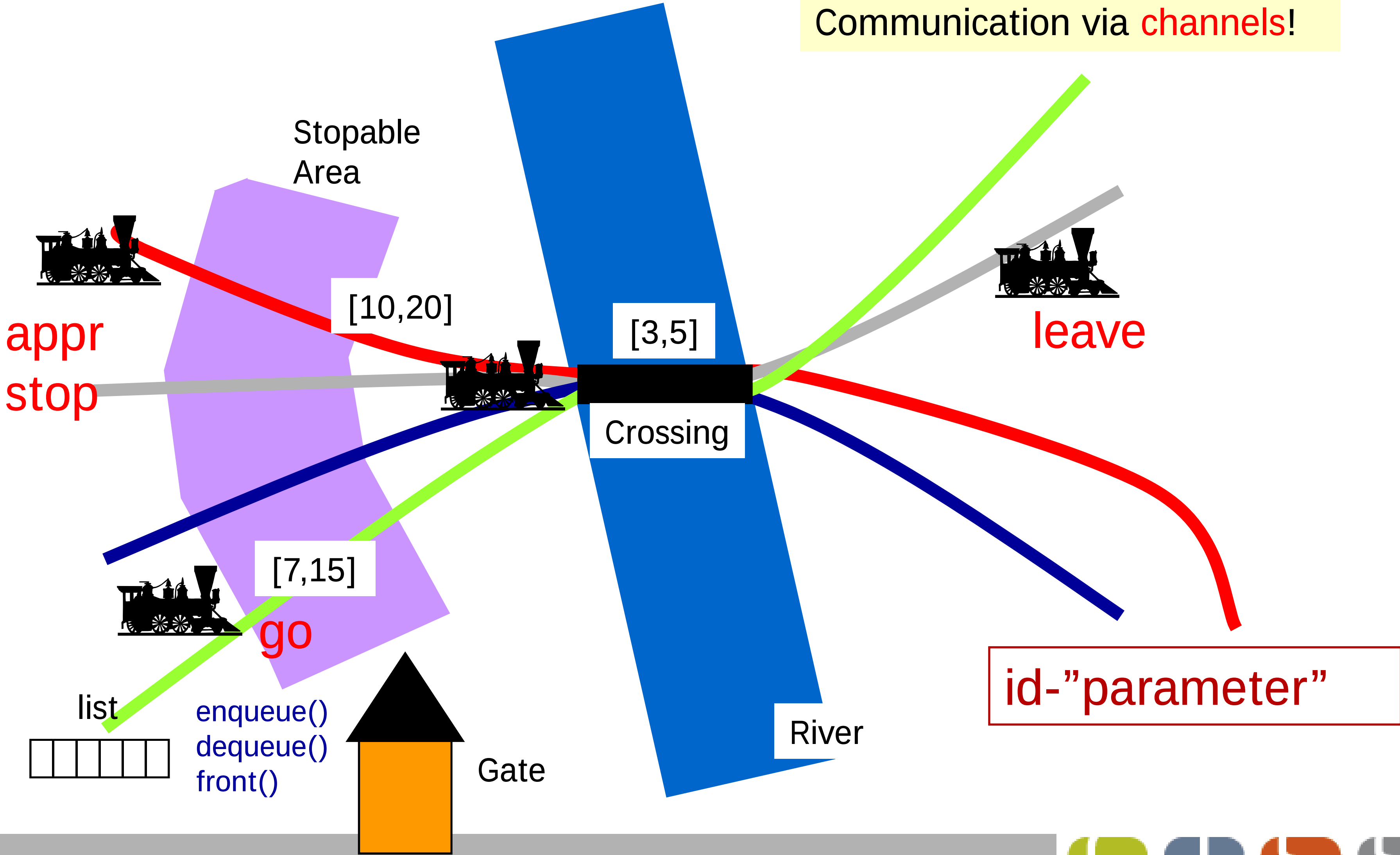
Modeling & Specification

Train Crossing

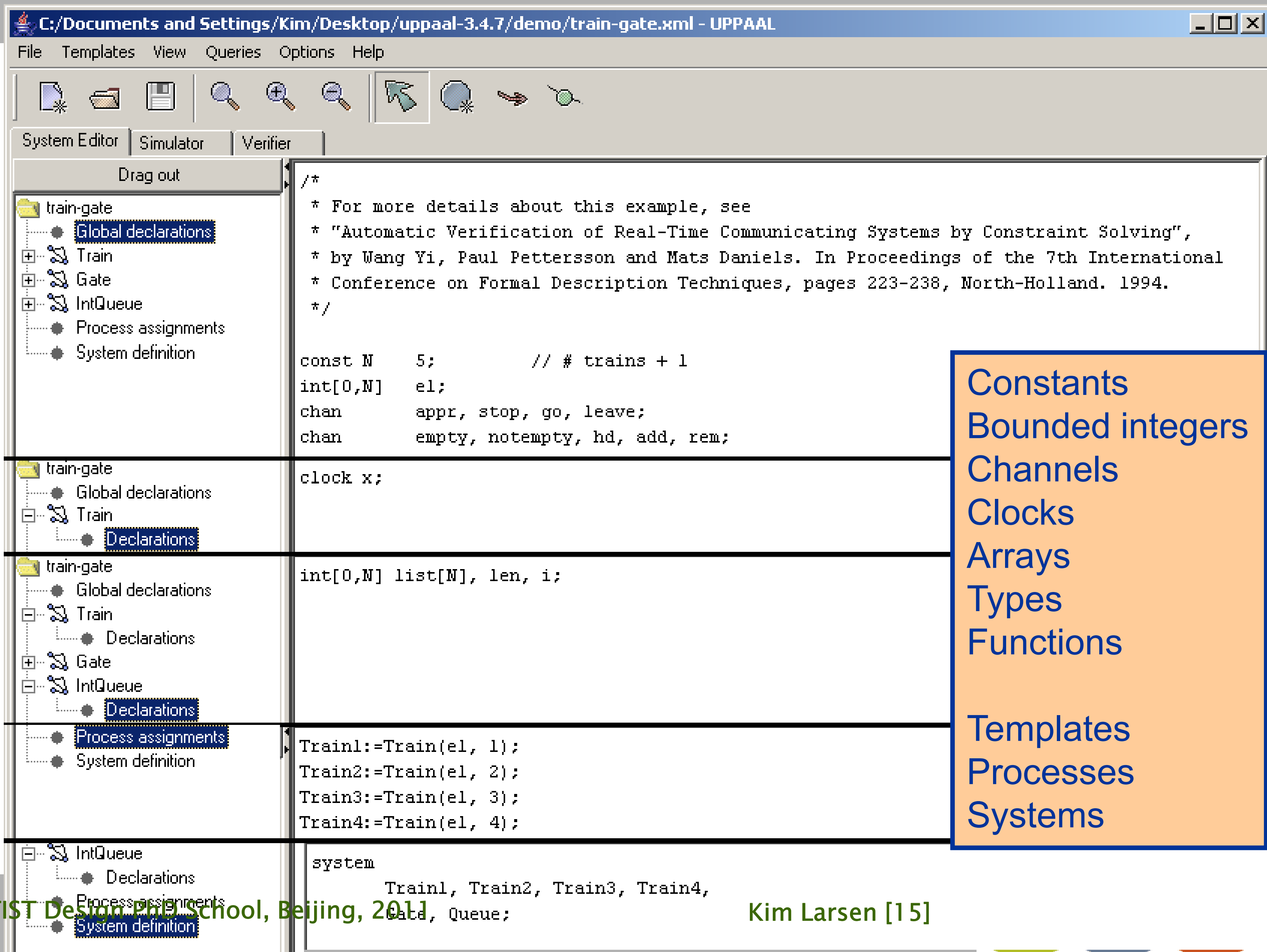


Train Crossing

Communication via channels!



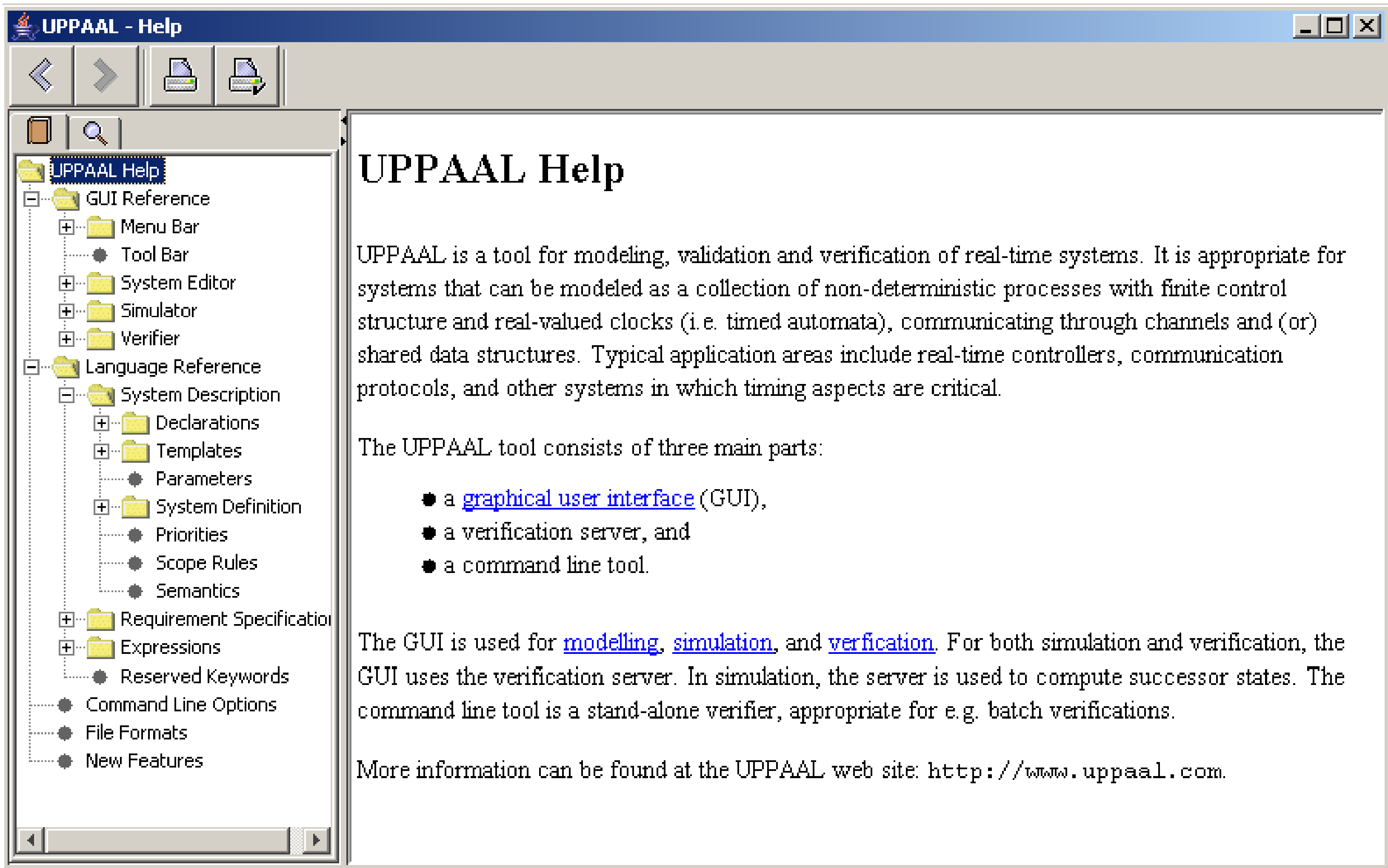
Declarations



Constants
Bounded integers
Channels
Clocks
Arrays
Types
Functions

Templates
Processes
Systems

UPPAAL Help



UPPAAL Help

UPPAAL is a tool for modeling, validation and verification of real-time systems. It is appropriate for systems that can be modeled as a collection of non-deterministic processes with finite control structure and real-valued clocks (i.e. timed automata), communicating through channels and (or) shared data structures. Typical application areas include real-time controllers, communication protocols, and other systems in which timing aspects are critical.

The UPPAAL tool consists of three main parts:

- a graphical user interface (GUI),
- a verification server, and
- a command line tool.

The GUI is used for modelling, simulation, and verfication. For both simulation and verification, the GUI uses the verification server. In simulation, the server is used to compute successor states. The command line tool is a stand-alone verifier, appropriate for e.g. batch verifications.

More information can be found at the UPPAAL web site: `http://www.uppaal.com`.

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

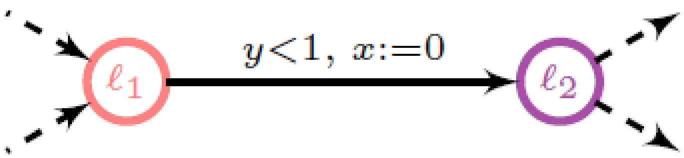
The expressions *P* and *Q* must be type safe, side effect free, and evaluate to a boolean.

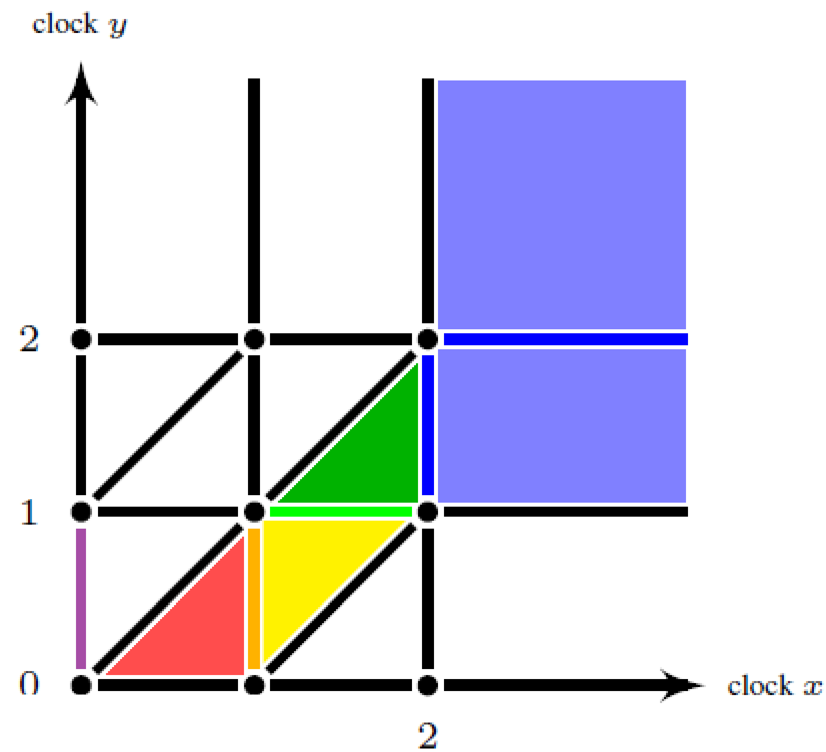
Only references to integer variables, constants, clocks, are allowed (and arrays of these).

Symbolic Verification

The UPPAAL Verification Engine

Regions – From Infinite to Finite

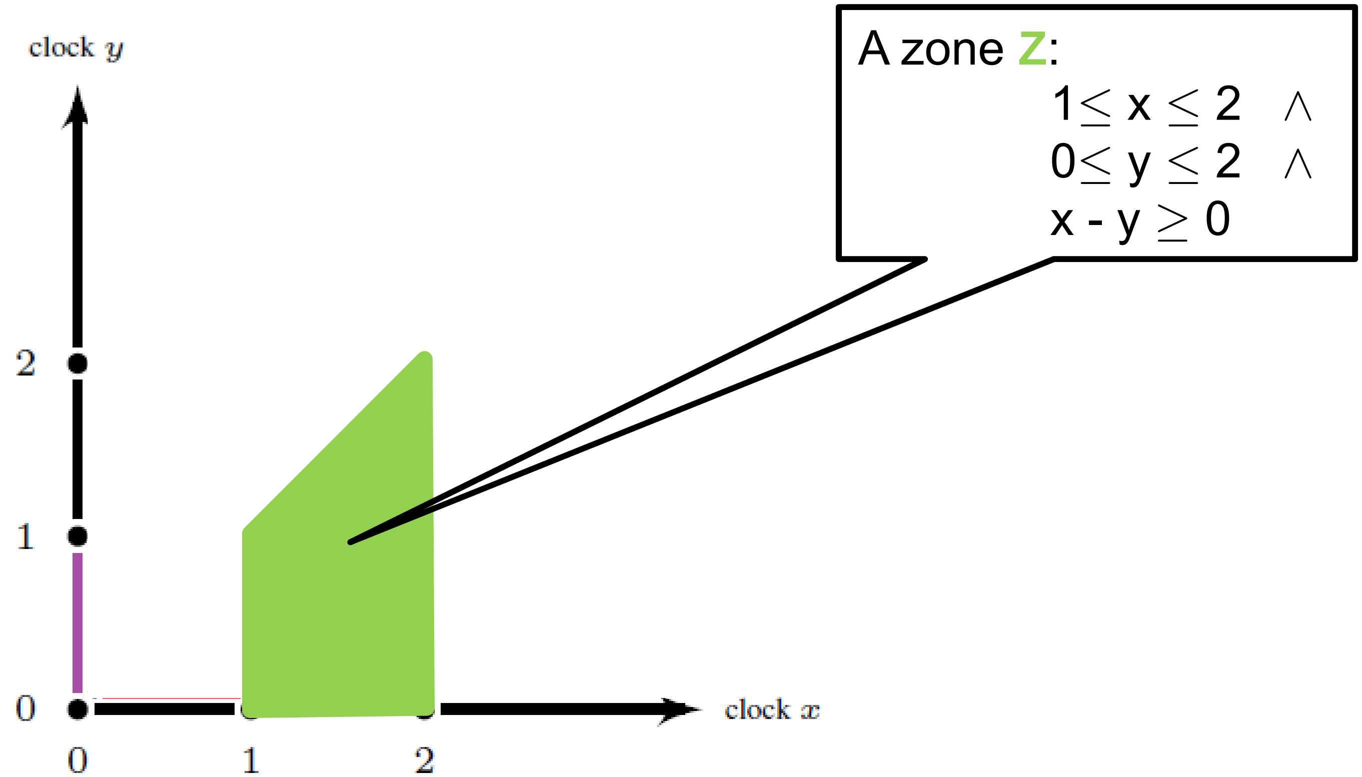




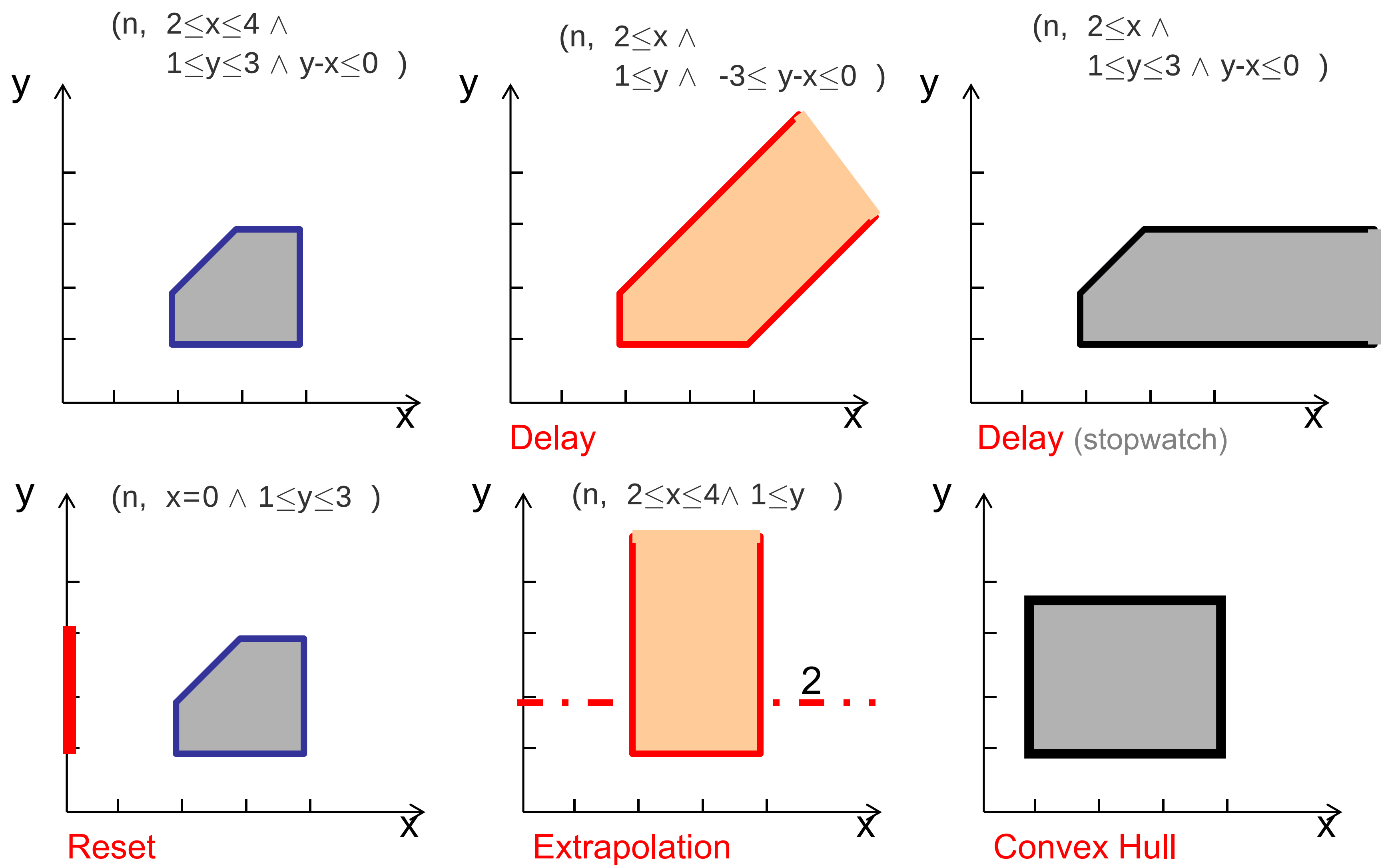
Theorem

The number of regions is $n! \cdot 2^n \cdot \prod_{x \in C}(2c_x + 2)$.

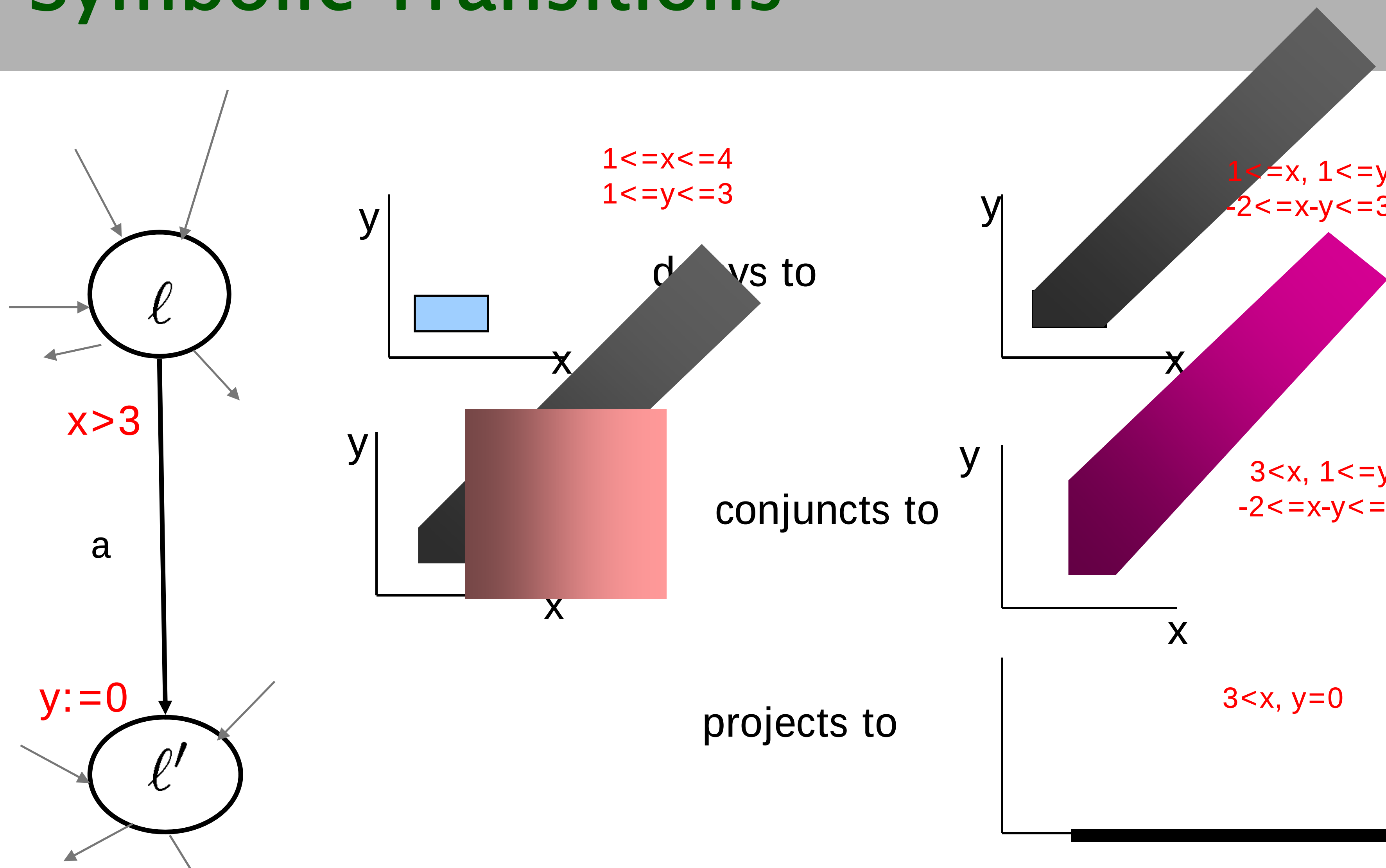
Zones – From Finite to Efficiency



Zones – Operations



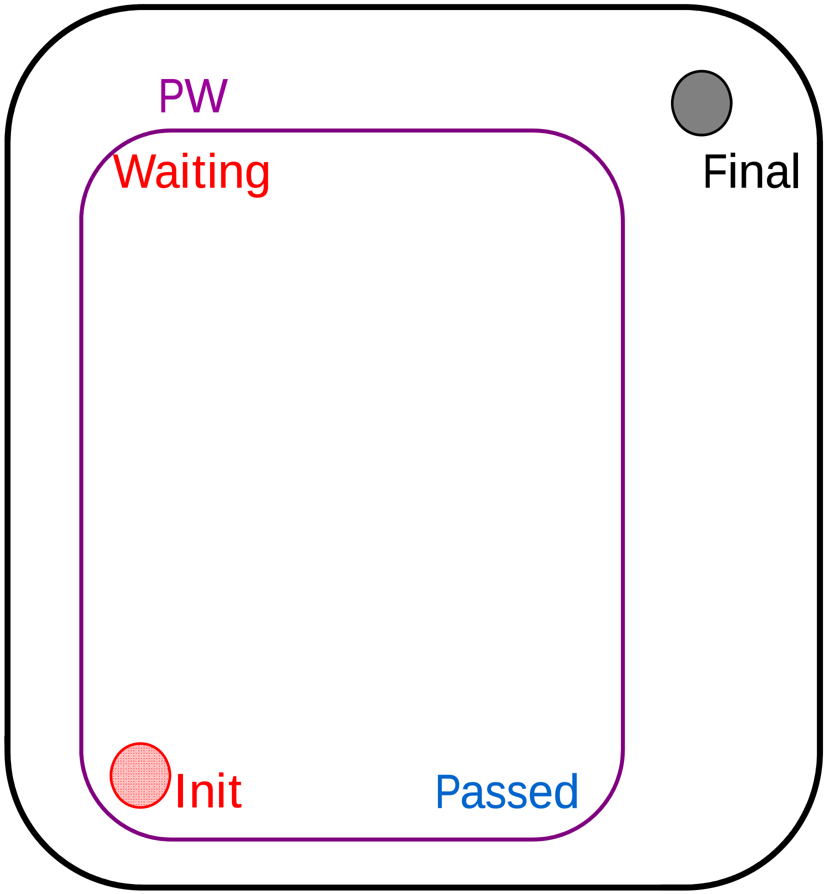
Symbolic Transitions



$$(\ell, 1 \le x \le 4, 1 \le y \le 3) \Rightarrow_a (\ell', 3 < x, y = 0)$$

Forward Reachability

Init -> Final ?



