# UPPAAL tutorial

- What's inside UPPAAL
- The UPPAAL input languages

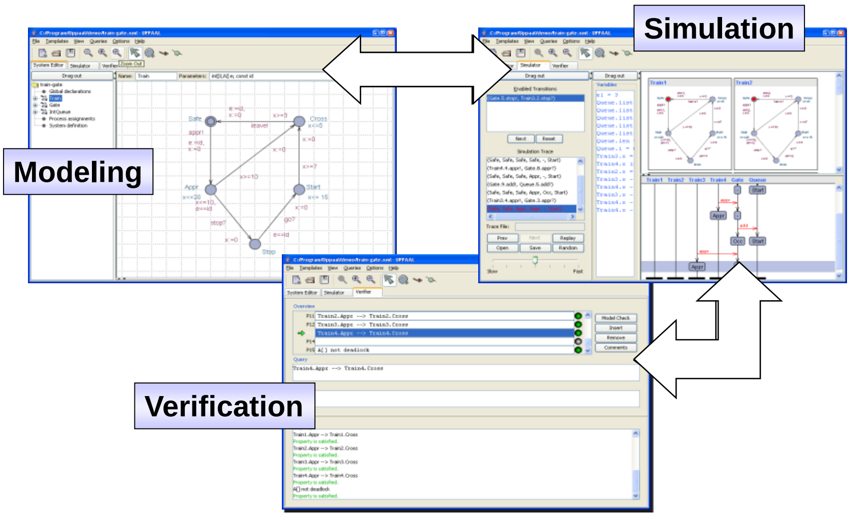
1

# UPPAAL tool

- Developed jointly by Uppsala & Aalborg University

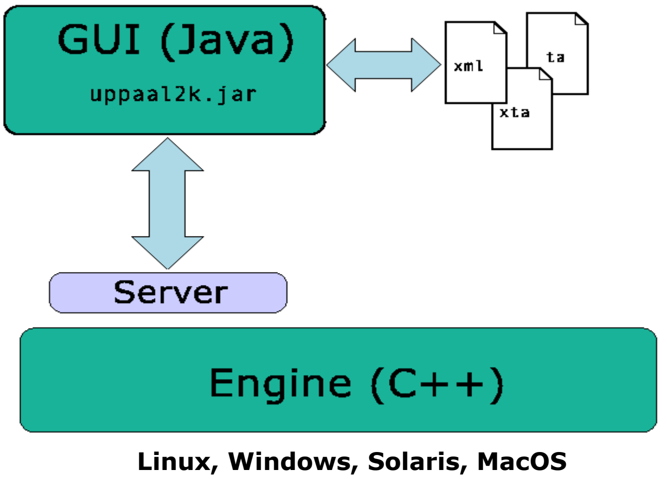
2

# UPPAAL Tool






# Architecture of UPPAAL




**Linux, Windows, Solaris, MacOS**

## What's inside UPPAAL



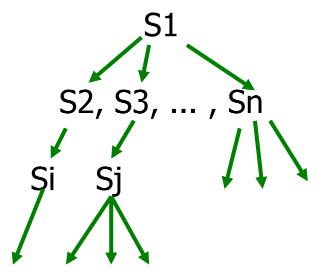
# OUTLINE

- Data Structures
  - DBM's (Difference Bounds Matrices)
  - Canonical and Minimal Constraints
- Algorithms
  - Reachability analysis
  - Liveness checking
- Verification Options

# All Operations on Zones

(needed for verification)

- Transformation
  - Conjunction
  - Post condition (delay)
  - Reset
- Consistency Checking
  - Inclusion
  - Emptiness



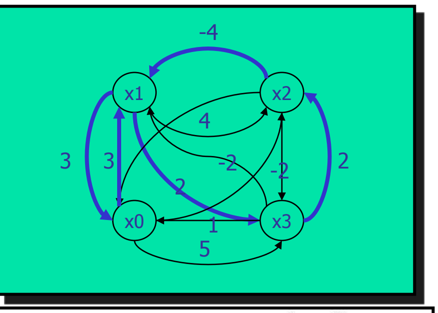


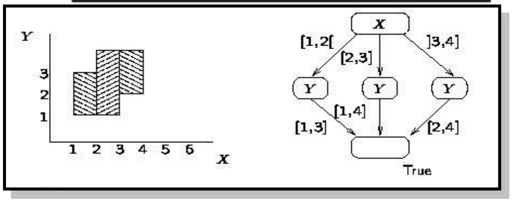

# Zones = Conjuctive constraints

- A zone Z is a conjunctive formula:
  $g_1$ & $g_2$ & … & $g_n$
  where $g_i$ may be $x_i \sim b_i$ or $x_i - x_j \sim b_{ij}$
- Use a zero-clock $x_0$ (constant 0), we have
  $\{x_i - x_j \sim b_{ij} \mid \sim \text{ is } < \text{ or } \leq, i,j \leq n\}$
- This can be represented as a MATRIX, DBM
  (Difference Bound Matrices)

# Datastructures for Zones in UPPAAL

- Difference Bounded Matrices
  [Bellman58, Dill89]
- Minimal Constraint Form
  [RTSS97]
- Clock Difference Diagrams
  [CAV99]






# Canonical Datastructures for Zones

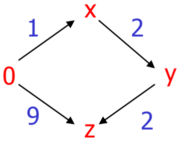
*Difference Bounded Matrices* Bellman 1958, Dill 1989

