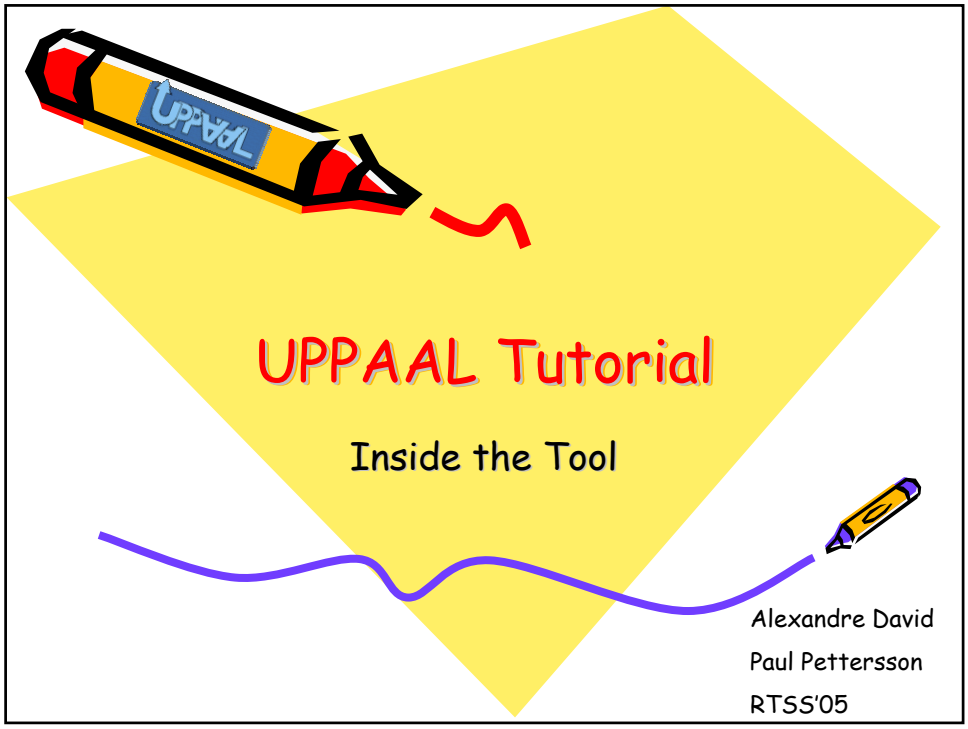
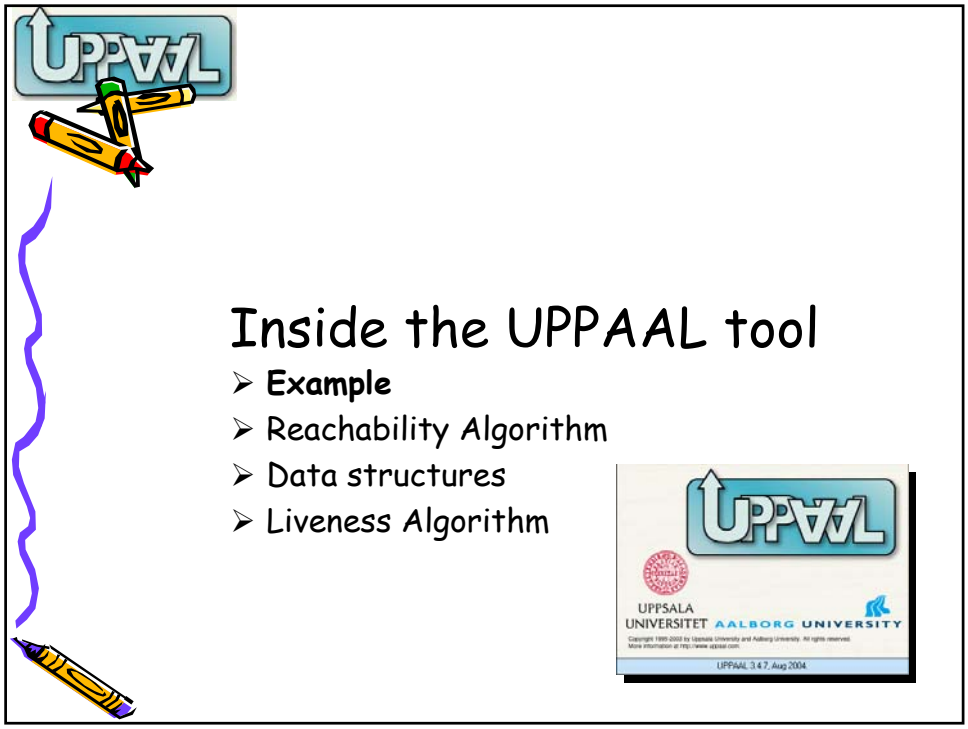


