Symbolic Real Time Model Checking

Kim G Larsen

Overview

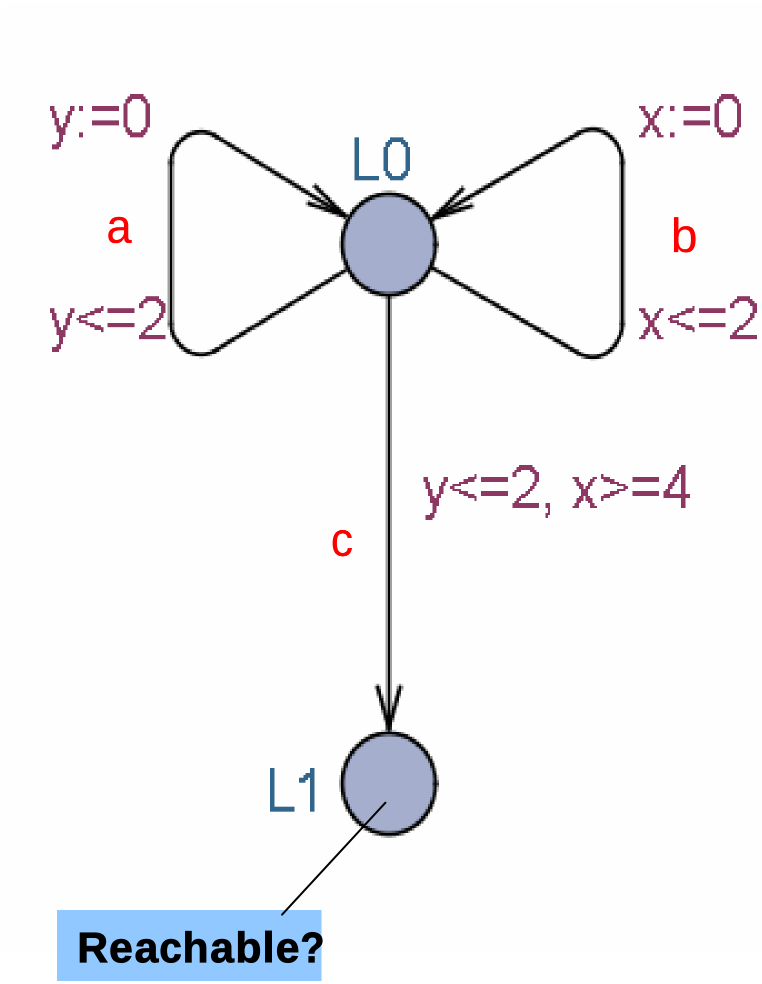
- Timed Automata – Decidability Results
- The UPPAAL Verification Engine
  - Datastructures for zones
  - Liveness Checking Algorithm
- Abstraction and Compositionality
- Further Optimizations

Timed Automata – Decidability Results

BRICS Basic Research in Computer Science

CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Decidability ?



OBSTACLE: Uncountably infinite state space

Derived Relations and Reachability

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

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

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

Definition

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

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

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

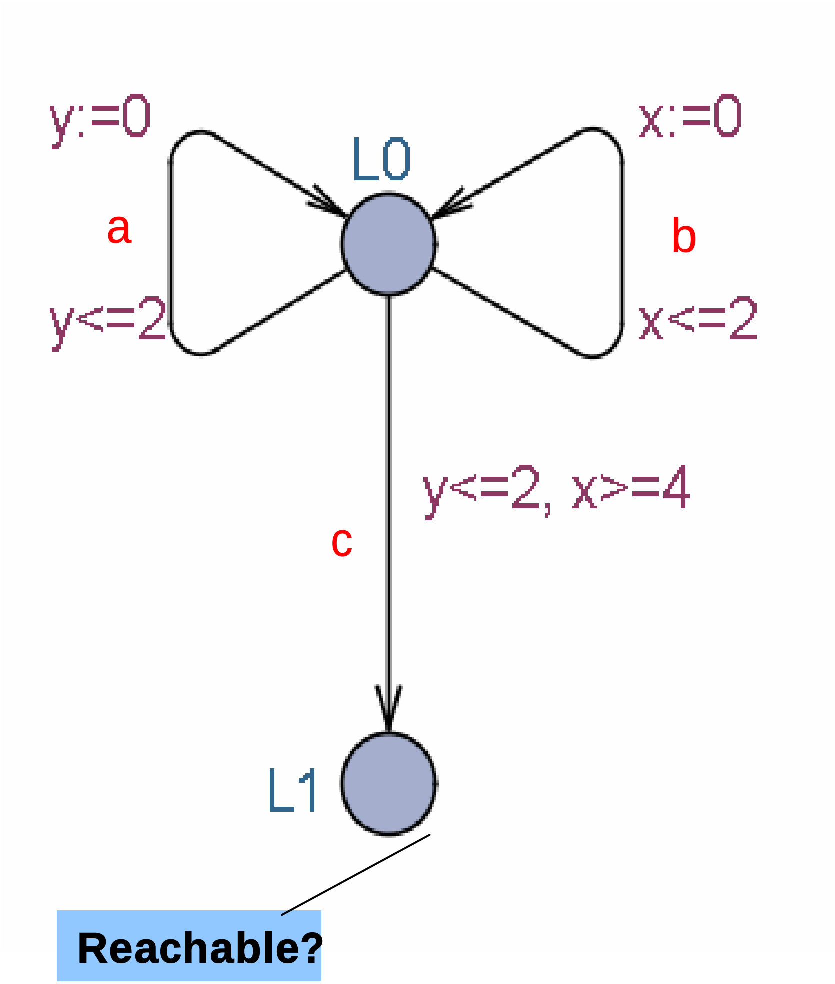
- $\pi \xrightarrow{\delta} \pi'$ iff $(l, u) \xrightarrow{\delta} (l', u')$ for some $(l, u) \in \pi$, $(l', u') \in \pi'$.
- $\pi \xrightarrow{a} \pi'$ iff $(l, u) \xrightarrow{a} (l', u')$ for some $(l, u) \in \pi$, $(l', u') \in \pi'$.

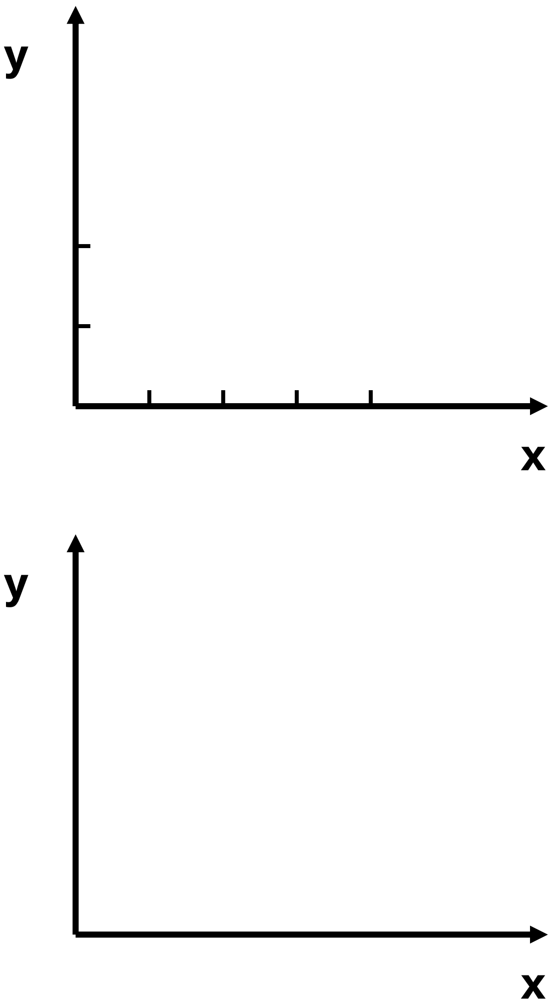
Theorem

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.

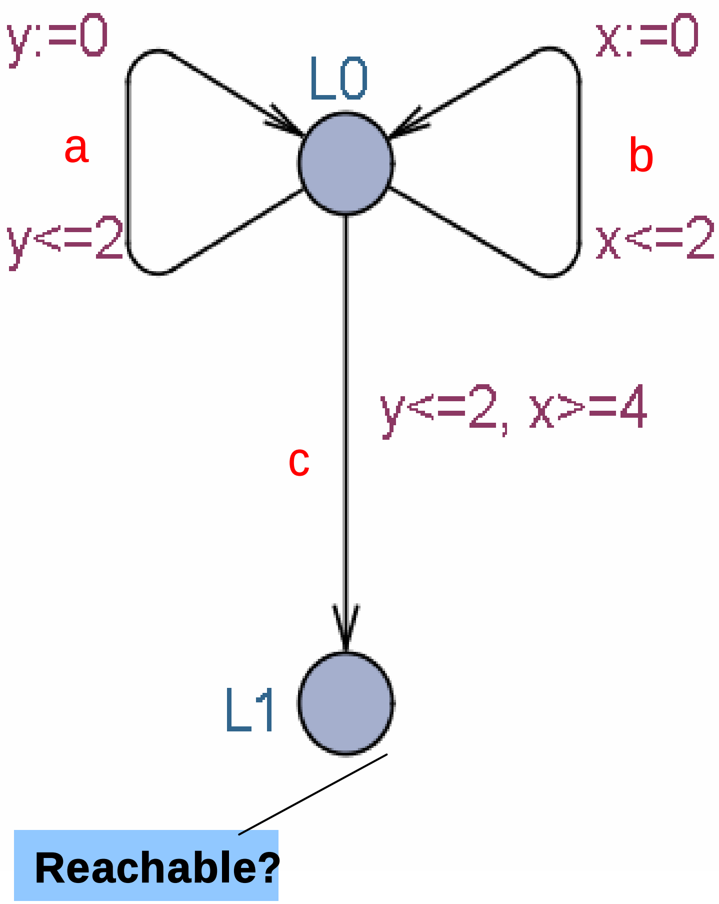
Stable Quotient

Partitioning

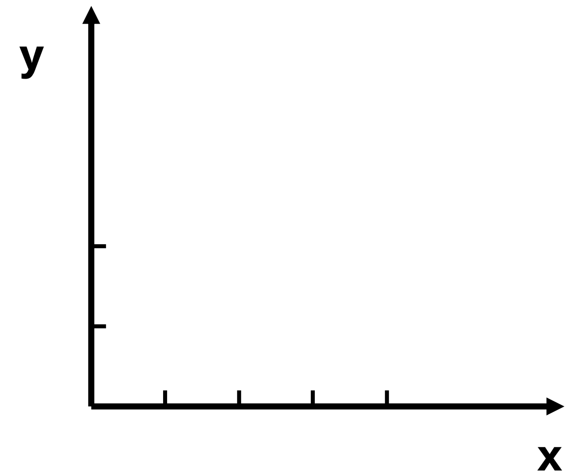


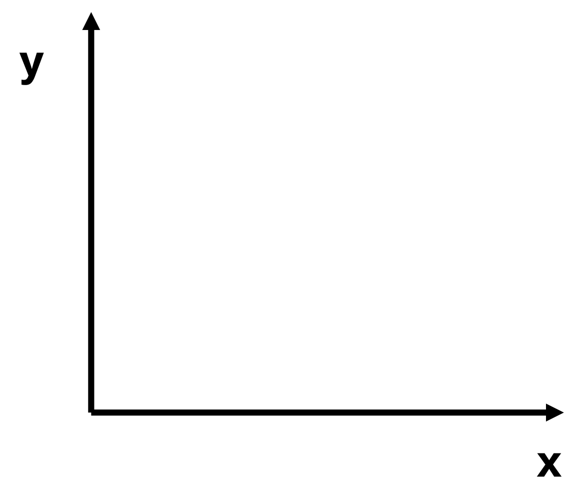


Stable Quotient

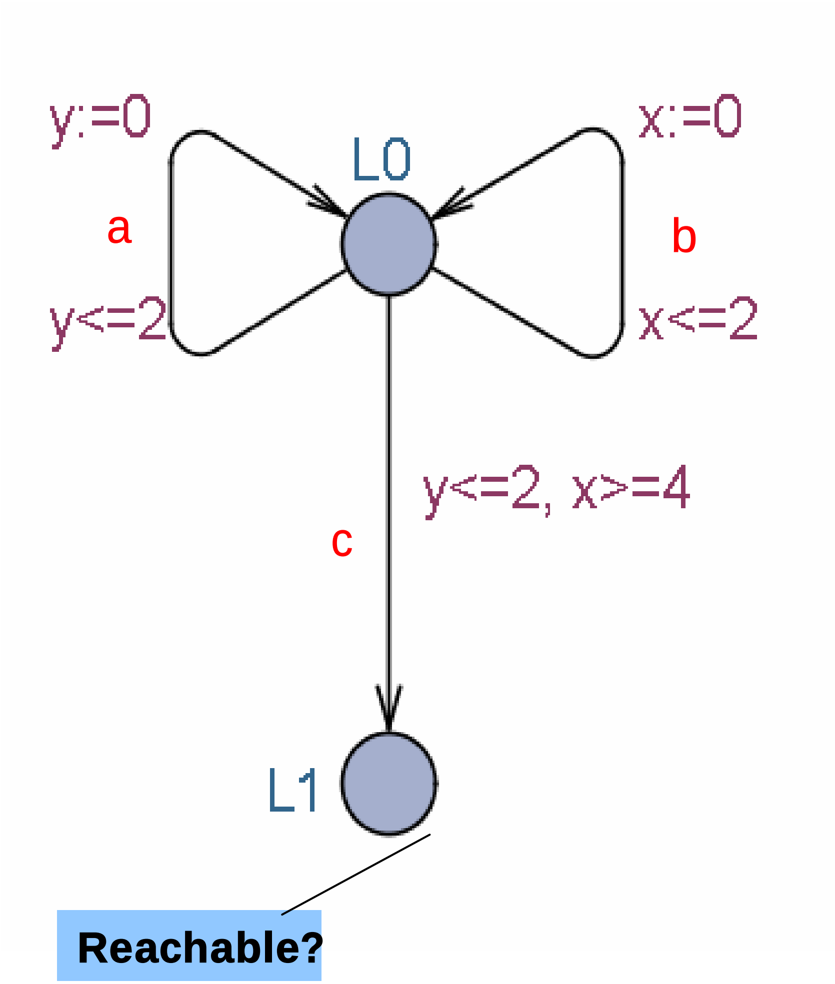


Partitioning

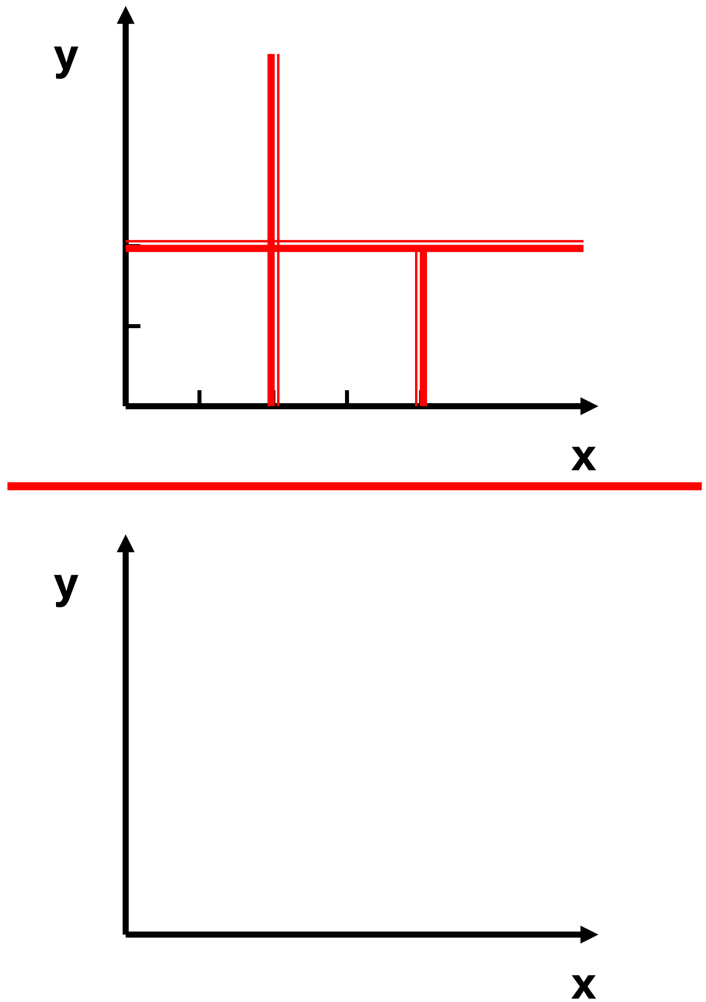




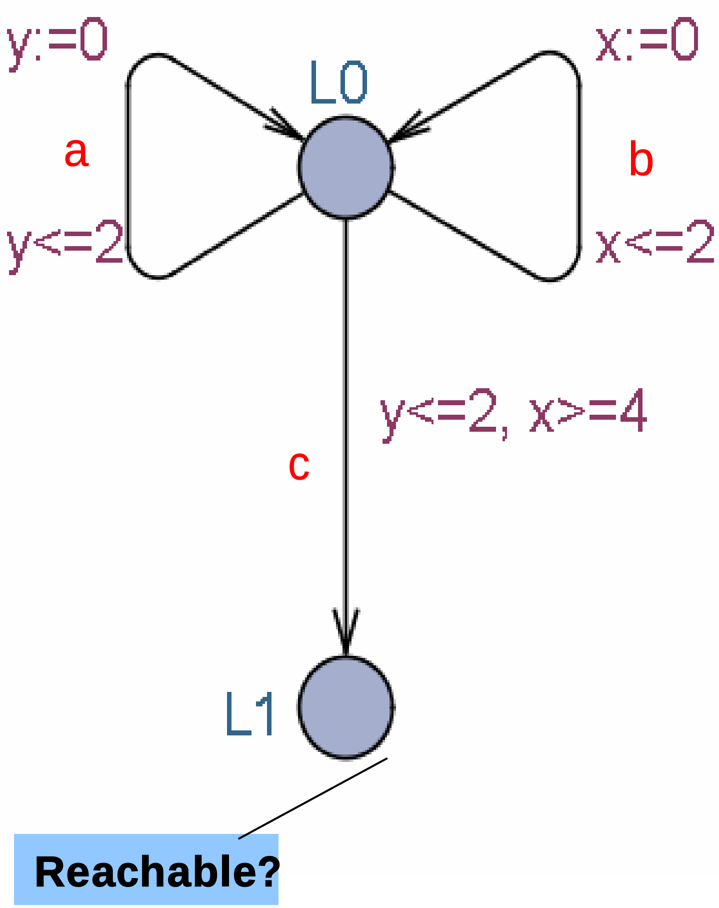
Stable Quotient



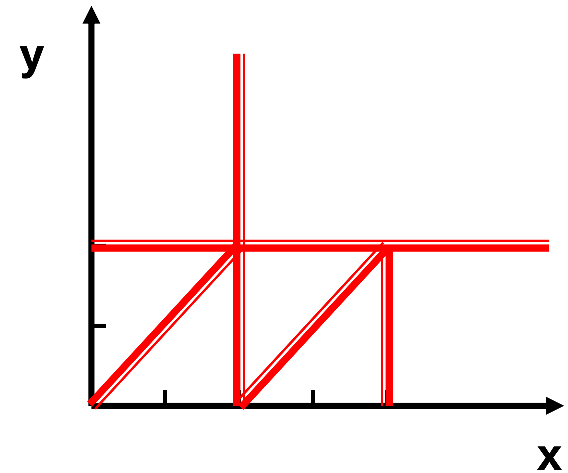
Partitioning

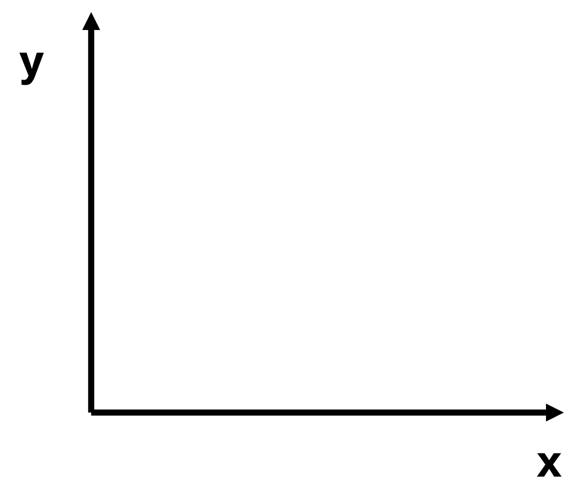


Stable Quotient

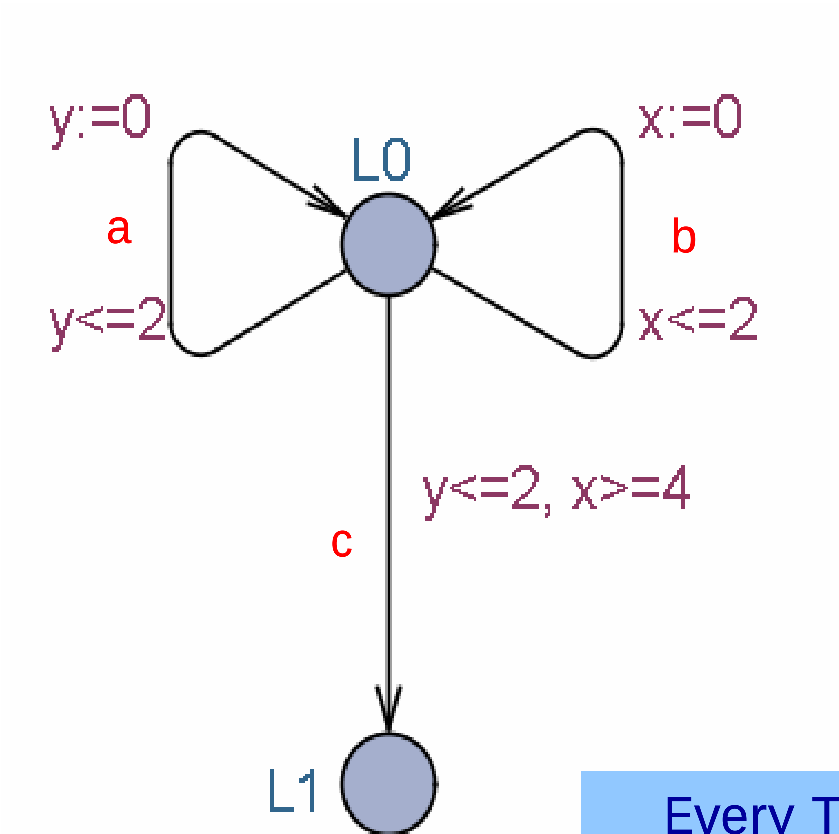


Partitioning





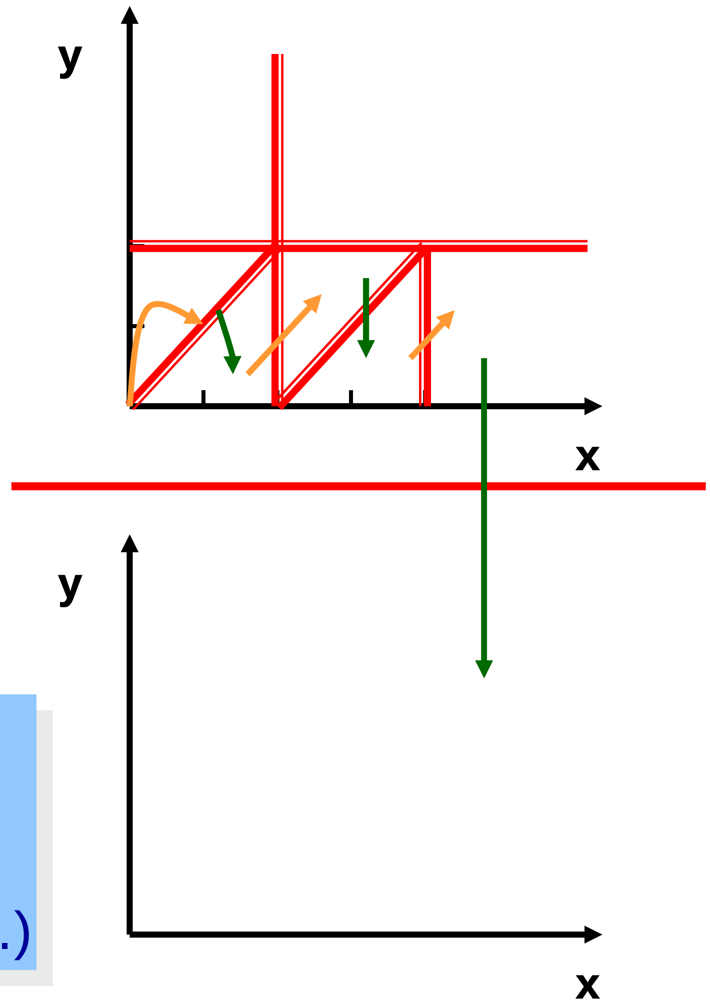
Stable Quotient



Reachable?