## Inclusion

**Z1**

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

**Graph**

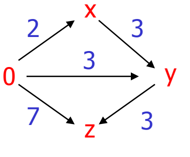



**? ⊆ ?**

**Z2**

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

**Graph**



## Canonical Dastructures for Zones

Bellman 1958, Dill 1989

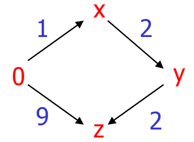
*Difference Bounded Matrices*

### Inclusion

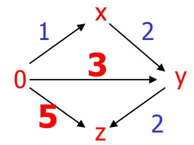
**Z1**

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

**Graph**




Shortest Path Closure



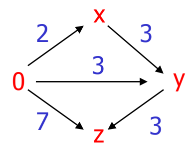

**? ⊆ ?**

**Z1 ⊆ Z2 !**

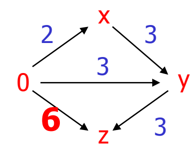
**Z2**

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

**Graph**




Shortest Path Closure






## Canonical Datastructures for Zones

*Difference Bounded Matrices*

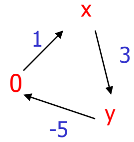
Bellman 1958, Dill 1989

### Emptiness

**Z**

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

**Graph**



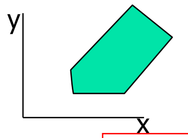

**Negative Cycle**
**iff**
**empty solution set**

# Canonical Datastructures for Zones

*Difference Bounded Matrices*

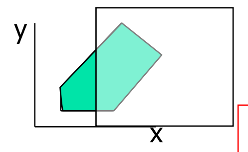
**Conjunction**

**Z**

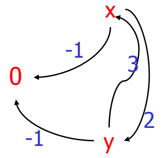


1<=x, 1<=y
-2<=x-y<=3

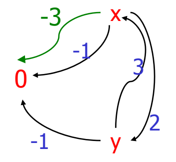
**Z∧g**

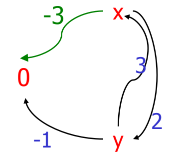


1<=x, 1<=y
-2<=x-y<=3
3<=x




Add new edge for g



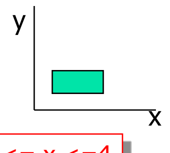

13

# Canonical Dastructures for Zones

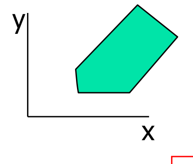
Difference Bounded Matrices

**Delay**

**Z**

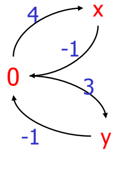


1<= x <=4
1<= y <=3

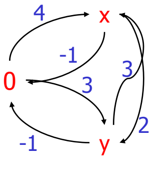


**Z↑**

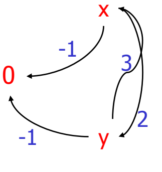
1<=x, 1<=y
-2<=x-y<=3




Shortest Path Closure



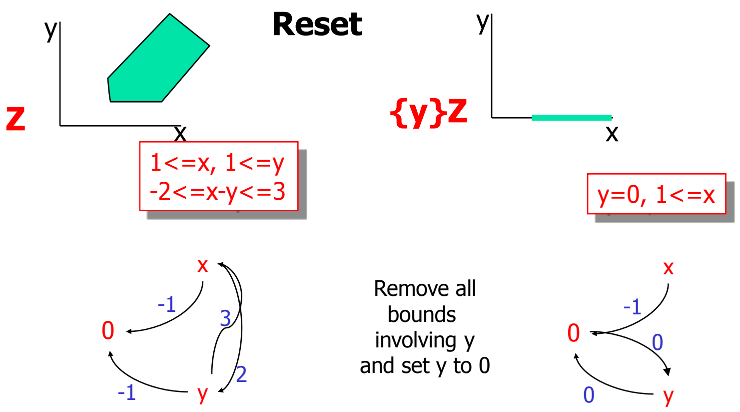

Remove upper bounds on clocks




14

## Canonical Datastructures for Zones

*Difference Bounded Matrices*




15

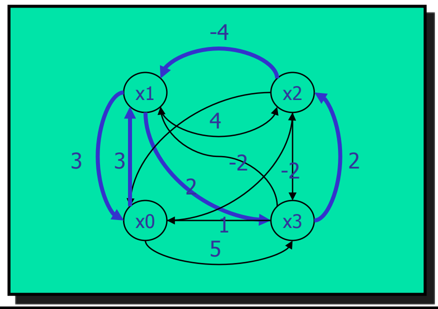
# COMPLEXITY

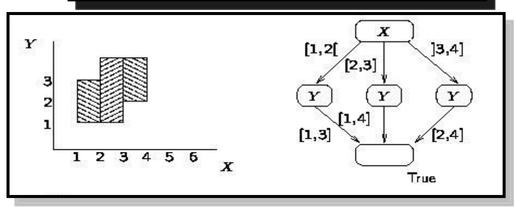
- Computing the shortest path closure, the cannonical form of a zone: $O(n^3)$ [Dijkstra's alg.]
- Run-time complexity, mostly in $O(n)$
  (when we keep all zones in cannonical form)

16

# Datastructures for Zones in UPPAAL

- Difference Bounded Matrices
  [Bellman58, Dill89]
- Minimal Constraint Form
  [RTSS97]
- Clock Difference Diagrams
  [CAV99]

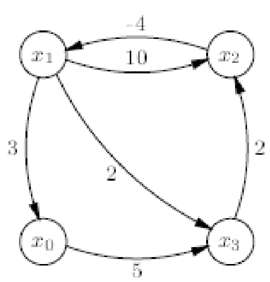


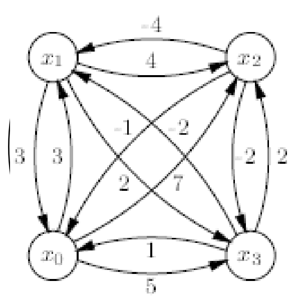



# Minimal Graph

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

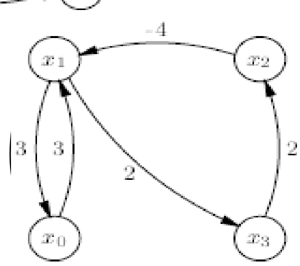



**Shortest Path Closure O(n³)**




(DBM)

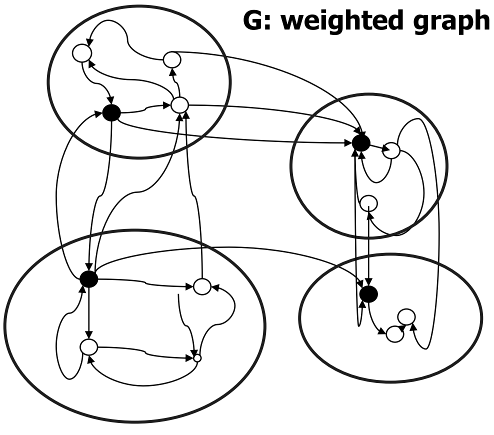
**Shortest Path Reduction O(n³)**




**Space** worst O(n²)
practice O(n)

(Minimal graph, a.ka. compact data structure)

