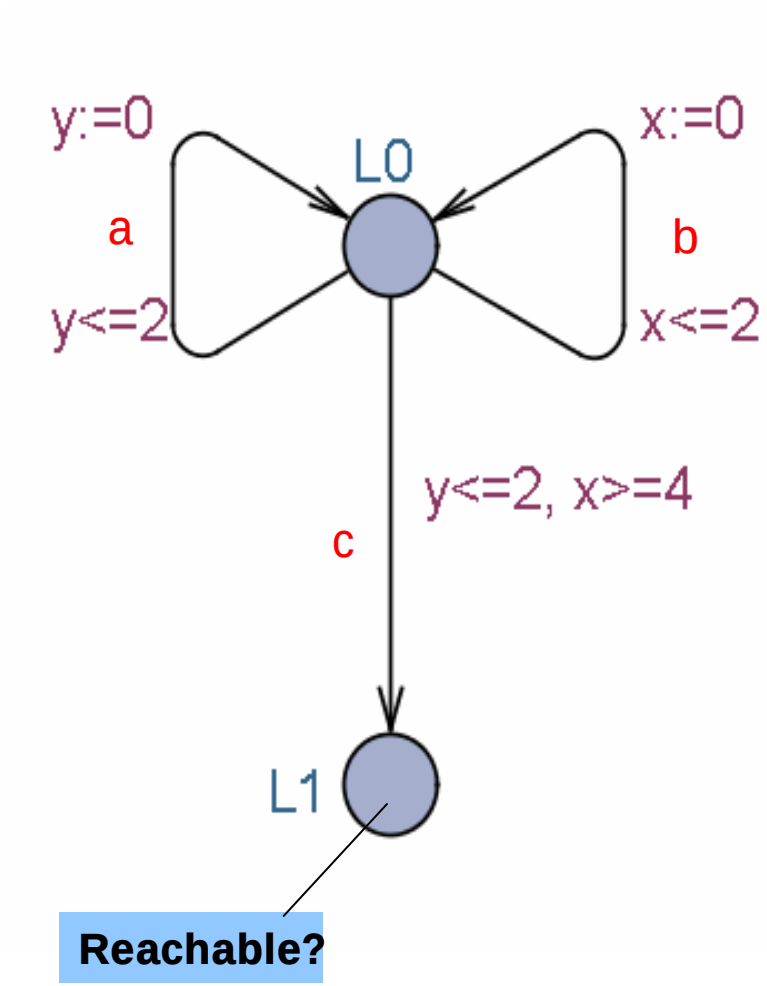
Timed Automata – Decidability Results

BRICS
Basic Research
in Computer Science

CSS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Decidability ?



OBSTACLE: Uncountably infinite state space

Derived Relations and Reachability

$$(l,u) \xrightarrow{\delta} (l',u') \quad \text{iff} \quad \exists d > 0.\, (l,u) \xrightarrow{\epsilon(d)} (l',u').$$

$$(l,u) \xrightarrow{\alpha} (l',u') \quad \text{iff} \quad \exists a \in Act.\, (l,u) \xrightarrow{a} (l',u')$$

$$(l,u) \rightsquigarrow (l',u') \quad \text{iff} \quad (l,u)(\xrightarrow{\delta} \cup \xrightarrow{\alpha})^*(l',u')$$

Definition

The set of reachable locations, $Reach(A)$, of a timed automaton A is defined as:

$$l \in Reach(A) \equiv^{\triangle} \exists u.\, (l_0,u_0) \rightsquigarrow (l,u)$$

Time Abstracted Bisimulation

Definition

Let $G \subseteq L$ be a set of goal locations. An equivalence relation R on $L \times \mathbb{R}^C$ is a TAB wrt G if whenever $(l,u)R(n,v)$ the following holds:

1. $l \in G$ iff $n \in G$,
2. whenever $(l,u) \xrightarrow{\delta} (l',u')$ then $(n,v) \xrightarrow{\delta} (n',v')$ with $(l',u')R(n',v')$
3. whenever $(l,u) \xrightarrow{a} (l',u')$ then $(n,v) \xrightarrow{a} (n',v')$ with $(l',u')R(n',v')$

Stable Quotient

Definition

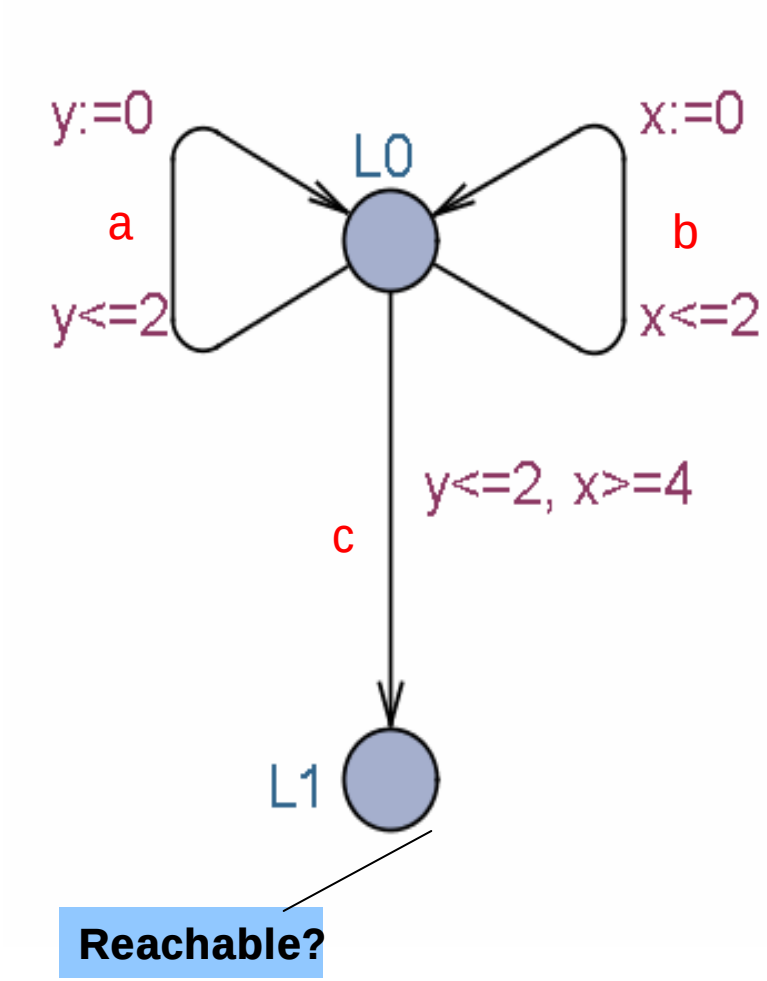
Let R be a TAB wrt G. The induced *quotient* has classes of R, $\pi \in (L \times \mathbb{R}^C/R)$, as states. For classes π, π' the transitions are

- $\pi \xrightarrow{\delta} \pi'$ iff $(l,u) \xrightarrow{\delta} (l',u')$ for some $(l,u) \in \pi$, $(l',u') \in \pi'$.
- $\pi \xrightarrow{a} \pi'$ iff $(l,u) \xrightarrow{a} (l',u')$ for some $(l,u) \in \pi$, $(l',u') \in \pi'$.

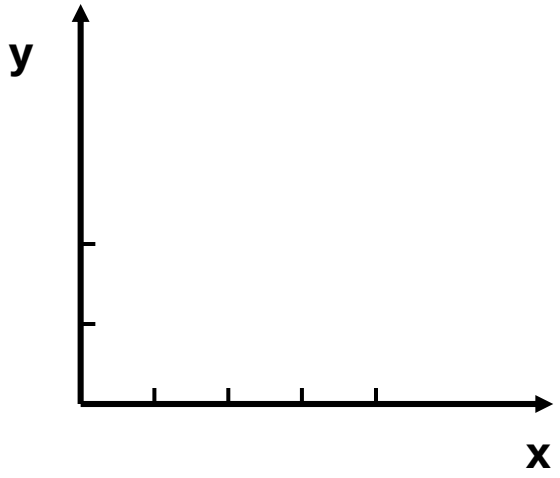
Theorem

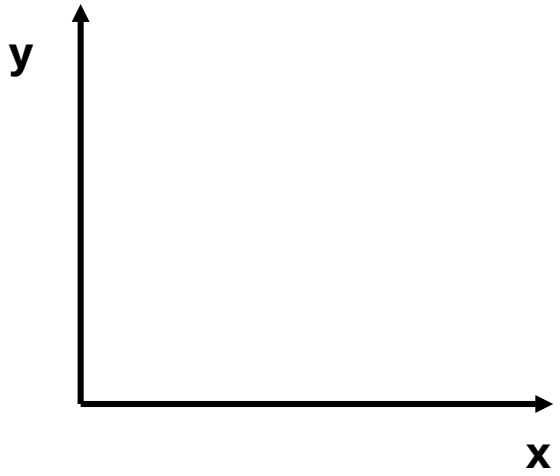
Let R be TAB wrt G. Then, a location from G is reachable iff there exists an equivalence class π of R such that π is reachable in the quotient and π contains a state whose location is in G.

Stable Quotient

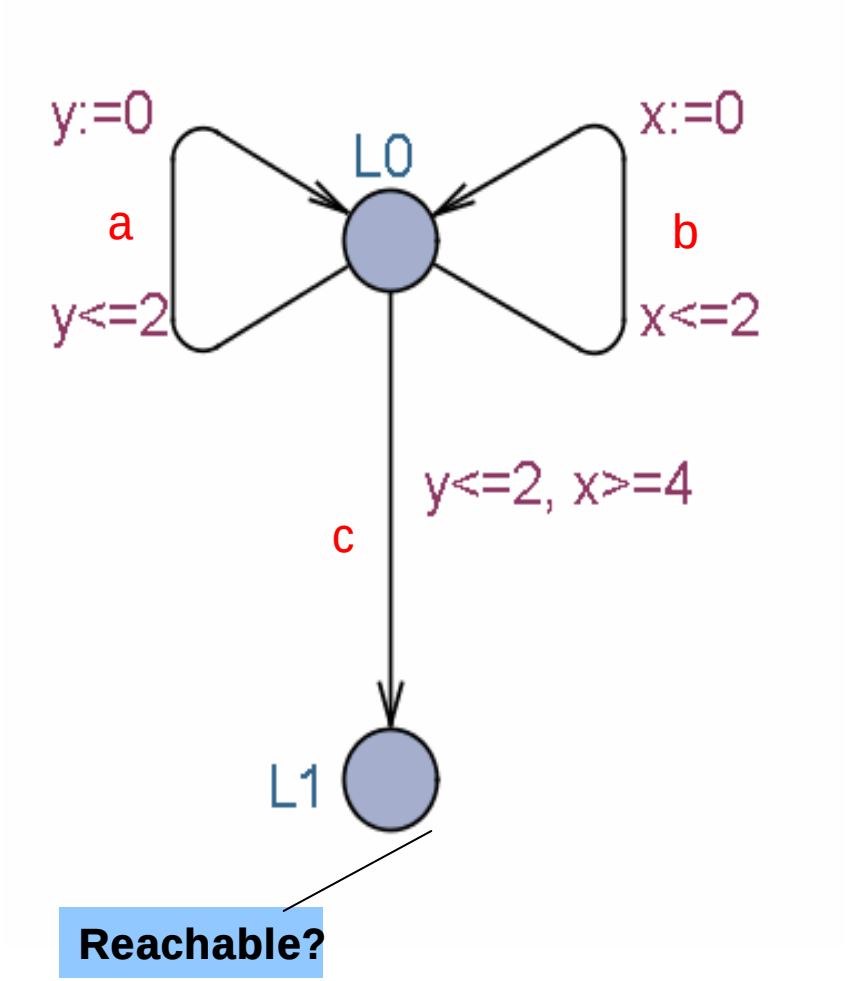


Partitioning

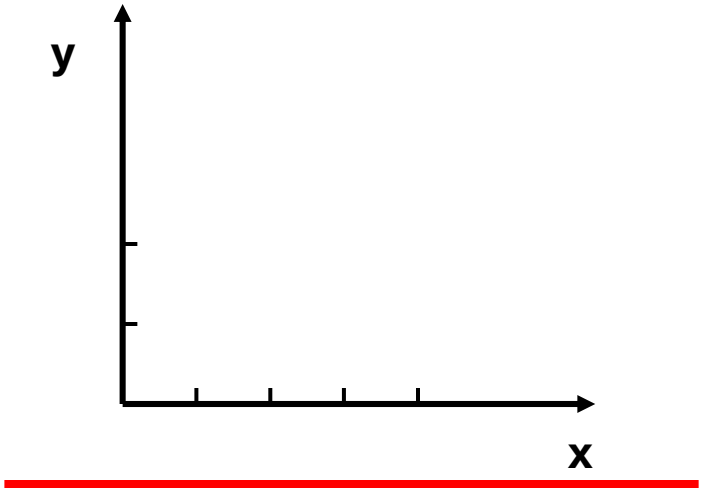


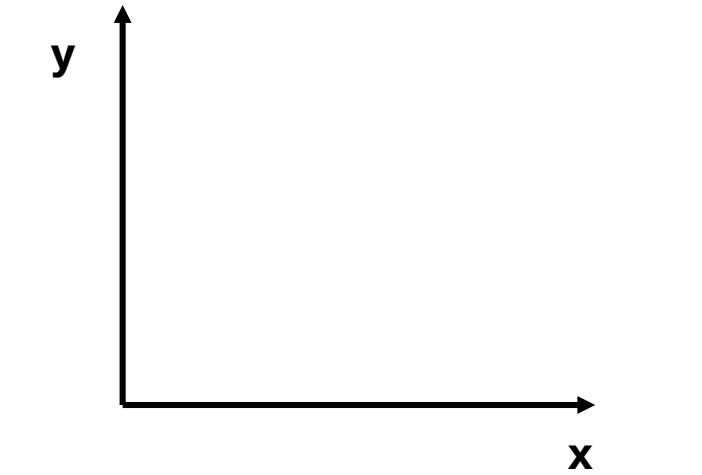


Stable Quotient

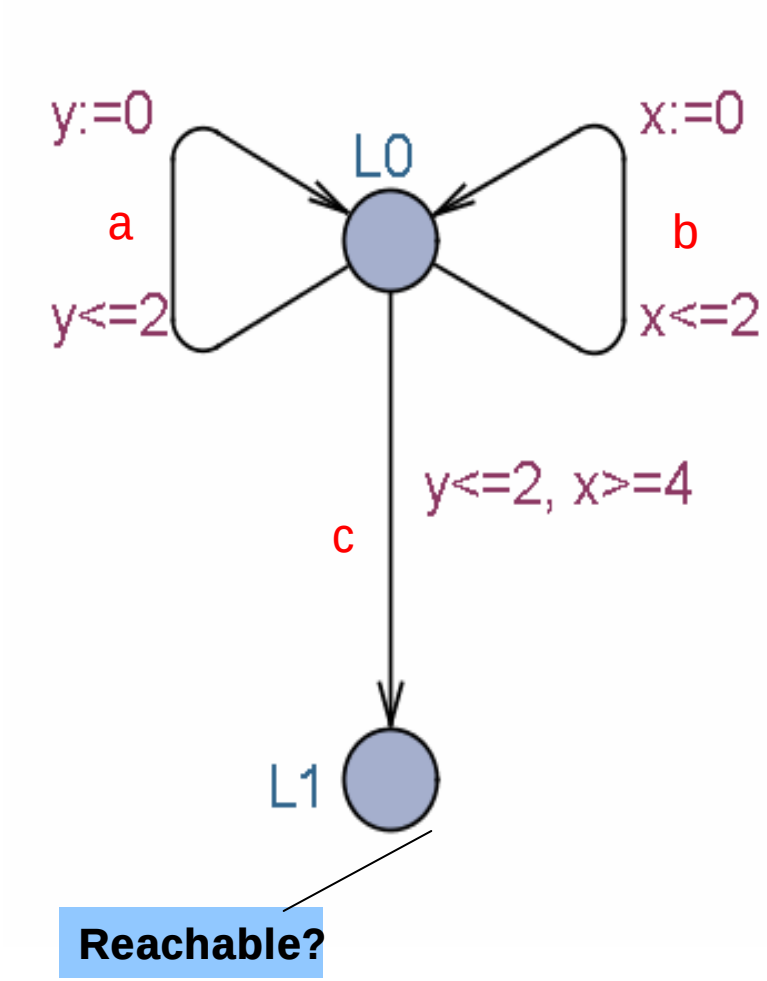


Partitioning

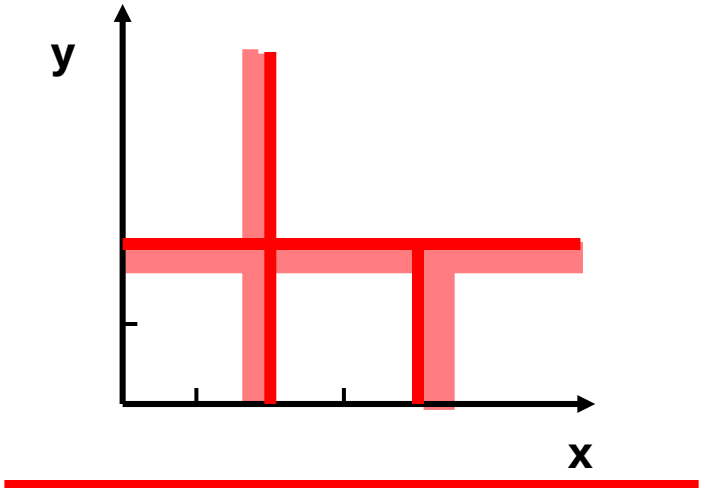


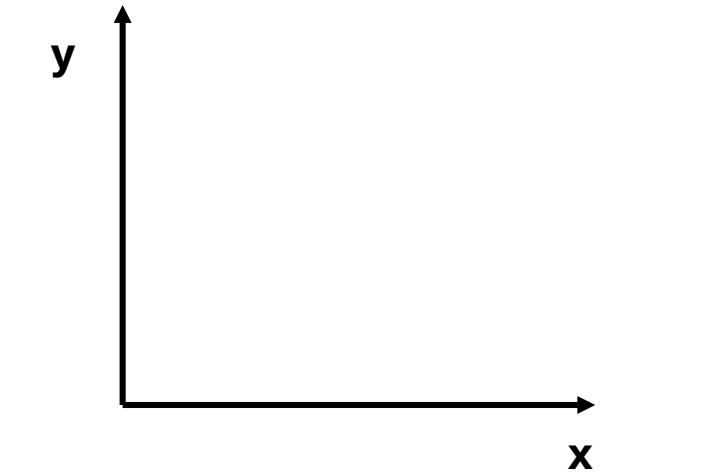


Stable Quotient

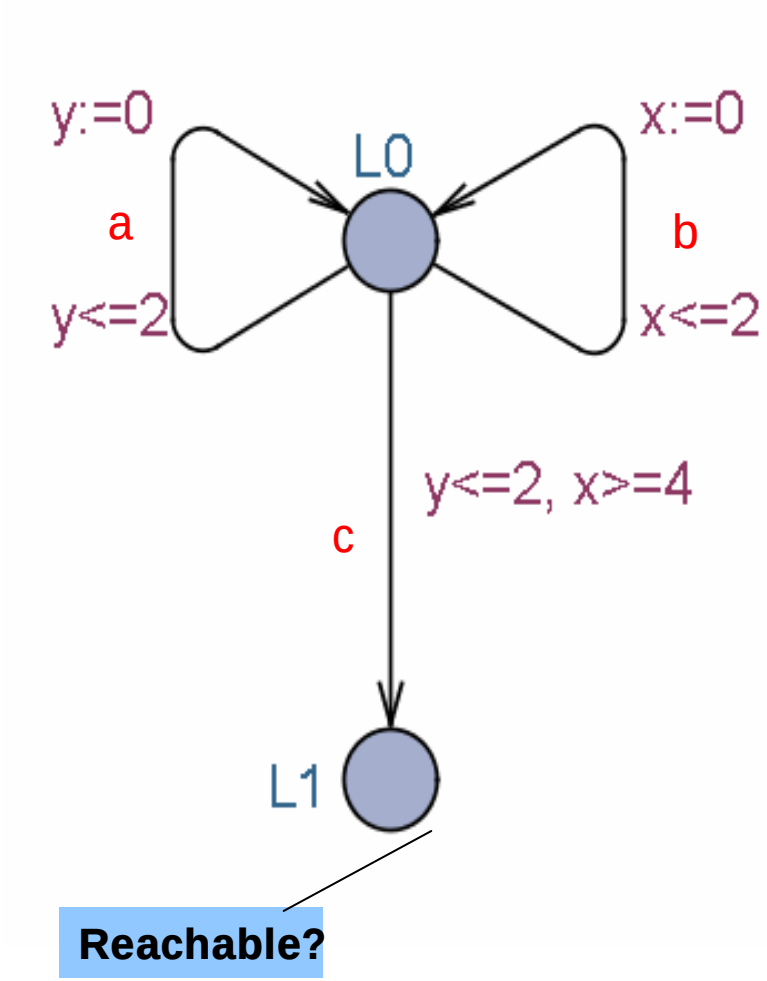


Partitioning

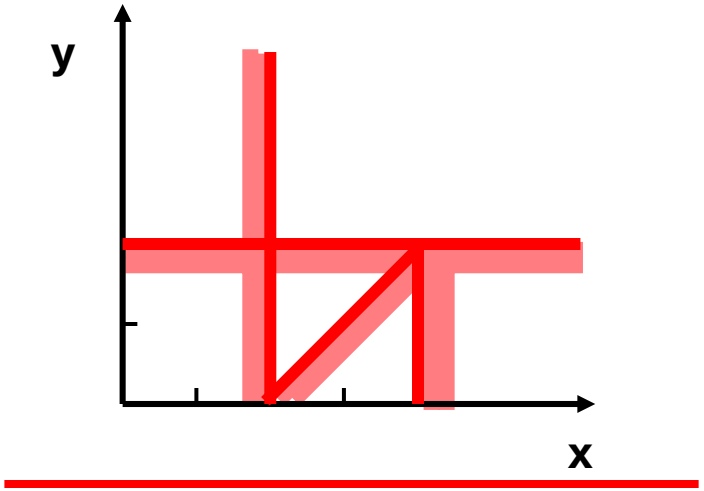


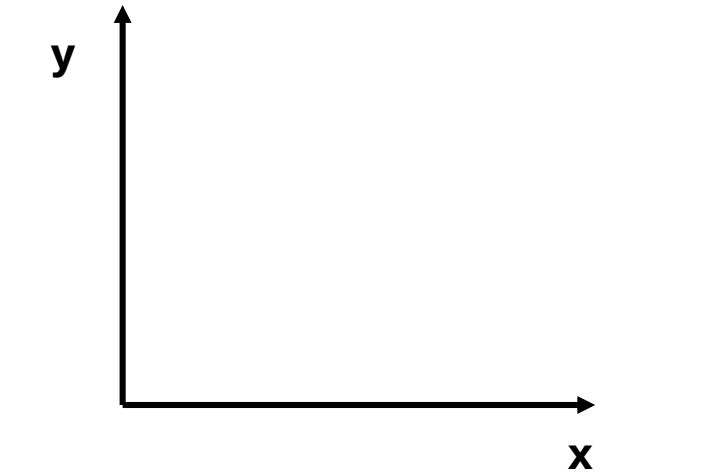


Stable Quotient

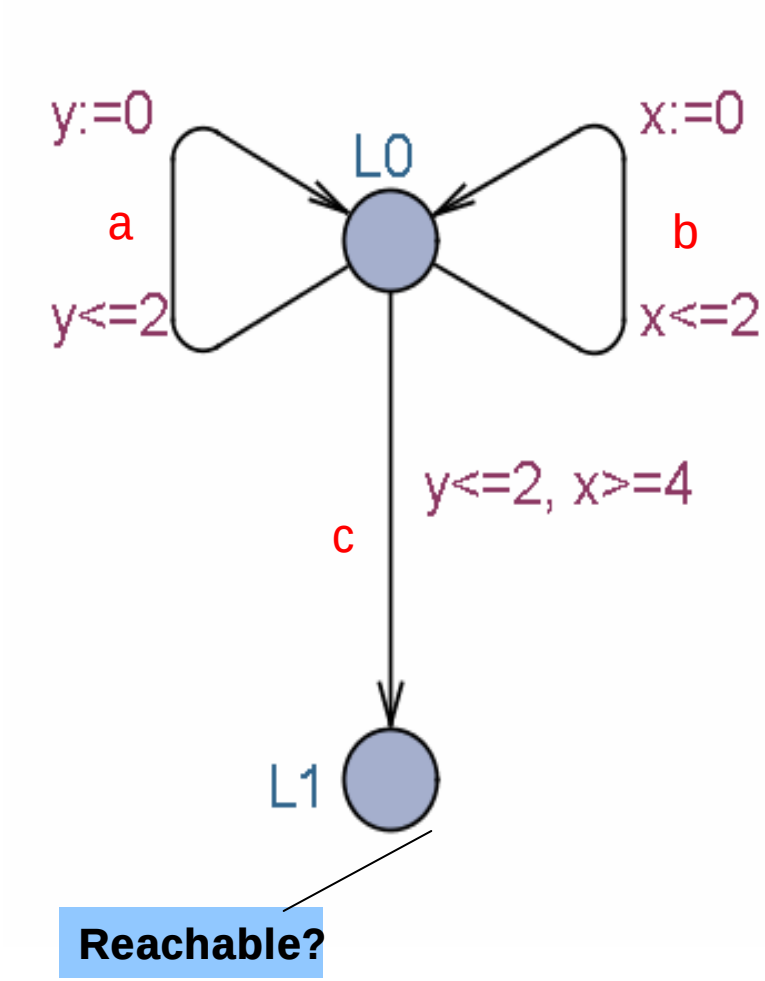


Partitioning

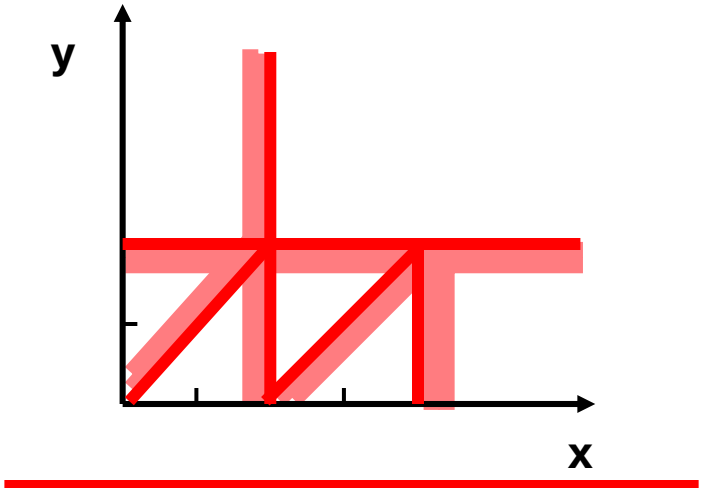


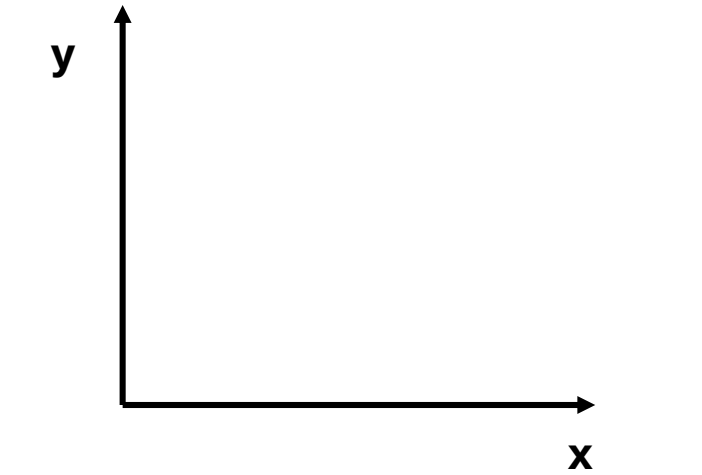


Stable Quotient

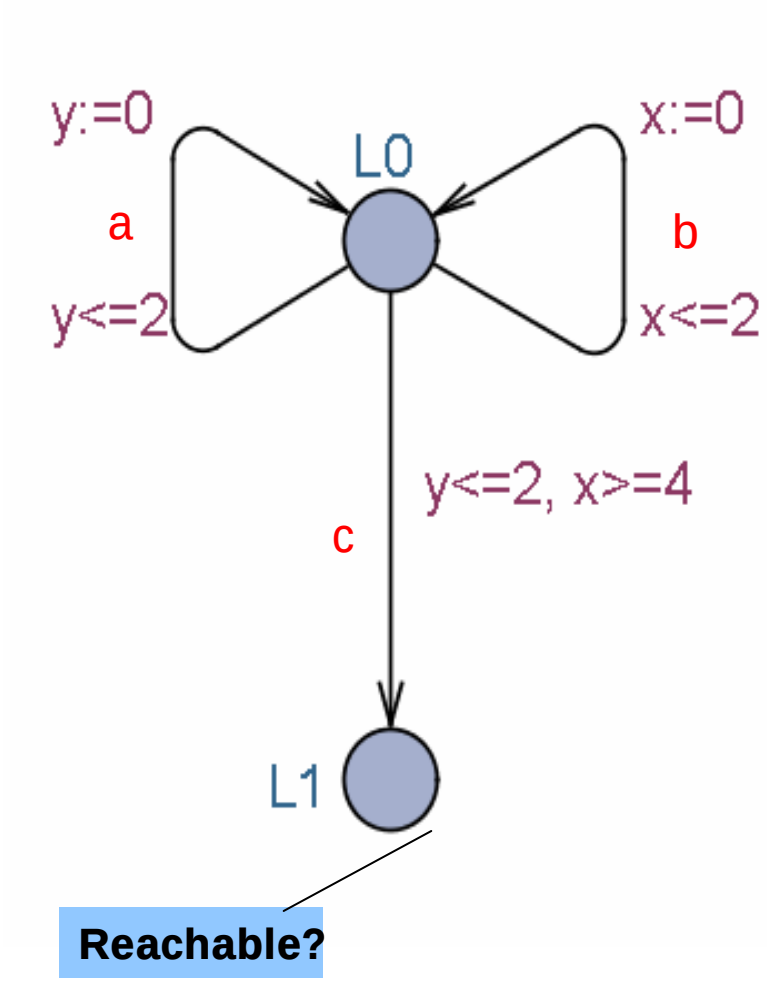


Partitioning

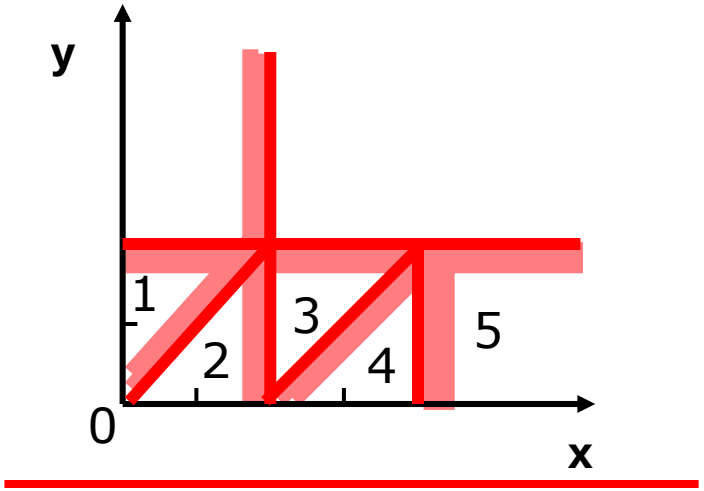


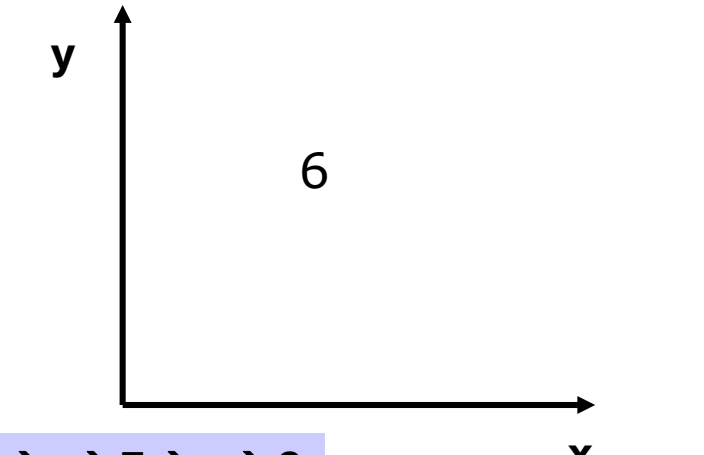


Stable Quotient



Partitioning





0 →ε→1→a→2→ε→3→a→4→ε→5→c→6

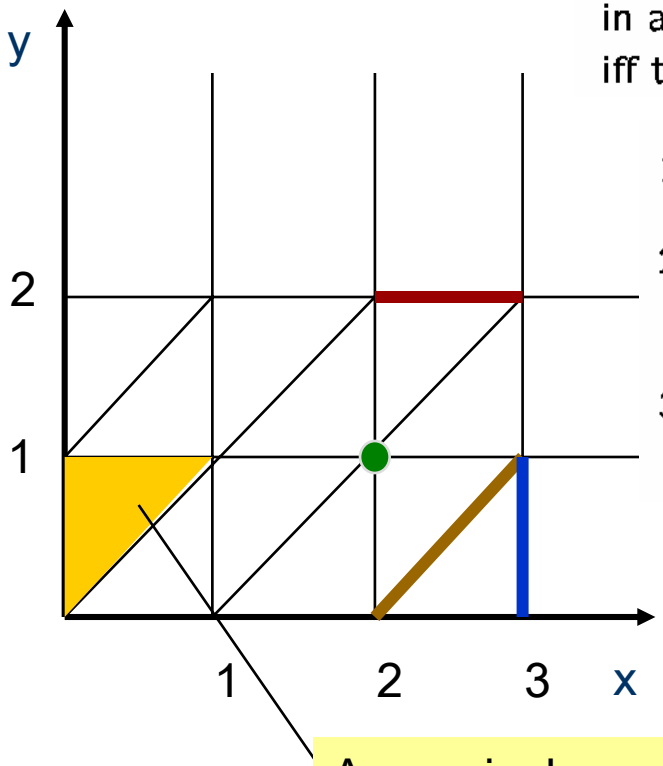
Region Equivalence

For each clock x let c_x be the largest integer with which x is compared in any guard or invariant of A. u and u' are *region equivalent*, $u \cong u'$ iff the following holds:

1. For all $x \in C$, either $\lfloor u(x) \rfloor = \lfloor u'(x) \rfloor$ or $u(x), u'(x) > c_x$;
2. For all $x, y \in C$ with $u(x) \leq c_x$ and $u(y) \leq c_y$, $fr(u(x)) \leq fr(u(y))$ iff $fr(u'(x)) \leq fr(u'(y))$;
3. For all $x \in C$ with $u(x) \leq c_x$, $fr(u(x)) = 0$ iff $fr(u'(x)) = 0$.

Regions

Finite Partitioning of State Space



For each clock x let c_x be the largest integer with which x is compared in any guard or invariant of A. u and u' are *region equivalent*, $u \cong u'$ iff the following holds:

1. For all $x \in C$, either $\lfloor u(x) \rfloor = \lfloor u'(x) \rfloor$ or $u(x), u'(x) > c_x$;
2. For all $x, y \in C$ with $u(x) \leq c_x$ and $u(y) \leq c_y$, $fr(u(x)) \leq fr(u(y))$ iff $fr(u'(x)) \leq fr(u'(y))$;
3. For all $x \in C$ with $u(x) \leq c_x$, $fr(u(x)) = 0$ iff $fr(u'(x)) = 0$.

An equivalence class (i.e. a *region*)
in fact there is only a *finite* number of regions!!

Logical Characterization of Regions

Each region may be represented by specifying

1. for every clock x a constraint from

$$\{x = c \mid c = 0, 1, \ldots, c_x\} \cup \{c - 1 < x < c \mid c = 1, \ldots, c_x\} \cup \{x > c_x\}$$

2. for every pair of clocks x, y such that $c-1 < x < c$ and $d-1 < y < d$ appears in 1., whether $fr(x)$ is $<$, $=$ or $>$ than $fr(y)$.

Theorem

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

Stability of Regions

Lemma

1. If $u \cong u'$ then $\forall d. \exists d'. u + d \cong u' + d'$;
2. If $u \cong u'$ then* $\forall g \in \mathcal{B}(X). u \models g \Leftrightarrow u' \models g$;
3. If $u \cong u'$ then $\forall r \subseteq C. u[r] \cong u'[r]$.