Every TA has a finite TAB quotient (region-constr.)

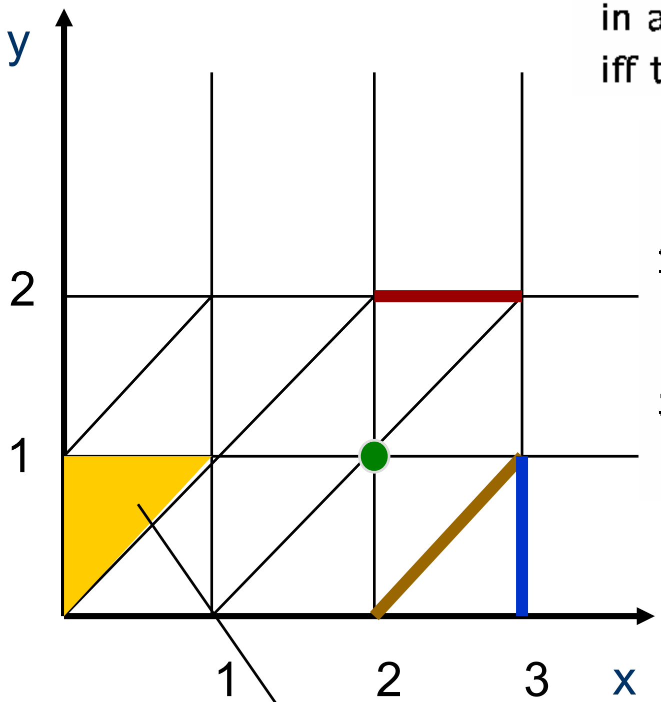
Partitioning



CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

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!!

BRICS
Basic Research
in Computer Science

Fundamental Results

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

CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Updatable Timed Automata

Patricia Bouyer, Catherine Dufourd,
Emmanuel Fleury, Antoine Petit

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

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

BRICS
Basic Research
in Computer Science

The UPPAAL Verification Engine

AALBORG UNIVERSITY DENMARK

BRICS
Basic Research in Computer Science

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Overview

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

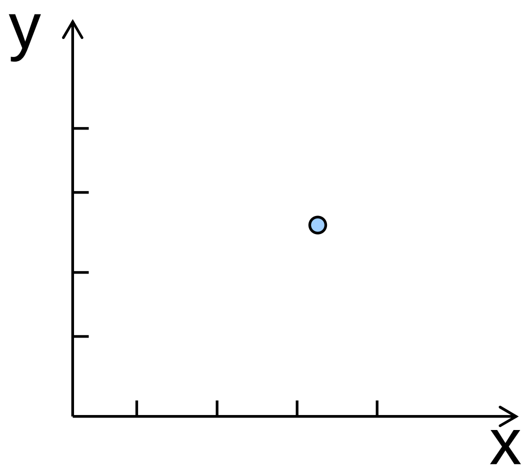
- Distributed UPPAAL [CAV2000, STTT2004]
- Unification & Sharing [FTRTFT2002, SPIN2003]
- Acceleration [FORMATS2002]
- Static Guard Analysis [TACAS2003,TACAS2004]
- Storage-Strategies [CAV2003]

Zones

From infinite to finite

State

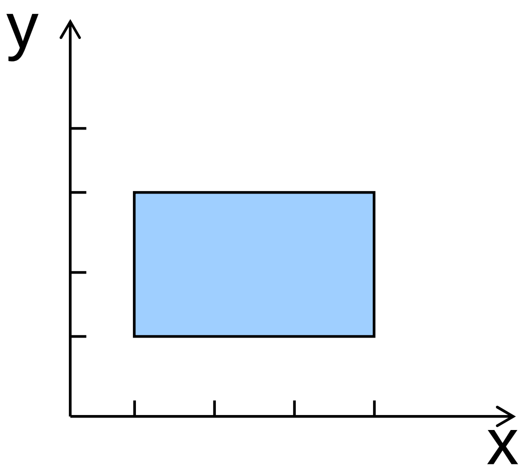
(n, x=3.2, y=2.5)



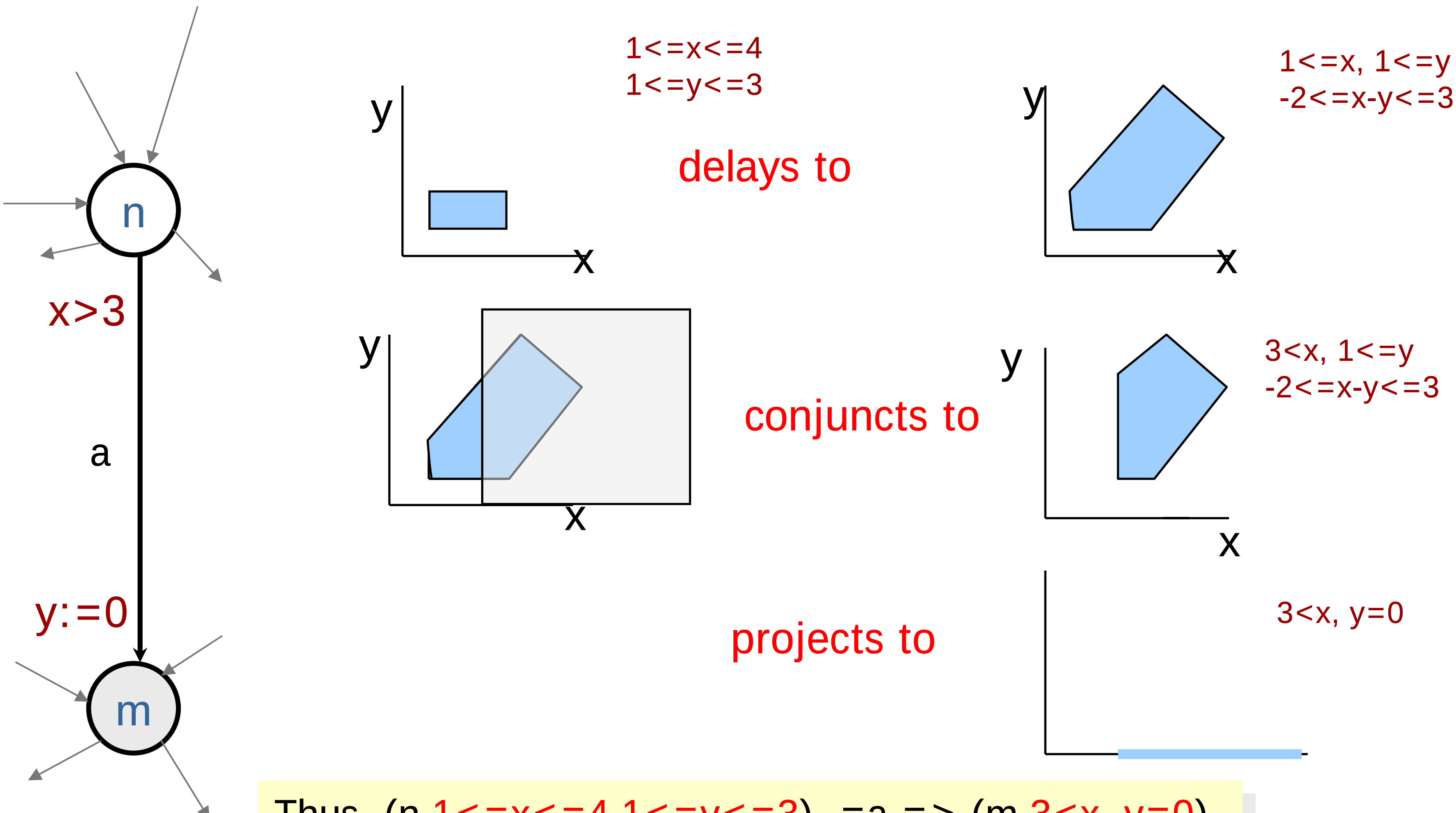
Symbolic state (set)

(n, $1 \leq x \leq 4$, $1 \leq y \leq 3$)

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



Symbolic Transitions



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

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)

BRICS Basic Research in Computer Science

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 $Z(x:=0)$

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

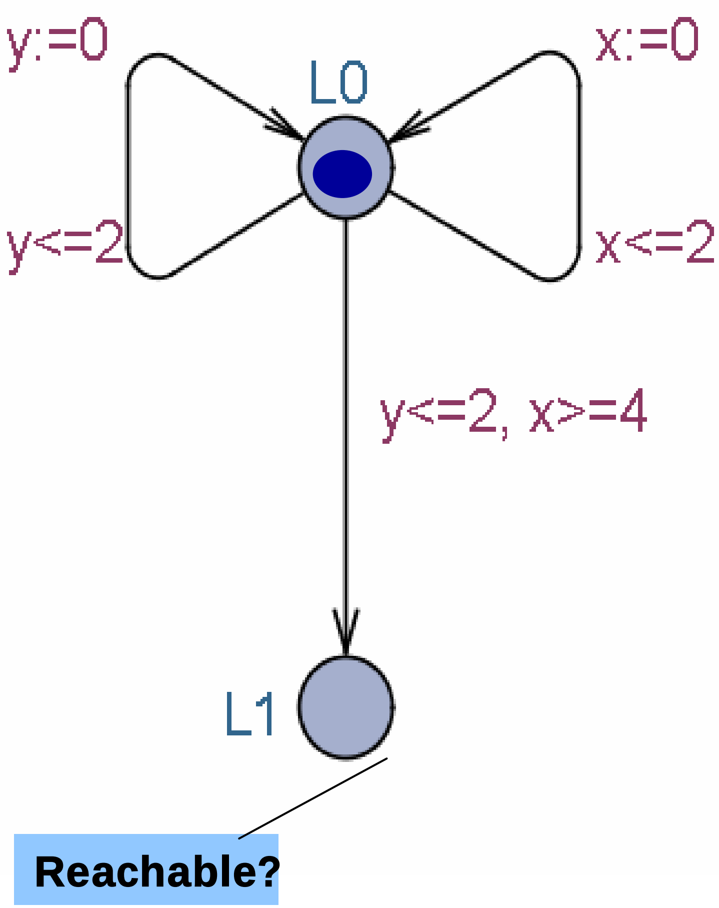
- Conjunction

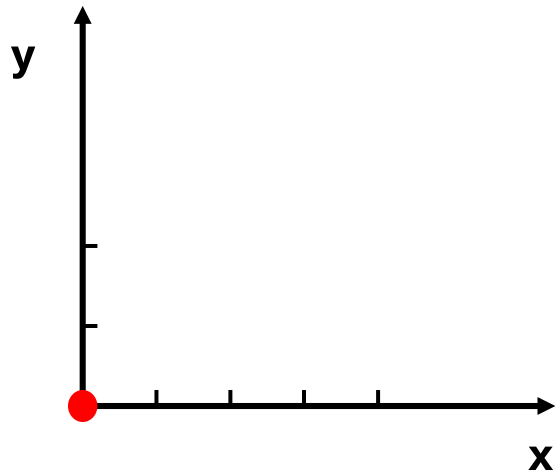
$$[Z\&g] = [Z] \cap [g]$$

THEOREM

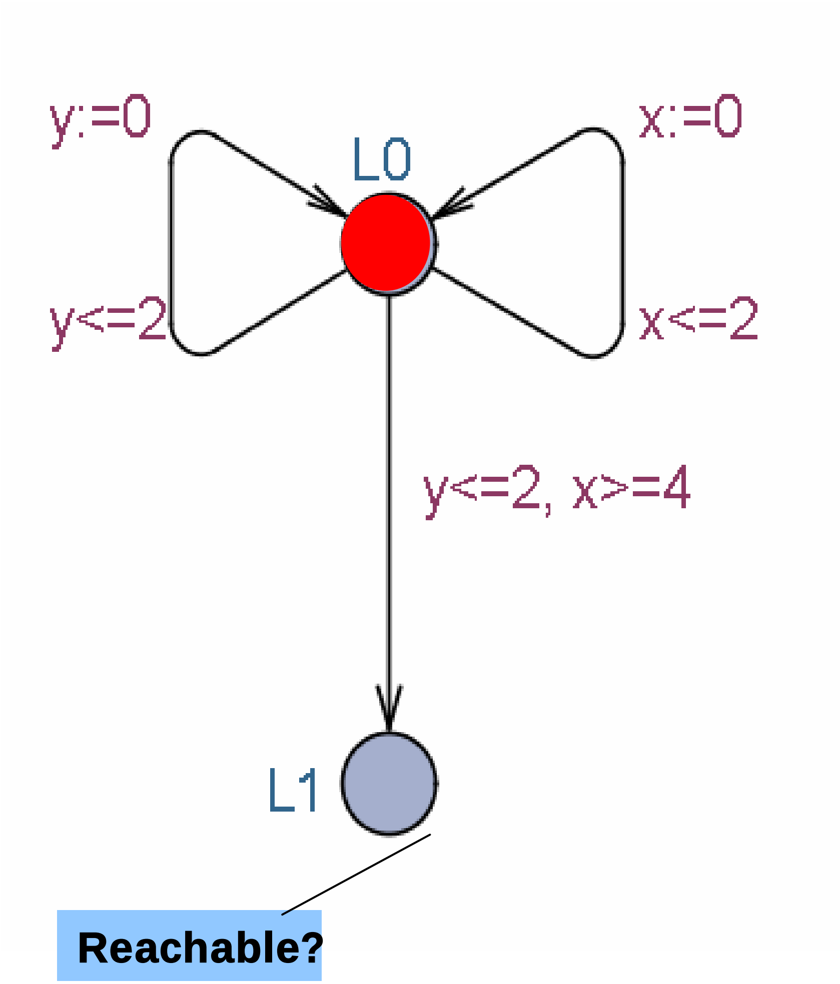
- The set of zones is closed under all constraint operations.
- That is, the result of the operations on a zone is a zone.
- That is, there will be a zone (a finite object i.e a zone/constraints) to represent the sets: [Z↑], [Z↓], [{x}Z], [Z&g].

