Beyond Liveness

Efficient Parameter Synthesis for Time Bounded Liveness

Gerd Behrmann
Kim Guldstrand Larsen
Jacob Illum Rasmussen

BRICS/CISS, Aalborg University, DENMARK

Safety & Liveness

Liveness Manifesto
"Beyond Safety", workshop
Schloss Ringberg, Germany 2004.

- **WANG YI:**
  - Safety properties
    = those that can be checked with
    reachability analysis
  - Liveness: properties
    = those that can not be checked without
    loop detection
  - It seems that liveness is very much related to QoS properties e.g.
    "Over time, every 100 events that occur,
    there must be at least 10 good ones".

- **LESLIE LAMPORT:**
  - Knowing that something will eventually happen isn't particularly useful; we'd like to know that it happens before the sun explodes in a few billion years.

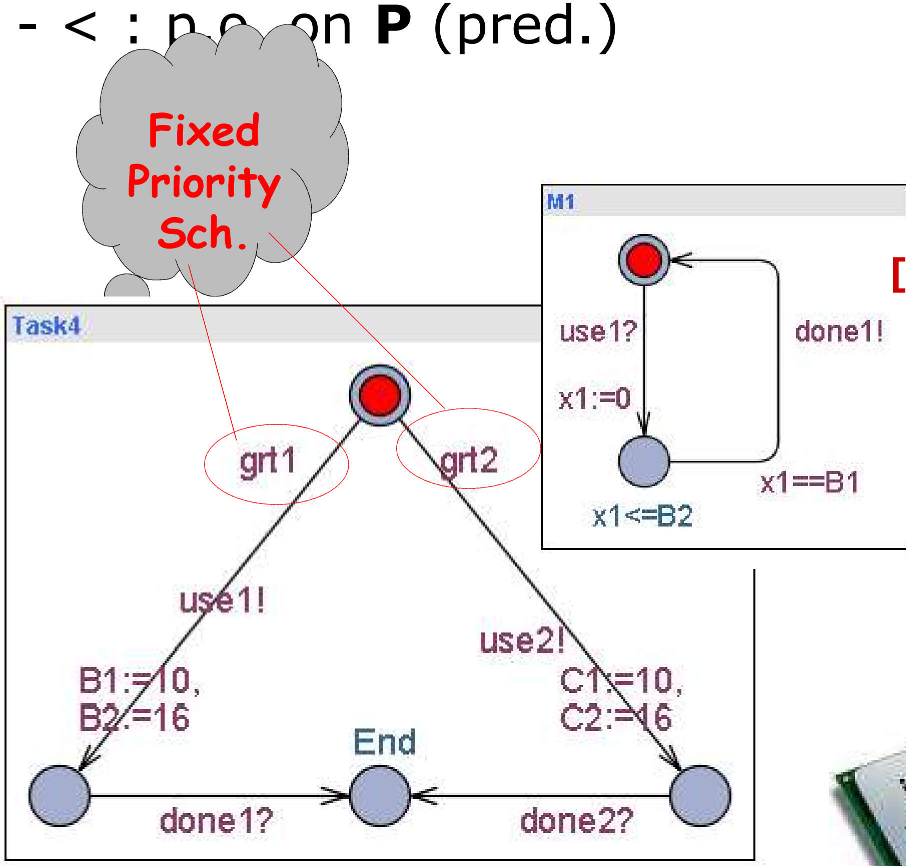
Safety & Liveness

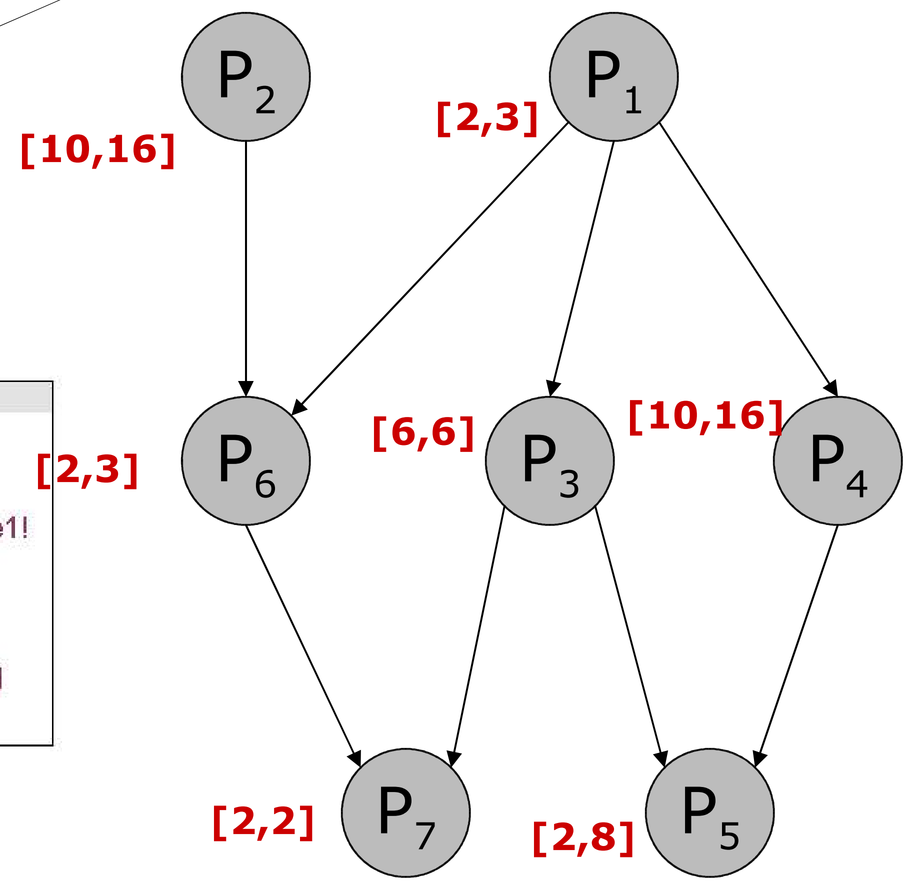
- **PAROSH ABDULLA:**
  - The traditional definition of liveness "something good will eventually occur" is not very useful for an engineer. It is not satisfactory to know that your program will terminate within one year. Bounded liveness is practically more relevant, but it is a safety property.
- **HARDI HUNGAR:**
  - Liveness is what remains if one abstracts away the time bounds which usually come with every response property.
  - 1. A liveness property is useless in practice if it is not accompanied by bound
    2. A liveness property accompanied by a bound is a safety property
    3. Consequence of the above: There is no liveness property which is useful in practice

Bounded Liveness

Task Graph Scheduling w Uncertainty,

- Task **P**={P_1,.., P_m}
- Machines **M**={M_1,..,M_n}
- Duration Δ : **P** ! $N_1 x N_1$
- < : p.o. on **P** (pred.)