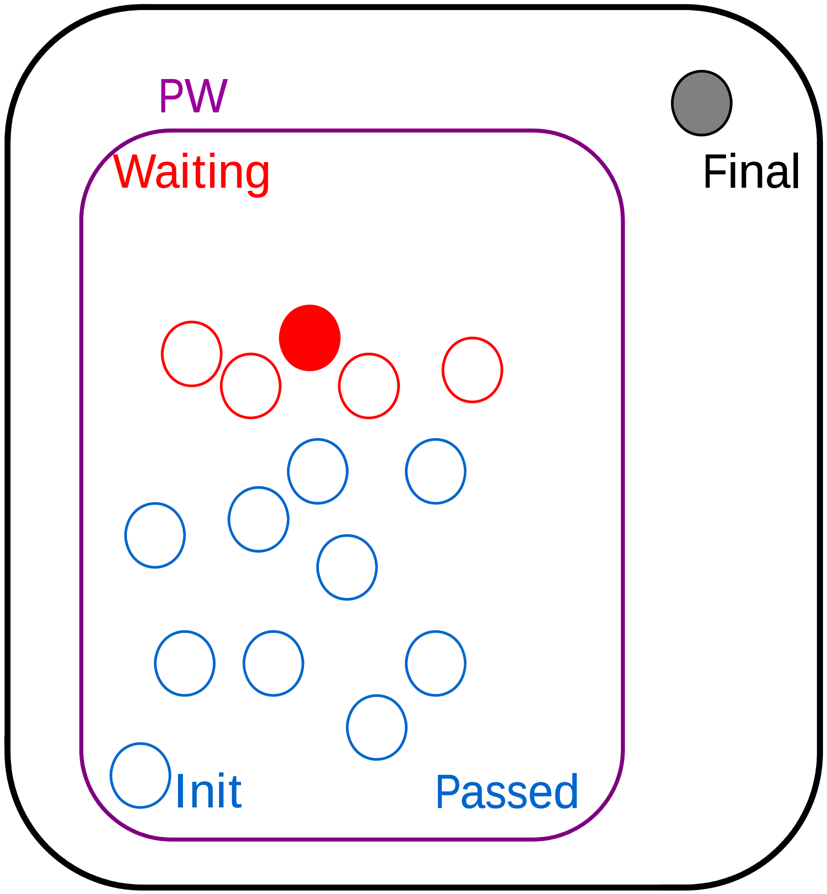
INITIAL Passed := Ø;
Waiting := $\{(n_0,Z_0)\}$

REPEAT
pick (n,Z) in Waiting
if (n,Z) = Final return true
for all $(n,Z) \rightarrow (n',Z')$:
if for some (n',Z'') $Z' \subseteq Z''$ continue
else add (n',Z') to Waiting
move (n,Z) to Passed

UNTIL Waiting = Ø
return false

Forward Reachability

Init -> Final ?



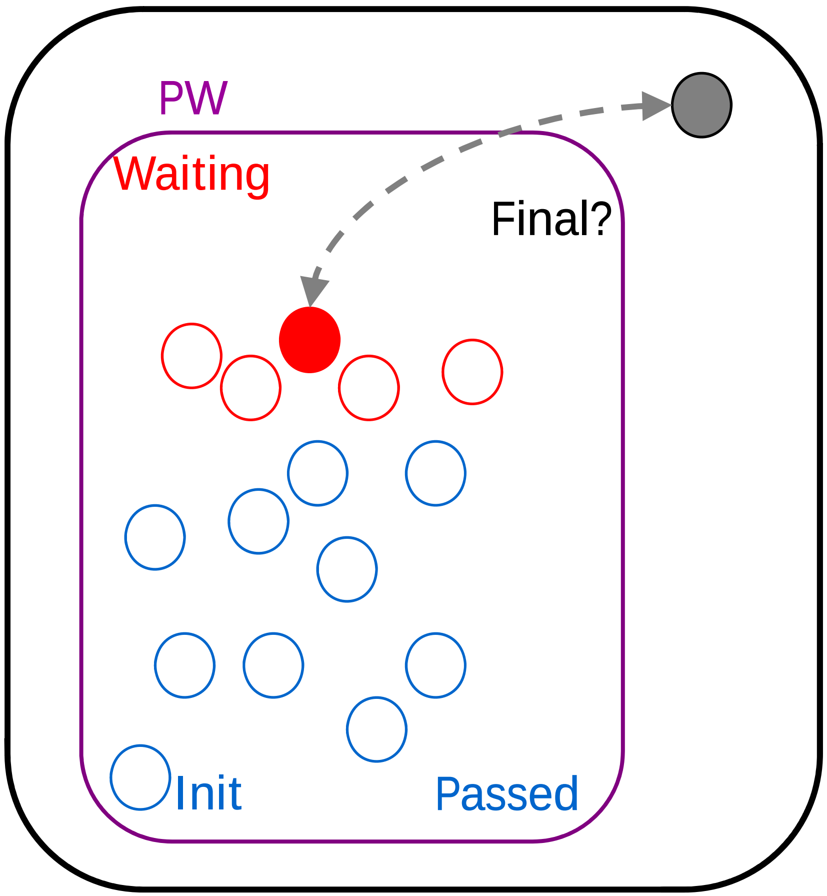
INITIAL Passed := Ø;
Waiting := {(n_0,Z_0)}

REPEAT
pick (n,Z) in Waiting
if (n,Z) = Final return true
for all (n,Z)→(n',Z'):
if for some (n',Z'') Z'⊆ Z'' continue
else add (n',Z') to Waiting
move (n,Z) to Passed

UNTIL Waiting = Ø
return false

Forward Reachability

Init -> Final ?



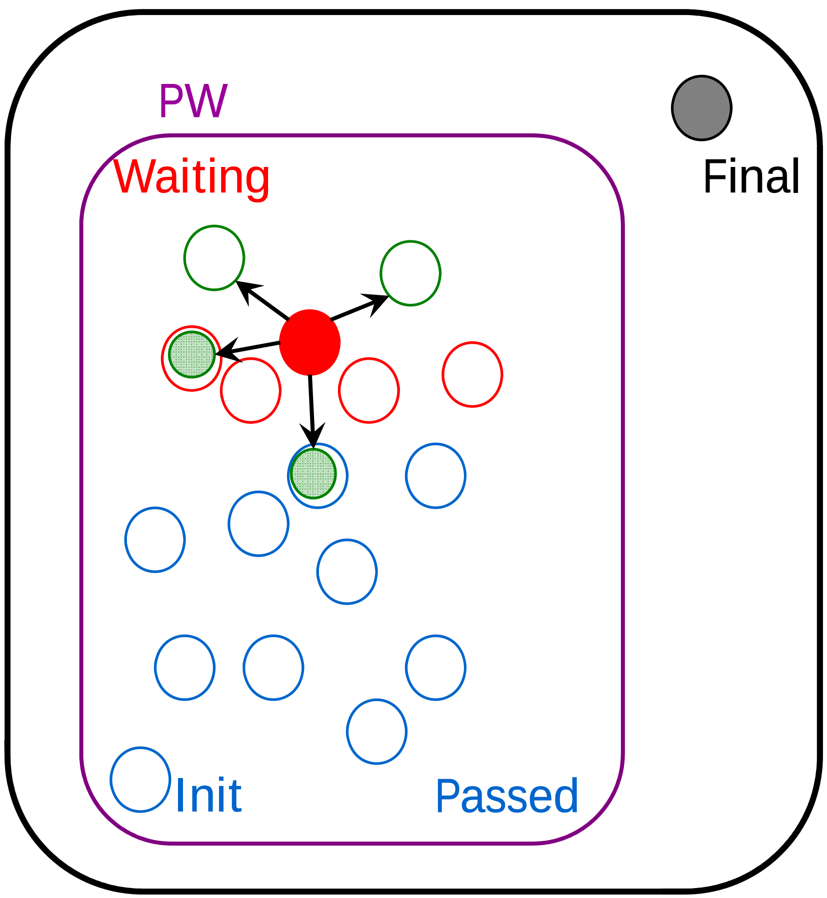
INITIAL Passed := Ø;
Waiting := {(n_0,Z_0)}

REPEAT
- pick (n,Z) in Waiting
- if (n,Z) = Final return true
- for all (n,Z)→(n',Z'):
  - if for some (n',Z'') Z'⊆ Z'' continue
  - else add (n',Z') to Waiting
  - move (n,Z) to Passed

UNTIL Waiting = Ø
return false

Forward Reachability

Init -> Final ?



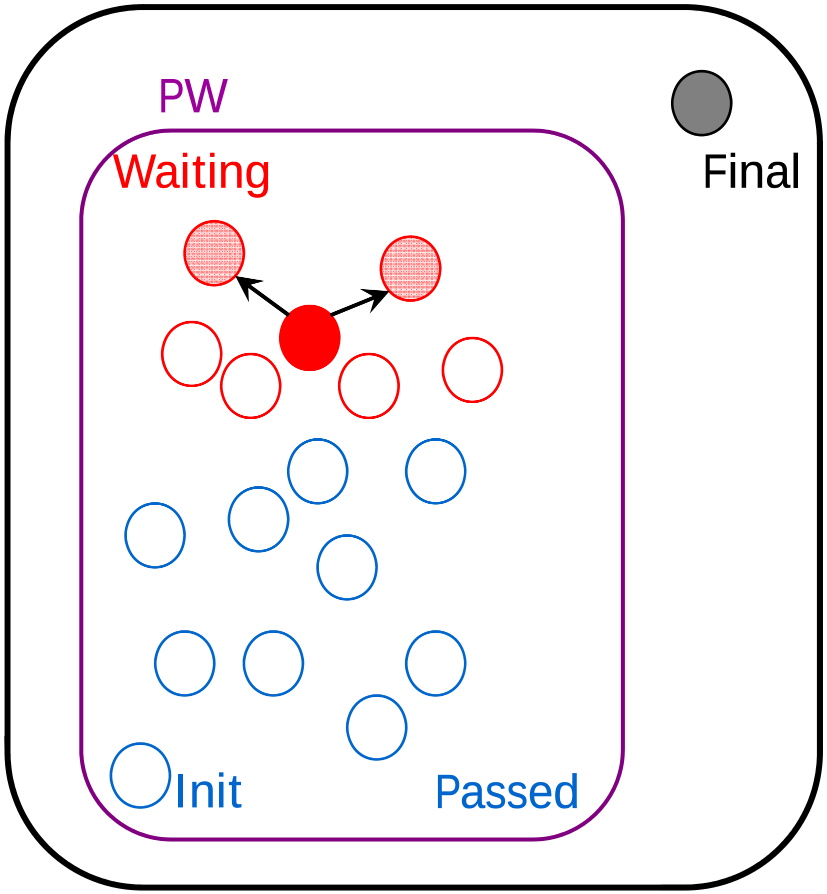
INITIAL Passed := Ø;
Waiting := {(n_0,Z_0)}

REPEAT
pick (n,Z) in Waiting
if (n,Z) = Final return true
for all (n,Z)→(n',Z'):
if for some (n',Z'') Z' ⊆ Z'' continue
else add (n',Z') to Waiting
move (n,Z) to Passed

UNTIL Waiting = Ø
return false

Forward Reachability

Init -> Final ?



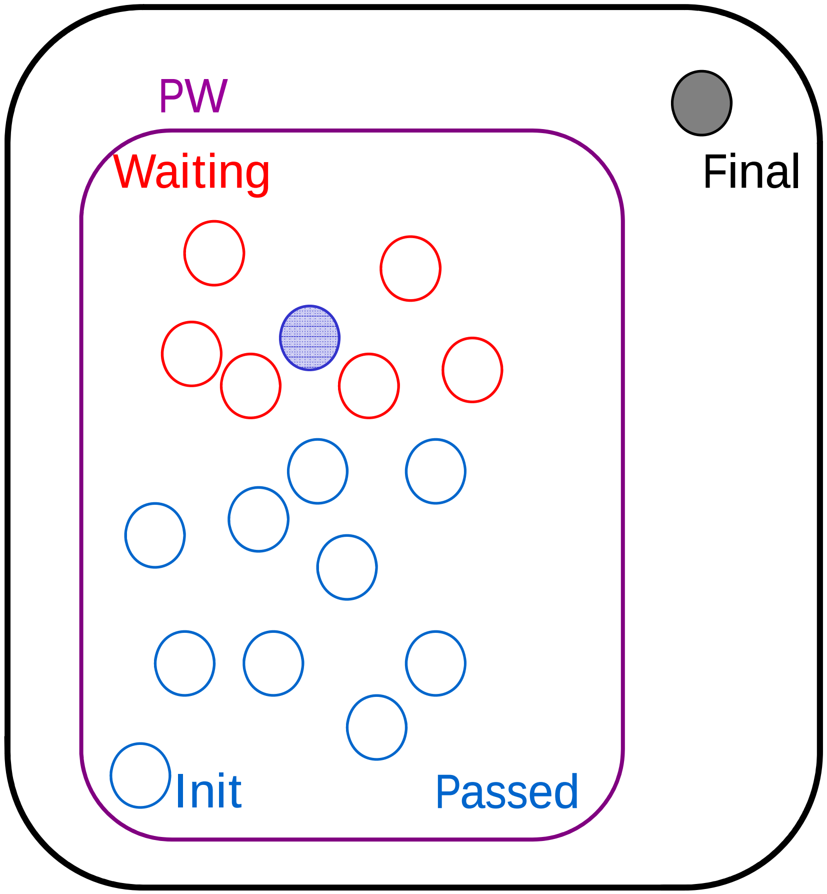
INITIAL Passed := Ø;
Waiting := {(n_0,Z_0)}

REPEAT
pick (n,Z) in Waiting
if (n,Z) = Final return true
for all (n,Z)→(n',Z'):
if for some (n',Z'') Z'⊆ Z'' continue
else add (n',Z') to Waiting
move (n,Z) to Passed

UNTIL Waiting = Ø
return false

Forward Reachability

Init -> Final ?



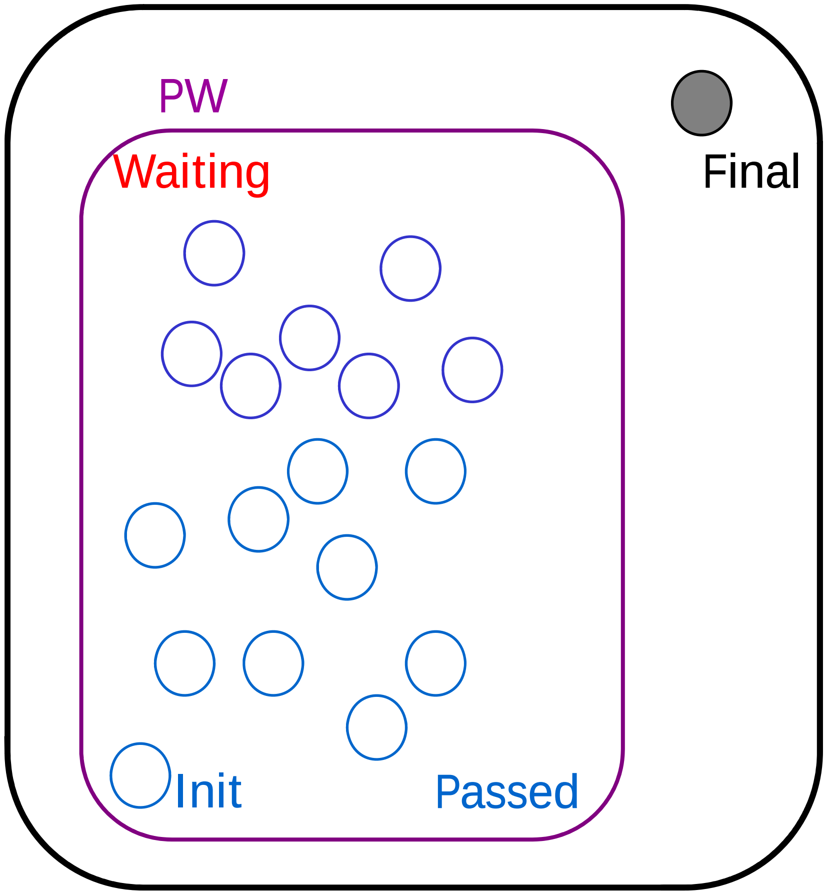
INITIAL Passed := Ø;
Waiting := $\{(n_0, Z_0)\}$

REPEAT
pick (n,Z) in Waiting
if (n,Z) = Final return true
for all $(n,Z) \rightarrow (n',Z')$:
if for some (n',Z'') $Z' \subseteq Z''$ continue
else add (n',Z') to Waiting
move (n,Z) to Passed

UNTIL Waiting = Ø
return false

Forward Reachability

Init -> Final ?

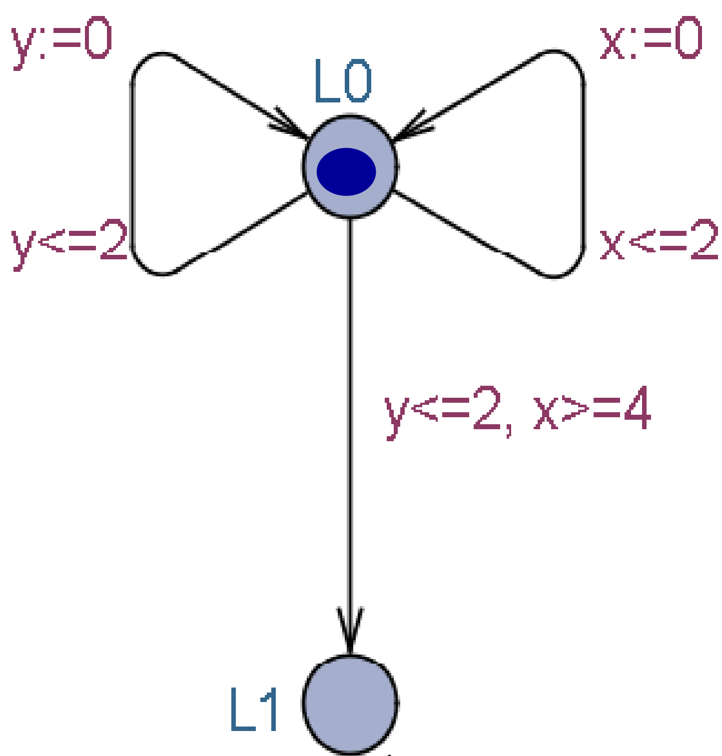


INITIAL Passed := Ø;
Waiting := $\{(n_0,Z_0)\}$

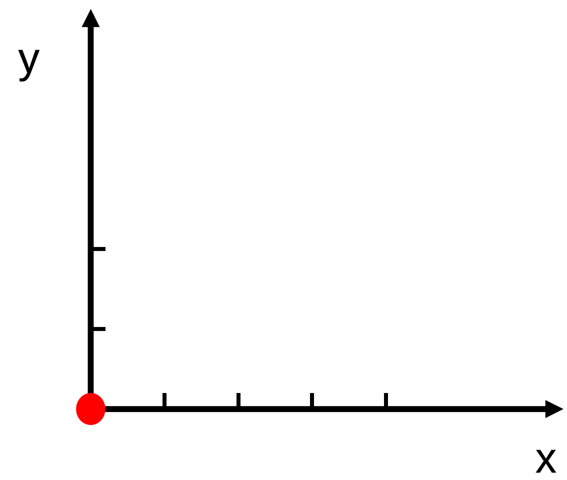
REPEAT
pick (n,Z) in Waiting
if (n,Z) = Final return true
for all $(n,Z) \rightarrow (n',Z')$:
if for some (n',Z'') $Z' \subseteq Z''$ continue
else add (n',Z') to Waiting
move (n,Z) to Passed

UNTIL Waiting = Ø
<u>return false</u>

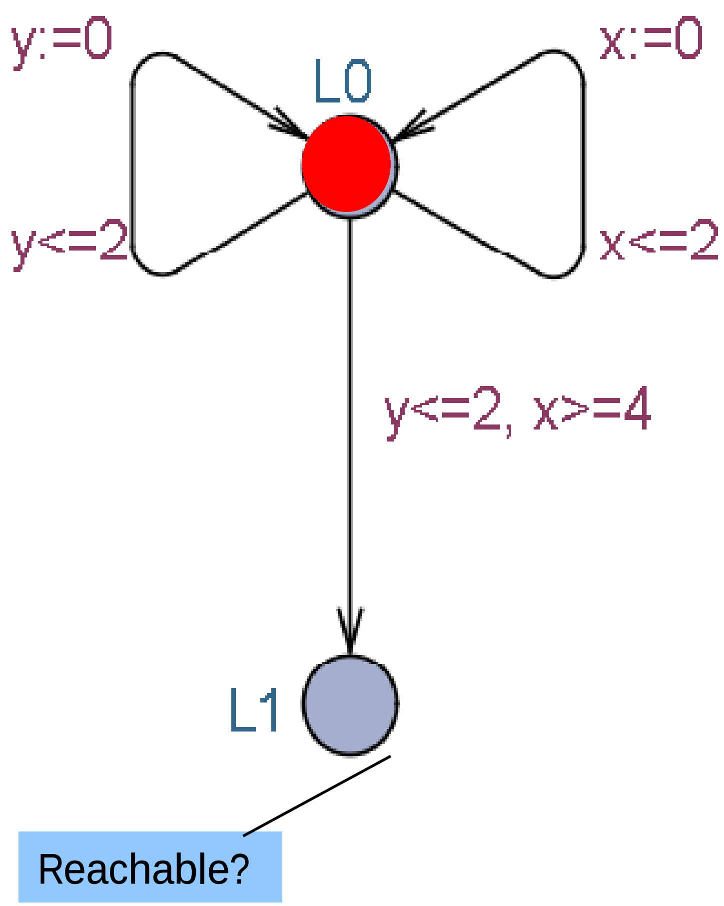
Symbolic Exploration

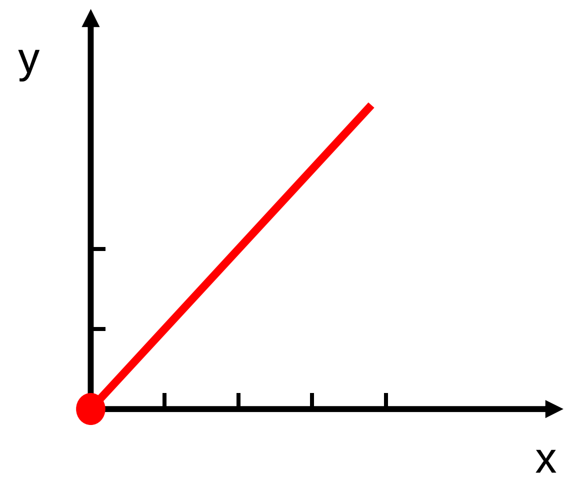


Reachable?