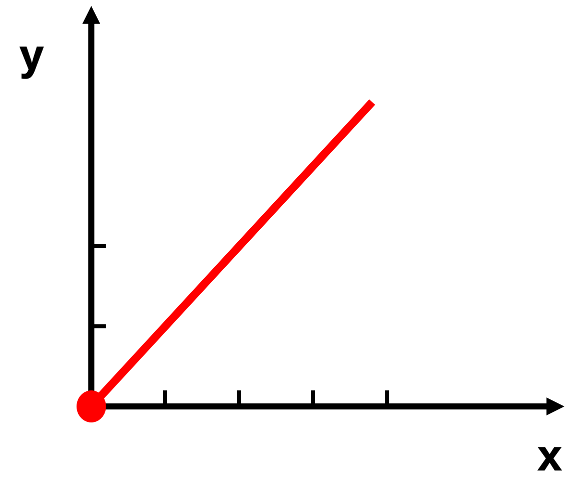
Symbolic Exploration



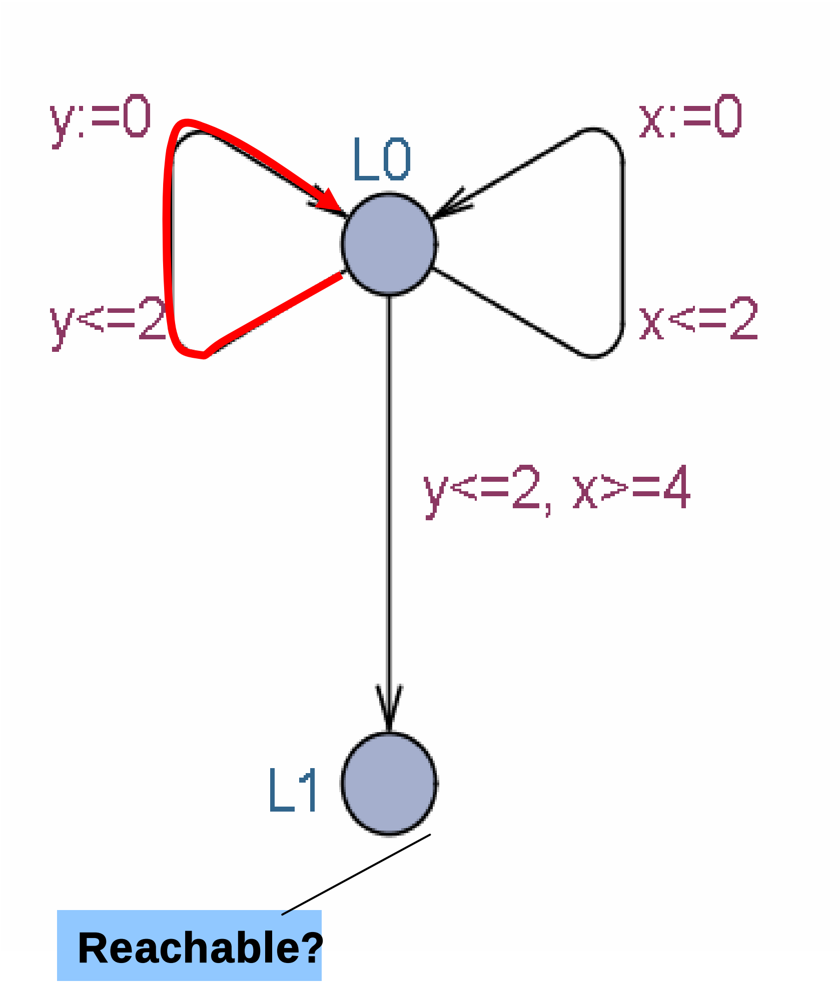


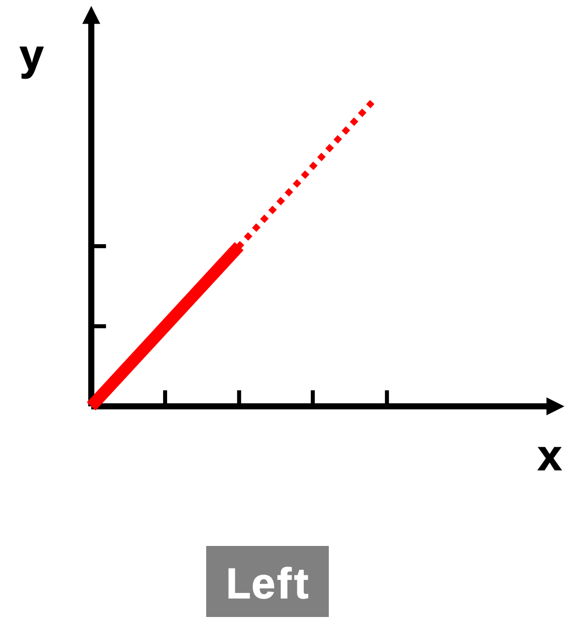
Symbolic Exploration



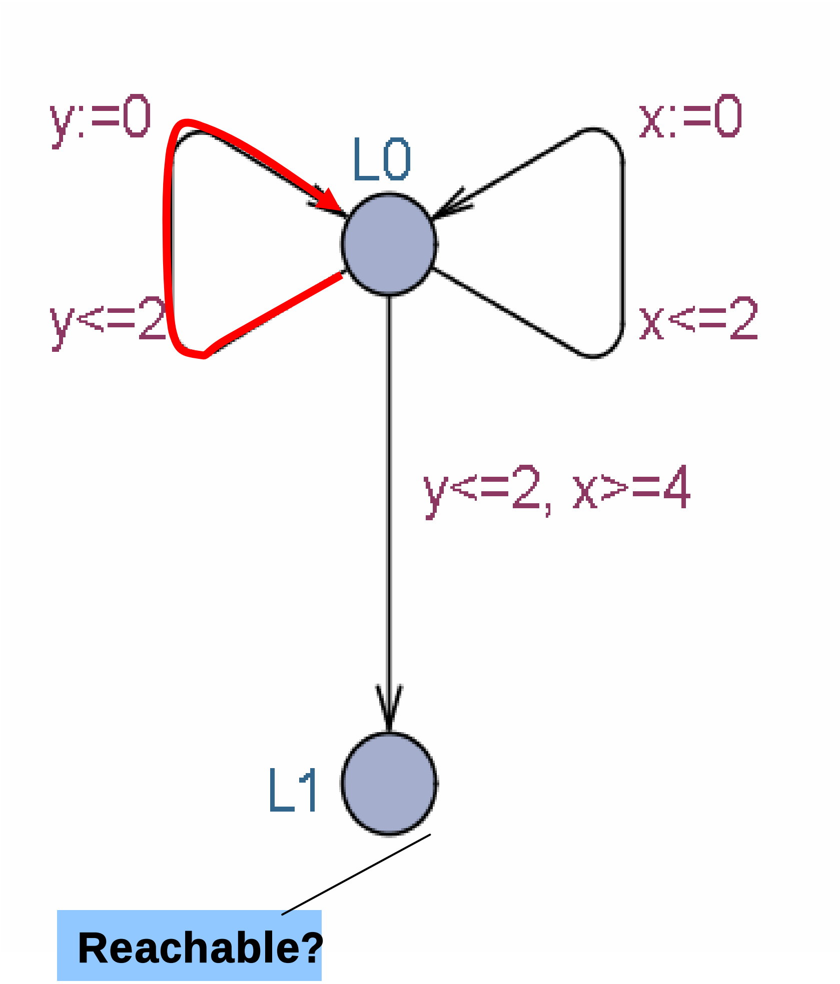


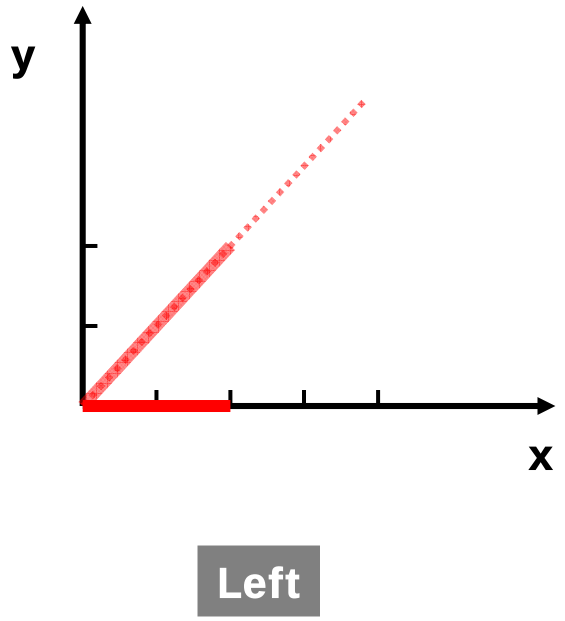
Symbolic Exploration



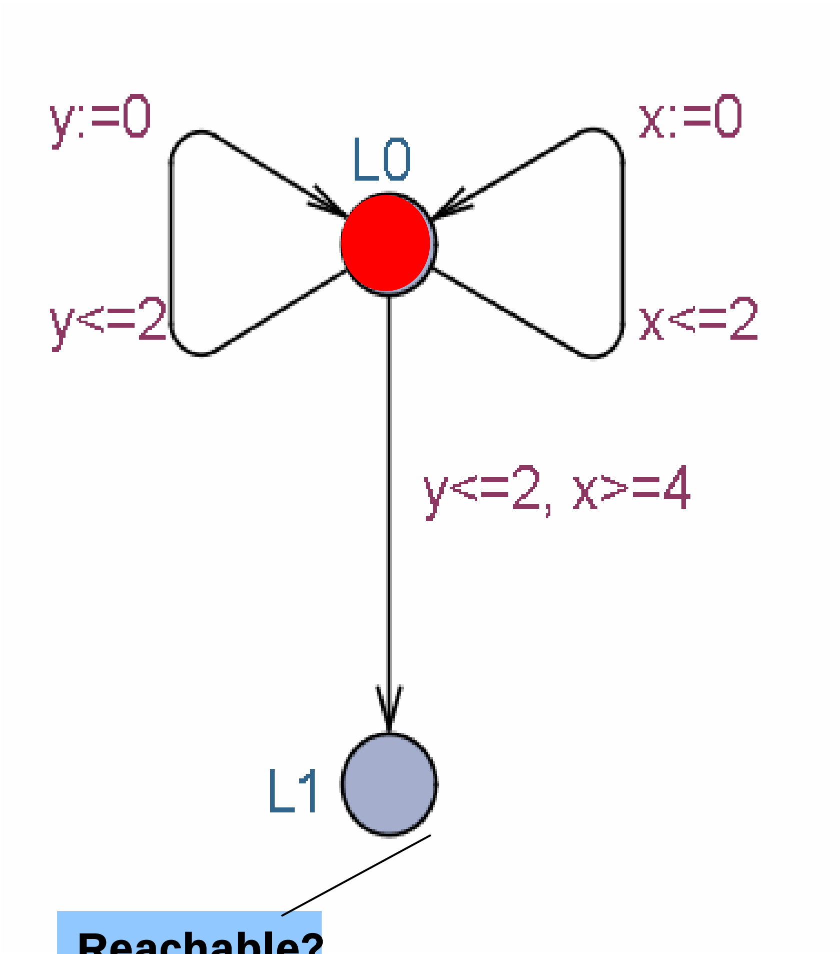


Symbolic Exploration

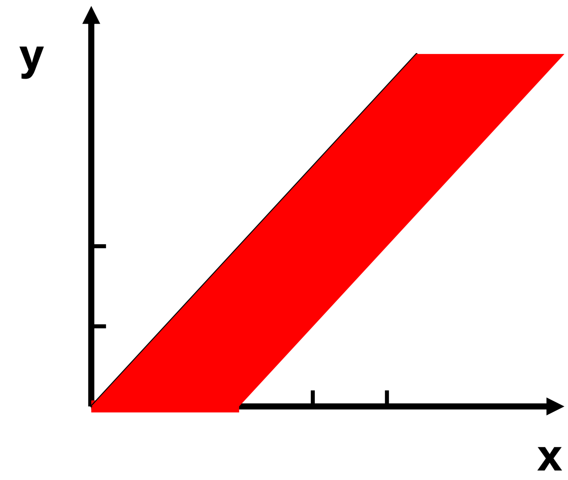




Symbolic Exploration

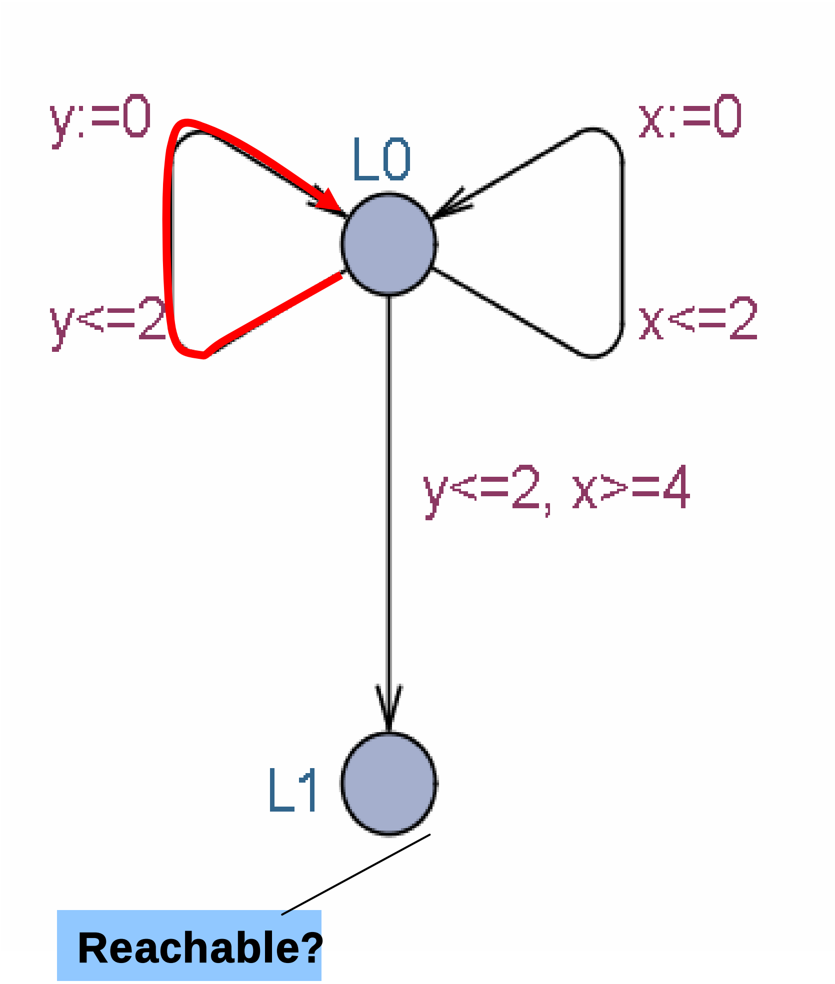


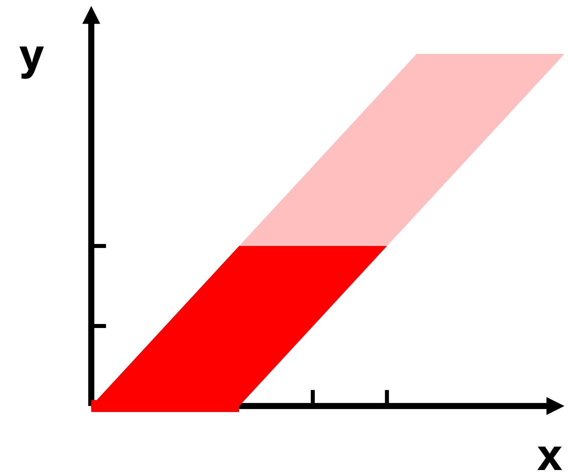
Reachable?



Delay

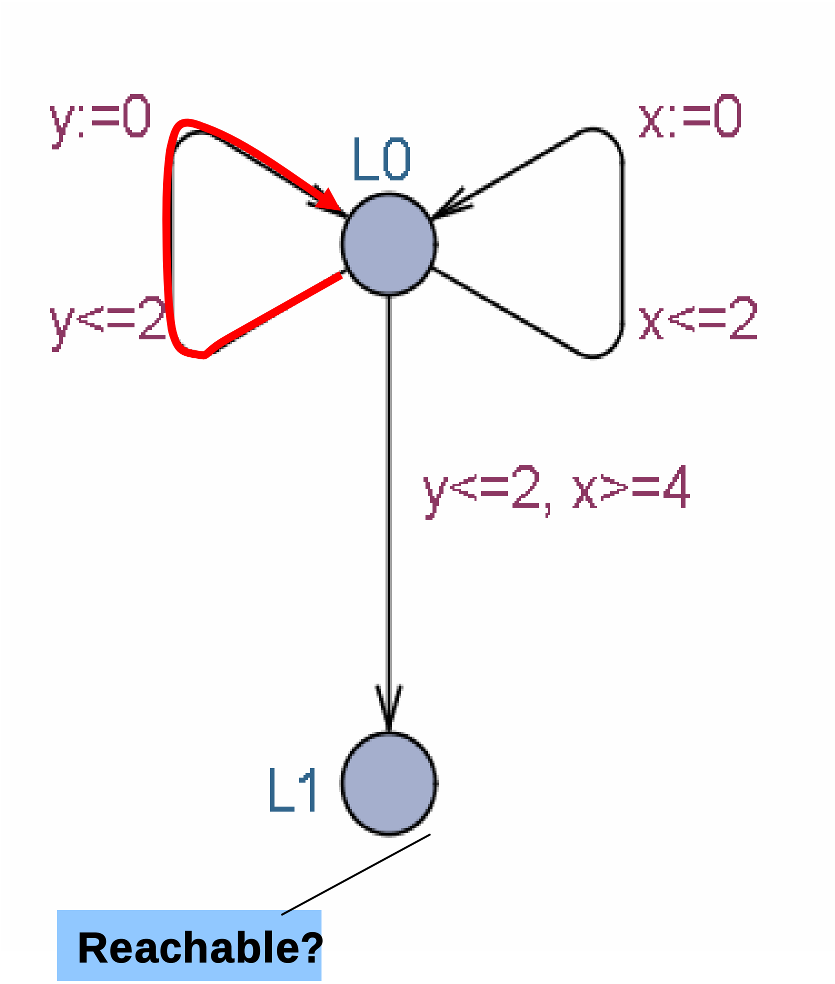
Symbolic Exploration

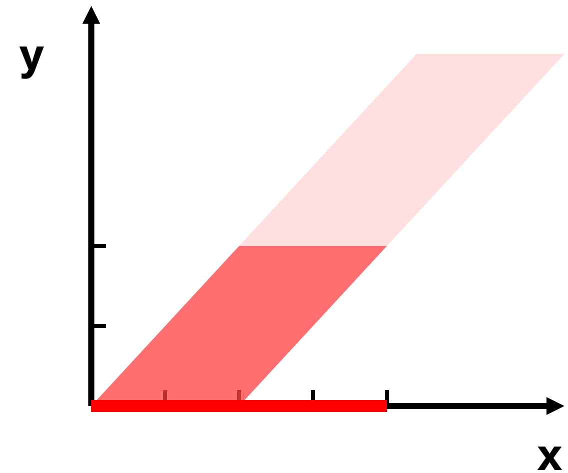




Left

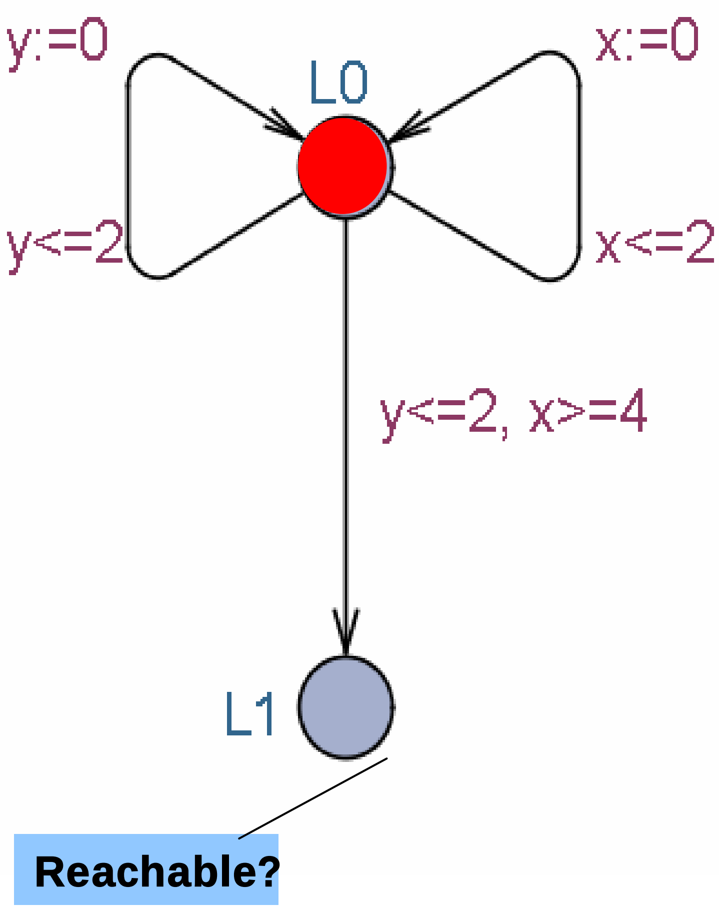
Symbolic Exploration

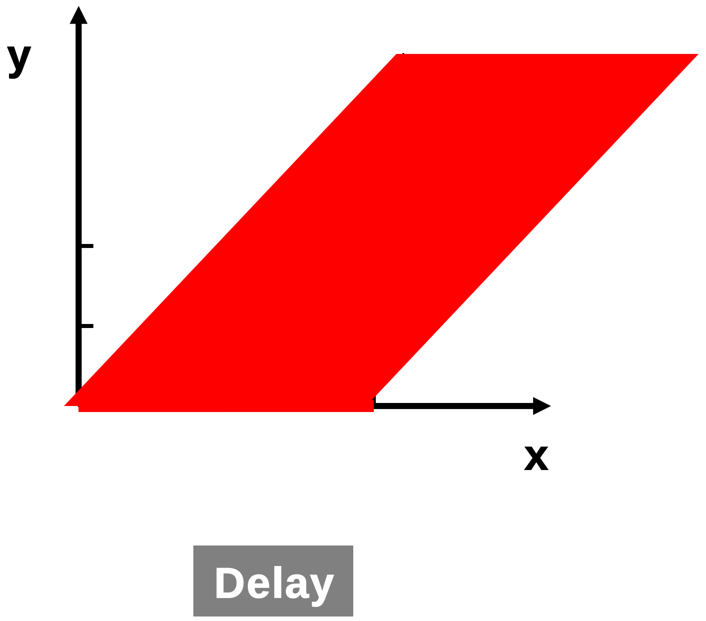




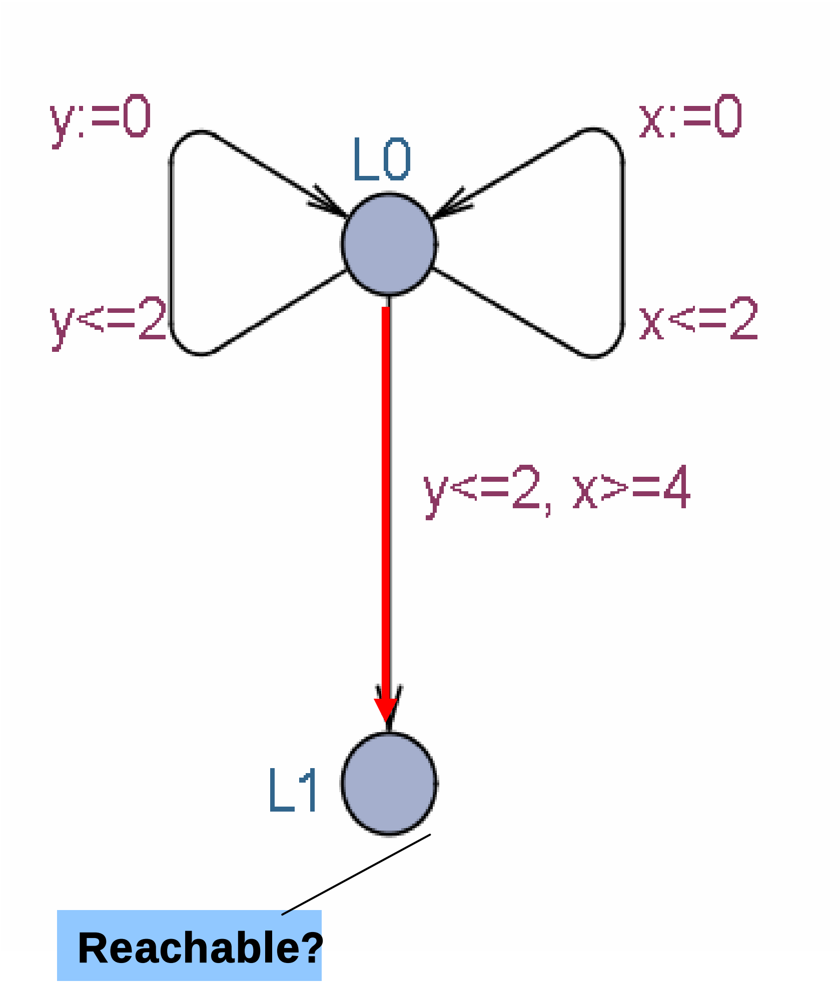
Left

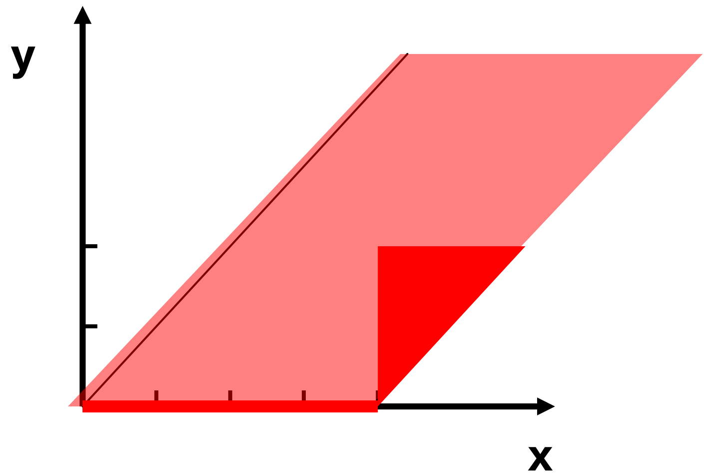
Symbolic Exploration





Symbolic Exploration

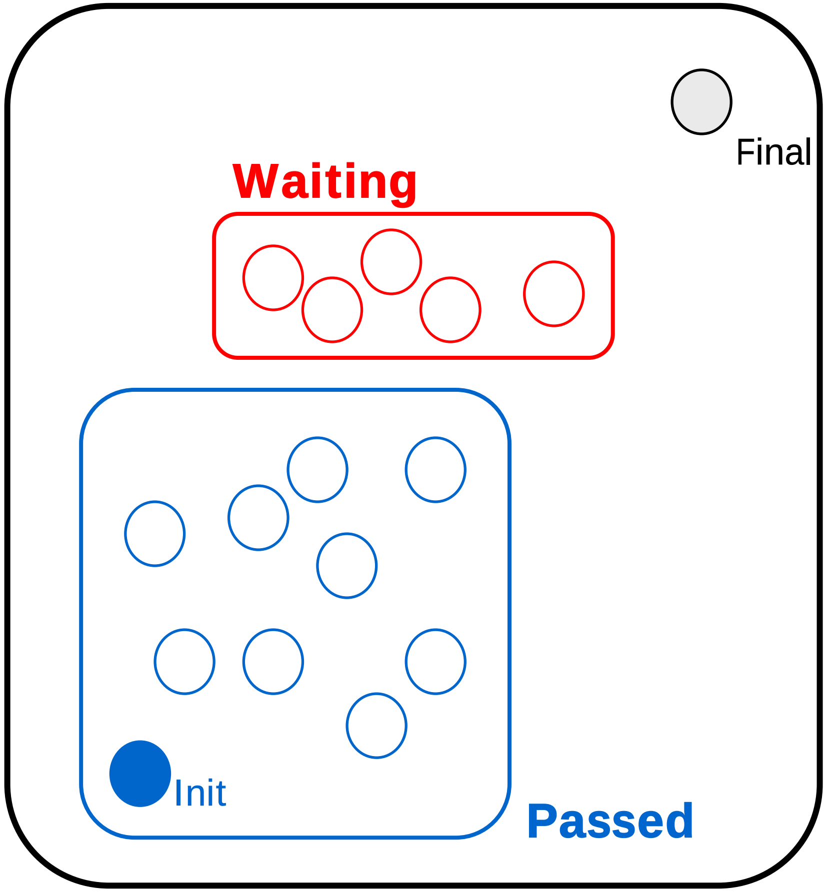




Down

Forward Rechability

Init -> Final ?



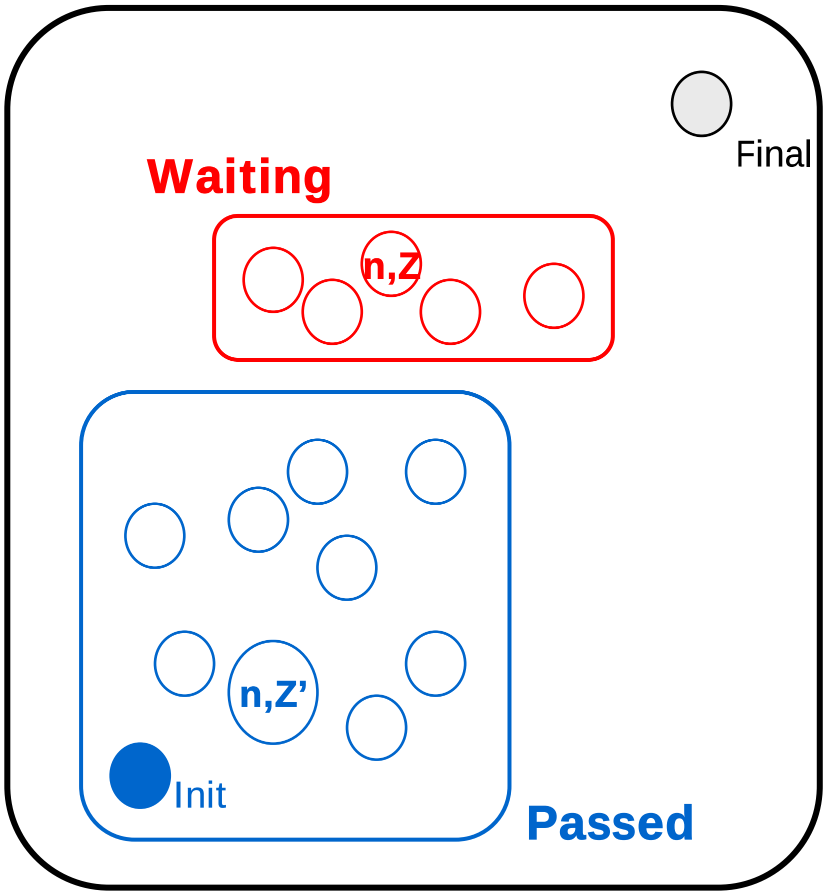
INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

REPEAT

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

Forward Rechability

Init -> Final ?



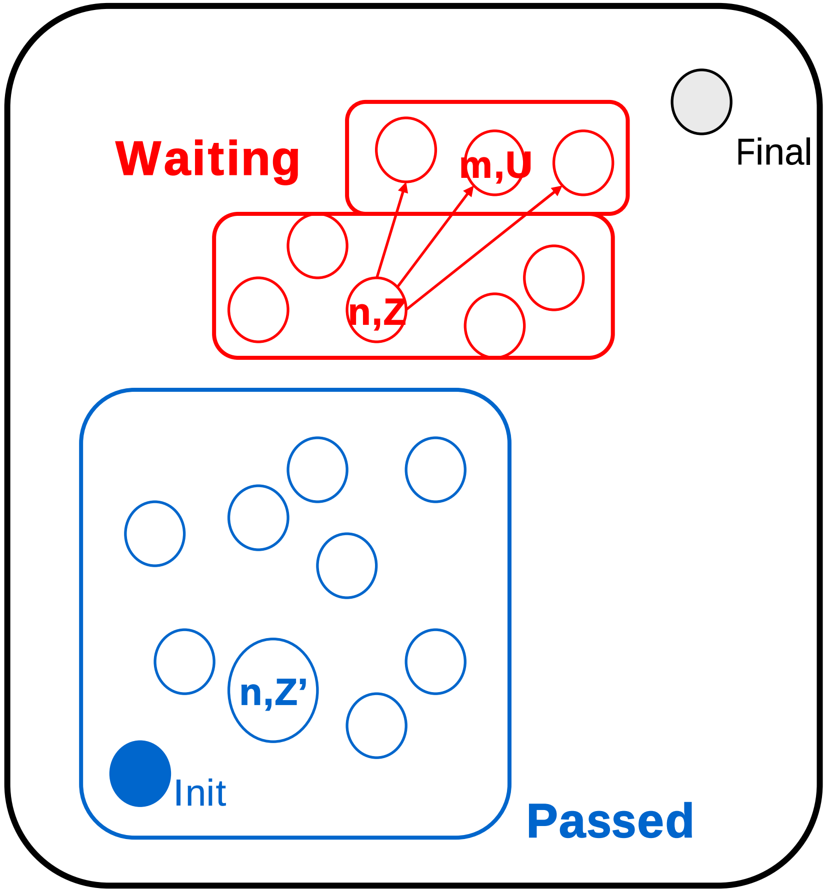
INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

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**

Forward Rechability

Init -> Final ?