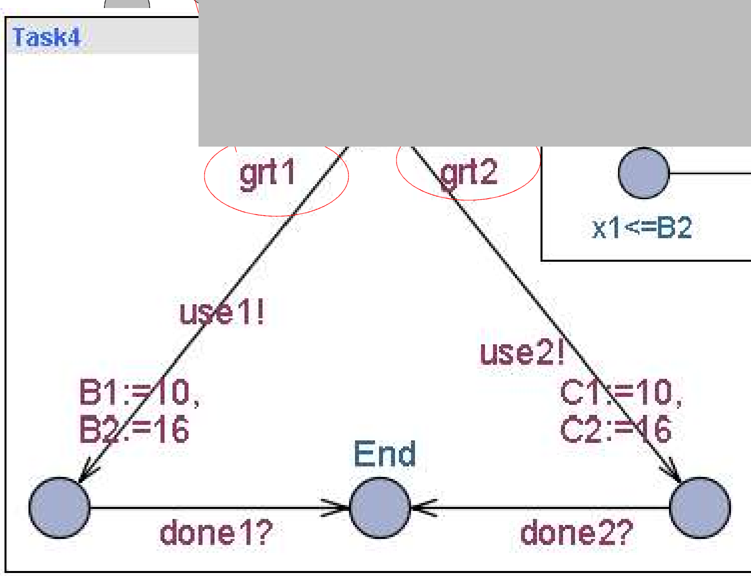
M = {M_1,M_2}

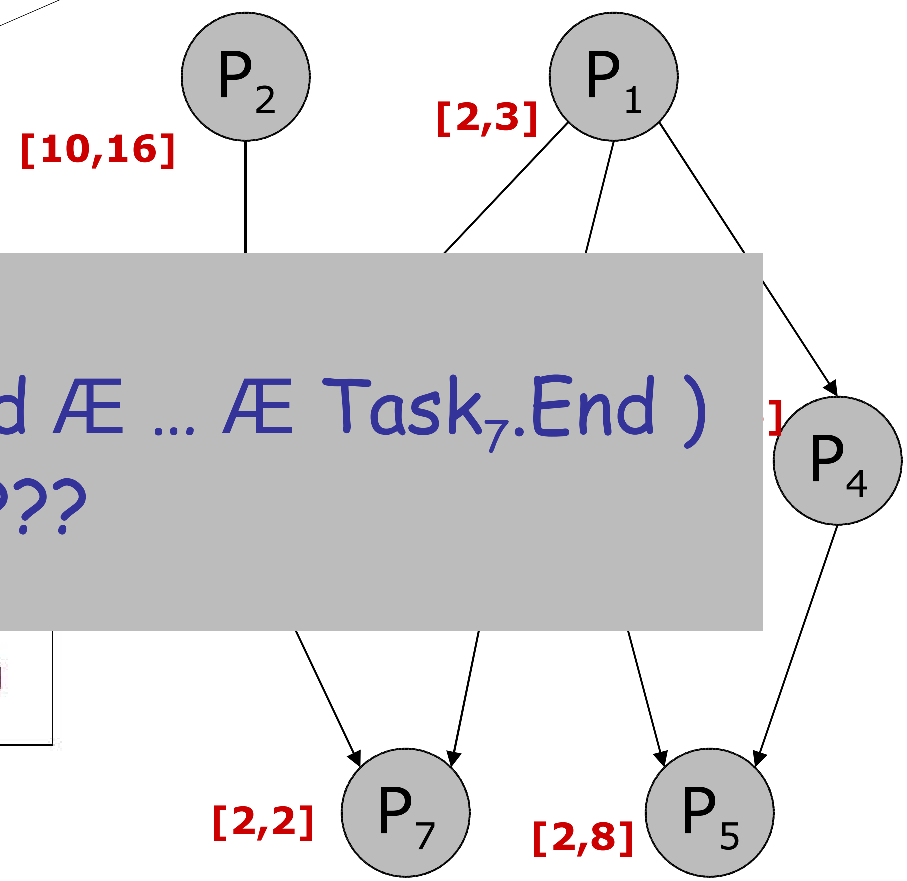
Bounded **Liveness**

Task Graph Scheduling w **Uncertainty**,

- Task **P**={P_1,.., P_m}
- Machines **M**={M_1,..,M_n}
- Duration Δ : **P** ! **N_1xN_1**
- < : p.o. on **P** (pred.)

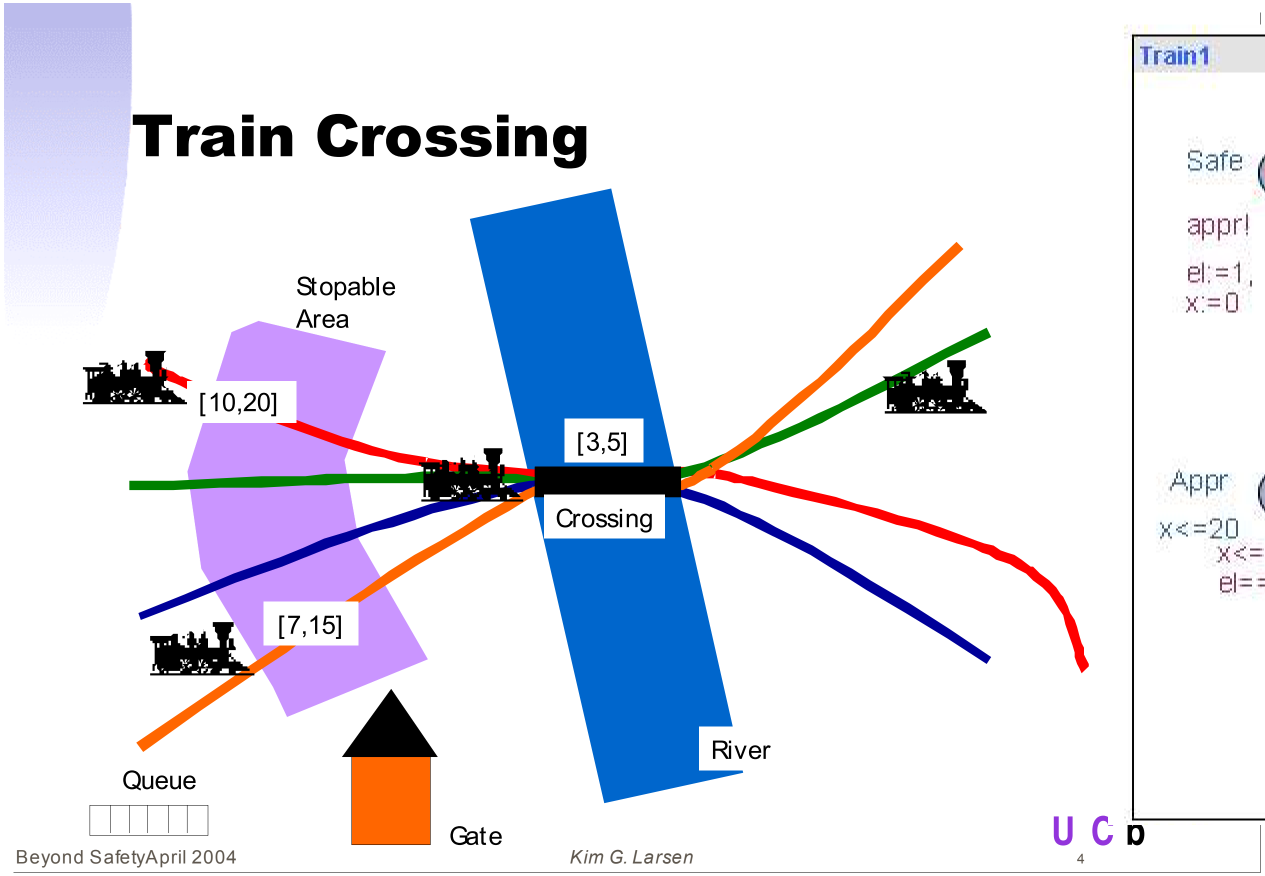
A}. 35 ($Task_1$.End Æ … Æ $Task_7$.End)
???