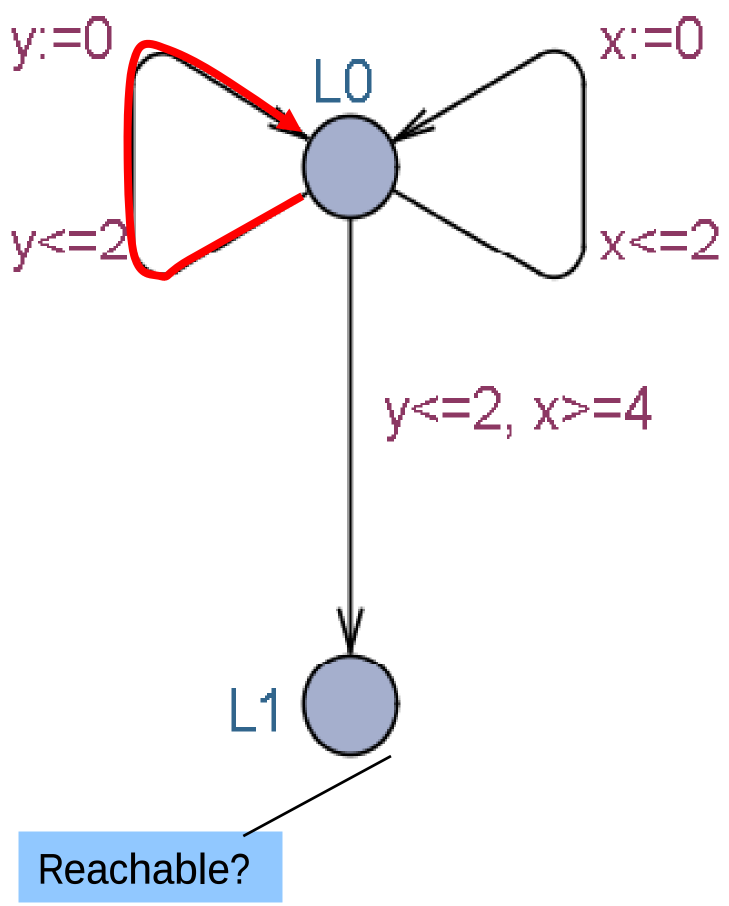
Symbolic Exploration

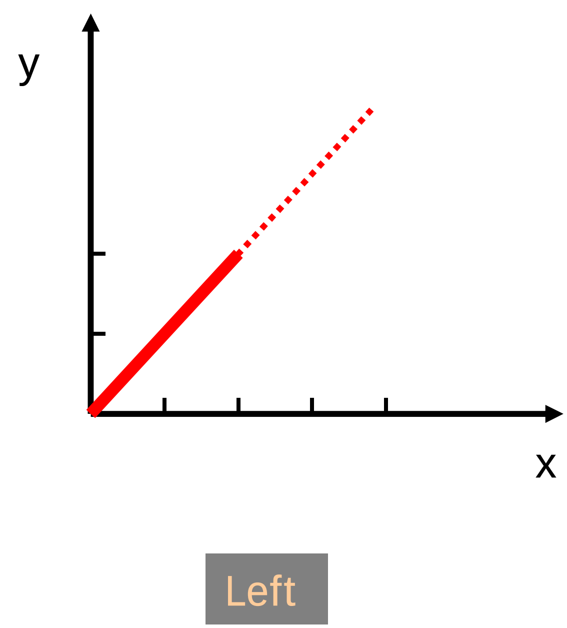




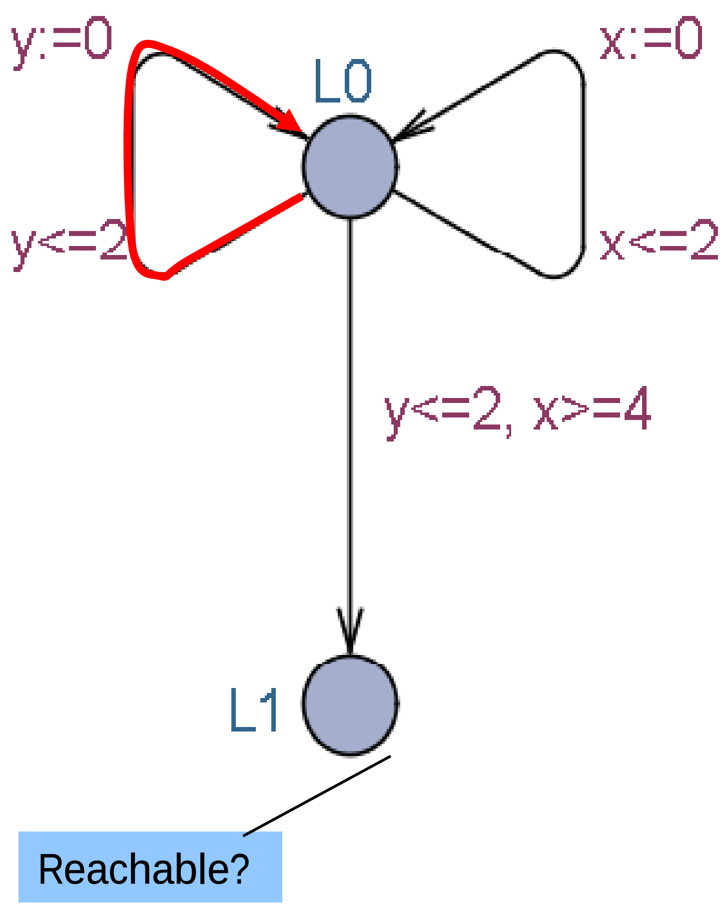
Delay

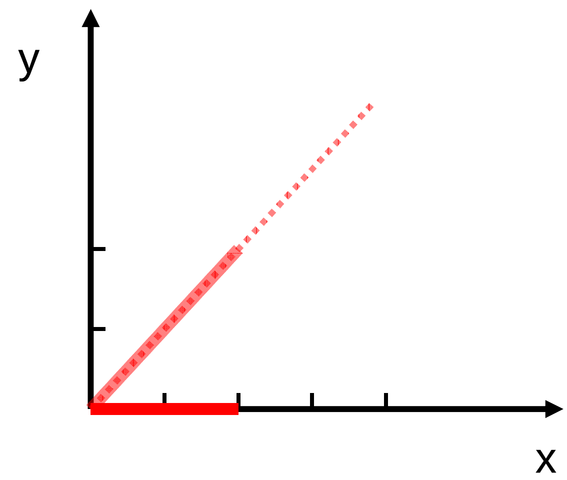
Symbolic Exploration





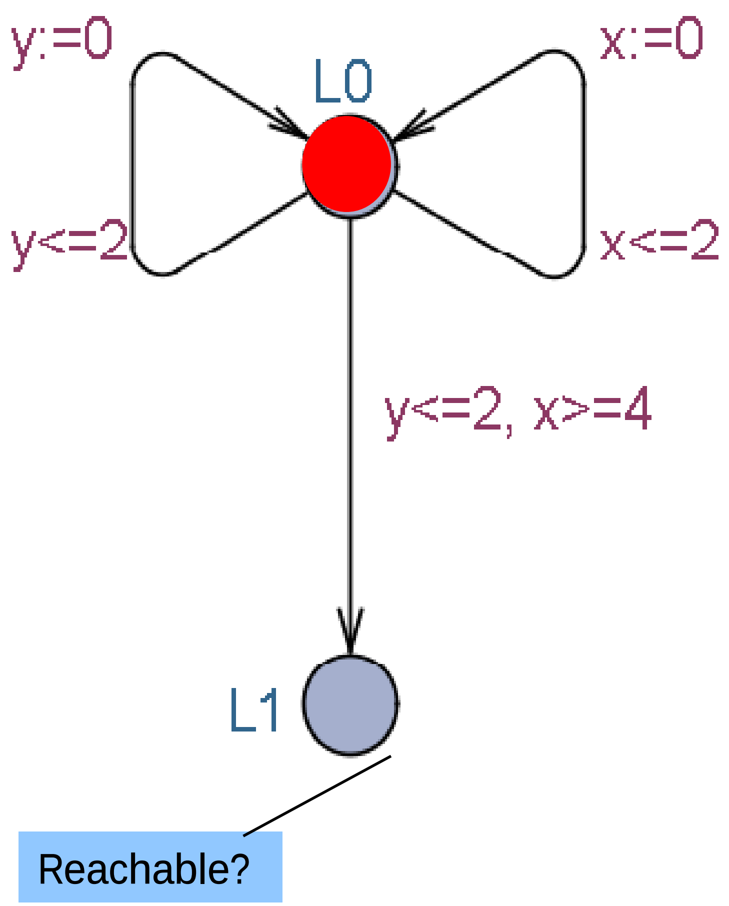
Symbolic Exploration

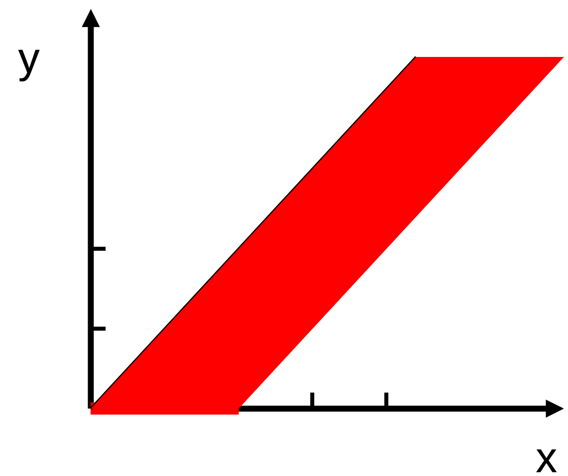




Left

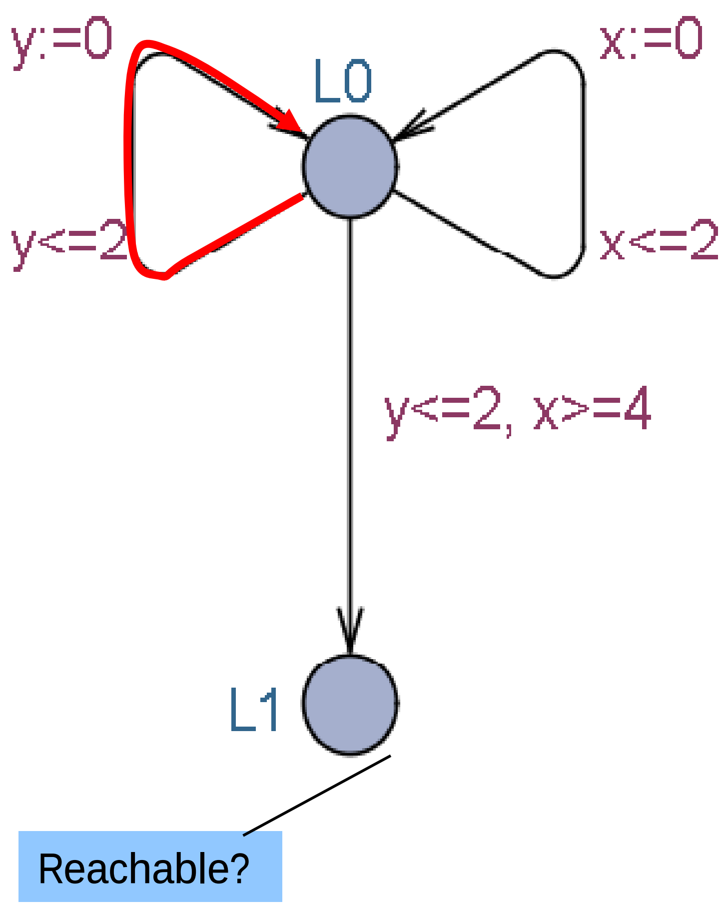
Symbolic Exploration

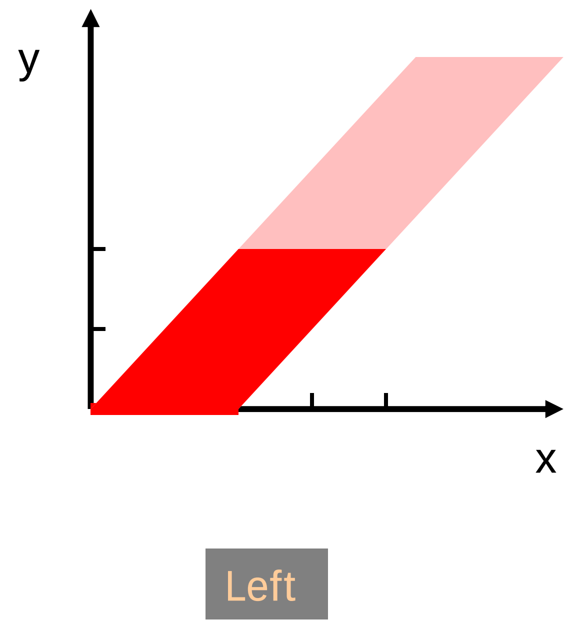




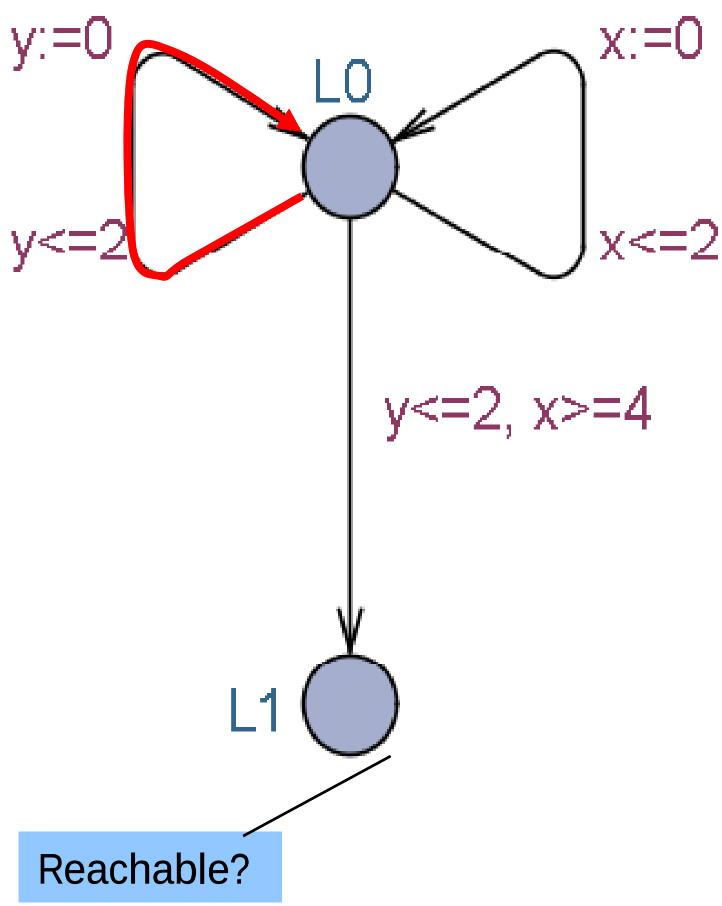
Delay

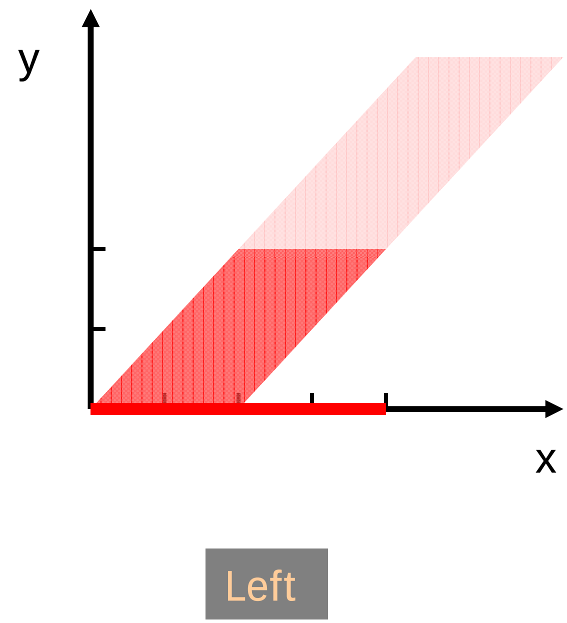
Symbolic Exploration



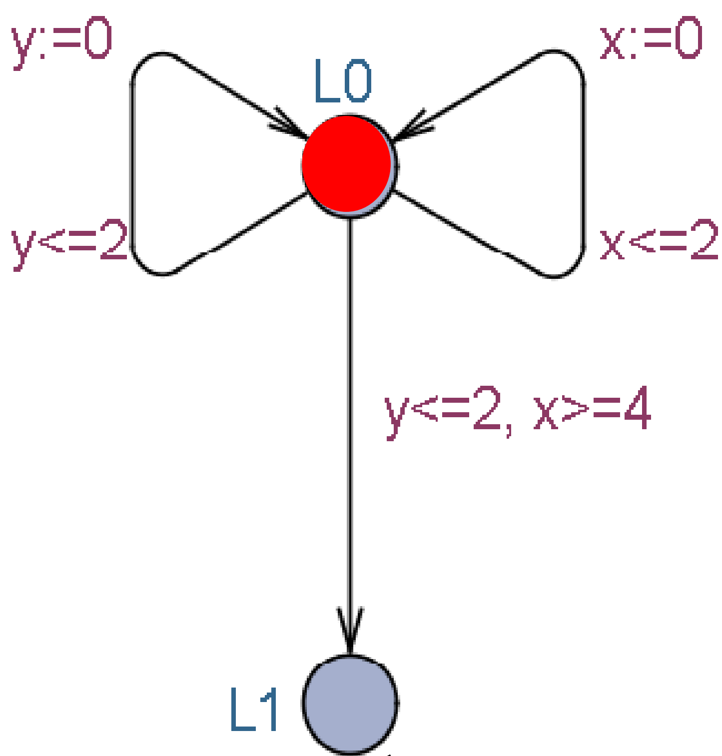


Symbolic Exploration

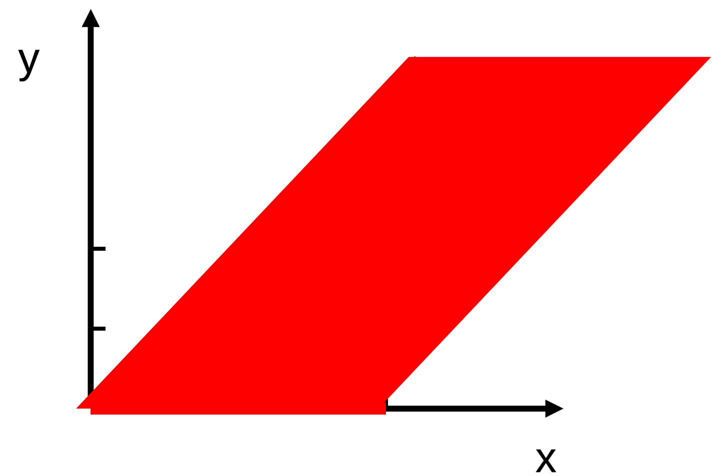




Symbolic Exploration

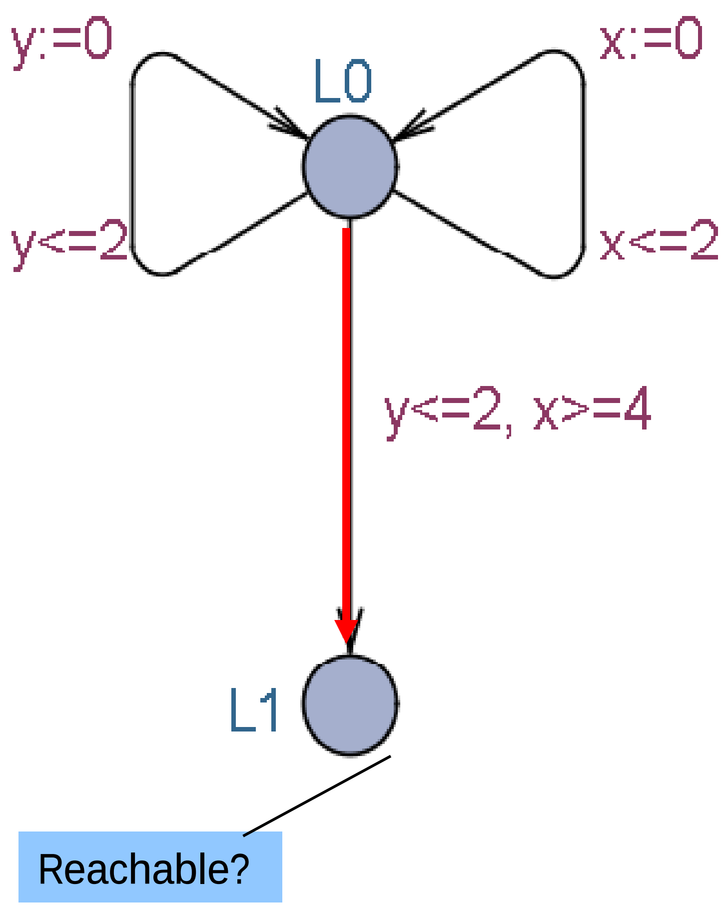


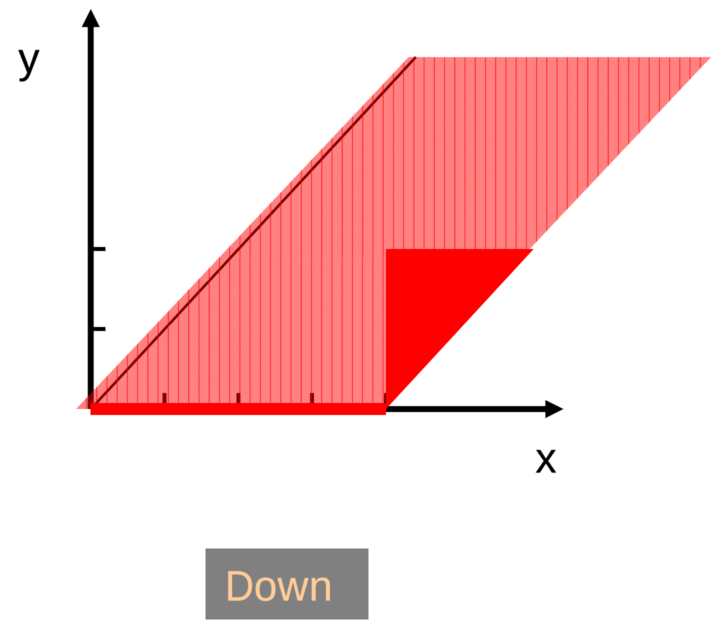
Reachable?



Delay

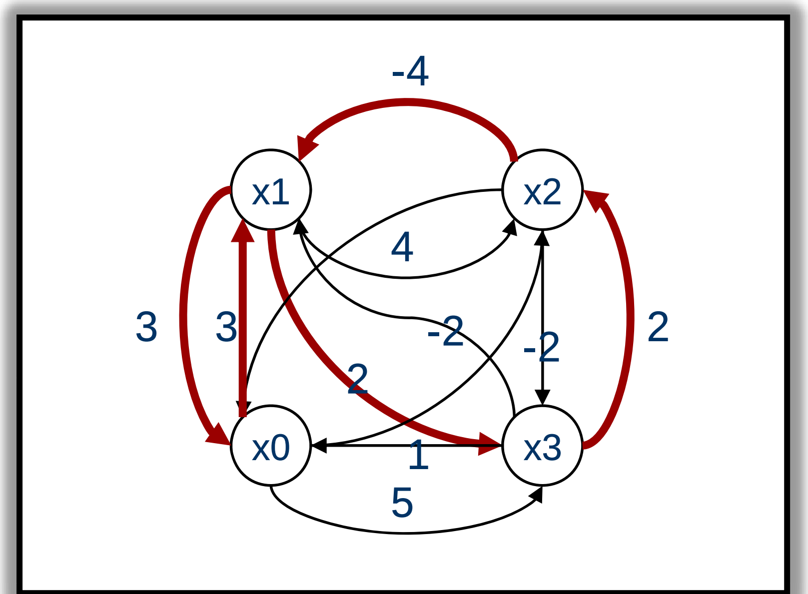
Symbolic Exploration

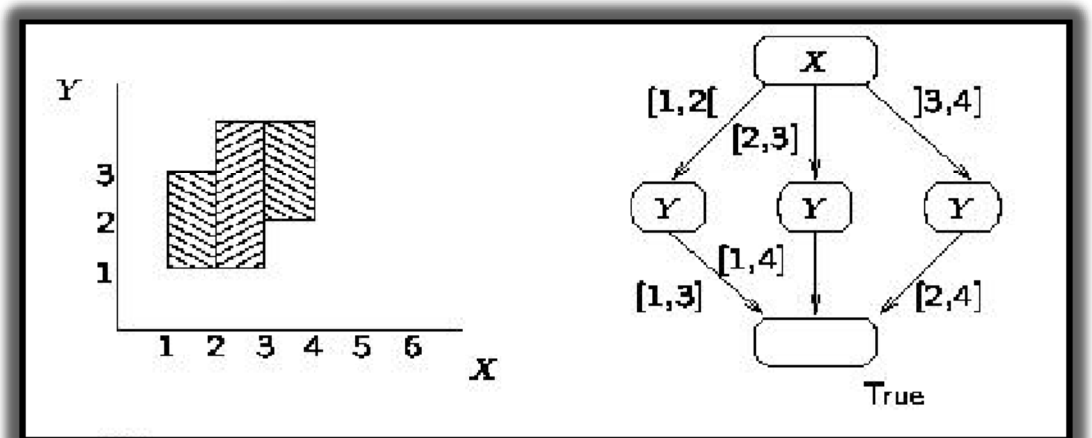




Datastructures for Zones

- Difference Bounded Matrices (DBMs)
- Minimal Constraint Form
  [RTSS97]
- Clock Difference Diagrams
  [CAV99]





Inclusion Checking (DBMs)

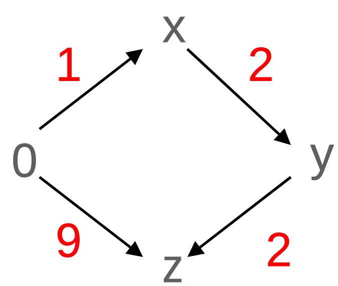
Bellman 1958, Dill 1989

Inclusion

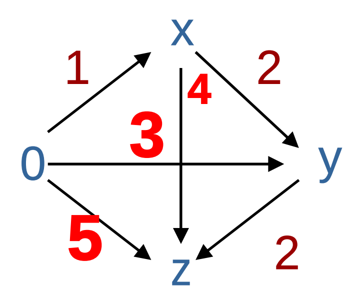
D1

x<=1
y-x<=2
z-y<=2
z<=9

Graph



Shortest Path Closure

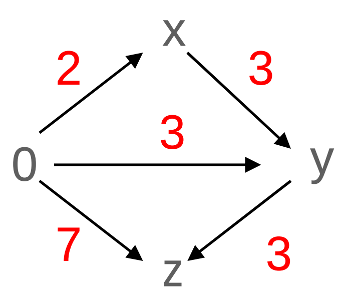


? $\subseteq$ **?**

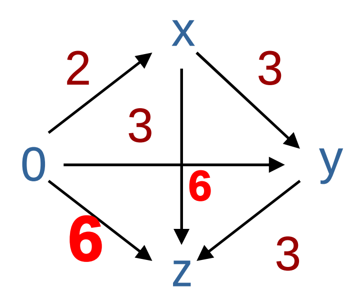
D2

x<=2
y-x<=3
y<=3
z-y<=3
z<=7

Graph

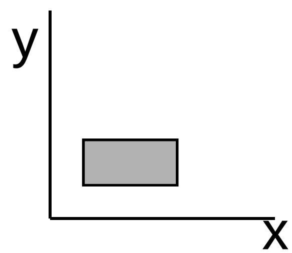


Shortest Path Closure

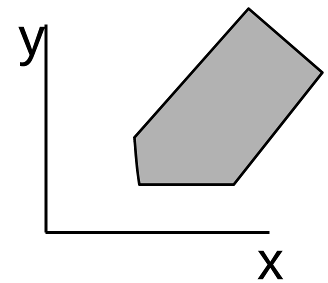


Future (DBMs)