UPPAAL Tutorial

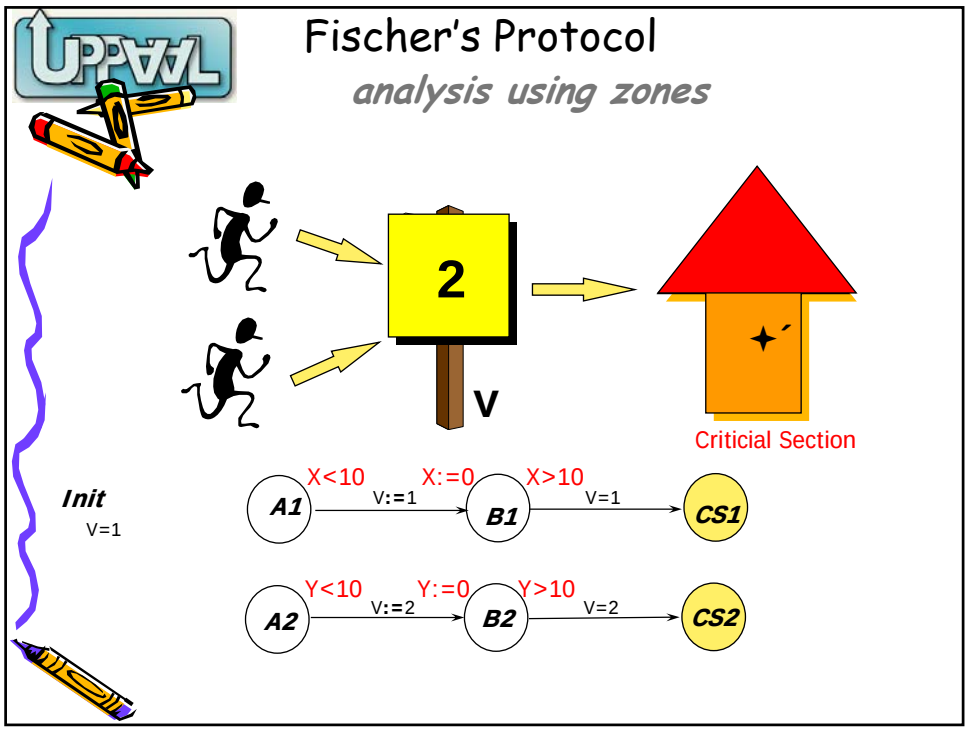
Inside the Tool

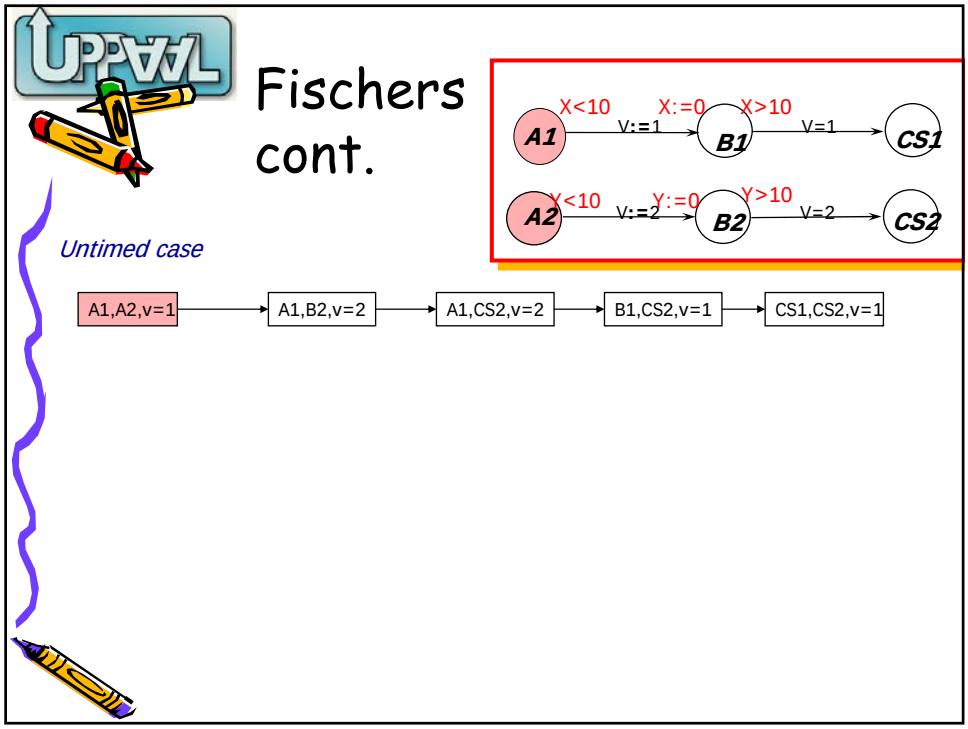
Alexandre David
Paul Pettersson
RTSS'05



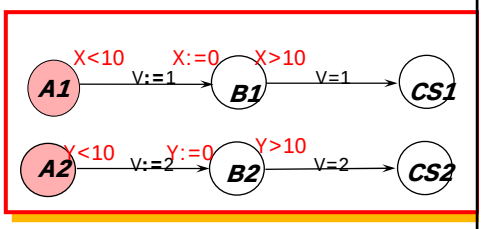
Inside the UPPAAL tool

- **Example**
- Reachability Algorithm
- Data structures
- Liveness Algorithm


Fischer's Protocol
analysis using zones
2
v
Criticial Section
Init
V=1
A1
X<10
V:=1
X:=0
B1
X>10
V=1
CS1
A2
Y<10
V:=2
Y:=0
B2
Y>10
V=2
CS2


Fischers cont.
A1
X<10
V:=1
X:=0
B1
X>10
V=1
CS1
A2
Y<10
V:=2
Y:=0
B2
Y>10
V=2
CS2
Untimed case
A1,A2,v=1
A1,B2,v=2
A1,CS2,v=2
B1,CS2,v=1
CS1,CS2,v=1

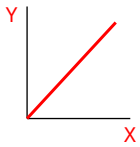
Fischers cont.



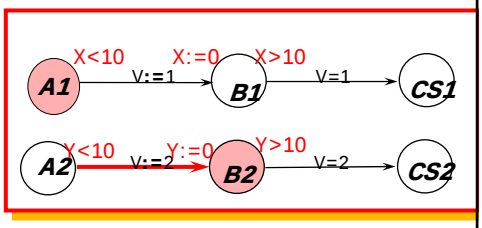
Untimed case

A1,A2,v=1 → A1,B2,v=2 → A1,CS2,v=2 → B1,CS2,v=1 → CS1,CS2,v=1

Taking time into account



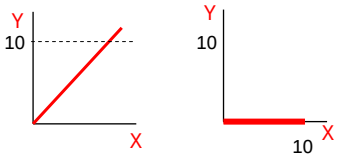
Fischers cont.