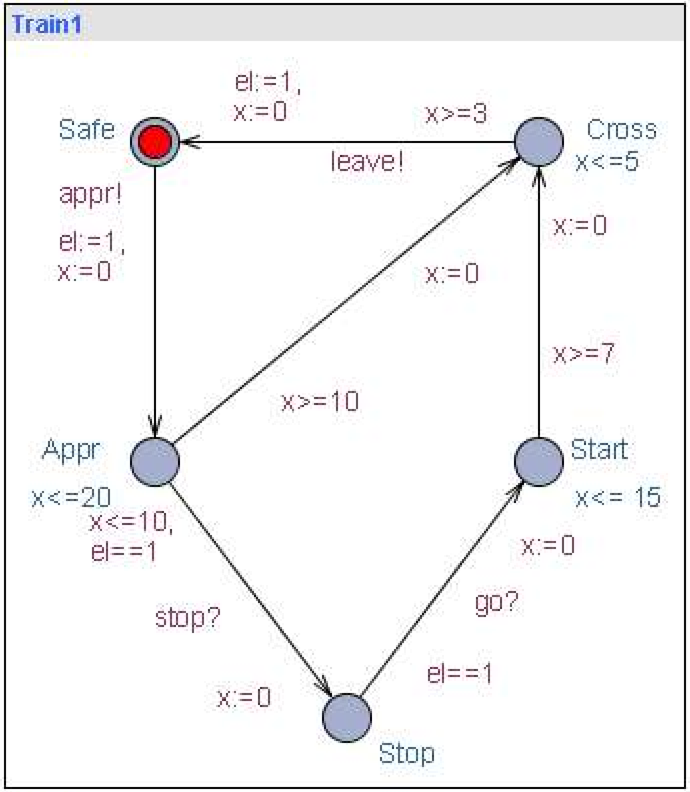




M = {M_1,M_2}

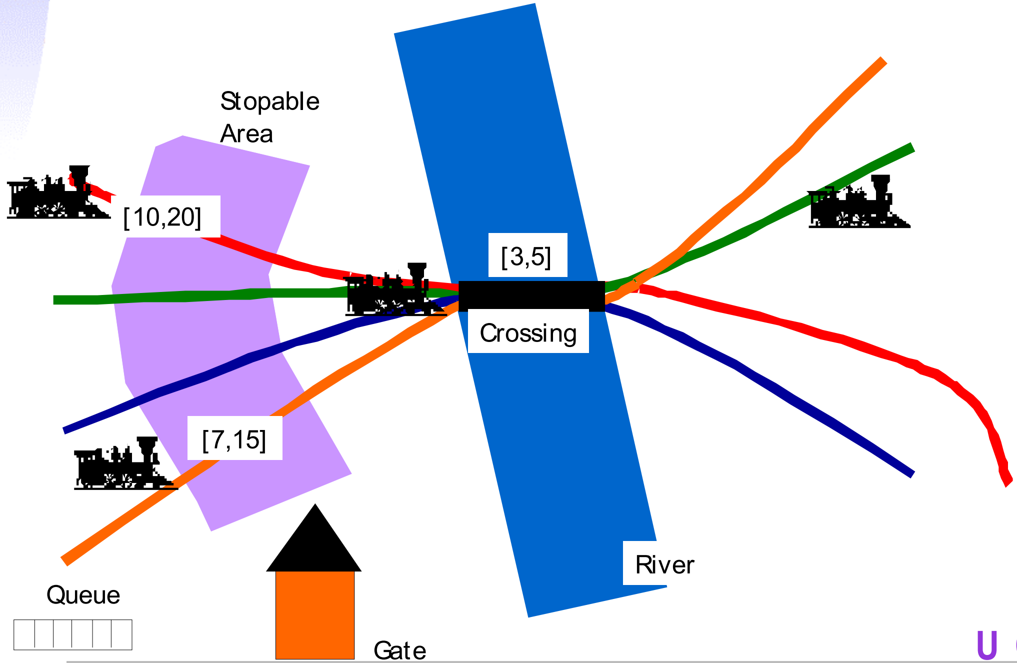
Bounded Liveness

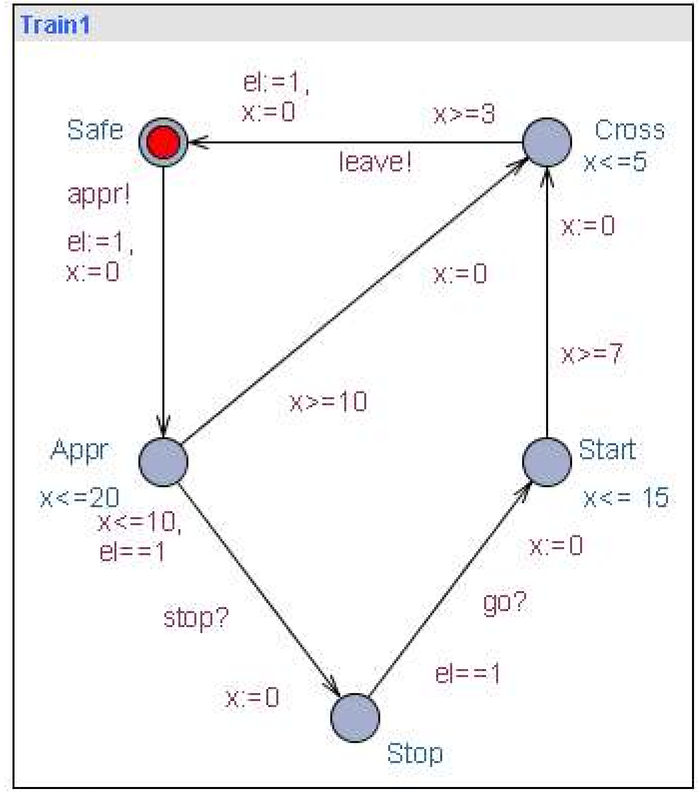




Bounded Liveness

Train Crossing





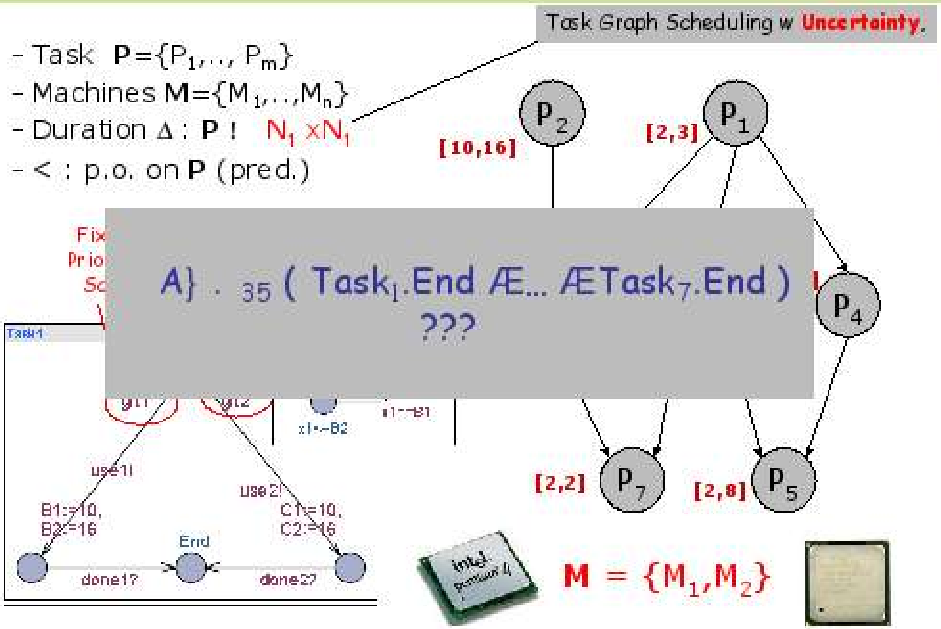
U C p

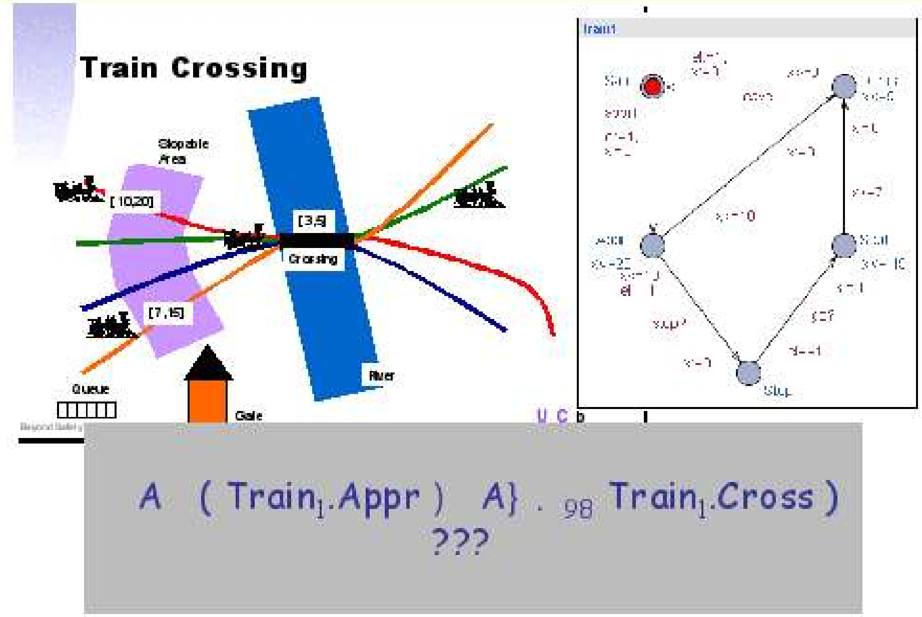
$A\ (\ Train_1.Appr\ )\ A\}._{98}\ Train_1.Cross\ )$

???

Beyond Liveness = ?

Beyond Liveness = Parameter Synthesis





How to synthesize the minimum value *p* for which a time-bounded liveness property is valid ?

Efficiency

Farn Wang, 2000: *Parameterized Regions*
Bruyere, Dall'Olio, Raskin, 2003:
Parameterized TCTL using Presburger Arithmetic
Metzner, 2004: *Binary Search*

Outline of Talk

- Simulation Graph for Timed Automata

- Reduction to Reachability Analysis
- Parameterized Liveness Analysis
- Experimental Results

- Extensions
  - Priced Timed Automata
    (Worst Cost Execution)
  - Timed Games
    (Time-optimal Winning Strategies)
- Conclusions

Outline of Talk

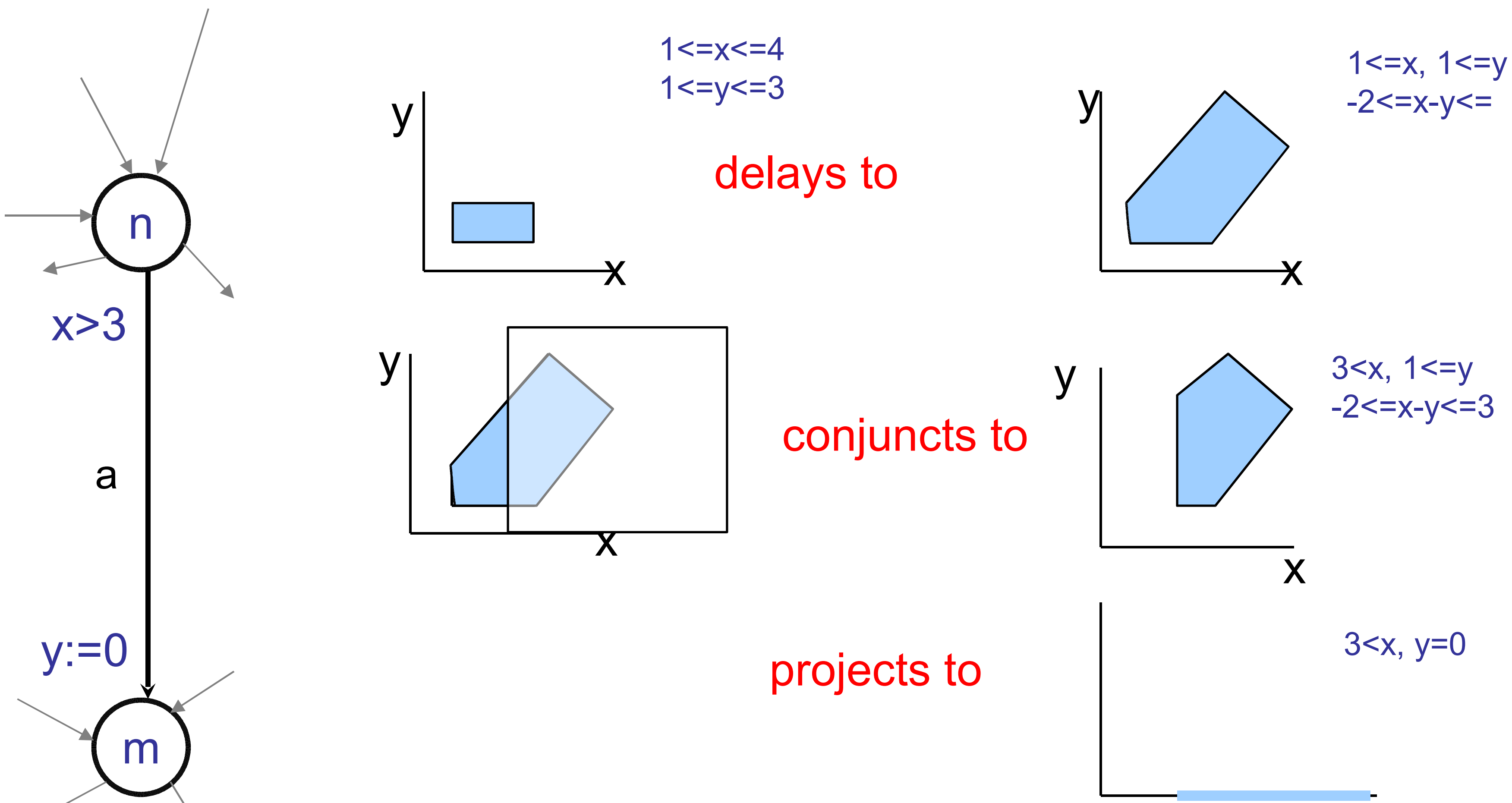
- Simulation Graph for Timed Automata

- Reduction to Reachability Analysis
- Parameterized Liveness Analysis
- Experimental Results