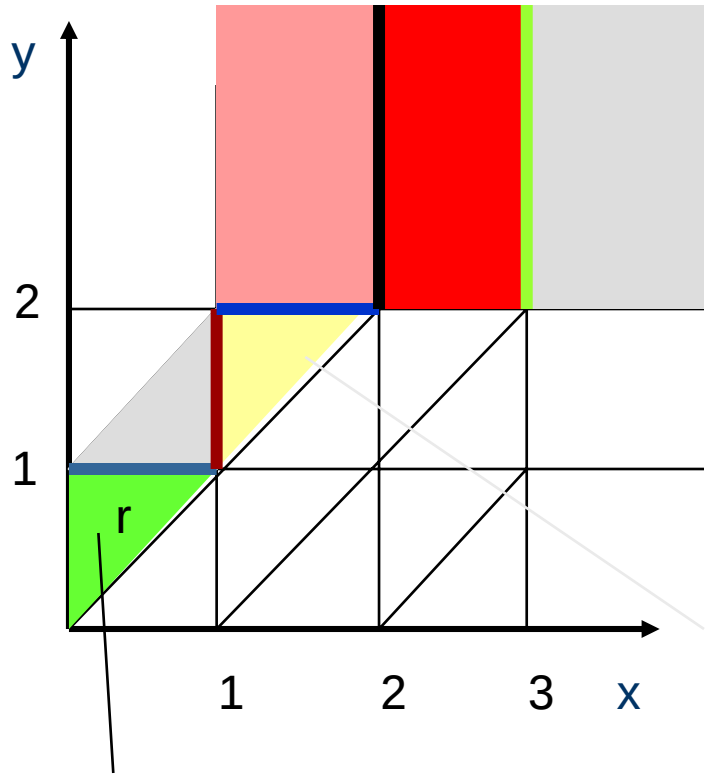
Theorem

Let $(l, u) \cong (l', u')$ iff $l = l'$ and $u \cong u'$. Then $\cong$ is a TAB with respect to any set of goal locations G.

*Here $\cong$ and g should agree on the maximal constants.

Regions

Successor Operation (wrt delay)



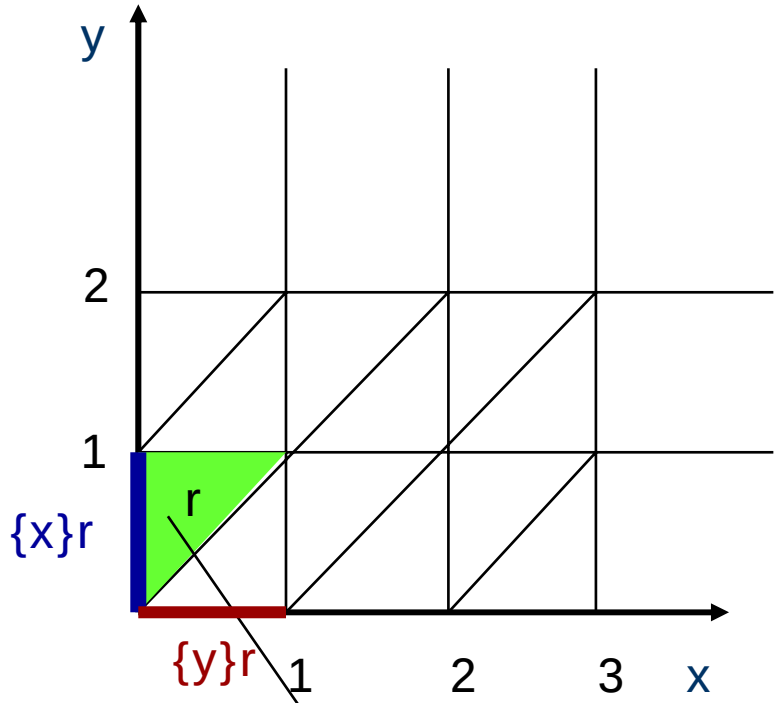
Successor regions, Succ(r)

An equivalence class (i.e. a *region*)

BRICS
Basic Research
in Computer Science

Regions

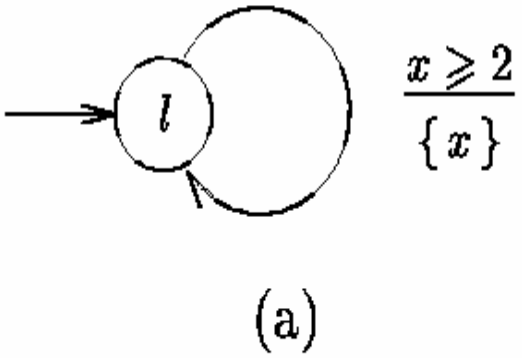
Reset Operation

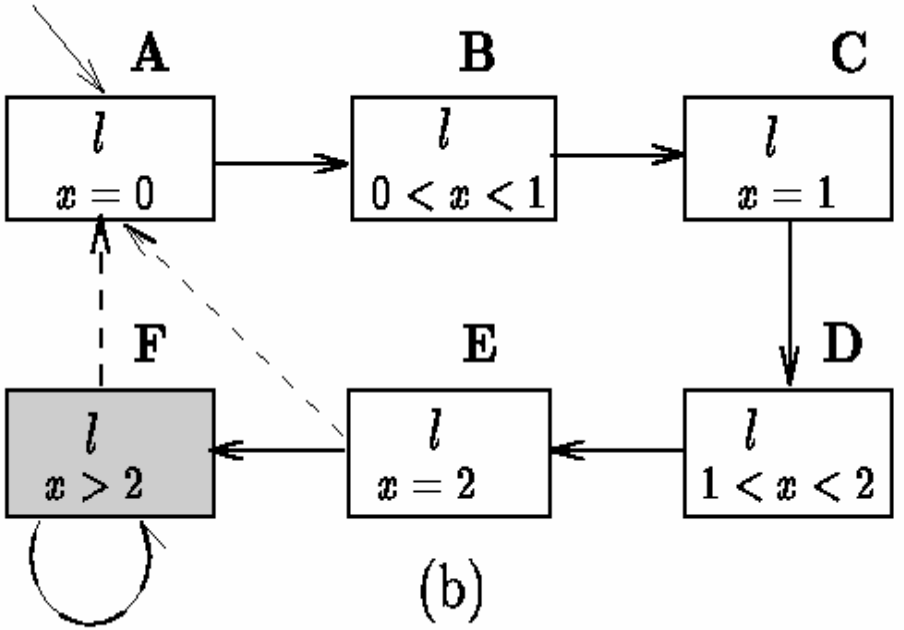


Reset regions

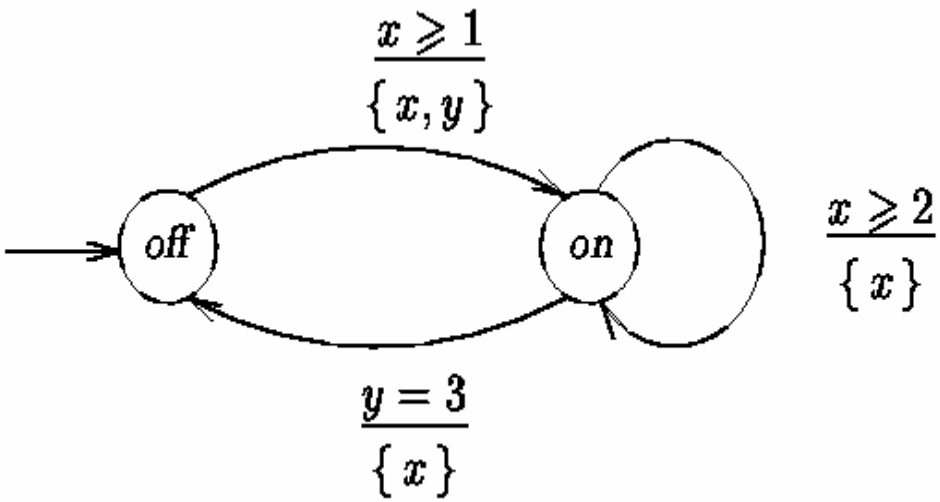
An equivalence class (i.e. a *region*) r

An Example Region Graph





Modified light switch



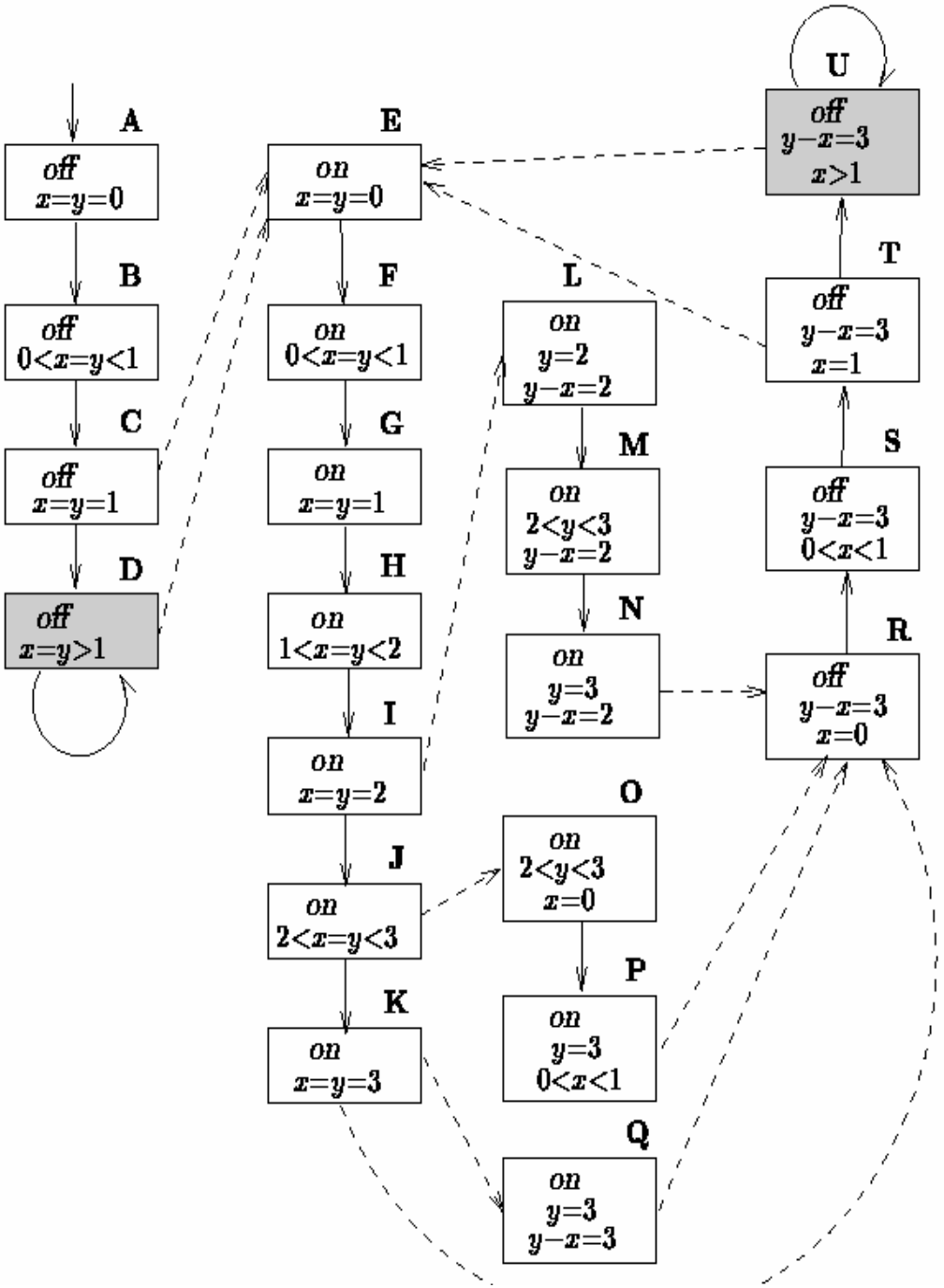
$inv(off) = \text{true}$

$inv(on) = y \leqslant 3$

$AG(x \leq y)$

$AG(on \Rightarrow AF off)$

$AG(on \Rightarrow AF_{\leq 9} off)$



Reachable part of region graph

Properties

$\mathrm{AG}(x \le y)$

$\mathrm{AG}(on \Rightarrow \mathrm{AF}off)$

$\mathrm{AG}(on \Rightarrow \mathrm{AF}_{\le 9}off)$

Finite Representation of Regions

Let Γ be a triple of the form:

$$\Gamma = \left\langle C_{>}, \mathcal{H}, [C_0, C_1, \ldots, C_k] \right\rangle$$

where

- $C_{>}, C_0, C_1, \cdots, C_k$ is a partitioning of C ($C_{>}$ and C_0 are allowed to be empty).
- $\mathcal{H} : (C \backslash C_{>}) \longrightarrow \mathbb{N}$ with $\mathcal{H}(x) \leq c_x$.

Representation of Regions (cont)

$u \in [\![\Gamma]\!]$ if and only if

- $\forall x \in C_{>}. u(x) > c_x$;
- $\forall x \in C \backslash C_{>}. \lfloor u(x) \rfloor = \mathcal{H}(x)$;
- $\forall x \in C_0. fr(u(x)) = 0$;
- $\forall x, y \in C_i. fr(u(x)) = fr(u(y))$;
- $\forall x \in C_i . y \in C_j. fr(u(x)) < fr(u(y)) \Leftrightarrow i < j$;

$[\![\Gamma]\!]$ is a region, and any region is representable as a triplet.

Decidability & Complexity

Theorem

The transition relations $\xrightarrow{\delta}$ and $\xrightarrow{a}$ between regions may be computed effectively using the triplet or the logical characterization of regions.

Theorem

Let A be a timed automaton with location l.
It is decidable whether $l \in Reach(A)$.

Theorem

Let A be a timed automaton with location l.
Deciding $l \in Reach(A)$ is PSPACE-complete.

Fundamental Results

- Reachability ☺ **Alur, Dill**
- Trace-inclusion **Alur, Dill**
  - Timed ☹ ; Untimed ☺
- Bisimulation
  - Timed ☺ **Cerans** ; Untimed ☺
- Model-checking ☺
  - TCTL, T_{mu}, L_{nu},...

Updatable Timed Automata

Patricia Bouyer, Catherine Dufourd,
Emmanuel Fleury, Antoine Petit

	Diagonal-free	W Diagonals
$x := c,\ x := y$	Pspace complete	Pspace complete
$x := x + 1$	Pspace complete	Undecidable
$x := y + c$		
$x := x - 1$	Undecidable	
$x :< c,\ x :\leq c$	Pspace complete	Pspace complete
$x :> c,\ x :\geq c$	Pspace complete	Undecidable
$x :\sim y + c$		
$(y+)c <: x :< (y+)d$		
$y + c <: x :< z + d$	Undecidable	

With $\sim \in \{<, \leq, \geq, >\}$ and $c, d \in \mathbb{Q}_+$

	Diagonal-free	W Diagonals
$:= c,\ x := y$	TA-bisimilar	TA-bisimilar
$x := x + 1$	TA-bisimilar	Turing
$x := y + c$		
$:< c,\ x :\leq c$	TA_ϵ	TA_ϵ
$x :> c,\ x :\geq c$	TA_ϵ	Turing
$x :\sim y + c$	TA_ϵ	
$(y+)c <: x :< (y+)d$	TA_ϵ	

With $\sim \in \{<, \leq, \geq, >\}$ and $c, d \in \mathbb{Q}_+$

BRICS
Basic Research
in Computer Science

TCTL:
Timed Computational Tree Logic

BRICS
Basic Research
in Computer Science

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

TCTL = CTL + Time

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

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

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

α – *constraints over formula clocks and automata clock*

z *in* ϕ – *"freeze operator" introduces new formula clock* z

E[ϕ U ϕ], A[ϕ U ϕ] - *like in CTL*

No EX ϕ

Derived Operators

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

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

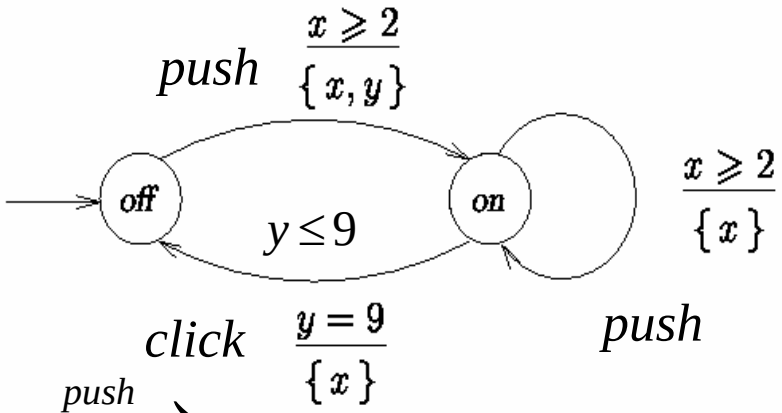
$$\boxed{\mathsf{EF}_{<5}\,\phi} \quad = \quad z \text{ in } \mathsf{EF}(z < 5 \wedge \phi)$$

The property ϕ becomes valid within 5 time units.

Paths

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

Example:



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

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

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

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

Elapsed time in path

$$\Delta(\sigma,0) = 0$$

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

Example:

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

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

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

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

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

BRICS
Basic Research
in Computer Science

TCTL Semantics

$$
\begin{array}{ll}
s, w \models p & \text{iff } p \in Label(s) \\
s, w \models \alpha & \text{iff } v \cup w \models \alpha \\
s, w \models \neg \phi & \text{iff } \neg (s, w \models \phi) \\
s, w \models \phi \vee \psi & \text{iff } (s, w \models \phi) \vee (s, w \models \psi) \\
s, w \models z \text{ in } \phi & \text{iff } s, \textbf{reset } z \text{ in } w \models \phi \\
s, w \models \mathsf{E}[\phi \mathsf{U} \psi] & \text{iff } \exists \sigma \in P^{\infty}_{\mathcal{M}}(s). \exists (i,d) \in Pos(\sigma). \\
 & \quad (\sigma(i,d), w{+}\Delta(\sigma,i) \models \psi \wedge \\
 & \quad (\forall (j,d') \ll (i,d). \sigma(j,d'), w{+}\Delta(\sigma,j) \models \phi \vee \psi)) \\
s, w \models \mathsf{A}[\phi \mathsf{U} \psi] & \text{iff } \forall \sigma \in P^{\infty}_{\mathcal{M}}(s). \exists (i,d) \in Pos(\sigma). \\
 & \quad ((\sigma(i,d), w{+}\Delta(\sigma,i)) \models \psi \wedge \\
 & \quad (\forall (j,d') \ll (i,d). (\sigma(j,d'), w{+}\Delta(\sigma,j)) \models \phi \vee \psi)).
\end{array}
$$

s - location

w - formula clock valuation

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

$Pos(\sigma)$ - positions in σ

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

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

BRICS
Basic Research
in Computer Science

Timeliness Properties

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

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

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

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

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

putbox occurs periodically (exactly) every 25 time units

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

The UPPAAL Verification Engine

BRICS
Basic Research
in Computer Science

Overview

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

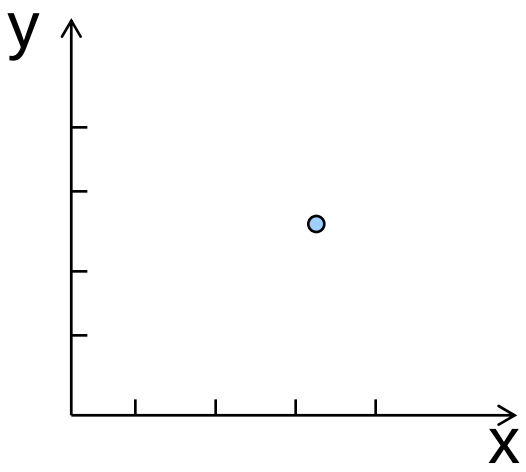
- Distributed UPPAAL [CAV2000, STTT2004]
- Unification & Sharing [FTRTFT2002, SPIN2003]
- Acceleration [FORMATS2002]
- Static Guard Analysis [TACAS2003,TACAS2004]
- Storage-Strategies [CAV2003]

Zones

From infinite to finite

State

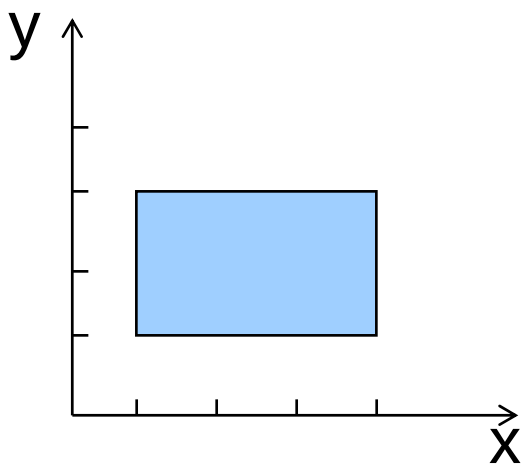
(n, x=3.2, y=2.5)



Symbolic state (set)

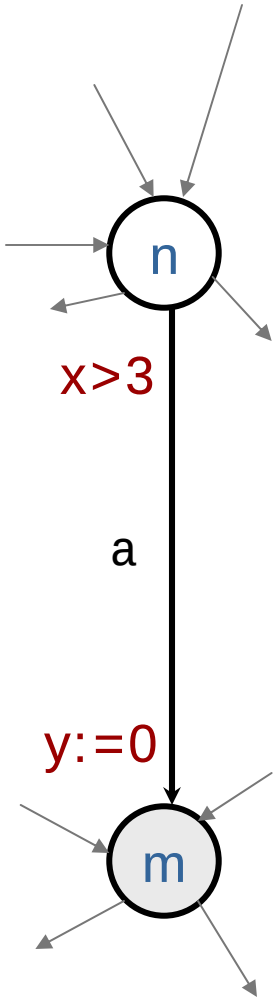
(n, 1≤x≤4, 1≤y≤ 3)

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

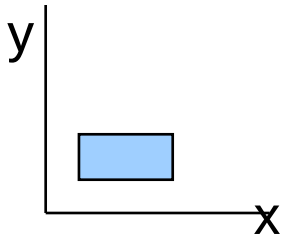


BRICS
Basic Research
in Computer Science

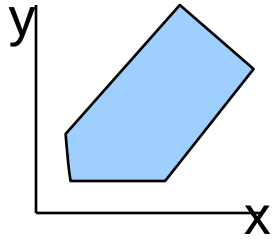
Symbolic Transitions



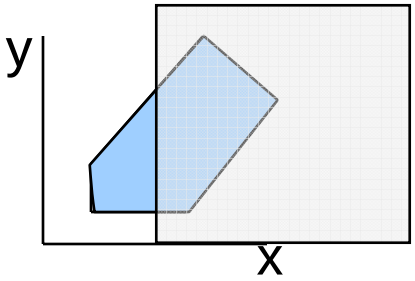
1<=x<=4
1<=y<=3



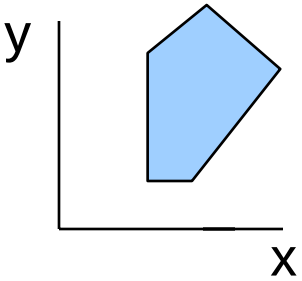
delays to



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

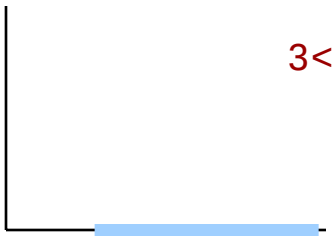


conjuncts to



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

projects to



3<x, y=0

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

Zones = Conjuctive Constraints

- A zone Z is a conjunctive formula:

$$g_1 \,\&\, g_2 \,\&\, \ldots \,\&\, g_n$$

  where g_i is a clock constraint $x_i \sim b_i$ or $x_i - x_j \sim b_{ij}$

- Use a zero-clock x_0 (constant 0)

- A zone can be re-written as a set:

$$\{x_i - x_j \sim b_{ij} \mid \sim \text{ is } < \text{ or } \leq,\ i,j \leq n\}$$

- This can be represented as a matrix, DBM (Difference Bound Matrices)

Operations on Zones

- Future delay $Z\uparrow$:

$$[Z\uparrow] = \{u+d \mid d \in R, u \in [Z]\}$$

- Past delay $Z\downarrow$:

$$[Z\downarrow] = \{u \mid u+d \in [Z] \text{ for some } d \in R\}$$

- Reset: $\{x\}Z$ or $Z(x:=0)$

$$[\{x\}Z] = \{u[0/x] \mid u \in [Z]\}$$

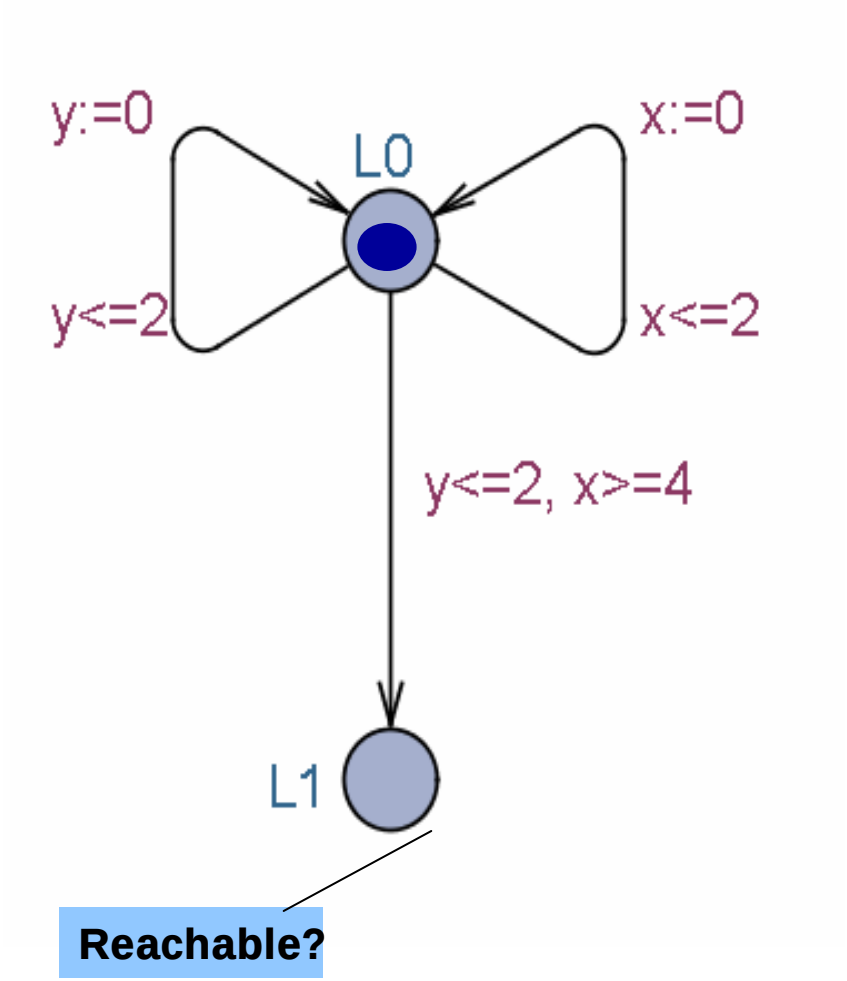
- Conjunction

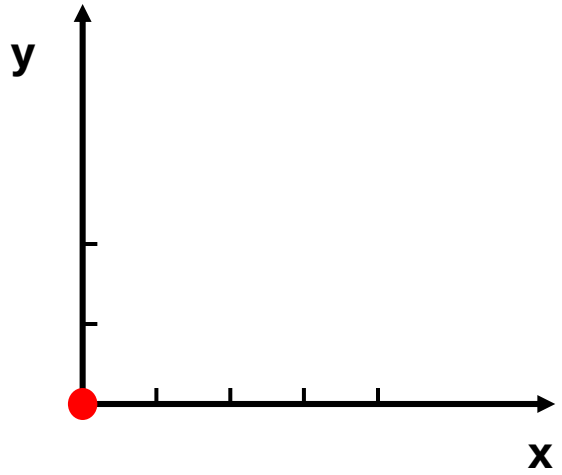
$$[Z\&g] = [Z] \cap [g]$$

THEOREM

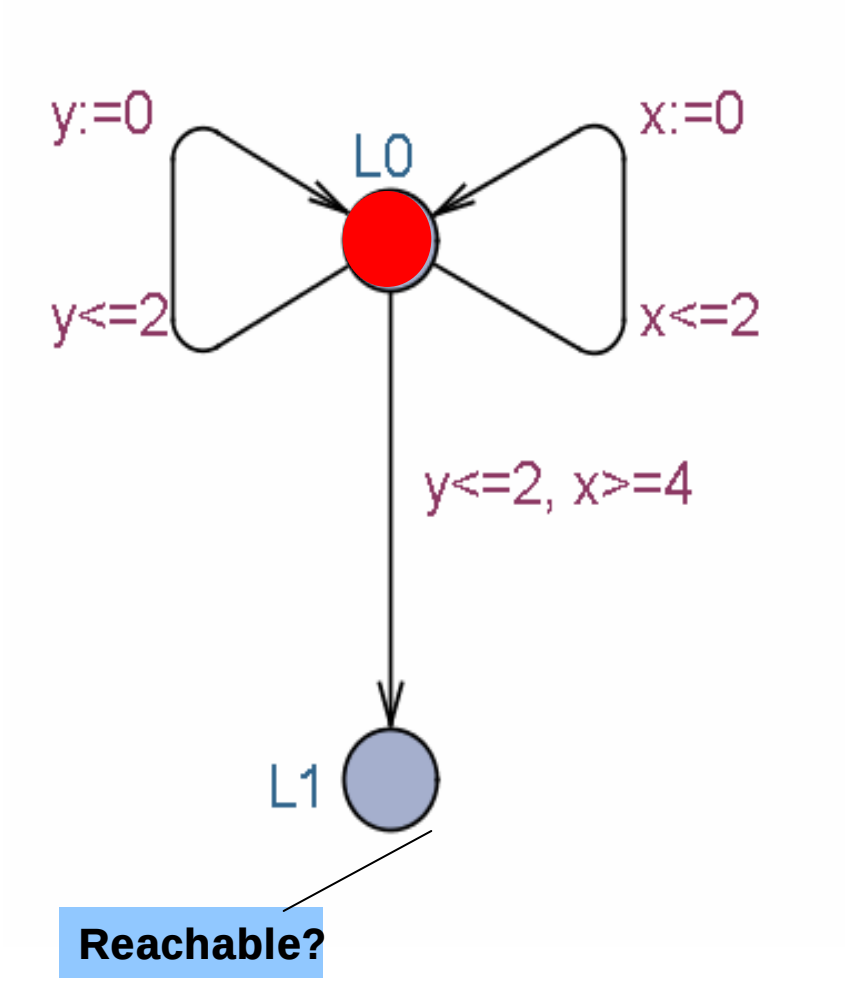
- The set of zones is closed under all constraint operations.
- That is, the result of the operations on a zone is a zone.
- That is, there will be a zone (a finite object i.e a zone/constraints) to represent the sets: [Z↑], [Z↓], [{x}Z], [Z&g].