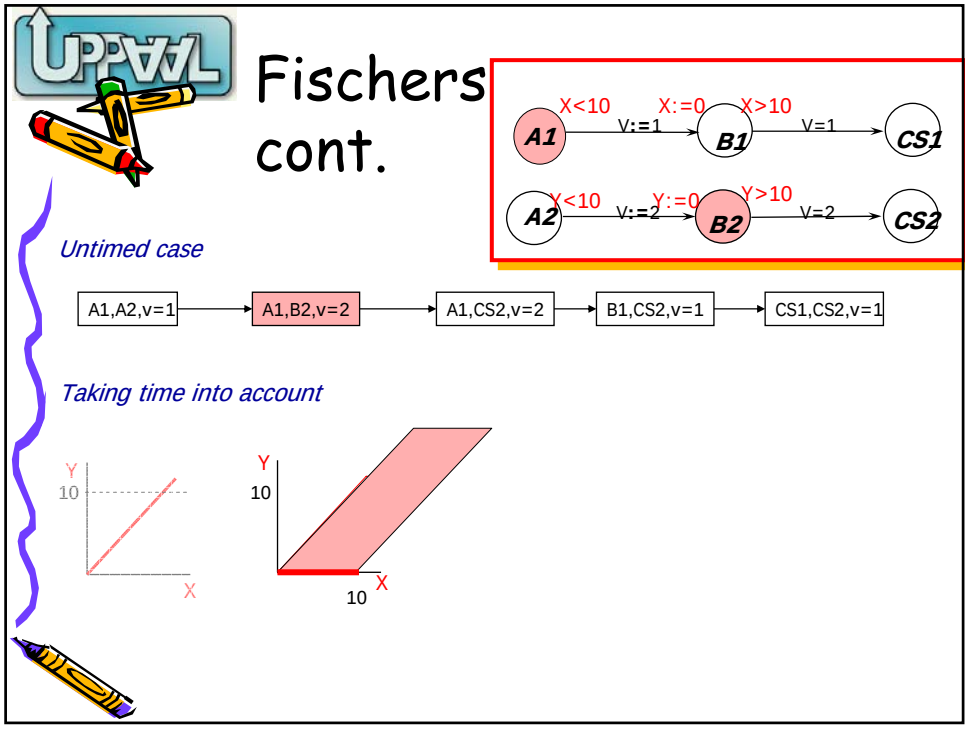


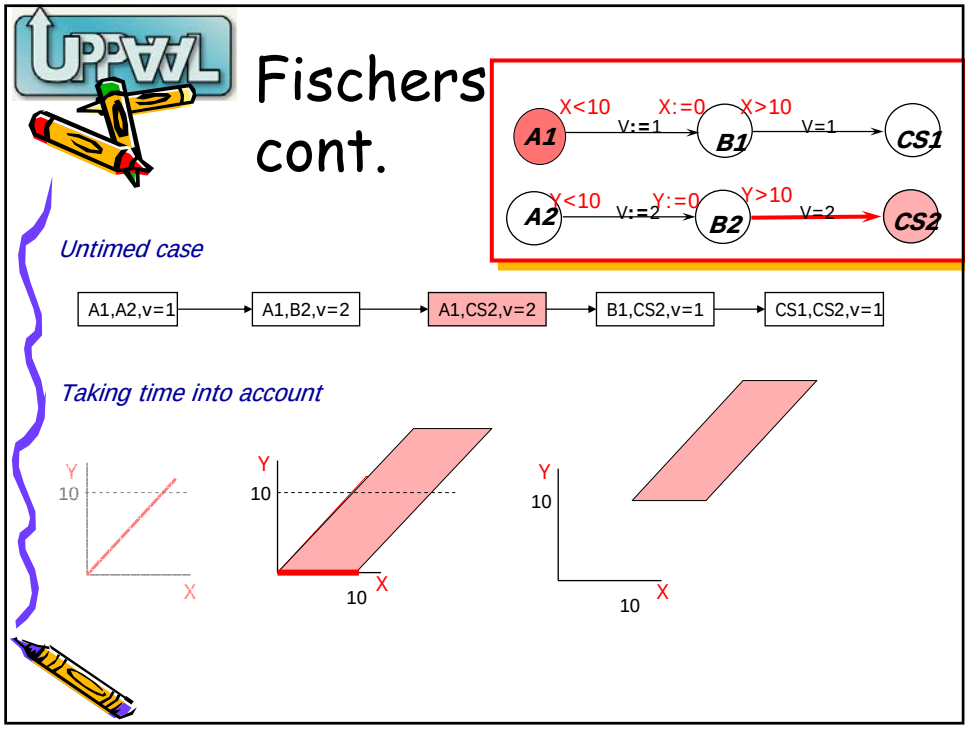
Untimed case

A1,A2,v=1 → A1,B2,v=2 → A1,CS2,v=2 → B1,CS2,v=1 → CS1,CS2,v=1

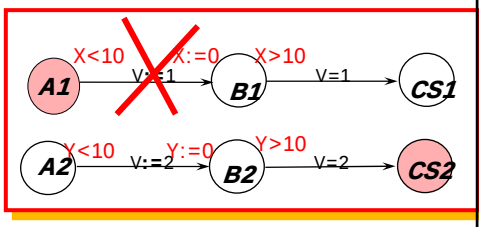
Taking time into account



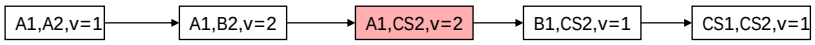

Fischers cont.
A1
B1
CS1
A2
B2
CS2
X<10
X:=0
X>10
V:=1
V=1
Y<10
Y:=0
Y>10
V:=2
V=2
Untimed case
A1,A2,v=1
A1,B2,v=2
A1,CS2,v=2
B1,CS2,v=1
CS1,CS2,v=1
Taking time into account
Y
10
X
10


Fischers cont.
A1
B1
CS1
A2
B2
CS2
X<10
X:=0
X>10
V:=1
V=1
Y<10
Y:=0
Y>10
V:=2
V=2
Untimed case
A1,A2,v=1
A1,B2,v=2
A1,CS2,v=2
B1,CS2,v=1
CS1,CS2,v=1
Taking time into account
Y
10
X
10

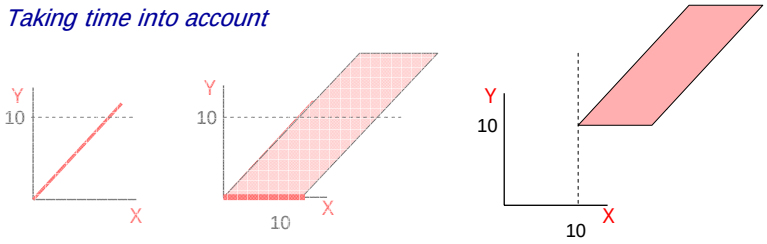
Fischers cont.