- Extensions
  - Priced Timed Automata
    (Worst Cost Execution)
  - Timed Games
    (Time-optimal Winning Strategies)
- Conclusions

Symbolic States & Transitions

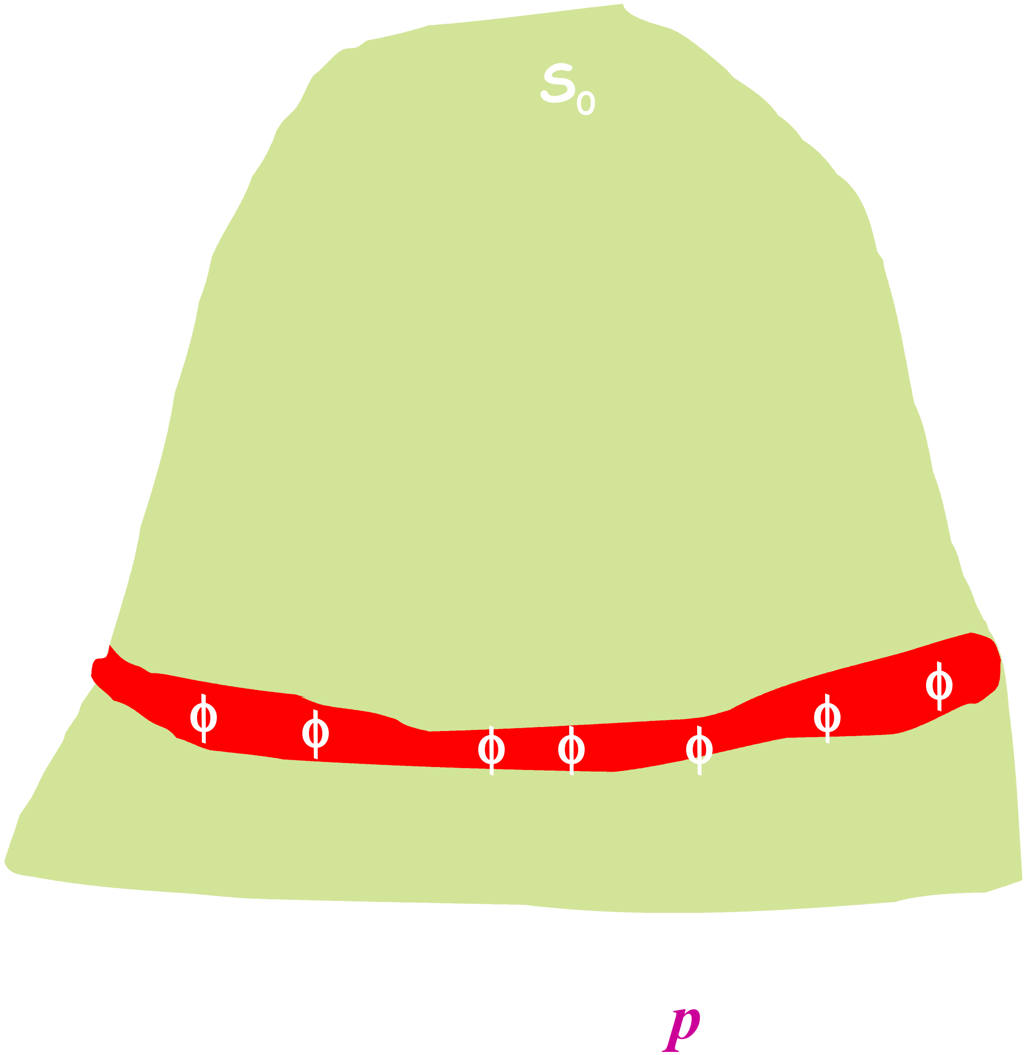
using Zones



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

Reduction to Reachability

$$
\begin{array}{l}
\underline{\textbf{proc}}\ \mathit{Reachable}(S_0,\varphi)\ \equiv \\
\quad \mathit{pre}(S_0 \models A\Diamond\varphi) \\
\quad \mathit{Wait} := \{\mathit{delay}(S_0[c \mapsto 0], \neg\varphi)\} \\
\quad \mathit{Passed} := \emptyset \\
\quad p := 0 \\
\quad \underline{\textbf{while}}\ \mathit{Wait} \neq \emptyset\ \underline{\textbf{do}} \\
\qquad\quad \text{let } S \in \mathit{Wait} \\
\qquad\quad \mathit{Wait} = \mathit{Wait} \setminus \{S\} \\
\qquad\quad p := \max(p, \max_c(S)) \\
\qquad\quad S := S \wedge \neg\varphi \\
\qquad\quad \underline{\textbf{foreach}}\ S' : S \stackrel{a}{\Rightarrow} S'\ \underline{\textbf{do}} \\
\qquad\qquad S' := \mathrm{delay}(S', \neg\varphi) \\
\qquad\qquad (S' := \mathrm{extrapolate}(S')^{\ddagger}) \\
\qquad\qquad \underline{\textbf{if}}\ \forall S'' \in \mathit{Passed} : S' \not\subseteq S'' \\
\qquad\qquad\quad \underline{\textbf{then}}\ \mathit{Wait} := \mathit{Wait} \cup \{S'\} \\
\qquad\qquad \underline{\textbf{fi}} \\
\qquad\quad \underline{\textbf{od}} \\
\qquad\quad \mathit{Passed} := \mathit{Passed} \cup \{S\} \\
\quad \underline{\textbf{od}} \\
\quad \underline{\textbf{exit}}(p) \\
\underline{\textbf{end}}
\end{array}
$$



Reduction to Reachability

Assumption: A}ϕ holds !!

Add clock *c* to model

Add global variable ***p***

while $Wait \neq \emptyset$ **do**

$\quad$ let $S \in Wait$

$\quad Wait = Wait \setminus \{S\}$

$\quad p := \max(p, \max_c(S))$

$\quad S := S \wedge \neg\varphi$

$\quad$ **foreach** $S' : S \overset{a}{\Rightarrow} S'$ **do**

$\quad\quad S' := \text{delay}(S', \neg\varphi)$

$\quad\quad (S' := \text{extrapolate}(S')^{\ddagger})$

$\quad\quad$ **if** $\forall S'' \in Passed : S' \not\subseteq S''$

$\quad\quad\quad$ **then** $Wait := Wait \cup \{S'\}$

$\quad\quad$ **fi**

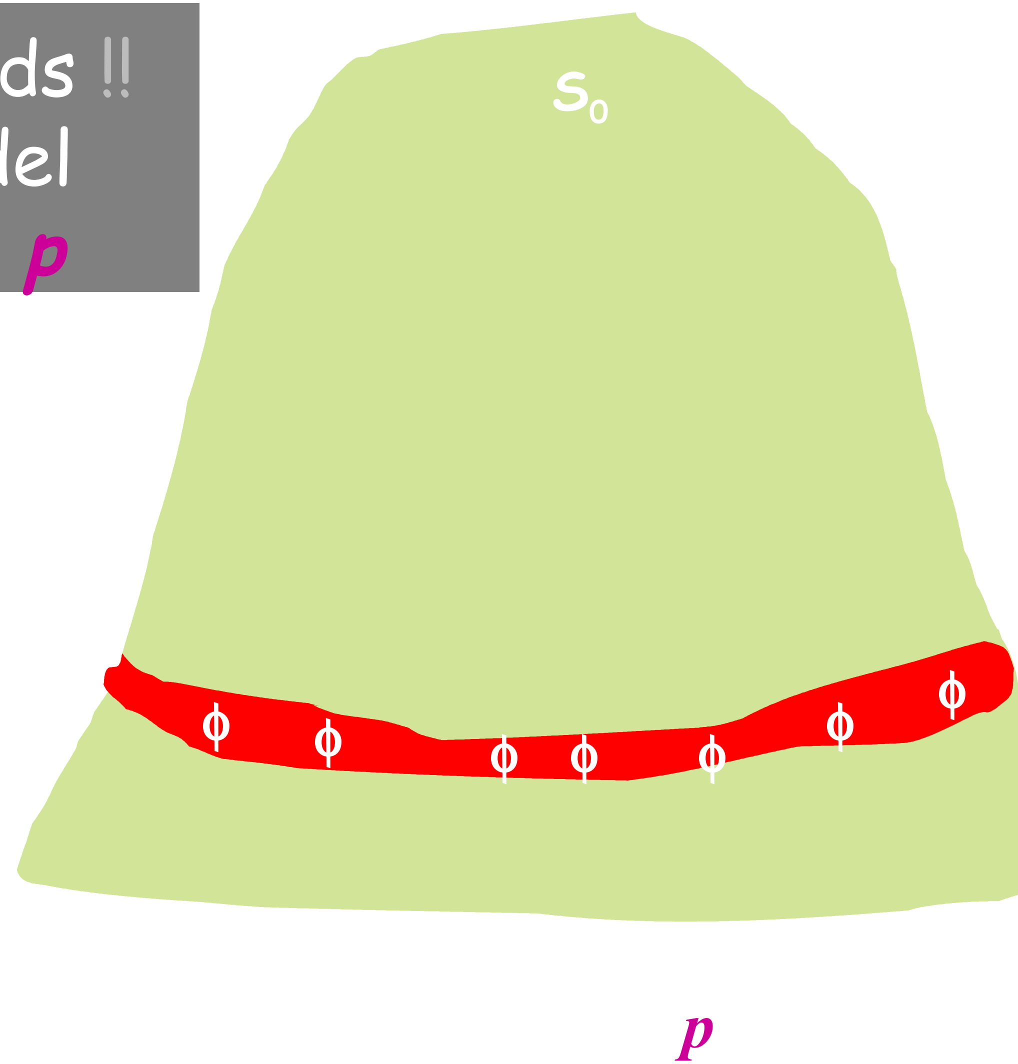
$\quad$ **od**

$\quad Passed := Passed \cup \{S\}$

od

exit(p)