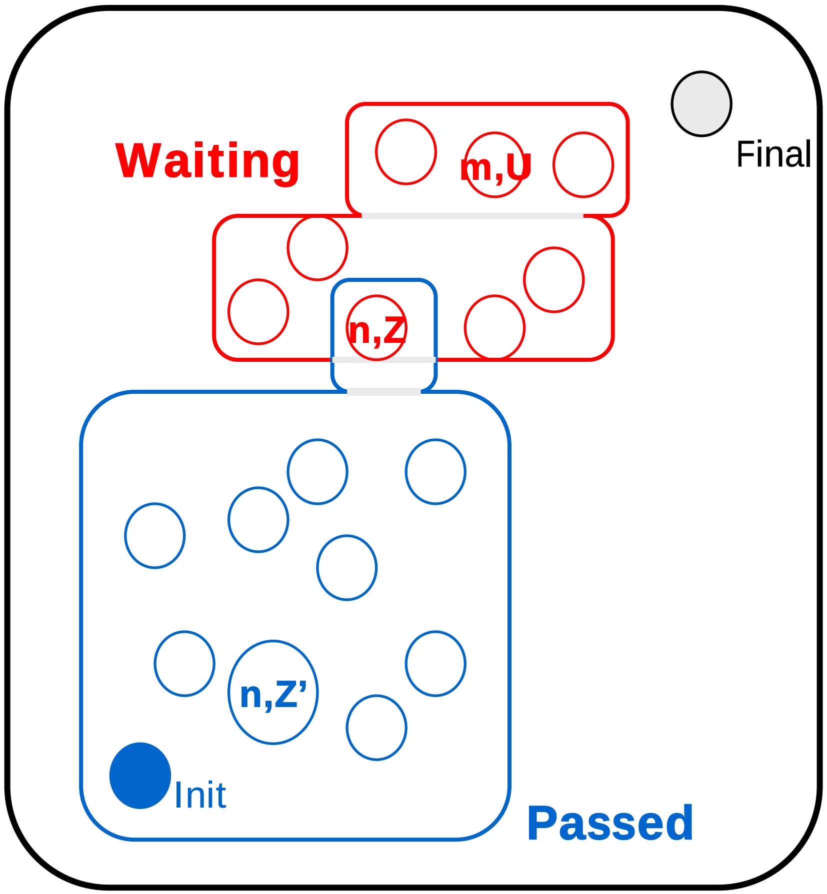
INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

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**

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

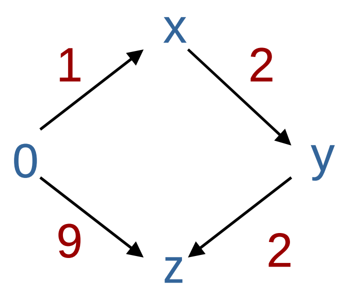
Bellman 1958, Dill 1989

Inclusion

D1

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

Graph

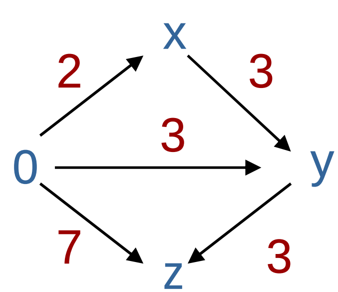


? $\subseteq$ **?**

D2

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

Graph



Canonical Datastructures for Zones

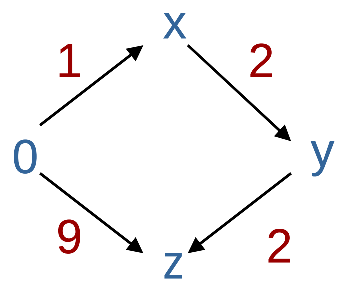
Difference Bounded Matrices

Inclusion

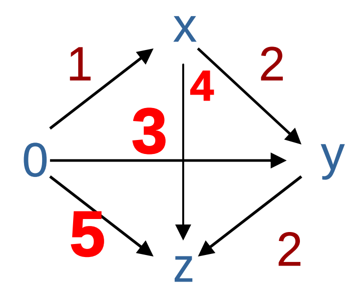
D1

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

Graph



Shortest Path Closure

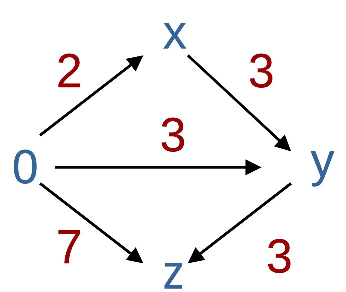


? $\subseteq$ **?**

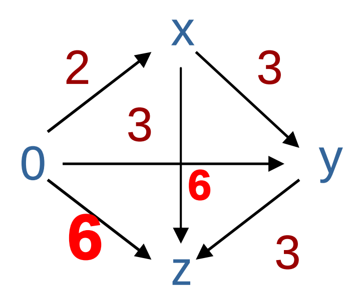
D2

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

Graph



Shortest Path Closure



Canonical Datastructures for Zones

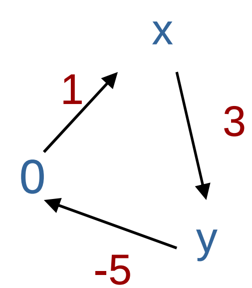
Difference Bounded Matrices

Emptiness

D

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

Graph



Negative Cycle
iff
empty solution set

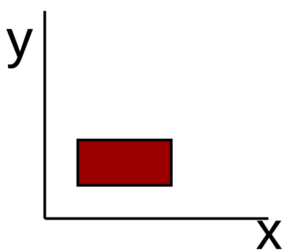
Compact

Canonical Datastructures for Zones

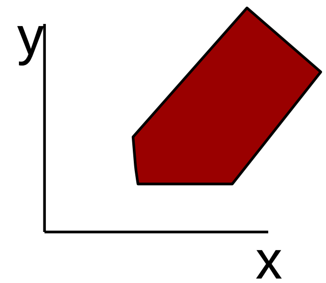
Difference Bounded Matrices

Future

D

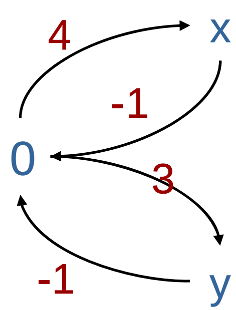


1<= x <=4
1<= y <=3

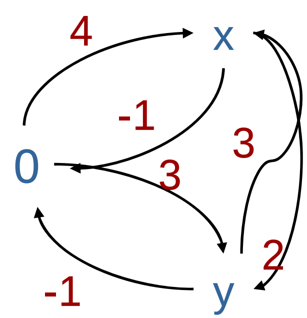


Future D

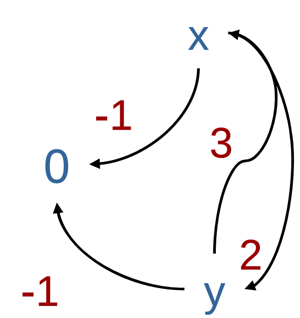
1<=x, 1<=y
-2<=x-y<=3



Shortest
Path
Closure



Remove
upper
bounds
on clocks



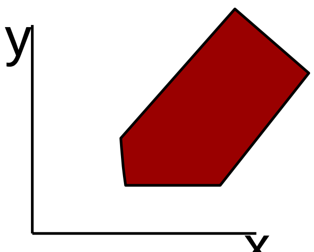
BRICS
Basic Research
in Computer Science

Canonical Datastructures for Zones

Difference Bounded Matrices

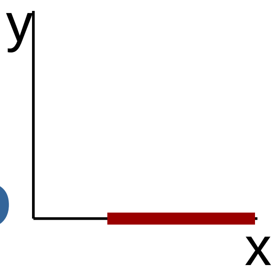
Reset

D

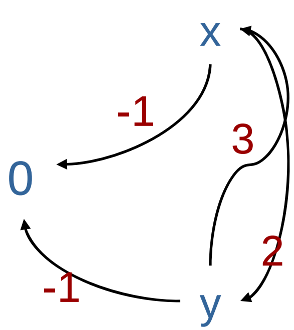


1<=x, 1<=y
-2<=x-y<=3

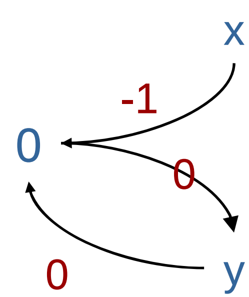
{y}D



y=0, 1<=x



Remove all bounds involving y and set y to 0

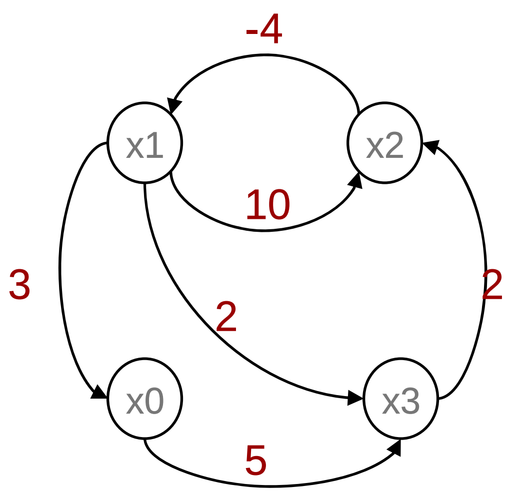


BRICS
Basic Research
in Computer Science

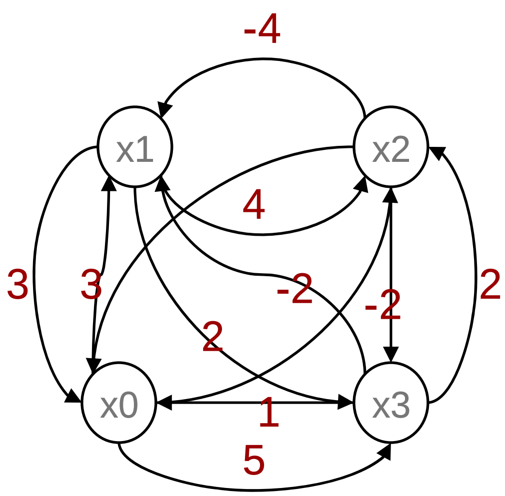
Canonical Datastructures for Zones

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)

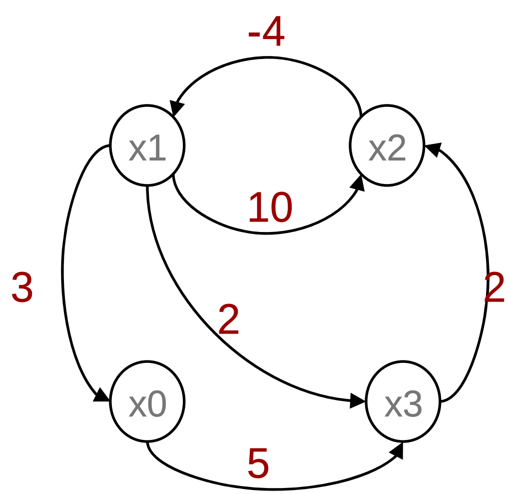


Canonical Datastructures for Zones

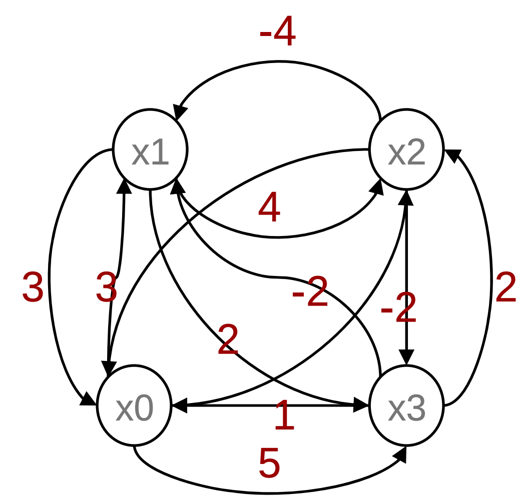
Minimal Constraint Form

RTSS 1997

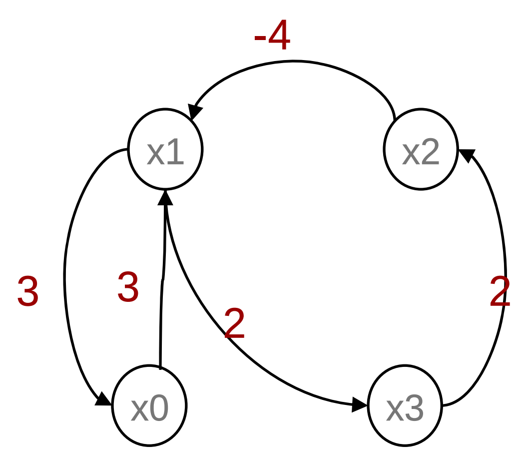
x1-x2<=4
x2-x1<=10
x3-x1<=2
x2-x3<=2
x0-x1<=3
x3-x0<=5



Shortest Path Closure O(n^3)

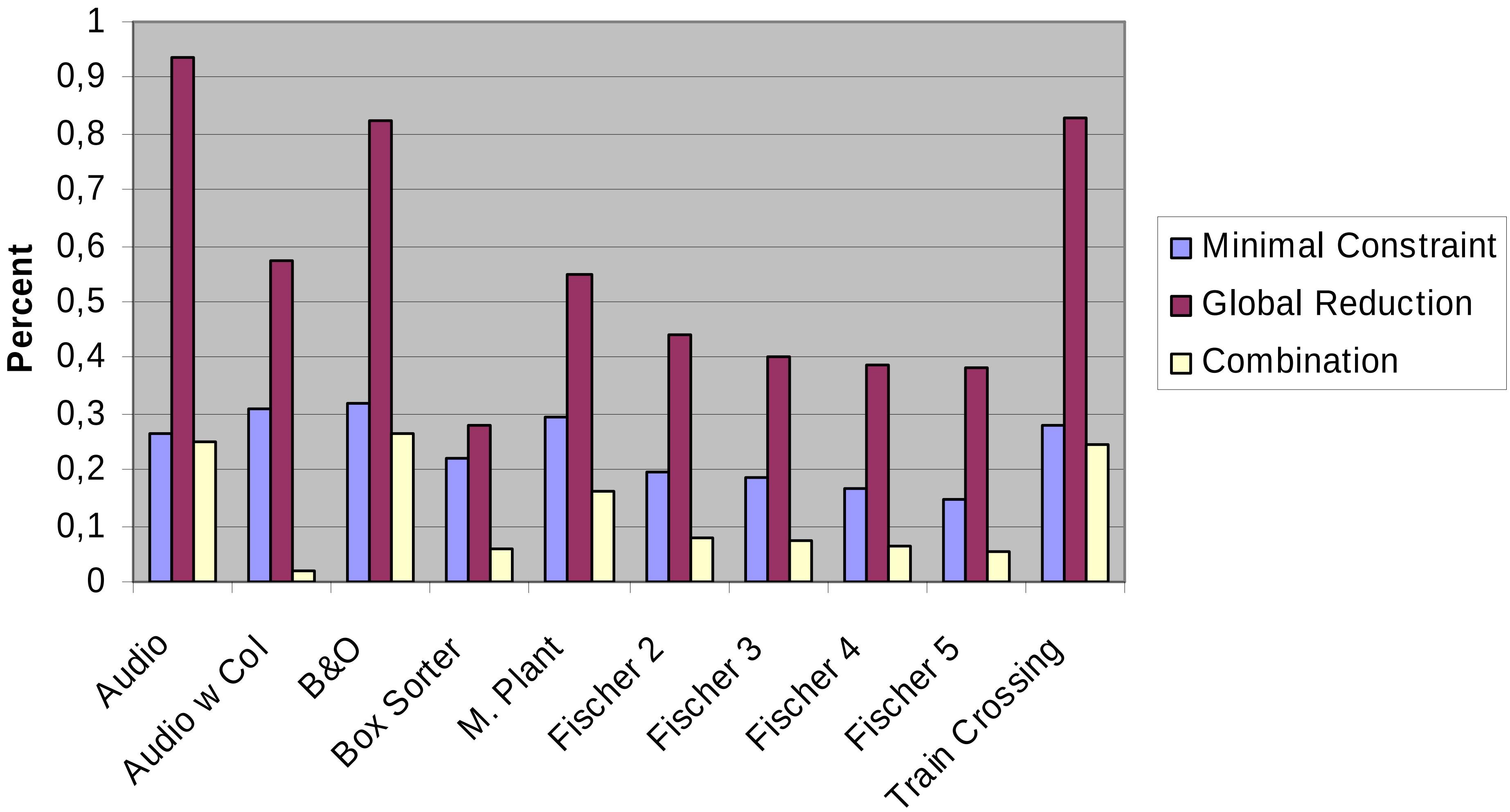


Shortest Path Reduction O(n^3)

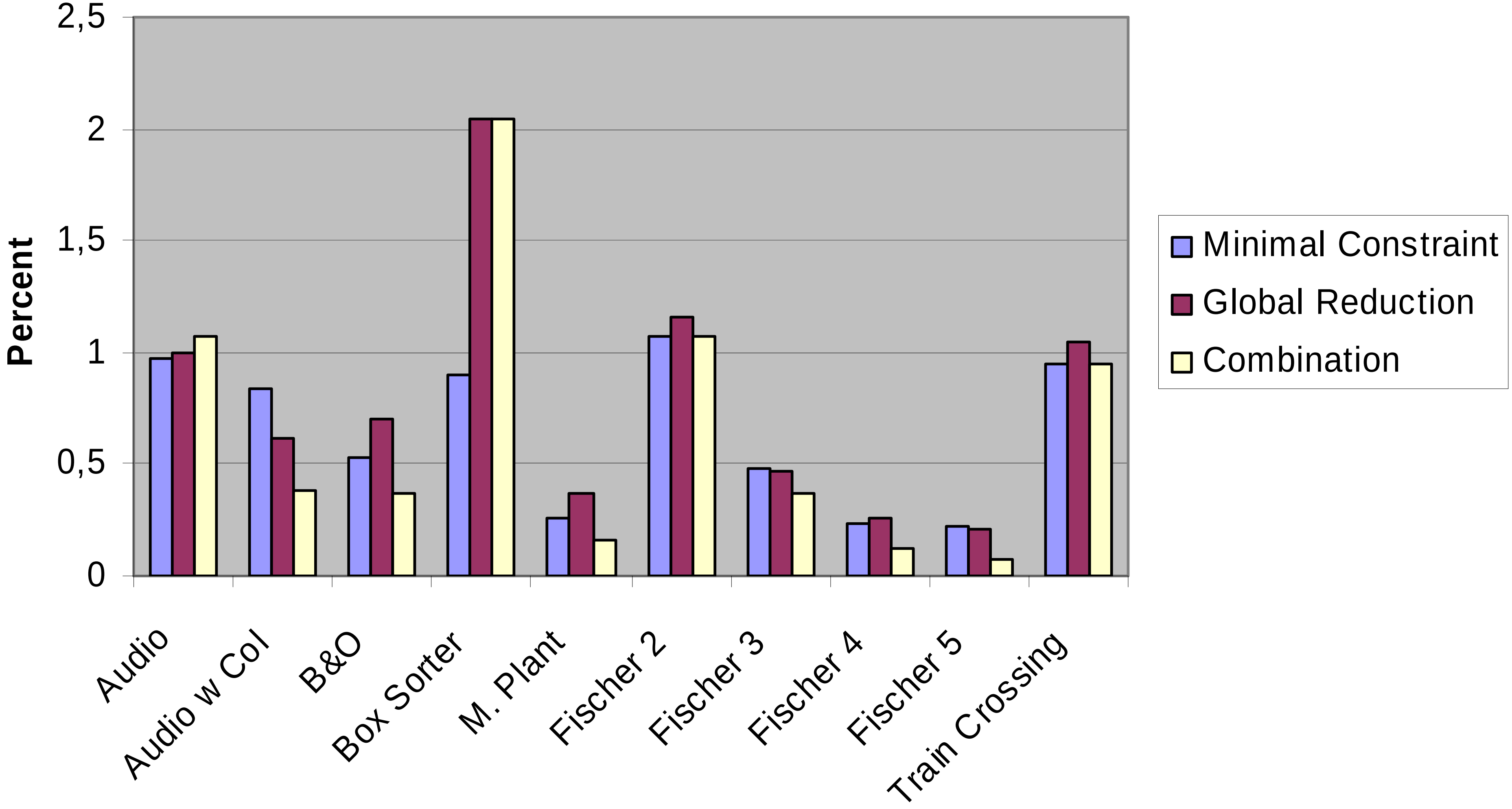


Space worst O(n^2)
practice O(n)

SPACE PERFORMANCE



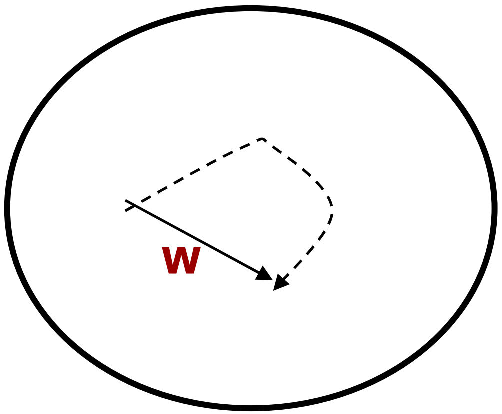
TIME PERFORMANCE



Shortest Path Reduction

1st attempt

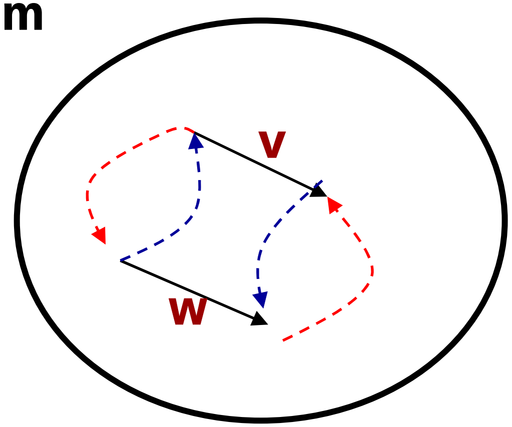
Idea



An edge is REDUNDANT if there exists an alternative path of no greater weight

THUS Remove all redundant edges!

Problem



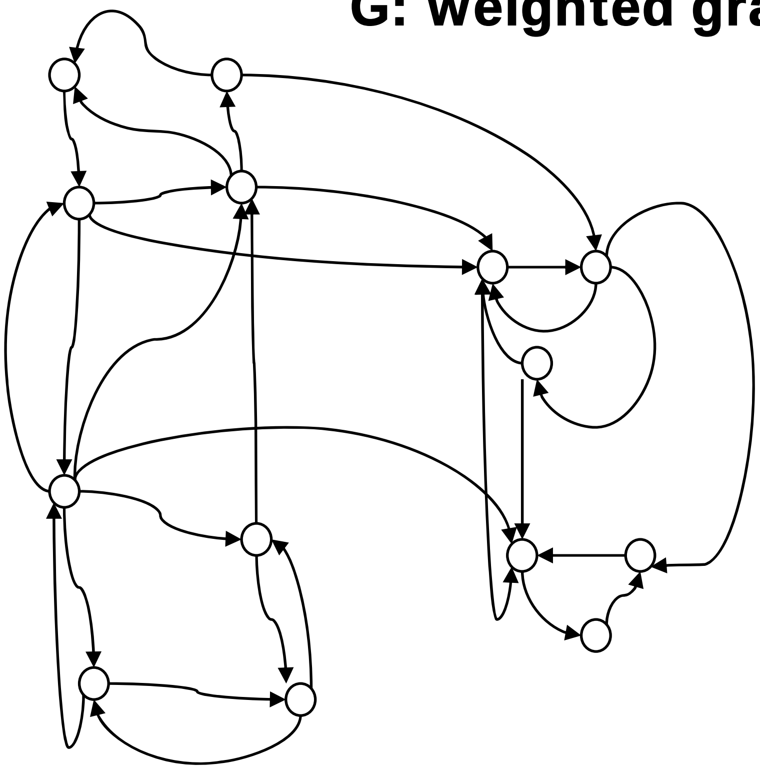
v and **w** are both redundant
Removal of one depends on presence of other.