Untimed case

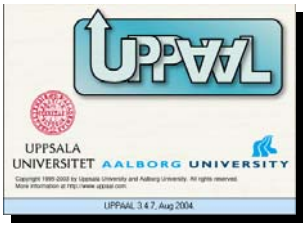


Taking time into account

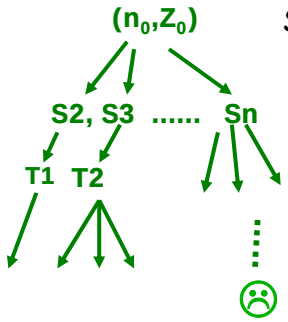


Inside the UPPAAL tool

- Example
- **Reachability Algorithm**
- Data structures
- Liveness Algorithm



We have a search problem



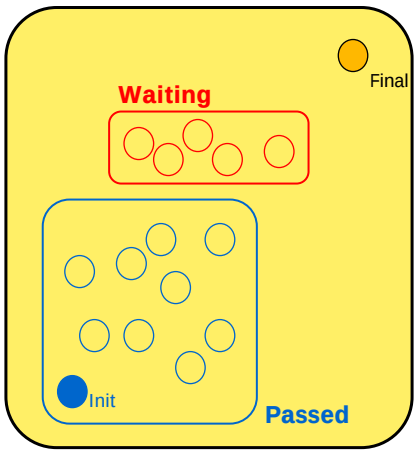
Symbolic state
Symbolic transitions

Reachable?

E<> ☹ = : A[] : ☹

Forward Reachability

Init -> Final ?



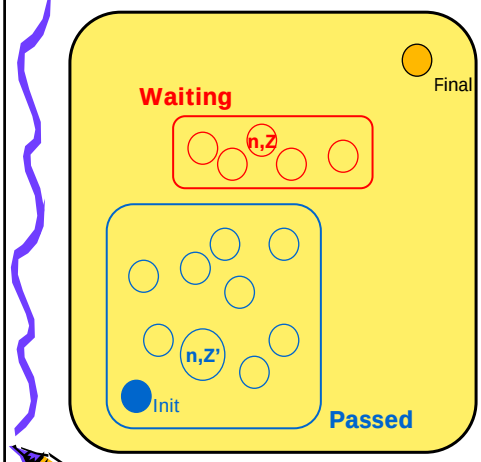
INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

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 ?



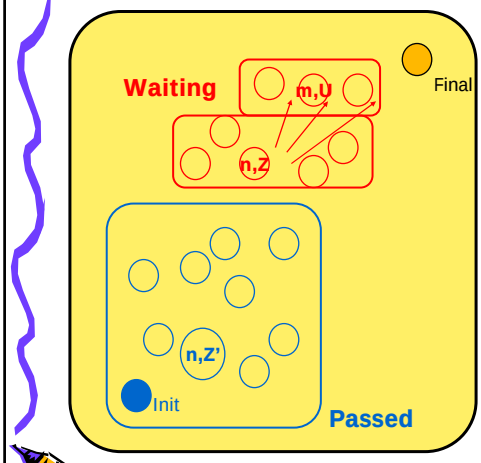
INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

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 ?



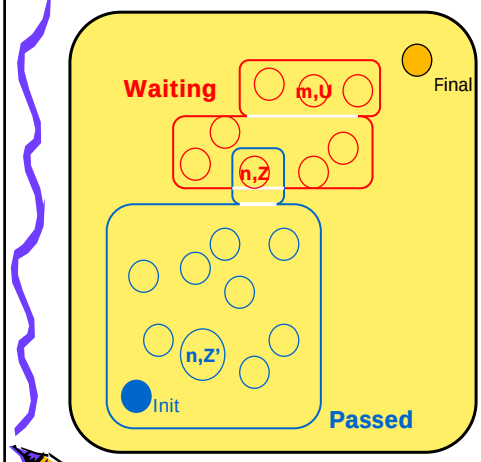
INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

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**

Forward Reachability

Init -> Final ?



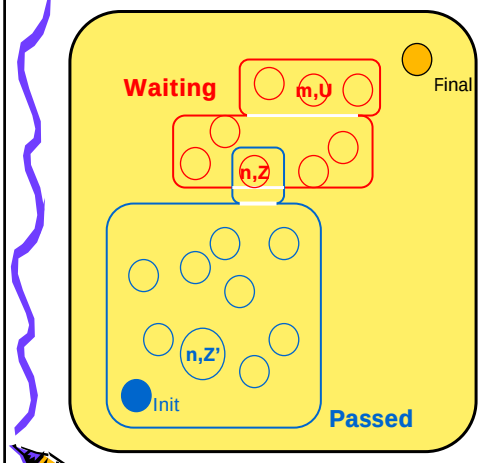
INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

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 ?



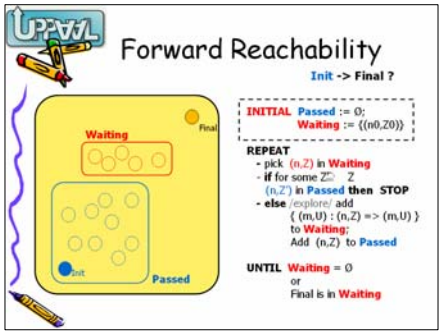
INITIAL **Passed** := Ø;
Waiting := {(n0,Z0)}

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**

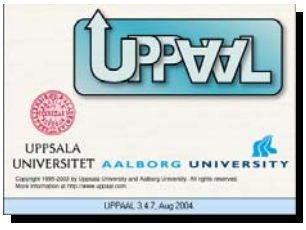
Issues

- Datastructures for **Passed** and **Waiting**
- Do we really need to *always* store in **Passed** ?
- Which symbolic state to select from **Waiting** ?
- Do we really need to add *all* successors ?