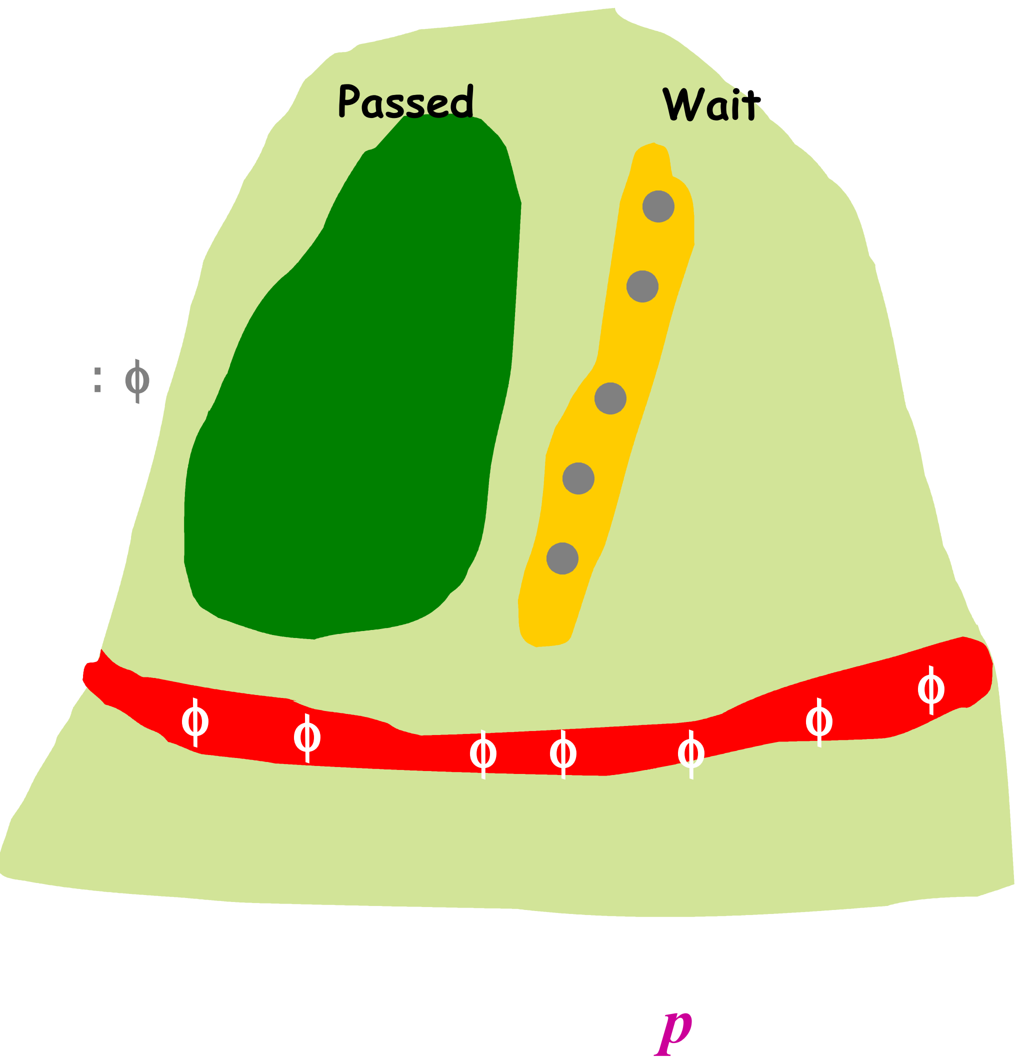
end



Reduction to Reachability

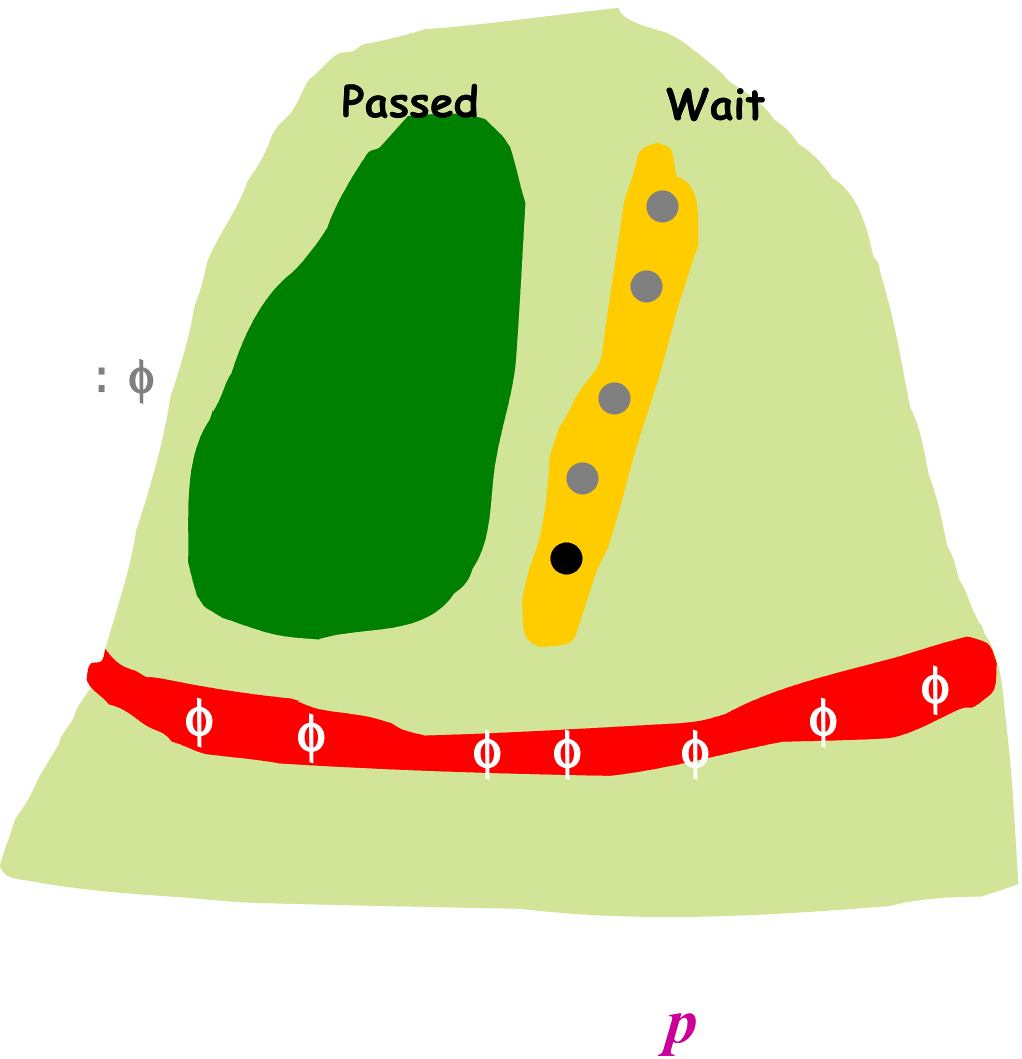
$\underline{\textbf{proc}}\ \mathit{Reachable}(S_0, \varphi)\ \equiv$
- $\mathit{pre}(S_0 \models A\Diamond\varphi)$
- $\mathit{Wait} := \{\mathit{delay}(S_0[c \mapsto 0], \neg\varphi)\}$
- $\mathit{Passed} := \emptyset$
- $p := 0$
- $\underline{\textbf{while}}\ \mathit{Wait} \neq \emptyset\ \underline{\textbf{do}}$
  - $\mathrm{let}\ S \in \mathit{Wait}$
  - $\mathit{Wait} = \mathit{Wait} \setminus \{S\}$
  - $p := \max(p, \max_c(S))$
  - $S := S \wedge \neg\varphi$
  - $\underline{\textbf{foreach}}\ S' : S \overset{a}{\Rightarrow} S'\ \underline{\textbf{do}}$
    - $S' := \mathrm{delay}(S', \neg\varphi)$
    - $(S' := \mathrm{extrapolate}(S')^{\ddagger})$
    - $\underline{\textbf{if}}\ \forall S'' \in \mathit{Passed} : S' \not\subseteq S''$
      - $\underline{\textbf{then}}\ \mathit{Wait} := \mathit{Wait} \cup \{S'\}$
    - $\underline{\textbf{fi}}$
  - $\underline{\textbf{od}}$
  - $\mathit{Passed} := \mathit{Passed} \cup \{S\}$
- $\underline{\textbf{od}}$
- $\underline{\textbf{exit}}(p)$