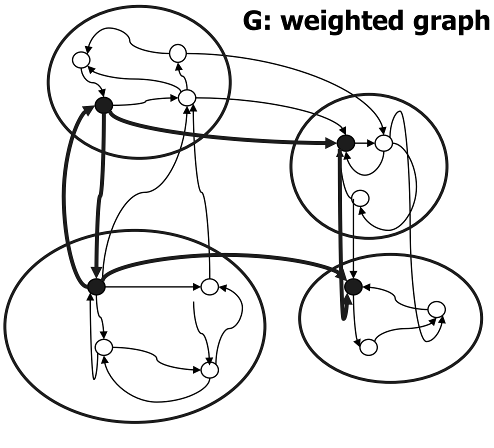
# Graph Reduction Algorithm




1. Equivalence classes based on 0-cycles.



# Graph Reduction Algorithm

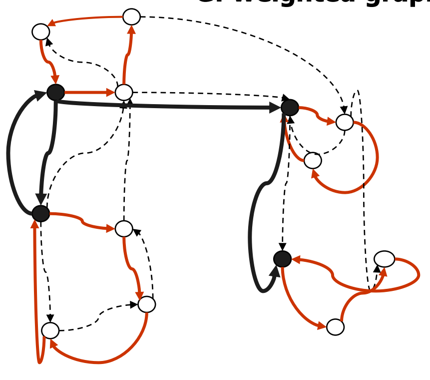



1. Equivalence classes based on 0-cycles.

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

# Graph Reduction Algorithm

**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

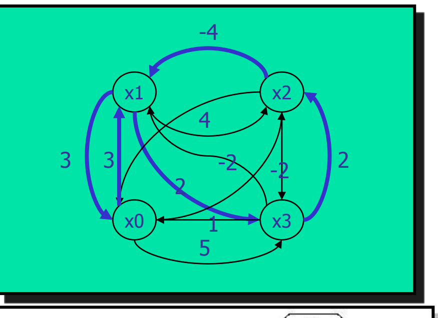


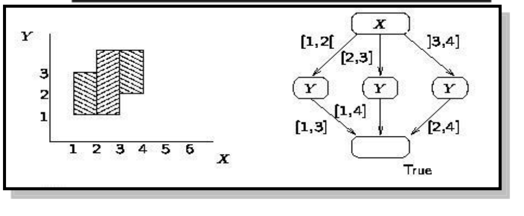
# Datastructures for Zones in UPPAAL

- Difference Bounded Matrices
  [Bellman58, Dill89]

- Minimal Constraint Form
  [RTSS97]

- Clock Difference Diagrams
  [CAV99]

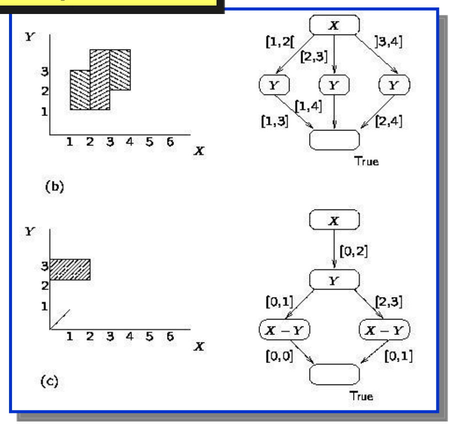


# Other Symbolic Datastructures

- NDD's Maler et. al.
- CDD's UPPAAL/CAV99
- DDD's Møller, Lichtenberg
- Polyhedra HyTech
- ......

**CDD-representations**






# Inside the UPPAAL tool

- Data Structures
  - DBM's (Difference Bounds Matrices)
  - Canonical and Minimal Constraints
- Algorithms
  - Reachability analysis
  - Liveness checking
- Verification Options

# Timed CTL in UPPAAL

**EF p | AG p | EG p | AF p | p - -> q**

**P ::= A.l | $g_c$ | $g_d$ | not p | p or p | p and p | p imply p**

*Process Location (a location in automaton A)*

*Clock constraint*

*predicate over data variables*

**p leads to q**
*denotes*
**AG (p imply AF q)**

25

# Timed CTL in UPPAAL

**EF p | AG p | EG p | AF p | p - -> q**

**P ::= A.l | $g_c$ | $g_d$ | not p | p or p | p and p | p imply p**

*Process Location (a location in automaton A)*

*Clock constraint*

*predicate over data variables*

**p leads to q**
*denotes*
**AG (p imply AF q)**

**SAFETY PROPERTIES**

26

# SAFETY Properties

in UPPAAL

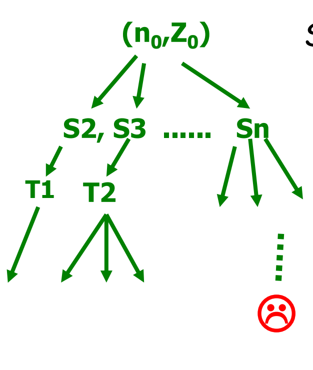
**F ::= EF P | AG P**

Reachability

Invariant = ¬ EF ¬ P
Thus, **AG P** is also checked by reachability analysis!

27

# We have a search problem




Symbolic state

Symbolic transitions

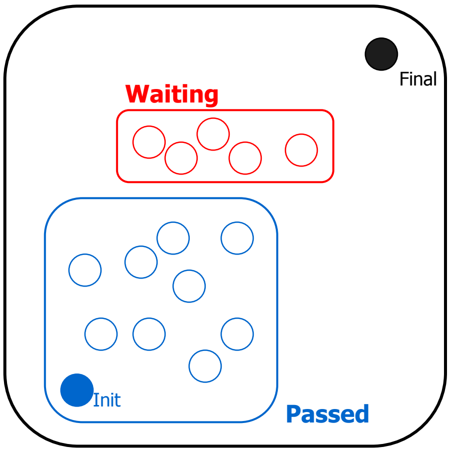
Reachable?

EF ☹

28

# Forward Reachability

**Init -> Final ?**




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

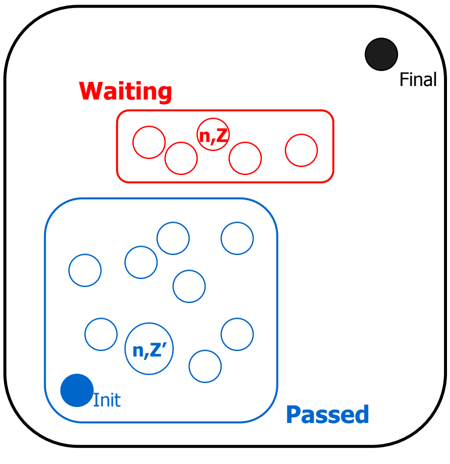
**REPEAT**
- pick (n,Z) in **Waiting**
- **if** for some Z′ ⊇ Z
  (n,Z′) in **Passed** **then** **STOP**
- **else** /explore/ add
  { (m,U) : (n,Z) => (m,U) }
  to **Waiting**;
  Add (n,Z) to **Passed**