D

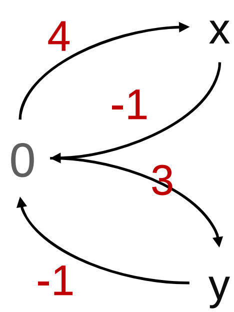


1<= x <=4
1<= y <=3

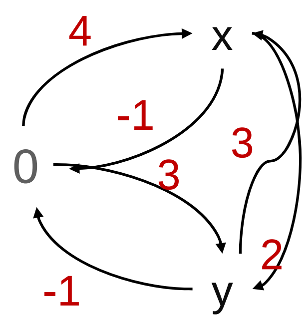


Future D

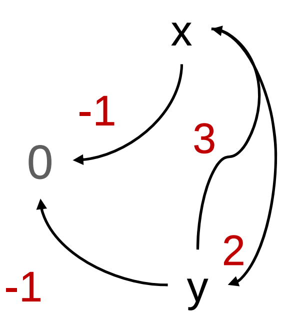
1<=x, 1<=y
-2<=x-y<=3



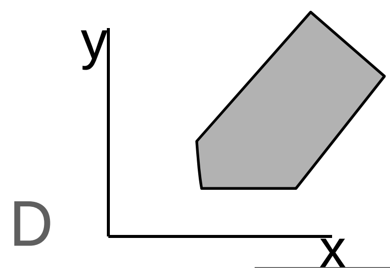
Shortest Path Closure



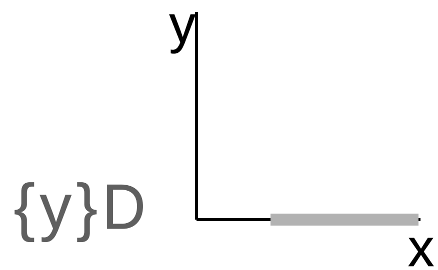
Remove upper bounds on clocks



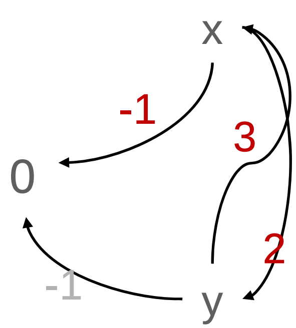
Reset (DBMs)



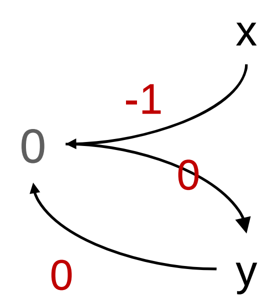
1<=x, 1<=y
-2<=x-y<=3



y=0, 1<=x



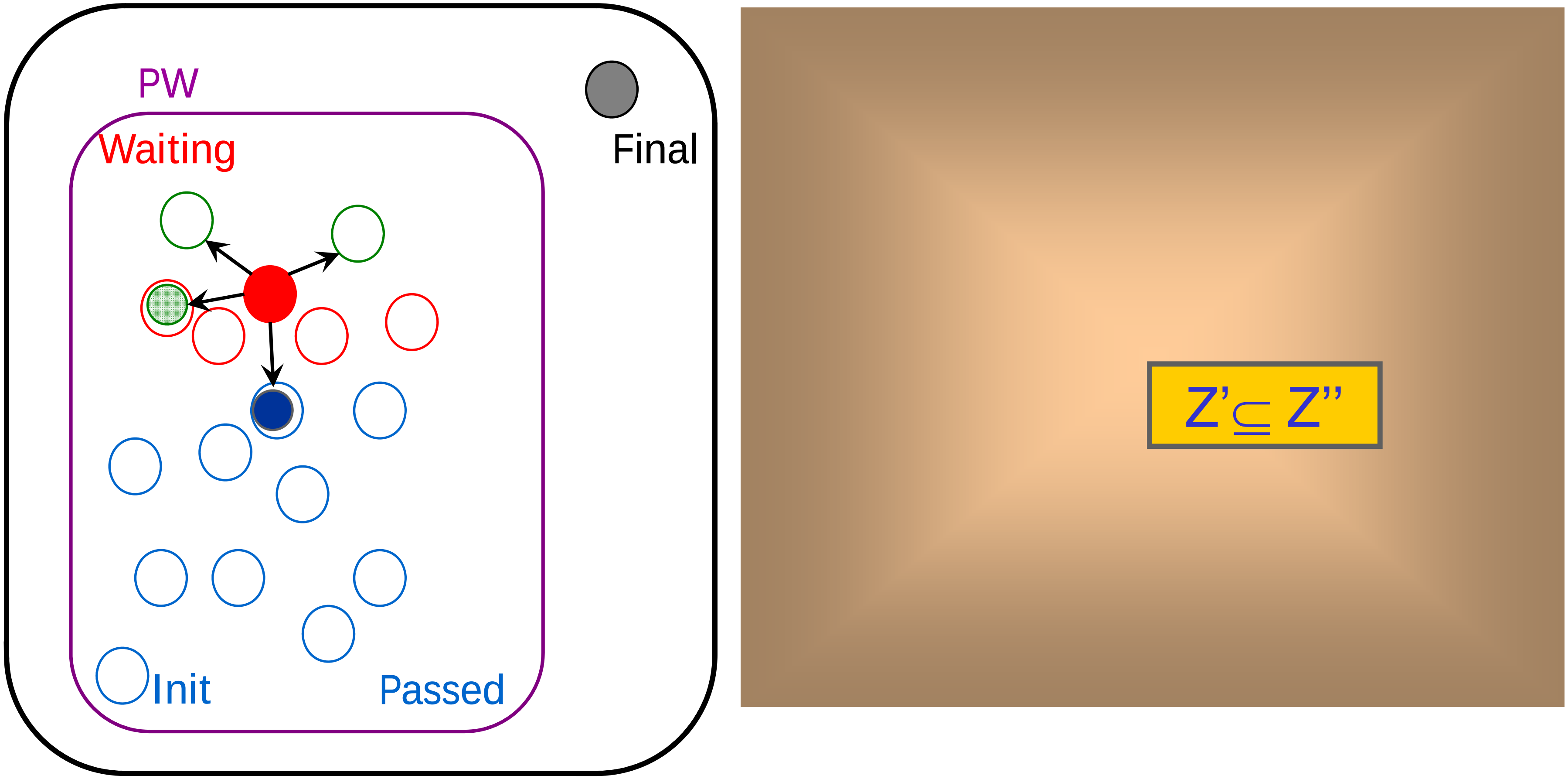
Remove all bounds involving y and set y to 0



Clock Difference Diagrams

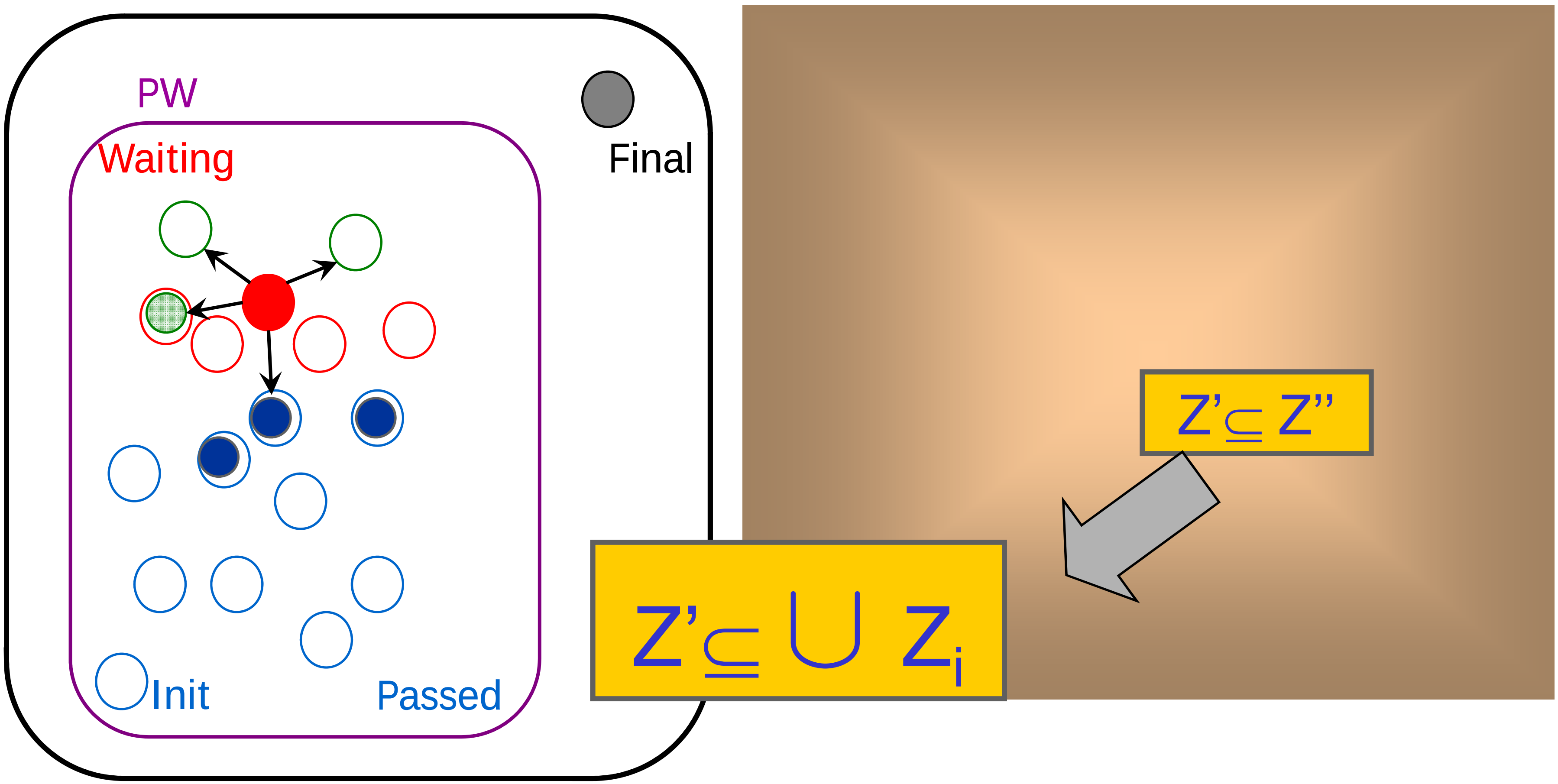
Earlier Termination

Init -> Final ?



Earlier Termination

Init -> Final ?

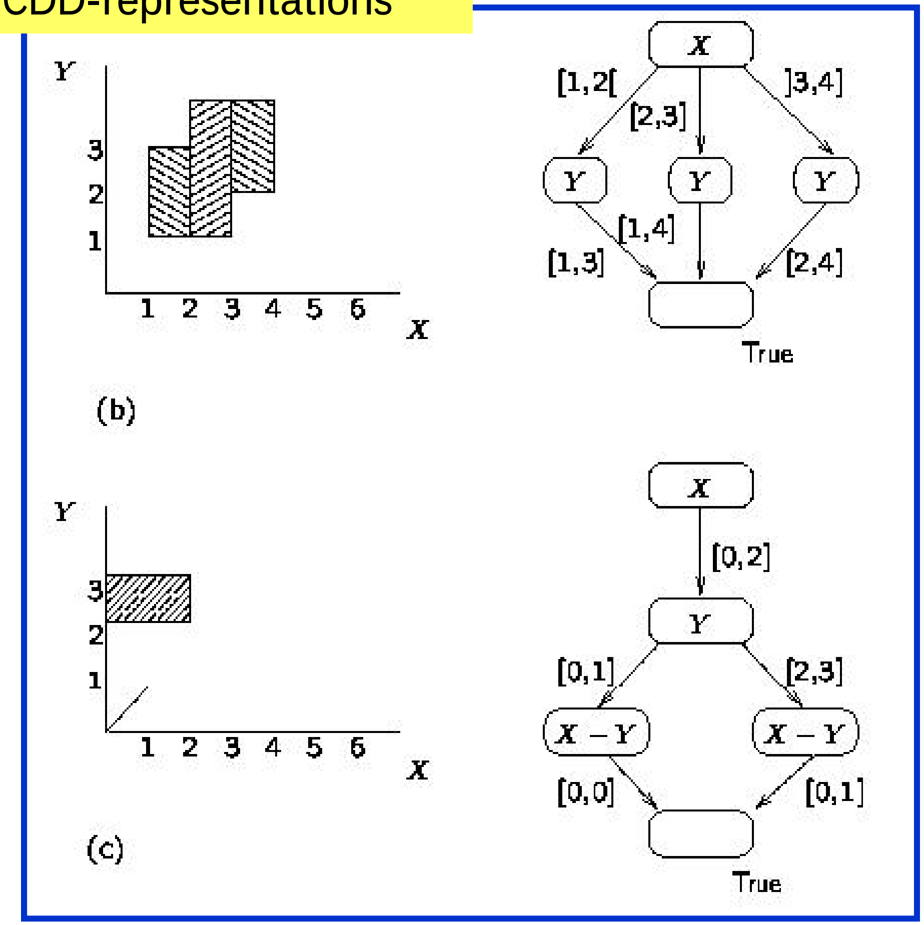


Clock Difference Diagrams

= Binary Decision Diagrams + Difference Bounded Matrices

CAV99

CDD-representations



- Nodes labeled with differences
- Maximal sharing of substructures (also across different CDDs)
- Maximal intervals
- Linear-time algorithms for set-theoretic operations.

- NDD's Maler et. al
- DDD's Møller, Lichtenberg

Clock Difference Diagrams I

A *Clock Difference Diagram* (CDD) is a directed acyclic graph consisting of a set of nodes V and two functions type : $V \rightarrow \mathcal{T}$ and succ : $V \rightarrow 2^{\mathcal{I} \times V}$ such that:

- V has exactly two *terminal nodes* called True and False, where $\mathsf{type}(\mathsf{True}) = \mathsf{type}(\mathsf{False}) = (0,0)$ and $\mathsf{succ}(\mathsf{True}) = \mathsf{succ}(\mathsf{False}) = \emptyset$.
- all other nodes $n \in V$ are *inner nodes*, which have attributed a type $\mathsf{type}(n) \in \mathcal{T}$ and a finite set of successors $\mathsf{succ}(n) = \{(I_1, n_1), \ldots, (I_k, n_K)\}$, where $(I_i, n_i) \in \mathcal{I} \times V$.

Disjoint & Ordered

For each inner node n, the following must hold:

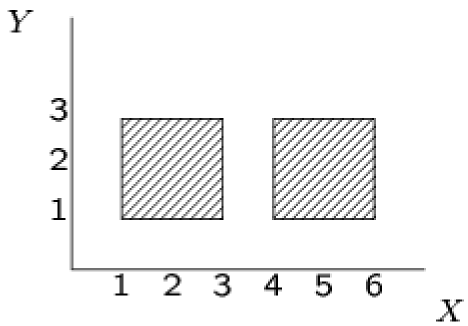
- the successors are *disjoint*: for $(I, m), (I', m') \in \mathsf{succ}(n)$ either $(I, m) = (I', m')$ or $I \cap I' = \emptyset$,
- the successor set is an $\mathbb{R}$-*cover*: $\bigcup\{I | \exists m . n \xrightarrow{I} m\} = \mathbb{R}$,
- the CDD is *ordered*: for all m, whenever $n \xrightarrow{I} m$ then $\mathsf{type}(m) \sqsubseteq \mathsf{type}(n)$

Reduced

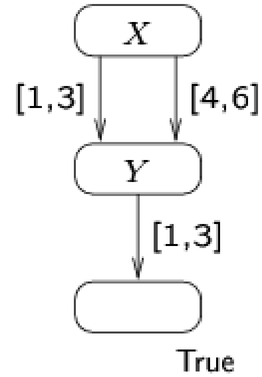
Further, the CDD is assumed to be *reduced*, i.e.

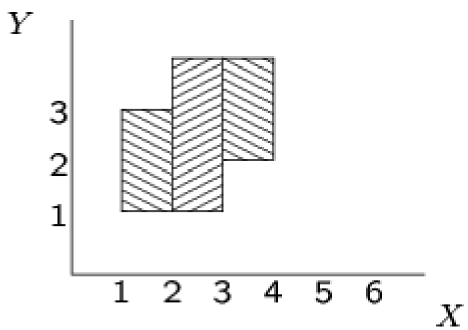
- it has *maximal sharing*: for all $n, m \in V$, whenever $\mathsf{succ}(n) = \mathsf{succ}(m)$ then $n = m$,
- it has *no trivial edges*: whenever $n \xrightarrow{I} m$ then $I \neq \mathbb{R}$,
- all intervals are *maximal*: whenever $n \xrightarrow{I_1} m, n \xrightarrow{I_2} m$ then $I_1 = I_2$ or $I_1 \cup I_2 \notin \mathcal{I}$

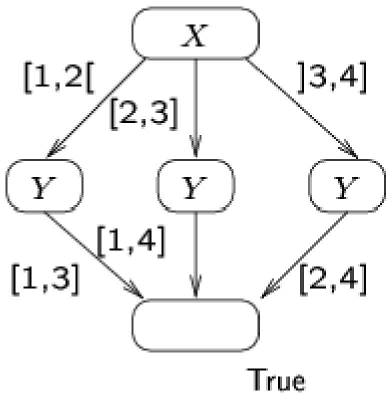
Clock Difference Diagrams



(a)







Makenode

Let t be a type and $S = \{(I_1, n_1), \ldots (I_k, n_K)\}$ a successor set. We want to extend a given CDD $C = (V, \mathsf{type}, \mathsf{succ})$ with a node n with these attributes.

MN(t, S):

  if $(\exists n \in V.\mathsf{type}(n) = t \wedge \mathsf{succ}(n) = S)$

  then return n

  else

    $V := V \cup \{n\}$ // n fresh

    $\mathsf{type} := \mathsf{type} \cup \{n \mapsto t\}$

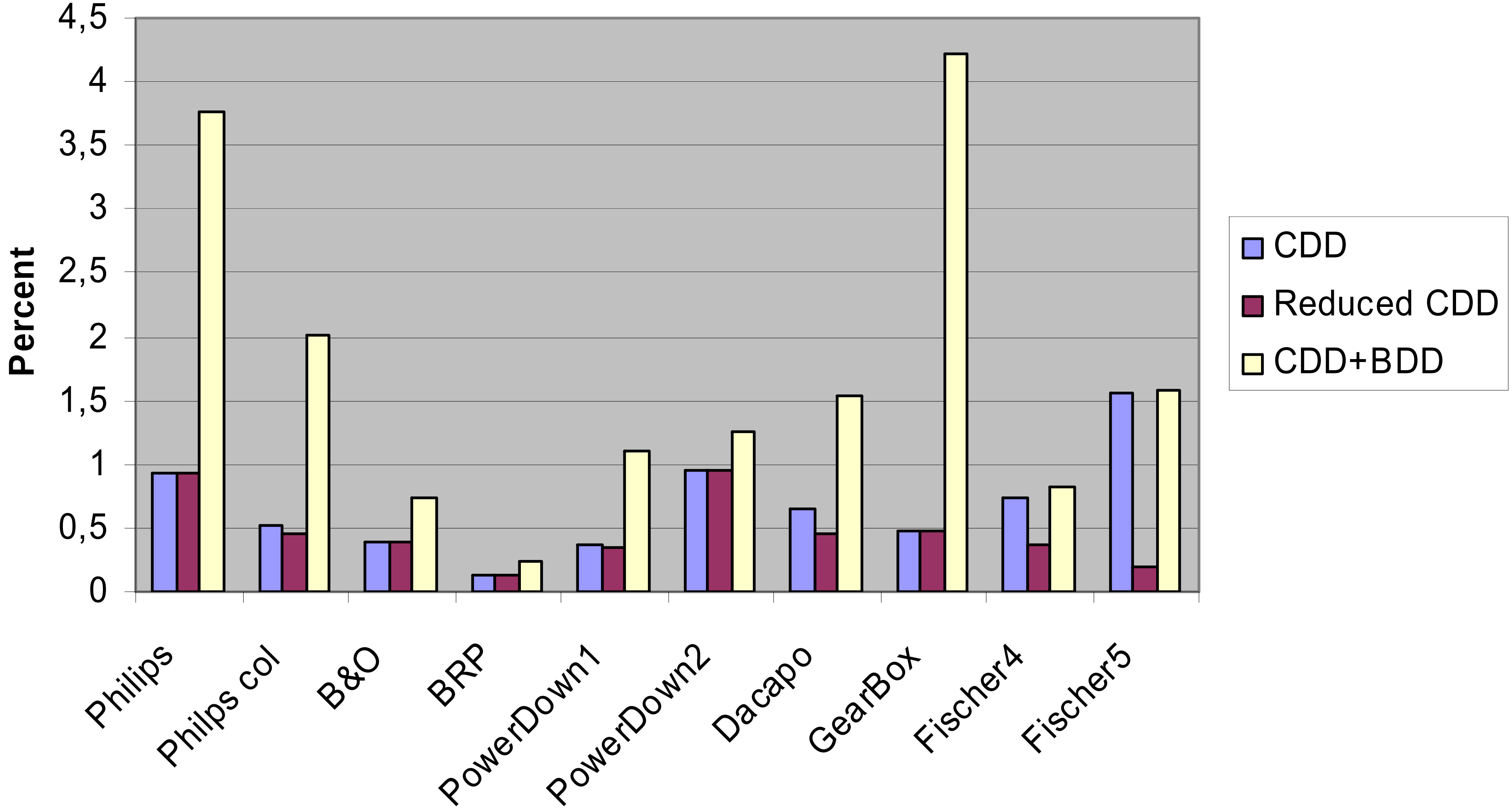
    $\mathsf{succ} := \mathsf{succ} \cup \{n \mapsto S\}$

    return n

  endif

MN in "constant" time

SPACE PERFORMANCE



Union

union(n_1, n_2)

    if $n_1 = \text{True}$ or $n_2 = \text{True}$ then return True

    elseif $n_1 = \text{False}$ then return n_2

    elseif $n_2 = \text{False}$ then return n_1

    else

        if $\text{type}(n_1) = \text{type}(n_2)$ then

            return $\text{MN}\Big(\text{type}(n_1), \{(I_1 \cap I_2, \text{union}(n_1', n_2')) \mid n_1 \xrightarrow{I_1} n_1', n_2 \xrightarrow{I_2} n_2', I_1 \cap I_2 \neq \emptyset\}\Big)$

        elseif $\text{type}(n_1) \sqsubseteq \text{type}(n_2)$ then

            return $\text{MN}\Big(\text{type}(n_1), \{(I_1, \text{union}(n_1', n_2)) \mid n_1 \xrightarrow{I_1} n_1'\}\Big)$

        elseif $\text{type}(n_2) \sqsubseteq \text{type}(n_1)$ then

            return $\text{MN}\Big(\text{type}(n_2), \{(I_2, \text{union}(n_1, n_2')) \mid n_2 \xrightarrow{I_2} n_2'\}\Big)$

        endif

    endif

Complement

$\text{complement}(n)$

$\quad$ if $n = \text{True}$ return False

$\quad$ elseif $n = \text{False}$ return True

$\quad$ elseif return $\text{MN}\Big(\text{type}(n), \{(I, \text{complement}(m)) \mid n \xrightarrow{I} m\}\Big)$

$\quad$ endif

Zones, CDD, Subset

Single zones may be represented as single path CDD's in the obvious manner.