$\underline{\textbf{end}}$



Reduction to Reachability

$$
\begin{array}{l}
\underline{\textbf{proc}}\ Reachable(S_0, \varphi)\ \equiv \\
\quad pre(S_0 \models A\Diamond\varphi) \\
\quad Wait := \{delay(S_0[c \mapsto 0], \neg\varphi)\} \\
\quad Passed := \emptyset \\
\quad p := 0 \\
\quad \underline{\textbf{while}}\ Wait \neq \emptyset\ \underline{\textbf{do}} \\
\qquad\quad \text{let}\ S \in Wait \\
\qquad\quad Wait = Wait \setminus \{S\} \\
\qquad\quad p := \max(p, \max_c(S)) \\
\qquad\quad S := S \wedge \neg\varphi \\
\qquad\quad \underline{\textbf{foreach}}\ S' : S \overset{a}{\Rightarrow} S'\ \underline{\textbf{do}} \\
\qquad\qquad S' := delay(S', \neg\varphi) \\
\qquad\qquad (S' := extrapolate(S')^{\ddagger}) \\
\qquad\qquad \underline{\textbf{if}}\ \forall S'' \in Passed : S' \not\subseteq S'' \\
\qquad\qquad\quad \underline{\textbf{then}}\ Wait := Wait \cup \{S'\} \\
\qquad\qquad \underline{\textbf{fi}} \\
\qquad\quad \underline{\textbf{od}} \\
\qquad\quad Passed := Passed \cup \{S\} \\
\quad \underline{\textbf{od}} \\
\quad \underline{\textbf{exit}}(p) \\
\underline{\textbf{end}}
\end{array}
$$



Reduction to Reachability

$\underline{\textbf{proc}}\ \mathit{Reachable}(S_0, \varphi)\ \equiv$

$\quad \mathit{pre}(S_0 \models A\Diamond\varphi)$

$\quad \mathit{Wait} := \{\mathit{delay}(S_0[c \mapsto 0], \neg\varphi)\}$

$\quad \mathit{Passed} := \emptyset$

$\quad p := 0$

$\quad \underline{\textbf{while}}\ \mathit{Wait} \neq \emptyset\ \underline{\textbf{do}}$

$\qquad\quad \mathrm{let}\ S \in \mathit{Wait}$

$\qquad\quad \mathit{Wait} = \mathit{Wait} \setminus \{S\}$

$\qquad\quad p := \max(p, \max_c(S))$

$\qquad\quad S := S \wedge \neg\varphi$

$\qquad\quad \underline{\textbf{foreach}}\ S' : S \overset{a}{\Rightarrow} S'\ \underline{\textbf{do}}$

$\qquad\qquad S' := \mathrm{delay}(S', \neg\varphi)$

$\qquad\qquad (S' := \mathrm{extrapolate}(S')^{\ddagger})$

$\qquad\qquad \underline{\textbf{if}}\ \forall S'' \in \mathit{Passed} : S' \not\subseteq S''$

$\qquad\qquad\quad \underline{\textbf{then}}\ \mathit{Wait} := \mathit{Wait} \cup \{S'\}$

$\qquad\qquad \underline{\textbf{fi}}$

$\qquad\quad \underline{\textbf{od}}$

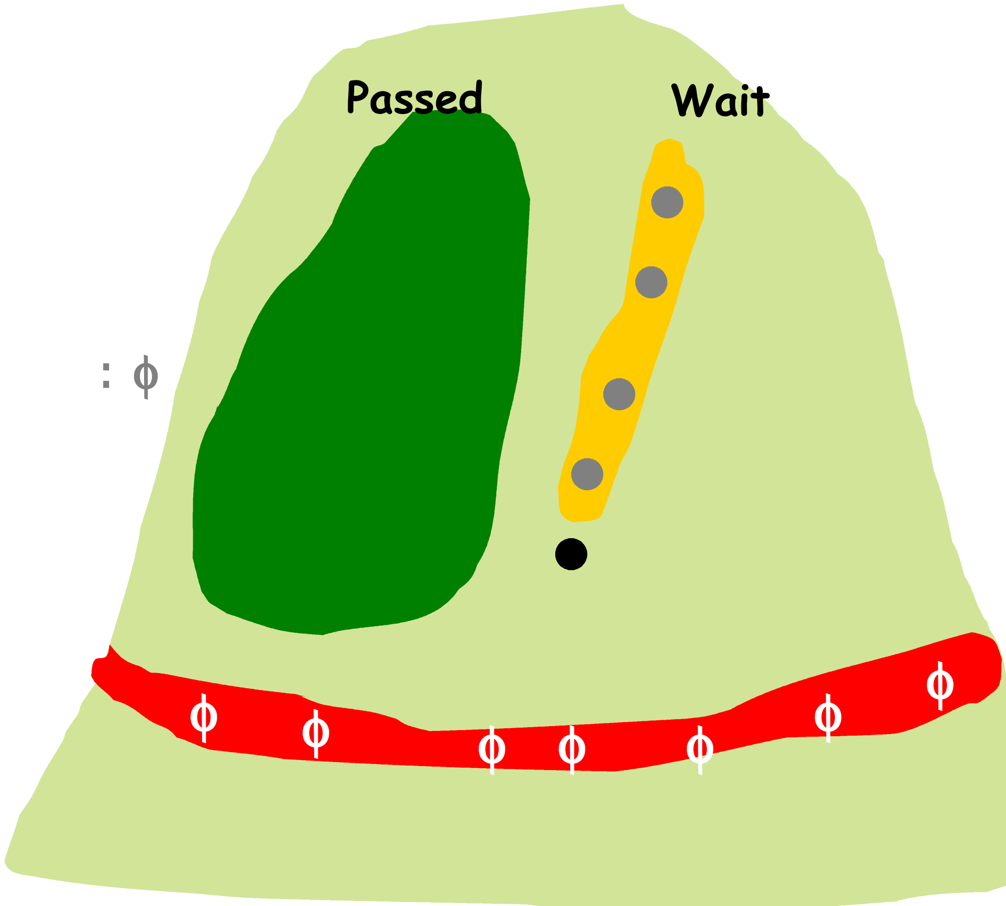
$\qquad\quad \mathit{Passed} := \mathit{Passed} \cup \{S\}$