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

Shortest Path Reduction

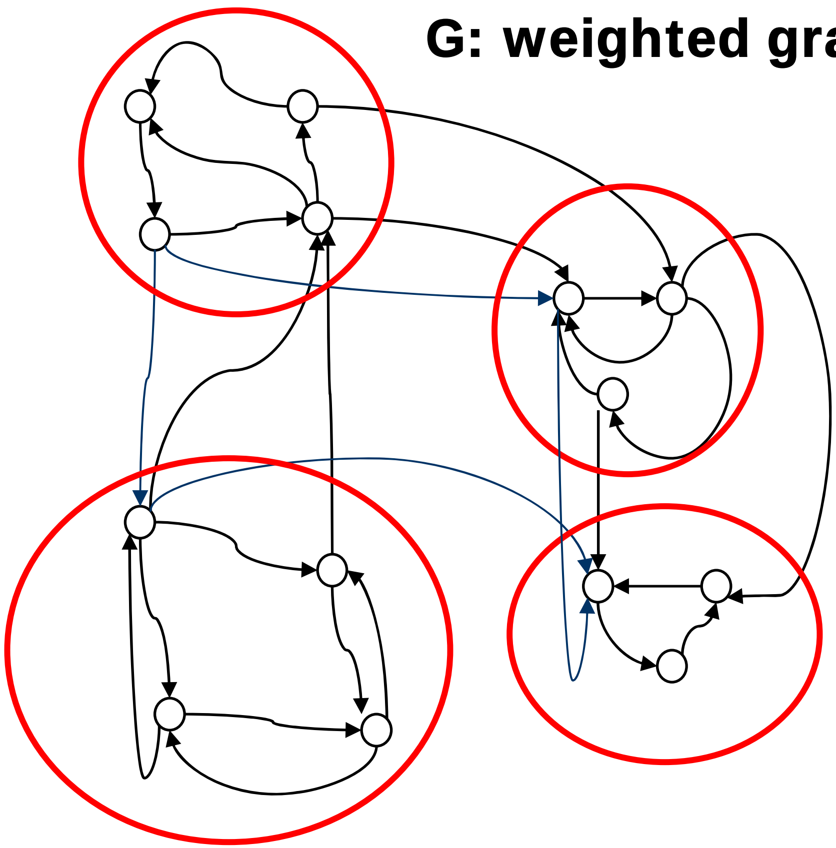
Solution

G: weighted graph



Shortest Path Reduction

Solution

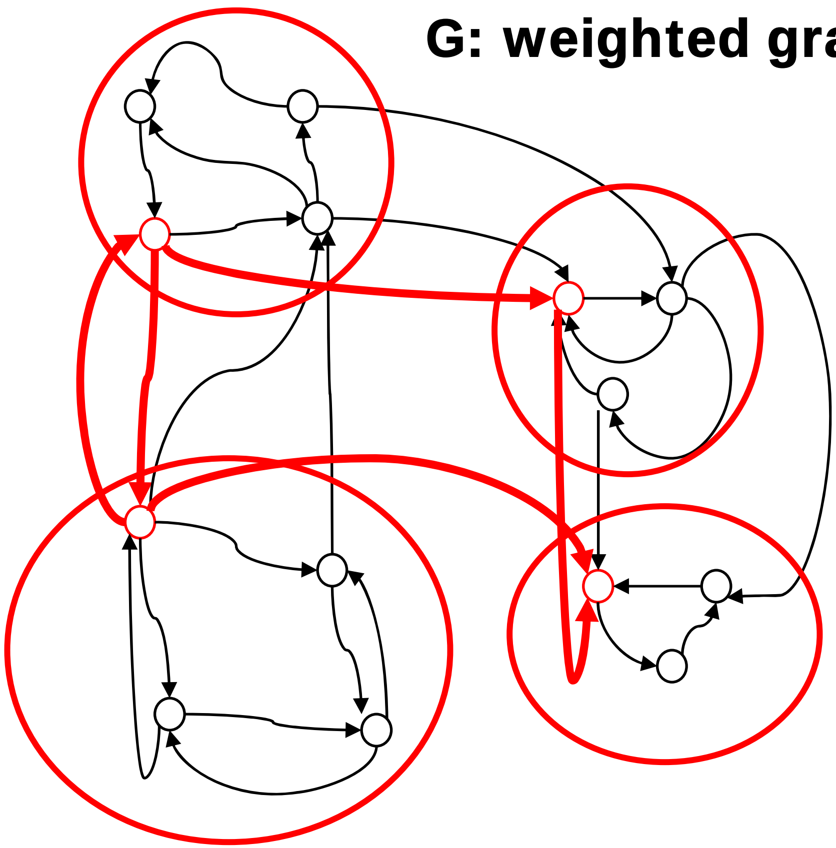


1. Equivalence classes based on 0-cycles.

Shortest Path Reduction

Solution

G: weighted graph



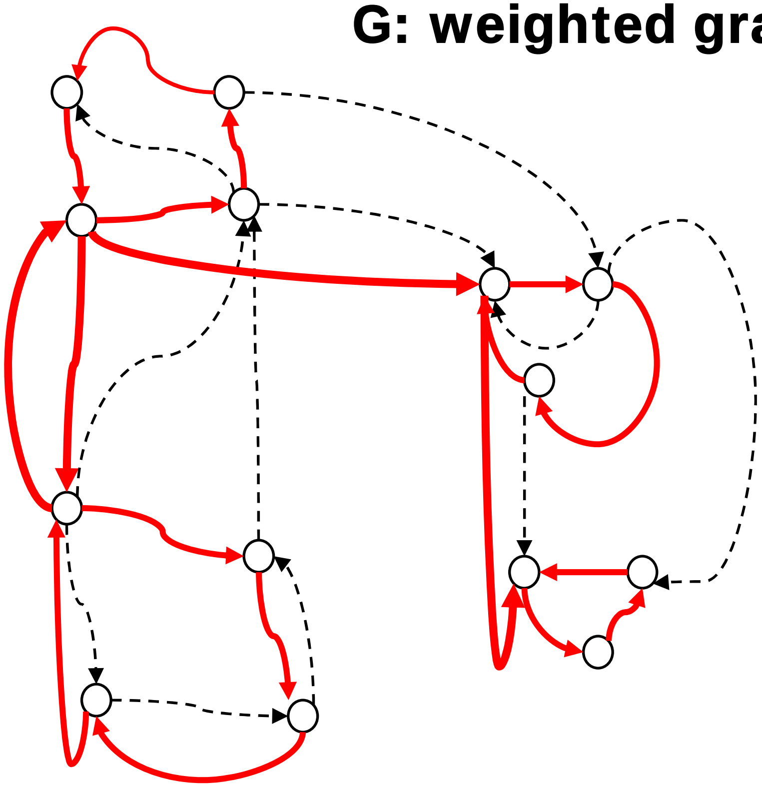
1. Equivalence classes based on 0-cycles.

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

Shortest Path Reduction

Solution

G: weighted graph



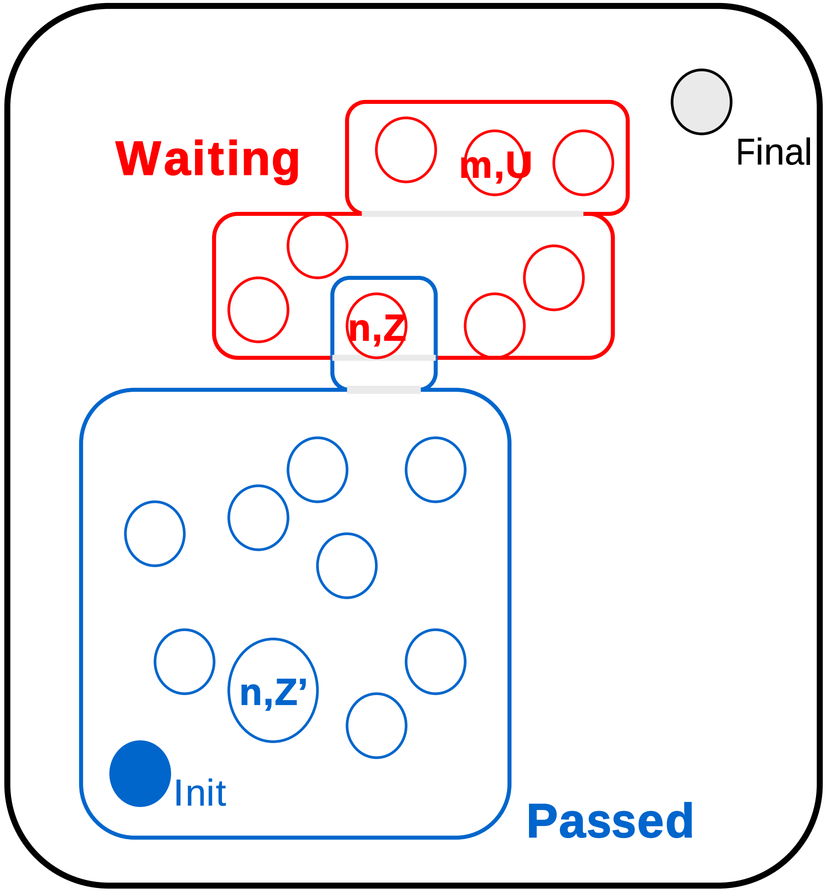
1. Equivalence classes based on 0-cycles.
2. Graph based on representatives. Safe to remove redundant edges
3. **Shortest Path Reduction** = One cycle pr. class + Removal of redundant edges between classes

Canonical given order of clocks

BRICS Basic Research in Computer Science

Earlier Termination

Init -> Final ?



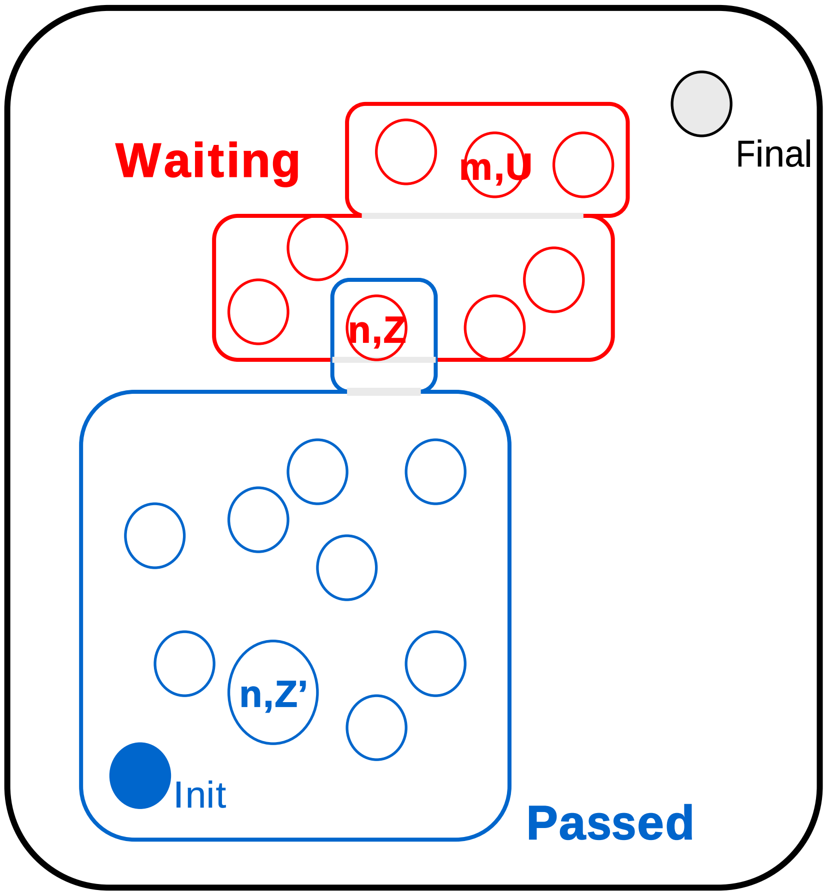
INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

REPEAT
- pick (n,Z) in **Waiting**
- **if** for some Z' ⊒ 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**

Earlier Termination

Init -> Final ?



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

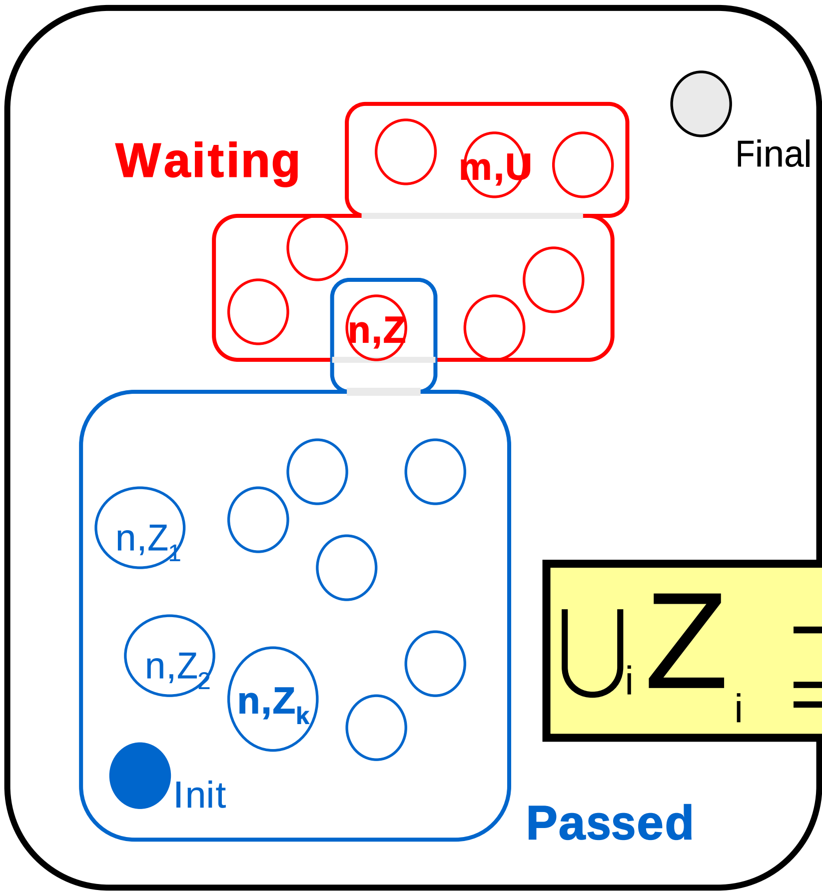
REPEAT

- 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**

Earlier Termination

Init -> Final ?



$$\bigcup_i Z_i \supseteq Z$$

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**

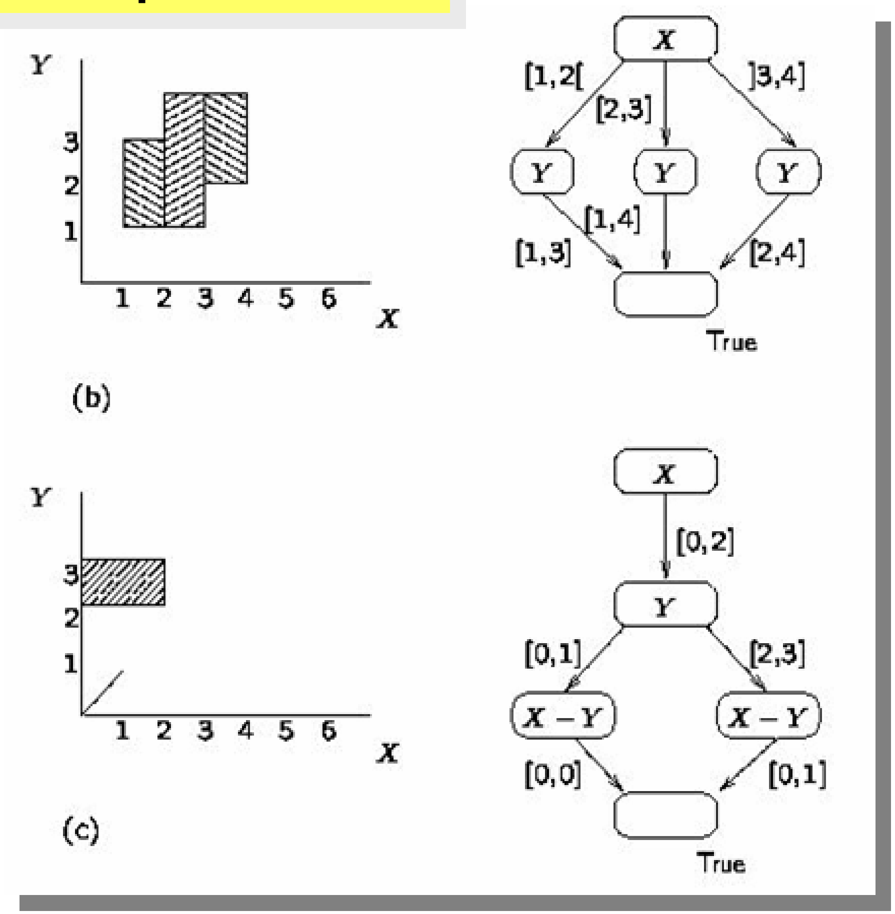
UNTIL **Waiting** = Ø
or
Final is in **Waiting**

Clock Difference Diagrams

= Binary Decision Diagrams + Difference Bounded Matrices

CAV99

CDD-representations

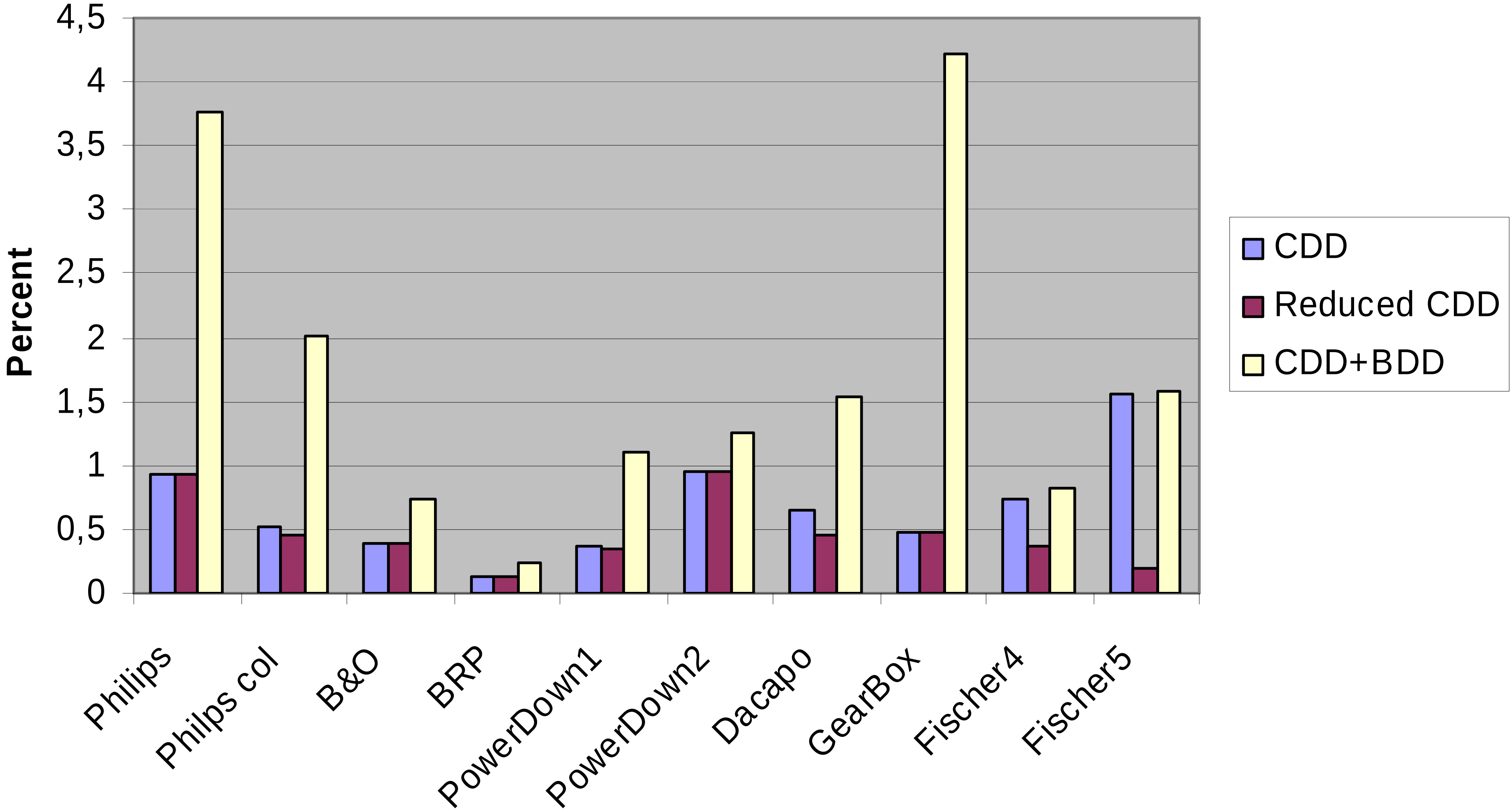


- Nodes labeled with differences
- Maximal sharing of substructures (also across different CDDs)
- Maximal intervals
- Linear-time algorithms for set-theoretic operations.