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



# Forward Reachability

**Init -> Final ?**



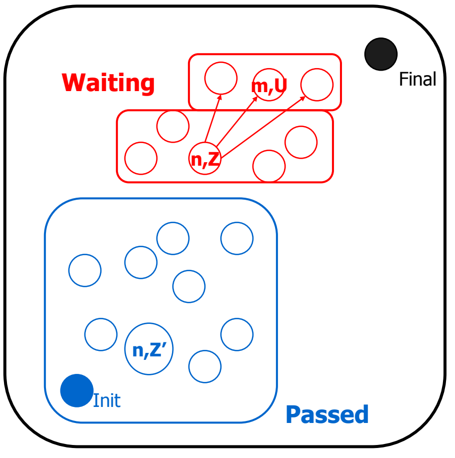

**INITIAL** **Passed** := Ø;
**Waiting** := {(n0,Z0)}

**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**

# Forward Reachability

**Init -> Final ?**




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

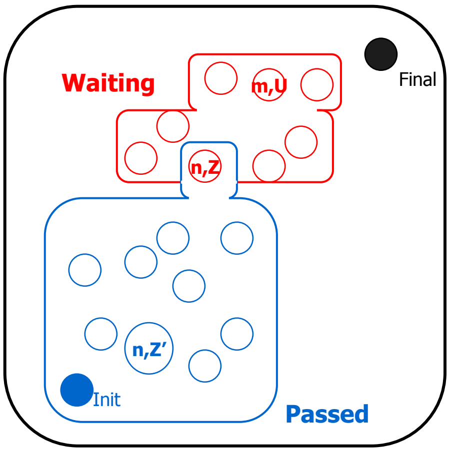
**REPEAT**
- pick (n,Z) in **Waiting**
- **if** for some Z' ⊇ Z (n,Z') in **Passed** **then STOP**
- **else** /explore/ add { (m,U) : (n,Z) => (m,U) } to **Waiting**;
  Add (n,Z) to **Passed**

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

31

# Forward Reachability

**Init -> Final ?**




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

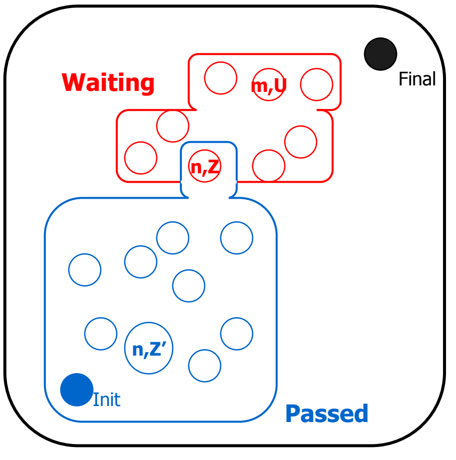
**REPEAT**
- pick (n,Z) in **Waiting**
- **if** for some Z' ⊇ Z (n,Z') in **Passed** **then STOP**
- **else** /explore/ add { (m,U) : (n,Z) => (m,U) } to **Waiting**;
  Add (n,Z) to **Passed**

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

32

# Forward Reachability

**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**



# Further question

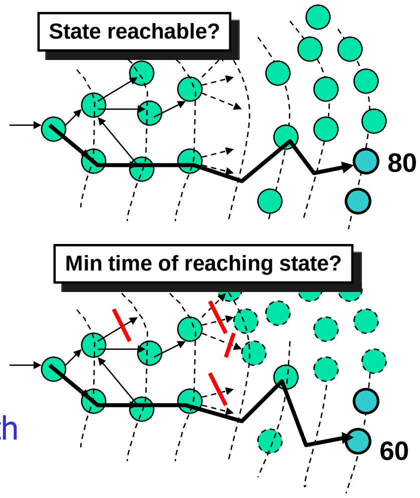
Can we find the path with shortest delay, leading to P ?
(i.e. a state satisfying P)

## OBSERVATION:

Many scheduling problems can be phrased naturally as reachability problems for timed automata.

# Verification vs. Optimization

- Verification Algorithms:
  - Checks a logical property of the entire state-space of a model.
  - Efficient Blind search.
- Optimization Algorithms:
  - Finds (near) optimal solutions.
  - Uses techniques to avoid non-optimal parts of the state-space (e.g. Branch and Bound).
- Goal: solve opt. problems with verification.

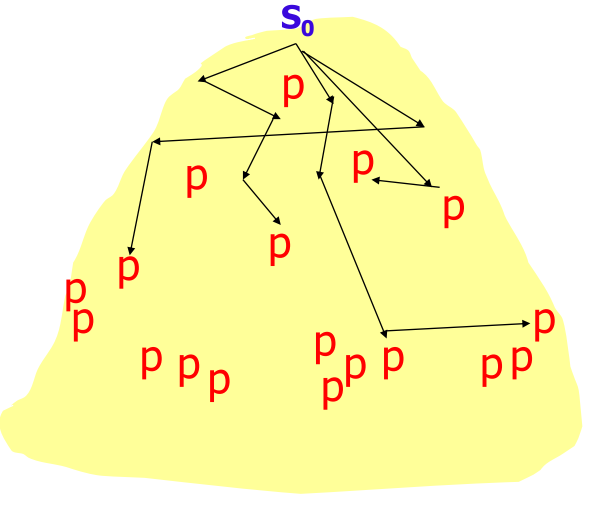





# OPTIMAL REACHABILITY

The maximal and minimal delay problem

Find the trace leading to P with min delay

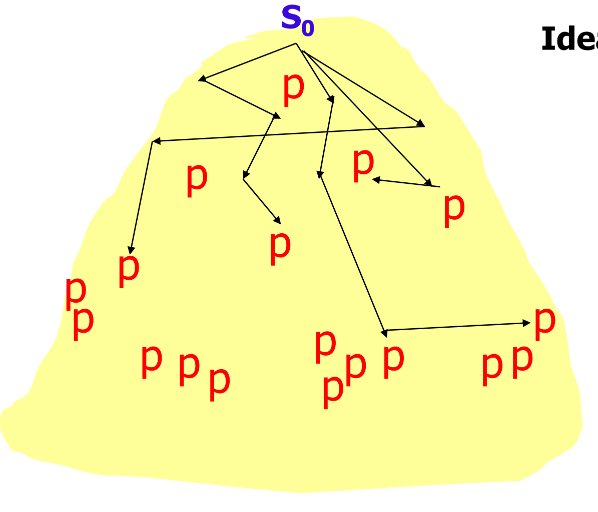



There may be a lot of pathes leading to P

Which one with the shortest delay?