$\quad \underline{\textbf{od}}$

$\quad \underline{\textbf{exit}}(p)$

$\underline{\textbf{end}}$

: ϕ



$p := \max(p, \max_c(S))$

Reduction to Reachability

proc $Reachable(S_0, \varphi) \equiv$

  $pre(S_0 \models A\Diamond\varphi)$

  $Wait := \{delay(S_0[c \mapsto 0], \neg\varphi)\}$

  $Passed := \emptyset$

  $p := 0$

  while $Wait \neq \emptyset$ **do**

      let $S \in Wait$

      $Wait = Wait \setminus \{S\}$

      $p := \max(p, \max_c(S))$

      $S := S \wedge \neg\varphi$

      foreach $S' : S \overset{a}{\Rightarrow} S'$ **do**

        $S' := delay(S', \neg\varphi)$

        $(S' := extrapolate(S')^{\ddagger})$

        if $\forall S'' \in Passed : S' \not\subseteq S''$

          then $Wait := Wait \cup \{S'\}$

        fi

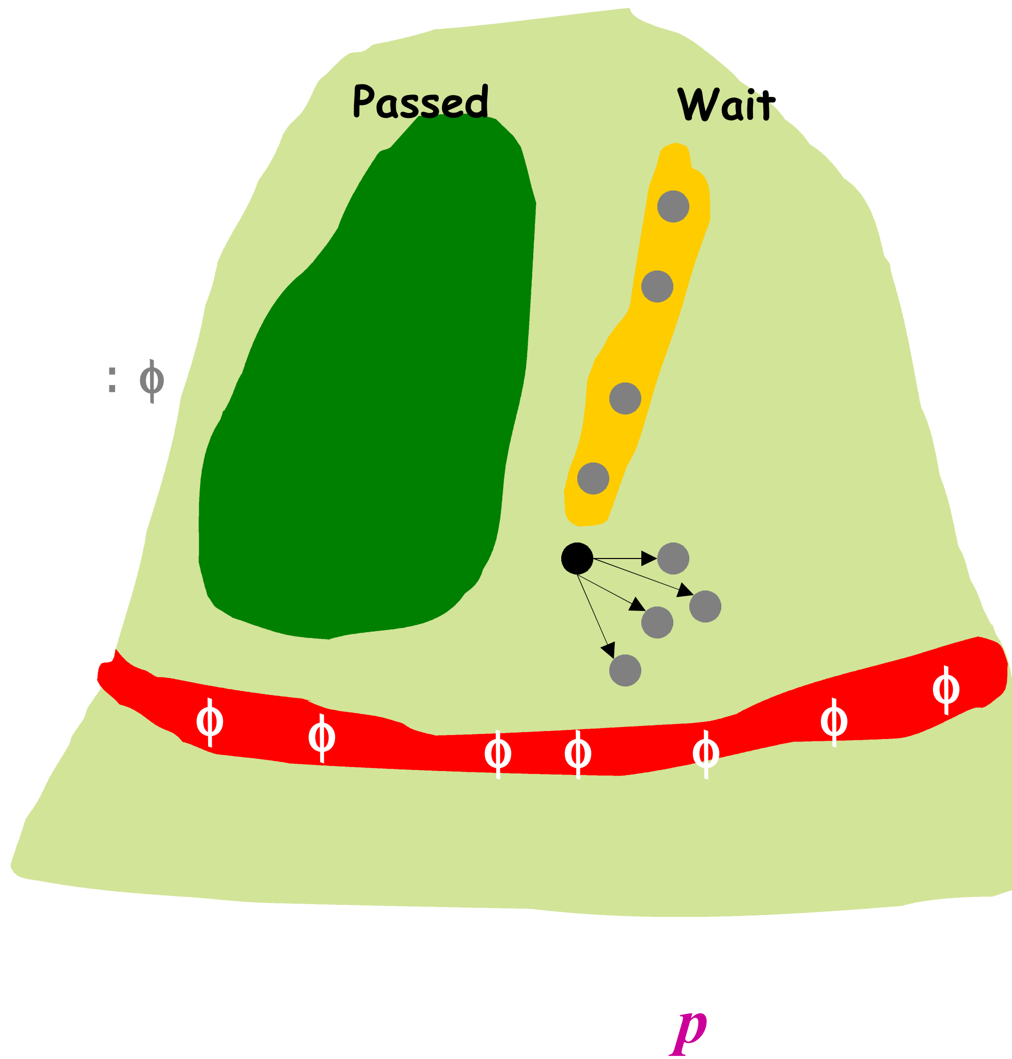
      od

      $Passed := Passed \cup \{S\}$

  od

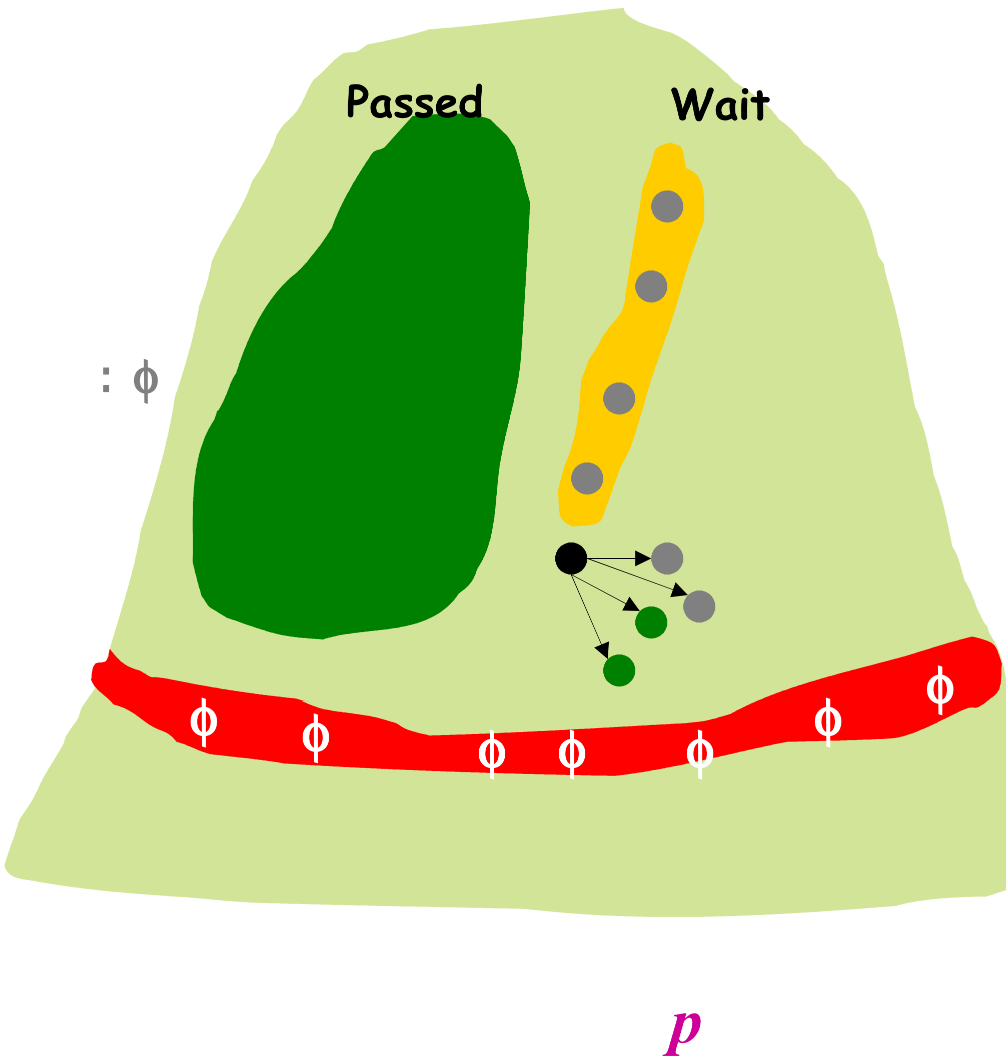
  exit(p)

end



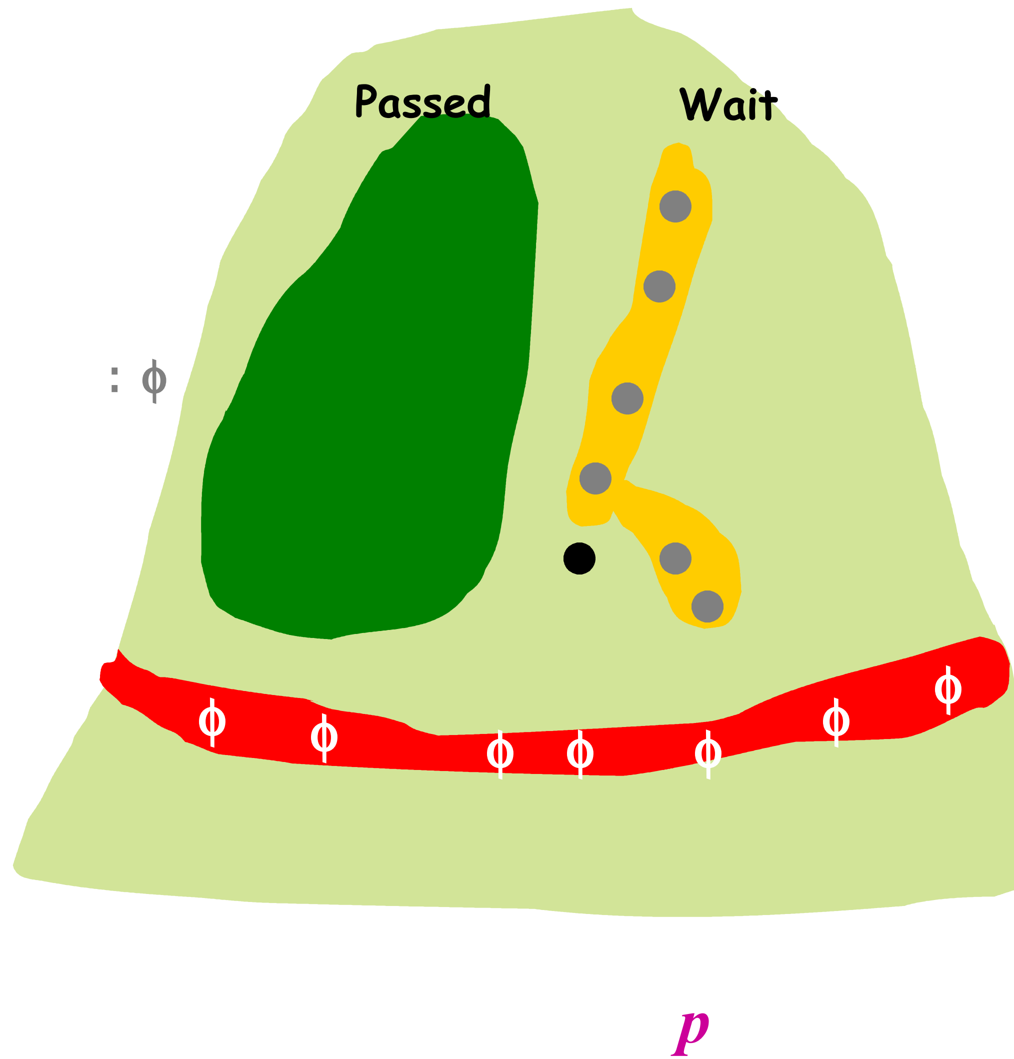
Reduction to Reachability

$\underline{\textbf{proc}}\ Reachable(S_0, \varphi)\ \equiv$
$\quad pre(S_0 \models A\Diamond\varphi)$
$\quad Wait := \{delay(S_0[c \mapsto 0], \neg\varphi)\}$
$\quad Passed := \emptyset$
$\quad p := 0$
$\quad \underline{\textbf{while}}\ Wait \neq \emptyset\ \underline{\textbf{do}}$
$\qquad \text{let } S \in Wait$
$\qquad Wait = Wait \setminus \{S\}$
$\qquad p := \max(p, \max_c(S))$
$\qquad S := S \wedge \neg\varphi$
$\qquad \underline{\textbf{foreach}}\ S' : S \stackrel{a}{\Rightarrow} S'\ \underline{\textbf{do}}$
$\qquad\quad S' := delay(S', \neg\varphi)$
$\qquad\quad (S' := extrapolate(S')^{\ddagger})$
$\qquad\quad \underline{\textbf{if}}\ \forall S'' \in Passed : S' \not\subseteq S''$
$\qquad\qquad \underline{\textbf{then}}\ Wait := Wait \cup \{S'\}$
$\qquad\quad \underline{\textbf{fi}}$
$\qquad \underline{\textbf{od}}$
$\qquad Passed := Passed \cup \{S\}$
$\quad \underline{\textbf{od}}$
$\quad \underline{\textbf{exit}}(p)$
$\underline{\textbf{end}}$