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

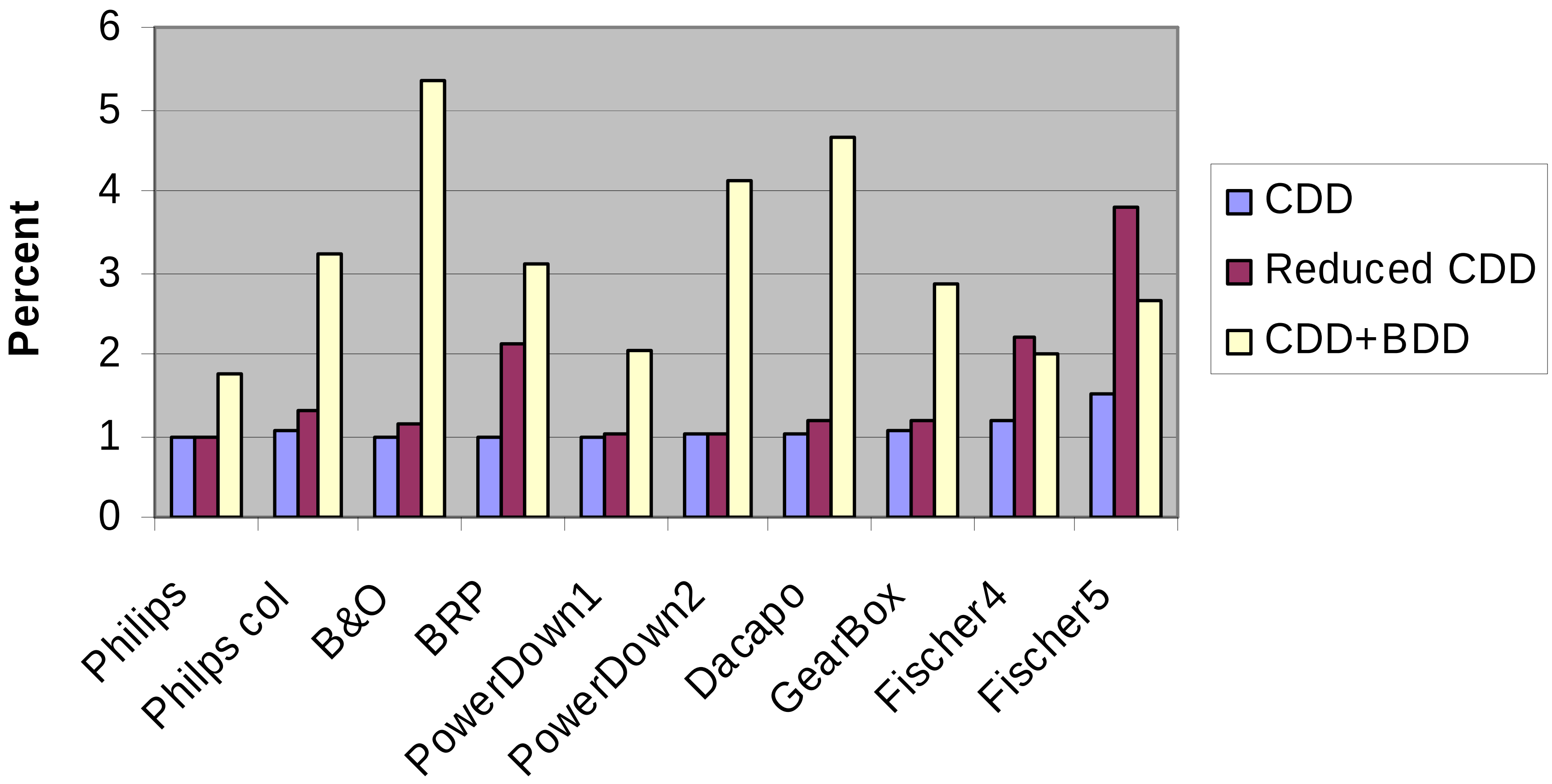
BRICS Basic Research in Computer Science

SPACE PERFORMANCE



in Computer Science

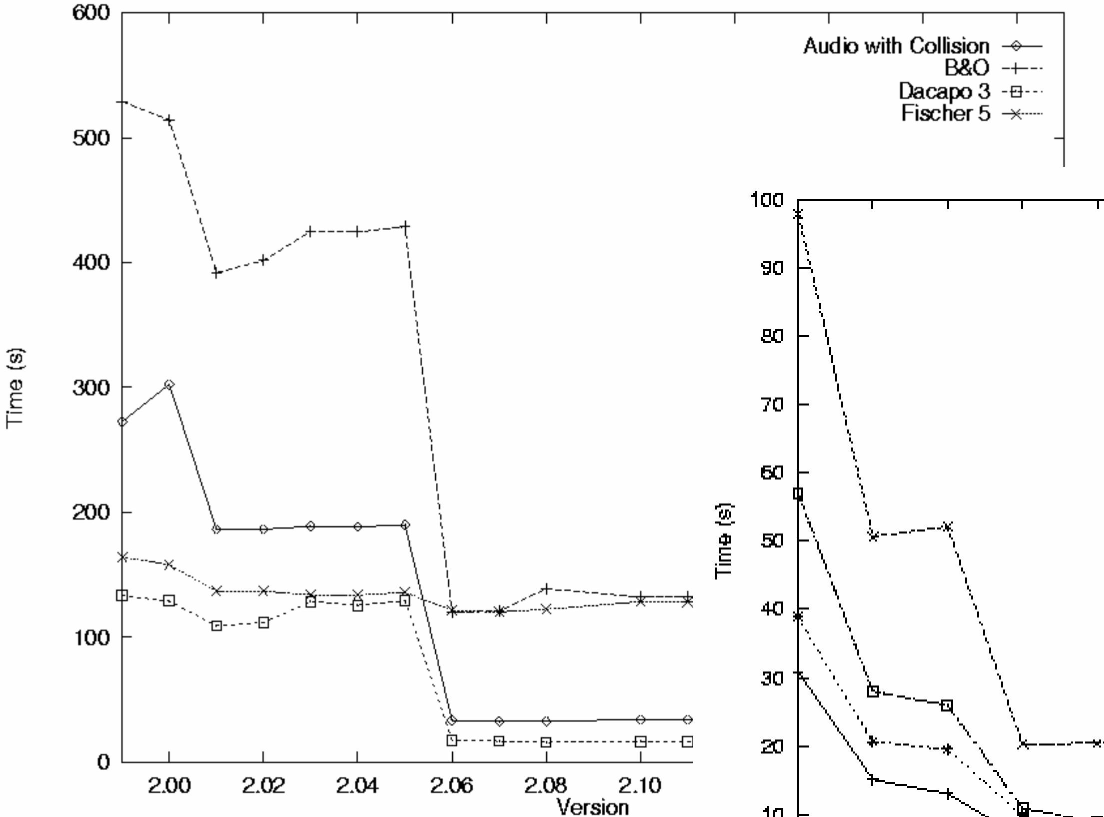
TIME PERFORMANCE



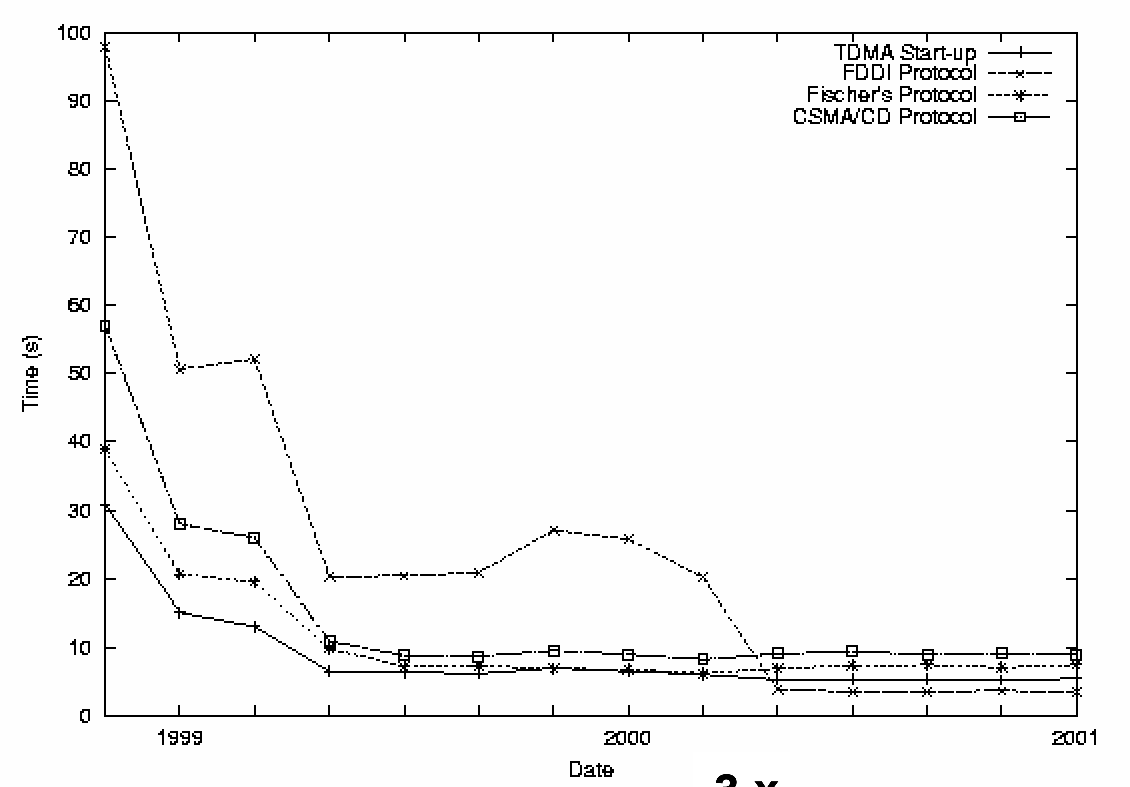
BRICS
Basic Research
in Computer Science

UPPAAL 1995 - 2001

Every 9 month 10 times better performance!



Dec'96



3.x

Liveness Properties

in UPPAAL

$F ::= E\Box\, P \quad |$ — **Possibly always** P

$A\Diamond\, P \quad |$ — **Eventually** P
is equivalent to $(\neg\, E\Box\, \neg\, P)$

$P \rightarrow Q$ — P **leads to** Q
is equivalent to
$A\Box\, (\, P \Rightarrow A\Diamond\, Q)$

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

Liveness

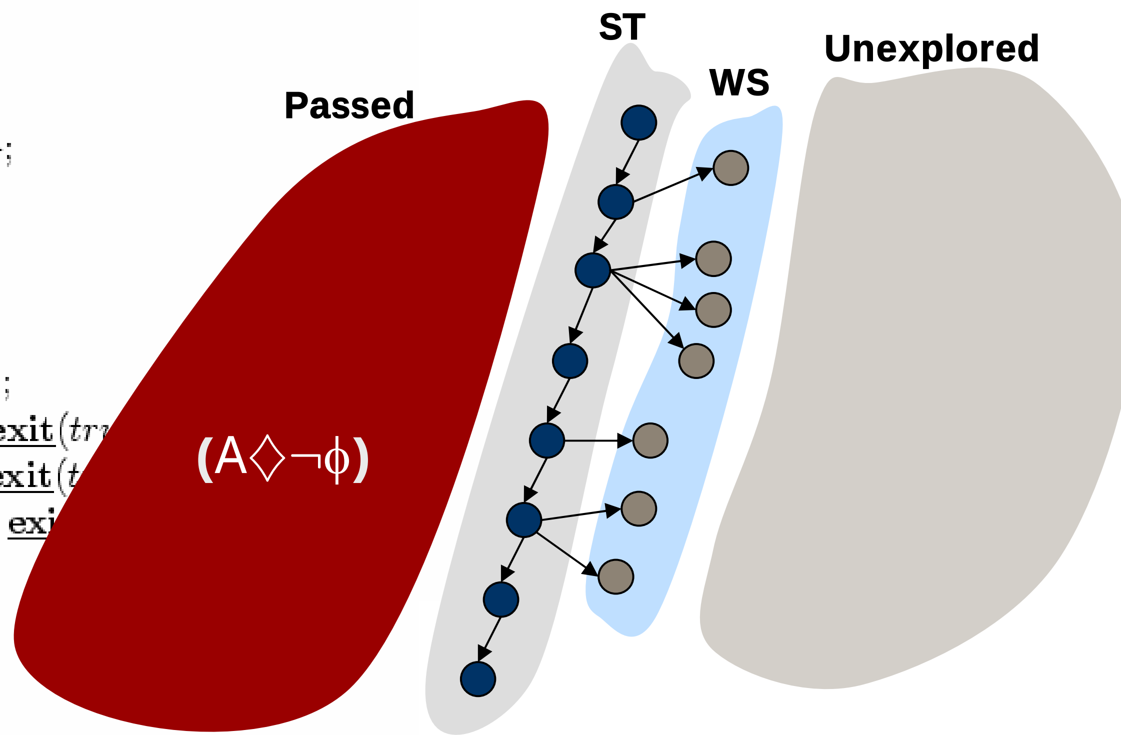
$E[]\phi \qquad (A\Diamond\neg\phi)$

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

Liveness

E[]ϕ (A◇¬ϕ)

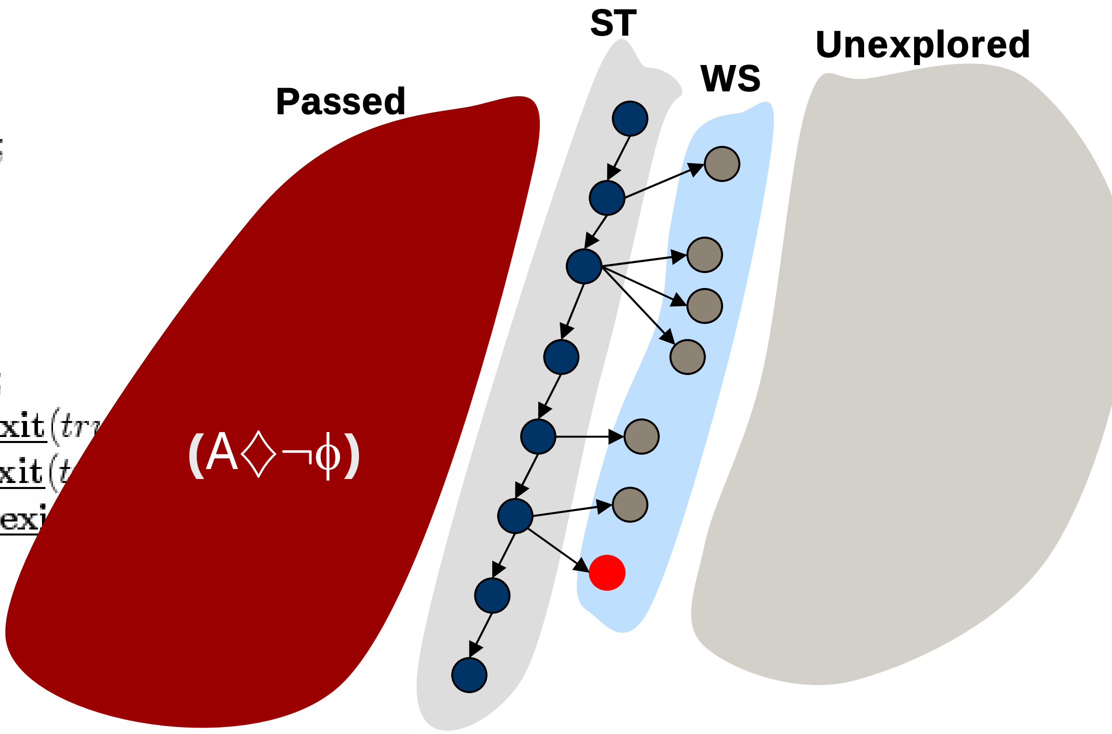
```
proc Liveness(s0, φ, Passed) ≡
  pre(s0 = delay(s0))
  pre(s0 ⊨ φ)
  pre(¬unboundeds0 ∧ ¬deadlocked(s0))
  pre(∀s ∈ Passed. s ⊨ A◇¬φ)
  WS := {s0};
  ST := ∅;
  while WS ≠ ∅ do
        s := pop(WS);
        while top(ST) ≠ parent(s) do
              Passed := Passed ∪ {pop(ST)};
        od
        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
```



Liveness

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

```
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
```

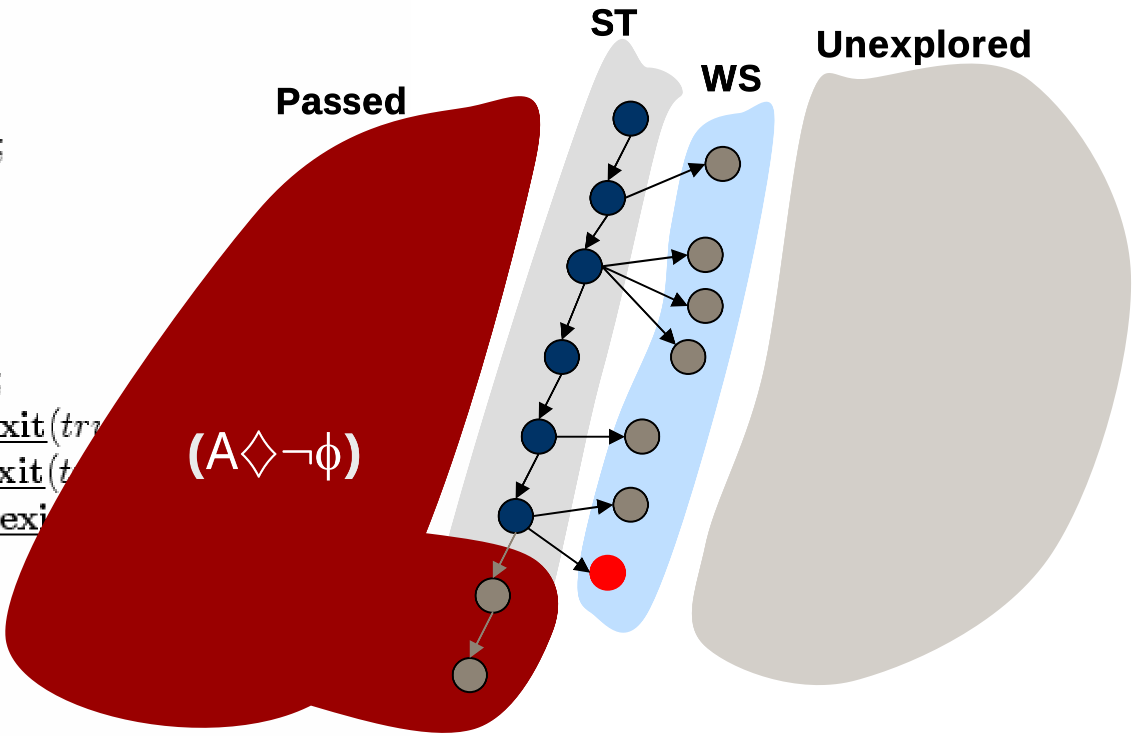


in UPPAAL

Liveness

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

```
proc Liveness(s0, φ, Passed) ≡
  pre(s0 = delay(s0))
  pre(s0 ⊨ φ)
  pre(¬unbounded s0 ∧ ¬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
```

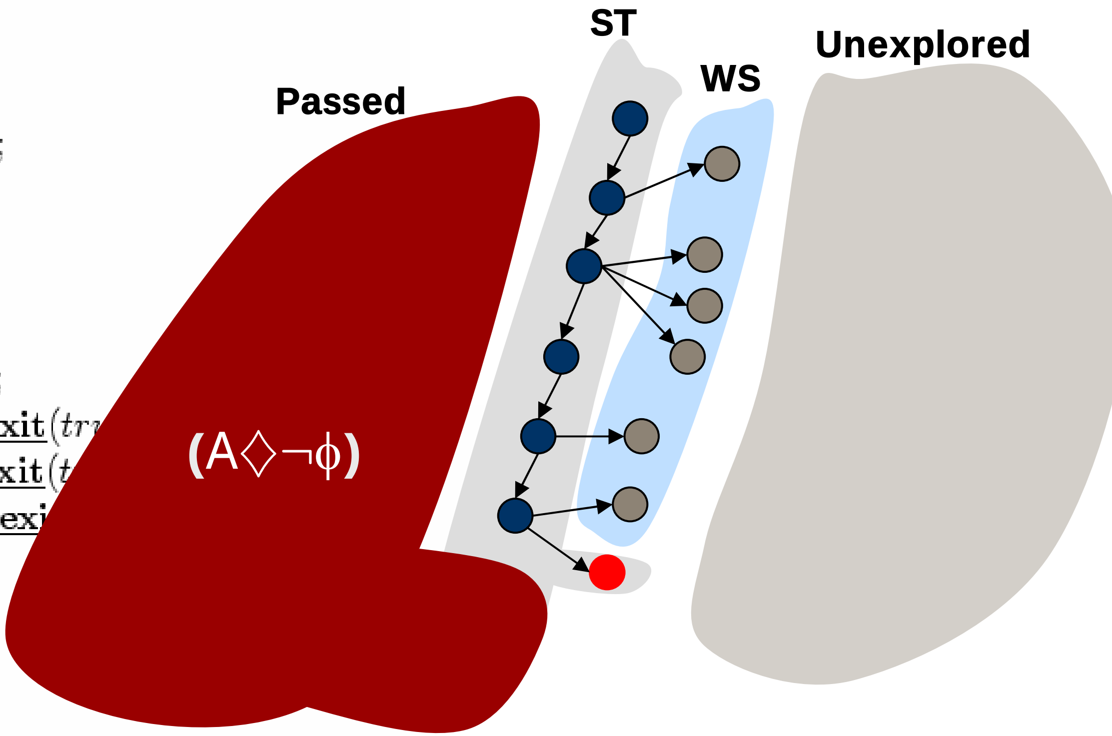


BRICS
Basic Research
in Computer Science

Liveness

E□ϕ (A◇¬ϕ)

```
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
```



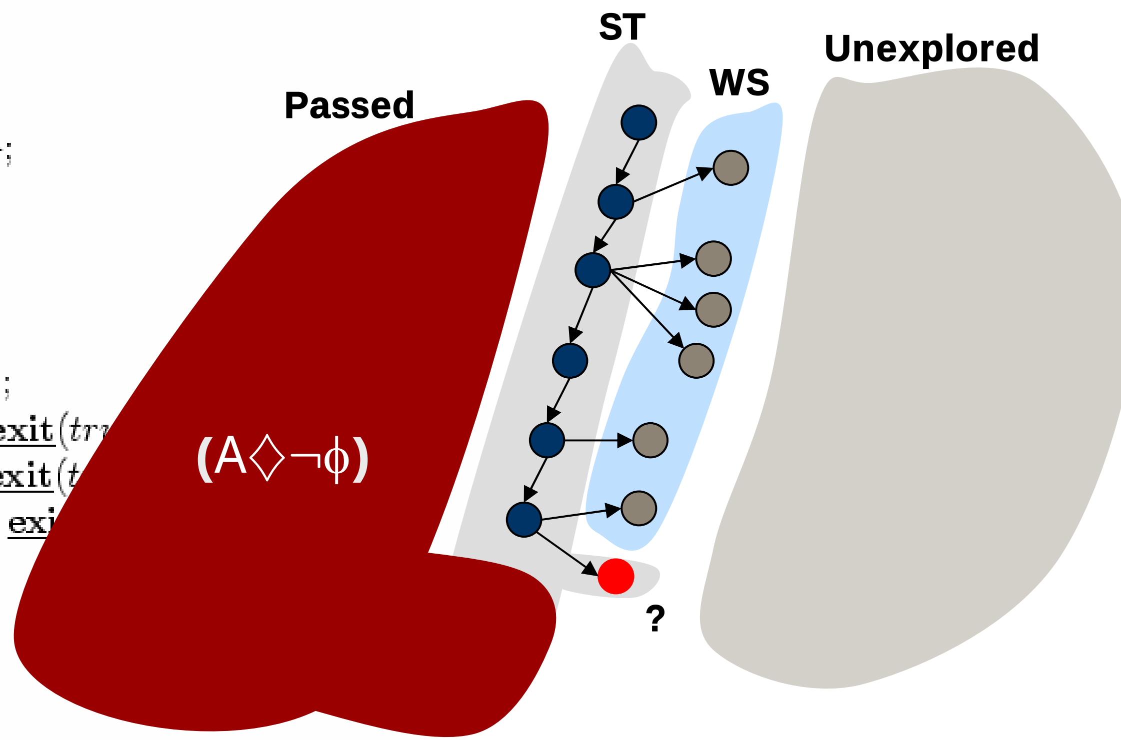
in UPPAAL

BRICS Basic Research in Computer Science

Liveness in UPPAAL

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

```
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
```