37

Find the trace leading to P with min delay

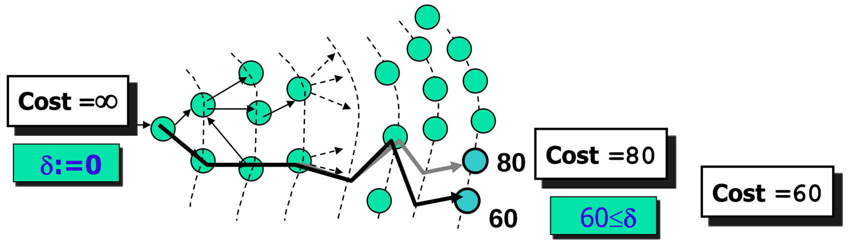



**Idea:** delay as "**Cost**" to reach a state, thus **cost** increases with time at rate 1

38

## An Simple Algorithm for minimal-cost reachability

- State-Space Exploration + Use of global variable **`Cost`** and global clock δ
- Update **`Cost`** whenever goal state with **min( C ) < `Cost`** is found:

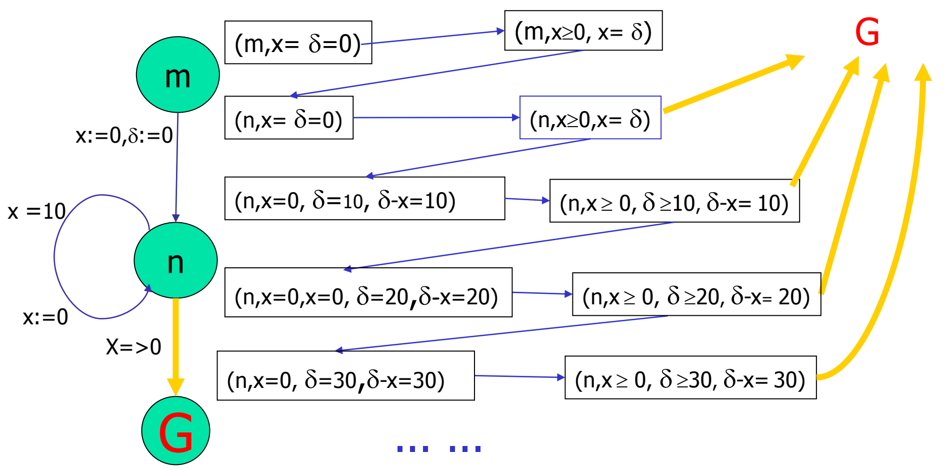



- Terminates when entire state-space is explored.

**Problem**: The search may never terminate!



## Example (min delay to reach G)




The minimal **delay** = 0 but the search may never terminate!

Problem: How to **symbolically** represent the zone **C.**

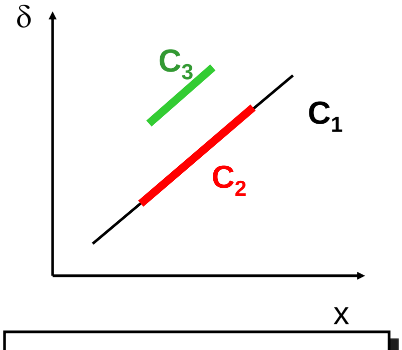
# Priced-Zone

- Cost = minimal total time
- **C** can be represented as the zone $Z^\delta$, where:
  - $Z^\delta$ original (ordinary) DBM plus…
  - $\delta$ clock keeping track of the cost/time.
- Delay, Reset, Conjunction etc. on Z are the standard DBM-operations
- Delay-Cost is incremented by Delay-operation on $Z^\delta$.

41

# Priced-Zone

- Cost = min total time
- **C** can be represented as the zone $Z^\delta$, where:
  - $Z^\delta$ is the original zone Z extended with the global clock $\delta$ keeping track of the cost/time.
  - Delay, Reset, Conjunction etc. on C are the standard DBM-operations
- But inclusion-checking will be different




Then: $C_3 \sqsubseteq C_2 \sqsubseteq C_1$

But: $C_3 \not\subseteq C_2 \subseteq C_1$

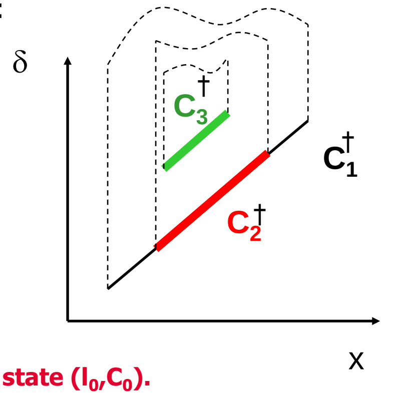
42

## Solution: $()^\dagger$-widening operation

- $()^\dagger$ removes upper bound on the $\delta$–clock:

  $C_3 \sqsubseteq C_2 \sqsubseteq C_1$

  $C_3^\dagger \subseteq C_2^\dagger \subseteq C_1^\dagger$

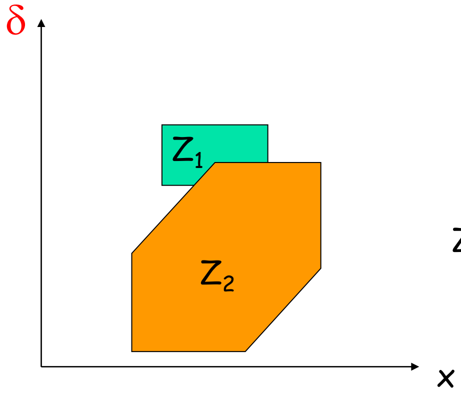



- In the Algorithm:
  - $Delay(C^\dagger) = ( Delay(C^\dagger) )^\dagger$
  - $Reset(x,C^\dagger) = ( Reset(x,C^\dagger) )^\dagger$
  - $C_1^\dagger \wedge g = ( C_1^\dagger \wedge g )^\dagger$

  - **It is suffices to apply $()^\dagger$ to the initial state $(l_0,C_0)$.**

43

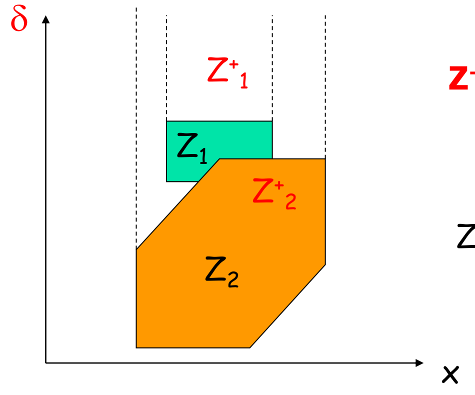
## Example (widening for Min)




$Z_1 \not\subseteq Z_2$

44

# Example (widening for Min)

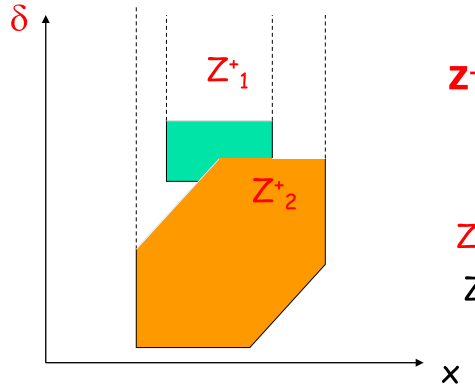



**$Z^+$= Widen(Z)**

$Z_1 \not\subseteq Z_2$



# Example (widening for Min)




**$Z^+$= Widen(Z)**

$Z^+_1 \subseteq Z^+_2$ !