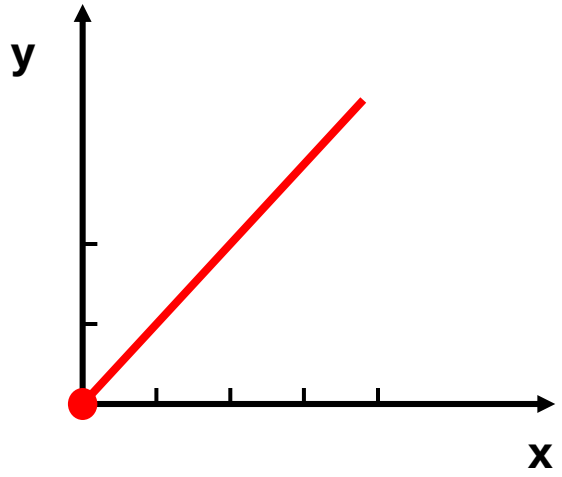
Symbolic Exploration





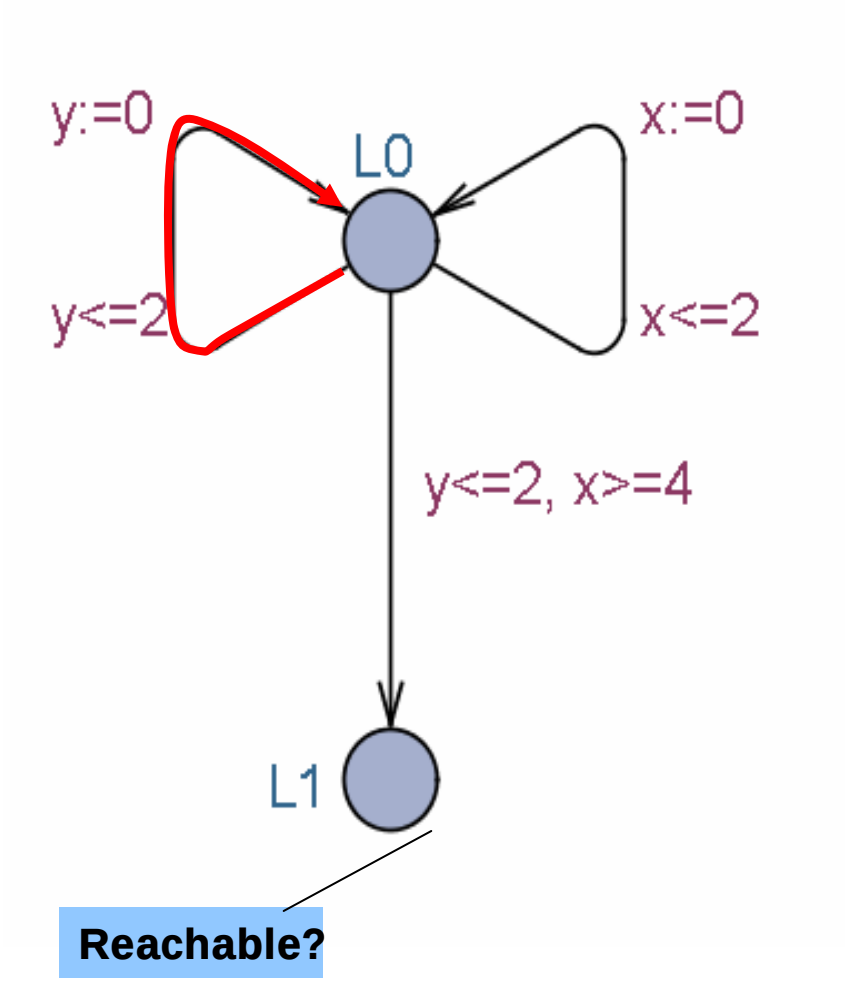
Symbolic Exploration

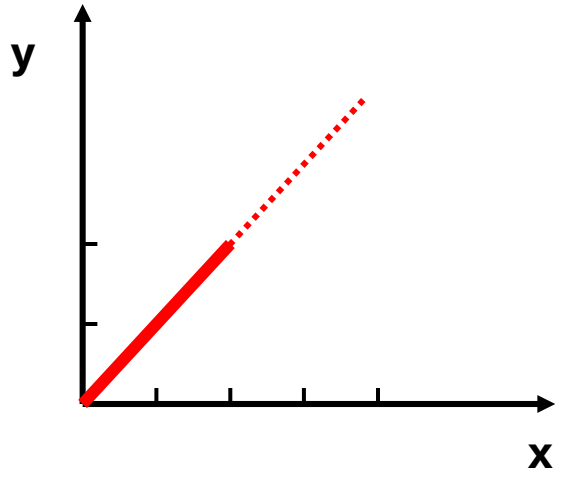




Delay

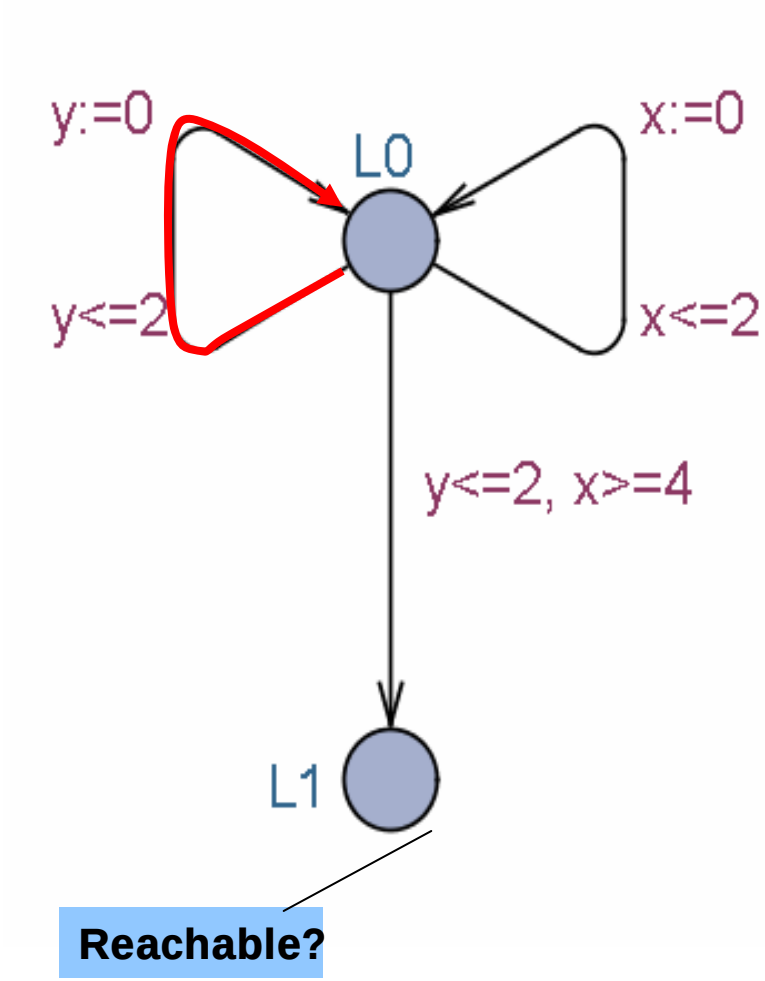
Symbolic Exploration

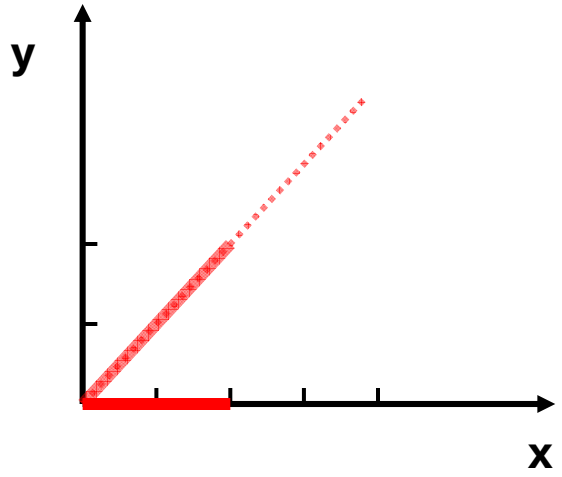




Left

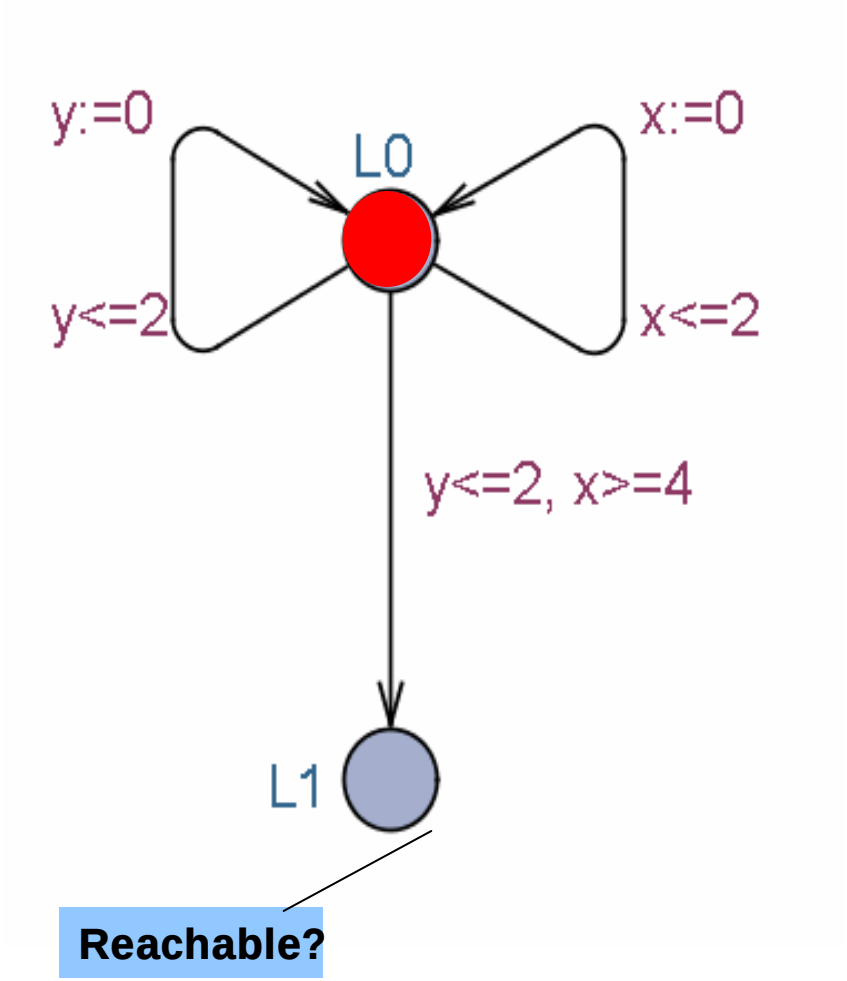
Symbolic Exploration

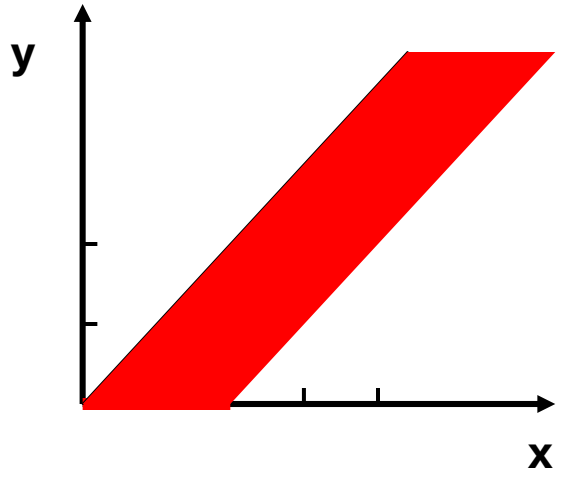




Left

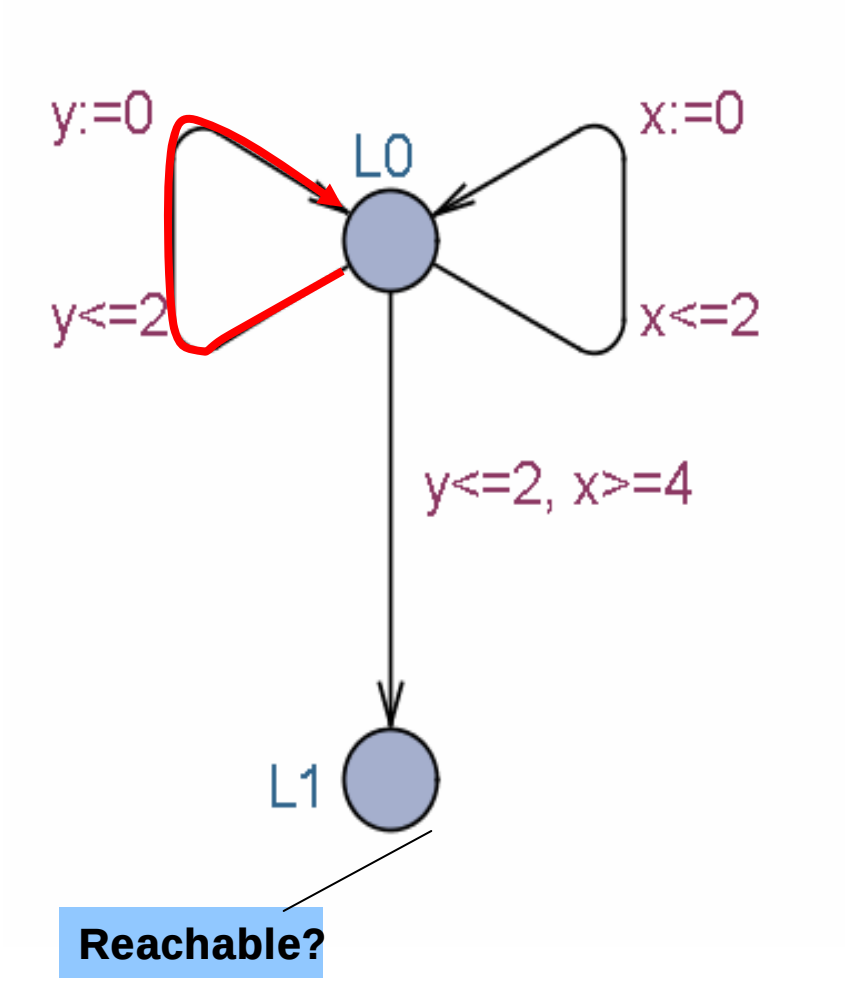
Symbolic Exploration

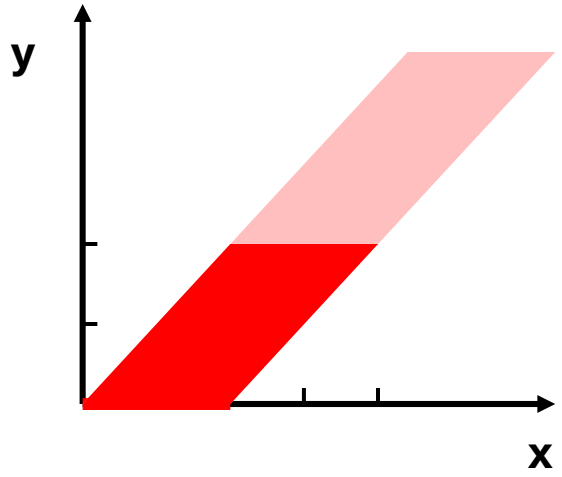




Delay

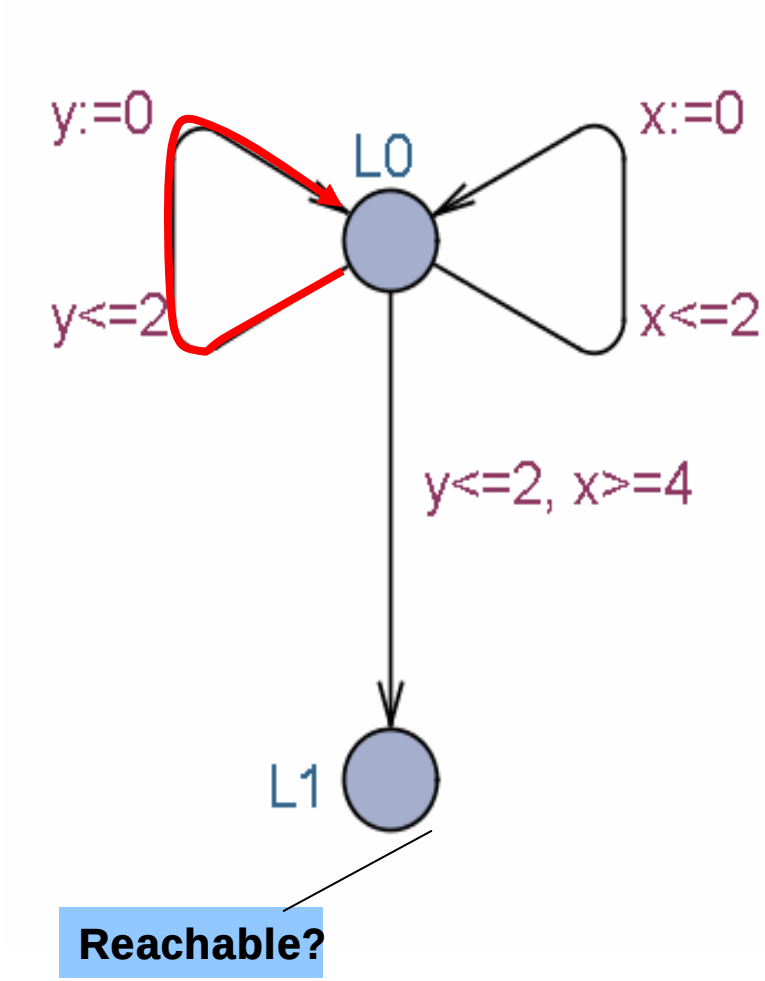
Symbolic Exploration

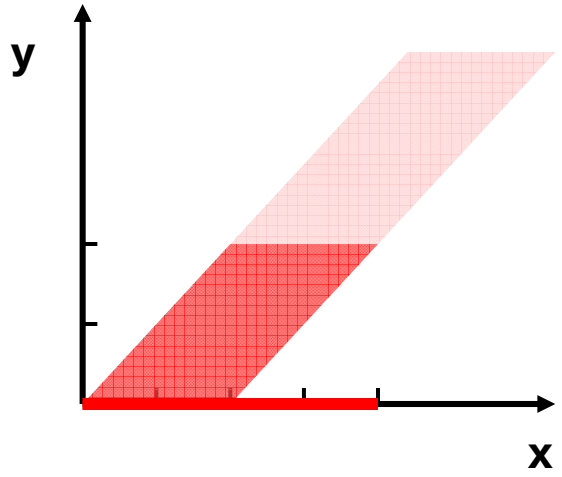




Left

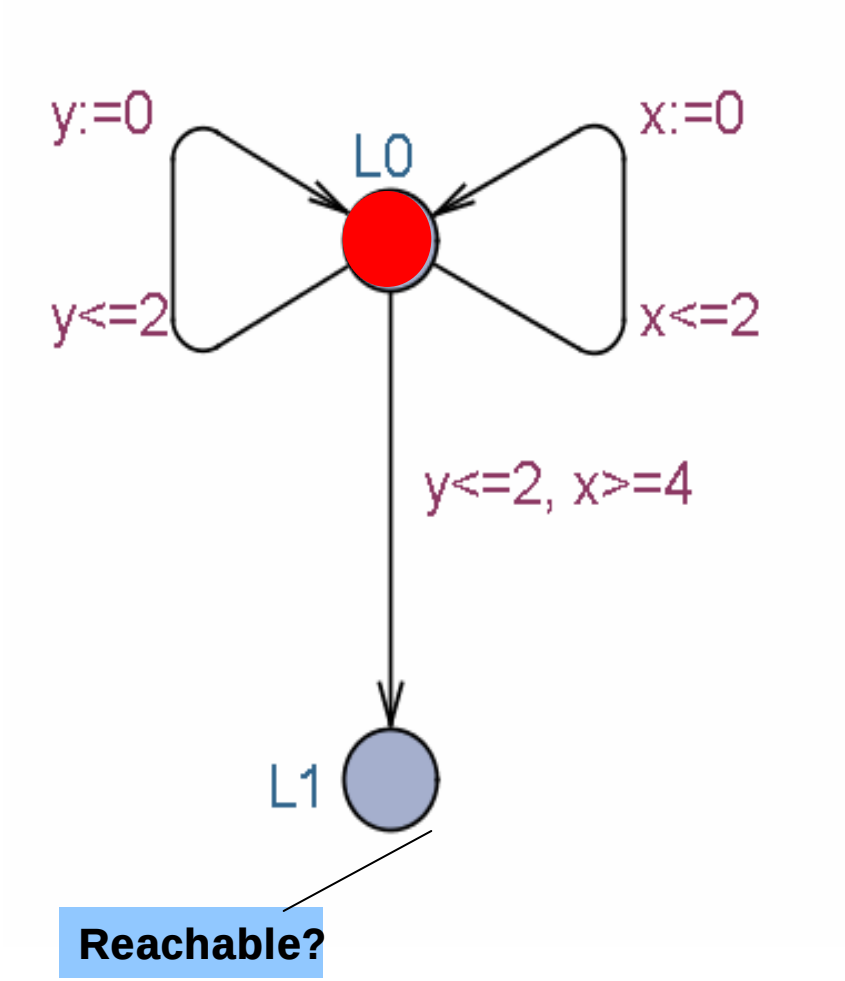
Symbolic Exploration

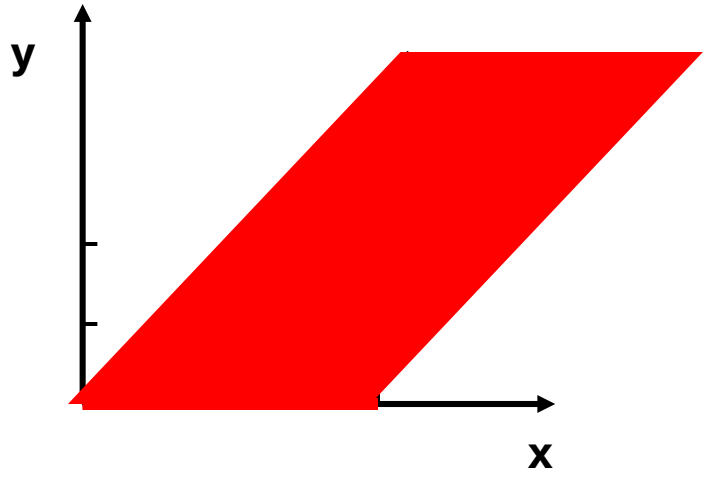




Left

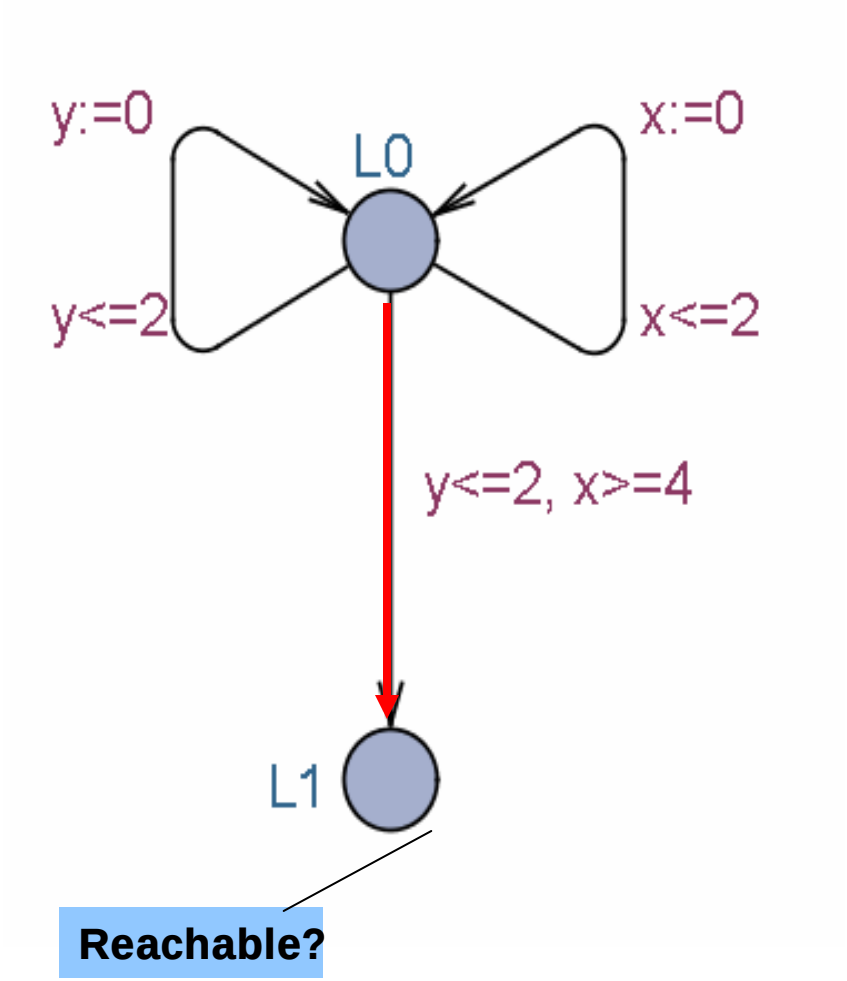
Symbolic Exploration

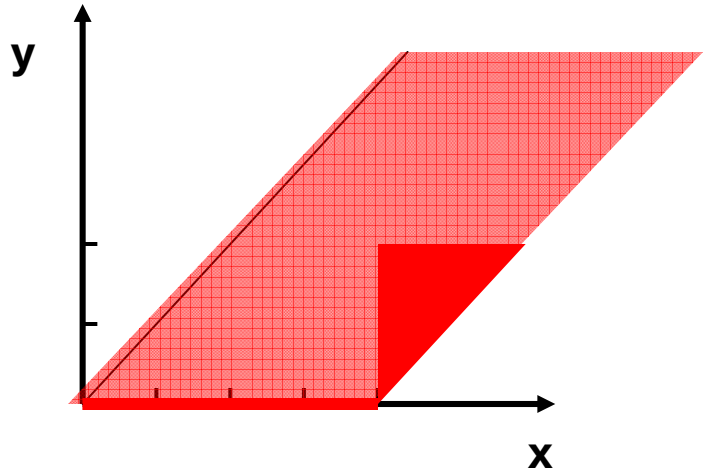




Delay

Symbolic Exploration

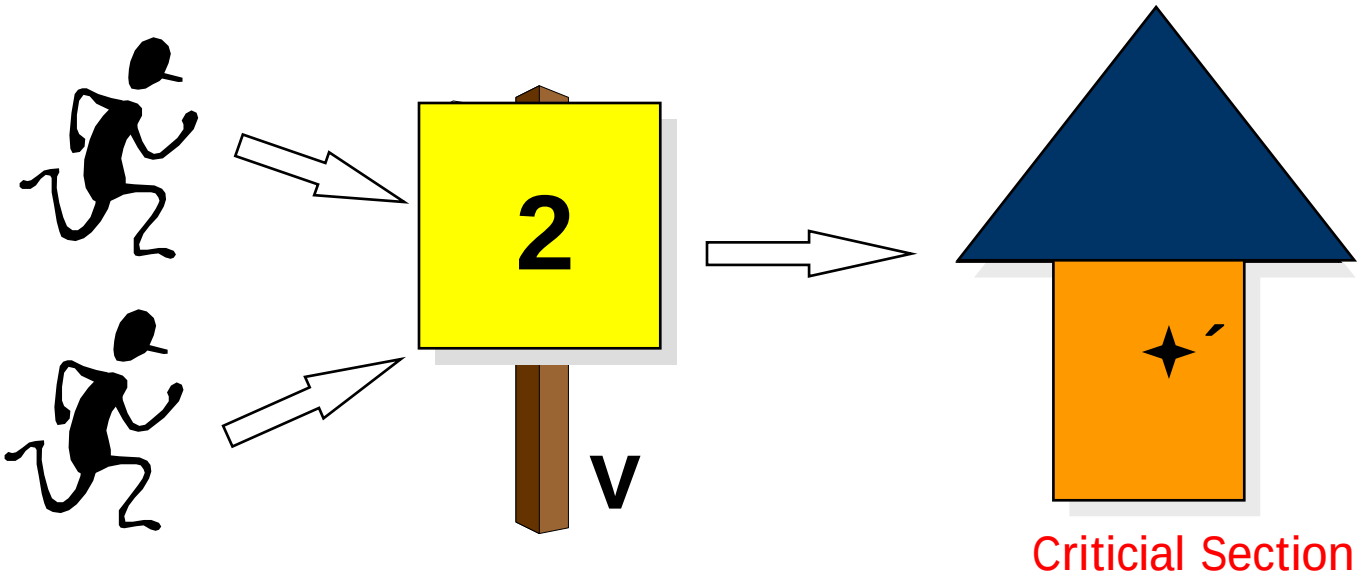




Down

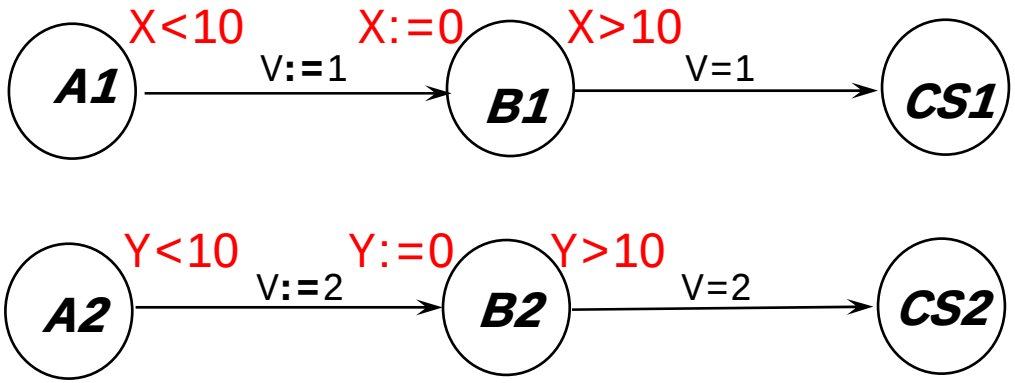
Fischer's Protocol

analysis using zones

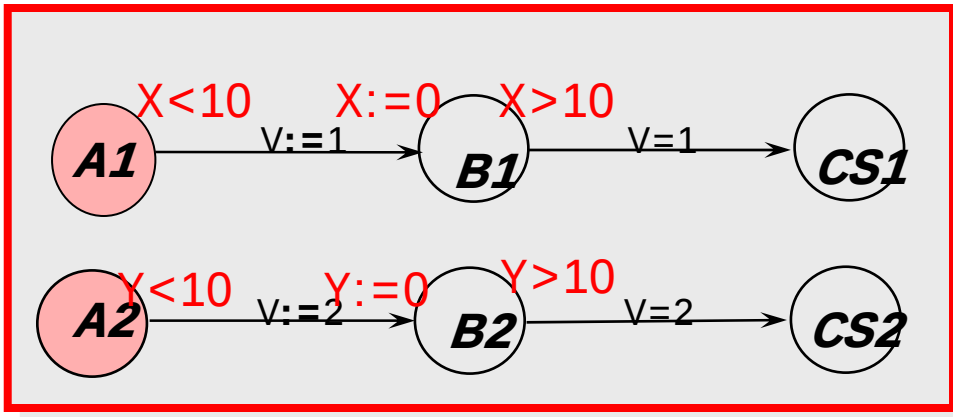


Init

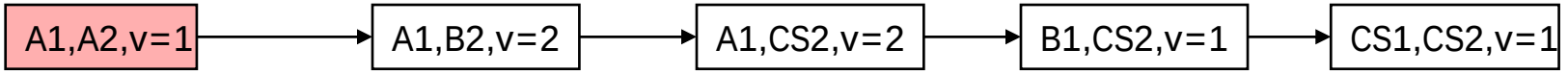
V=1



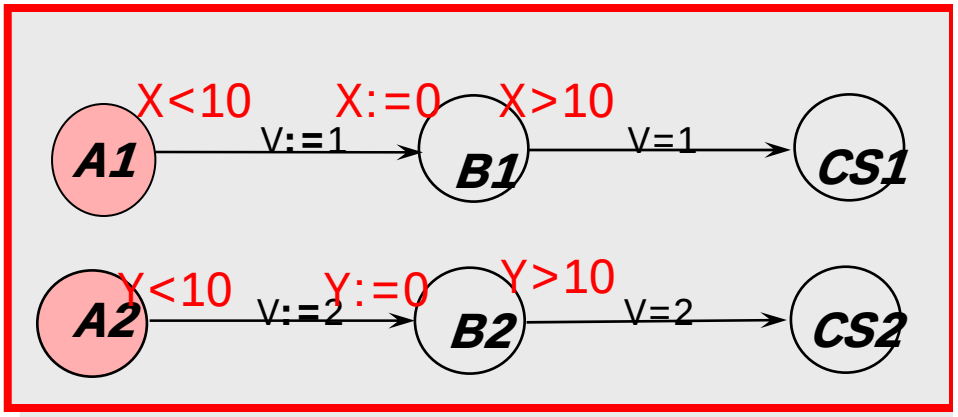
Fischers cont.



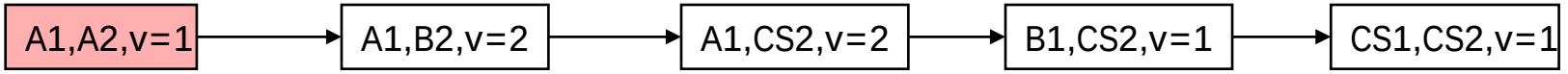
Untimed case



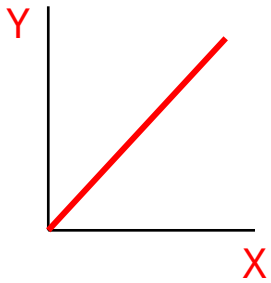
Fischers cont.



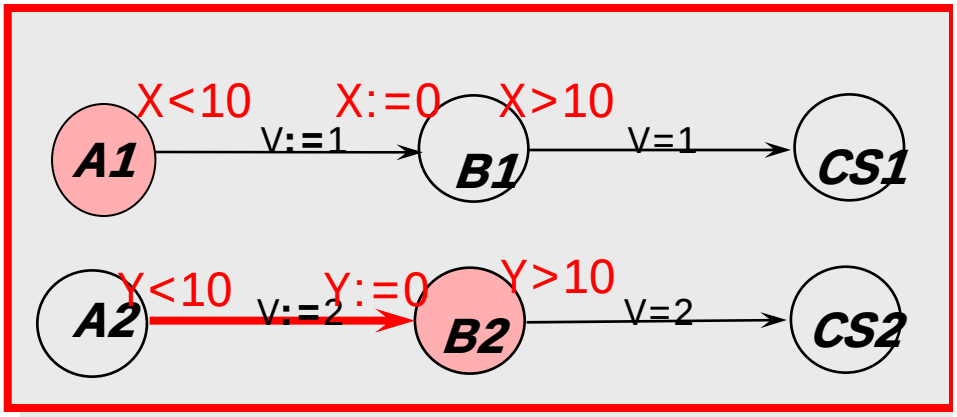
Untimed case



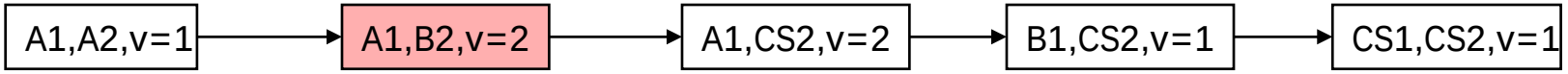
Taking time into account



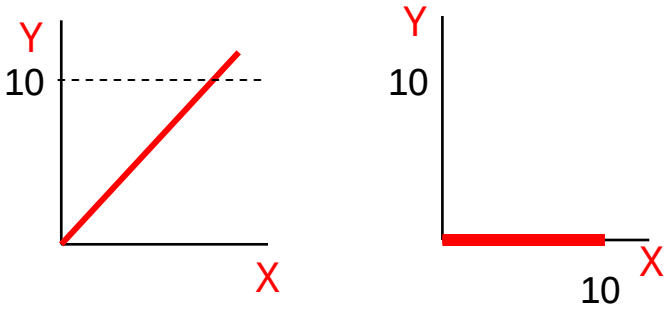
Fischers cont.



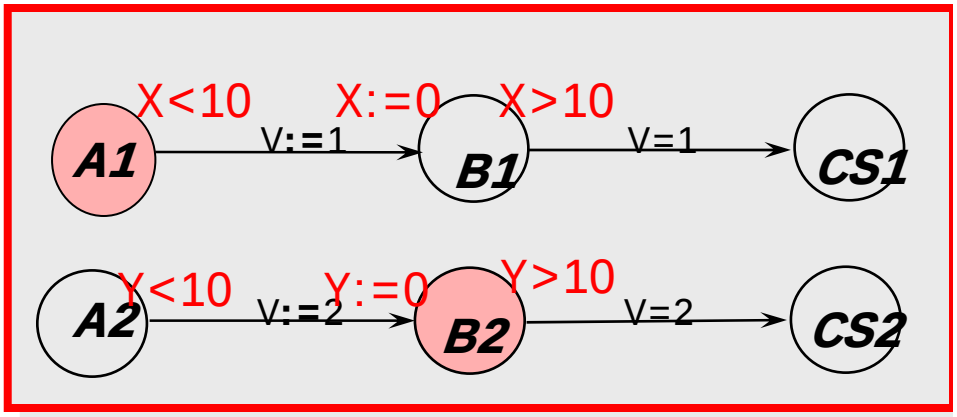
Untimed case



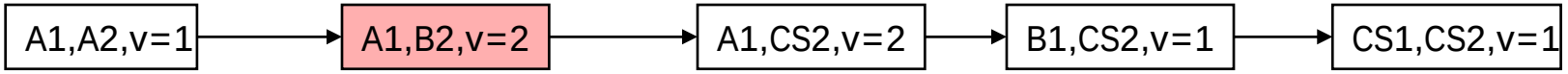
Taking time into account



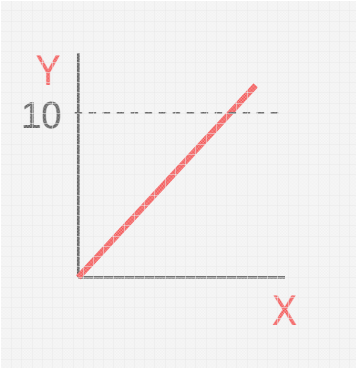
Fischers cont.

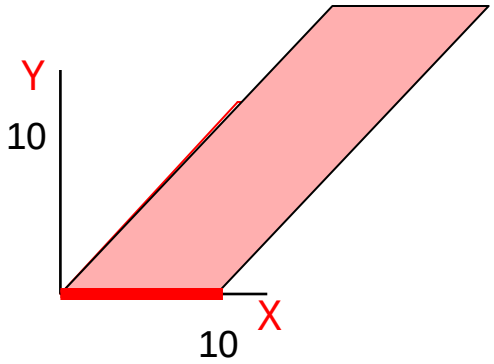


Untimed case



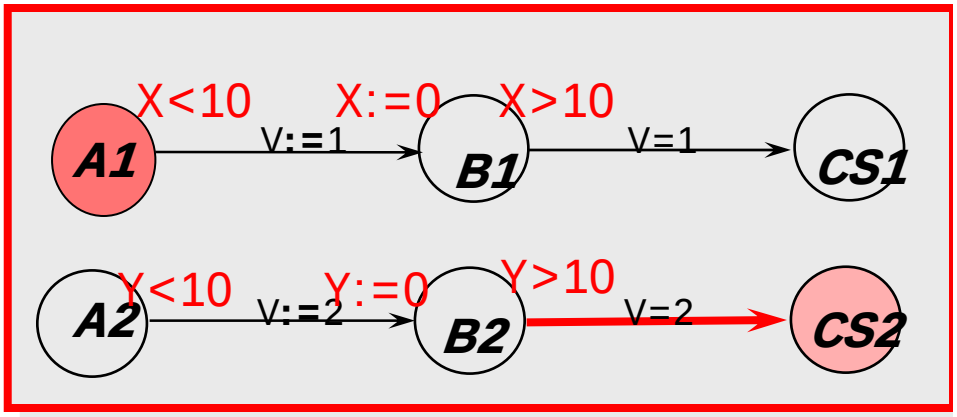
Taking time into account



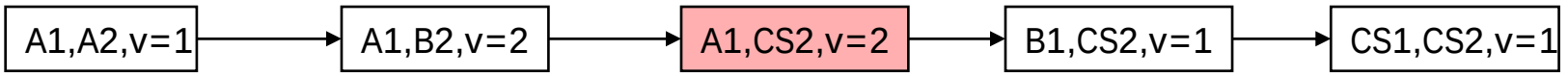


BRICS
Basic Research
in Computer Science

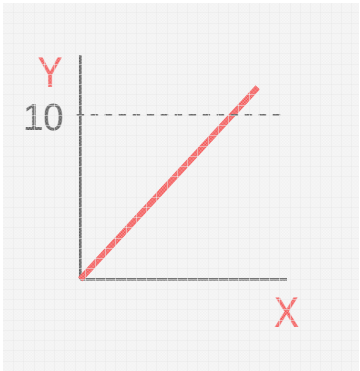
Fischers cont.