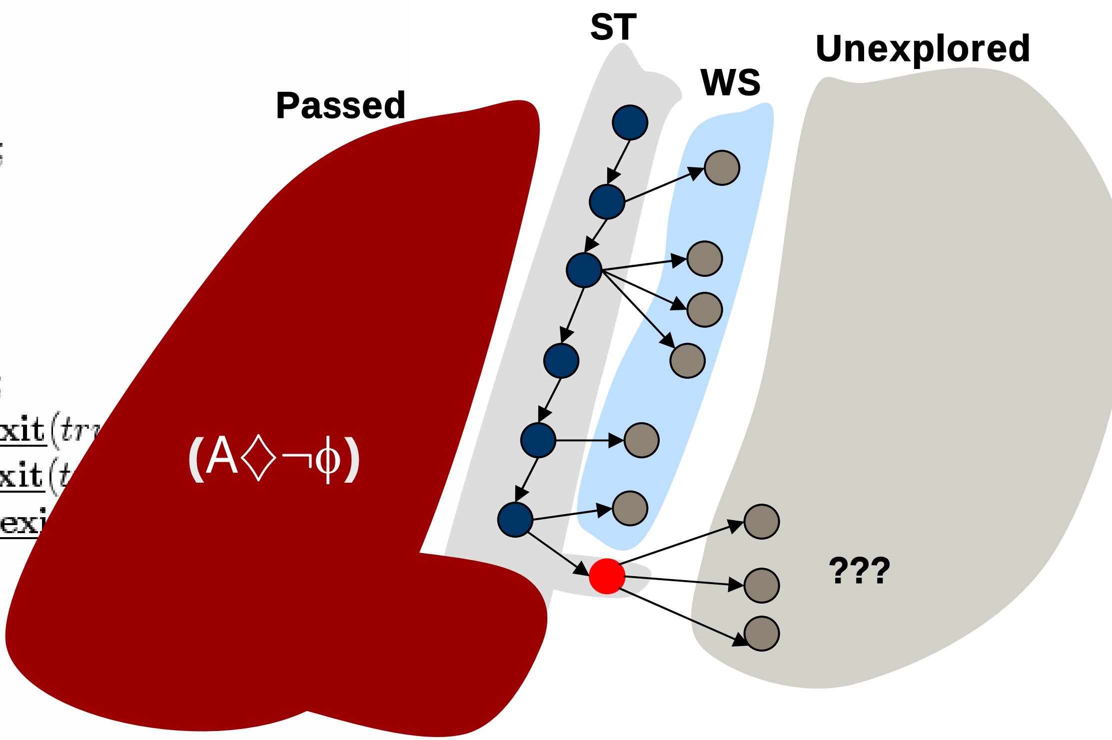


Liveness

in UPPAAL

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

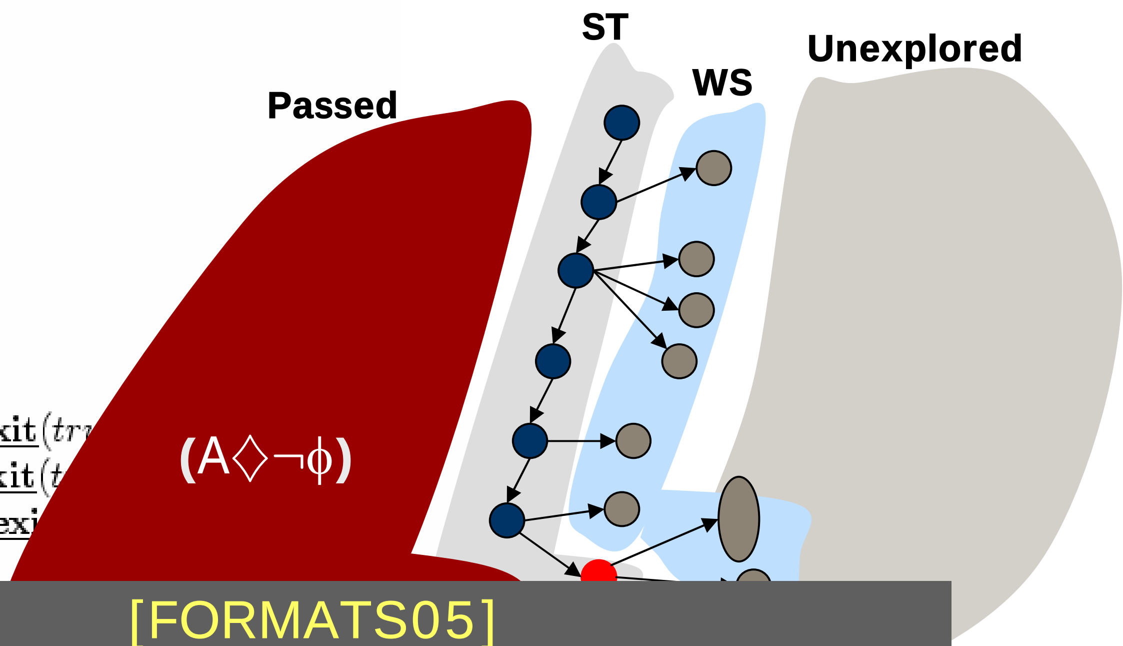
$\underline{\text{proc}}\ \mathit{Liveness}(s_0, \varphi, \mathit{Passed}) \equiv$
    $\mathit{pre}(s_0 = \mathit{delay}(s_0))$
    $\mathit{pre}(s_0 \models \varphi)$
    $\mathit{pre}(\neg \mathit{unbounded} s_0 \wedge \neg \mathit{deadlocked}(s_0))$
    $\mathit{pre}(\forall s \in \mathit{Passed}.\,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



Liveness

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

```
proc Liveness(s0, φ, Passed) ≡
  pre(s0 = delay(s0))
  pre(s0 ⊨ φ)
  pre(¬unbounded s0 ∧ ¬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
```

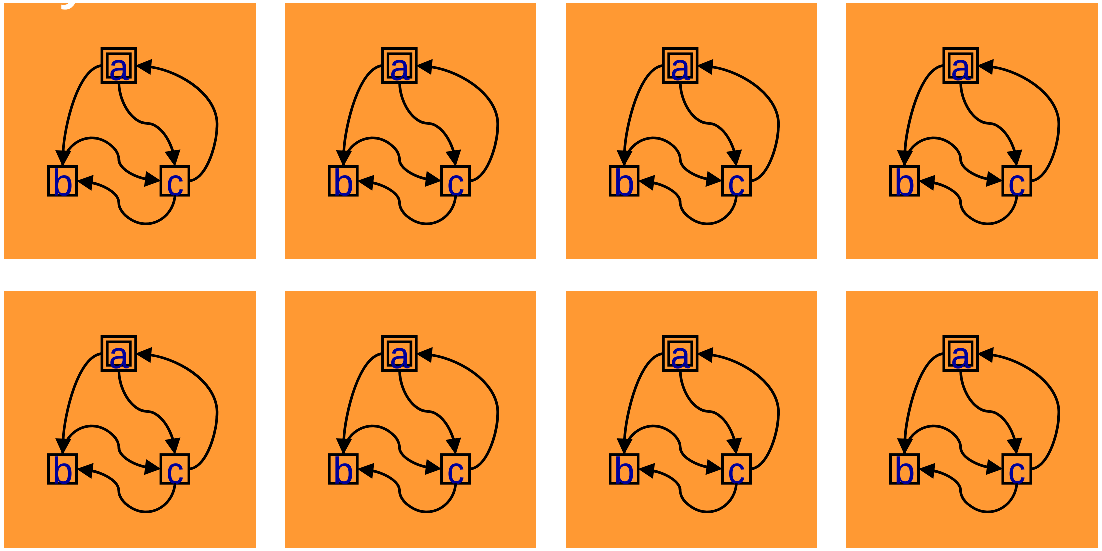


[FORMATS05]

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

Compositionality & Abstraction

The State Explosion Problem

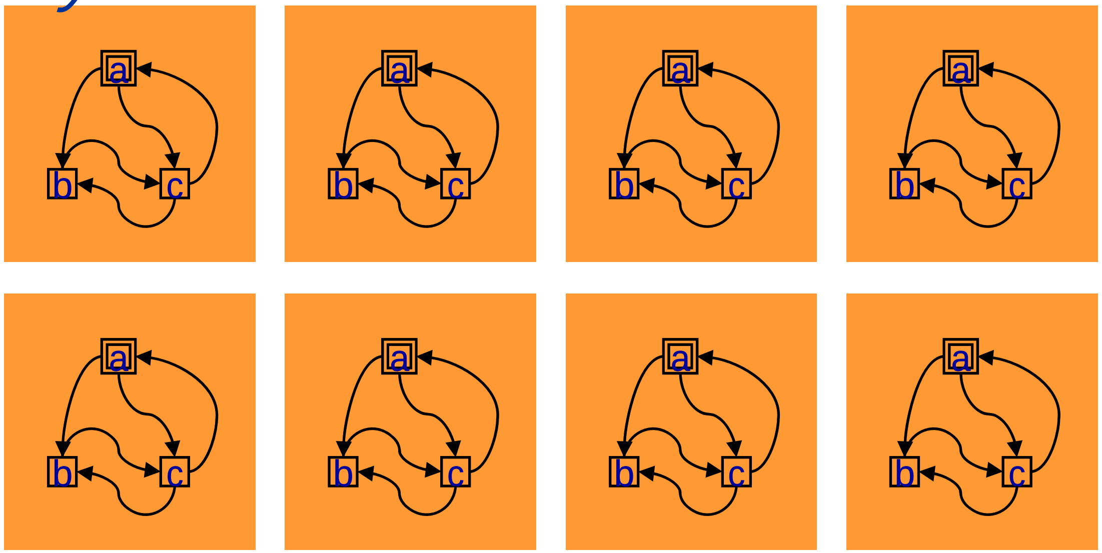


sat φ

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

Abstraction

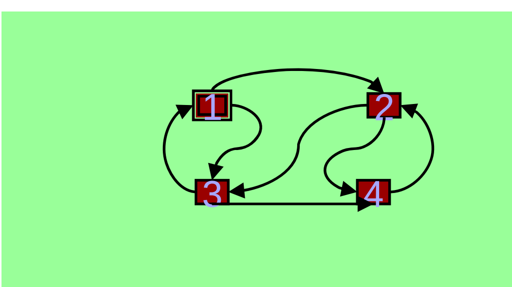
Sys



sat φ

REDUCE TO

Abs

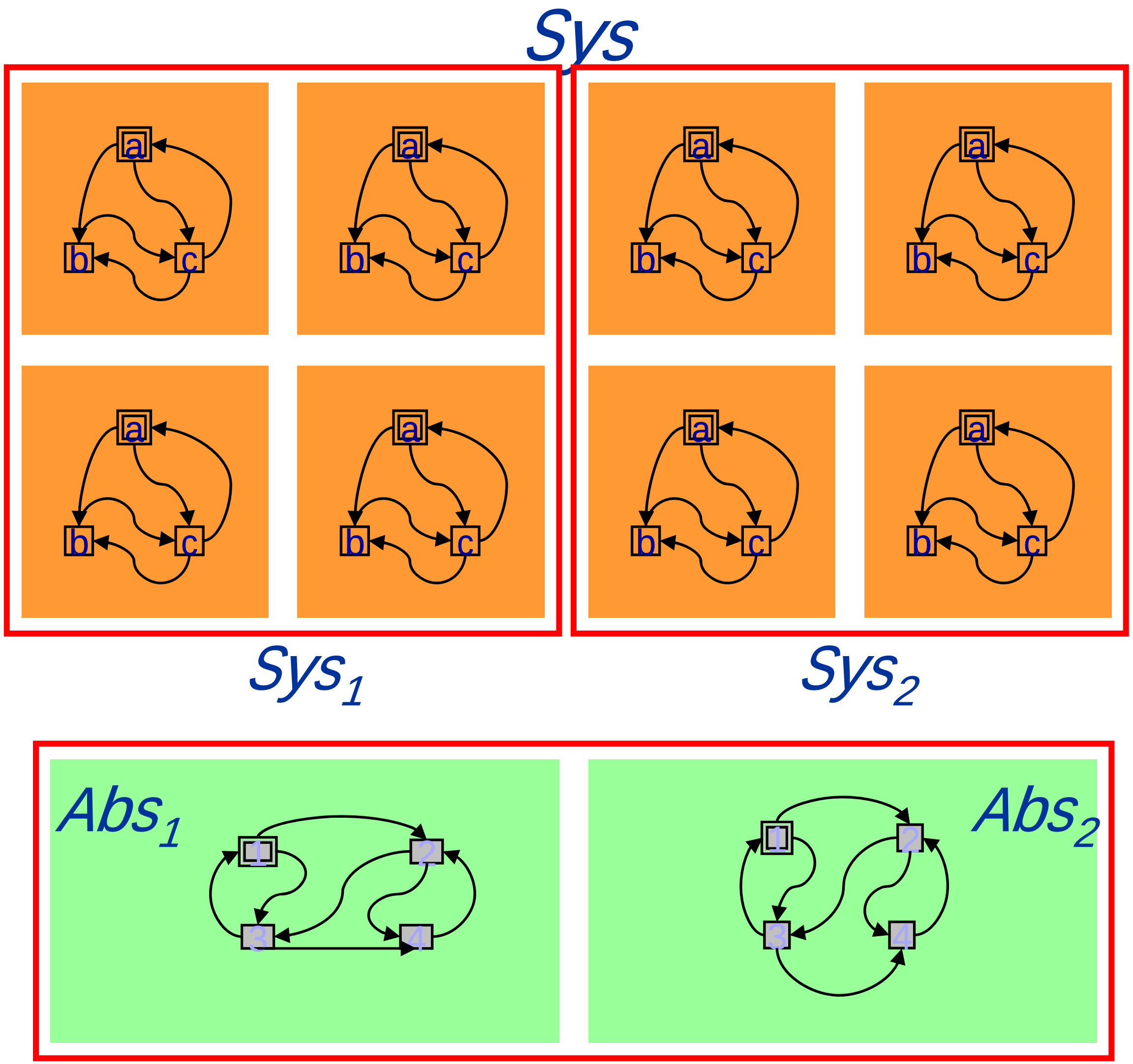


sat φ

Preserving safety properties

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

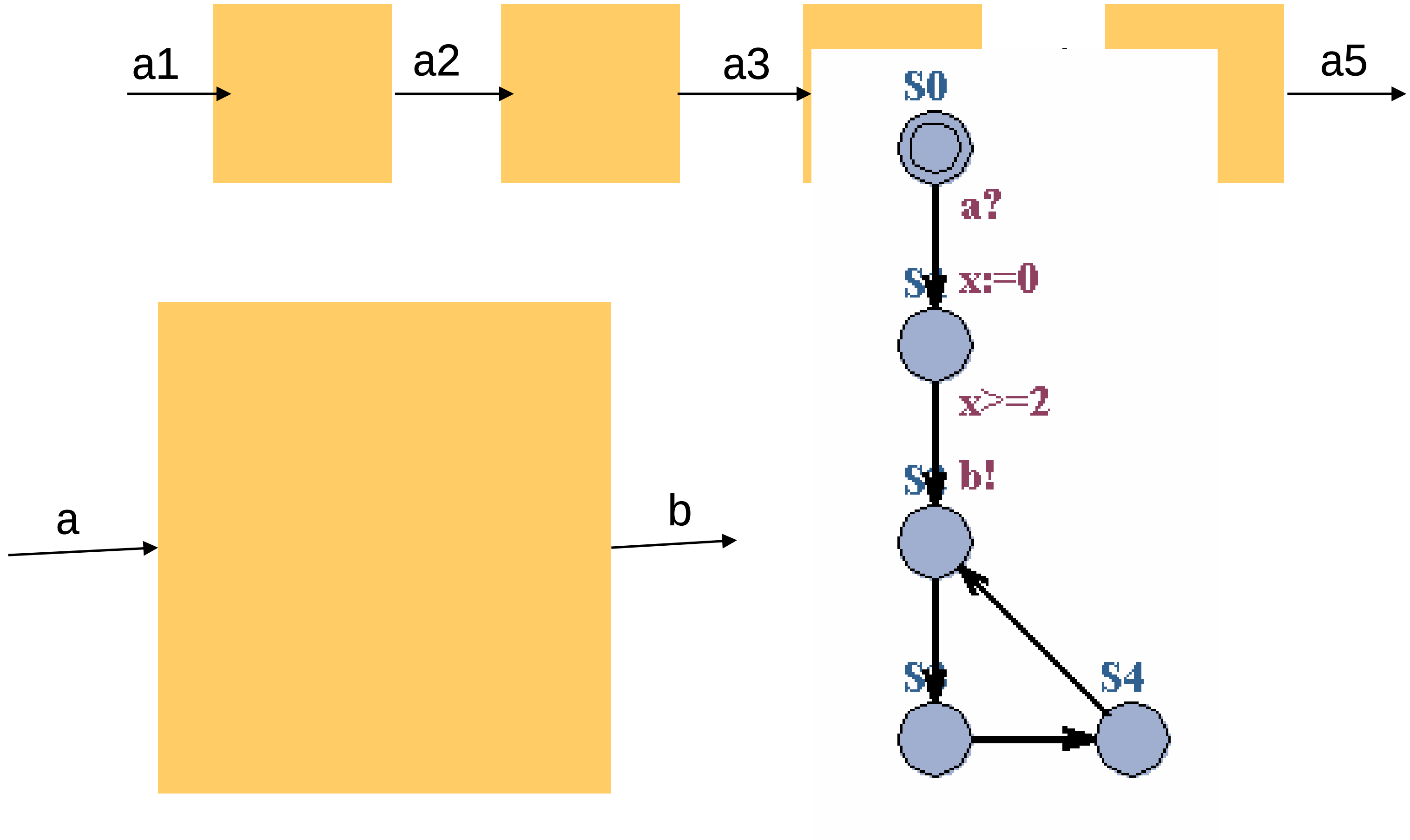
Compositionality



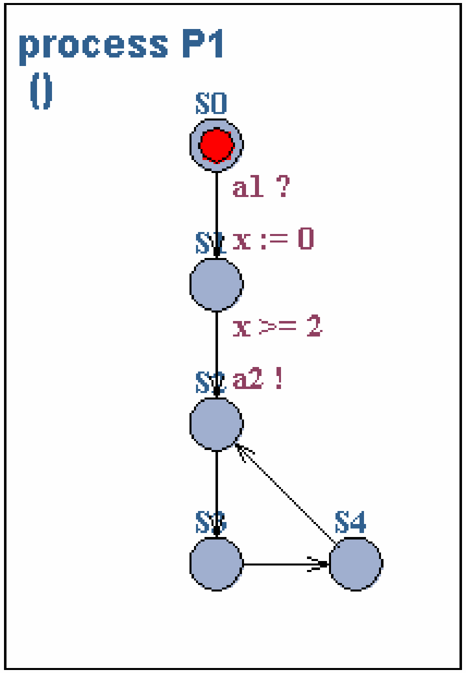
$$\frac{Sys_1 \leq Abs_1 \qquad Sys_2 \leq Abs_2}{Sys_1 \mid Sys_2 \leq Abs_1 \mid Abs_2}$$

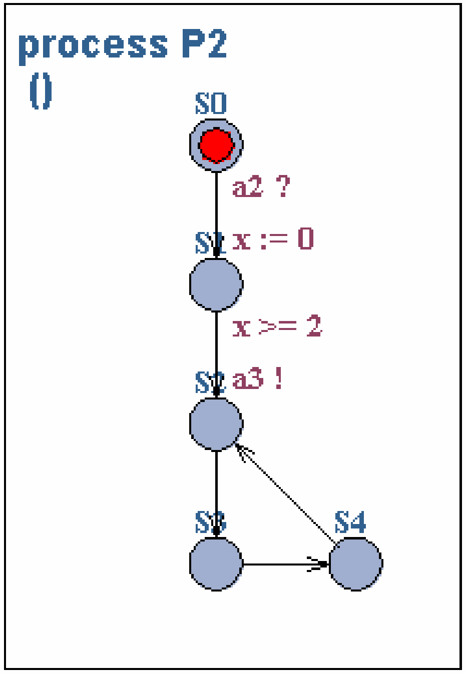
$$\frac{Sys_1 \leq Abs_1 \qquad Sys_2 \leq Abs_2 \qquad Abs_1 \mid Abs_2 \leq Abs}{Sys \leq Abs}$$

Abstraction *Example*

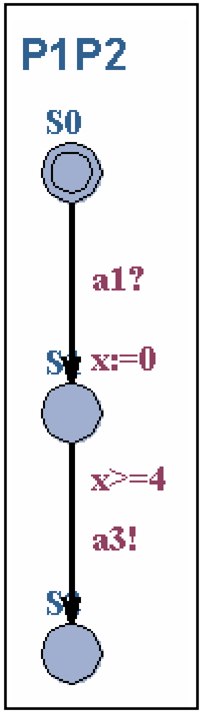


Example Continued





abstracted by



Proving abstractions

using reachability

process

()

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

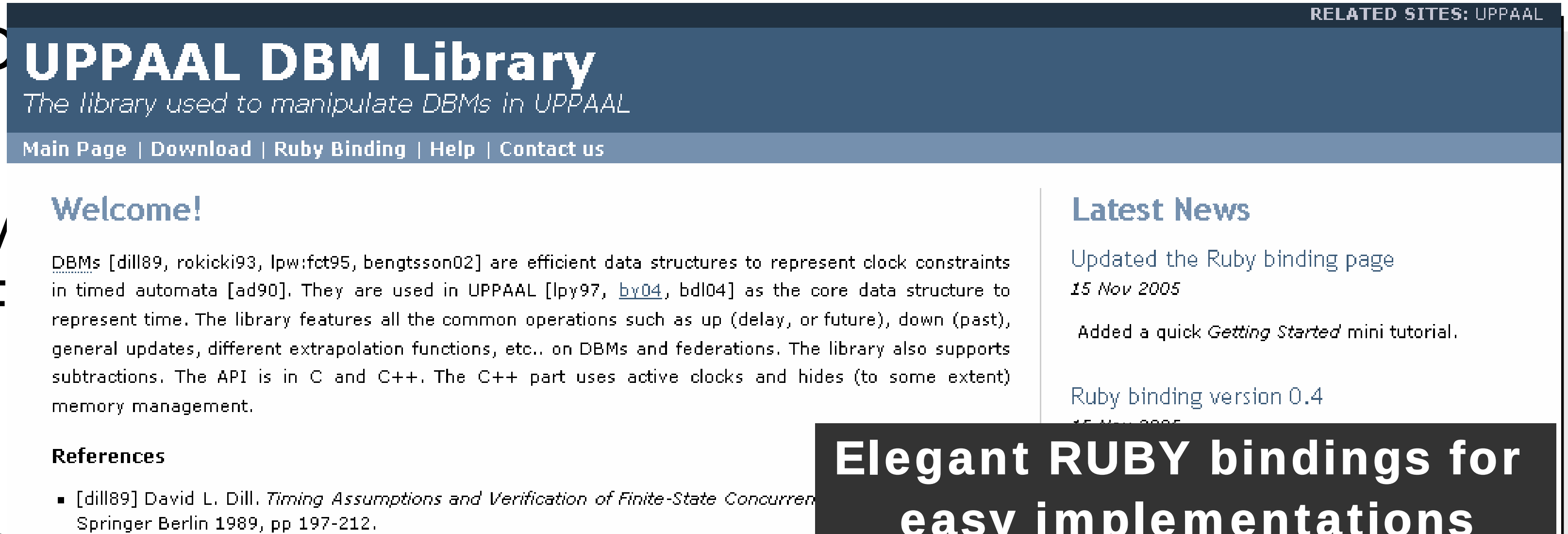
Henrik Ejersbo Jensen PhD Thesis 1999

BRICS
Basic Research in Computer Science

Further Optimizations

Datastructures for Zones

- D
- M
  F
- C
  D



RELATED SITES: UPPAAL

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 constraints in timed automata [ad90]. They are used in UPPAAL [lpy97, by04, bdl04] as the core data structure to 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 library also supports subtractions. The API is in C and C++. The C++ part uses active clocks and hides (to some extent) memory management.

References

- [dill89] David L. Dill. *Timing Assumptions and Verification of Finite-State Concurren* Springer Berlin 1989, pp 197-212.
- [rokicki93] Tomas Gerhard Rokicki. *Representing an* University 1993.
- [lpw:fct95] Kim G. Larsen, Paul Pettersson, and W Fundamentals of Computation Theory 1995, LNCS 965
- [bengtsson02] Johan Bengtsson. *Clocks, DBM, and* University 2002.
- [ad90] Rajeev Alur and David L. Dill. *Automata* Colloquium on Algorithms, Languages, and Programm
- [lpy97] Kim G. Larsen, Paul Pettersson, and Wang Software Tools for Technology Transfer , October 199
- [by04] Johan Bengtsson and Wang Yi. *Timed Automa* and Petri Nets 2004, LNCS 3098.
- [bdl04] Gerd Behrmann, Alexandre David, and Kim

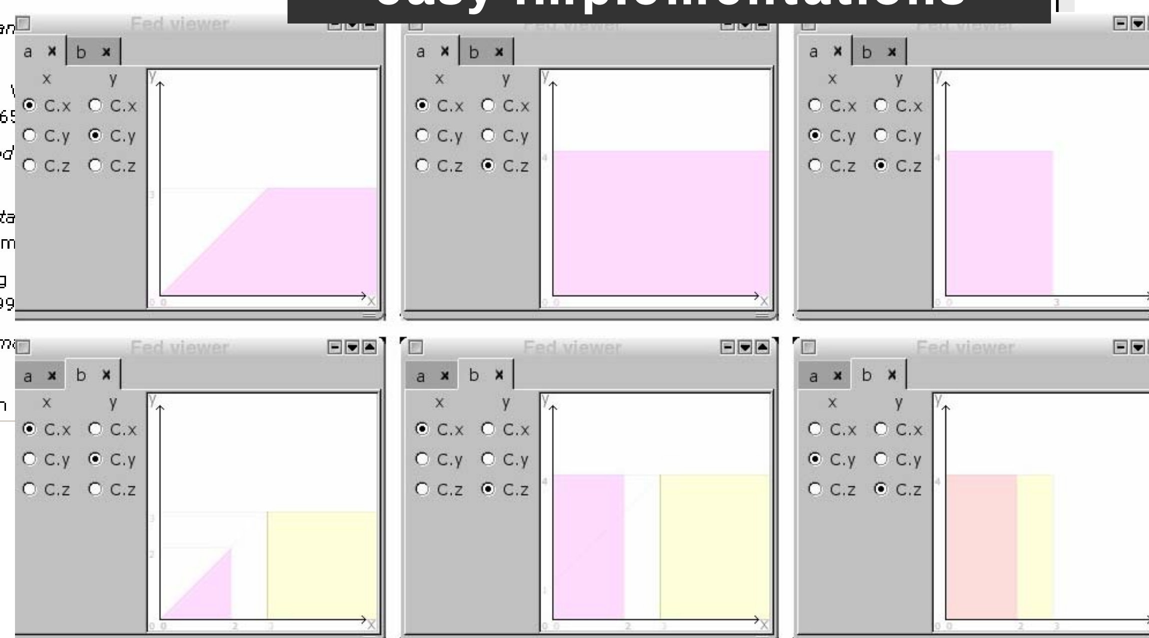
Latest News

Updated the Ruby binding page
15 Nov 2005

Added a quick *Getting Started* mini tutorial.