$Z_1 \sqsubseteq Z_2$

# An Algorithm (Min)

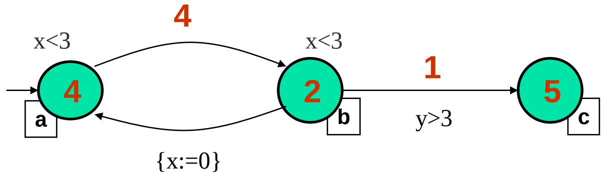
```
Cost:=∞, Pass := {}, Wait := {(l0,C0)}
while Wait ≠ {} do
   select (l,C) from Wait
   if (l,C) ⊨ P and Min(C)<Cost then Cost:= Min(C)
   if (l,C) ⊑ (l,C') for some (l,C') in Pass then skip
       otherwise add (l,C) to Pass
       and forall (m,C') such that (l,C) ⇝ (m,C'):
         add (m,C') to Wait
Return Cost
```

One-step reachability relation

**Output**: `Cost` = the min cost of a found trace satisfying `P`.



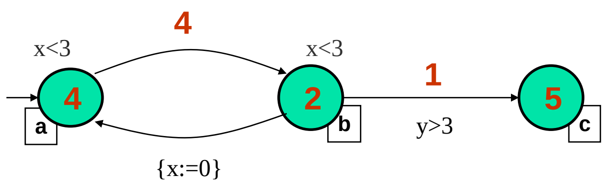
# Further reading: Priced Timed Automata [Larsen et al]




- Timed Automata + Costs on transitions and locations.
- Uniformly Priced = Same cost in all locations (edges may have different costs).

- Cost of performing transition: Transition cost.
- Cost of performing delay **d**: ( **d** x location cost ).

# Priced Timed Automata




**Trace:**

(**a**,x=y=0) $\xrightarrow[4]{}$ (**b**,x=y=0) $\xrightarrow[2.5 \times 2]{\varepsilon(2.5)}$ (**b**,x=y=2.5) $\xrightarrow[0]{}$ (**a**,x=0,y=2.5)

**Cost of Execution Trace:**

Sum of costs: **4 + 5 + 0 = 9**

**Problem:** Finding the minimum cost of reaching **c** !



# Inside the UPPAAL tool

- Data Structures
  - DBM's (Difference Bounds Matrices)
  - Canonical and Minimal Constraints
- Algorithms
  - Reachability analysis
  - Liveness checking
- Verification Options

# Timed CTL in UPPAAL

**EF p | AG p | EG p | AF p | p - -> q**

**P ::= A.l | $g_c$ | $g_d$ | not p | p or p | p and p | p imply p**

*Process Location (a location in automaton A)*

*Clock constraint*

*predicate over data variables*

**LIVENESS PROPERTIES**

**SAFETY PROPERTIES**

**p leads to q** *denotes* **AG (p imply AF q)**

51

# LIVENESS Properties

**in UPPAAL**

**F ::= EG p | AF p | p - -> q**

**Possibly always** P is equivalent to (: AF : P)

**Eventually** P is equivalent to (: EG : P)

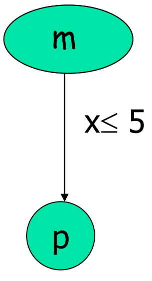
P **leads to** Q is equivalent to AG ( P imply AF Q)

52

# Question

AF P

*"P will be true for sure in future"*



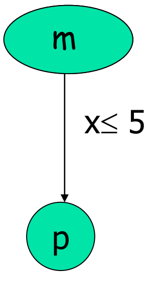

**??** Does this automaton satisfy AF P



# Note that

AF P

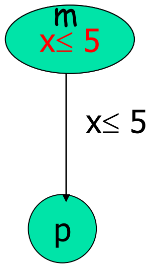
*"P will be true for sure in future"*




**NO !!!!** there is a path:
(m, x=0) →(m,x=1)→(m,2) **...** (m,x=k) **...**
Idling forever in location m

# Note that

AF P "P *will be true for sure in future*"




This automaton satisfies AF P

55

# Algorithm for checking AF P

**Eventually** P

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

56

# Question: Time bound synthesis

AF P    "P will be true eventually"
But no time bound is given.

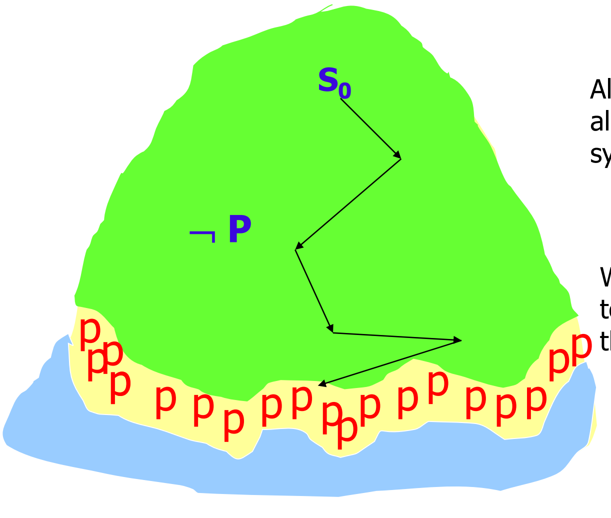
Assume AF P is satisfied by an automaton A.
Can we calculate the Max time bound?

OBS: we know how to calculate the Min !

57

Assume AF P is satisfied

**Find the trace leading to P with the max delay**




Almost the same algorithm as for synthesizing Min

We need to explore the Green part

58

# An Algorithm (Max)

```
Cost:=0, Pass := {}, Wait := {(l0,C0)}
while Wait ≠ {} do
   select (l,C) from Wait
   if (l,C) ⊨ P and Max(C)>Cost then Cost:= Max(C)
   else if forall (l,C') in Pass: C ⋢ C' then
      add (l,C) to Pass
      forall (m,C') such that (l,C) ⇝ (m,C'):
         add (m,C') to Wait
Return Cost
```

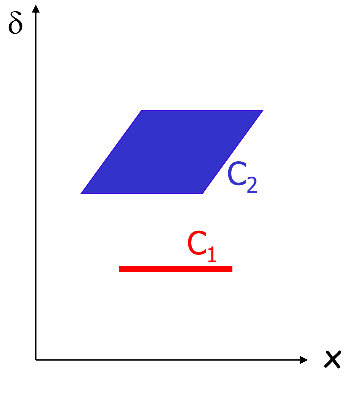
One-step reachability relation

**Output**: `Cost` = the max cost of a found trace satisfying `P`.

**BUT**: $\sqsubseteq$ is defined on zones where the lower bound of "cost" is removed

59

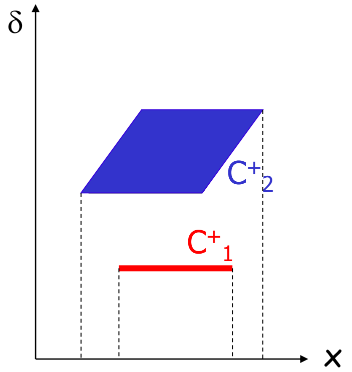
## Zone-Widening operation for Max




$C_1 \not\sqsubseteq C_2$

60

## Zone-Widening operation for Max




$C_1 \not\subseteq C_2$

$C^+_1 \subseteq C^+_2$

$C_1 \sqsubseteq C_2$ !