It may be advantageous to represent zones using a *minimal* set of constraints (see [RTSS97]).

subset(D, n) // D const. sys., n CDD-node

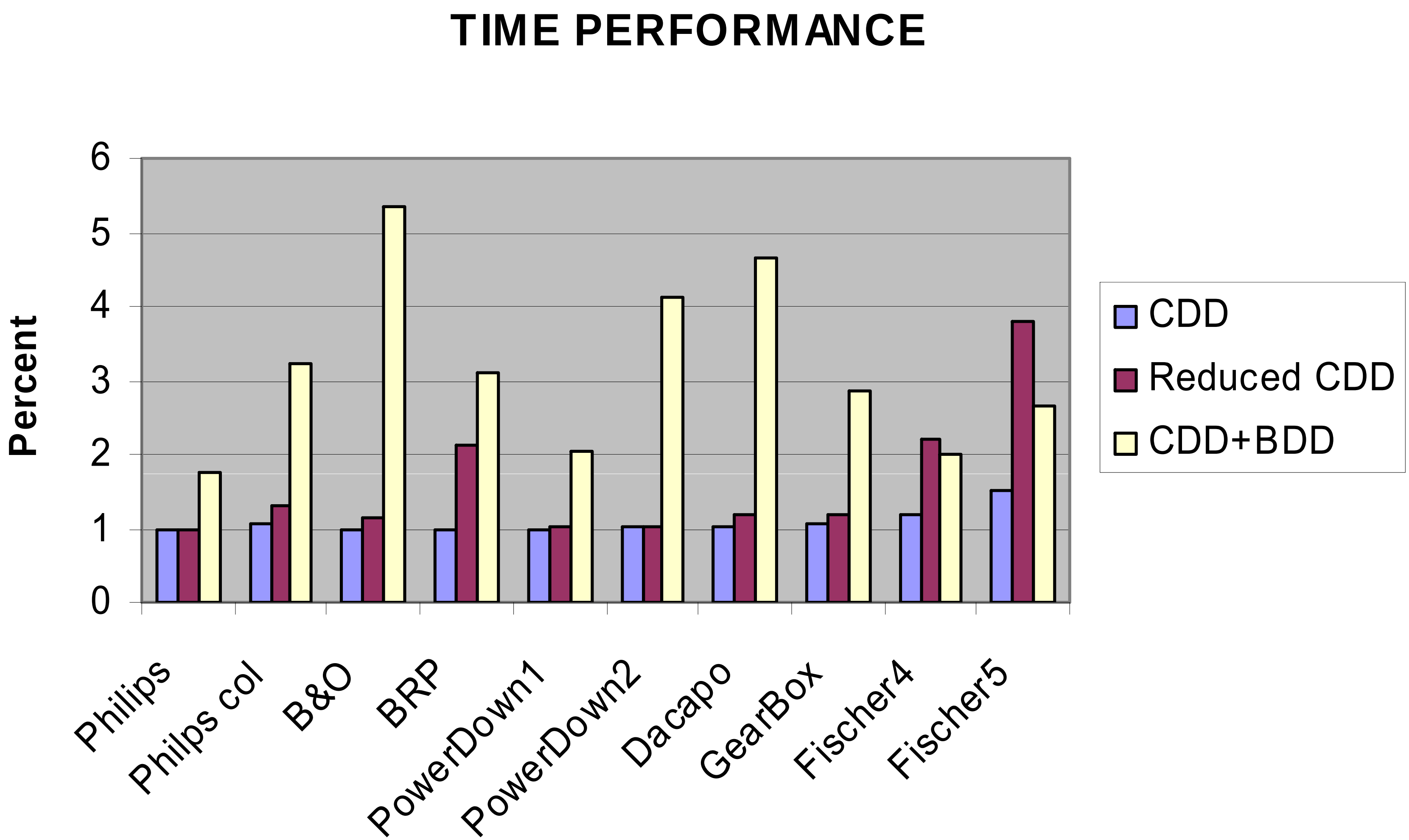
if $D =$ false or $n =$ True then return true

elseif $n =$ False then return false

else return $\bigwedge_{n \xrightarrow{I} m}$ subset$(D \wedge I_n), m)$

endif

where I_n is the constraint $X_i - X_j \in I$ if type$(n) = (i, j)$.


TIME PERFORMANCE
Percent
6
5
4
3
2
1
0
Philips
Philps col
B&O
BRP
PowerDown1
PowerDown2
Dacapo
GearBox
Fischer4
Fischer5
CDD
Reduced CDD
CDD+BDD

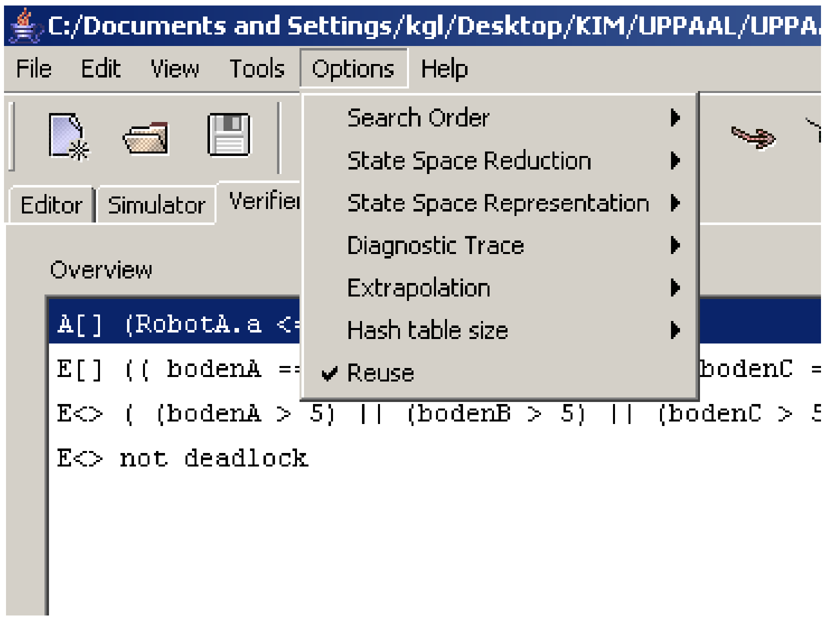
Related & Recent Work

- DDD: Andersen et al.
- NDD: Asarin, Bozga, Kerbrat, Maler, Pnueli, Rasse.
- IDD: Strehl, Thiele.

- Recent work on fully symbolic engine for TA:
  - Georges Morbe, Florian Pigorsch and Christoph Scholl: *Fully Symbolic Model Checking for Timed Automata.* CAV 2011.

Verification Options

Verification Options



Search Order
- Depth First
- Breadth First

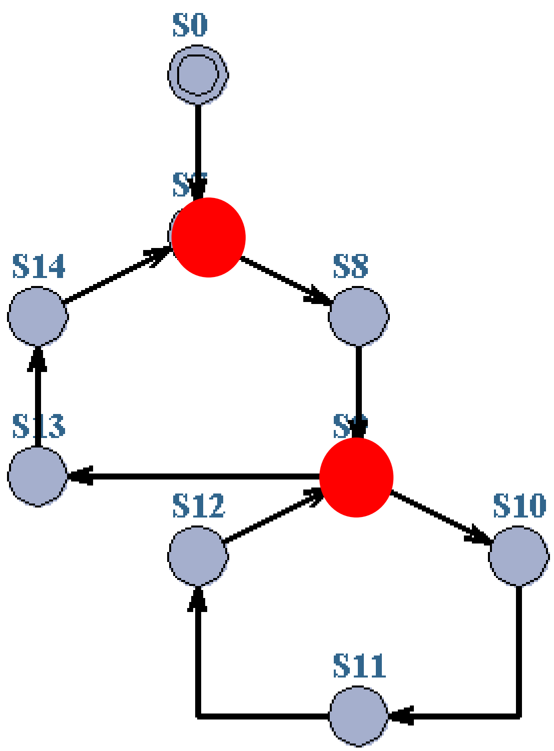
State Space Reduction
- None
- Conservative
- Aggressive

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

Diagnostic Trace
- Some
- Shortest
- Fastest

Extrapolation
Hash Table size
Reuse

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$_{total}$

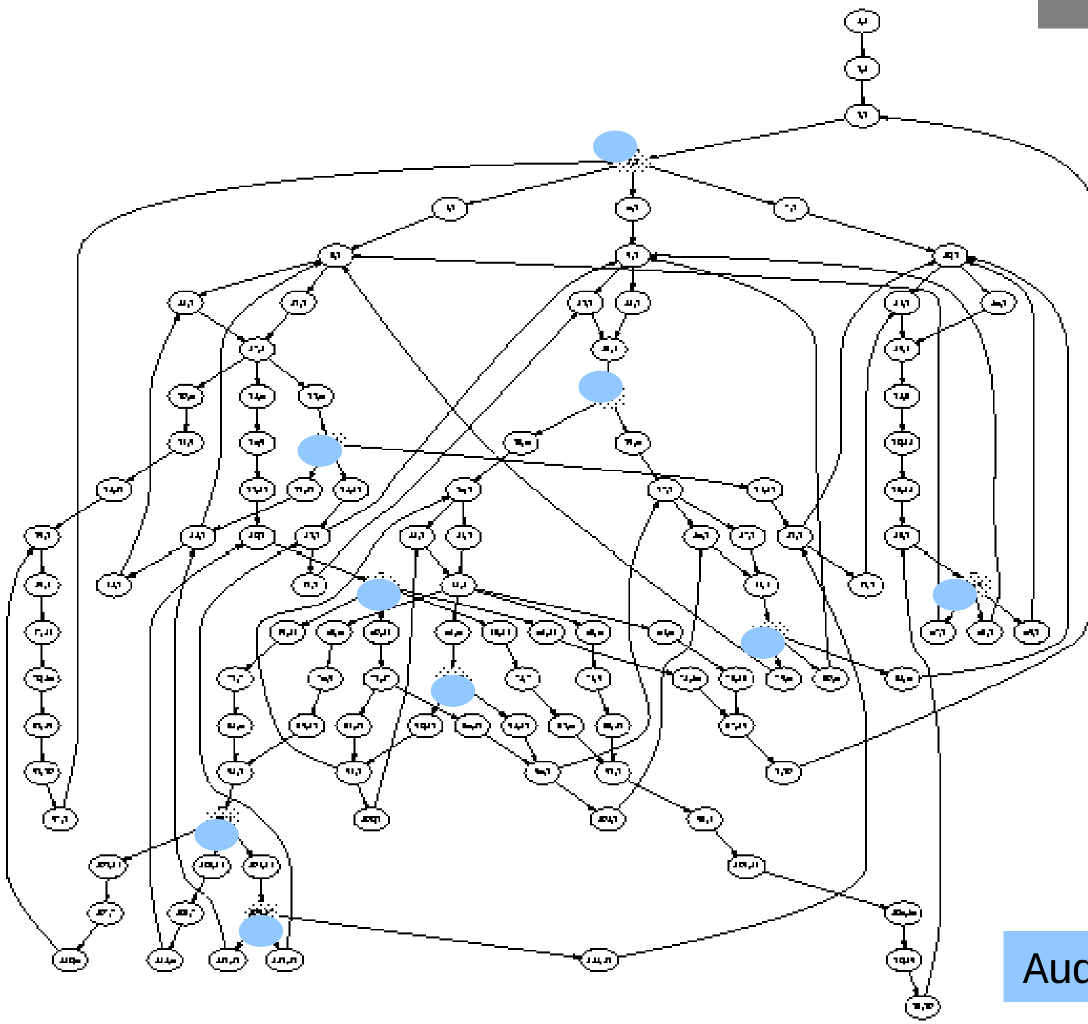
$\rightarrow$

81 states$_{entrypoint}$

$\rightarrow$

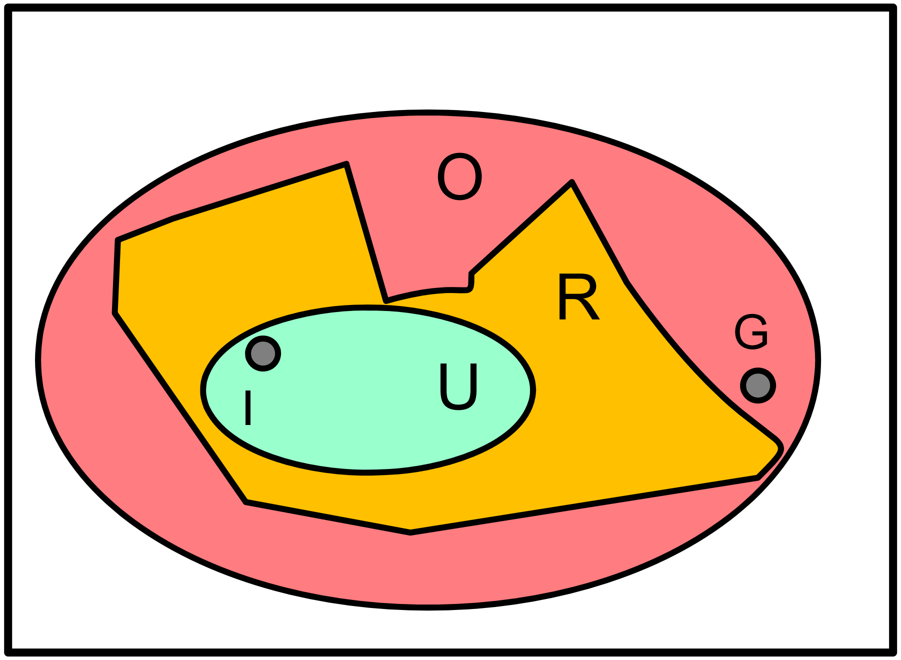
9 states

Time OH less than 10%



Audio Protocol

Over/Under Approximation



Declared State Space

Question:

$G \in R$?

How to use:

$G \in O$?

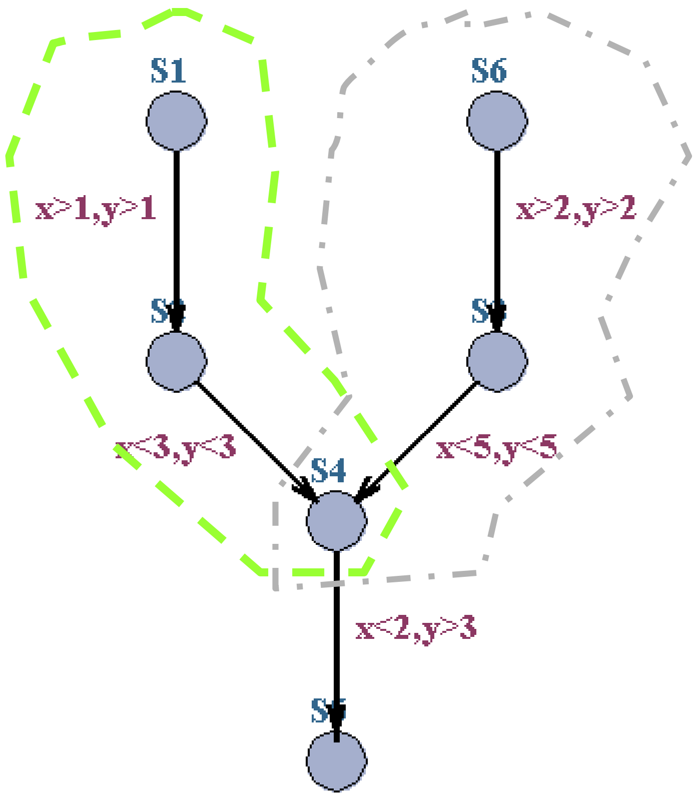
$G \in U$?

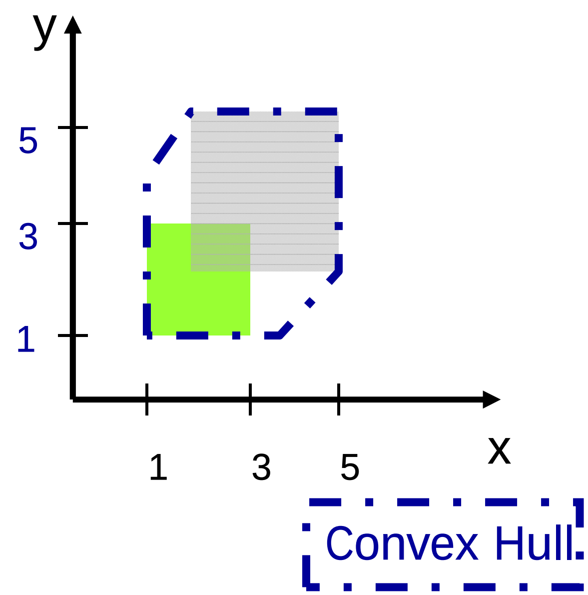
$$G \in U \Rightarrow G \in R$$

$$\neg(G \in O) \Rightarrow \neg(G \in R)$$

Over-approximation

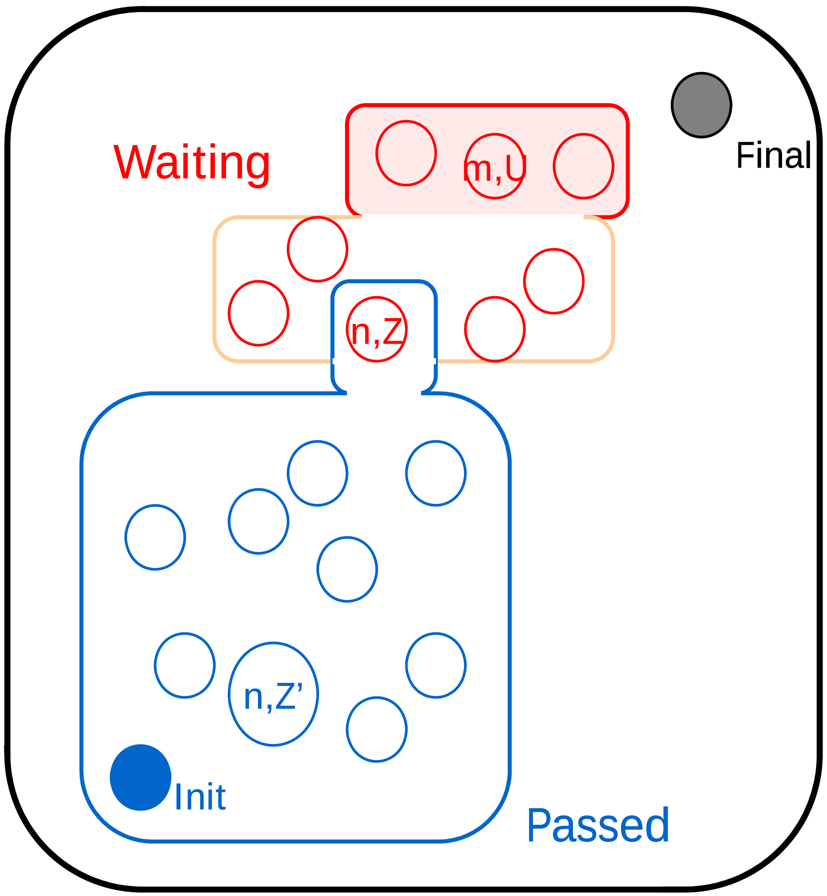
Convex Hull



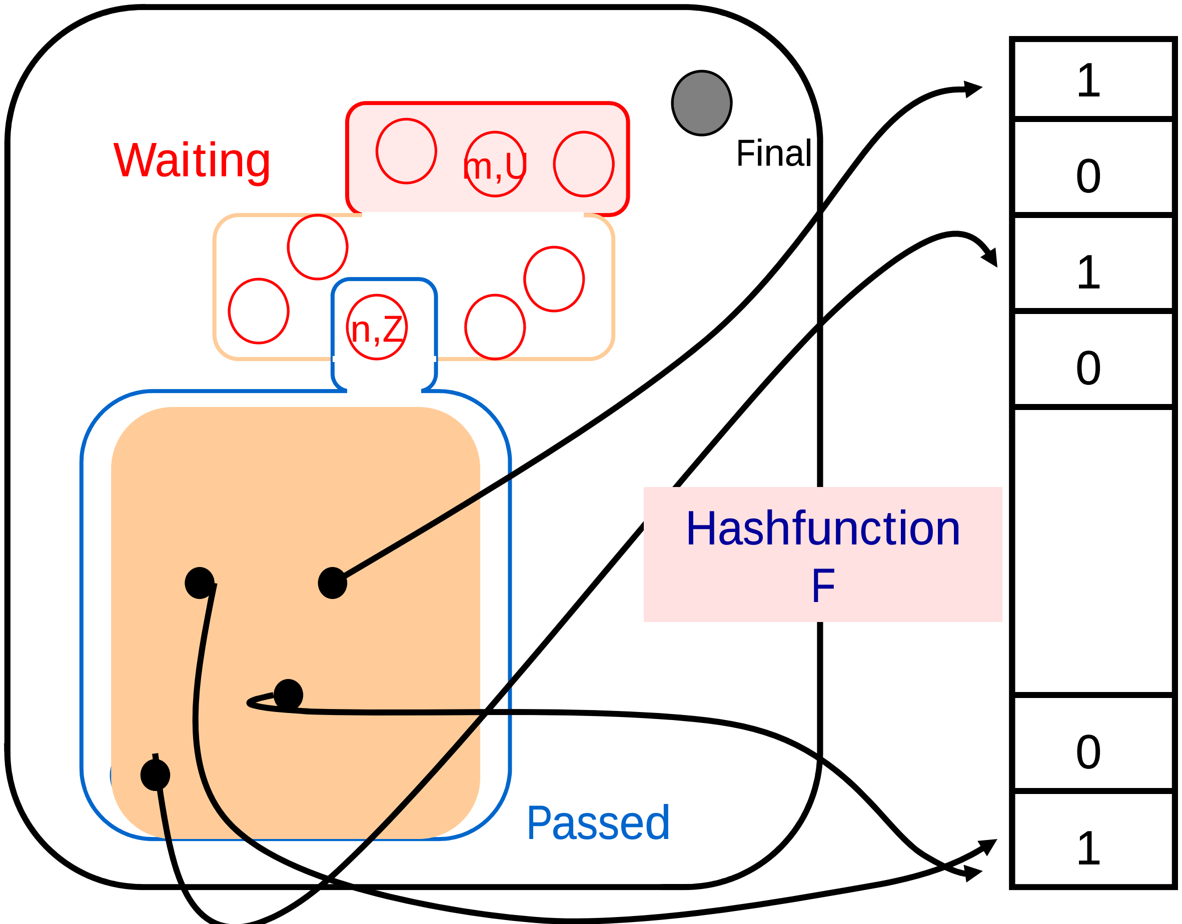


TACAS04: An EXACT method performing as well as Convex Hull has been developed based on abstractions taking max constants into account distinguishing between clocks, locations and $\leq$ & $\geq$

Under-approximation
Bitstate Hashing



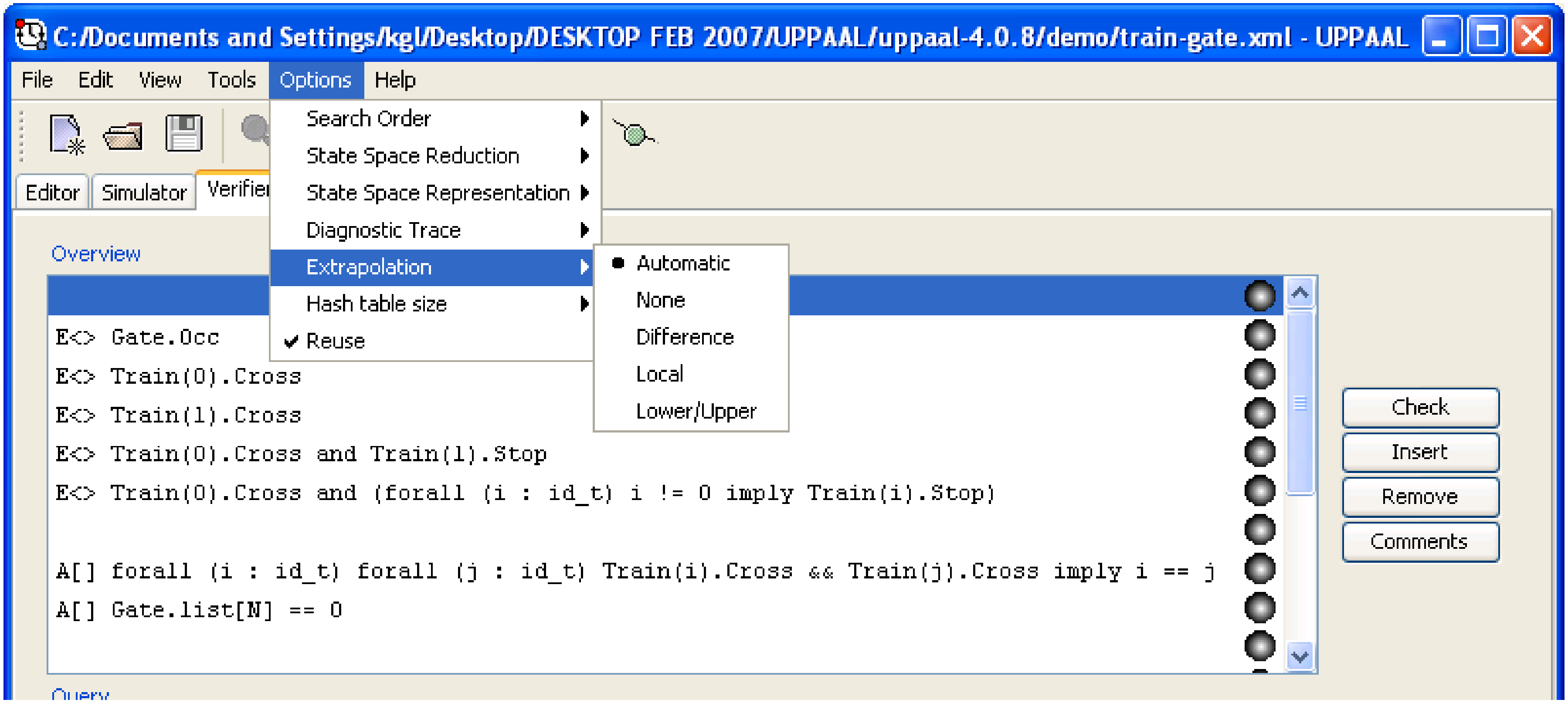
Under-approximation
Bitstate Hashing



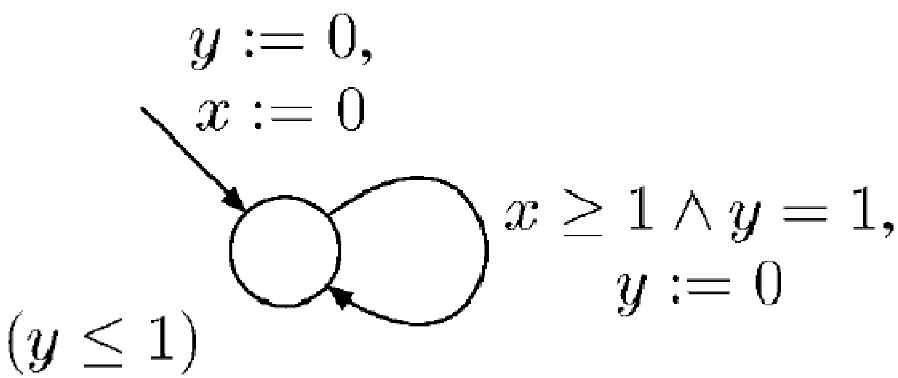
Passed= Bitarray

UPPAAL
4 - 512 Mbits

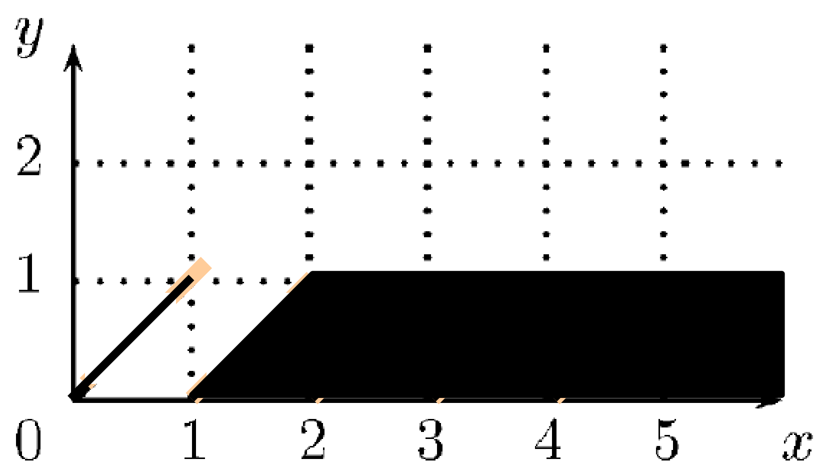
Extrapolation



Forward Symbolic Exploration



TERMINATION
not
garanteed



Need for
Finite
Abstractions

Abstractions

$$a : \mathcal{P}(R^X_{\geq 0}) \hookrightarrow \mathcal{P}(R^X_{\geq 0}) \text{ such that } W \subseteq a(W)$$

$$\frac{(\ell, W) \Rightarrow (\ell', W')}{(\ell, W) \Rightarrow_a (\ell', a(W'))} \quad \text{if } W = a(W)$$