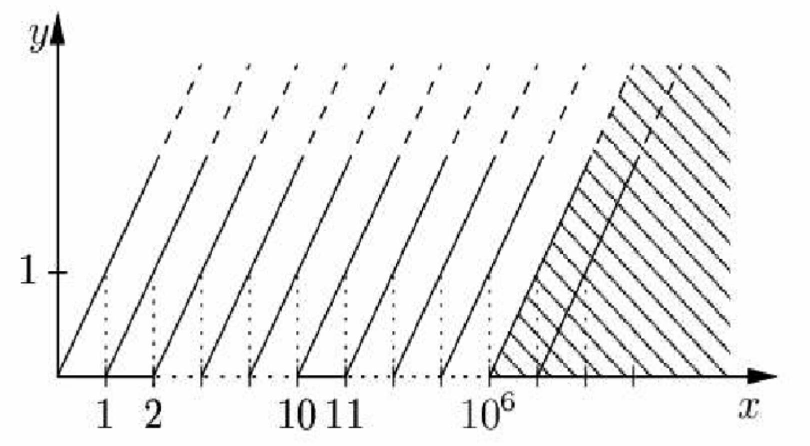
Ruby binding version 0.4

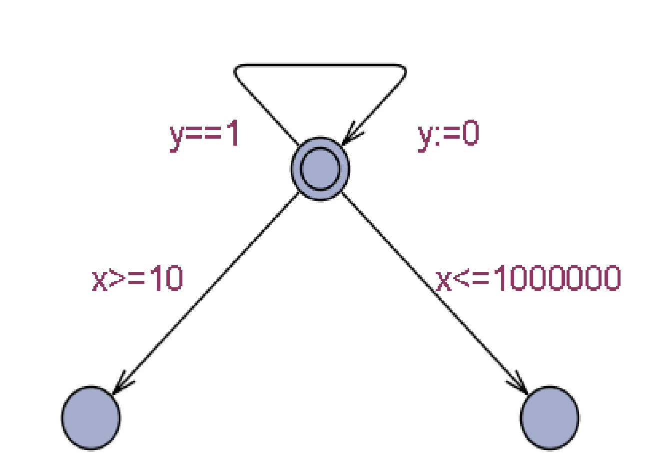
Elegant RUBY bindings for easy implementations

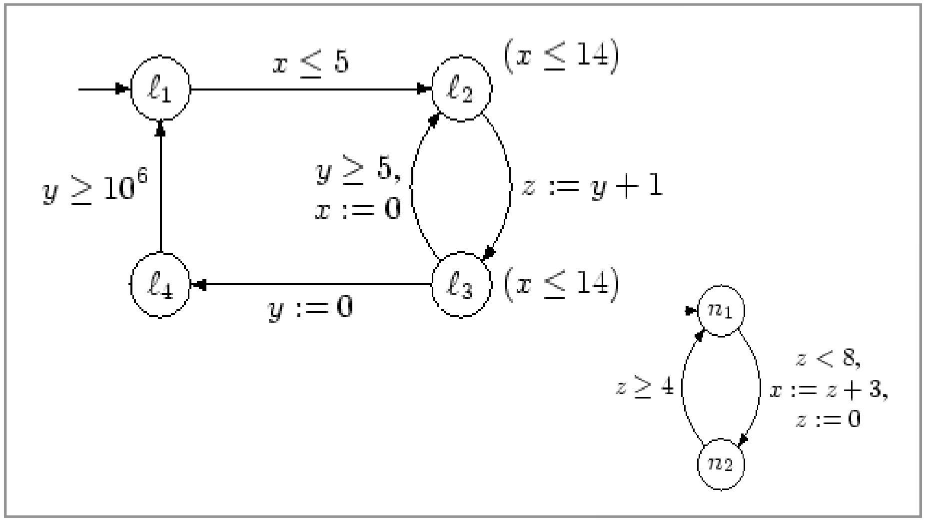


Zone Abstractions

[TACAS03,TACAS04]





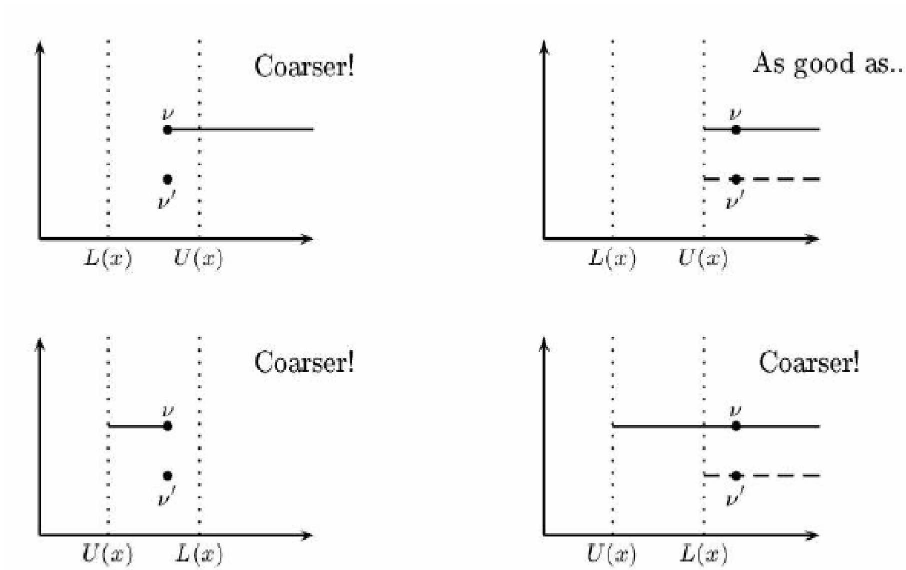


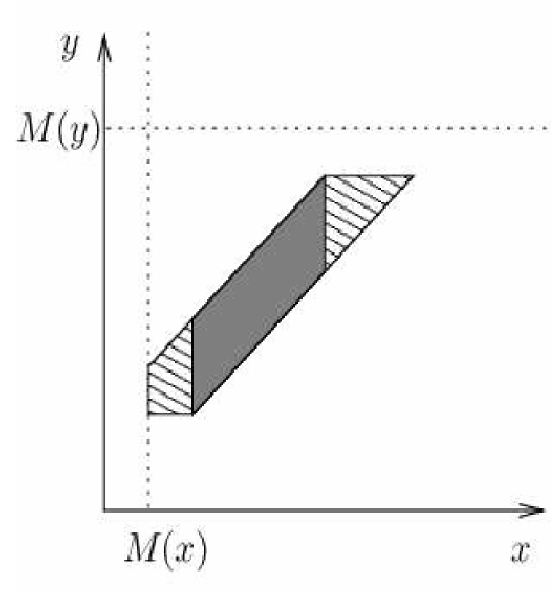
- Abstraction taking maximum constant into account necessary for termination
- Utilization of distinction between lower and upper bounds
- Utilization of location-dependency

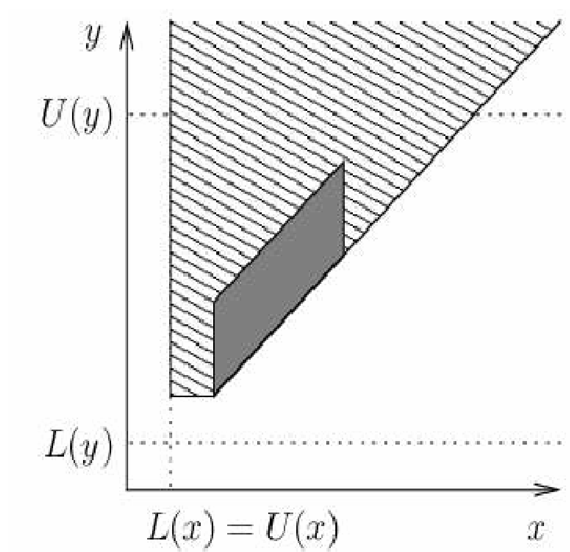
BRICS
Basic Research
in Computer Science

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

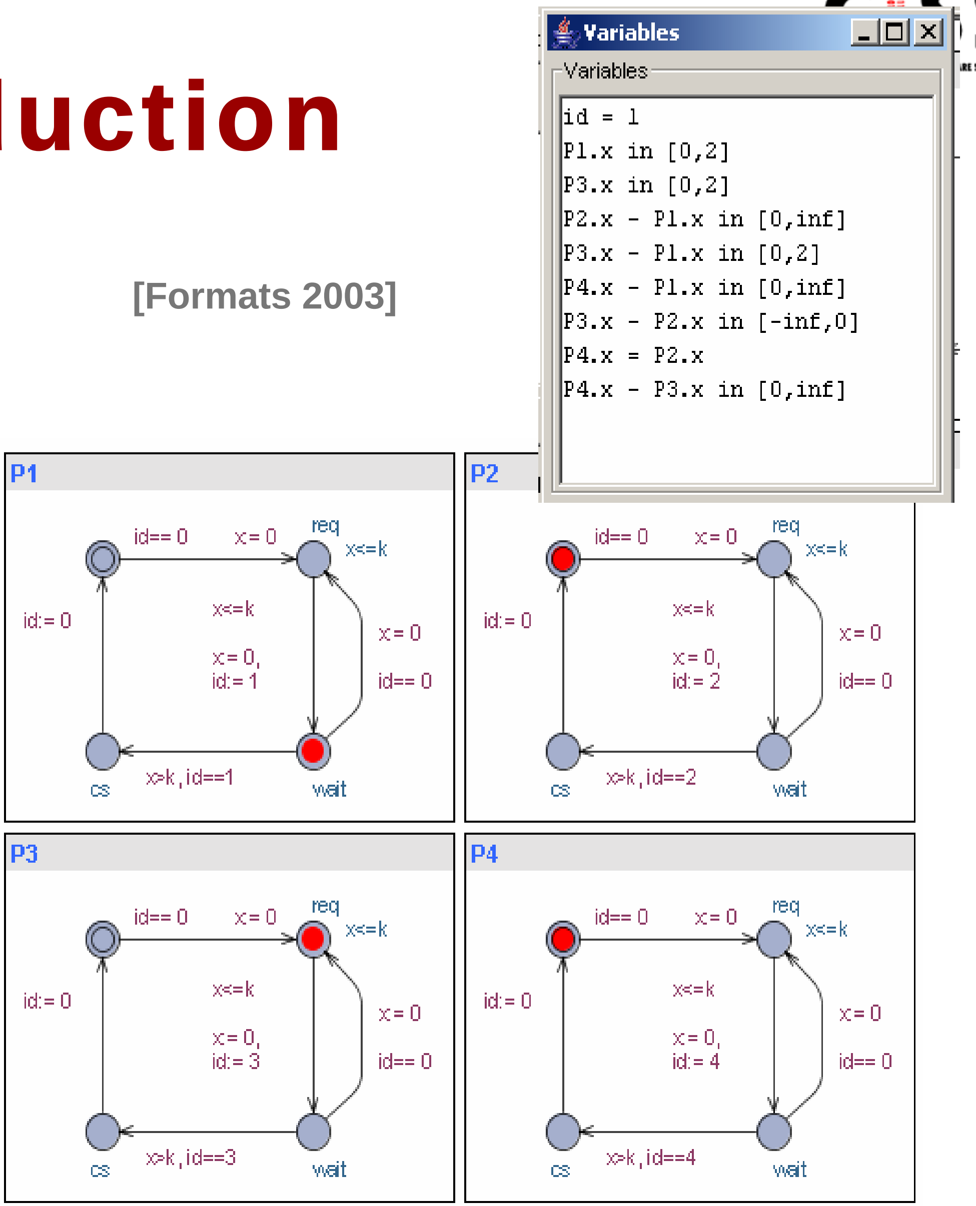
Zone abstractions

	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]



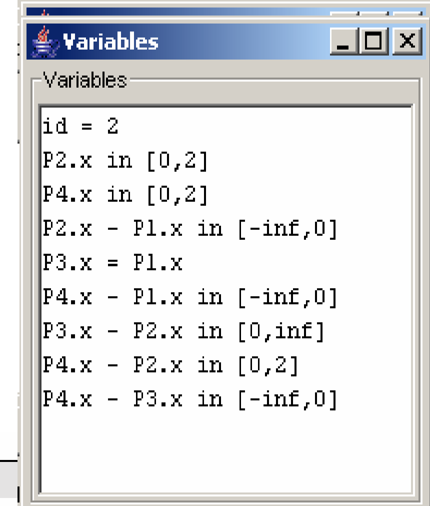
BRICS
Basic Research
in Computer Science

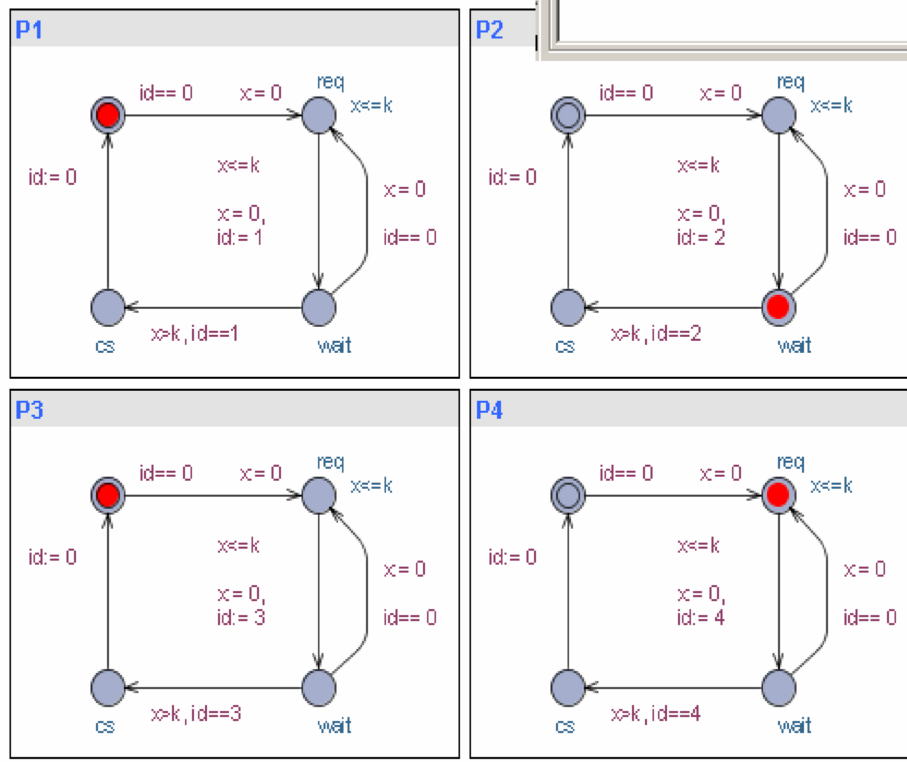
Symmetry Reduction

- 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

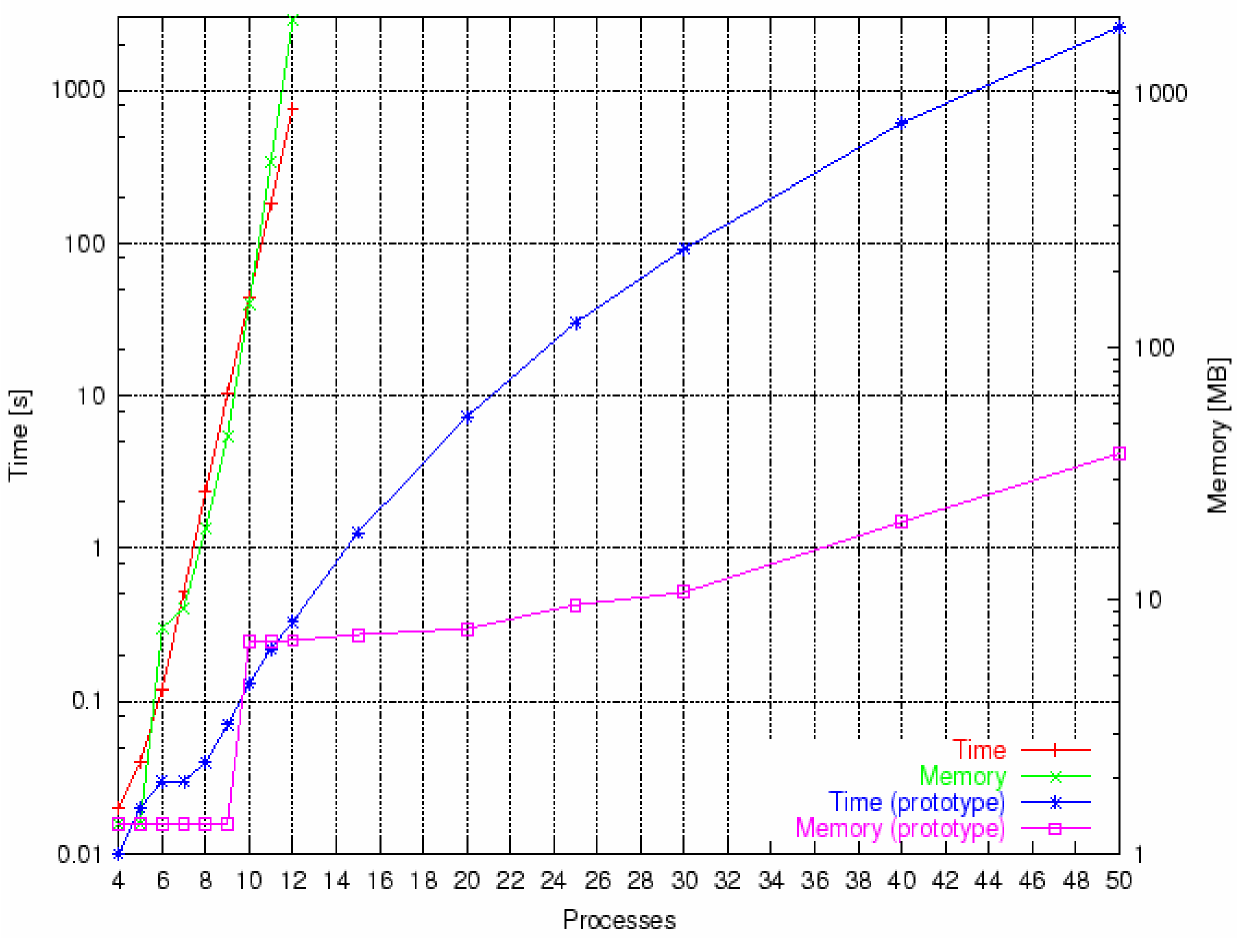




BRICS
Basic Research
in Computer Science

Symmetry Reduction

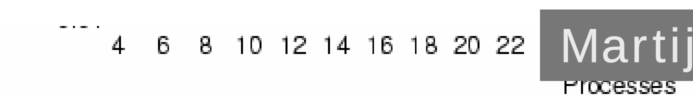
[Formats 2003]



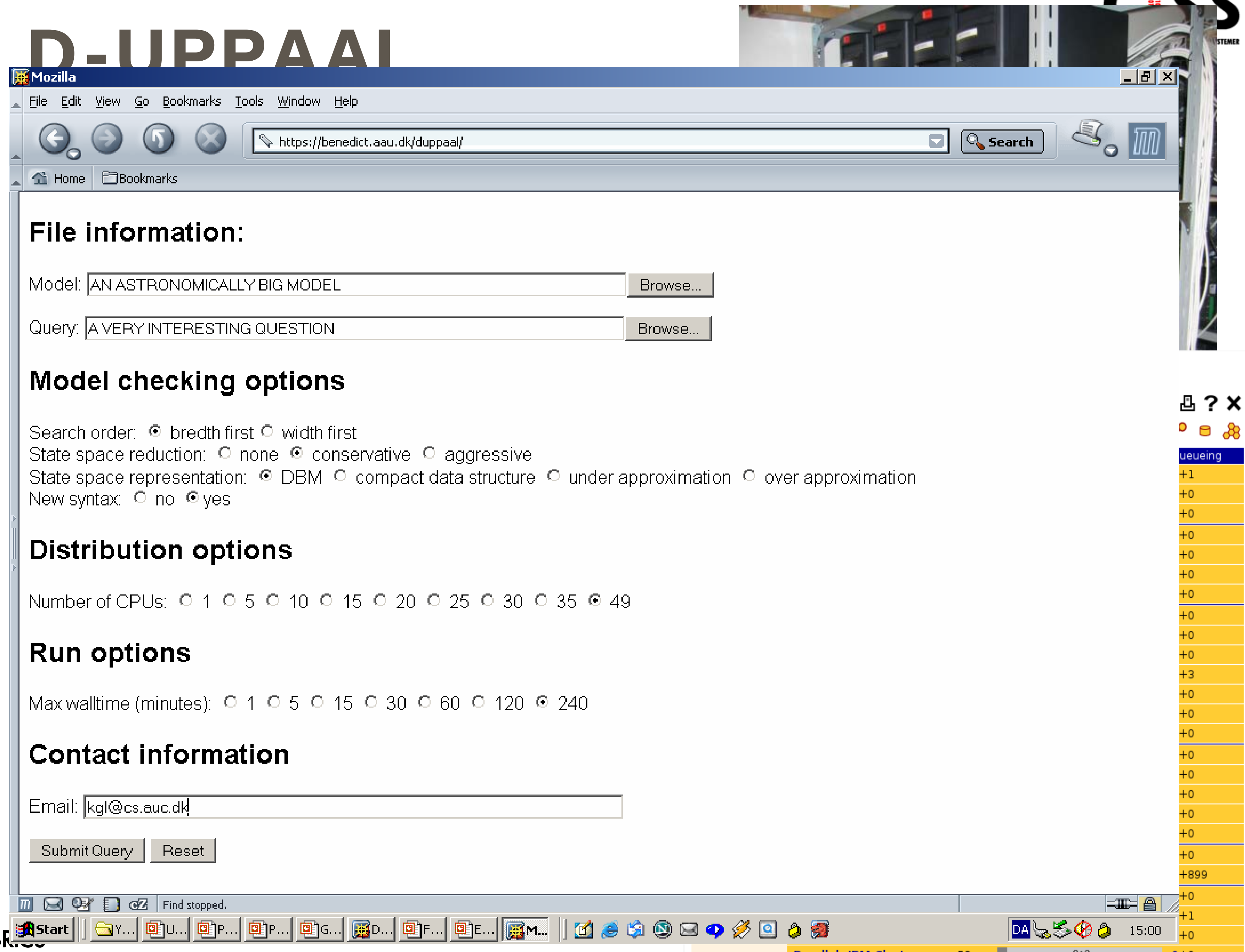
Symmetry Reduction

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



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
Parallab IBM Cluster