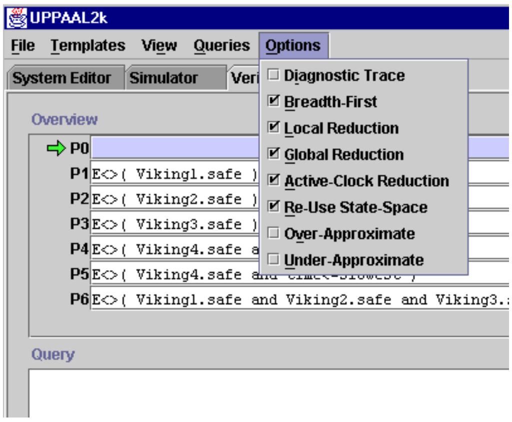


# Inside the UPPAAL tool

- Data Structures
  - DBM's (Difference Bounds Matrices)
  - Canonical and Minimal Constraints
- Algorithms
  - Reachability analysis
  - Liveness checking
  - Termination

➡ Verification Options

# Verification Options




- Diagnostic Trace

- Breadth-First
- Depth-First

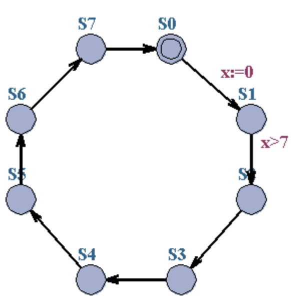
- Local Reduction
- Active-Clock Reduction
- Global Reduction

- Re-Use State-Space

- Over-Approximation
- Under-Approximation

63

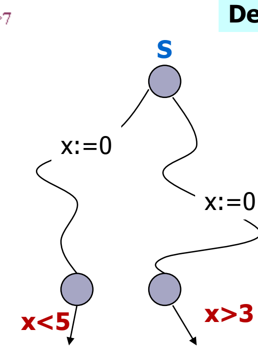
# Inactive (passive) Clock Reduction




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

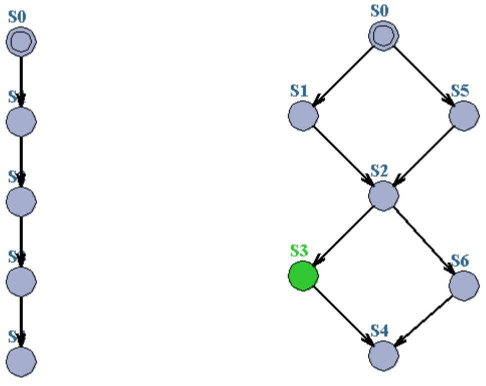
**Definition**

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




64

## Global Reduction
(When to store symbolic state)

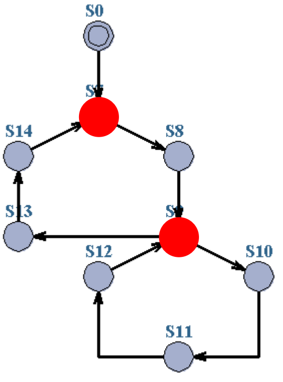



However,
**Passed** list useful for efficiency

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



## Global Reduction [RTSS97]
(When to store symbolic state)




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

[RTSS97,CAV03]

# To Store Or Not To Store?

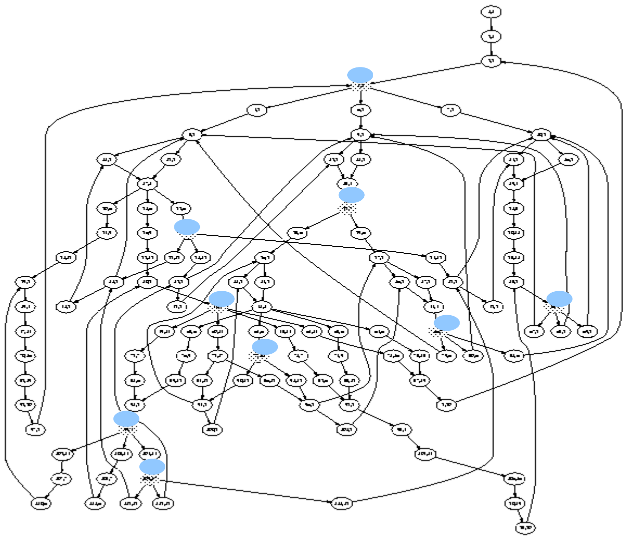
117 $states_{total}$

↓

81 $states_{entrypoint}$

↓

9 states

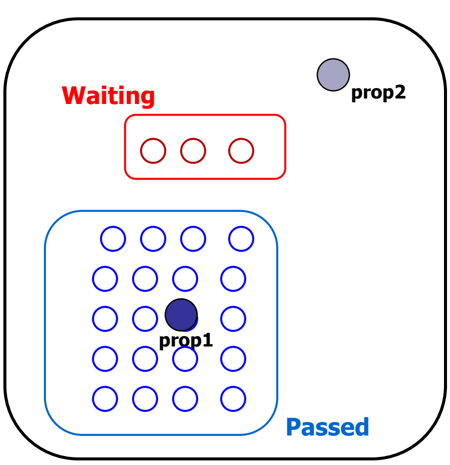


**Time OH less than 10%**

(need to re-explore some states)

7

# Reuse of State Space



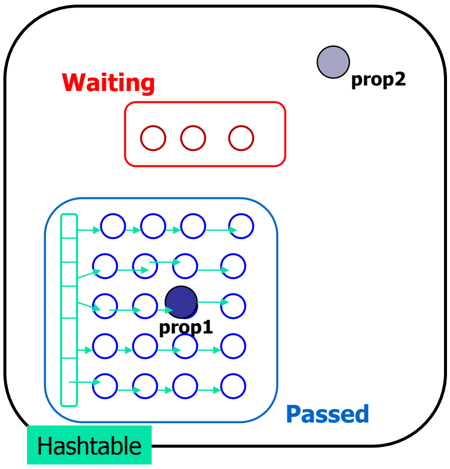

```
A[]  prop1

A[]  prop2
A[]  prop3
A[]  prop4
A[]  prop5
 .
 .
 .
A[]  propn
```

Search in existing **Passed** list before continuing search

Which order to search?