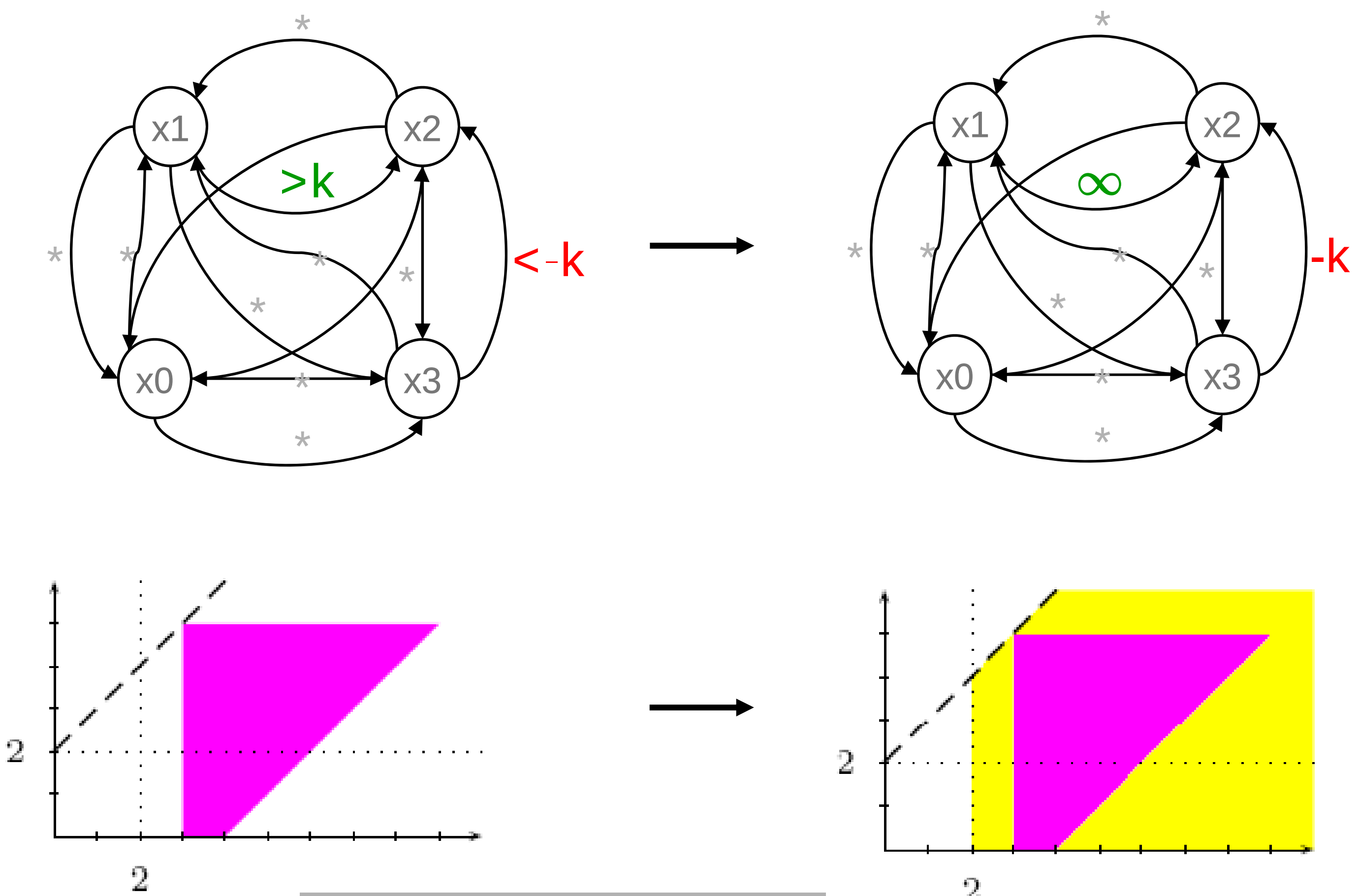
We want $\Rightarrow_a$ to be:

- sound & complete wrt reachability
- finite
- easy to compute
- as coarse as possible

Abstraction by Extrapolation

[Daws,Tripakis 98]

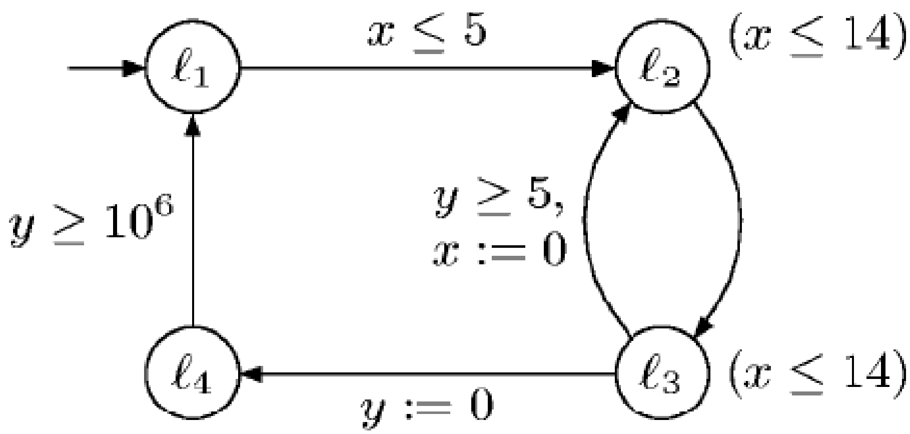
Let *k* be the largest constant appearing in the TA



Sound & Complete
Ensures Termination

Location Dependency

[Behrmann, Bouyer, Fleury, Larsen 03]



$k_x = 5 \quad k_y = 10^6$

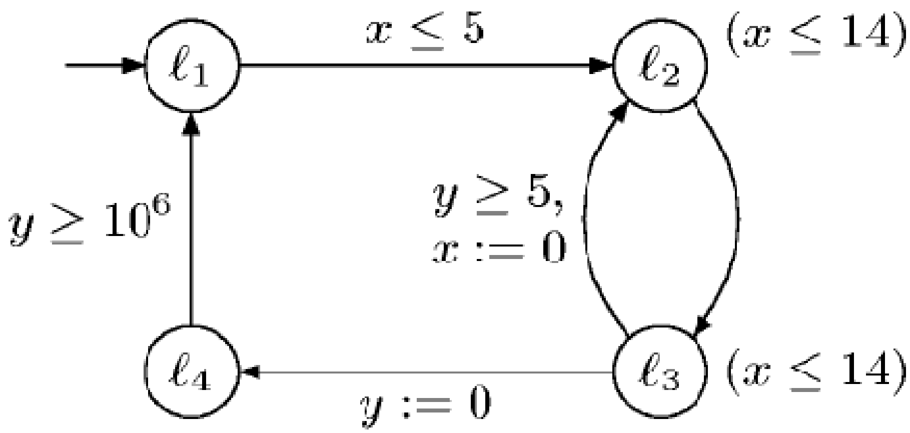
Will generate all symbolic states of the form

$(l_2,\ x \in [0,14]\ ,\ y \in [5,14n]\ ,\ y-x \in [5,14n-14])$

for n $\leq 10^6/14$!!

But $y \geq 10^6$ is not RELEVANT in l_2

Location Dependent Constants



$k_x = 5 \quad k_y = 10^6$

k_x^i	$= 14$	for $i \in \{1,2,3,4\}$
k_y^i	$= 5$	for $i \in \{1,2,3\}$
k_y^4	$= 10^6$	

k_j^i may be found as solution to simple linear constraints!

Active Clock Reduction:

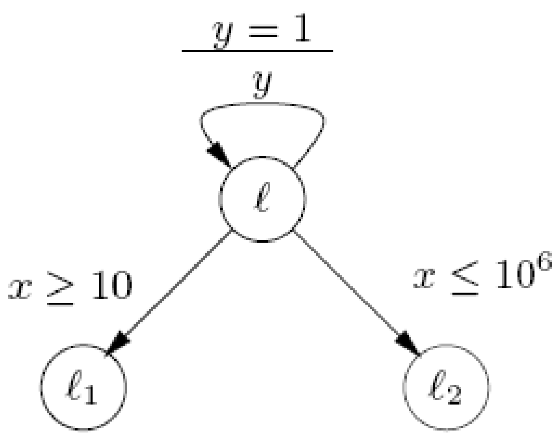
$k_i^j = -\infty$

Experiments

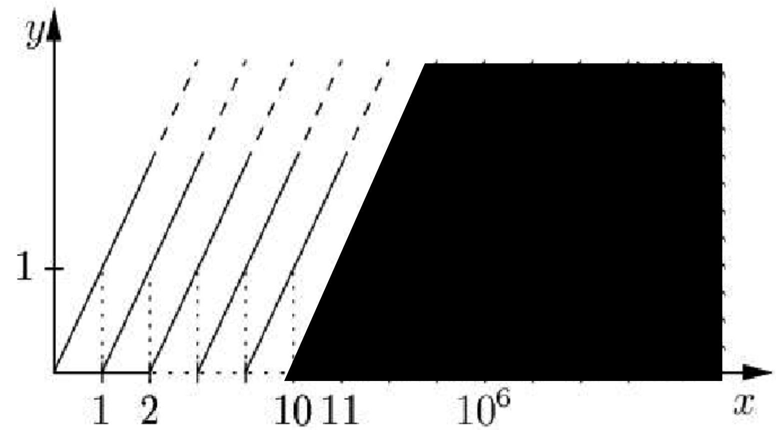
	Constant BIG	Global Method	Active-clock Reduction	Local Constants
Naive Example	10^3	0.05s/1MB	0.05s/1MB	0.00s/1MB
	10^4	4.78s/3MB	4.83s/3MB	0.00s/1MB
	10^5	484s/13MB	480s/13MB	0.00s/1MB
	10^6	stopped	stopped	0.00s/1MB
Two Processes	10^3	3.24s/3MB	3.26s/3MB	0.01s/1MB
	10^4	5981s/9MB	5978s/9MB	0.37s/2MB
	10^5	stopped	stopped	72s/5MB
Asymmetric Fischer	10^3	0.01s/1MB	0.01s/1MB	0.01s/1MB
	10^4	2.20s/3MB	2.20s/3MB	0.85s/2MB
	10^5	333s/19MB	333s/19MB	160s/13MB
	10^6	33307s/122MB	33238s/122MB	16330s/65MB
Bang & Olufsen	25000	stopped	159s/243MB	123s/204MB

Lower and Upper Bounds

[Behrmann, Bouyer, Larsen, Pelanek 04]



$k_x^l = 10^6$



Given that $x \leq 10^6$ is an *upper* bound implies that

(l,v_x,v_y) *simulates* (l,v'_x,v_y)

whenever $v'_x \geq v_x \geq 10$.

For reachability downward closure wrt simulation suffices!

Simulation

$\preccurlyeq$ is the largest relation satisfying

1. if $(\ell_1, \nu_1) \preccurlyeq (\ell_2, \nu_2)$ then $\ell_1 = \ell_2$
2. if $(\ell_1, \nu_1) \preccurlyeq (\ell_2, \nu_2)$ and $(\ell_1, \nu_1) \longrightarrow (\ell_1', \nu_1')$, then there exists (ℓ_2', ν_2') such that $(\ell_2, \nu_2) \longrightarrow (\ell_2', \nu_2')$ and $(\ell_1', \nu_1') \preccurlyeq (\ell_2', \nu_2')$
3. if $(\ell_1, \nu_1) \preccurlyeq (\ell_2, \nu_2)$ and $(\ell_1, \nu_1) \xrightarrow{\epsilon(\delta)} (\ell_1, \nu_1 + \delta)$, then there exists δ' such that $(\ell_2, \nu_2) \xrightarrow{\epsilon(\delta')} (\ell_2, \nu_2 + \delta')$ and $(\ell_1, \nu_1 + \delta) \preccurlyeq (\ell_2, \nu_2 + \delta')$

Proposition

If $(\ell, \nu_1) \preccurlyeq (\ell, \nu_2)$ and if a discrete state ℓ' is reachable from (ℓ, ν_1), then it is also reachable from (ℓ, ν_2).

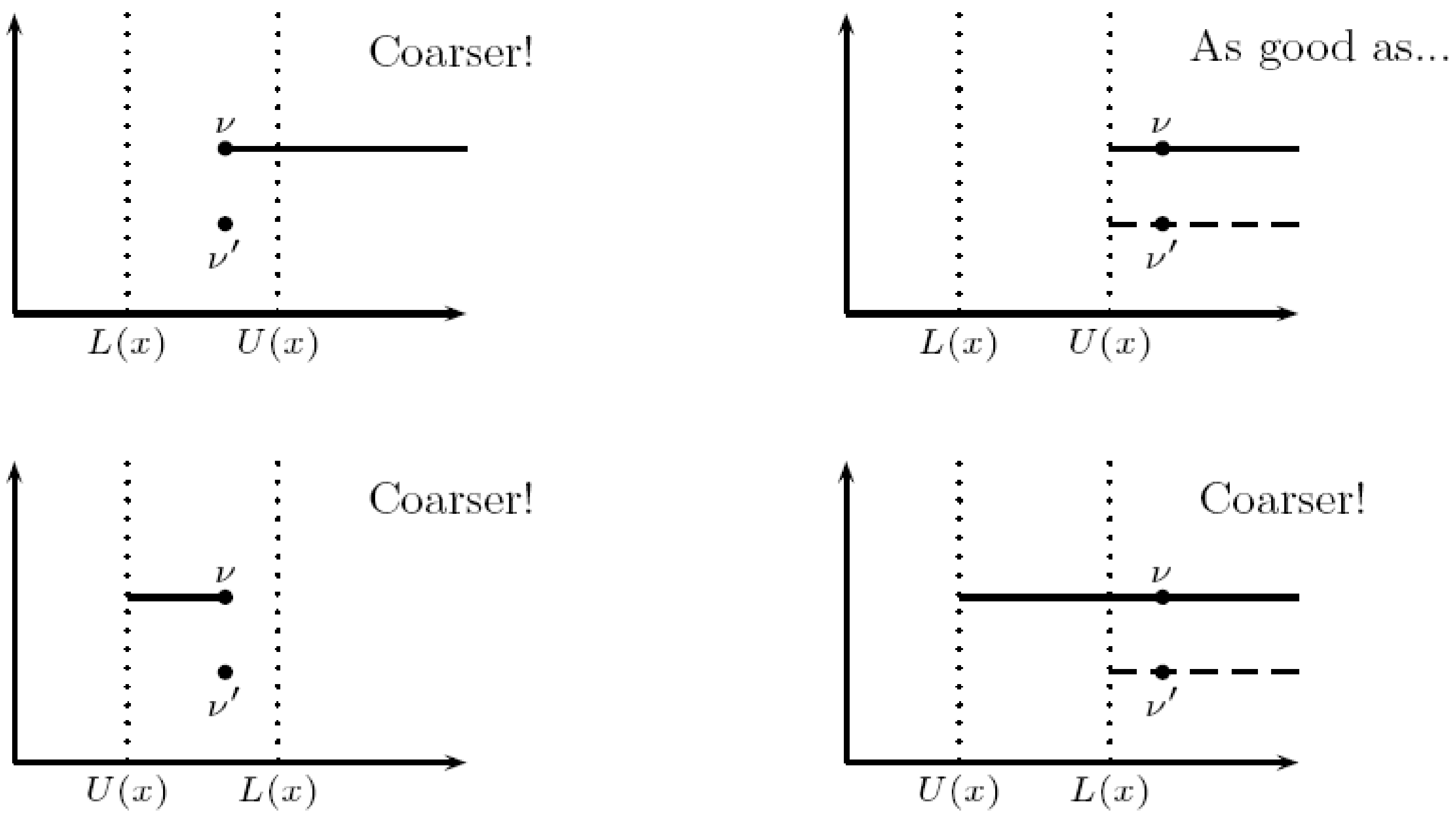
Maximal Bounds

$M(x)$: the maximum constant k with $x \sim k$,
$L(x)$: the maximum constant k with $x\{\geq, >\}k$,
$U(x)$: the maximum constant k with $x\{\leq, <\}k$.

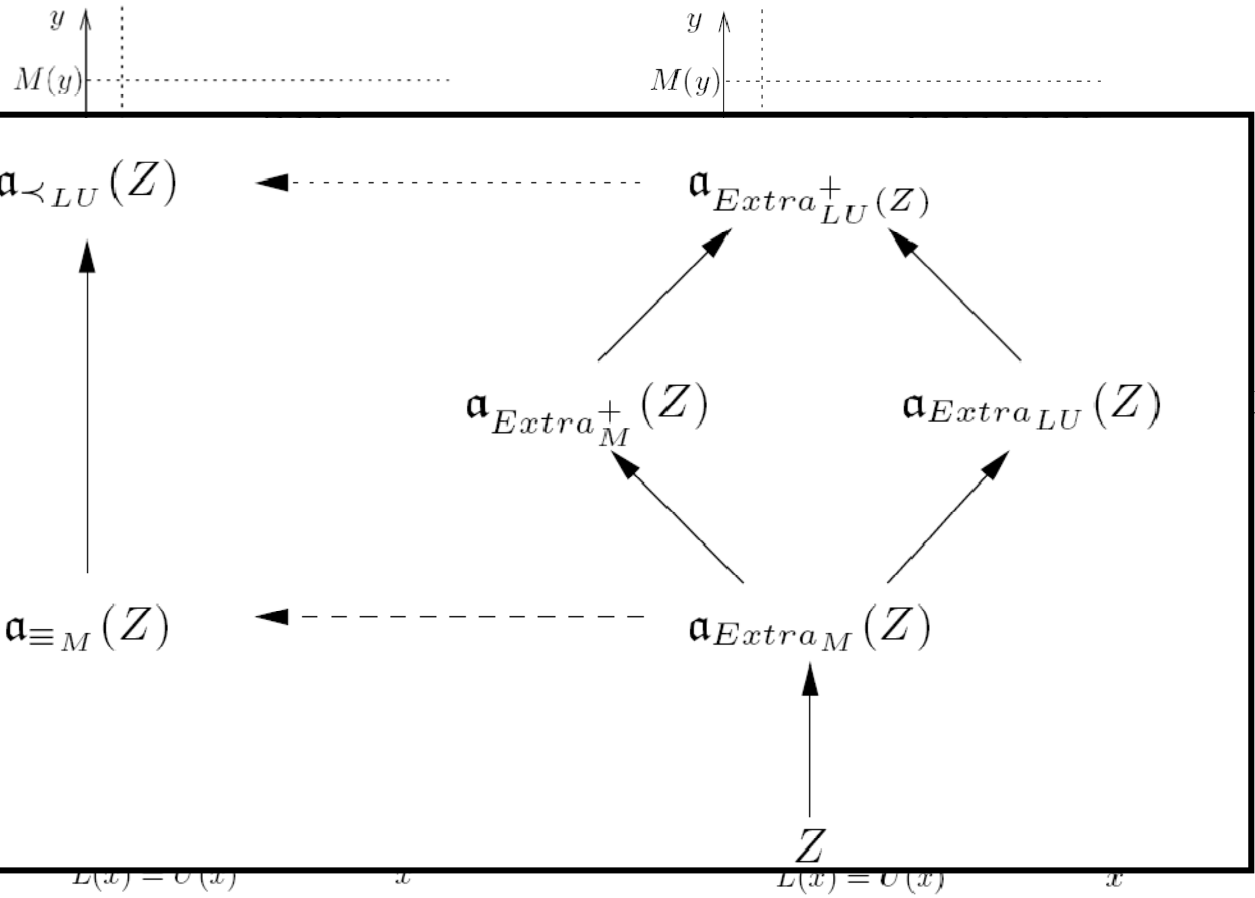
$$\nu \equiv_M \nu' \stackrel{\text{def}}{\Longleftrightarrow}$$
$$\forall x \in X : \text{either } \nu(x) = \nu'(x) \text{ or } (\nu(x) > M(x) \text{ and } \nu'(x) > M(x))$$

$$\nu' \prec_{LU} \nu \stackrel{def}{\Longleftrightarrow} \textit{for each clock } x, \begin{cases} \textit{either } \nu'(x) = \nu(x) \\ \textit{or } L(x) < \nu'(x) < \nu(x) \\ \textit{or } U(x) < \nu(x) < \nu'(x) \end{cases}$$

Maximum Bounds Abstraction



Extrapolation Using Zones



xtrapolation

l w LU

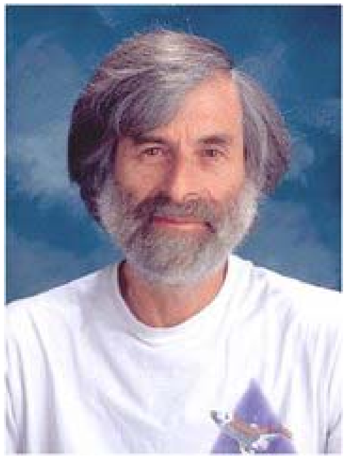
Experiments

		Classical			Loc. dep. Max			Loc. dep. LU			Convex Hull		
		-n1			-n2			-n3			-A		
	Model	Time	States	Mem	Time	States	Mem	Time	States	Mem	Time	States	Mem
Fischer	f5	4.02	82,685	5	0.24	16,980	3	0.03	2,870	3	0.03	3,650	3
	f6	597.04	1,489,230	49	6.67	158,220	7	0.11	11,484	3	0.10	14,658	3
	f7				352.67	1,620,542	46	0.47	44,142	3	0.45	56,252	5
	f8							2.11	164,528	6	2.08	208,744	12
	f9							8.76	598,662	19	9.11	754,974	39
	f10							37.26	2,136,980	68	39.13	2,676,150	143
	f11							152.44	7,510,382	268			
CSMA/CD	c5	0.55	27,174	3	0.14	10,569	3	0.02	2,027	3	0.03	1,651	3
	c6	19.39	287,109	11	3.63	87,977	5	0.10	6,296	3	0.06	4,986	3
	c7				195.35	813,924	29	0.28	18,205	3	0.22	14,101	4
	c8							0.98	50,058	5	0.66	38,060	7
	c9							2.90	132,623	12	1.89	99,215	17
	c10							8.42	341,452	29	5.48	251,758	49
	c11							24.13	859,265	76	15.66	625,225	138
	c12							68.20	2,122,286	202	43.10	1,525,536	394
	bus	102.28	6,727,443	303	66.54	4,620,666	254	62.01	4,317,920	246	45.08	3,826,742	324
	philips	0.16	12,823	3	0.09	6,763	3	0.09	6,599	3	0.07	5,992	3
	sched	17.01	929,726	76	15.09	700,917	58	12.85	619,351	52	55.41	3,636,576	427

Additional "secrets"

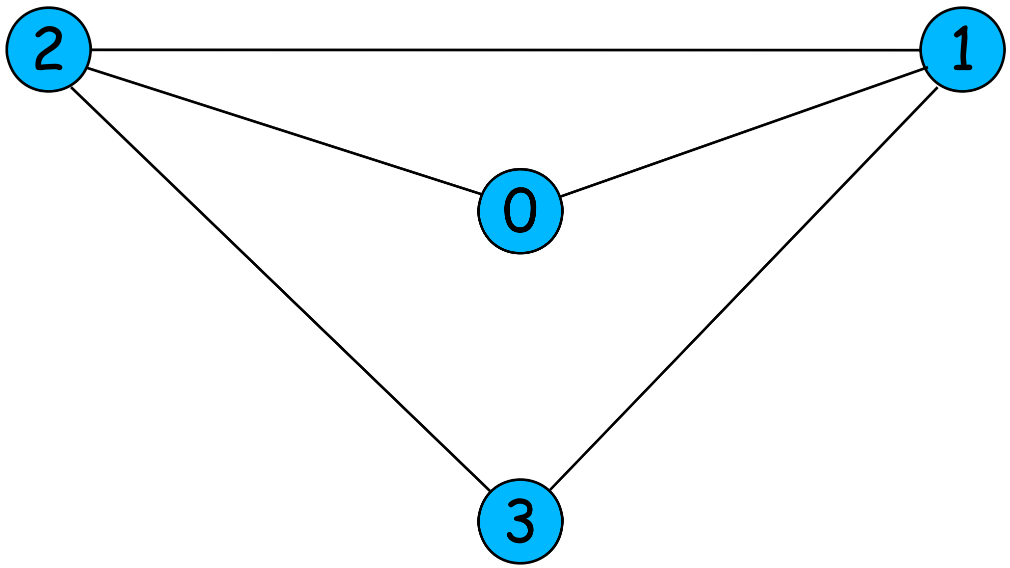
- Sharing among symbolic states
  - location vector / discrete values / zones
- Distributed implementation of UPPAAL
- Symmetry Reduction
- Sweep Line Method
- Guiding wrt Heuristic Value
  - User-supplied / Auto-generated
- Slicing wrt "C" Code