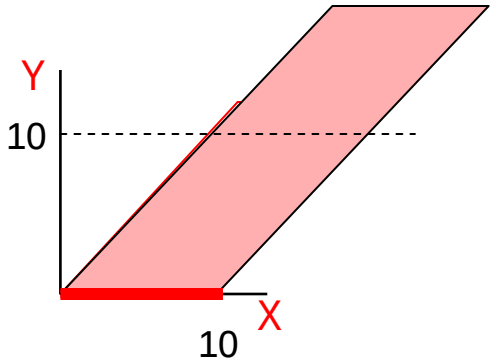


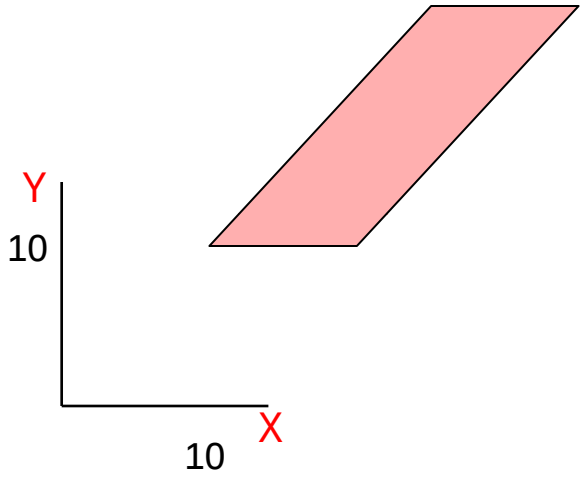
Untimed case



Taking time into account

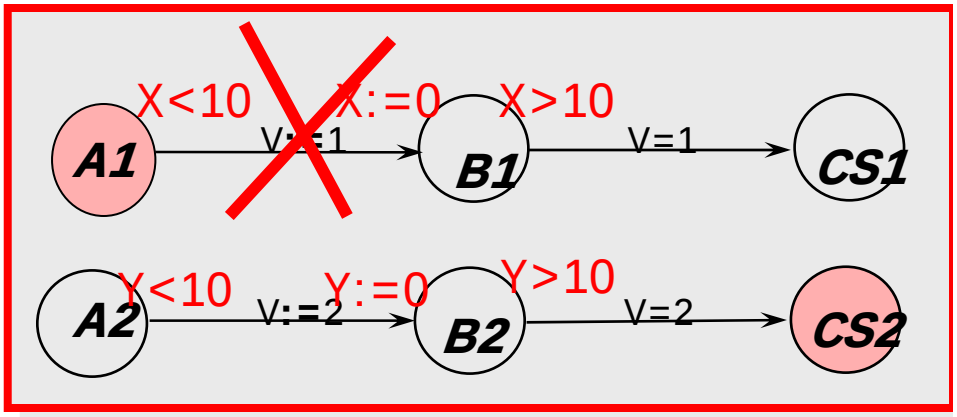




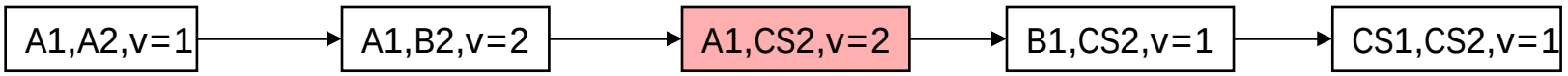


BRICS
Basic Research
in Computer Science

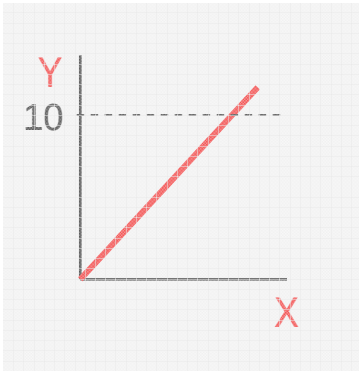
Fischers cont.

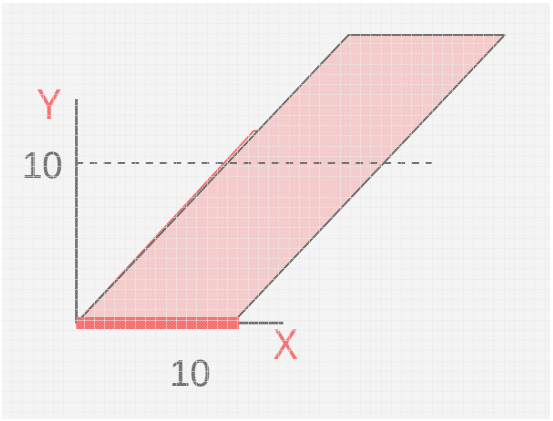


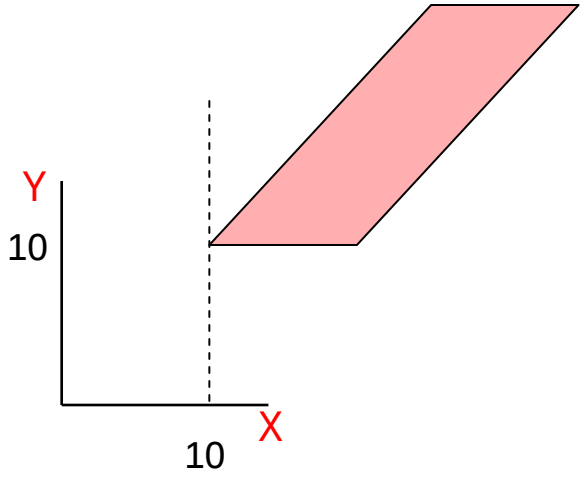
Untimed case



Taking time into account



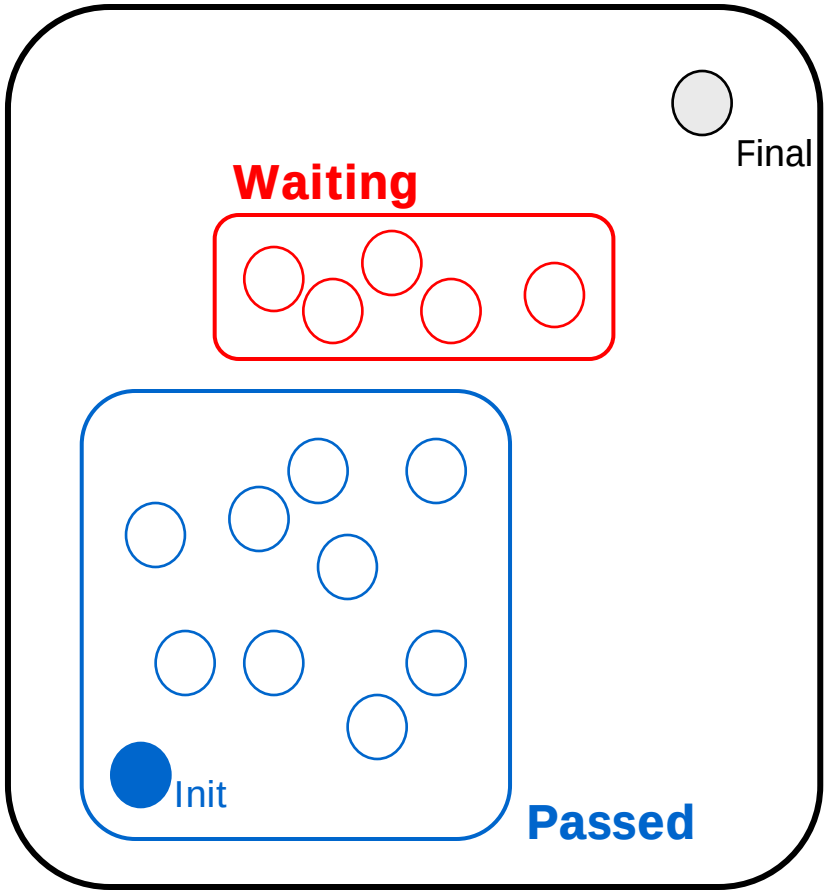




BRICS
Basic Research
in Computer Science

Forward Rechability

Init -> Final ?



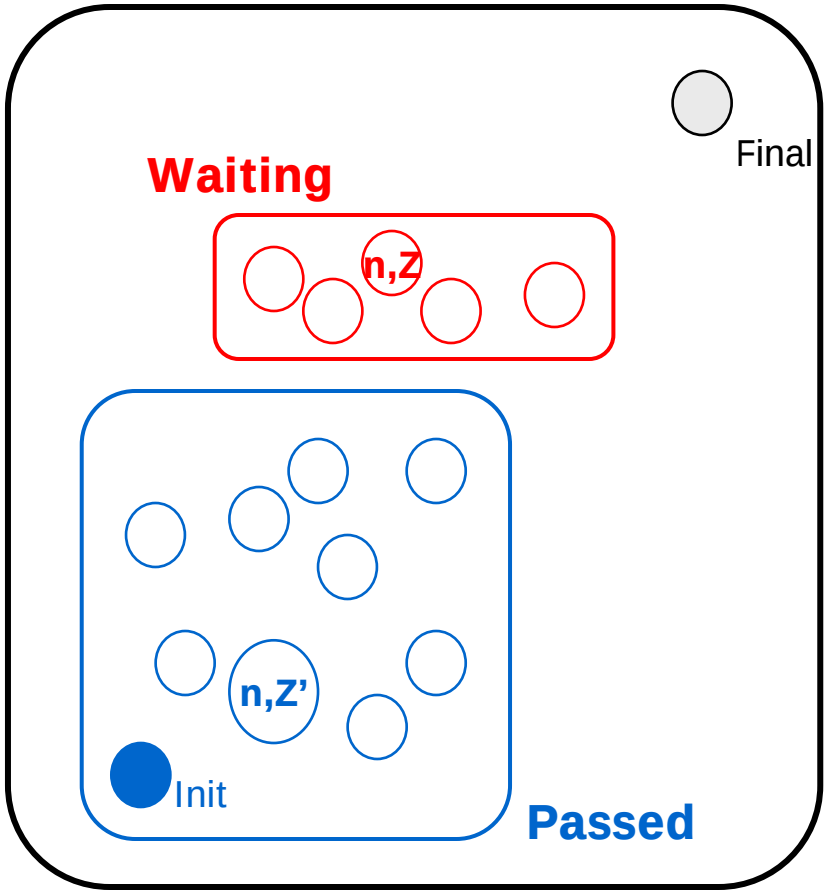
INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

REPEAT

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

Forward Rechability

Init -> Final ?



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

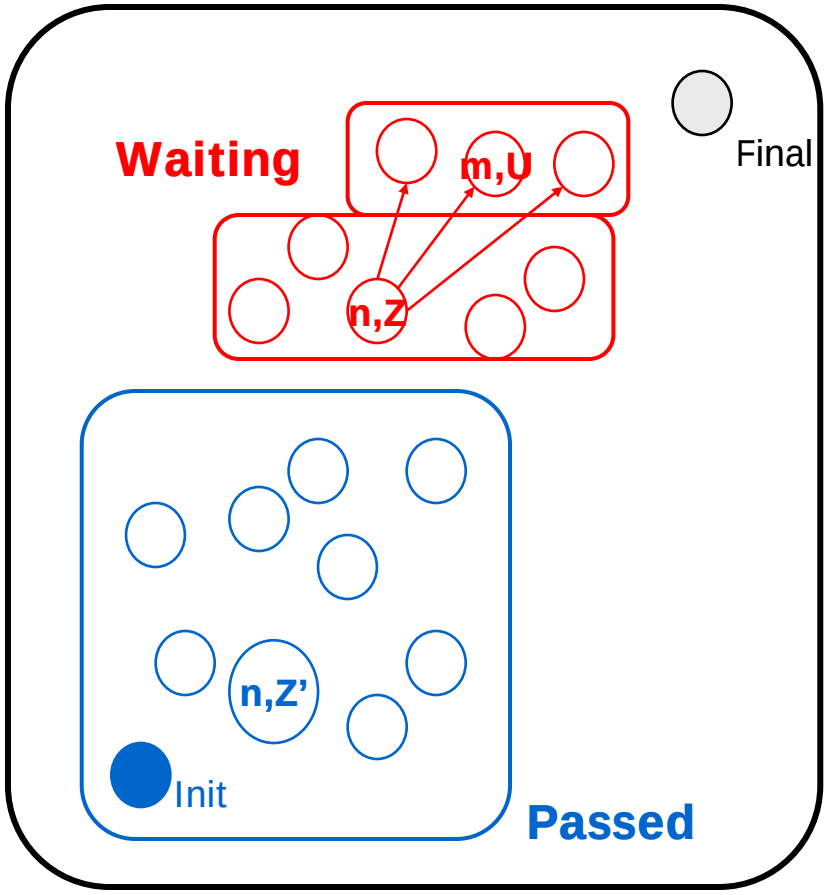
REPEAT
- pick (n,Z) in **Waiting**
- **if** for some Z' ⊇ Z
  (n,Z') in **Passed** **then** **STOP**

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

BRICS
Basic Research
in Computer Science

Forward Rechability

Init -> Final ?



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

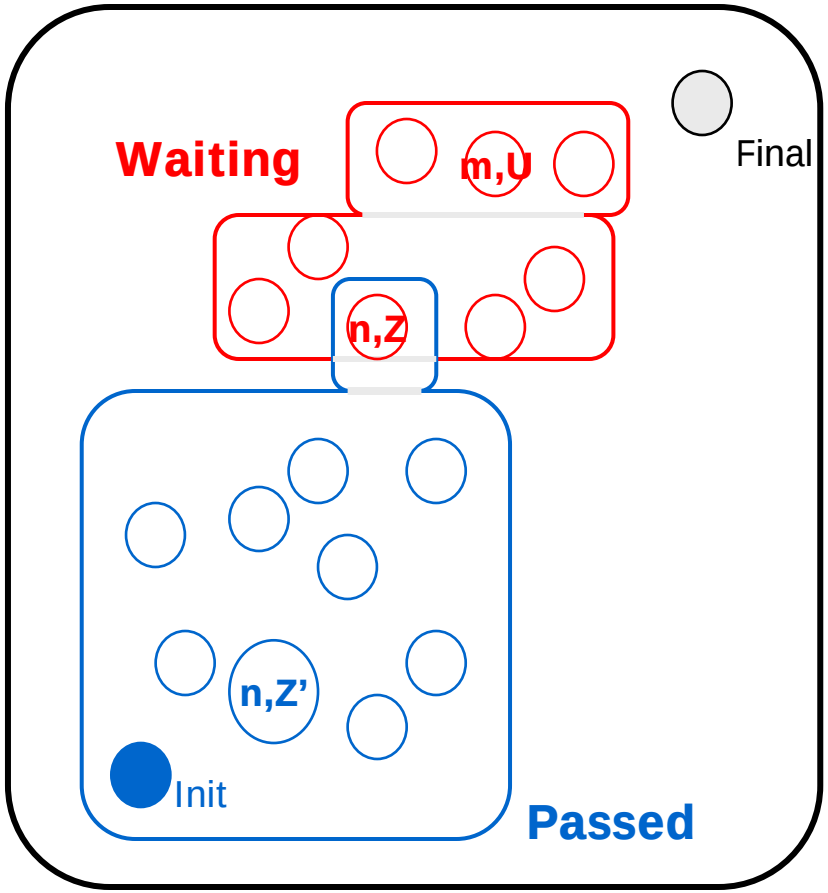
REPEAT
- pick (n,Z) in **Waiting**
- **if** for some Z' ⊇ Z
  (n,Z') in **Passed** **then** **STOP**
- **else** /explore/ add
  { (m,U) : (n,Z) => (m,U) }
  to **Waiting**;

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

BRICS
Basic Research
in Computer Science

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**

BRICS
Basic Research
in Computer Science

Canonical Datastructures for Zones

Difference Bounded Matrices

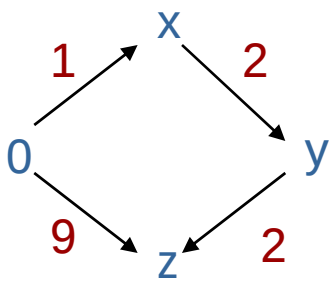
Bellman 1958, Dill 1989

Inclusion

D1

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

Graph

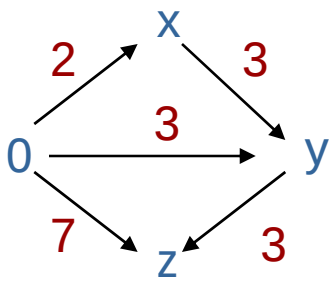


? ⊆ ?

D2

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

Graph



BRICS
Basic Research
in Computer Science

Canonical Datastructures for Zones

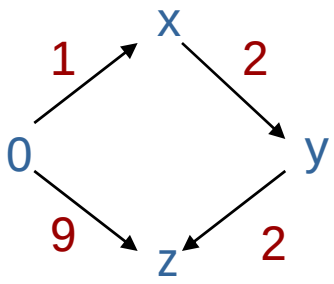
Difference Bounded Matrices

Inclusion

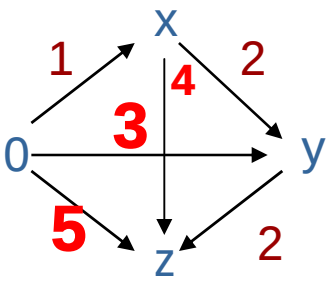
D1

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

Graph



Shortest Path Closure

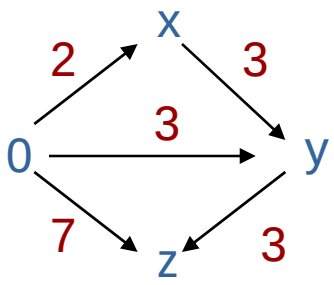


$? \subseteq ?$

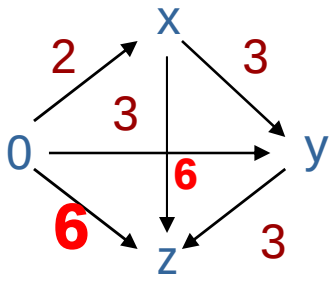
D2

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

Graph



Shortest Path Closure



BRICS Basic Research in Computer Science

Canonical Datastructures for Zones

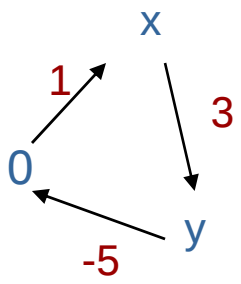
Difference Bounded Matrices

Emptiness

D

$x <= 1$
$y >= 5$
$y - x <= 3$

Graph



Negative Cycle
iff
empty solution set

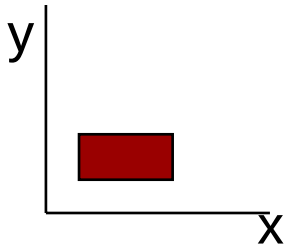
Compact

BRICS
Basic Research
in Computer Science

Canonical Datastructures for Zones

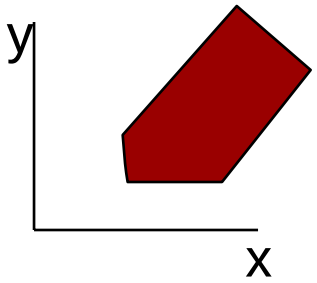
Difference Bounded Matrices

Future



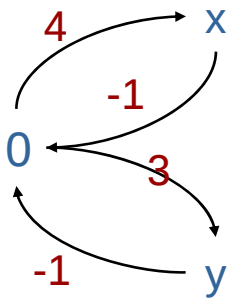
D

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

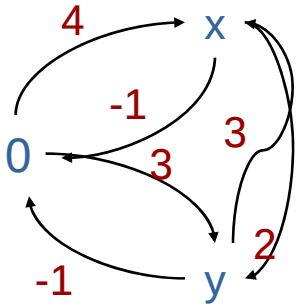


Future D

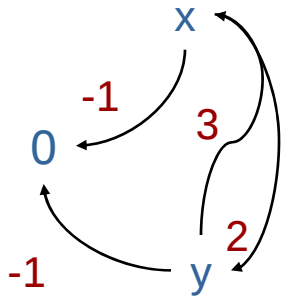
1<=x, 1<=y
-2<=x-y<=3



Shortest Path Closure



Remove upper bounds on clocks

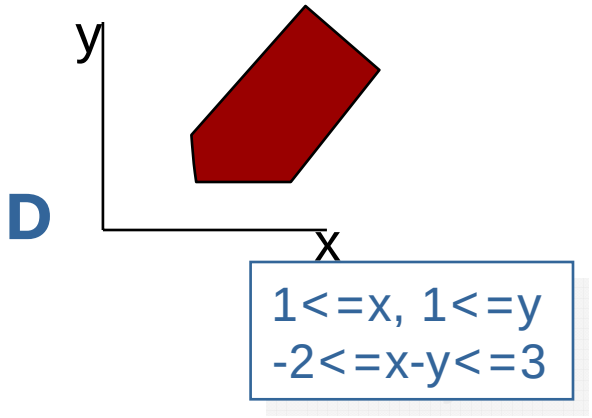


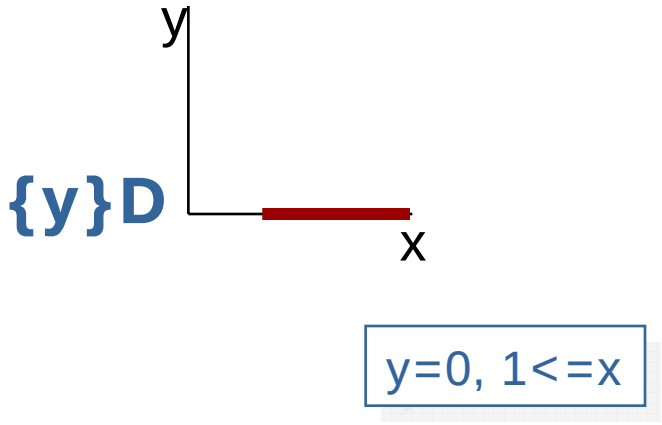
BRICS
Basic Research
in Computer Science

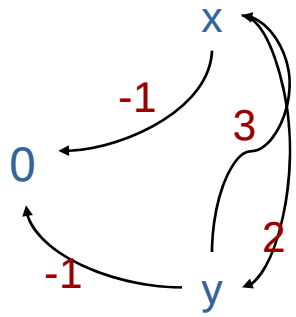
Canonical Datastructures for Zones

Difference Bounded Matrices

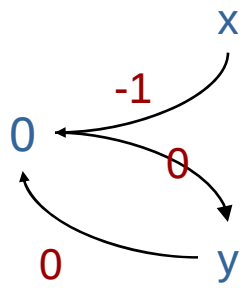
Reset







Remove all bounds involving y and set y to 0

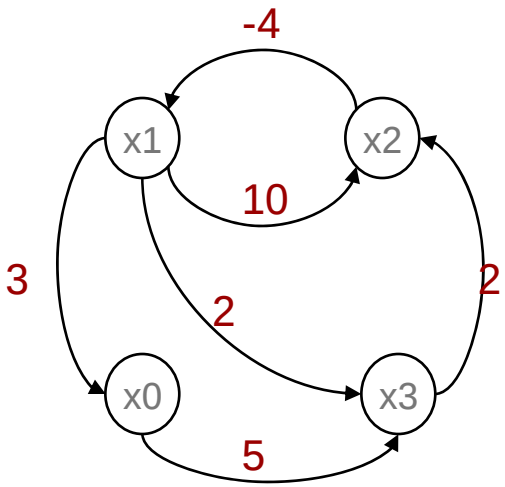


BRICS Basic Research in Computer Science

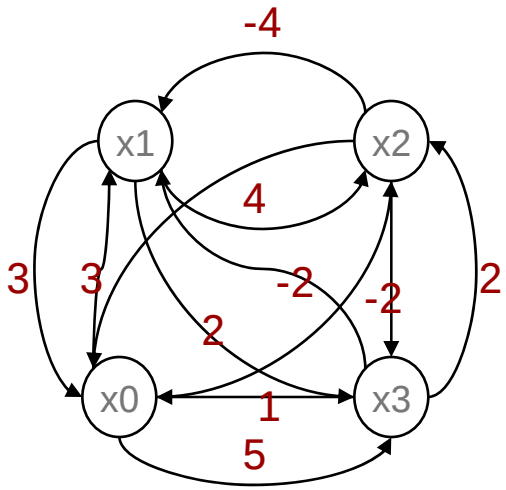
Canonical Datastructures for Zones

Difference Bounded Matrices

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



Shortest Path Closure O(n^3)



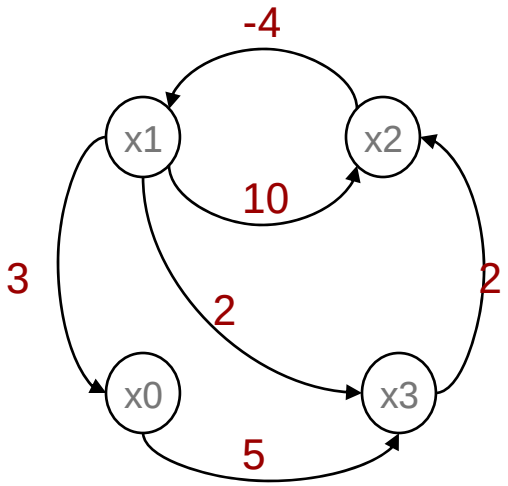
BRICS
Basic Research
in Computer Science

Canonical Datastructures for Zones

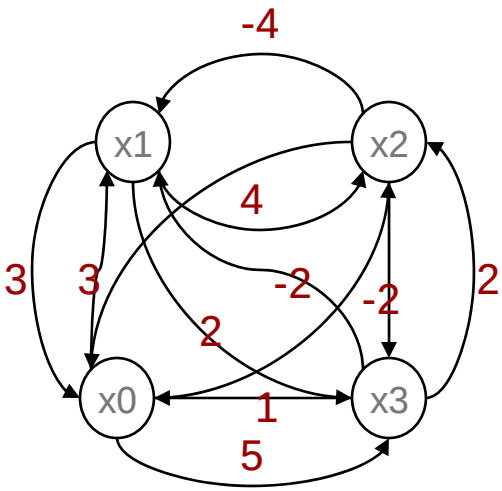
Minimal Constraint Form

RTSS 1997

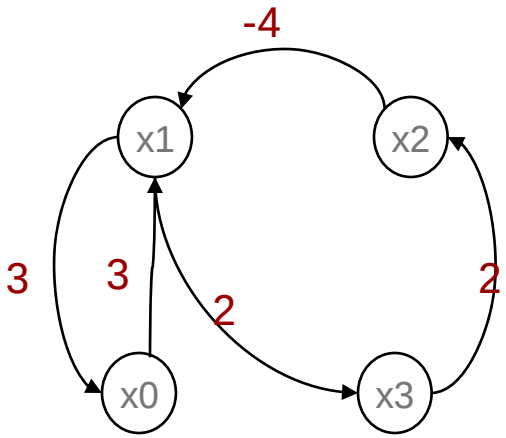
x1-x2<=4
x2-x1<=10
x3-x1<=2
x2-x3<=2
x0-x1<=3
x3-x0<=5



Shortest Path Closure O(n^3)



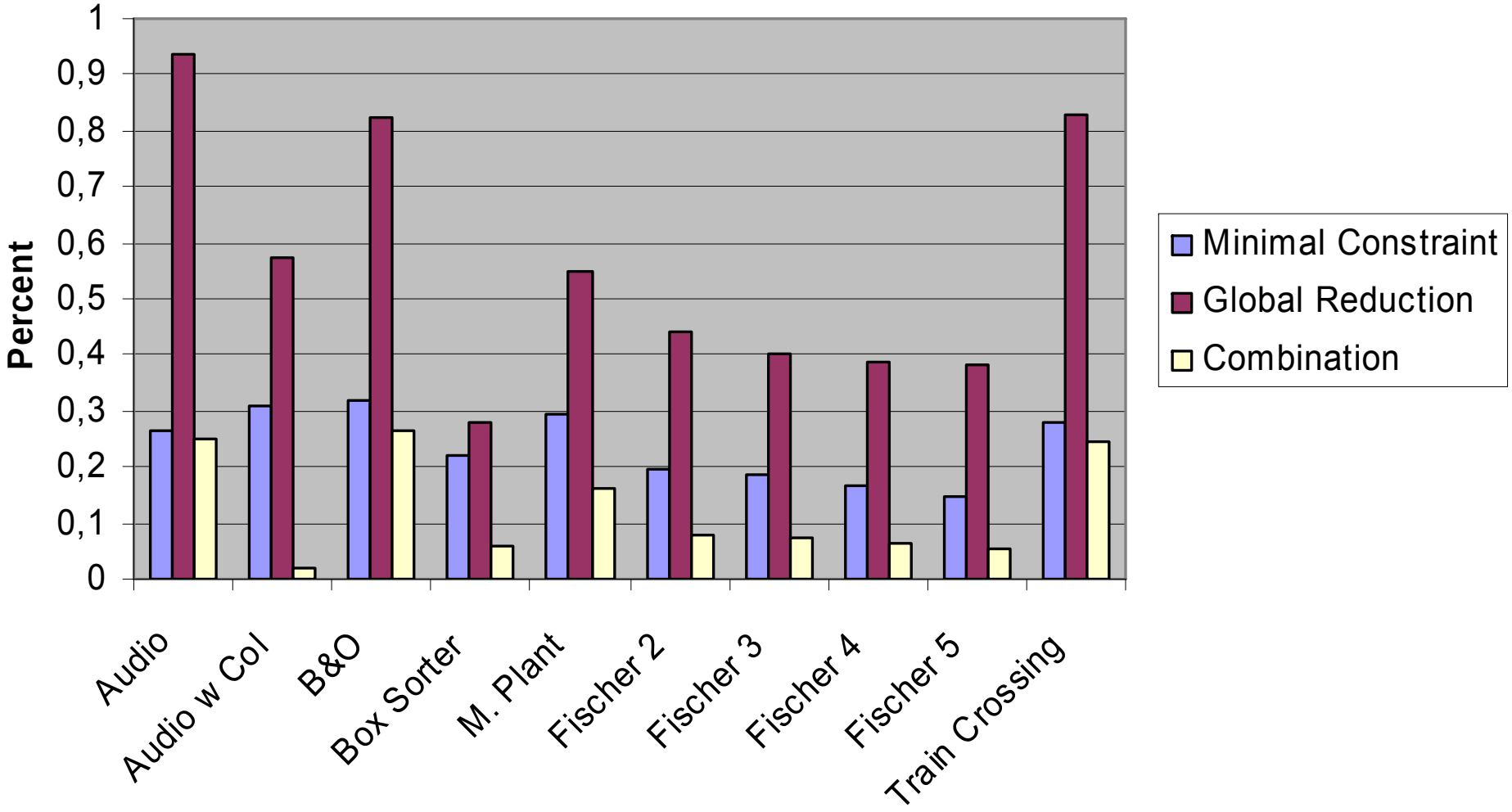
Shortest Path Reduction O(n^3)



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

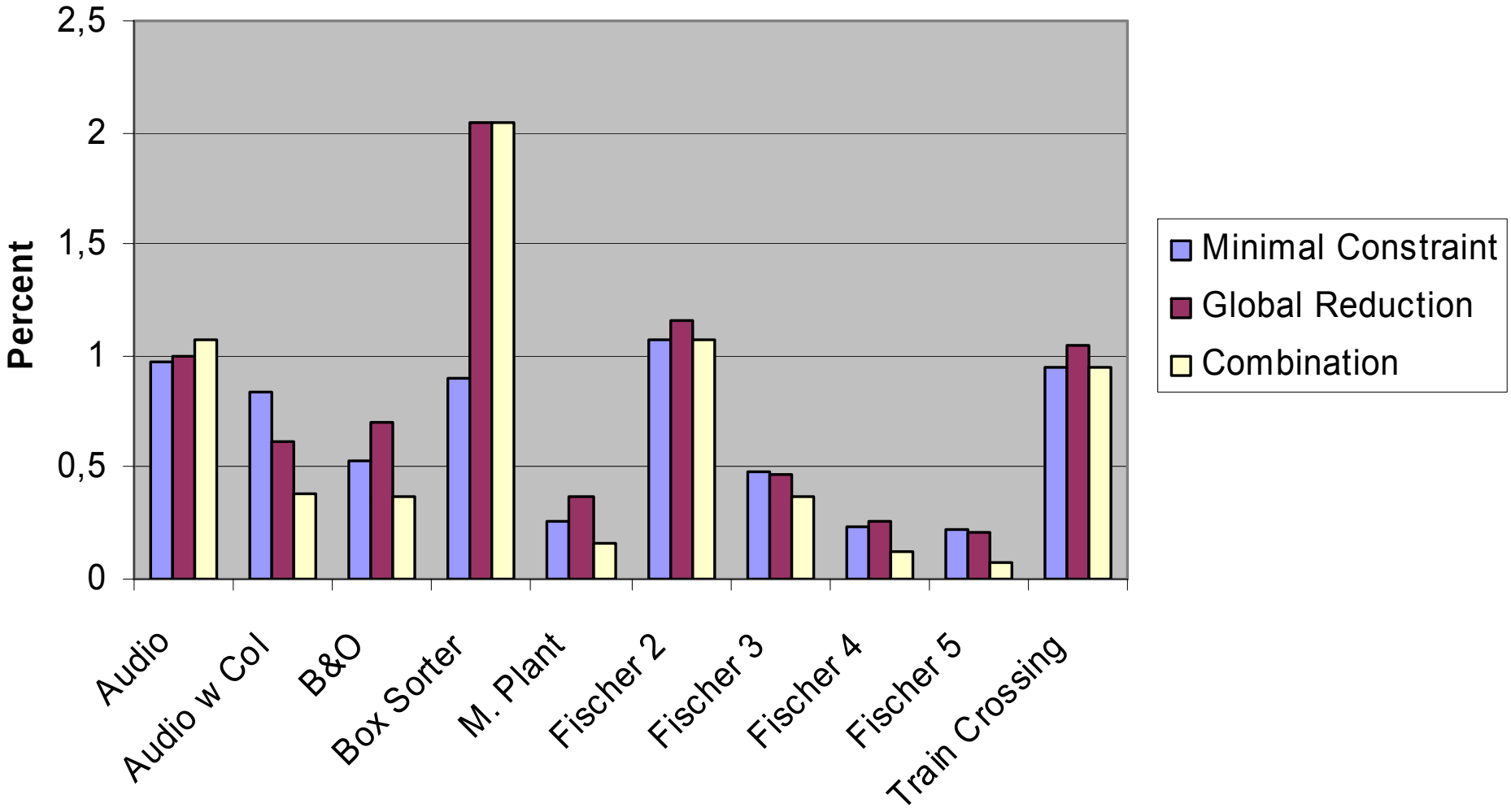
BRICS
Basic Research
in Computer Science

SPACE PERFORMANCE



BRICS
Basic Research
in Computer Science

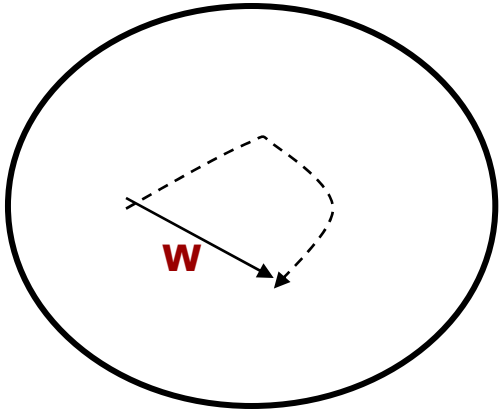
TIME PERFORMANCE



Shortest Path Reduction

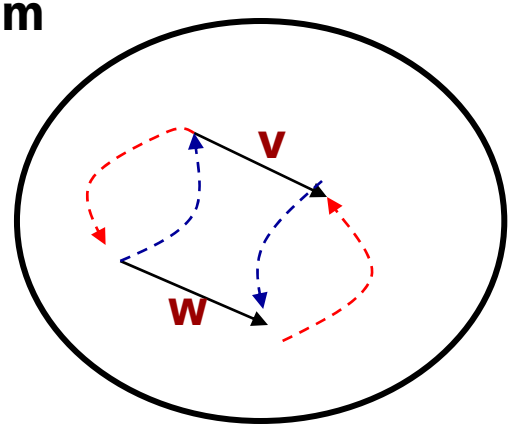
1st attempt

Idea



An edge is REDUNDANT if there exists an alternative path of no greater weight
THUS Remove all redundant edges!