Inside the UPPAAL tool

- Example
- Reachability Algorithm
- **Data structures**
- Liveness Algorithm



[Bellman58, Dill89]

Canonical Datastructures for Zones

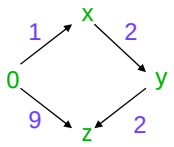
Difference Bounded Matrices

Inclusion

D1

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

Graph

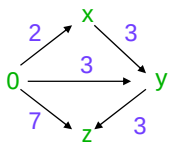


? ⊆ **?**

D2

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

Graph



Canonical Datastructures for Zones

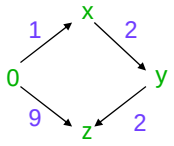
Difference Bounded Matrices

Inclusion

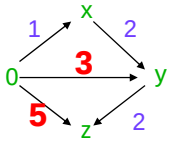
D1

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

Graph



Shortest Path Closure

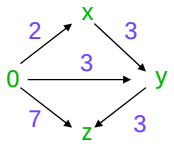


? ⊆ **?**

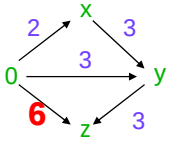
D2

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

Graph



Shortest Path Closure



Canonical Datastructures for Zones

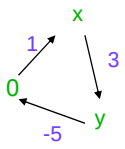
Difference Bounded Matrices

Emptiness

D

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

Graph

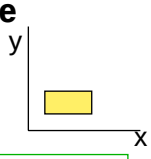


Negative Cycle
iff
empty solution set

Canonical Datastructures for Zones