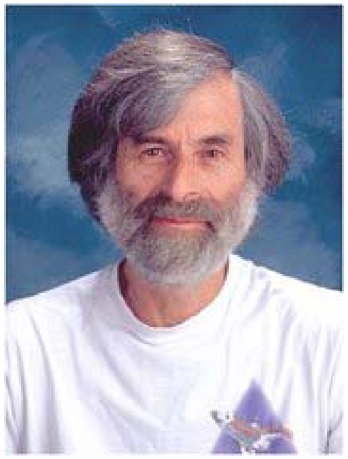
Leader Election Protocol



Protocol analysed in UPPAAL by
Leslie Lamport
CHARME'05

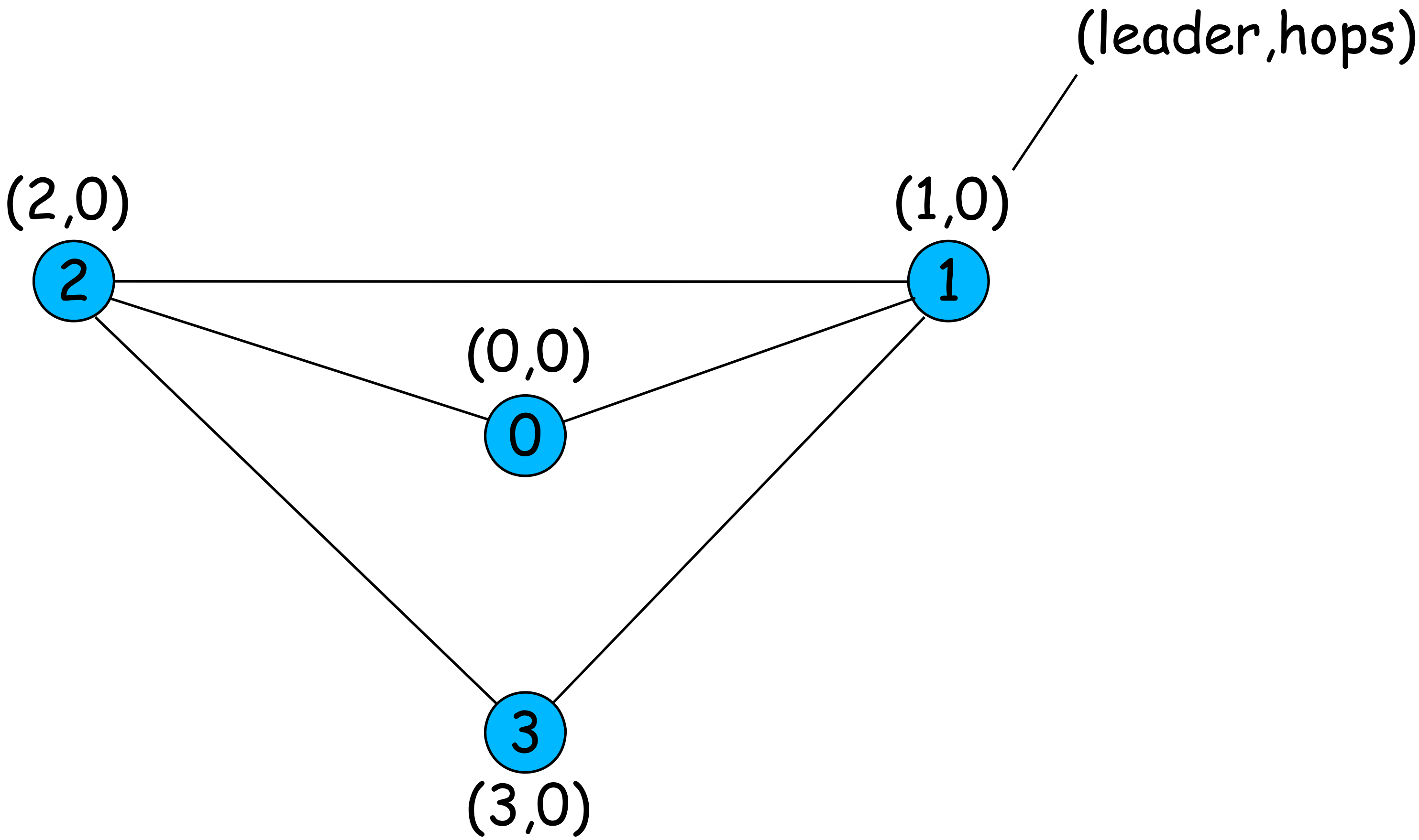
Leader Election



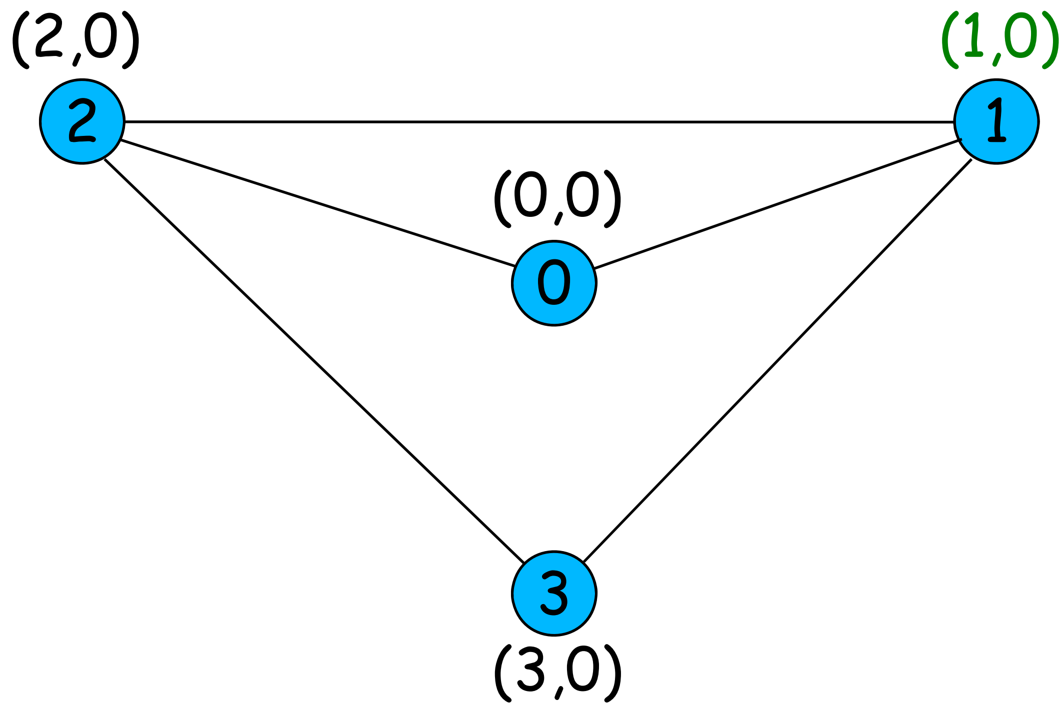


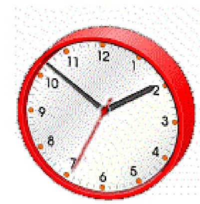
Protocol by
Leslie Lamport

Leader Election

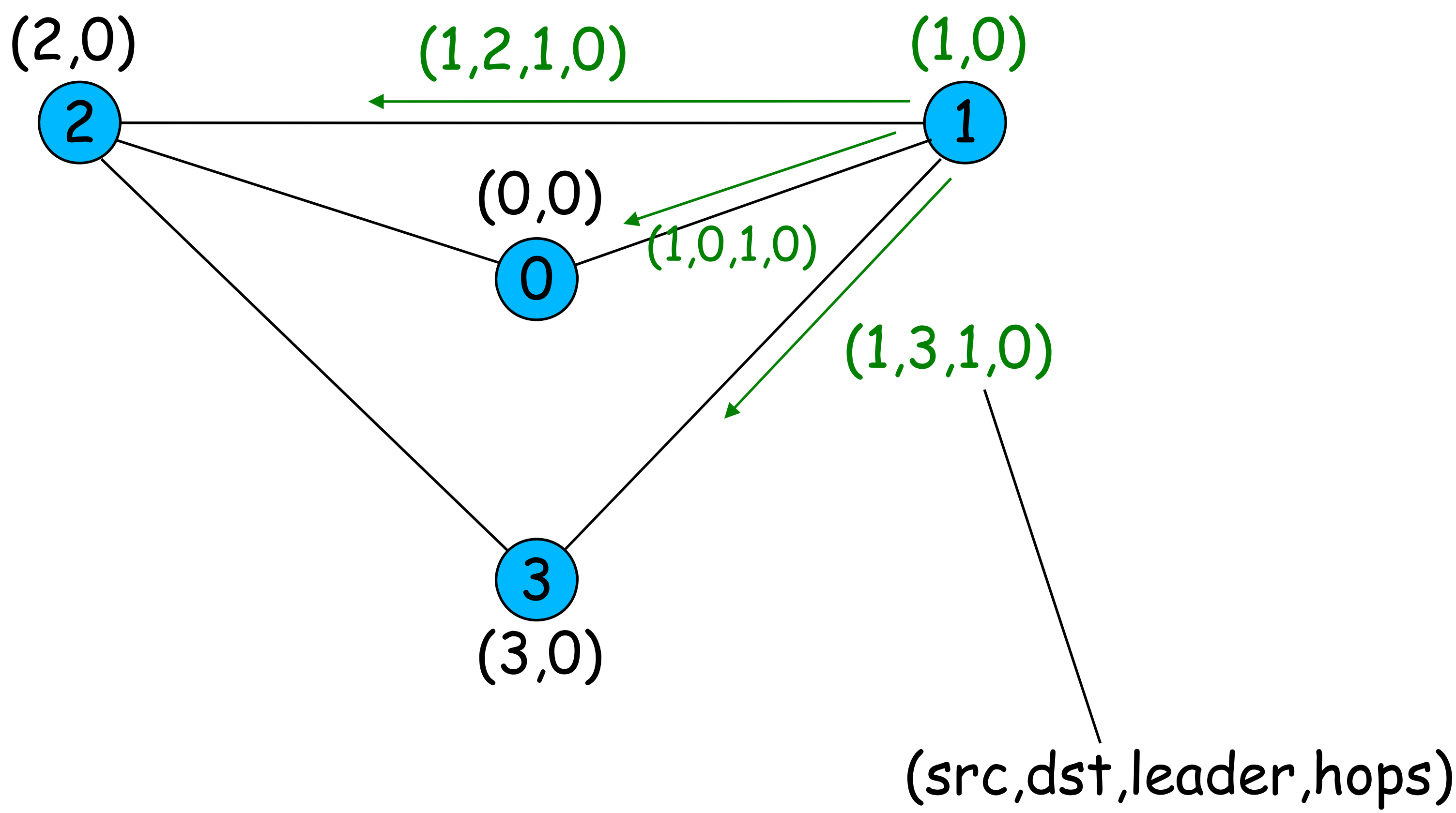


Timeout

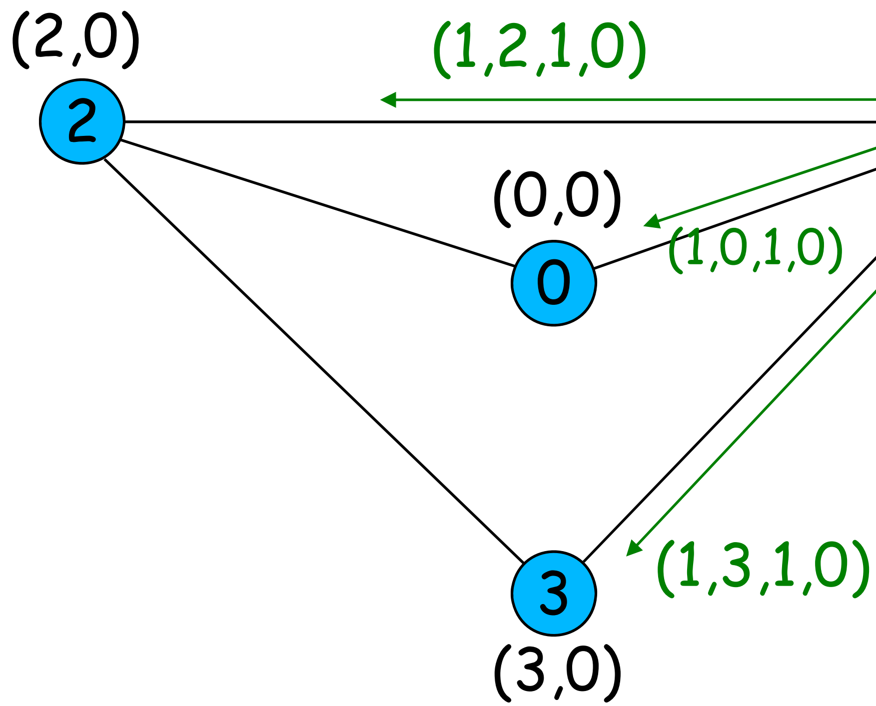




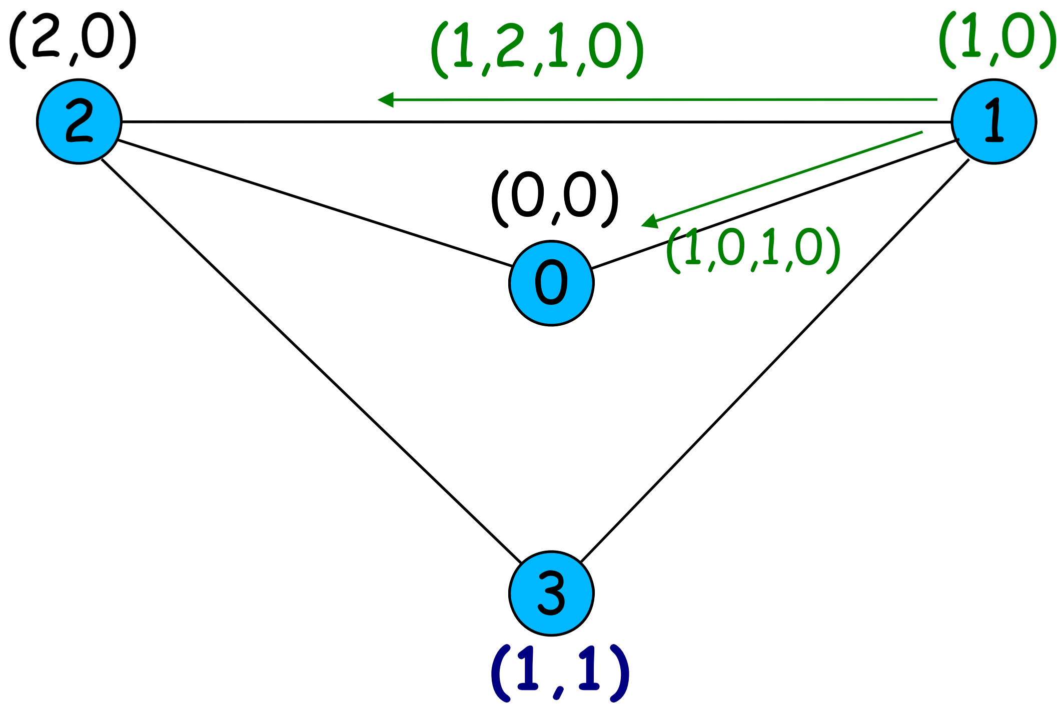
Flooding



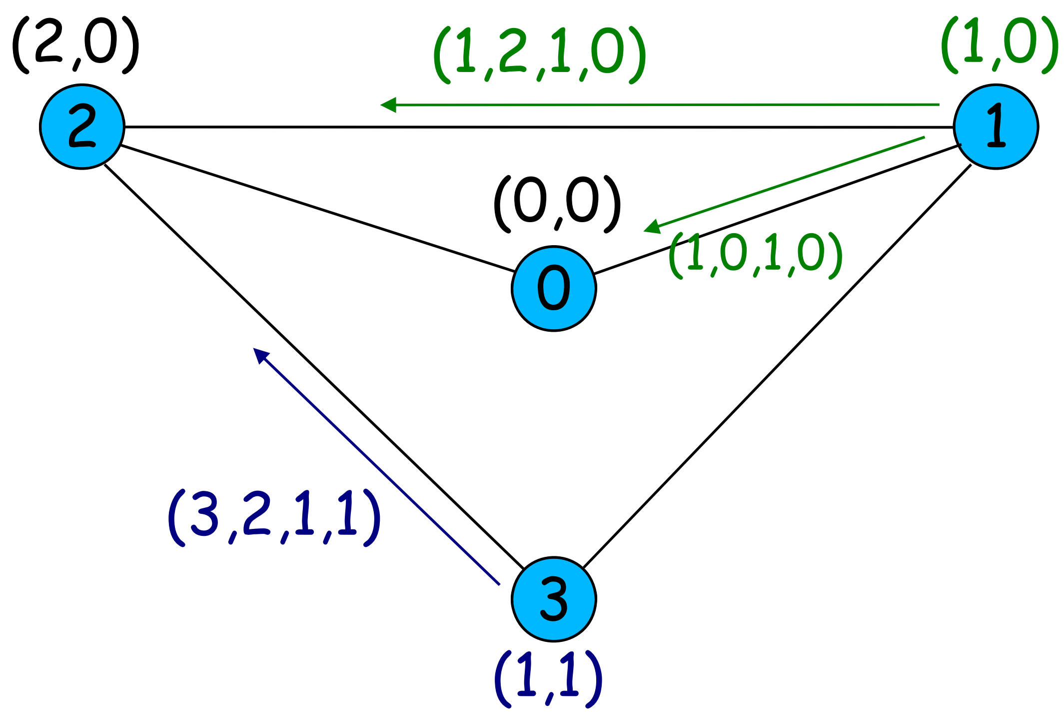
Flooding



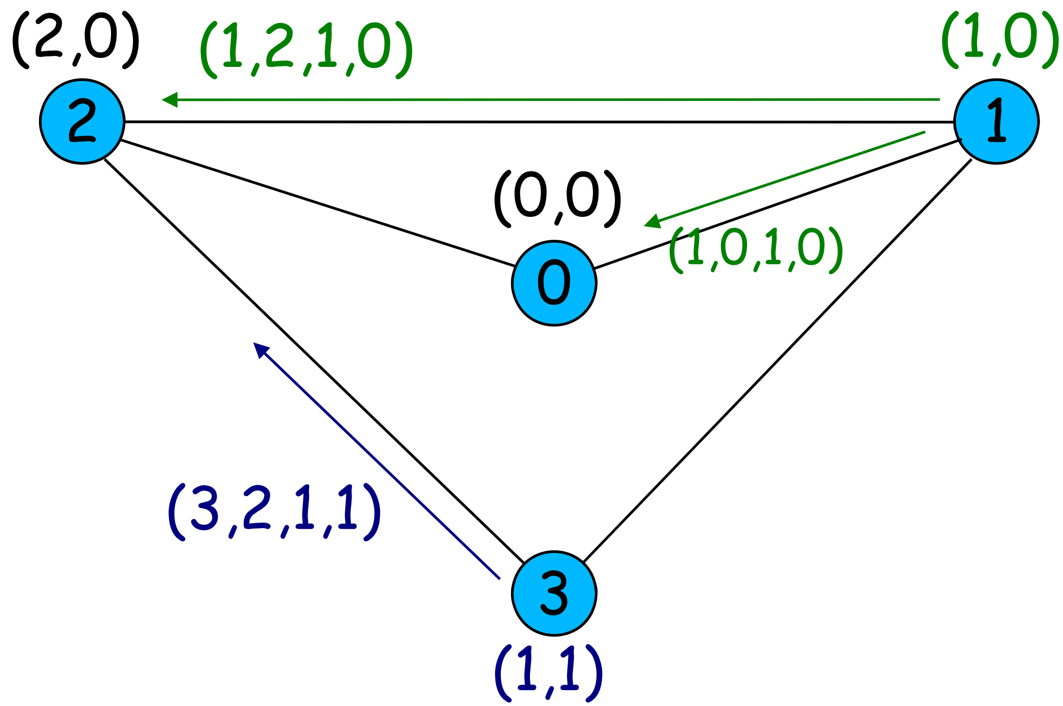
Flooding



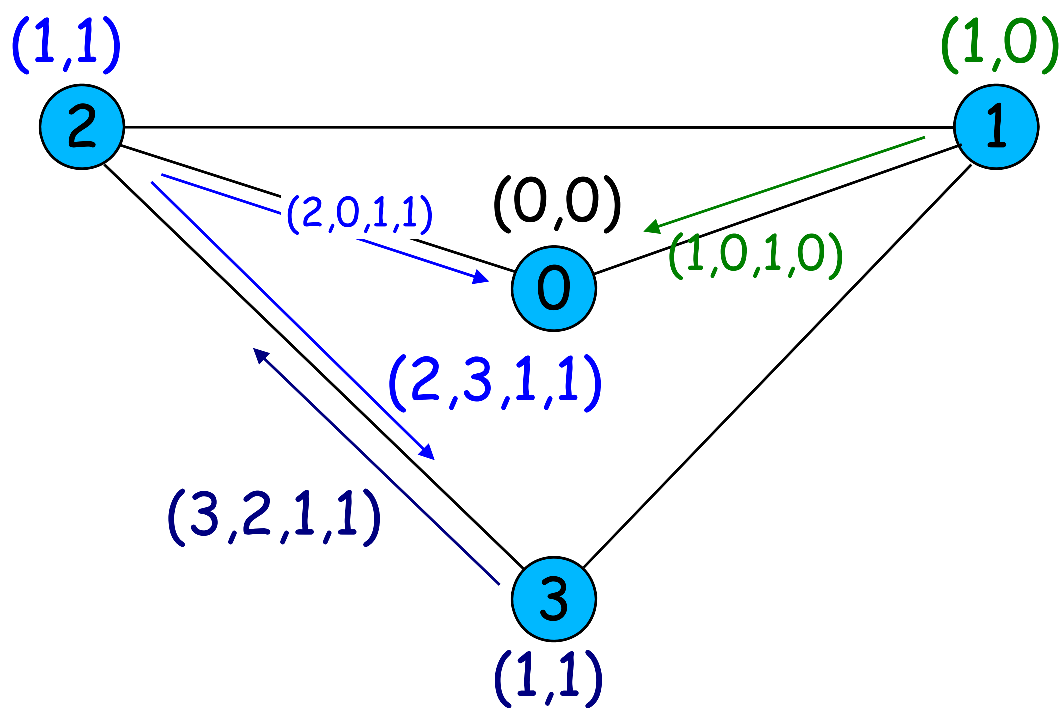
Forwarding



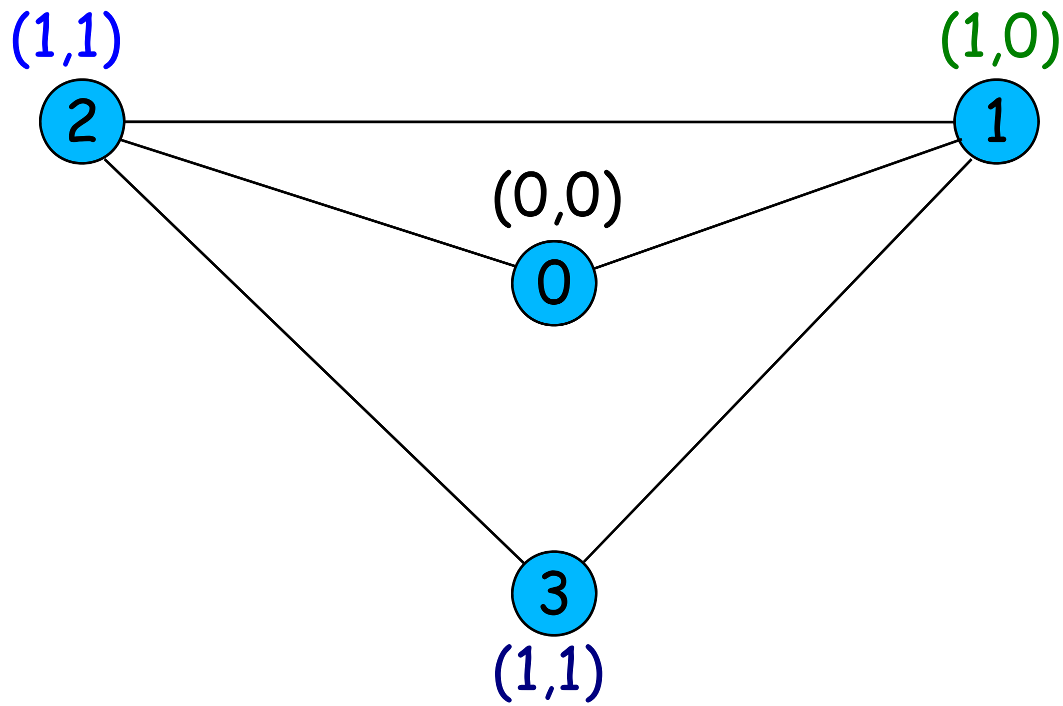
Forwarding



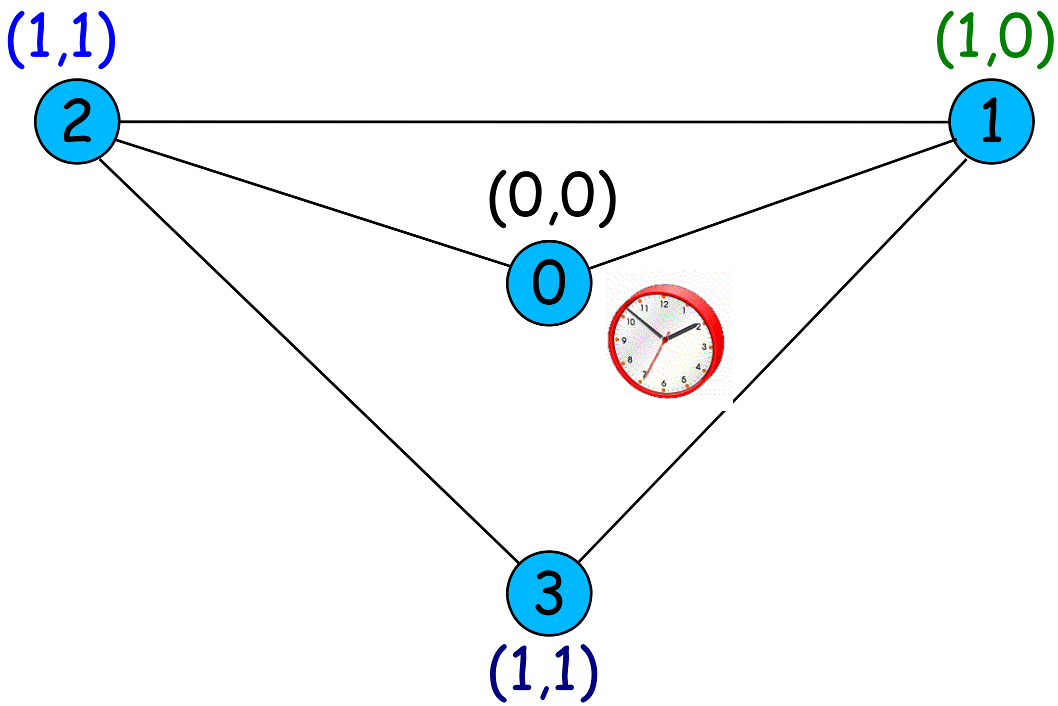
Forwarding



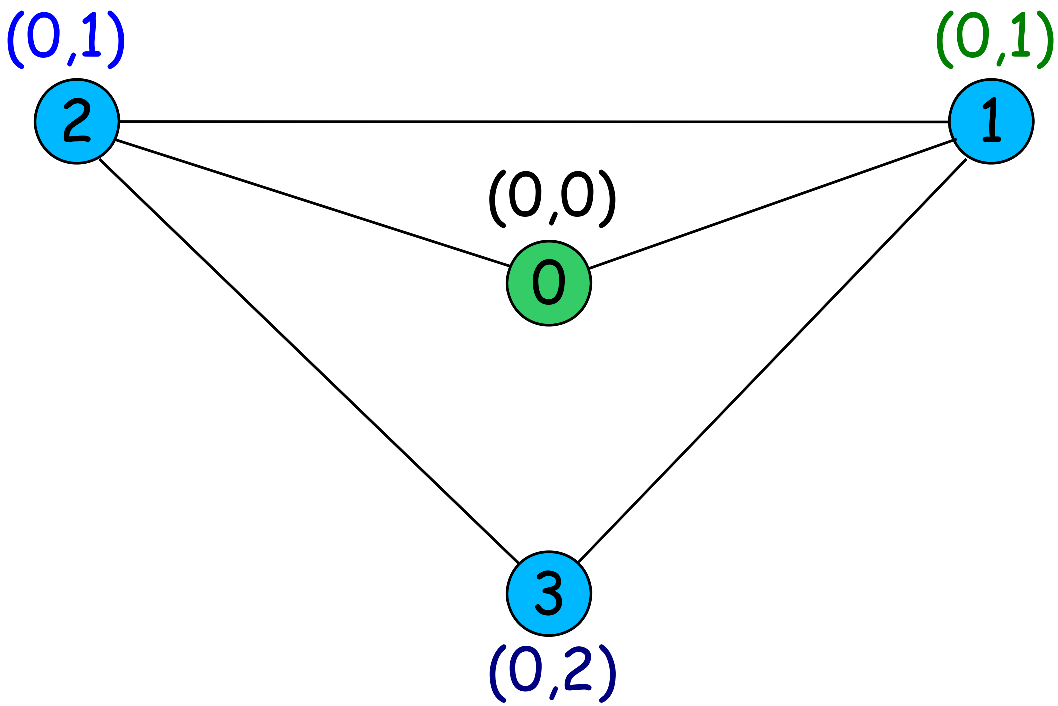
Leader Election



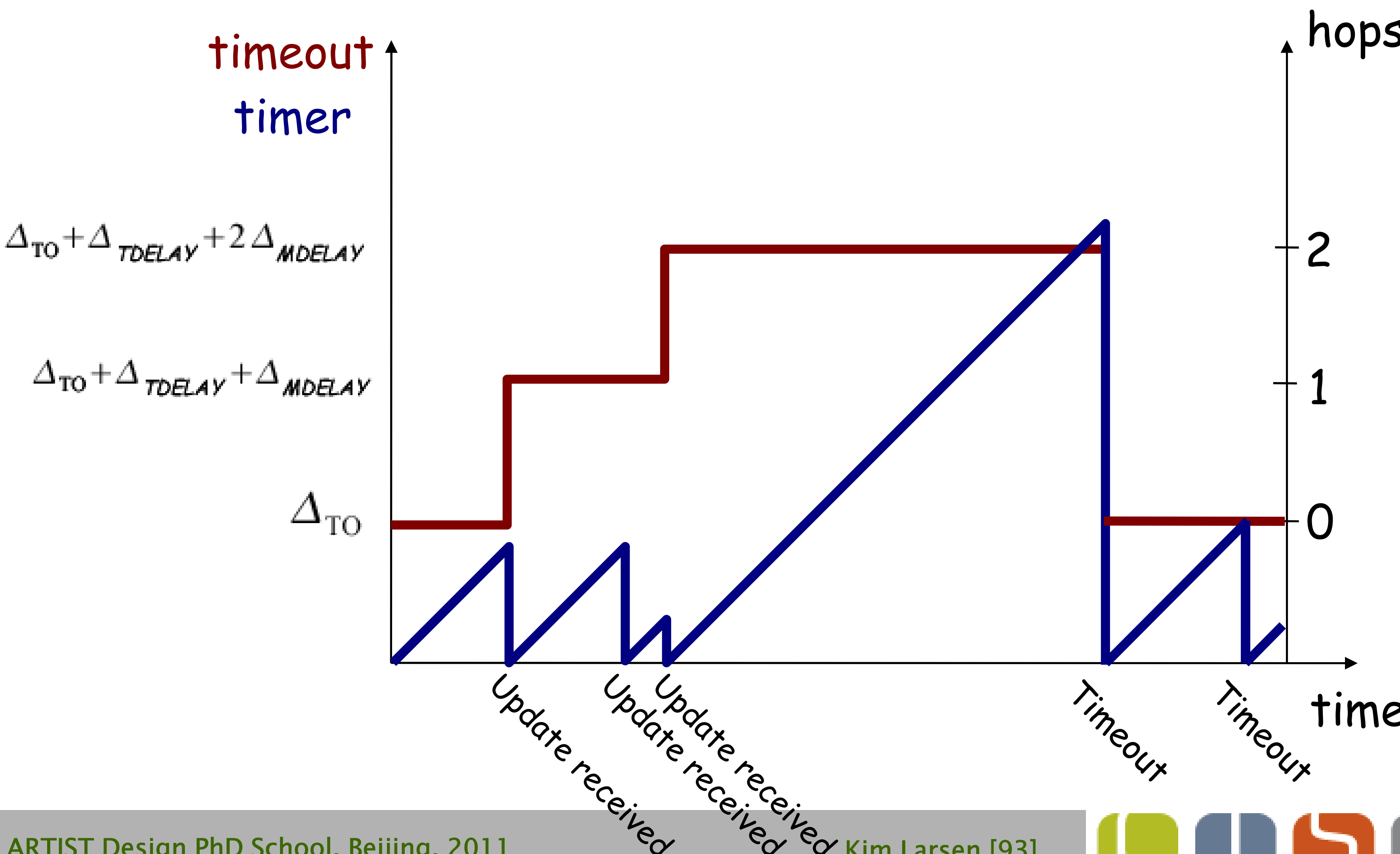
Leader Election



Leader Election



Variable timeout



Leader Election

Claim to be verified

Correct leader is known at a node *i* after

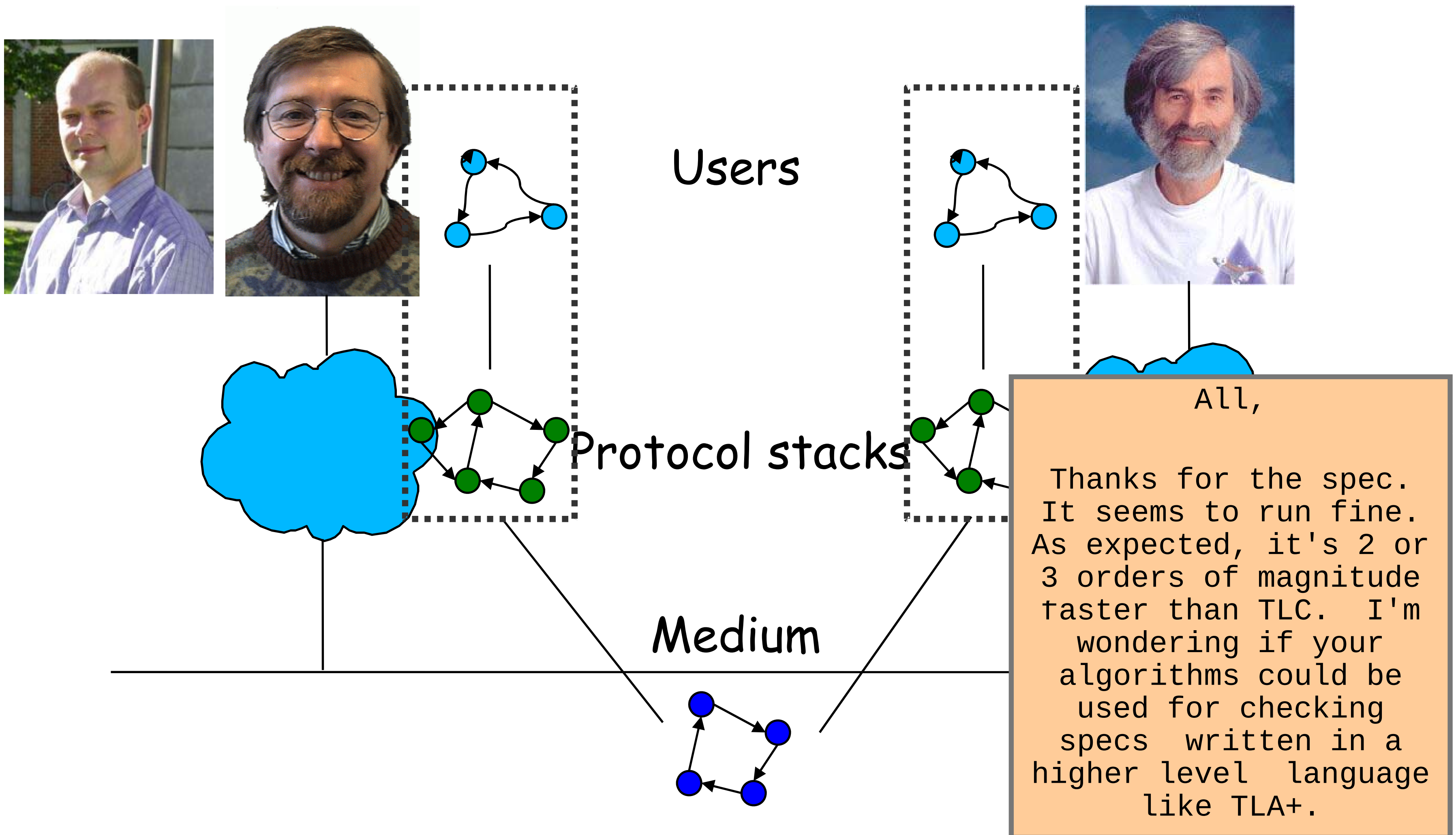
$$t(i) = \Delta_{TO} + \Delta_{TDELAY} + d_i \cdot \Delta_{MDELAY}$$