Problem

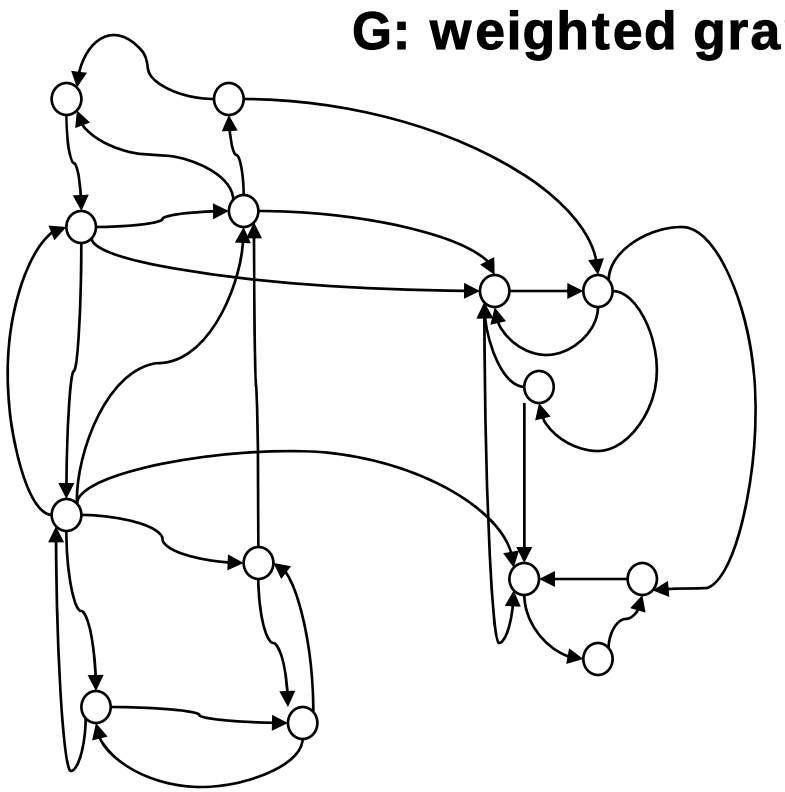


v and **w** are both redundant
Removal of one depends on presence of other.

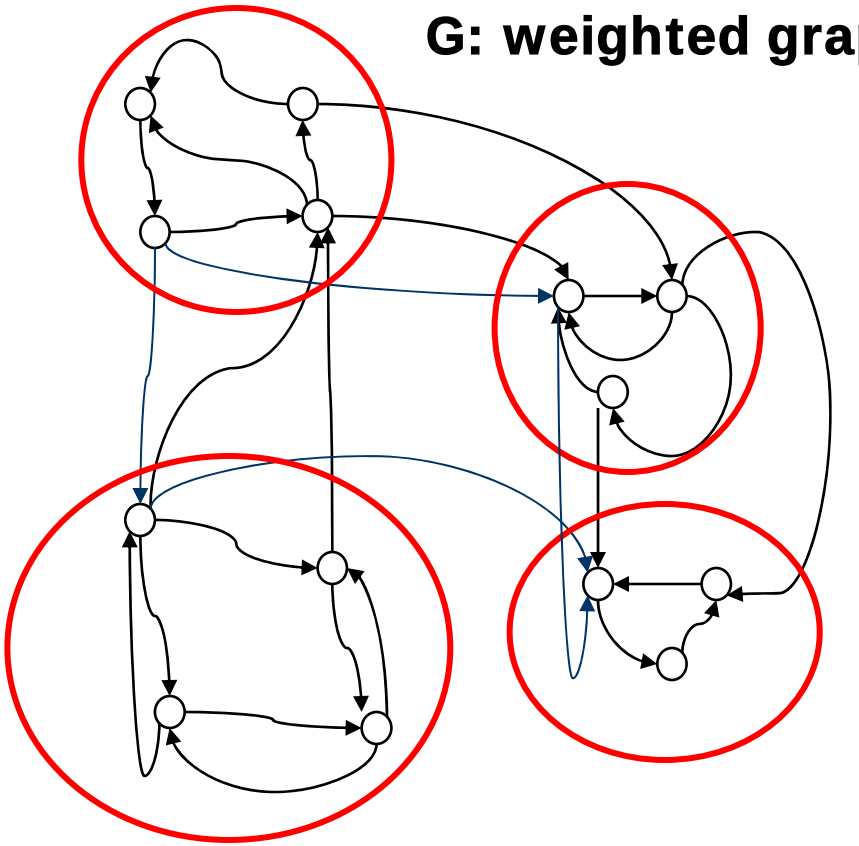
Observation: If no zero- or negative cycles then SAFE to remove all redundancies.

Shortest Path Reduction

Solution

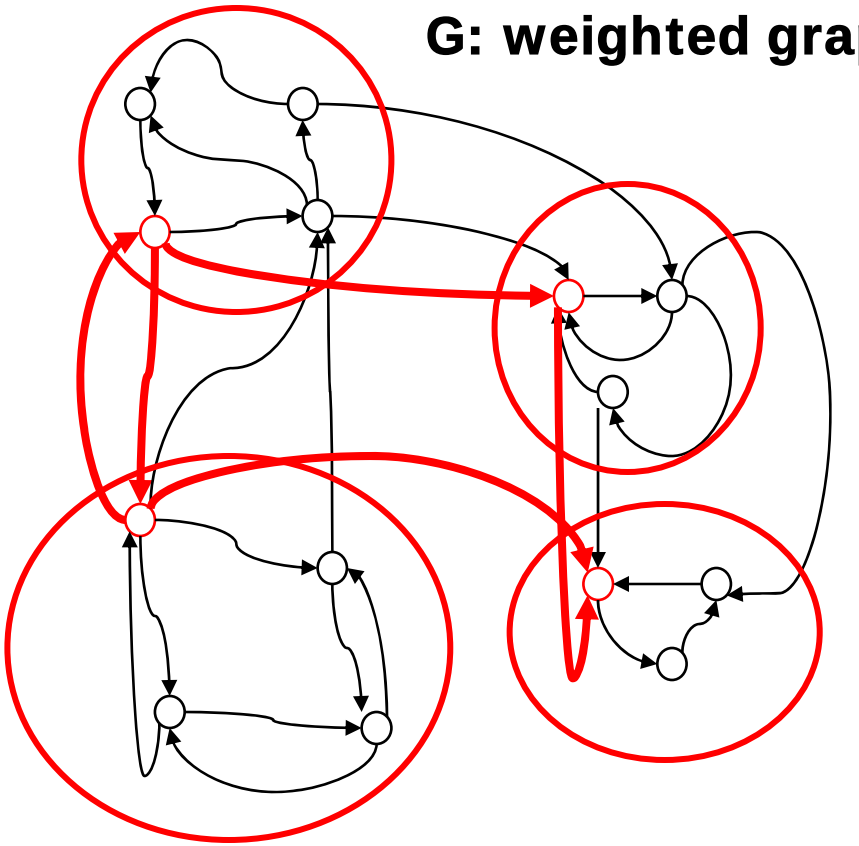


Shortest Path Reduction
Solution



1. Equivalence classes based on 0-cycles.

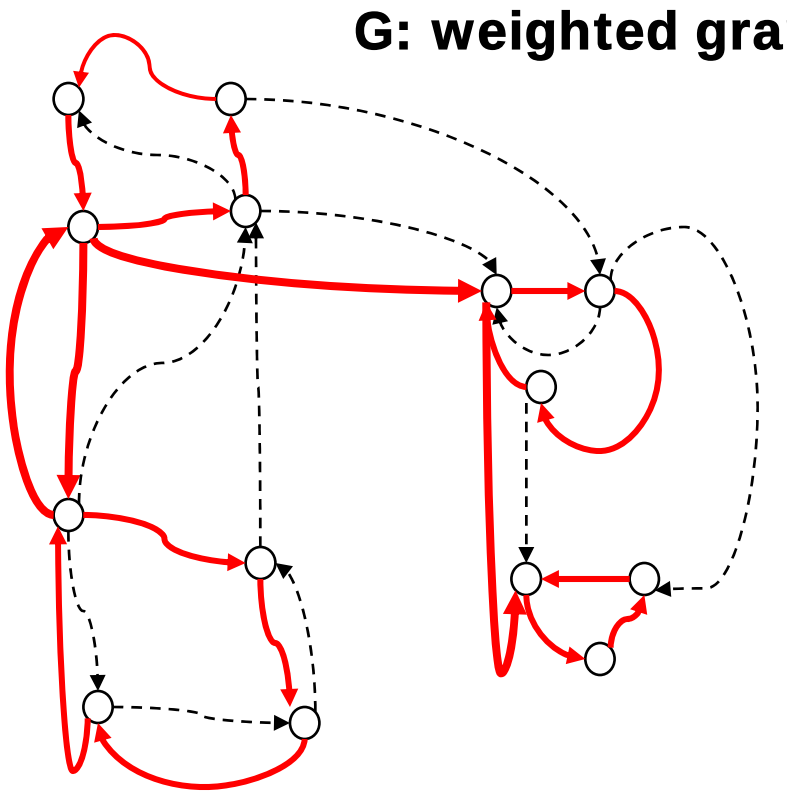
Shortest Path Reduction
Solution



1. Equivalence classes based on 0-cycles.

2. Graph based on representatives.
   Safe to remove redundant edges

Shortest Path Reduction
Solution



1. Equivalence classes based on 0-cycles.

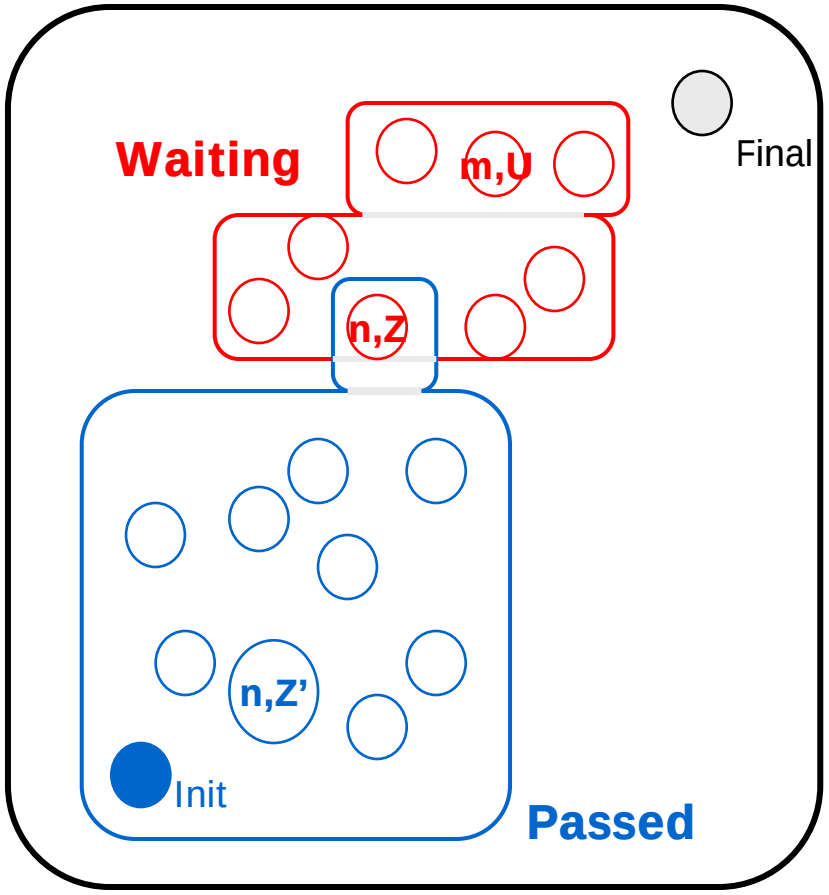
2. Graph based on representatives.
   Safe to remove redundant edges

3. **Shortest Path Reduction**
   =
   One cycle pr. class
   +
   Removal of redundant edges between classes

Canonical given order of clocks

Earlier Termination

Init -> Final ?



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

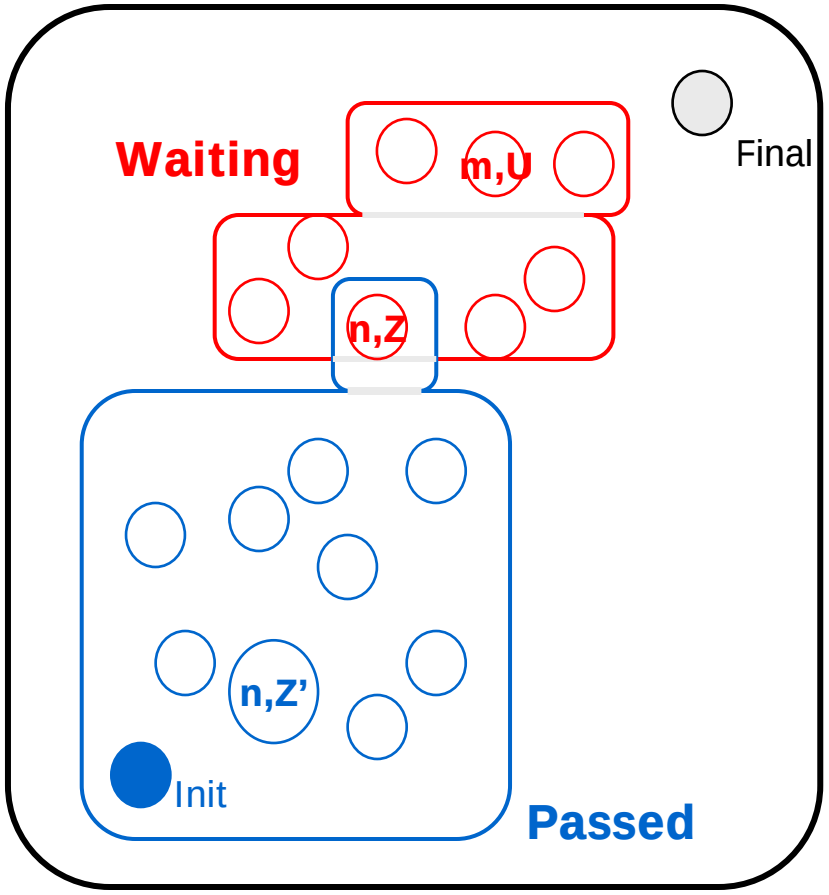
REPEAT
- pick (n,Z) in **Waiting**
- **if** for some Z' ⊇ Z
  (n,Z') in **Passed** **then** **STOP**
- **else** /explore/ add
  { (m,U) : (n,Z) => (m,U) }
  to **Waiting**;
  Add (n,Z) to **Passed**

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

BRICS
Basic Research
in Computer Science

Earlier Termination

Init -> Final ?



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

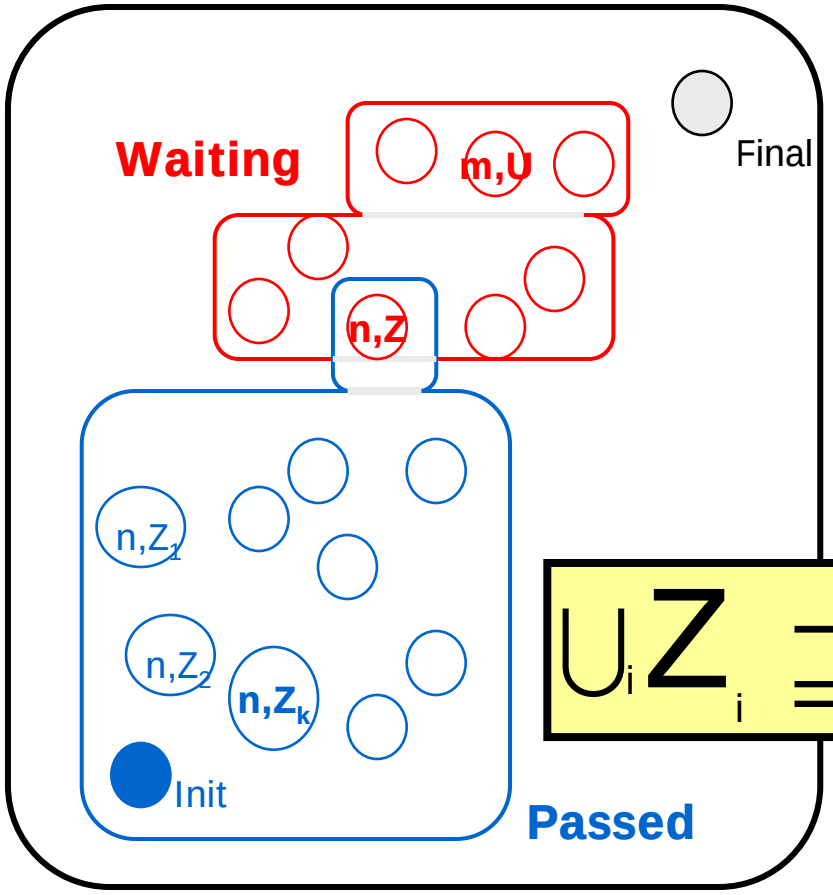
REPEAT
- pick (n,Z) in **Waiting**
- **if** for some $Z' \supseteq Z$ (n,Z') in **Passed** **then** **STOP**
- **else** /explore/ add
  { (m,U) : (n,Z) => (m,U) }
  to **Waiting**;
  Add (n,Z) to **Passed**

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

BRICS
Basic Research
in Computer Science

Earlier Termination

Init -> Final ?



$$\bigcup_i Z_i \supseteq Z$$

$Z' \supseteq Z$

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

REPEAT
  - pick (n,Z) in Waiting
  - if for some
    (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
```

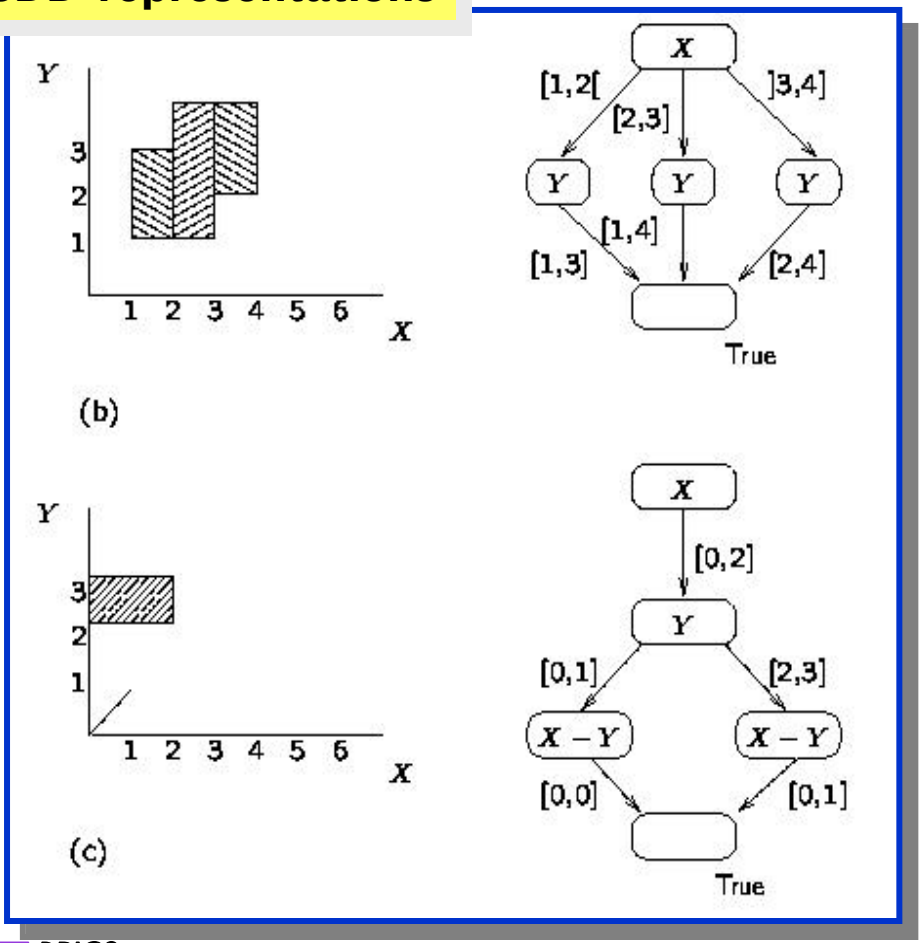
BRICS Basic Research in Computer Science

Clock Difference Diagrams

= Binary Decision Diagrams + Difference Bounded Matrices

CAV99

CDD-representations

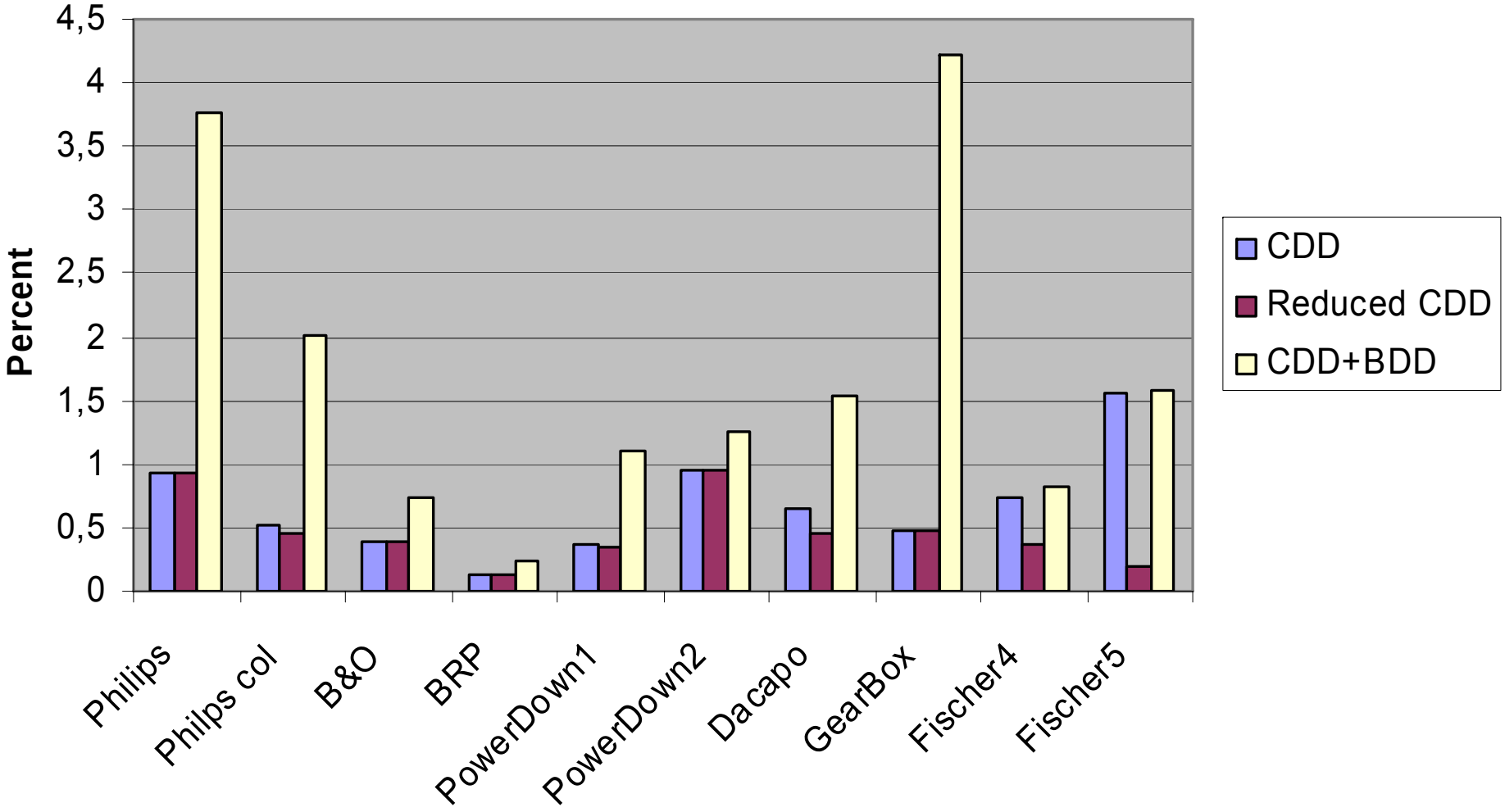


- Nodes labeled with differences
- Maximal sharing of substructures (also across different CDDs)
- Maximal intervals
- Linear-time algorithms for set-theoretic operations.

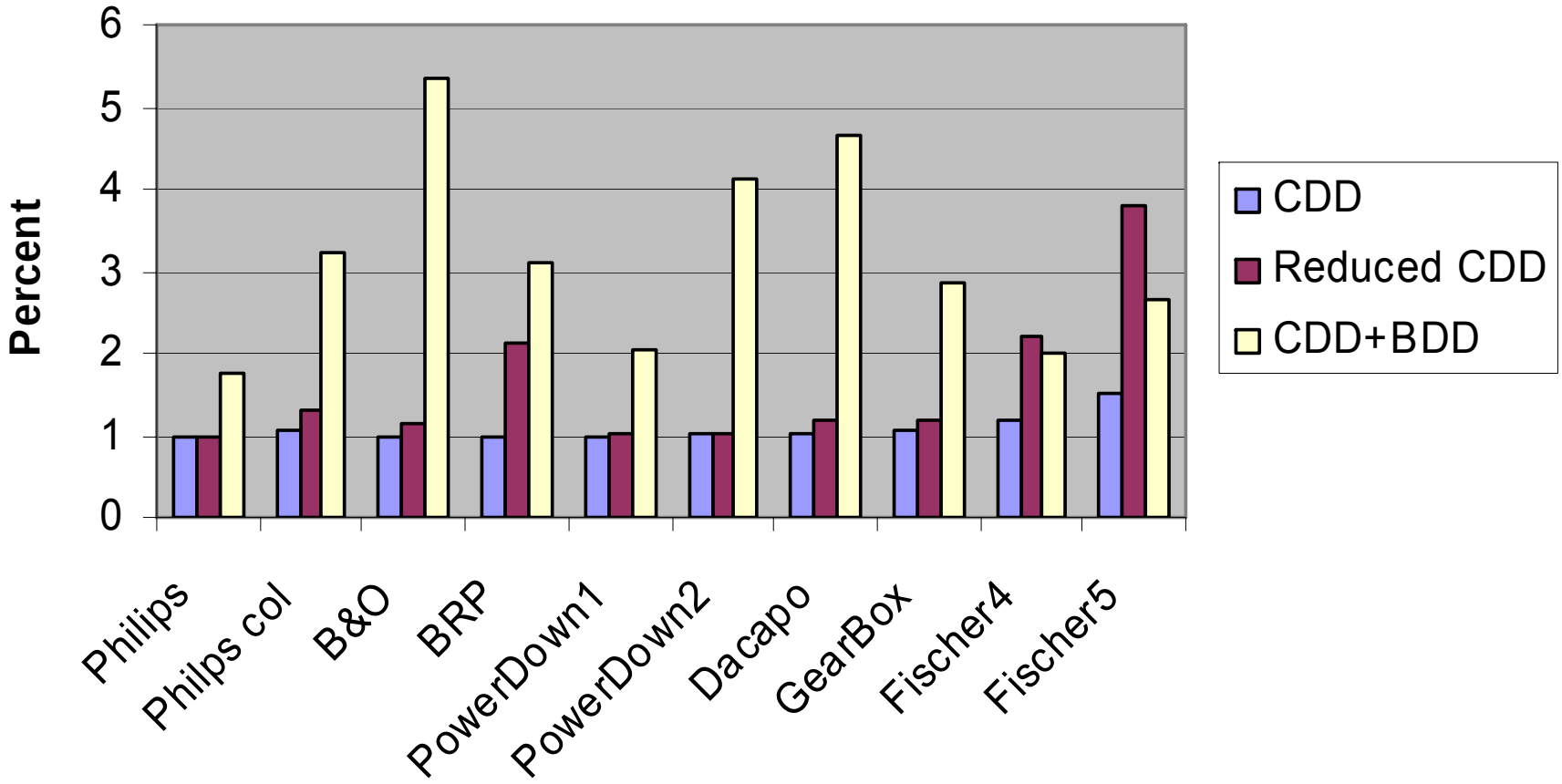
- NDD's Maler et. al
- DDD's Møller, Lichtenberg

BRICS
Basic Research
in Computer Science

SPACE PERFORMANCE

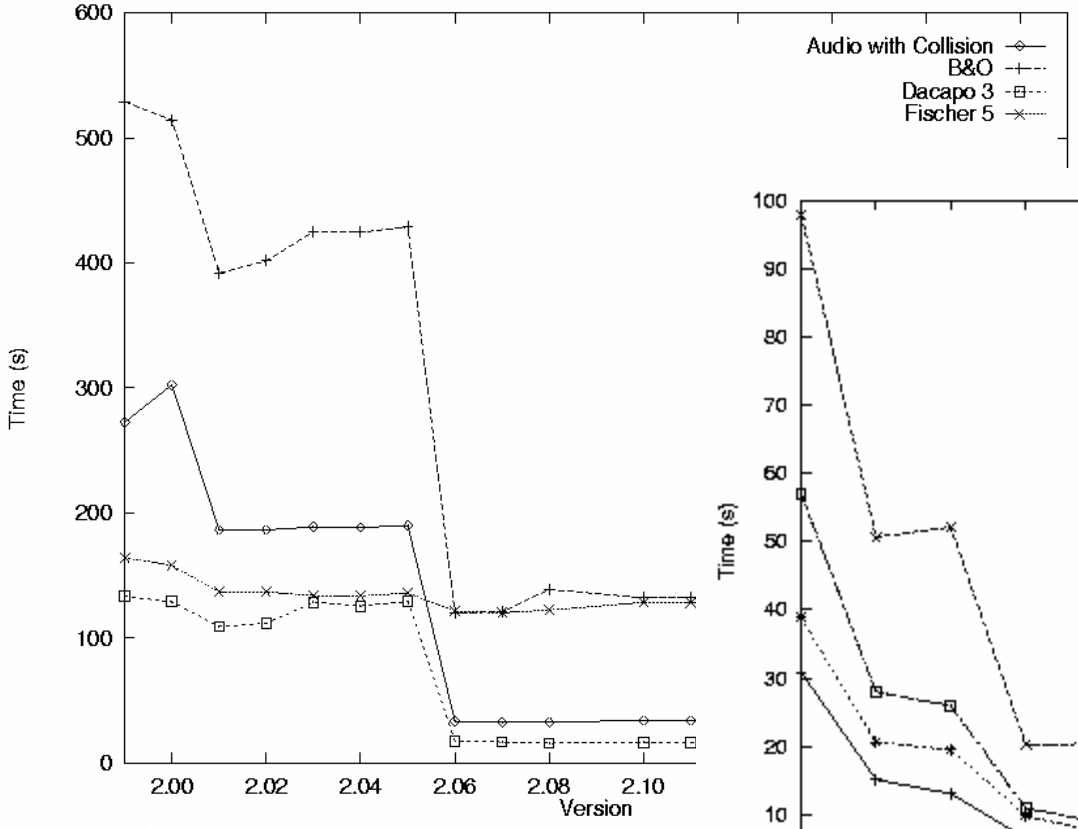


TIME PERFORMANCE

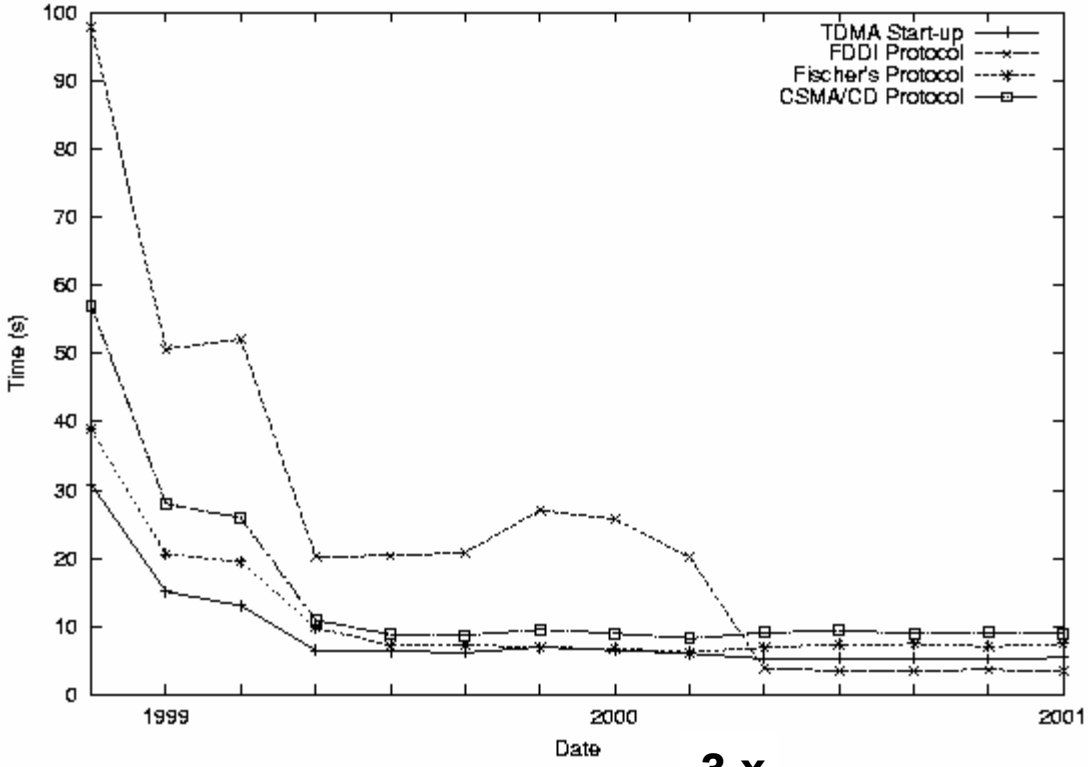


UPPAAL 1995 - 2001

Every 9 month
10 times better
performance!



Dec'96



3.x

BRICS
Basic Research
in Computer Science

Liveness Checking

BRICS
Basic Research
in Computer Science

Liveness Properties

in UPPAAL

F ::= E□ P | — **Possibly always** P

A◇ P | — **Eventually** P
is equivalent to (¬ E□ ¬ P)

P → Q — P **leads to** Q
is equivalent to
A□ (P ⇒ A◇ Q)

Bouajjani, Tripakis, Yovine'97
On-the-fly symbolic model checking of TCTL

CISS in UPPAAL

Liveness

$E[]\phi$ $(A\Diamond\neg\phi)$

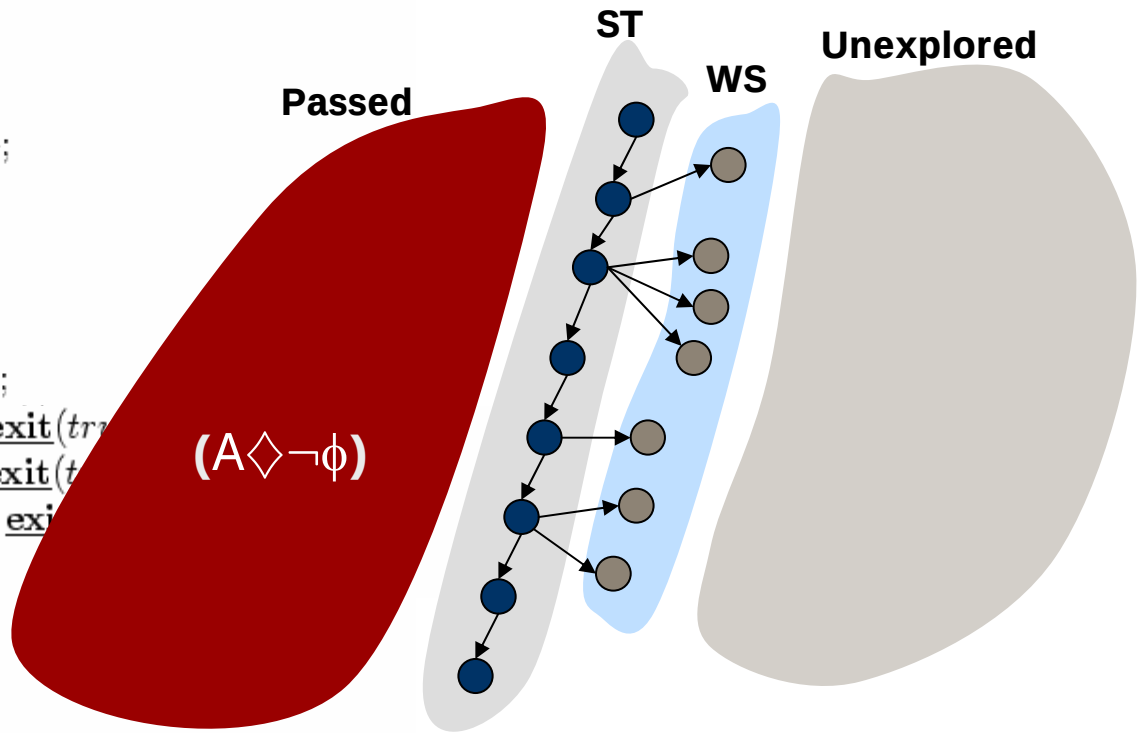
$\underline{\text{proc}}\ Liveness(s_0, \varphi, Passed)\ \equiv$
$\quad pre(s_0 = delay(s_0))$
$\quad pre(s_0 \models \varphi)$
$\quad pre(\neg unbounded s_0 \wedge \neg deadlocked(s_0))$
$\quad pre(\forall s \in Passed.\ s \models A\Diamond\neg\varphi)$
$\quad WS := \{s_0\};$
$\quad ST := \emptyset;$
$\quad \underline{\text{while}}\ WS \neq \emptyset\ \underline{\text{do}}$
$\qquad s := pop(WS);$
$\qquad \underline{\text{while}}\ top(ST) \neq parent(s)\ \underline{\text{do}}$
$\qquad\quad Passed := Passed \cup \{pop(ST)\};$
$\qquad \underline{\text{od}}$
$\qquad push(ST, s);$
$\qquad \underline{\text{if}}\ \forall s' \in Passed.\ s \not\subseteq s'$
$\qquad\quad \underline{\text{then}}\ \underline{\text{foreach}}\ t : s \stackrel{a}{\Rightarrow} t\ \underline{\text{do}}$
$\qquad\qquad \underline{\text{if}}\ t \models \varphi\ \underline{\text{then}}\ t := delay(t);$
$\qquad\qquad\quad \underline{\text{if}}\ unbounded(t)\ \underline{\text{then}}\ \underline{\text{exit}}(true)\ \underline{\text{fi}}$
$\qquad\qquad\quad \underline{\text{if}}\ deadlocked(t)\ \underline{\text{then}}\ \underline{\text{exit}}(true)\ \underline{\text{fi}}$
$\qquad\qquad\quad \underline{\text{if}}\ \exists t' \in ST.\ t = t'\ \underline{\text{then}}\ \underline{\text{exit}}(true)\ \underline{\text{fi}}$
$\qquad\qquad\quad push(WS, t);$
$\qquad\qquad \underline{\text{fi}}$
$\qquad\quad \underline{\text{od}}$
$\qquad \underline{\text{fi}}$
$\quad \underline{\text{od}}$
$\quad \underline{\text{exit}}(false);$
$\underline{\text{end}}$

BRICS
Basic Research
in Computer Science

Liveness

$E[]\phi \quad (A\Diamond\neg\phi)$