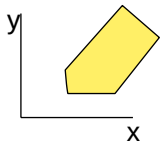
Difference Bounded Matrices

Future



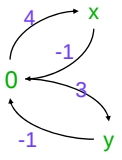
D

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

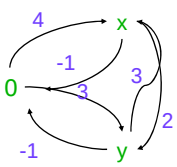


Future D

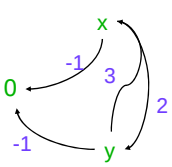
1<=x, 1<=y
-2<=x-y<=3

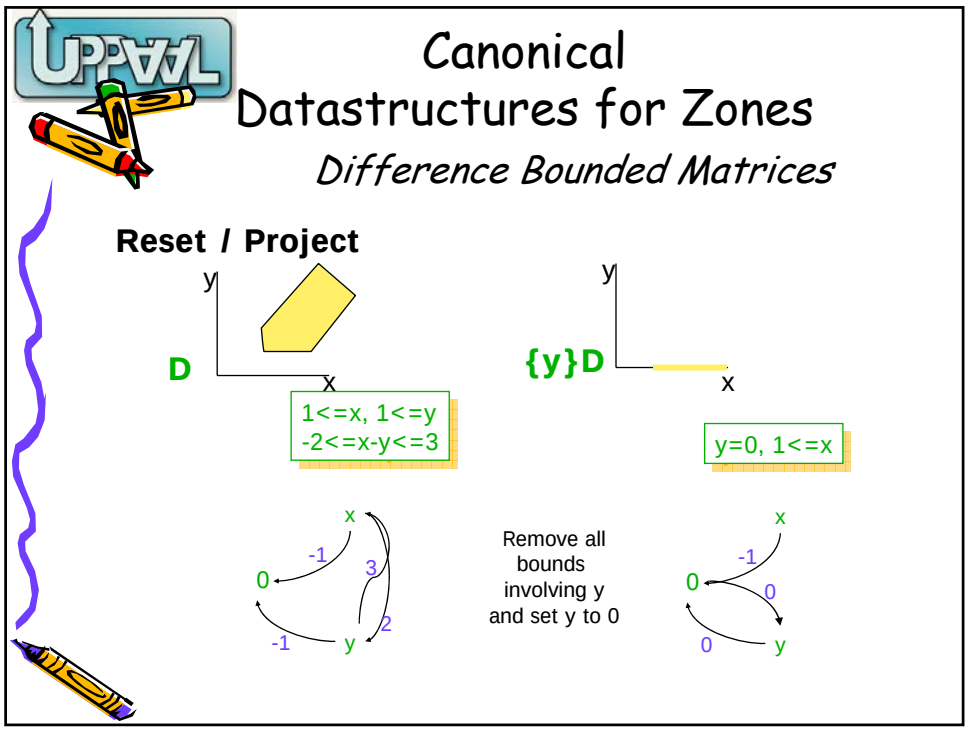


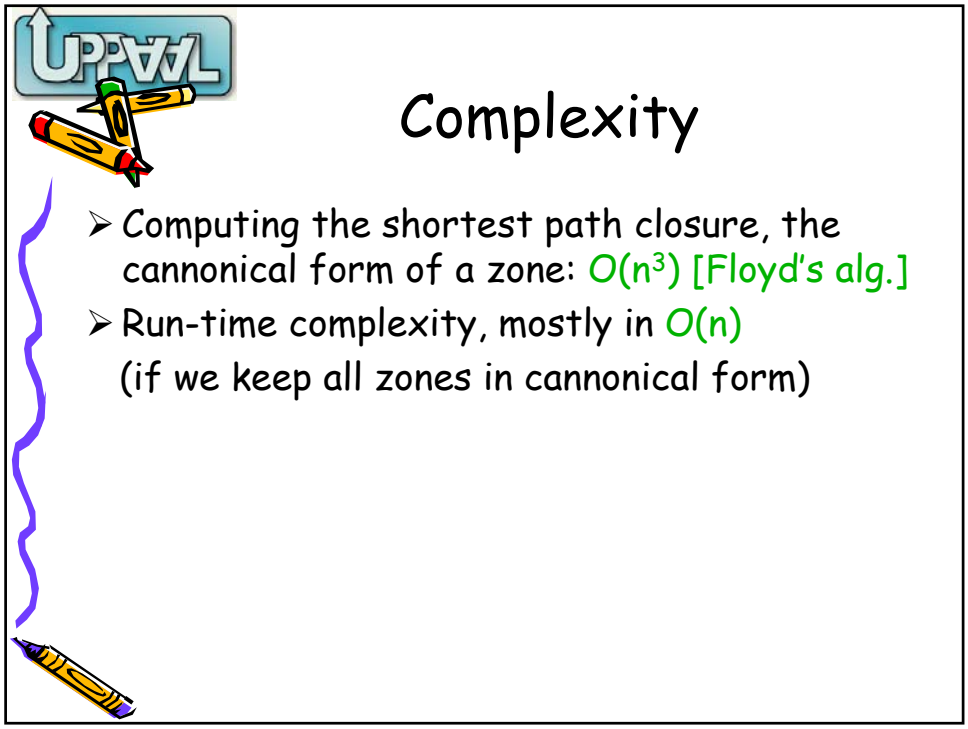
Shortest Path Closure

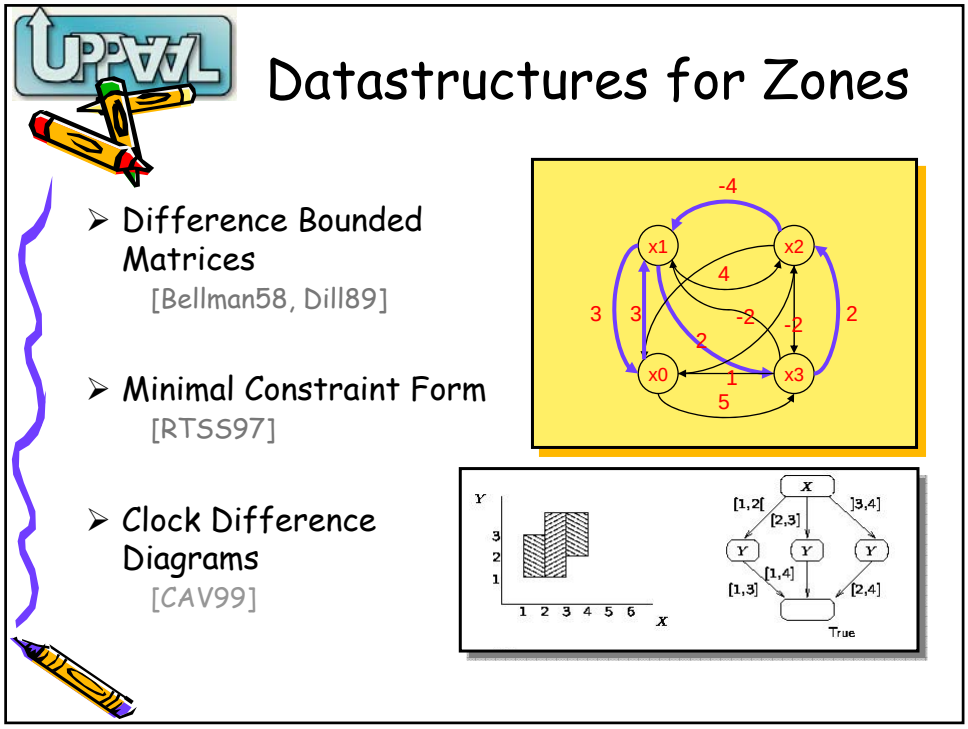


Remove upper bounds on clocks



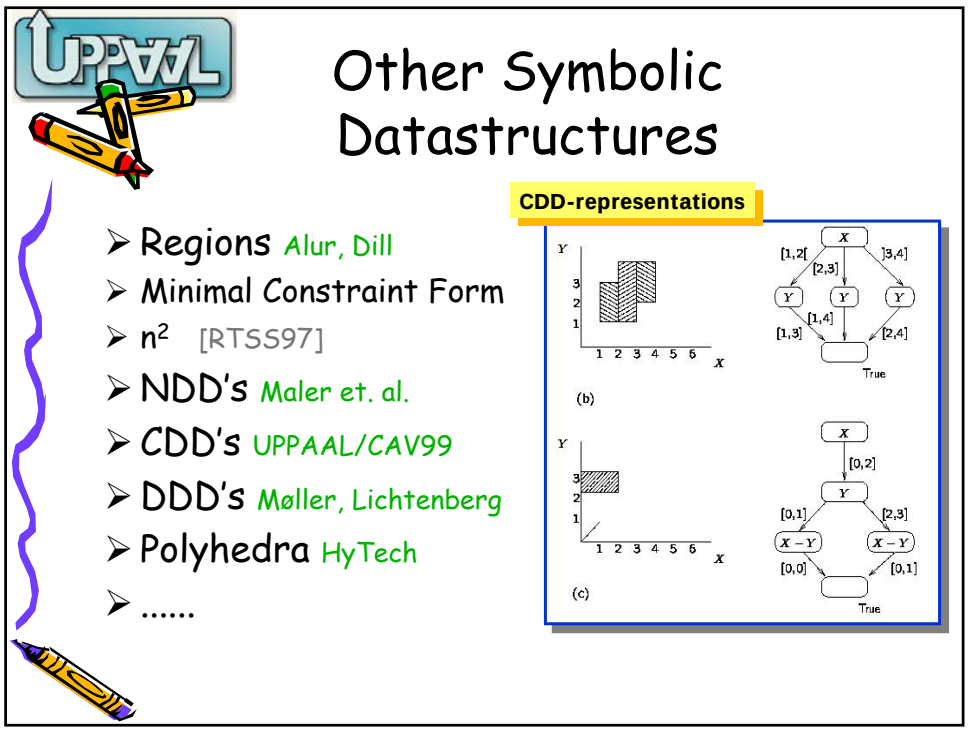

Canonical
Datastructures for Zones
Difference Bounded Matrices
Reset / Project
y
D
x
1<=x, 1<=y
-2<=x-y<=3
y
{y}D
x
y=0, 1<=x
x
-1
3
0
-1
y
2
Remove all
bounds
involving y
and set y to 0
x
-1
0
0
0
y


Complexity
Computing the shortest path closure, the cannonical form of a zone: O(n3) [Floyd's alg.]
Run-time complexity, mostly in O(n)
(if we keep all zones in cannonical form)



Datastructures for Zones

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



Other Symbolic Datastructures

- Regions Alur, Dill
- Minimal Constraint Form
- n^2 [RTSS97]
- NDD's Maler et. al.
- CDD's UPPAAL/CAV99
- DDD's Møller, Lichtenberg
- Polyhedra HyTech
- .....