68

# Reuse of State Space



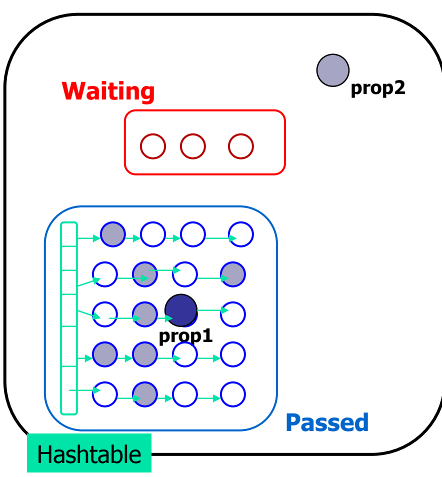

```
A[]  prop1

A[]  prop2
A[]  prop3
A[]  prop4
A[]  prop5
 .
 .
 .
A[]  propn
```

Search in existing **Passed** list before continuing search

Which order to search?



# Reuse of State Space



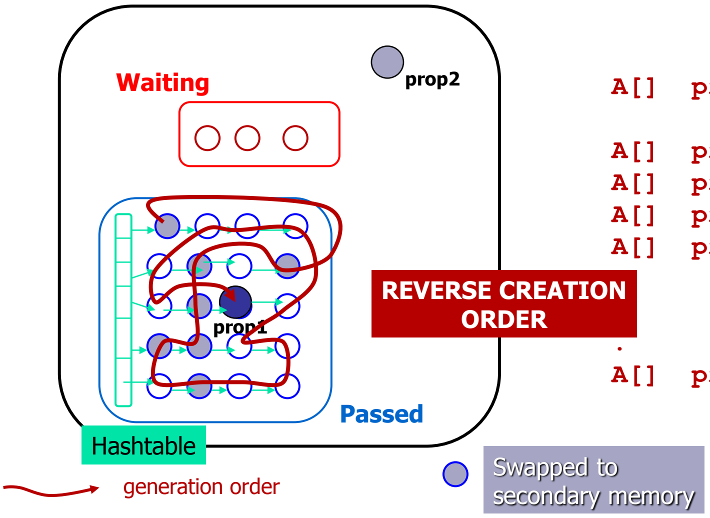

```
A[]  prop1

A[]  prop2
A[]  prop3
A[]  prop4
A[]  prop5
 .
 .
 .
A[]  propn
```

Search in existing **Passed** list before continuing search

Which order to search?

Swapped to secondary memory

# Reuse of State Space




```
A[]  prop1

A[]  prop2
A[]  prop3
A[]  prop4
A[]  prop5
.
A[]  propn
```

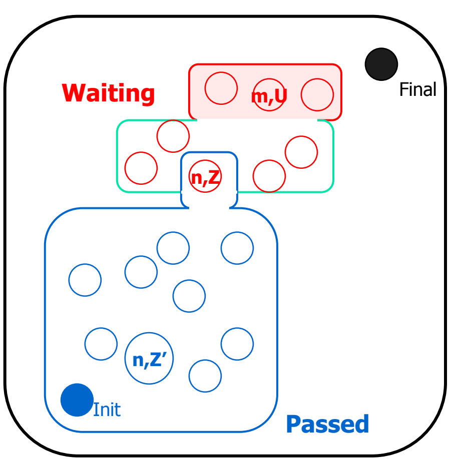
Search in existing **Passed** list before continuing search

Which order to search?



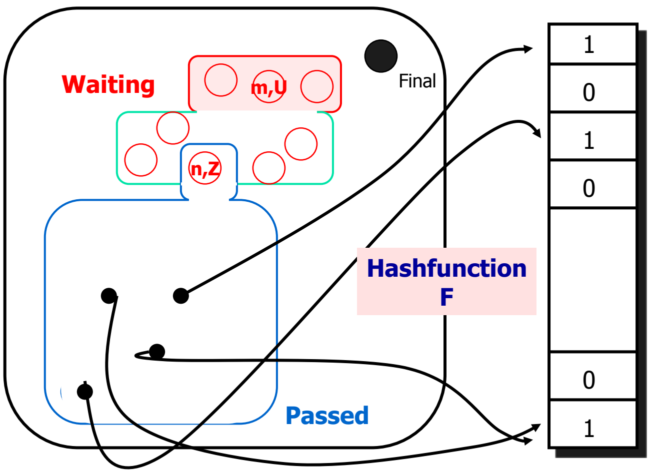
# Under-approximation

## *Bitstate Hashing* (Holzman,SPIN)



# Under-approximation

## *Bitstate Hashing*




**Passed**=
Bitarray

**UPPAAL**
8 Mbits



# Bit-state Hashing

```
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
```

**Passed**(**F**(n,Z)) = **1**

**Passed**(**F**(n,Z)) := **1**

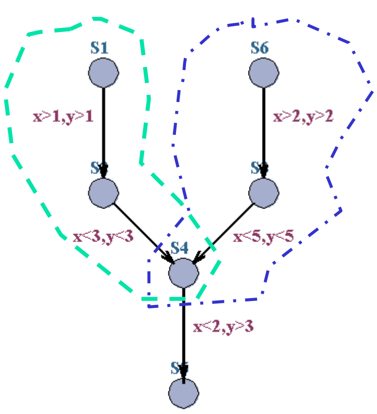
# Under Approximation
(good for finding Bugs quickly, debugging)

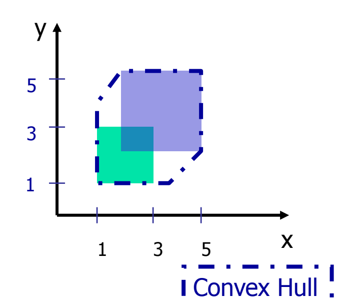
- Possitive answer is safe (you can trust)
  - You can trust your tool if it tells:
    a state is reachable (it means Reachable!)
- Negative answer is Inconclusive
  - You should not trust your tool if it tells:
    a state is non-reachable
  - Some of the branch may be terminated by conflict (the same hashing value of two states)



# Over-approximation

*Convex Hull*



## Over-Approximation
(good for safety property-checking)

- Possitive answer is Inconclusive
  - a state is reachable means Nothing
  (you should not trust your tool when it says so)
  - Some of the transitions may be enabled by Enlarged zones
- Negative answer is safe
  - a state is not reachable means Non-reachable
  (you can trust your tool when it says so)