```
proc Liveness(s0, φ, Passed) ≡
  pre(s0 = delay(s0))
  pre(s0 ⊨ φ)
  pre(¬unboundeds0 ∧ ¬deadlocked(s0))
  pre(∀s ∈ Passed. s ⊨ A◇¬φ)
  WS := {s0};
  ST := ∅;
  while WS ≠ ∅ do
      s := pop(WS);
      while top(ST) ≠ parent(s) do
          Passed := Passed ∪ {pop(ST)};
      od
      push(ST, s);
      if ∀s' ∈ Passed. s ⋢ s'
        then foreach t : s ⇒a t do
            if t ⊨ φ then t := delay(t);
              if unbounded(t) then exit(tr
              if deadlocked(t) then exit(t
              if ∃t' ∈ ST. t = t' then exi
              push(WS, t);
            fi
          od
      fi
  od
  exit(false);
end
```

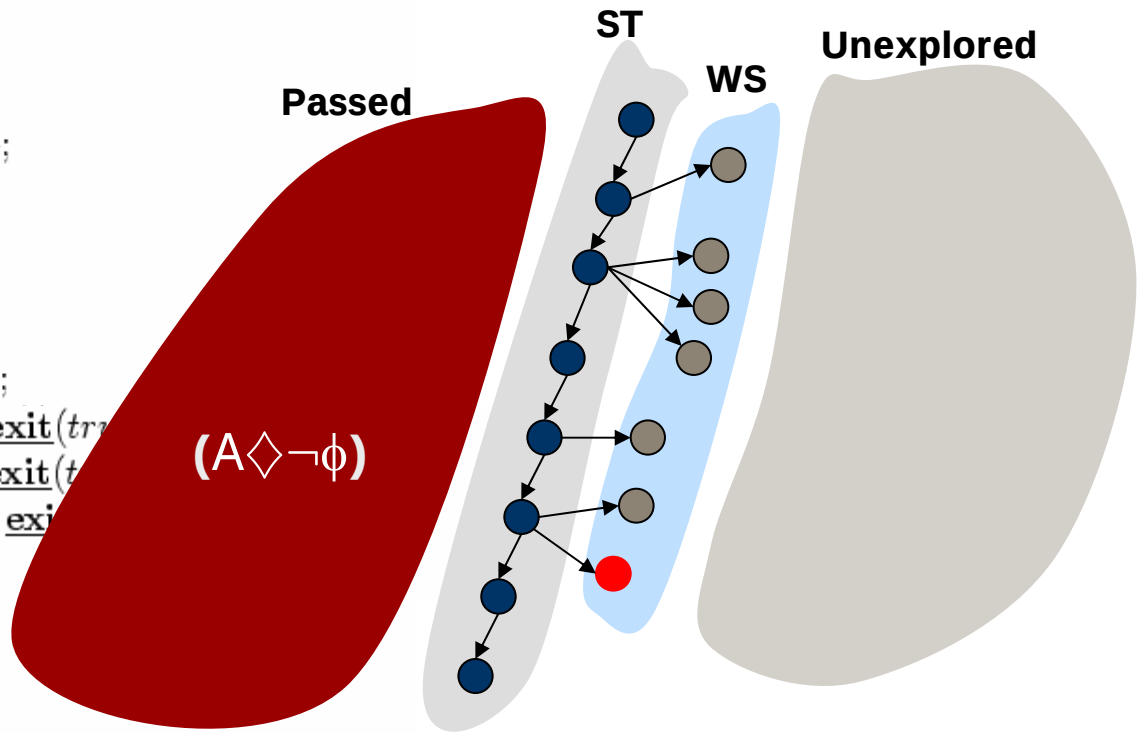


BRICS
Basic Research
in Computer Science

Liveness

$E\Box\phi \quad (A\Diamond\neg\phi)$

```
proc Liveness(s0, φ, Passed) ≡
  pre(s0 = delay(s0))
  pre(s0 ⊨ φ)
  pre(¬unboundeds0 ∧ ¬deadlocked(s0))
  pre(∀s ∈ Passed. s ⊨ A◇¬φ)
  WS := {s0};
  ST := ∅;
  while WS ≠ ∅ do
    ● s := pop(WS);
      while top(ST) ≠ parent(s) do
            Passed := Passed ∪ {pop(ST)};
      od
      push(ST, s);
      if ∀s' ∈ Passed. s ⊈ s'
        then foreach t : s ⇒a t do
               if t ⊨ φ then t := delay(t);
                 if unbounded(t) then exit(tr
                 if deadlocked(t) then exit(t
                 if ∃t' ∈ ST. t = t' then exi
                 push(WS, t);
               fi
             od
      fi
  od
  exit(false);
end
```

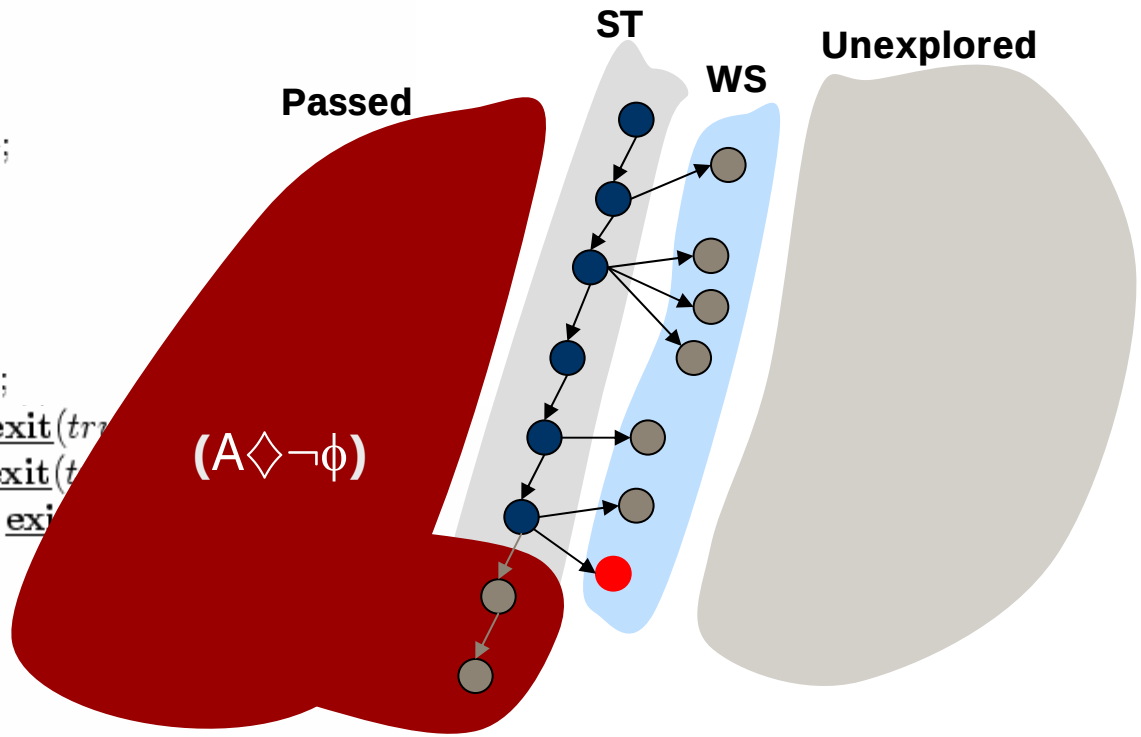


BRICS
Basic Research
in Computer Science

Liveness

$E\Box\phi \quad (A\Diamond\neg\phi)$

```
proc Liveness(s0, φ, Passed) ≡
  pre(s0 = delay(s0))
  pre(s0 ⊨ φ)
  pre(¬unboundeds0 ∧ ¬deadlocked(s0))
  pre(∀s ∈ Passed. s ⊨ A◇¬φ)
  WS := {s0};
  ST := ∅;
  while WS ≠ ∅ do
      s := pop(WS);
      while top(ST) ≠ parent(s) do
          Passed := Passed ∪ {pop(ST)};
      od
      push(ST, s);
      if ∀s' ∈ Passed. s ⋢ s'
        then foreach t : s ⇒^a t do
            if t ⊨ φ then t := delay(t);
              if unbounded(t) then exit(tr
              if deadlocked(t) then exit(t
              if ∃t' ∈ ST. t = t' then exi
              push(WS, t);
            fi
          od
      fi
  od
  exit(false);
end
```

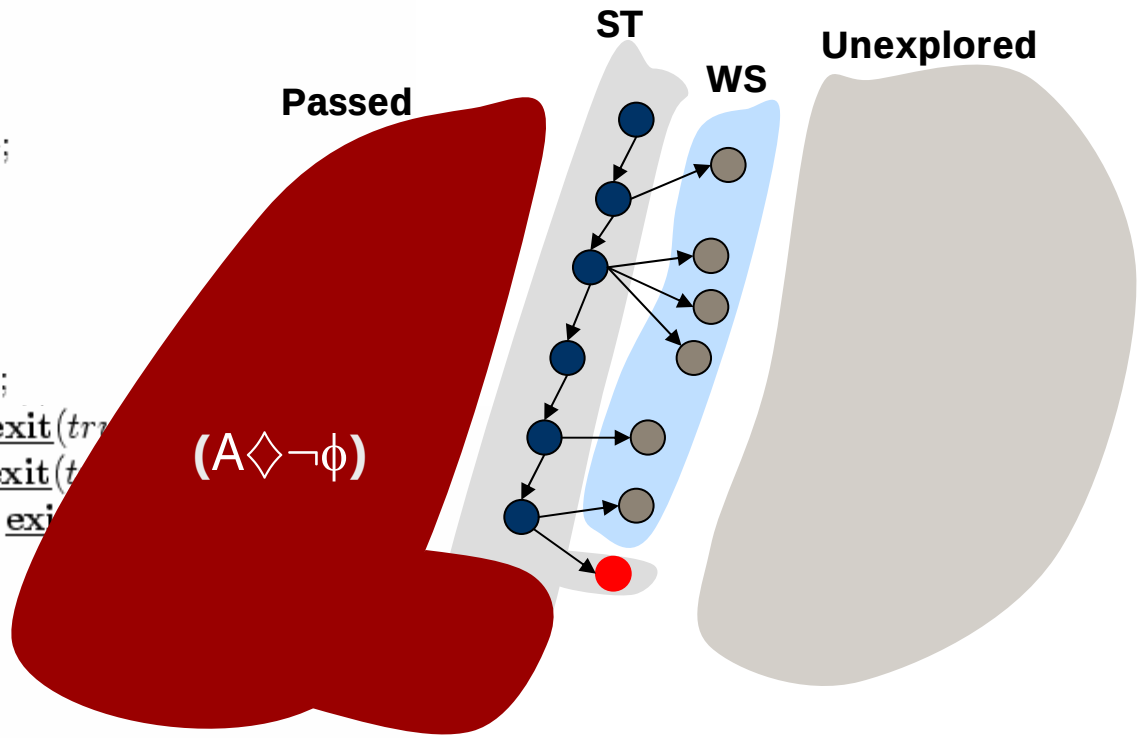


BRICS
Basic Research
in Computer Science

Liveness

$E\Box\phi \quad (A\Diamond\neg\phi)$

```
proc Liveness(s0, φ, Passed) ≡
  pre(s0 = delay(s0))
  pre(s0 ⊨ φ)
  pre(¬unboundeds0 ∧ ¬deadlocked(s0))
  pre(∀s ∈ Passed. s ⊨ A◇¬φ)
  WS := {s0};
  ST := ∅;
  while WS ≠ ∅ do
      s := pop(WS);
      while top(ST) ≠ parent(s) do
          Passed := Passed ∪ {pop(ST)};
      od
      push(ST, s);
      if ∀s' ∈ Passed. s ⊈ s'
        then foreach t : s ⇒a t do
            if t ⊨ φ then t := delay(t);
              if unbounded(t) then exit(tr
              if deadlocked(t) then exit(t
              if ∃t' ∈ ST. t = t' then exi
              push(WS, t);
            fi
          od
      fi
  od
  exit(false);
end
```

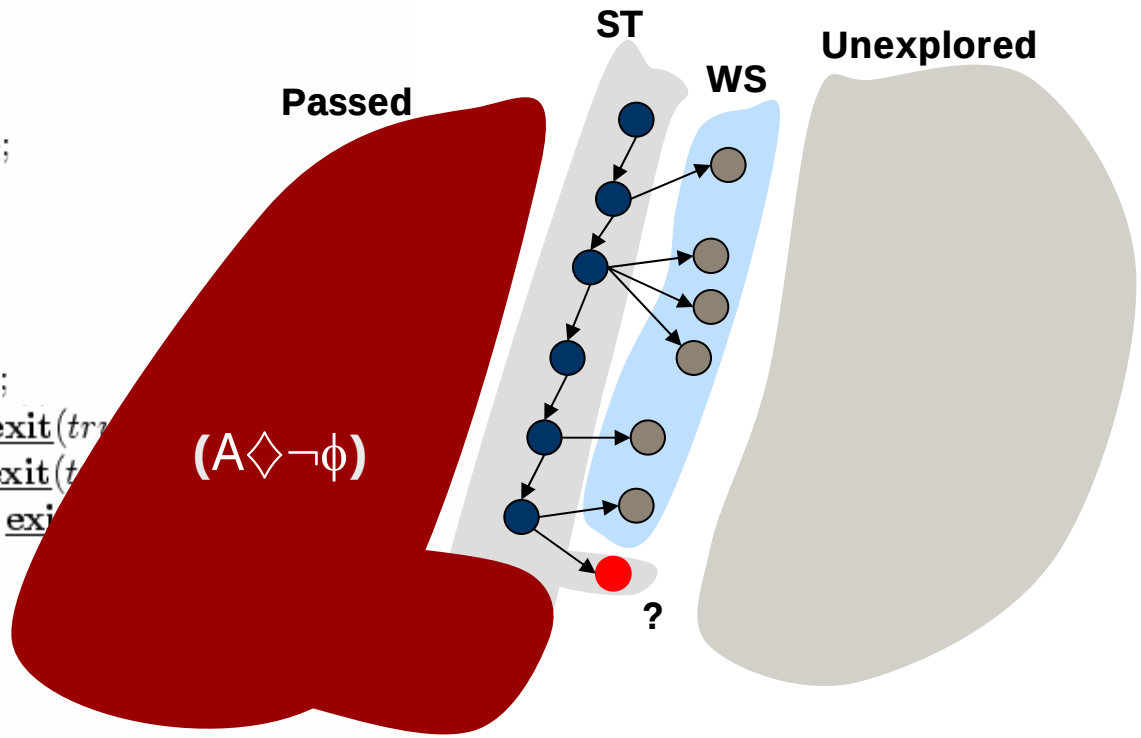


BRICS
Basic Research
in Computer Science

Liveness

$E\Box\phi \quad (A\Diamond\neg\phi)$

```
proc Liveness(s0, φ, Passed) ≡
  pre(s0 = delay(s0))
  pre(s0 ⊨ φ)
  pre(¬unboundeds0 ∧ ¬deadlocked(s0))
  pre(∀s ∈ Passed. s ⊨ A◇¬φ)
  WS := {s0};
  ST := ∅;
  while WS ≠ ∅ do
      s := pop(WS);
      while top(ST) ≠ parent(s) do
          Passed := Passed ∪ {pop(ST)};
      od
      push(ST, s);
    ● if ∀s' ∈ Passed. s ⋢ s'
        then foreach t : s ⇒^a t do
            if t ⊨ φ then t := delay(t);
              if unbounded(t) then exit(tr
              if deadlocked(t) then exit(t
              if ∃t' ∈ ST. t = t' then exi
              push(WS, t);
            fi
          od
      fi
  od
  exit(false);
end
```

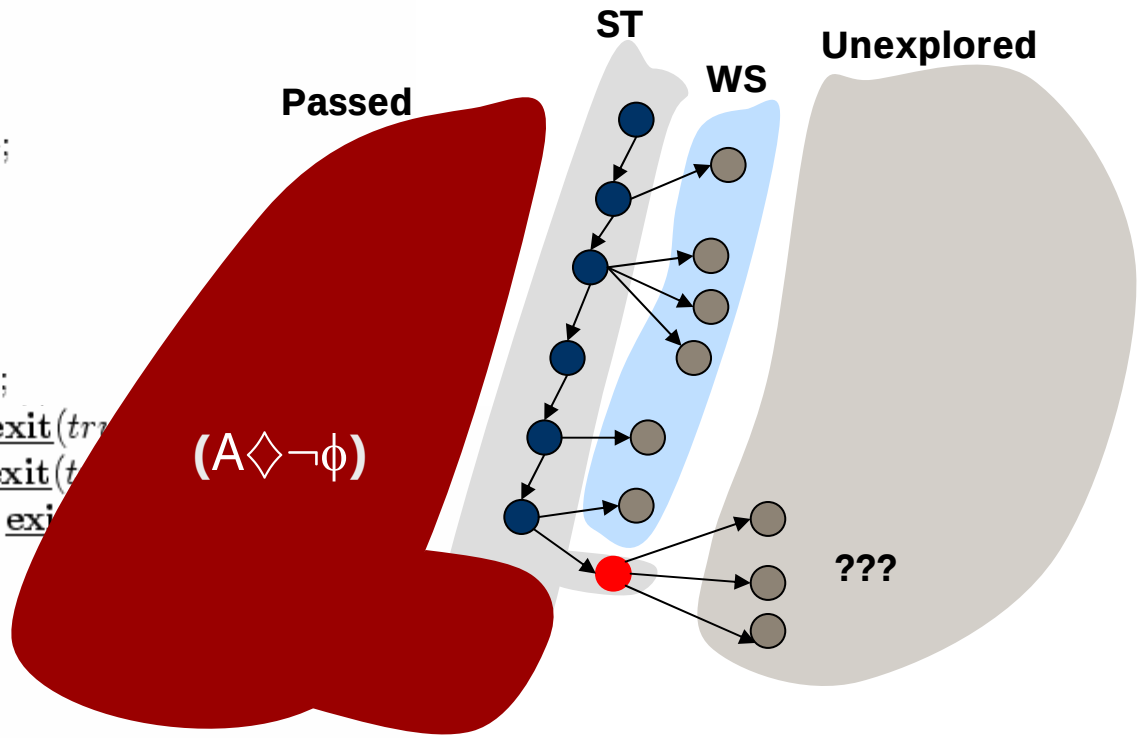


BRICS
Basic Research
in Computer Science

Liveness

$E\Box\phi \quad (A\Diamond\neg\phi)$

```
proc Liveness(s0, φ, Passed) ≡
  pre(s0 = delay(s0))
  pre(s0 ⊨ φ)
  pre(¬unboundeds0 ∧ ¬deadlocked(s0))
  pre(∀s ∈ Passed. s ⊨ A◇¬φ)
  WS := {s0};
  ST := ∅;
  while WS ≠ ∅ do
      s := pop(WS);
      while top(ST) ≠ parent(s) do
          Passed := Passed ∪ {pop(ST)};
      od
      push(ST, s);
      if ∀s' ∈ Passed. s ⋢ s'
      then foreach t : s ⇒a t do
          if t ⊨ φ then t := delay(t);
              if unbounded(t) then exit(tr
              if deadlocked(t) then exit(t
              if ∃t' ∈ ST. t = t' then exi
              push(WS, t);
          fi
      od
      fi
  od
  exit(false);
end
```

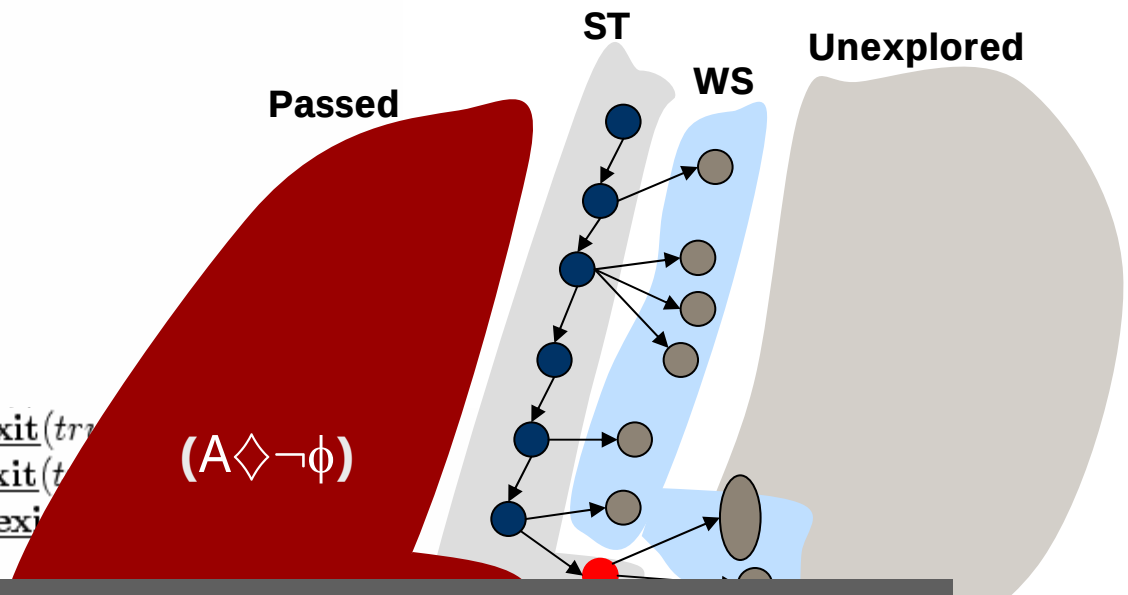


BRICS
Basic Research
in Computer Science

Liveness

$E\Box\phi \quad (A\Diamond\neg\phi)$

```
proc Liveness(s0, φ, Passed) ≡
  pre(s0 = delay(s0))
  pre(s0 ⊨ φ)
  pre(¬unboundeds0 ∧ ¬deadlocked(s0))
  pre(∀s ∈ Passed. s ⊨ A◇¬φ)
  WS := {s0};
  ST := ∅;
  while WS ≠ ∅ do
      s := pop(WS);
      while top(ST) ≠ parent(s) do
          Passed := Passed ∪ {pop(ST)};
      od
      push(ST, s);
      if ∀s' ∈ Passed. s ⋢ s'
        then foreach t : s ⇒^a t do
            if t ⊨ φ then t := delay(t);
              if unbounded(t) then exit(tr
              if deadlocked(t) then exit(t
              if ∃t' ∈ ST. t = t' then exi
              push(WS, t);
            fi
          od
      fi
  od
  exit(false);
end
```



[FORMATS05]
Extensions allowing for automatic synthesis of smallest bound t such that $A\Diamond_{\leq t}\phi$ holds

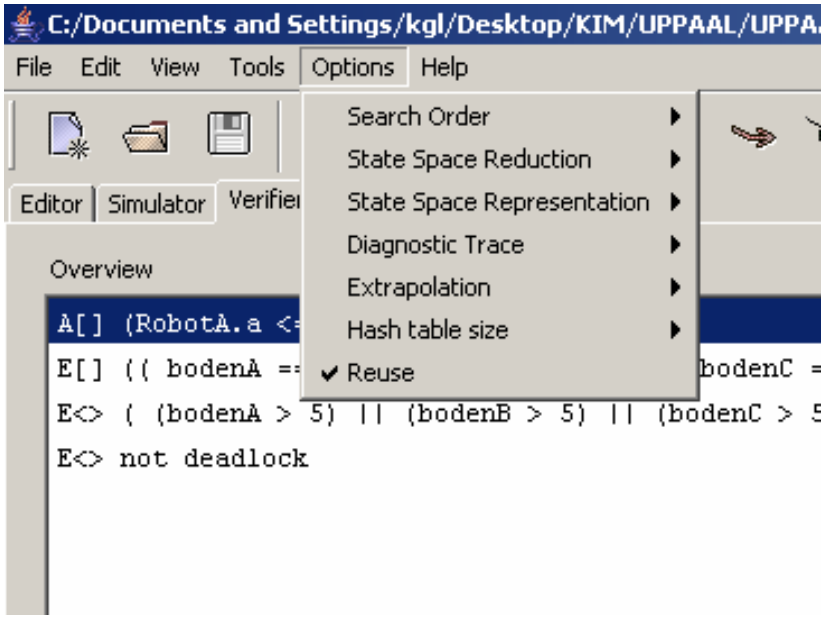
BRICS
Basic Research in Computer Science

Verification Options

BRICS
Basic Research
in Computer Science

CSS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Verification Options



Search Order
- Depth First
- Breadth First

State Space Reduction
- None
- Conservative
- Aggressive

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

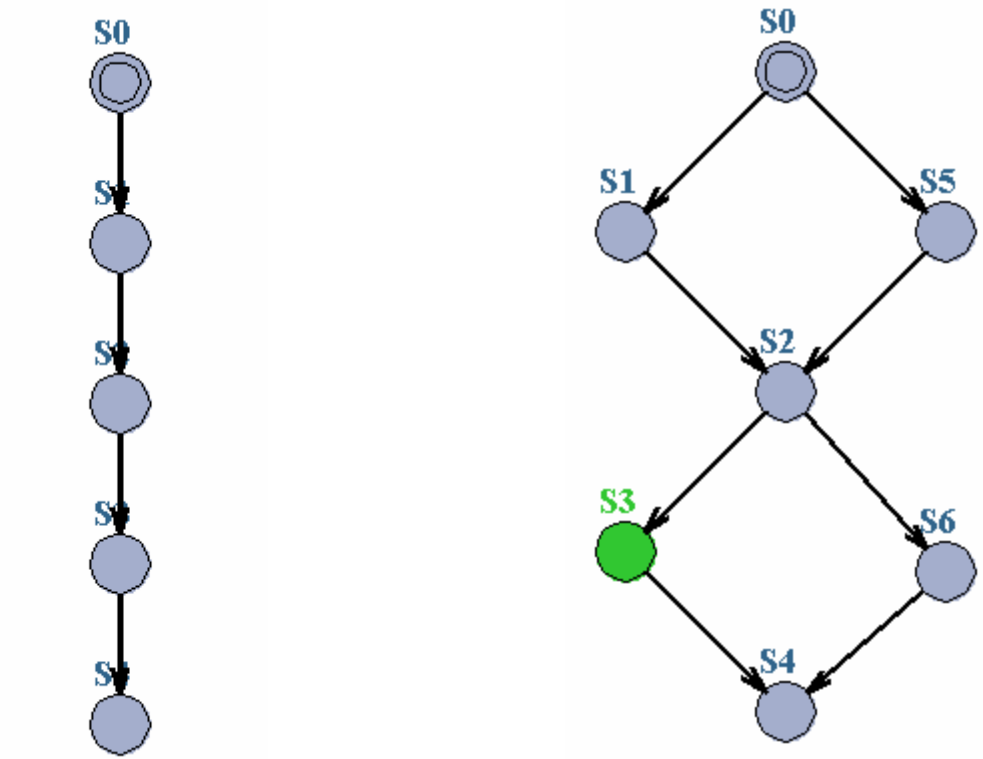
Diagnostic Trace
- Some
- Shortest
- Fastest

Extrapolation

Hash Table size

Reuse

State Space Reduction

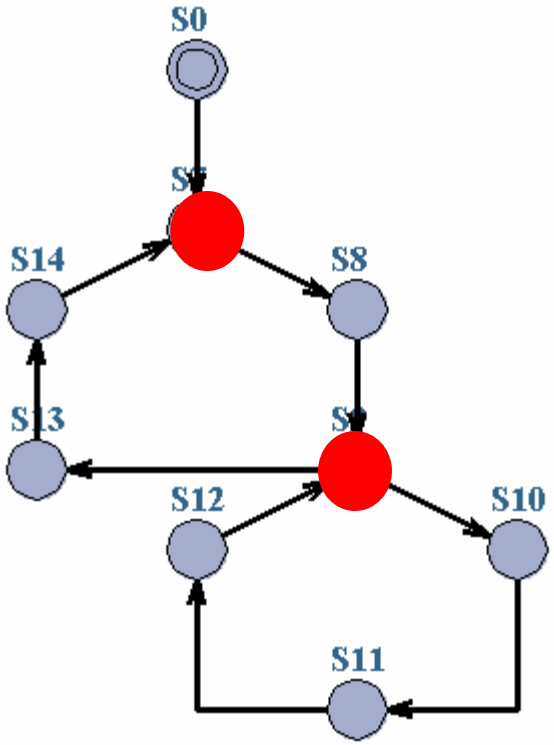


However,
Passed list useful for efficiency

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

BRICS
Basic Research
in Computer Science

State Space Reduction



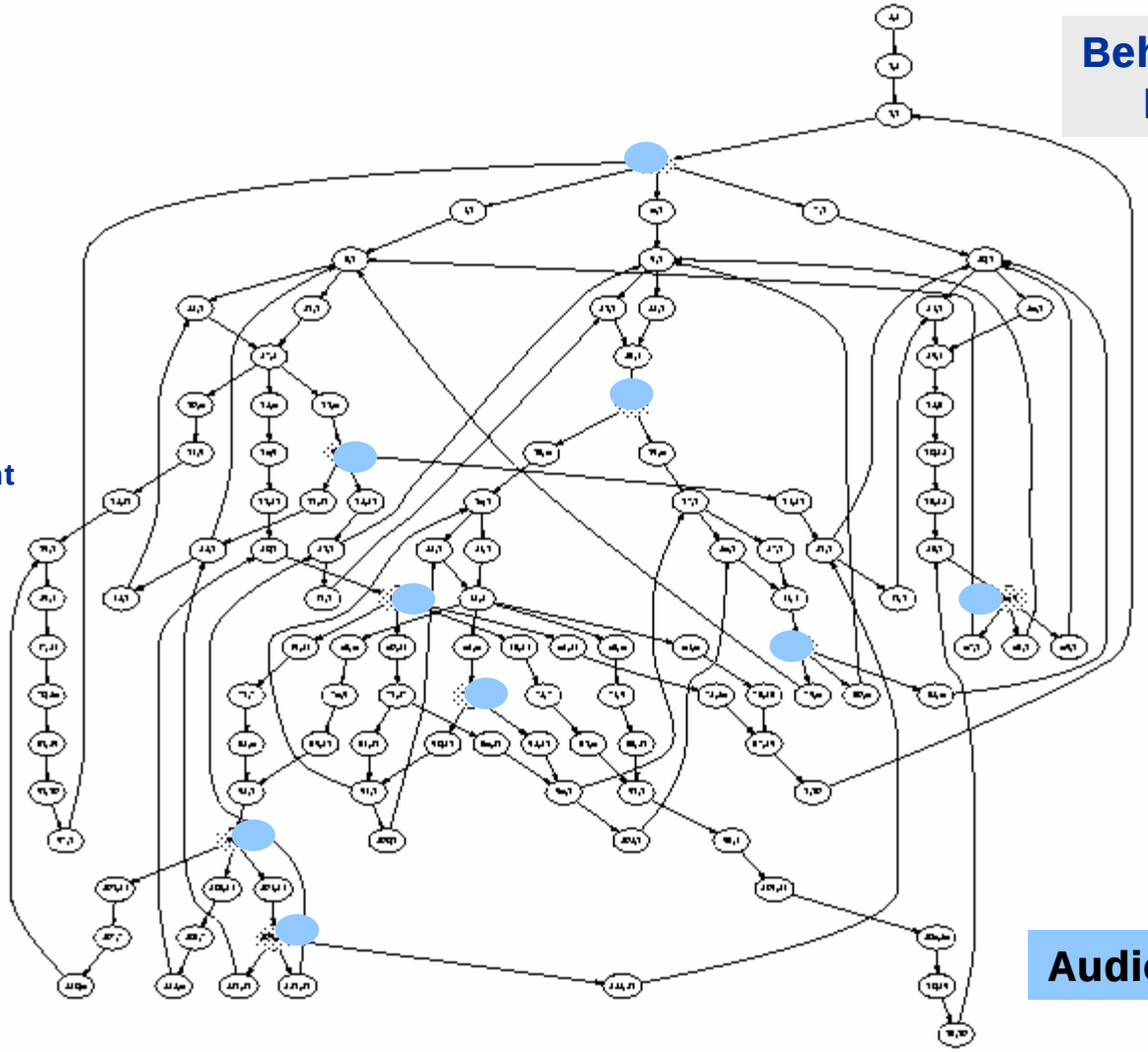
Cycles:

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

BRICS
Basic Research in Computer Science

To Store or Not to Store

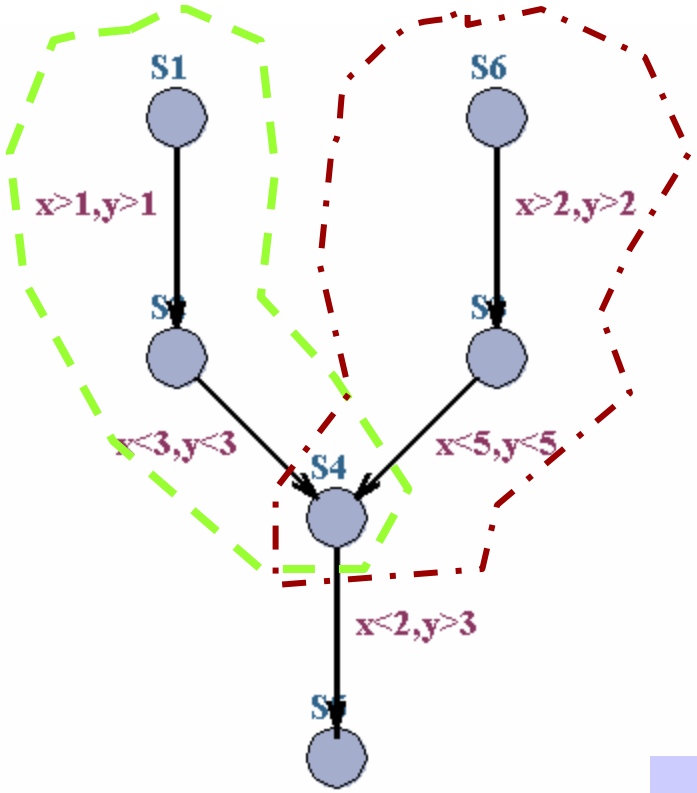
Behrmann, Larsen, Pelanek 2003

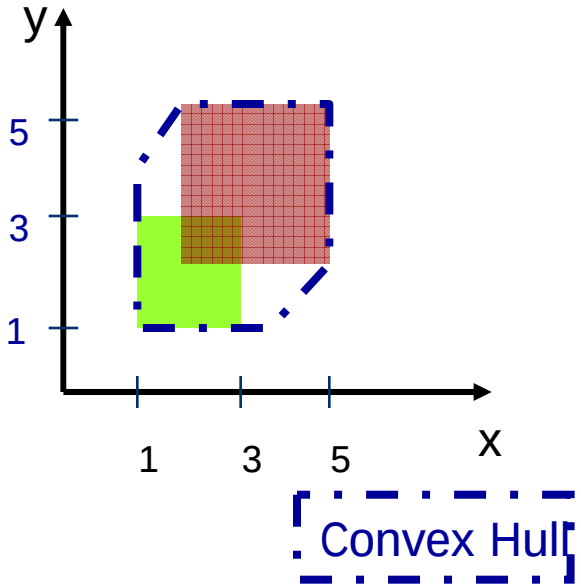
	entry points	covering set	successors	random $p = 0.1$	distance $k = 10$	combination $k = 3$
Fischer	27.1%	42.1%	47.9%	53.7%	67.6%	56.9%
3,077	1.00	1.66	1.00	4.51	2.76	6.57
BRP	70.5%	16.5%	19.8%	18.3%	15.8%	7.6%
6,060	1.01	1.20	1.03	1.78	1.34	1.68
Token Ring	33.0%	10.3%	20.7%	17.2%	17.5%	16.8%
15,103	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

BRICS
Basic Research
in Computer Science

Over-approximation

Convex Hull



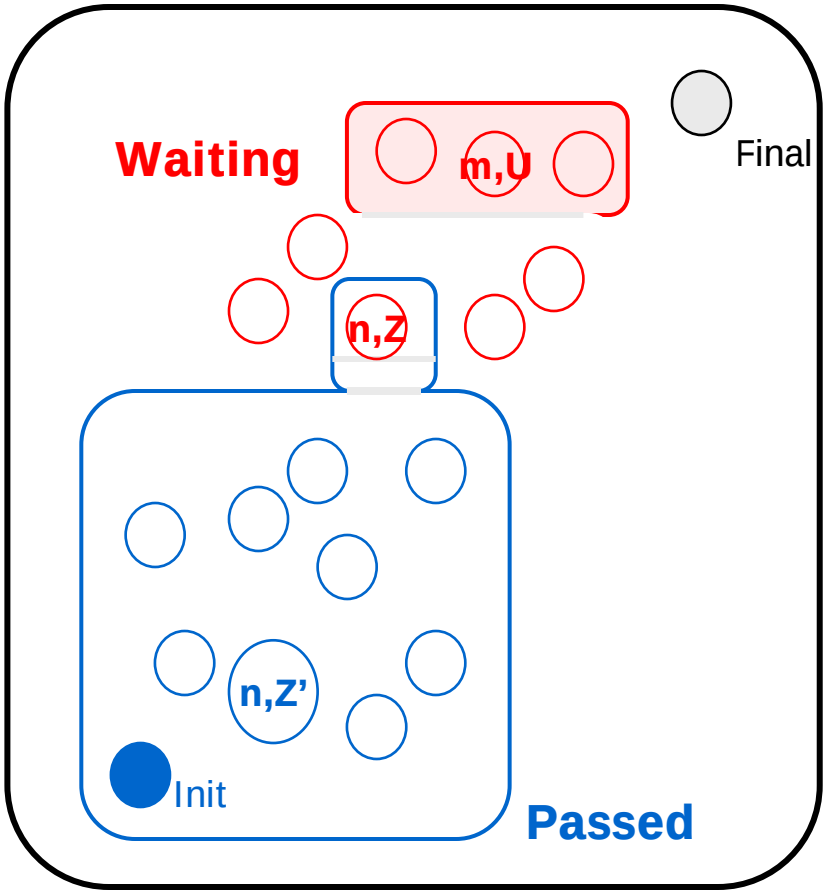


TACAS04: An **EXACT** method performing as well as Convex Hull has been developed based on abstractions taking max constants into account distinguishing between clocks, locations and $\leq$ & $\geq$

BRICS
Basic Research
in Computer Science

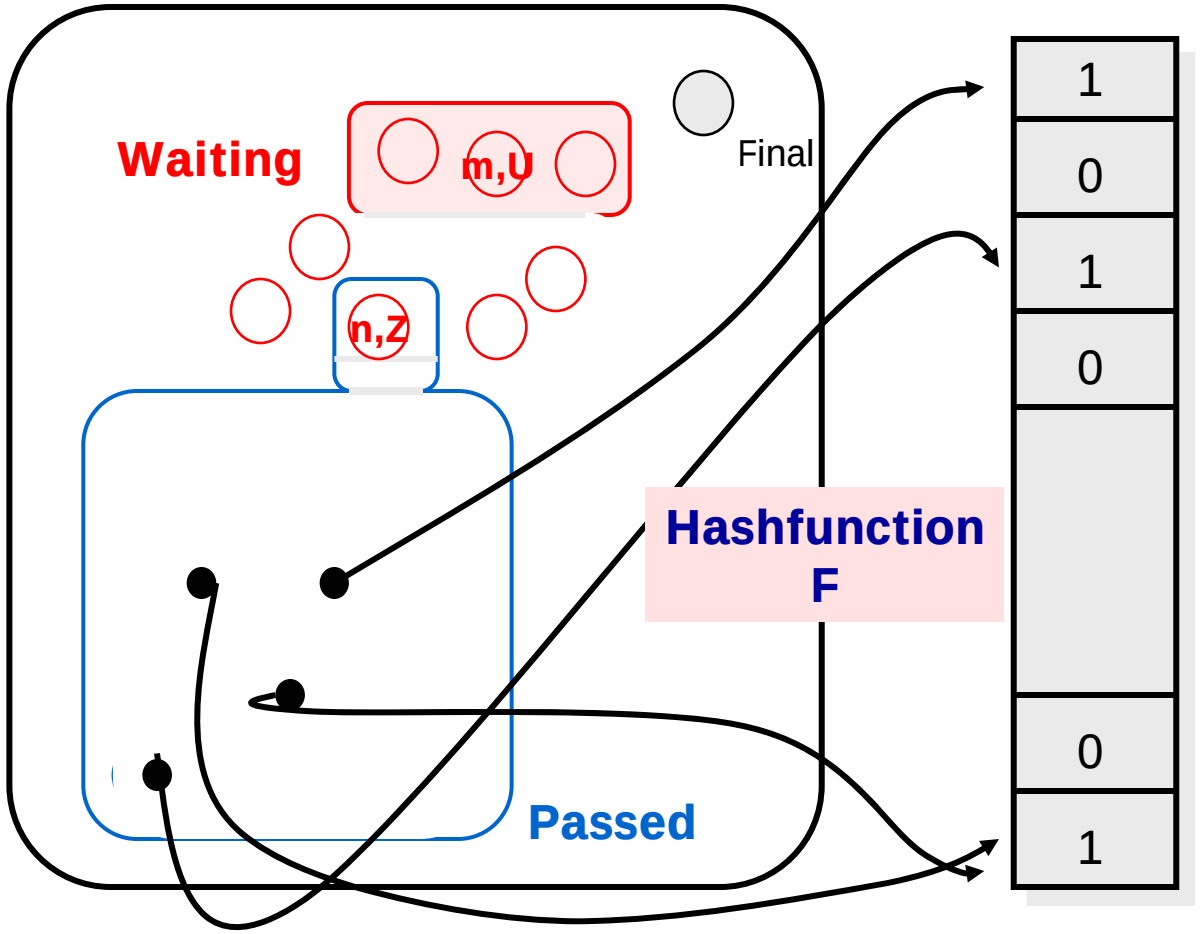
Under-approximation

Bitstate Hashing



Under-approximation

Bitstate Hashing



Passed =
Bitarray

UPPAAL
8 Mbits

BRICS
Basic Research
in Computer Science

Modelling Patterns

BRICS
Basic Research
in Computer Science

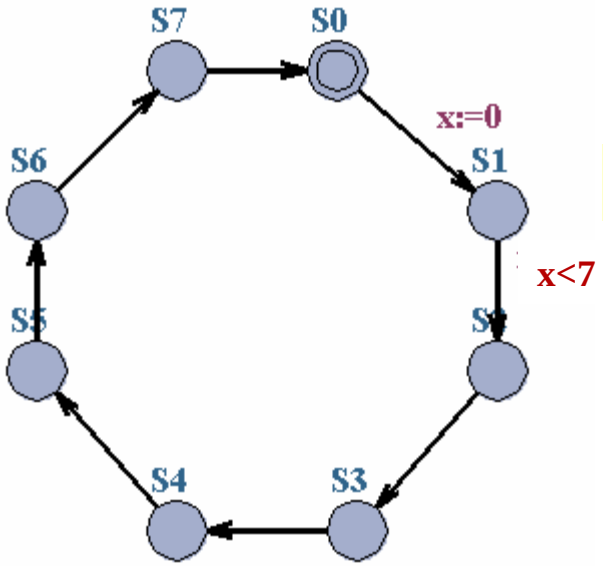
CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Variable Reduction

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