Inside the UPPAAL tool

- Example
- Reachability Algorithm
- Data structures
- **Liveness Algorithm**

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)

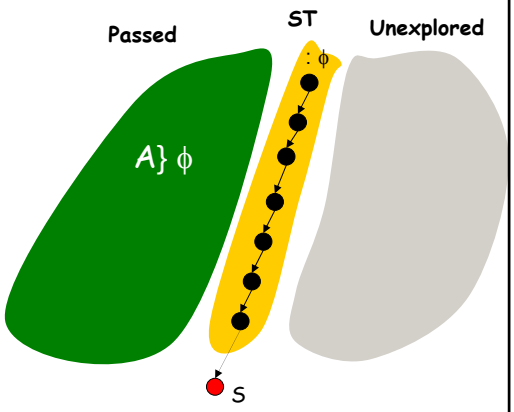
Bouajjani, Tripakis, Yovine'97
On-the-fly symbolic model checking of TCTL

28

Liveness Algorithm

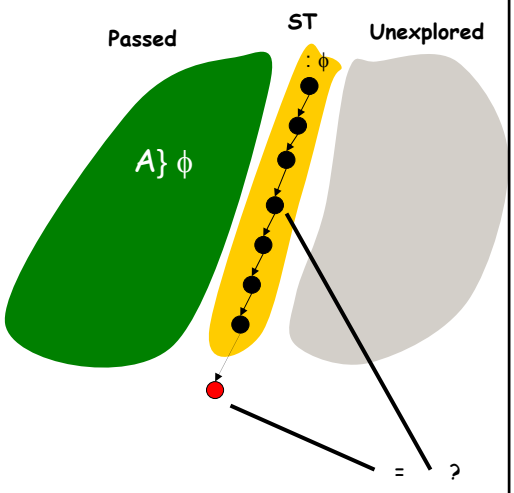
Bouajjani, Tripakis, Yovine, 97

```
proc Eventually(S0, φ) ≡
  ST := ∅
  Passed := ∅
  Search(delay(S0, ¬φ))
  exit(true)
end
proc Search(S) ≡
  if loop(S, ST) then exit(false) fi
  S := S ∧ ¬φ
  push(ST, S)
  if unbounded(S) ∨ deadlocked(S) then
                    exit(false) fi
  if ∀S' ∈ Passed : S ⋢ S'
    then foreach S' : S ⇒a S' do
           Search(delay(S', ¬φ))
         od
  fi
  Passed := Passed ∪ {pop(ST)}
end
```



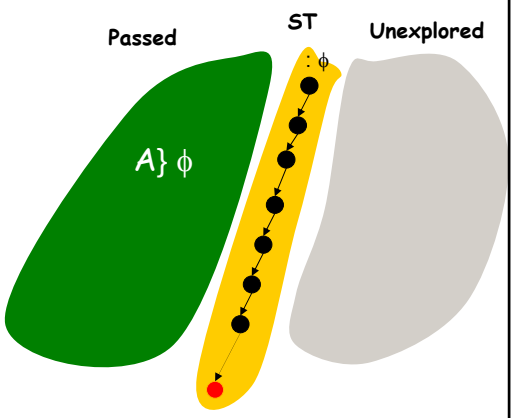
Liveness Algorithm

```
proc Eventually(S0, φ) ≡
  ST := ∅
  Passed := ∅
  Search(delay(S0, ¬φ))
  exit(true)
end
proc Search(S) ≡
  if loop(S, ST) then exit(false) fi
  S := S ∧ ¬φ
  push(ST, S)
  if unbounded(S) ∨ deadlocked(S) then
                    exit(false) fi
  if ∀S' ∈ Passed : S ⋢ S'
    then foreach S' : S ⇒a S' do
           Search(delay(S', ¬φ))
         od
  fi
  Passed := Passed ∪ {pop(ST)}
end
```

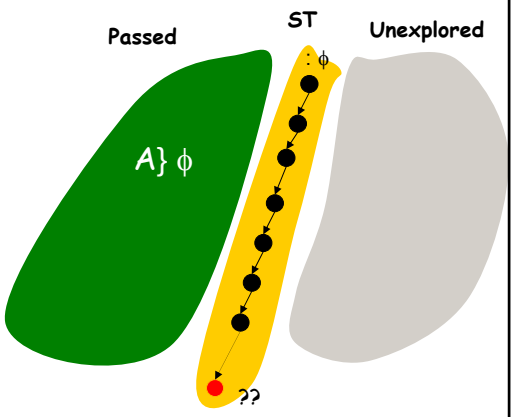
Liveness Algorithm

```
proc Eventually(S0, φ) ≡
  ST := ∅
  Passed := ∅
  Search(delay(S0, ¬φ))
  exit(true)
end
proc Search(S) ≡
  if loop(S, ST) then exit(false) fi
  S := S ∧ ¬φ
  push(ST, S)
  if unbounded(S) ∨ deadlocked(S) then
                    exit(false) fi
  if ∀S' ∈ Passed : S ⋢ S'
    then foreach S' : S ⇒a S' do
           Search(delay(S', ¬φ))
         od
  fi
  Passed := Passed ∪ {pop(ST)}
end
```



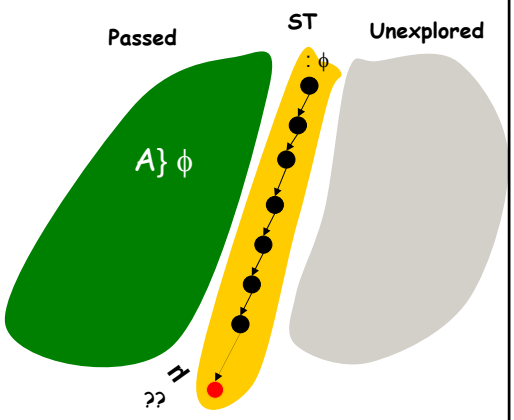
Liveness Algorithm

```
proc Eventually(S0, φ) ≡
  ST := ∅
  Passed := ∅
  Search(delay(S0, ¬φ))
  exit(true)
end
proc Search(S) ≡
  if loop(S, ST) then exit(false) fi
  S := S ∧ ¬φ
  push(ST, S)
  if unbounded(S) ∨ deadlocked(S) then
                    exit(false) fi
  if ∀S' ∈ Passed : S ⋢ S'
    then foreach S' : S ⇒a S' do
           Search(delay(S', ¬φ))
         od
  fi
  Passed := Passed ∪ {pop(ST)}
end
```

Liveness Algorithm

```
proc Eventually(S0, φ) ≡
  ST := ∅
  Passed := ∅
  Search(delay(S0, ¬φ))
  exit(true)
end
proc Search(S) ≡
  if loop(S, ST) then exit(false) fi
  S := S ∧ ¬φ
  push(ST, S)
  if unbounded(S) ∨ deadlocked(S) then
                    exit(false) fi
  if ∀S' ∈ Passed : S ⋢ S'
    then foreach S' : S ⇒[a] S' do
           Search(delay(S', ¬φ))
         od
  fi
  Passed := Passed ∪ {pop(ST)}
end
```