A model checking problem

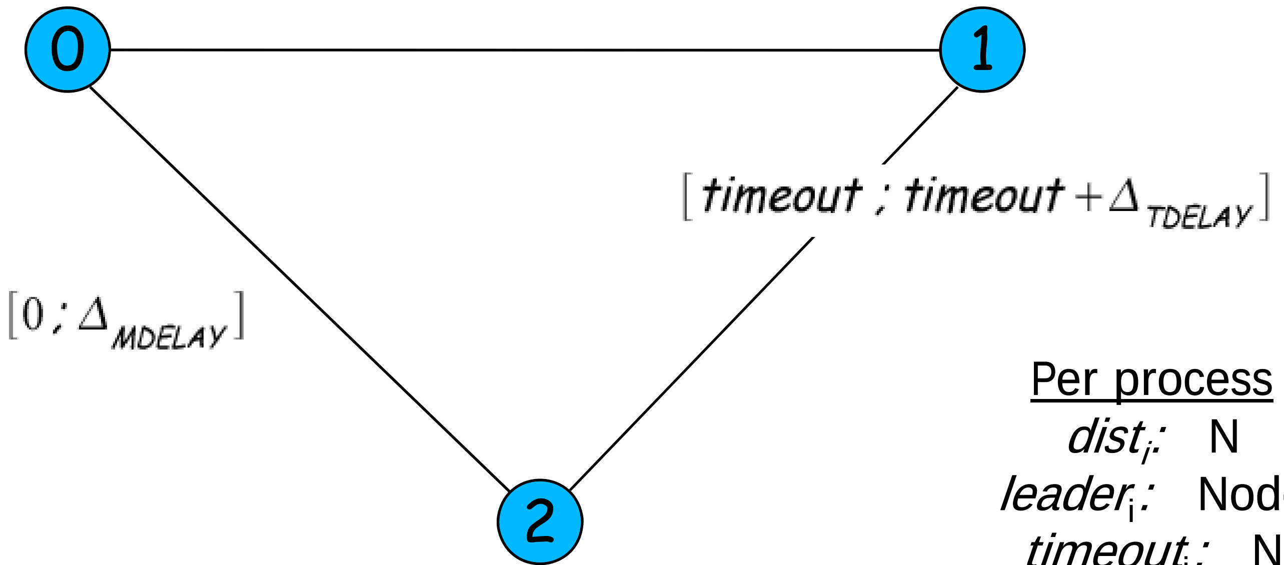
$$IMP \models \Box_{>t(i)}\ l(i)=L(i)$$

for all i.

Modelling (RT) protocols



Modelling the election protocol



<u>Per process</u>

$dist_i$: N
$leader_i$: Node
$timeout_i$: N

<u>Static</u>

$Topology : Node \times Node \rightarrow B$

<u>Message</u>

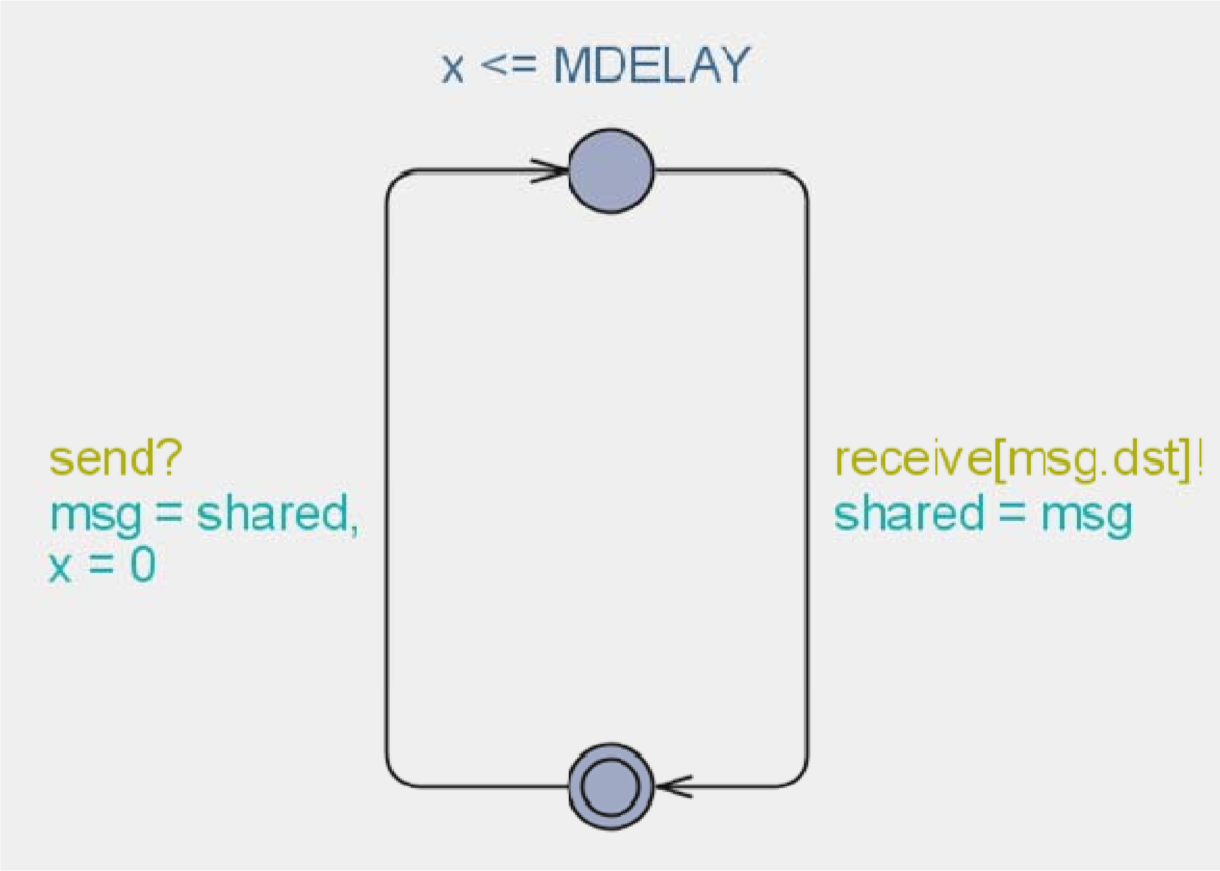
src: Node
dst: Node
leader: Node
hopss: N

Global Declaration

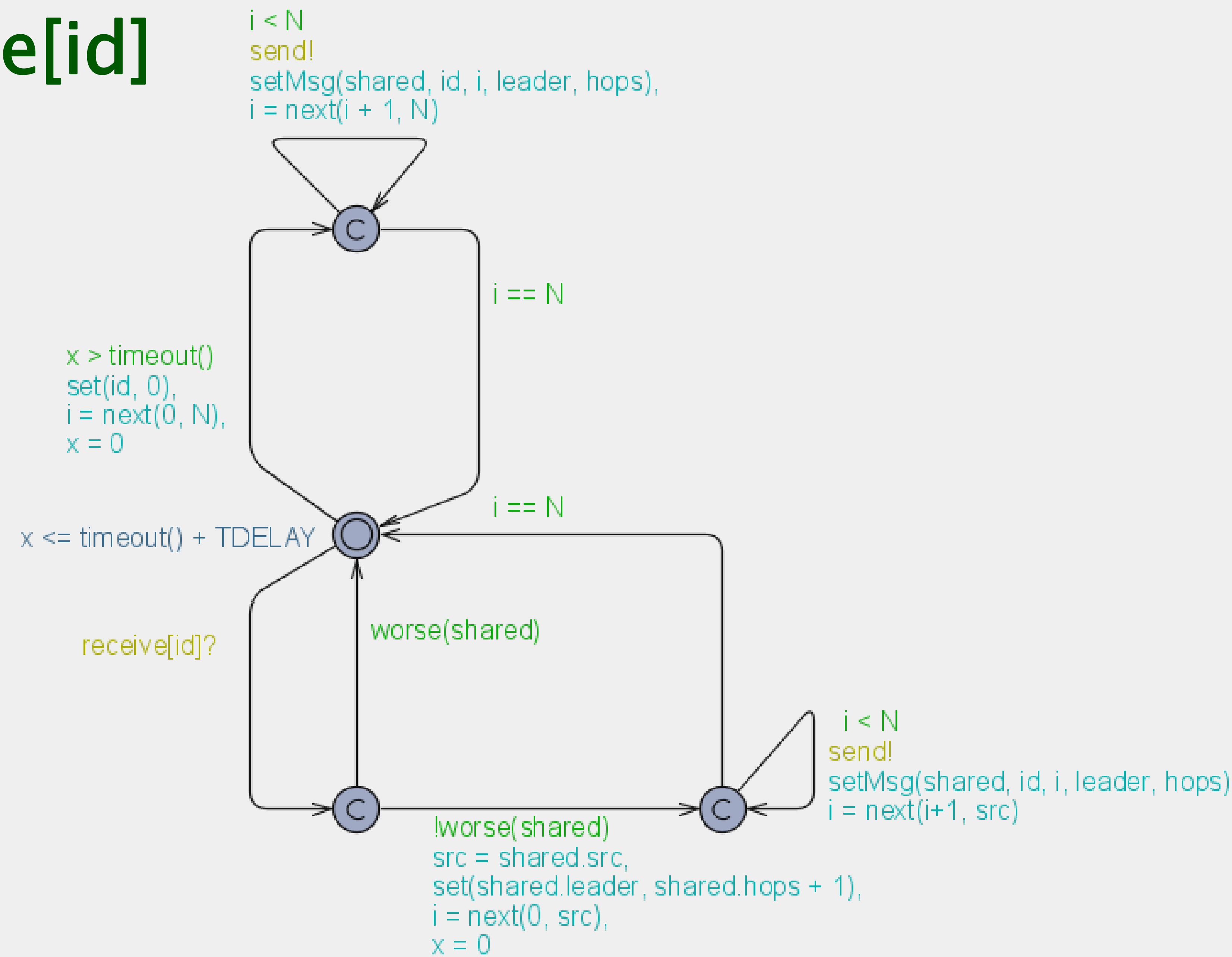
```
void setMsg(msg_t &msg, id_t src, id_t dst, id_t leader, int[0,N] hops)
{
  msg.src = src;
  msg.dst = dst;
  msg.leader = leader;
  msg.hops = hops;
}

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]



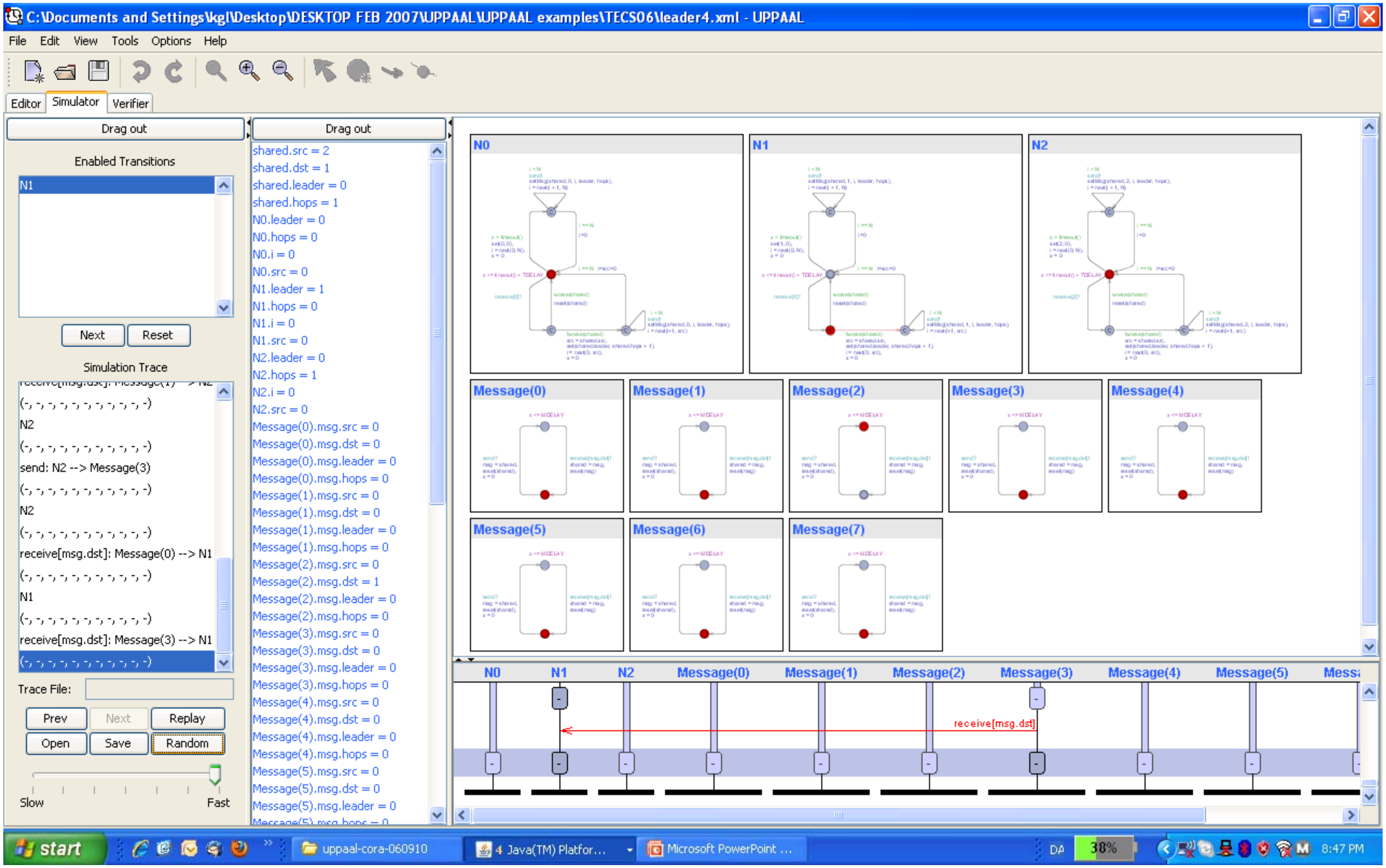
i < N
send!
setMsg(shared, id, i, leader, hops),
i = next(i + 1, N)
i == N
x > timeout()
set(id, 0),
i = next(0, N),
x = 0
i == N
x <= timeout() + TDELAY
receive[id]?
worse(shared)
i < N
send!
setMsg(shared, id, i, leader, hops)
i = next(i+1, src)
!worse(shared)
src = shared.src,
set(shared.leader, shared.hops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