```
// Remove the front element of the queue
void dequeue()
{
        int i = 0;
        len -= 1;
        while (i < len)
        {
                list[i] = list[i + 1];
                i++;
        }
        list[i] = 0;
}
```

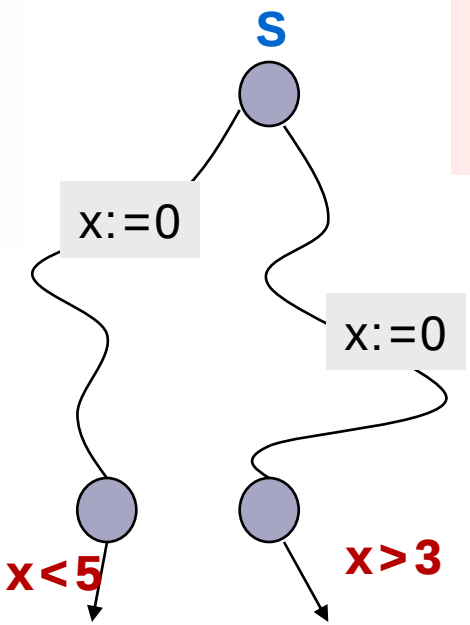
Clock Reduction



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

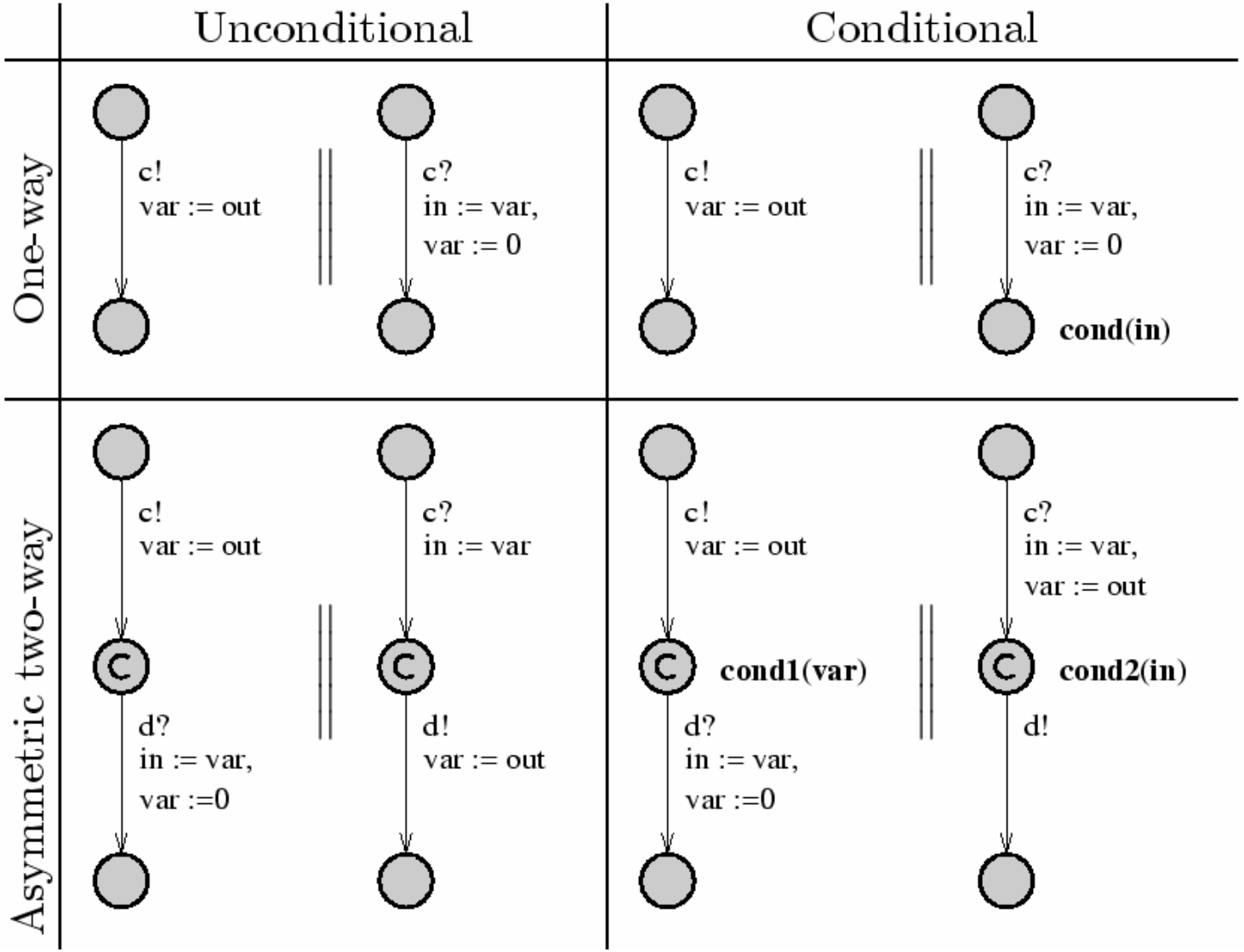
Definition

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



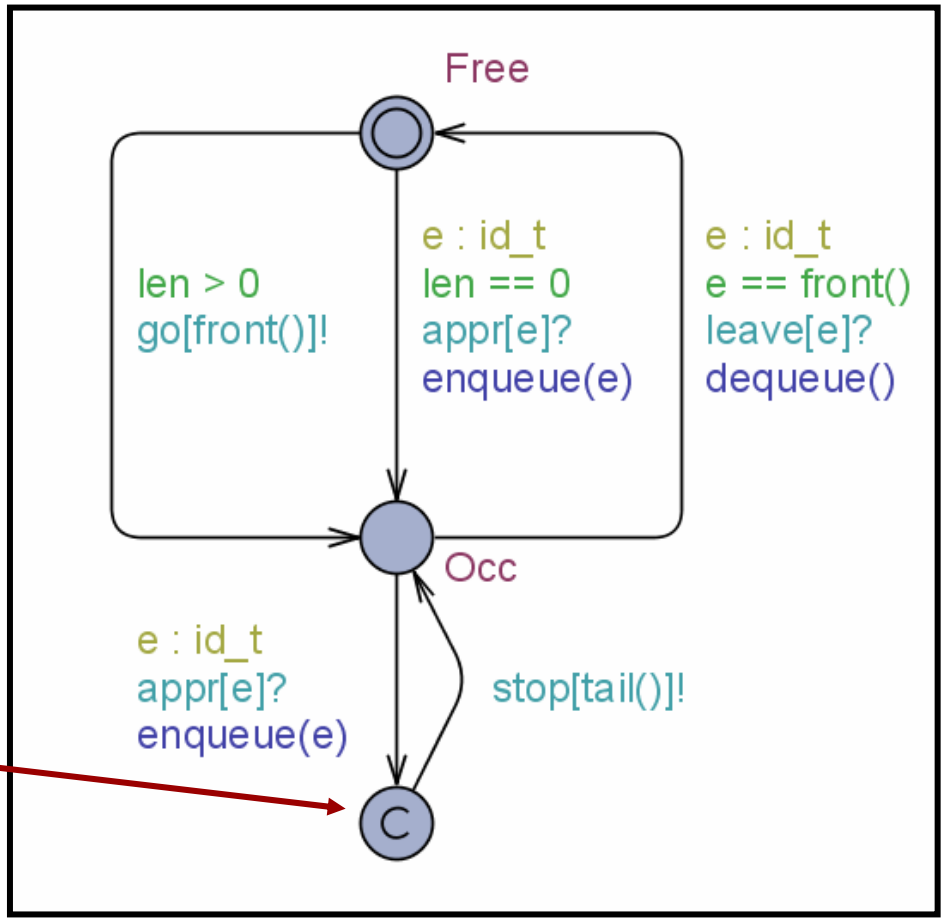
BRICS
Basic Research
in Computer Science

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.

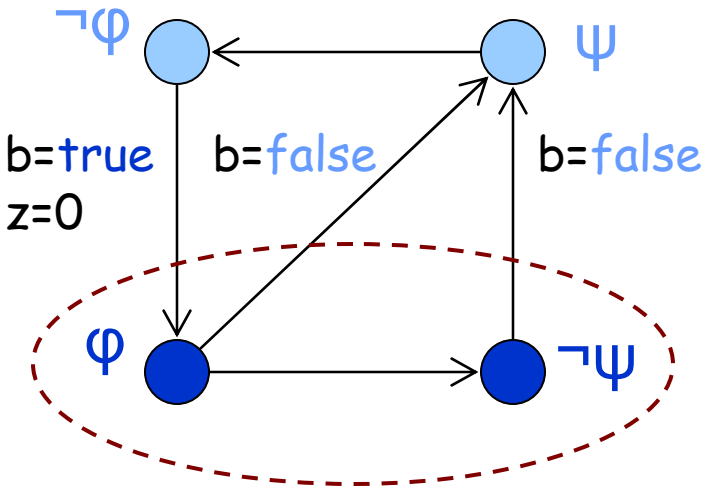


Bounded Liveness

- Intent: Check for properties that are guaranteed to hold eventually within some upper (time) bound.
  - Provide additional information (with a valid bound).
  - More efficient verification.
  - φ leadsto$_{\leq t}$ ψ reduced to $A\Box(b \Rightarrow z \leq t)$ with bool b set to *true* and clock z reset when φ starts to hold. When ψ starts to hold, set b to *false*.

Bounded Liveness

- The truth value of b indicates whether or not ψ should hold in the future.

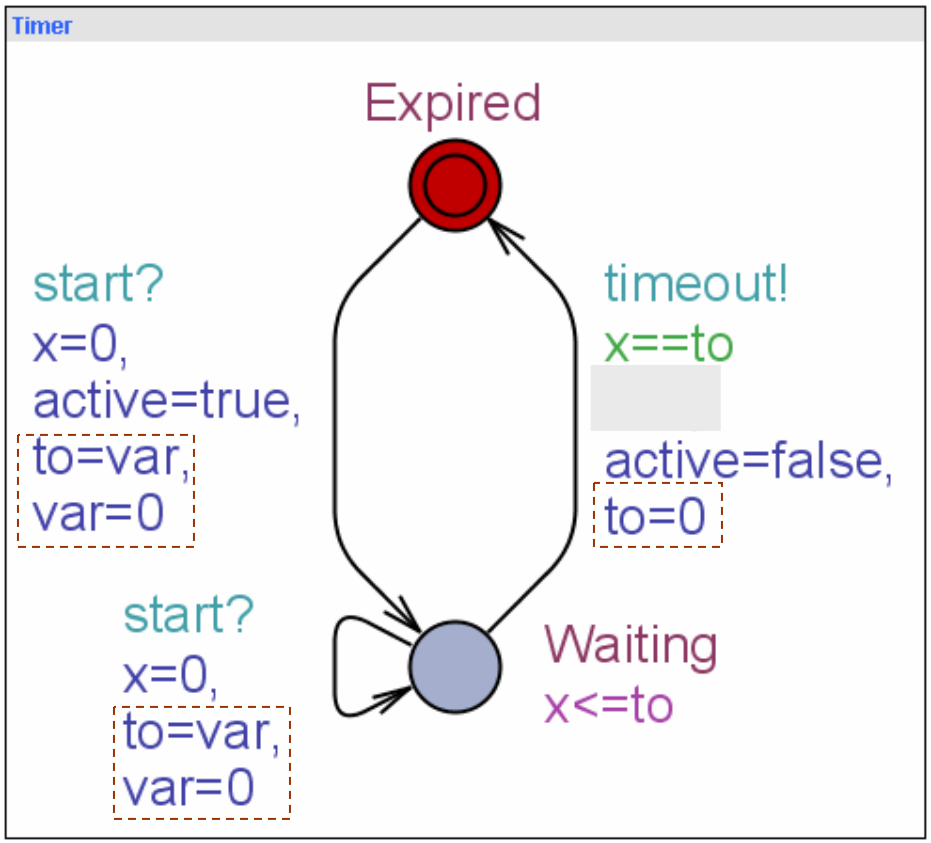


A[] (b imply z≤t)
b --> not b (for non zenoness)
E<> b (for meaningful check)

Timers

Parametric timer:

- (re-)start(value)
  `start! var=value`
- expired?
  `active` (bool)
  `active go?`
  (bool+urgent chan)
- time-out event
  `timeout?`



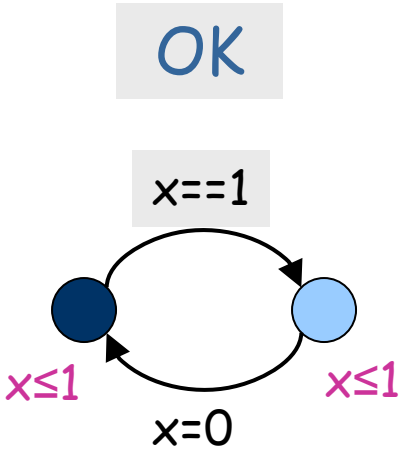
Declare 'to' with a tight range.

Zenoness

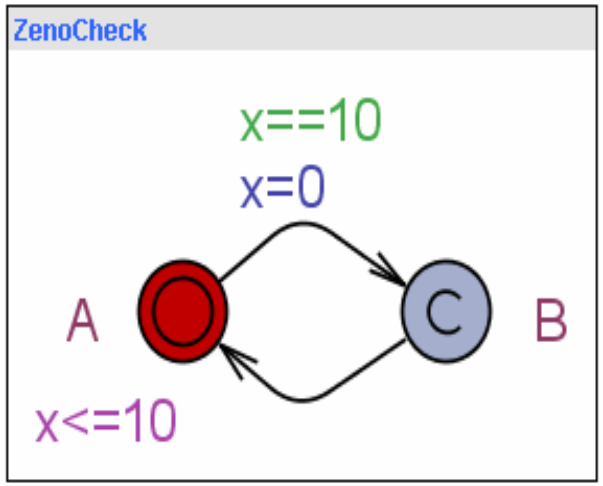
- Problem: UPPAAL does not check for zenoness directly.
  - A model has "zeno" behavior if it can take an infinite amount of actions in finite time.
  - That is usually not a desirable behavior in practice.
  - Zeno models may wrongly conclude that some properties hold though they logically should not.
  - Rarely taken into account.

- Solution: Add an observer automata and check for non-zenoness, i.e., that time will always pass.

Zenoness



Detect by
•adding the observer:



Constant (10) can be anything (>0), but choose it well w.r.t. your model for efficiency. Clocks 'x' are local.

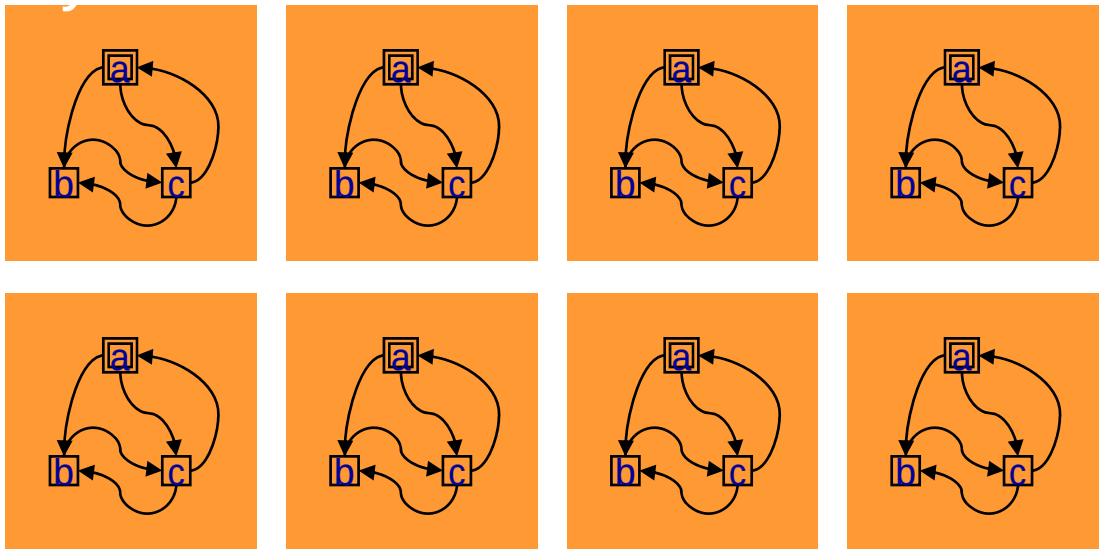
•and check the property
`ZenoCheck.A --> ZenoCheck.B`

Compositionality & Abstraction

BRICS
Basic Research
in Computer Science

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

The State Explosion Problem



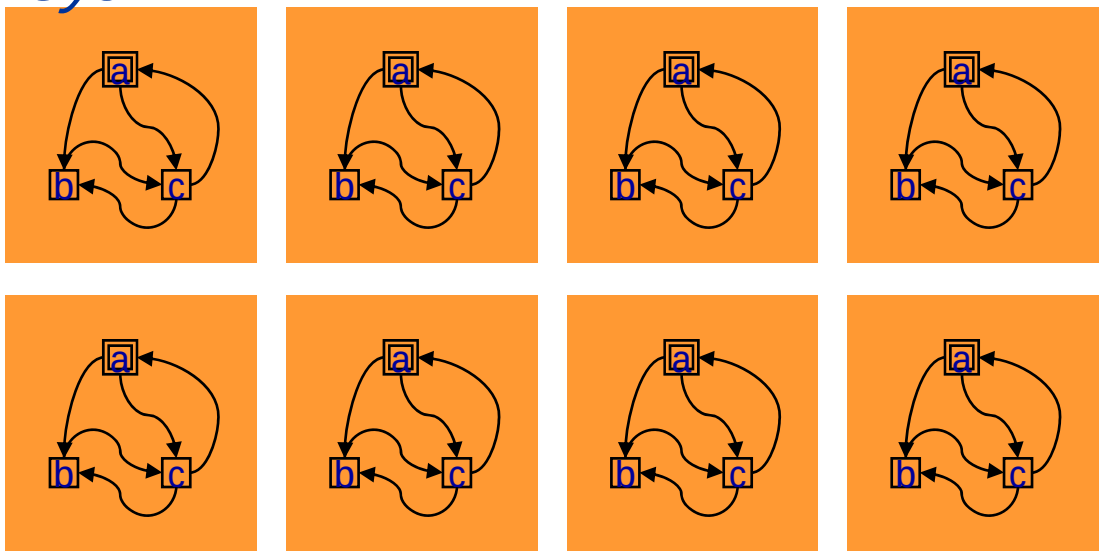
$sat\ \varphi$

Model-checking is either
EXPTIME-complete or PSPACE-complete
(for TA's this is true even for a single TA)

BRICS
Basic Research
in Computer Science

Abstraction

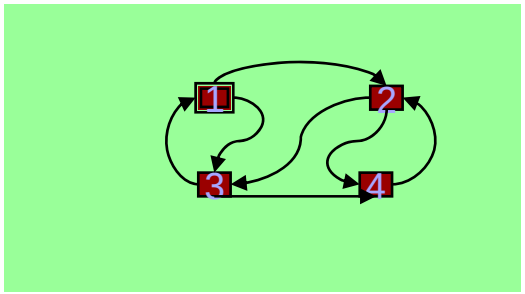
Sys



$sat \ \varphi$

REDUCE TO

Abs



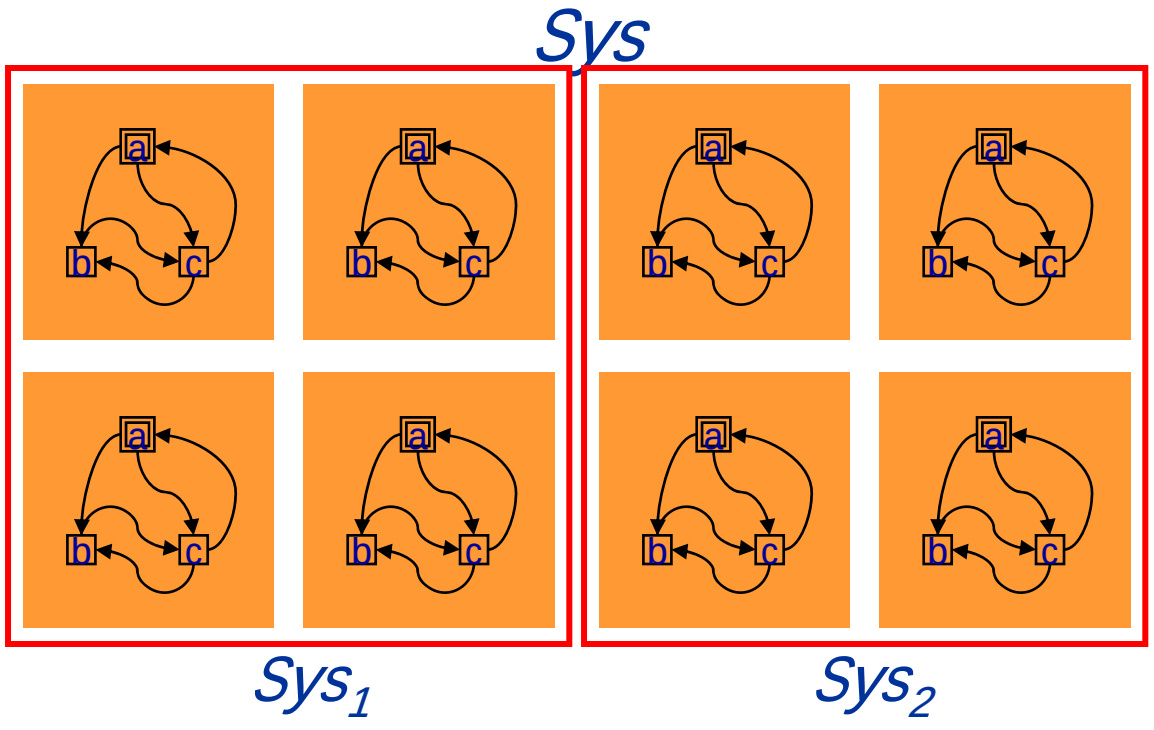
$sat \ \varphi$

Preserving safety properties

$$\frac{Abs \ sat \ \varphi \qquad Sys \leq Abs}{Sys \ sat \ \varphi}$$

BRICS
Basic Research
in Computer Science

Compositionality

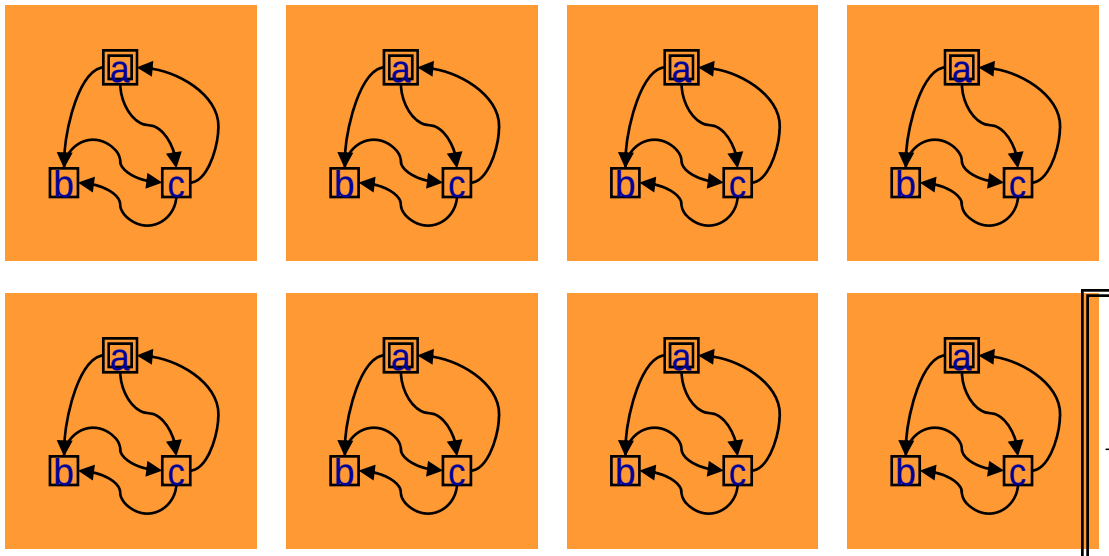


$$\frac{Sys_1 \le Abs_1 \quad Sys_2 \le Abs_2}{Sys_1 \mid Sys_2 \le Abs_1 \mid Abs_2}$$

$$\frac{Sys_1 \le Abs_1 \quad Sys_2 \le Abs_2 \quad Abs_1 \mid Abs_2 \le Abs}{Sys \le Abs}$$

Abstraction & Compositionality

dealing w stateexplosion

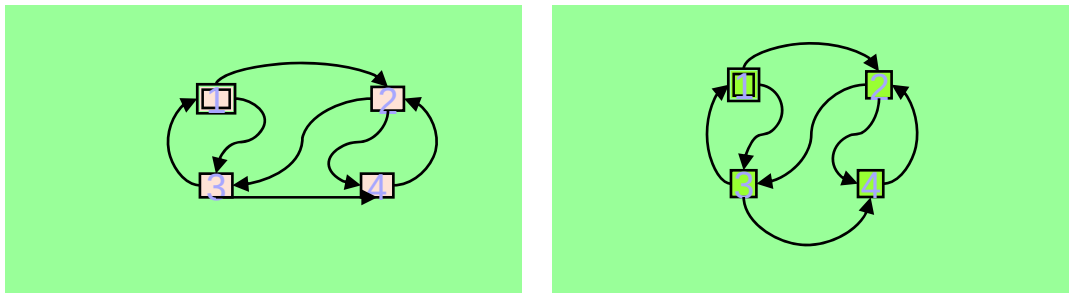


"trace" inclusion

$$\frac{C_1 \prec A_1 \qquad C_2 \prec A_2}{C_1 | C_2 \prec A_1 | A_2}$$

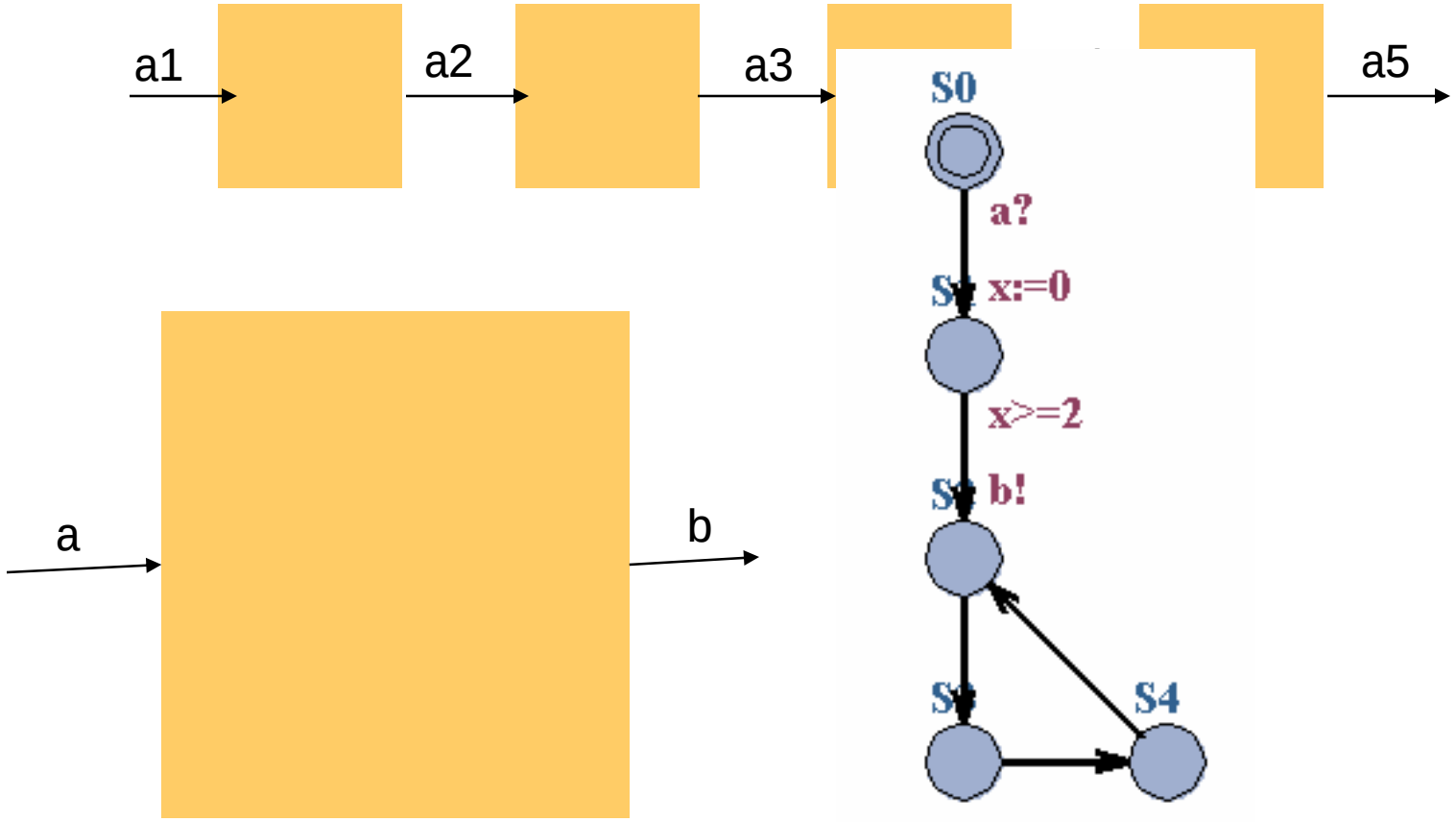
Concrete

Abstract



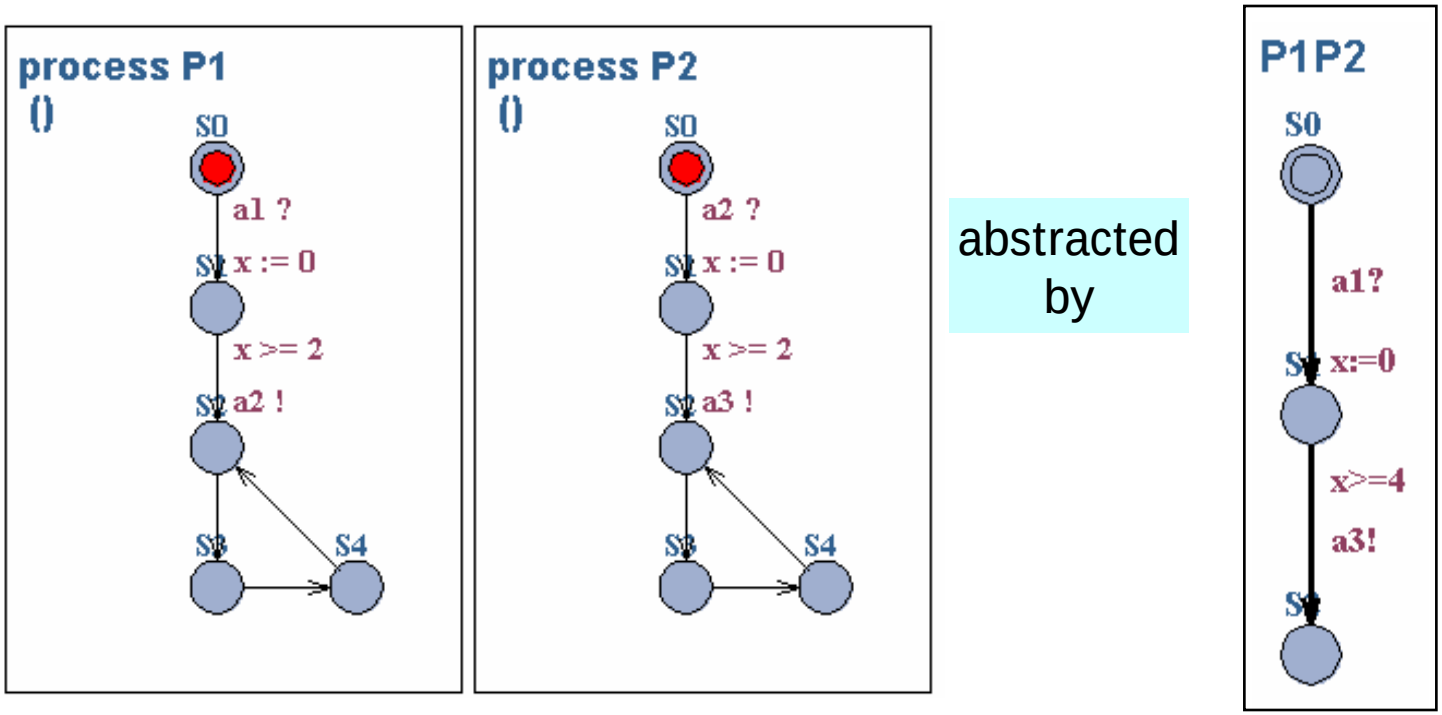
BRICS
Basic Research
in Computer Science

Abstraction *Example*



BRICS
Basic Research
in Computer Science

Example *Continued*



Proving abstractions

using reachability

Applied to

IEEE 1394a Root contention protocol (Simons, Stoelinga)

B&O Power Down Protocol (Ejersbo, Larsen, Skou, FTRTFT2k)

nizes BAD computations of PoP1

A[] not TestAbstPoP1.BAD

Henrik Ejersbo Jensen PhD Thesis 1999

BRICS Basic Research in Computer Science

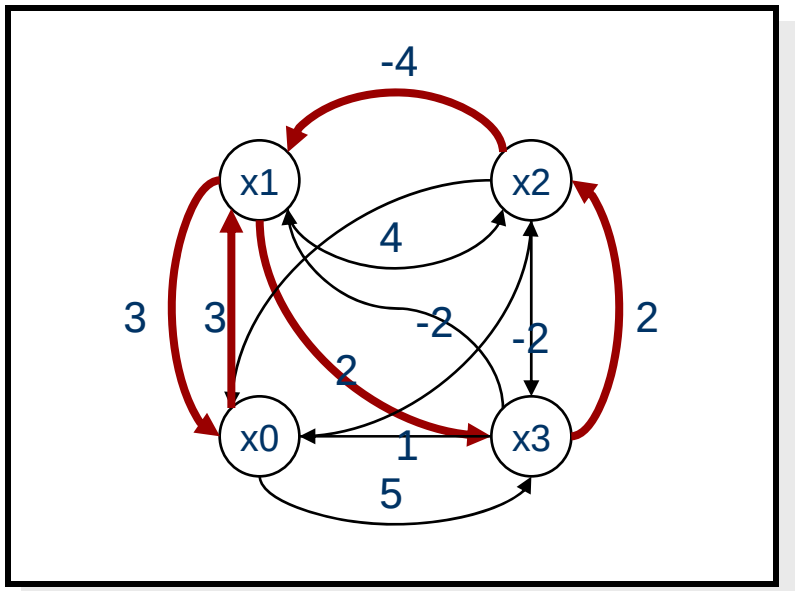
Further Optimizations

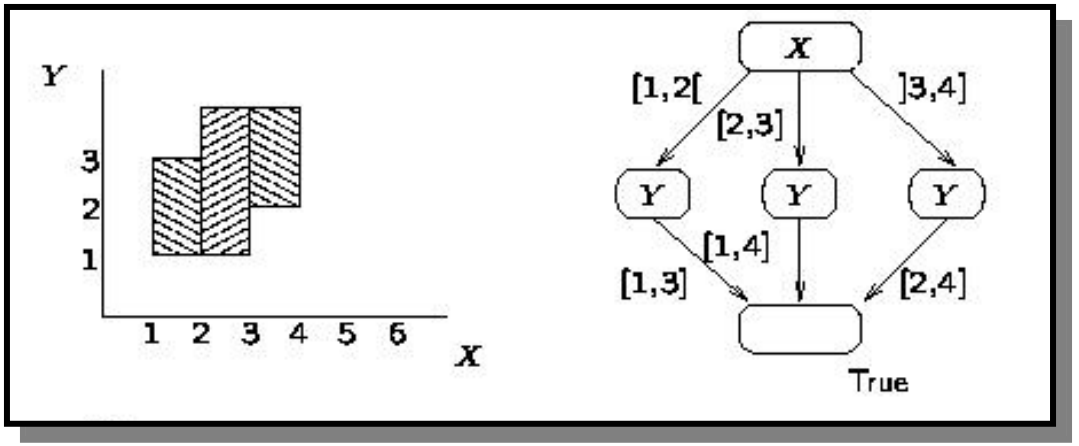
BRICS
Basic Research
in Computer Science

CSS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Datastructures for Zones

- Difference Bounded Matrices (DBMs)
- Minimal Constraint Form
  [RTSS97]
- Clock Difference Diagrams
  [CAV99]
- PW List
  [SPIN03]





BRICS
Basic Research
in Computer Science

Datastructures for Zo

- Difference Bounded

Alexandre David

Elegant RUBY bindings for easy implementations

UPPAAL DBM Library

The library used to manipulate DBMs in UPPAAL

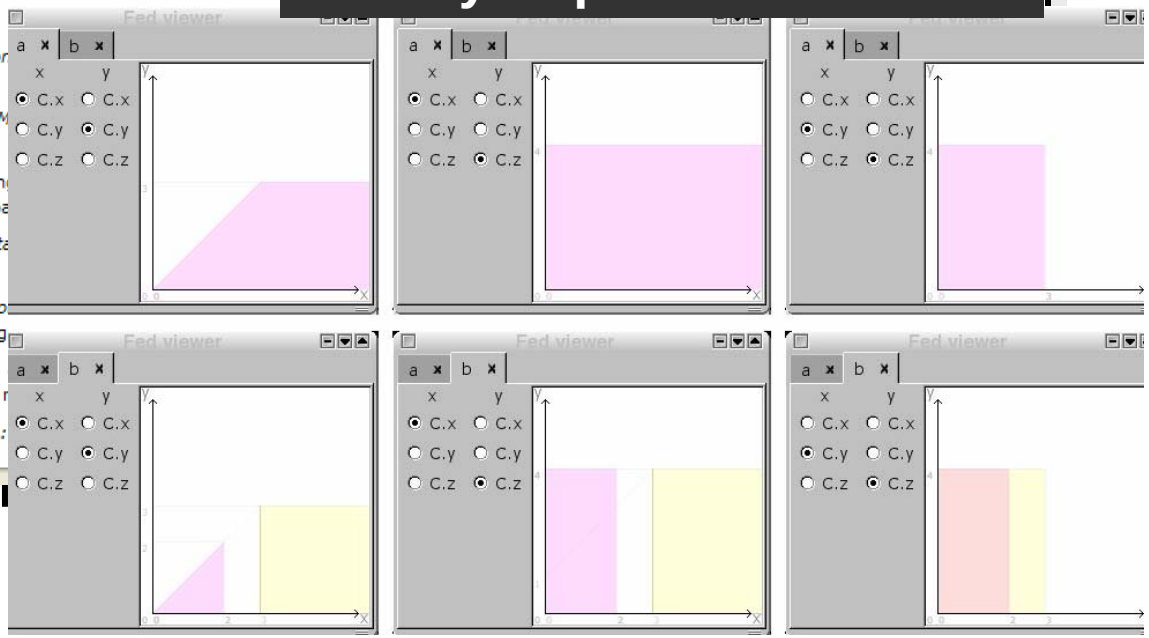
Main Page | Download | Ruby Binding | Help | Contact us

Welcome!

DBMs [dill89, rokicki93, lpw:fct95, bengtsson02] are efficient data structures to represent clock constrain in timed automata [ad90]. They are used in UPPAAL [lpy97, by04, bdl04] as the core data structure represent time. The library features all the common operations such as up (delay, or future), down (past), general updates, different extrapolation functions, etc.. on DBMs and federations. The lib subtractions. The API is in C and C++. The C++ part uses active clocks and hides memory management.

References

- [dill89] David L. Dill. *Timing Assumptions and Verificatio* Springer Berlin 1989, pp 197-212.