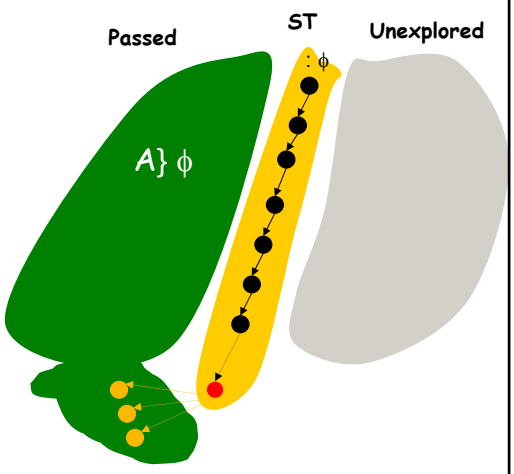
Liveness Algorithm

```
proc Eventually(S0, φ) ≡
  ST := ∅
  Passed := ∅
  Search(delay(S0, ¬φ))
  exit(true)
end
proc Search(S) ≡
  if loop(S, ST) then exit(false) fi
  S := S ∧ ¬φ
  push(ST, S)
  if unbounded(S) ∨ deadlocked(S) then
                    exit(false) fi
  if ∀S' ∈ Passed : S ⋢ S'
    then foreach S' : S ⇒[a] S' do
           Search(delay(S', ¬φ))
         od
  fi
  Passed := Passed ∪ {pop(ST)}
end
```

Passed
ST
Unexplored
: ϕ
A} ϕ

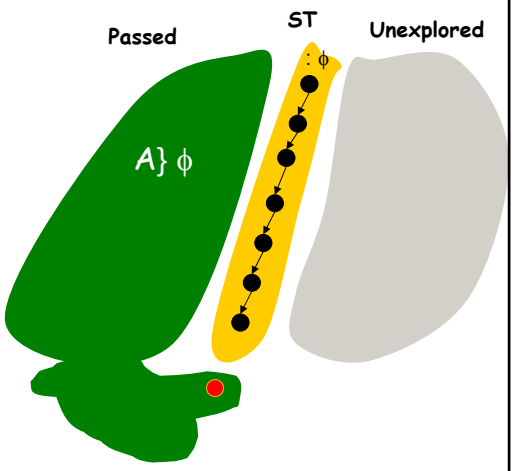
Liveness Algorithm

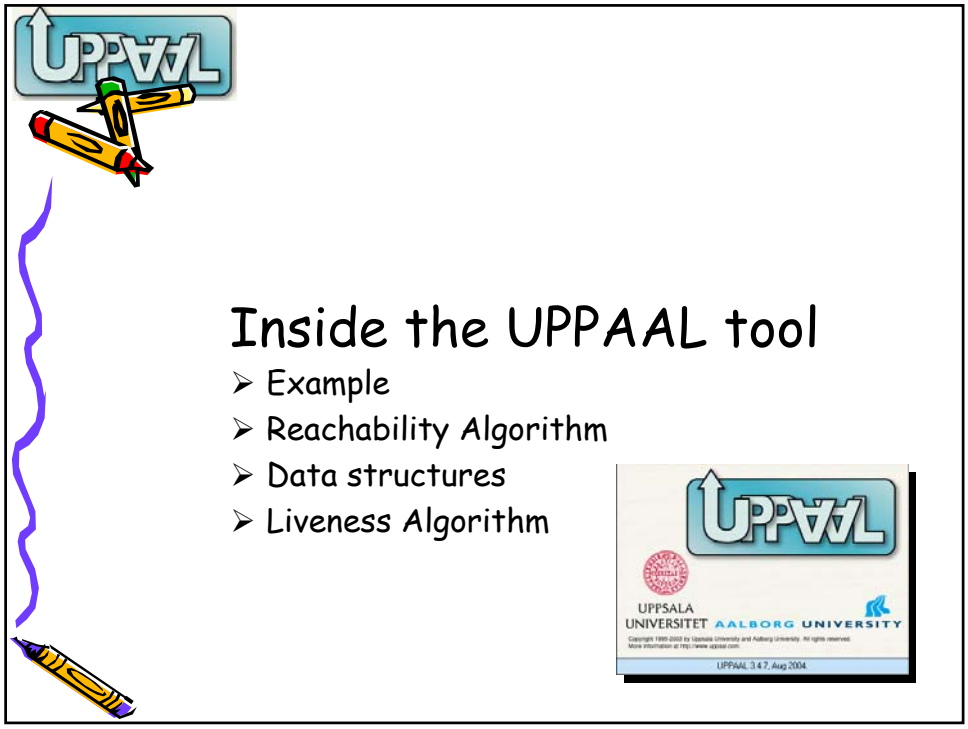
```
proc Eventually(S0, φ) ≡
  ST := ∅
  Passed := ∅
  Search(delay(S0, ¬φ))
  exit(true)
end
proc Search(S) ≡
  if loop(S, ST) then exit(false) fi
  S := S ∧ ¬φ
  push(ST, S)
  if unbounded(S) ∨ deadlocked(S) then
                    exit(false) fi
  if ∀S' ∈ Passed : S ⋢ S'
    then foreach S' : S ⇒a S' do
           Search(delay(S', ¬φ))
         od
  fi
  Passed := Passed ∪ {pop(ST)}
end
```



Liveness Algorithm

```
proc Eventually(S0, φ) ≡
  ST := ∅
  Passed := ∅
  Search(delay(S0, ¬φ))
  exit(true)
end
proc Search(S) ≡
  if loop(S, ST) then exit(false) fi
  S := S ∧ ¬φ
  push(ST, S)
  if unbounded(S) ∨ deadlocked(S) then
                    exit(false) fi
  if ∀S' ∈ Passed : S ⋢ S'
    then foreach S' : S ⇒a S' do
           Search(delay(S', ¬φ))
         od
  fi
  Passed := Passed ∪ {pop(ST)}
end
```




Inside the UPPAAL tool
Example
Reachability Algorithm
Data structures
Liveness Algorithm
UPPAAL
UPPSALA
UNIVERSITET
AALBORG UNIVERSITY
UPPAAL 3.4.7, Aug 2004