- [rokicki93] Tomas Gerhard Rokicki. *Representing and M* University 1993.
- [lpw:fct95] Kim G. Larsen, Paul Pettersson, and Wan Fundamentals of Computation Theory 1995, LNCS 965 pa
- [bengtsson02] Johan Bengtsson. *Clocks, DBM, and St* University 2002.
- [ad90] Rajeev Alur and David L. Dill. *Automata fo* Colloquium on Algorithms, Languages, and Programming
- [lpy97] Kim G. Larsen, Paul Pettersson, and Wang Yi. Software Tools for Technology Transfer , October 1997,
- [by04] Johan Bengtsson and Wang Yi. *Timed Automata:*

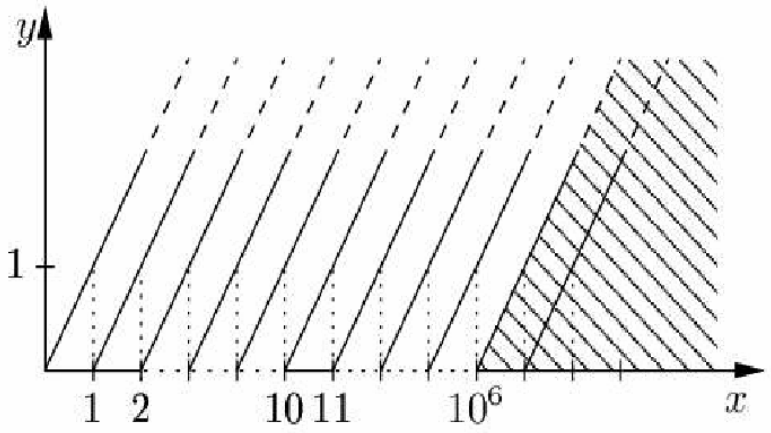


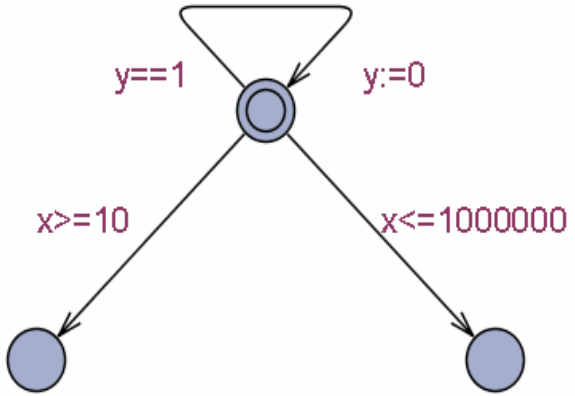
[SPIN03]

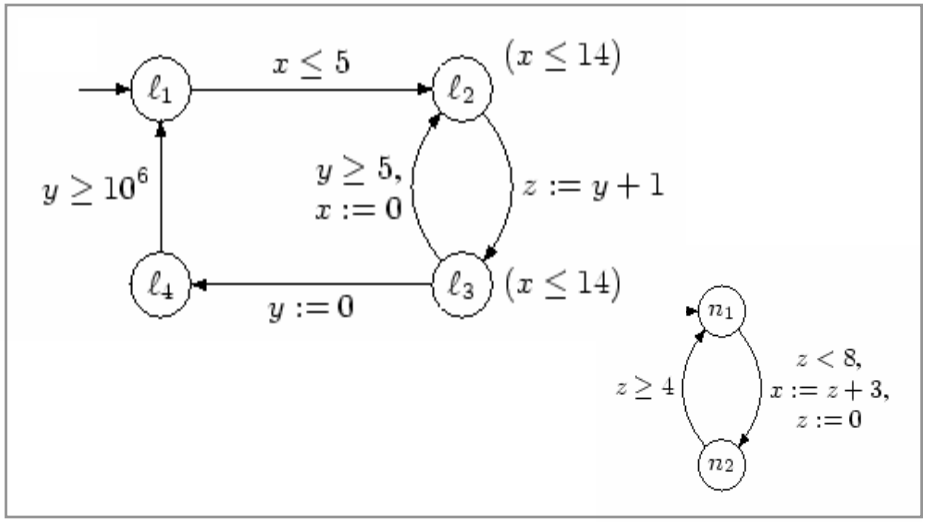
BRICS
Basic Research
in Computer Science

Zone Abstractions

[TACAS03,TACAS04]





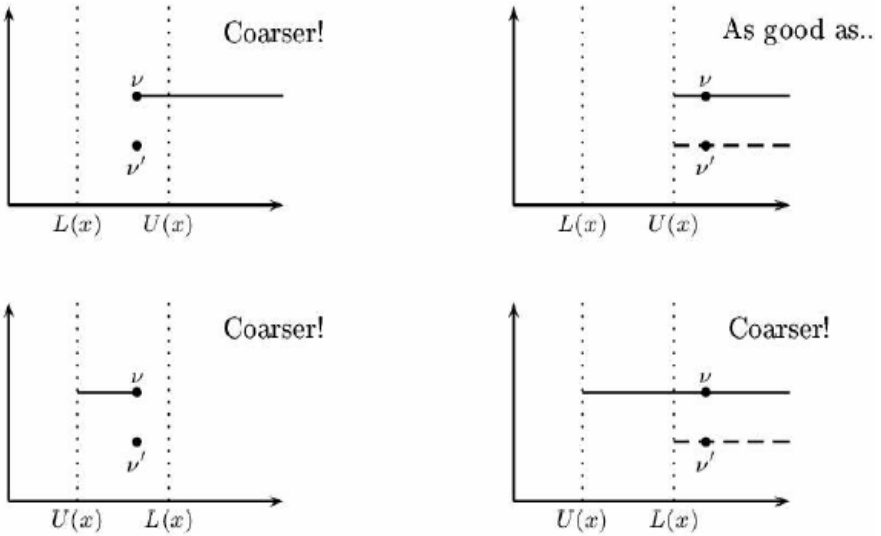


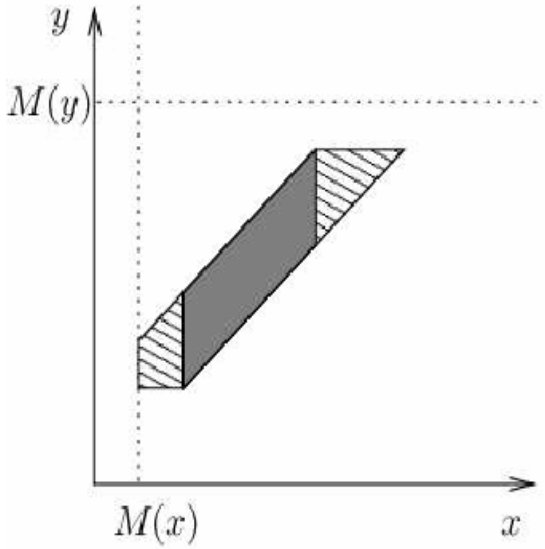
- Abstraction taking maximum constant into account necessary for termination
- Utilization of distinction between lower and upper bounds
- Utilization of location-dependency

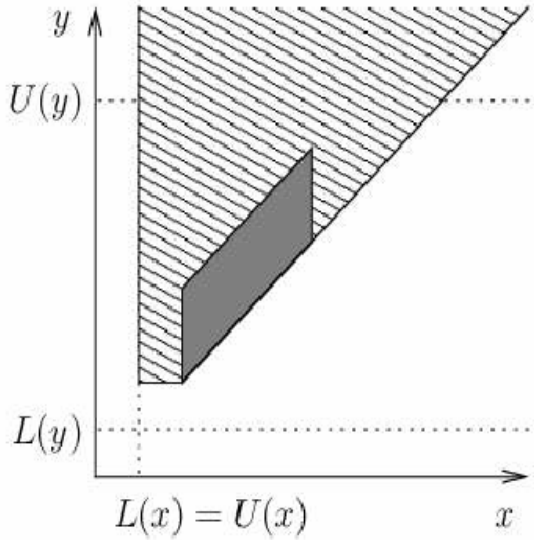
BRICS
Basic Research
in Computer Science

LU Abstraction

[TACAS04]







THEOREM

For any state in the LU- abstraction there is a state in the original set simulating it

→

LU abstraction is exact wrt reachability

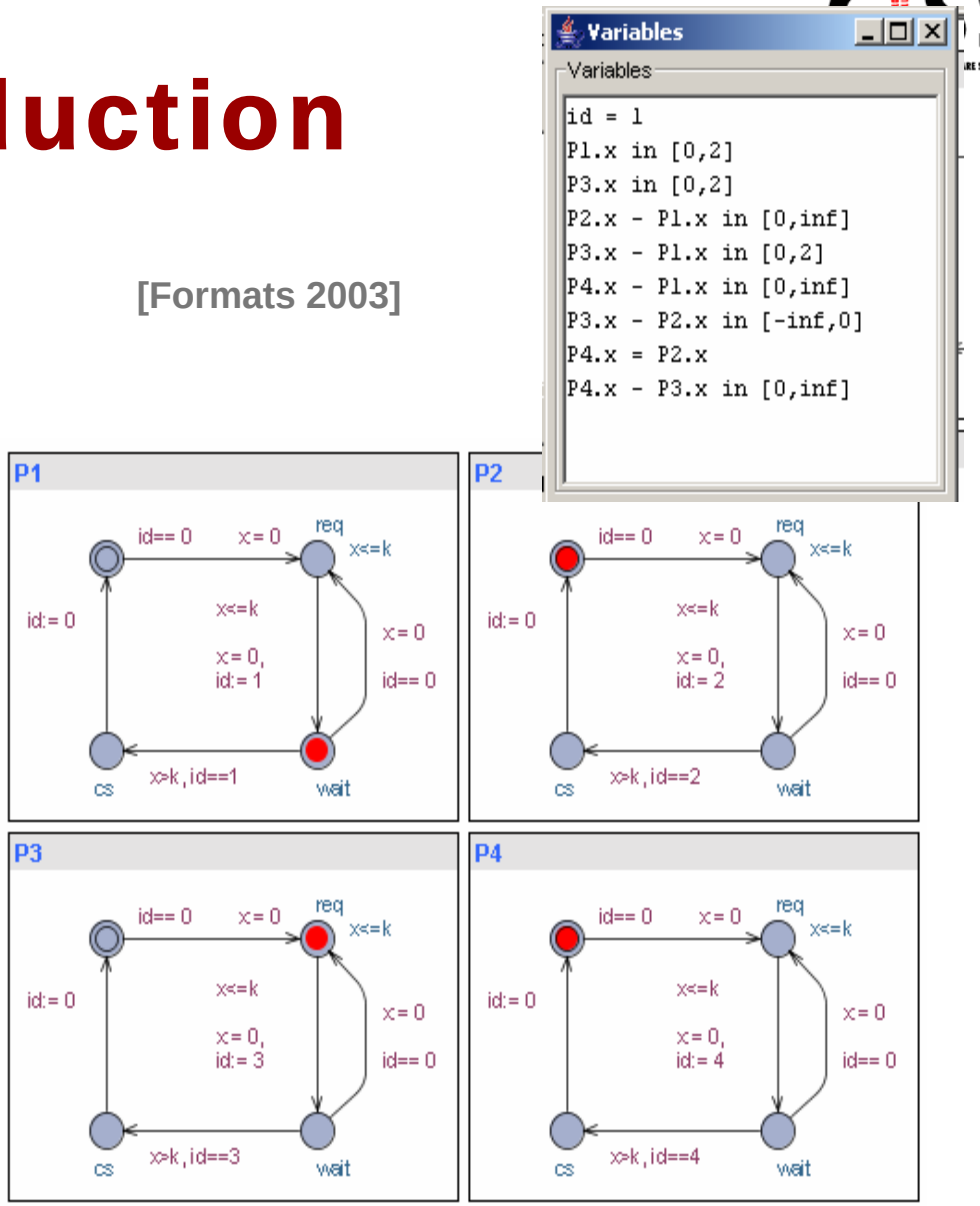
Zone abstractions

	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
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			
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

Symmetry Reduction

- Exploitation of full symmetry may give factorial reduction
- Many timed systems are inherently symmetric
- Computation of canonical state representative using swaps.

[Formats 2003]



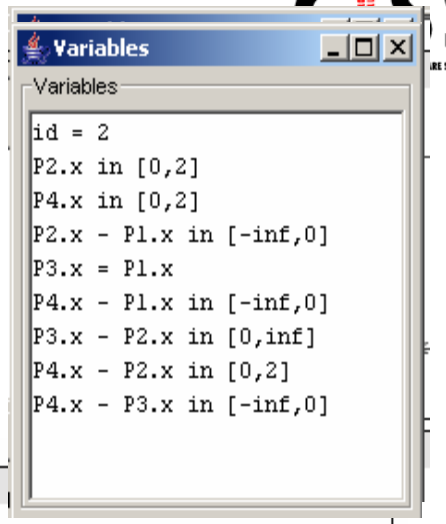
BRICS
Basic Research
in Computer Science

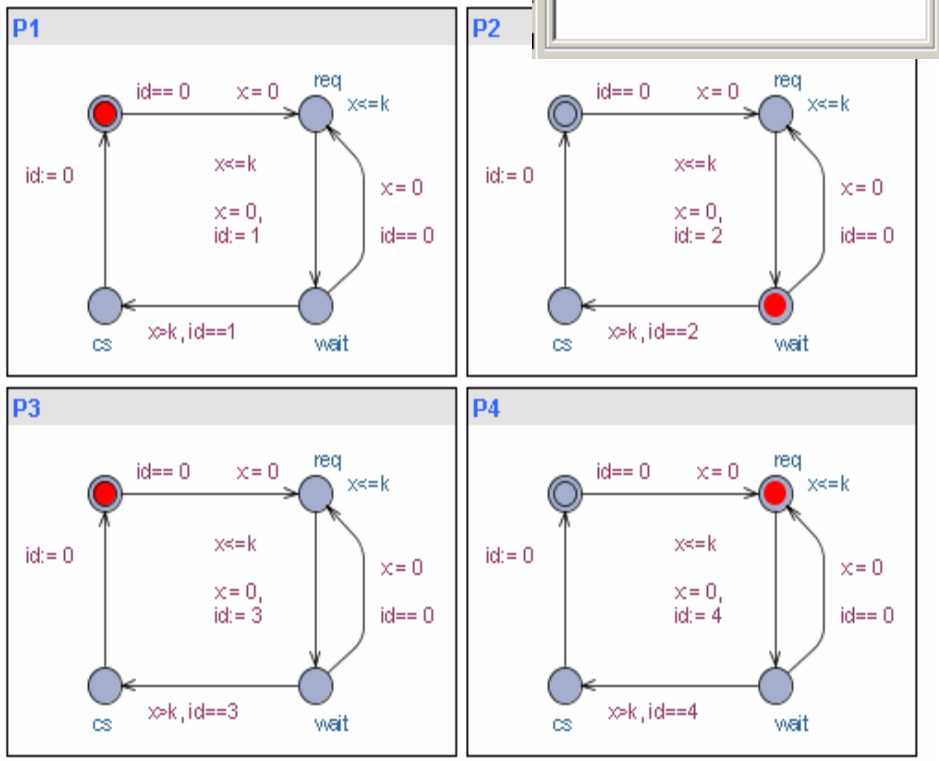
Symmetry Reduction

- Exploitation of full symmetry may give factorial reduction
- Many timed systems are inherently symmetric
- Computation of canonical state representative using swaps.

[Formats 2003]

SWAP: 1→2 ; 3→4

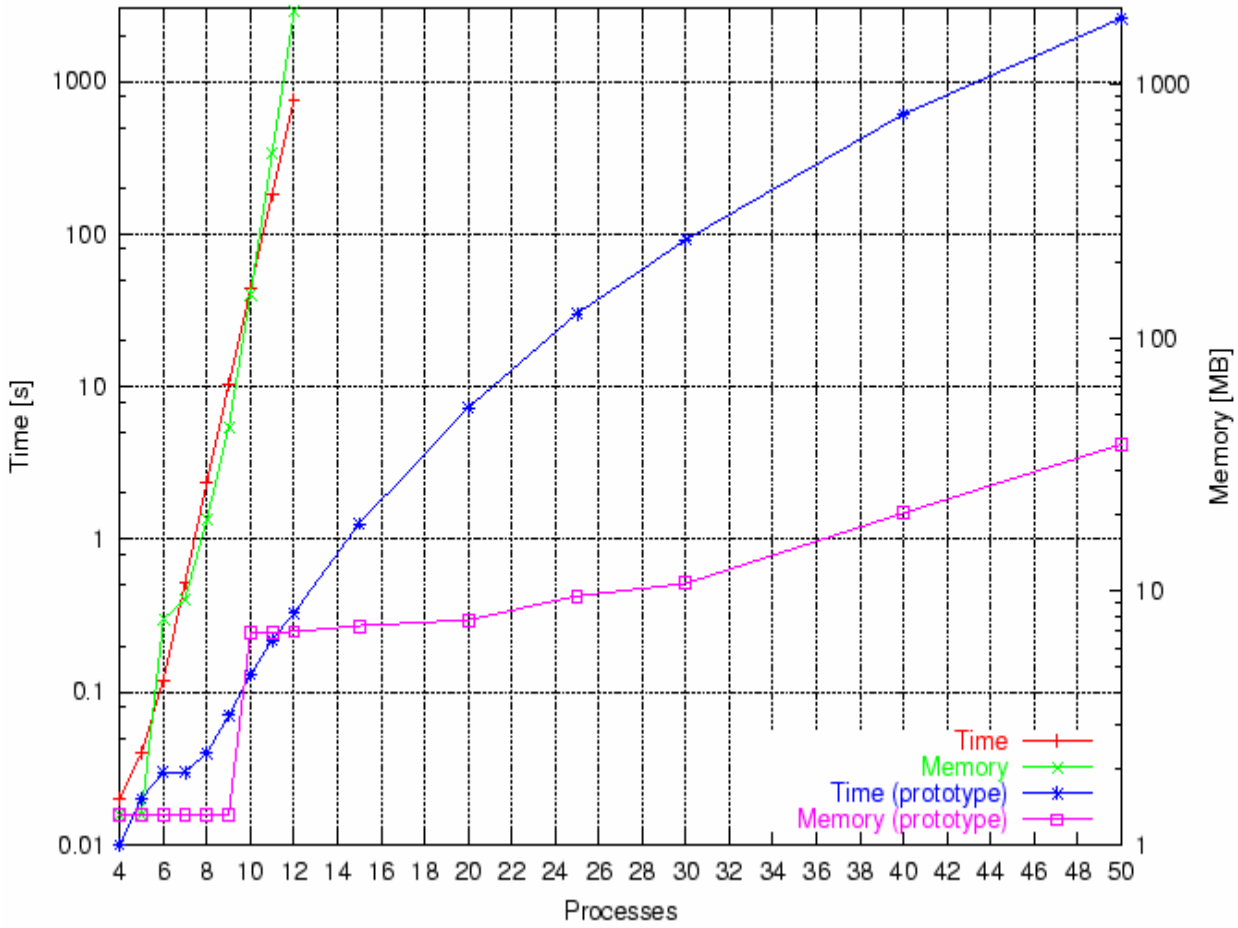




BRICS
Basic Research
in Computer Science

Symmetry Reduction

[Formats 2003]



BRICS
Basic Research
in Computer Science

Symmetry Reduction

UPPAAL 3.6

- Iterators `for (i: int[0,4]) { }`
- Quantifiers `forall (i: int[0,4]) a[i]==0`
- Selection `select i: int[0,4]; guard...`
- Template sets `process P[4](...) {  }`
- Scalar set based symmetry reduction
- Compact state-space representations
- Priorities

Gerd Behrmann

Martijn Henriks, Nijmegen U

BRICS
Basic Research
in Computer Science

D-UPPAAL

Gerd Behrmann

- Distributed implementation of UPPAAL on PC-cluster [CAV'00, PDMC'02, STTT'03].
- WWW frontend available
- Applications
  - Synthesis of Dynamic Voltage Scaling strategies (CISS).
  - Ad-hoc mobile real-time protocol (Leslie Lamport) - 25GB in 3 min!
- Running on NorduGrid.
  Local cluster: 50 CPUs and 50GB of RAM
- To be used as inspiration for verification GRID platform within ARTIST2 NoE.

BRICS
Basic Research
in Computer Science

D-UPPAAL
Gerd Behrman

- Distributed implementation of UPPAAL on PC-cluster [CAV'00, PDMC'02, STTT'03].
- WWW frontend available.
- Applications
  - Synthesis of Dynamic Volta Scaling strategies (CISS).
  - Ad-hoc mobile real-time pr (Leslie Lamport) - 25GB in
- Running on NorduGrid. Local cluster: 50 CPUs and 50 RAM
- To be used as inspiration for verification GRID platform with ARTIST2 NoE.

Grid Monitor

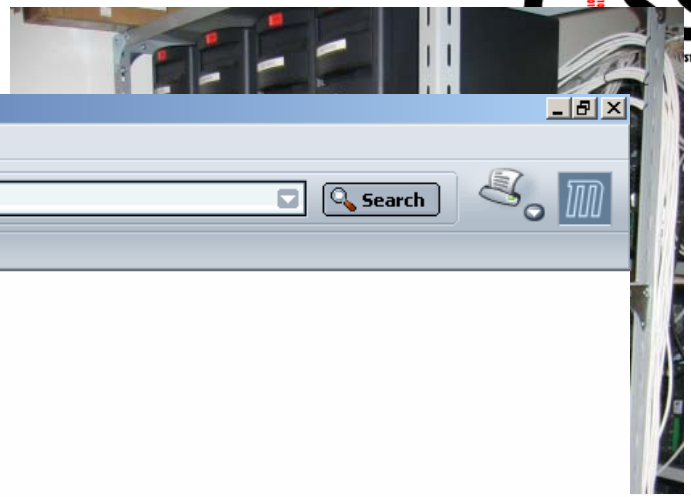
2004-11-22 CET 12:46:48

Processes: Grid Local

Country	Site	CPUs	Load (processes: Grid+local)	Queueing
Australia	Alfred (UniMelb)	90	7+24	9+1
	Atlas (UniMelb)	24	0+1	0+0
	Charm (UniMelb)	20	0+0	0+0
Denmark	Aalborg Grid Gateway	50	13+0	0+0
	DistLab (DIKU)	10	0+0	0+0
	HEPAX1	1	0+0	0+0
	Morpheus	18	0+0	0+0
Estonia	CMS Production server	5	0+0	0+0
	CMS test cluster	1	0+0	0+0
	EENet Kriit	6	6+0	4+0
	UT Chemistry	17	0+17	0+3
	UT CS Antarctica Clus>	20	8+0	0+0
	UT IMCB Anakonda clus>	16	0+0	0+0
	UT Physics Cluster	0		0+0
Finland	Alpha (HIP)	1	0+0	0+0
	CSC Kirppu	1	0+0	0+0
	CSC Lude	8	0+0	0+0
	Mill (Physicum)	56	0+18	8+0
	Testbed0 (HIP)	1	0+0	0+0
Germany	FZK cluster	1060	76+426	0+0
	LRZ cluster	260	0+244	0+899
Norway	Bergen Grid Cluster	4	0+0	0+0
	Oslo Grid Cluster	43	0+14	0+1
	Oslo Temp Cluster	7	0+0	0+0
	Parallab IBM Cluster	58	0+3	0+0

BRICS
Basic Research
in Computer Science

D-UPPAAL



Mozilla

File Edit View Go Bookmarks Tools Window Help

https://benedict.aau.dk/duppaal/ Search

Home Bookmarks

File information:

Model: AN ASTRONOMICALLY BIG MODEL Browse...

Query: A VERY INTERESTING QUESTION Browse...

Model checking options

Search order: bredth first width first
State space reduction: none conservative aggressive
State space representation: DBM compact data structure under approximation over approximation
New syntax: no yes

Distribution options

Number of CPUs: 1 5 10 15 20 25 30 35 49

Run options

Max walltime (minutes): 1 5 15 30 60 120 240

Contact information

Email: kgl@cs.auc.dk

Submit Query Reset

Find stopped.

Start 15:00

ueueing
+1
+0
+0
+0
+0
+0
+0
+0
+0
+0
+0
+3
+0
+0
+0
+0
+0
+0
+0
+899
+0
+1
+0