Reduction to Reachability

proc $Reachable(S_0, \varphi) \equiv$
$pre(S_0 \models A\Diamond\varphi)$
$Wait := \{delay(S_0[c \mapsto 0], \neg\varphi)\}$
$Passed := \emptyset$
$p := 0$
while $Wait \neq \emptyset$ **do**
let $S \in Wait$
$Wait = Wait \setminus \{S\}$
$p := \max(p, \max_c(S))$
$S := S \wedge \neg\varphi$
foreach $S' : S \overset{a}{\Rightarrow} S'$ **do**
$S' := \mathrm{delay}(S', \neg\varphi)$
$(S' := \mathrm{extrapolate}(S')^{\ddagger})$
if $\forall S'' \in Passed : S' \not\subseteq S''$
then $Wait := Wait \cup \{S'\}$
fi
od
$Passed := Passed \cup \{S\}$
od
exit(p)
end



Reduction to Reachability

proc $Reachable(S_0, \varphi) \equiv$

  $pre(S_0 \models A\Diamond\varphi)$

  $Wait := \{delay(S_0[c \mapsto 0], \neg\varphi)\}$

  $Passed := \emptyset$

  $p := 0$

  while $Wait \neq \emptyset$ **do**

    let $S \in Wait$

    $Wait = Wait \setminus \{S\}$

    $p := \max(p, \max_c(S))$

    $S := S \wedge \neg\varphi$

    foreach $S' : S \overset{a}{\Rightarrow} S'$ **do**

      $S' := delay(S', \neg\varphi)$

      $(S' := extrapolate(S')^{\ddagger})$

      if $\forall S'' \in Passed : S' \not\subseteq S''$

        then $Wait := Wait \cup \{S'\}$

      fi

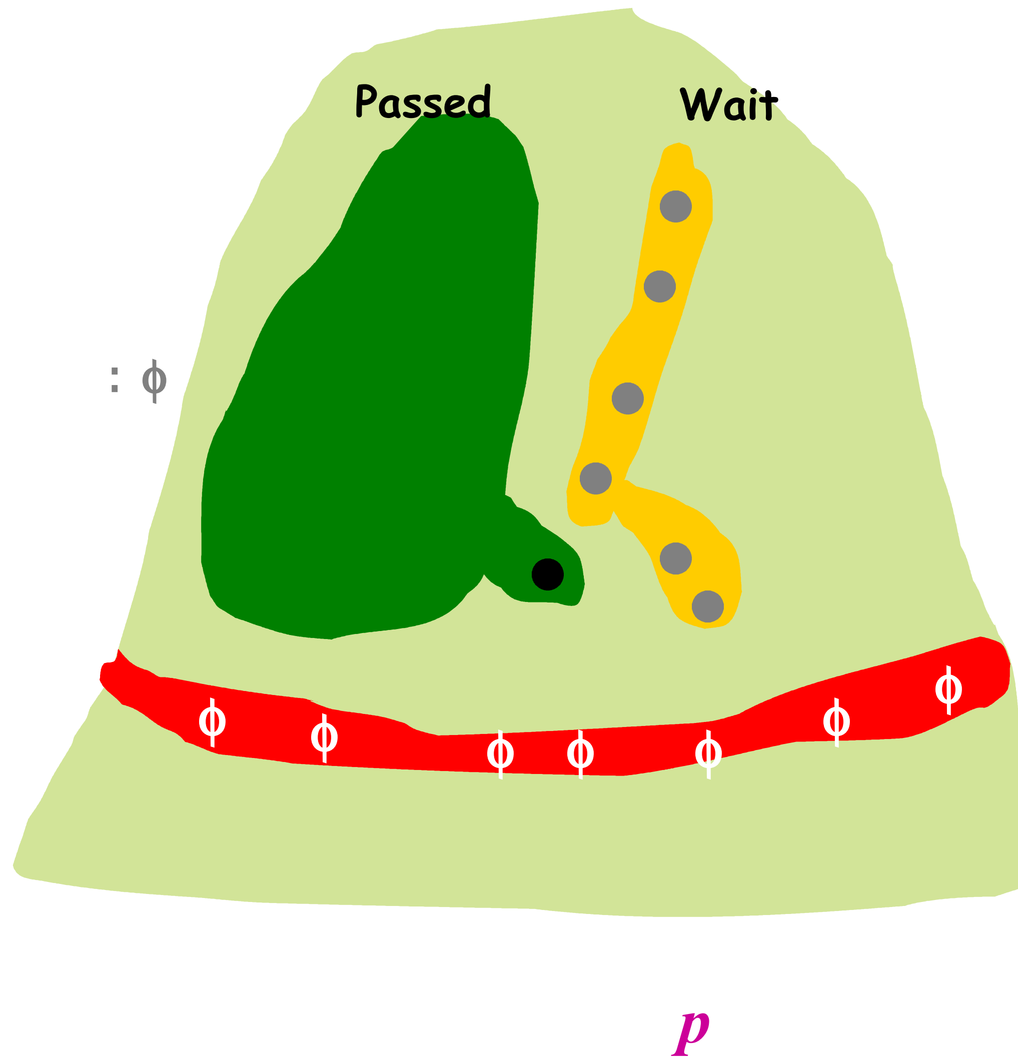
    od

    $Passed := Passed \cup \{S\}$

  od

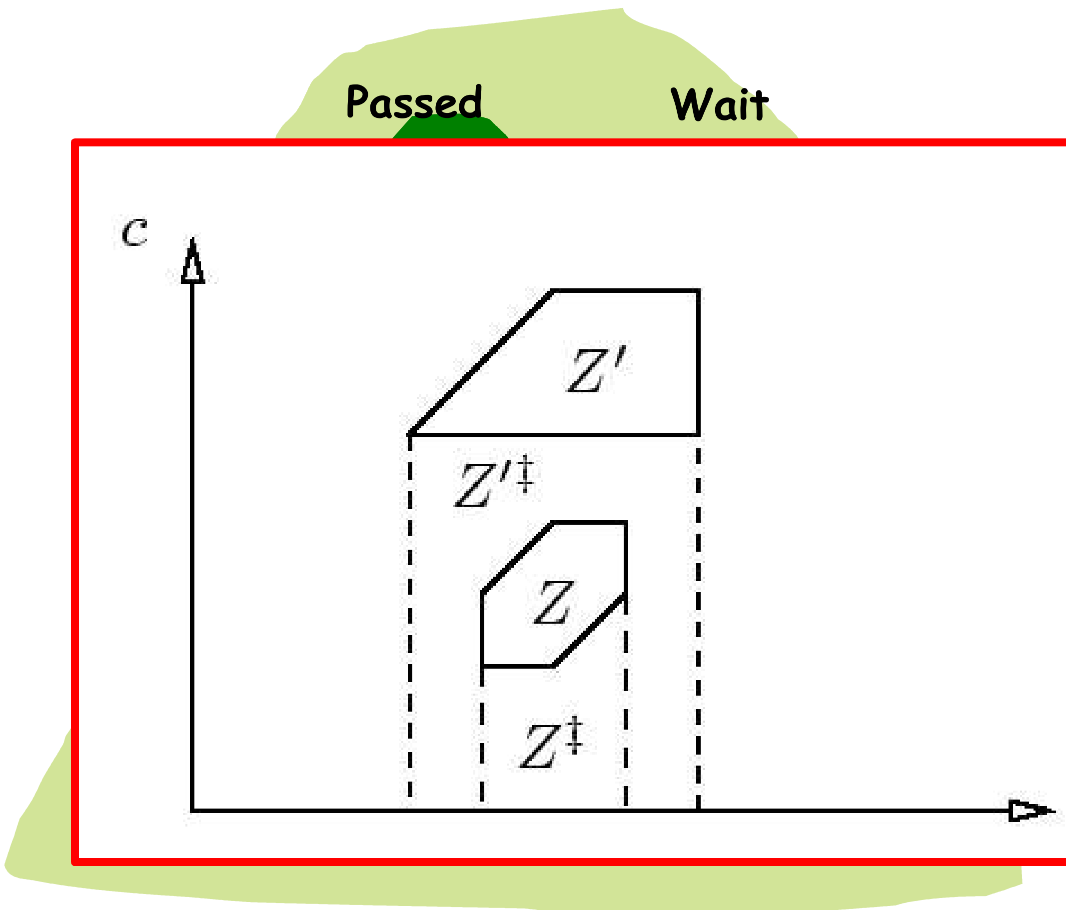
  exit(p)

end



Reduction to Reachability

proc $Reachable(S_0, \varphi) \;\equiv$
  $pre(S_0 \models A\Diamond\varphi)$
  $Wait := \{delay(S_0[c \mapsto 0], \neg\varphi)\}$
  $Passed := \emptyset$
  $p := 0$
  while $Wait \neq \emptyset$ **do**
    let $S \in Wait$
    $Wait = Wait \setminus \{S\}$
    $p := \max(p, \max_c(S))$
    $S := S \wedge \neg\varphi$
    foreach $S' : S \stackrel{a}{\Rightarrow} S'$ **do**
      $S' := delay(S', \neg\varphi)$
      $(S' := \mathrm{extrapolate}(S')^{\ddagger})$
      if $\forall S'' \in Passed : S' \not\subseteq S''$
        then $Wait := Wait \cup \{S'\}$
      fi
    od
    $Passed := Passed \cup \{S\}$
  od
  exit(p)
end



p

Outline of Talk

- Simulation Graph for Timed Automata

- Reduction to Reachability Analysis
- Parameterized **Liveness Analysis**
- Experimental Results

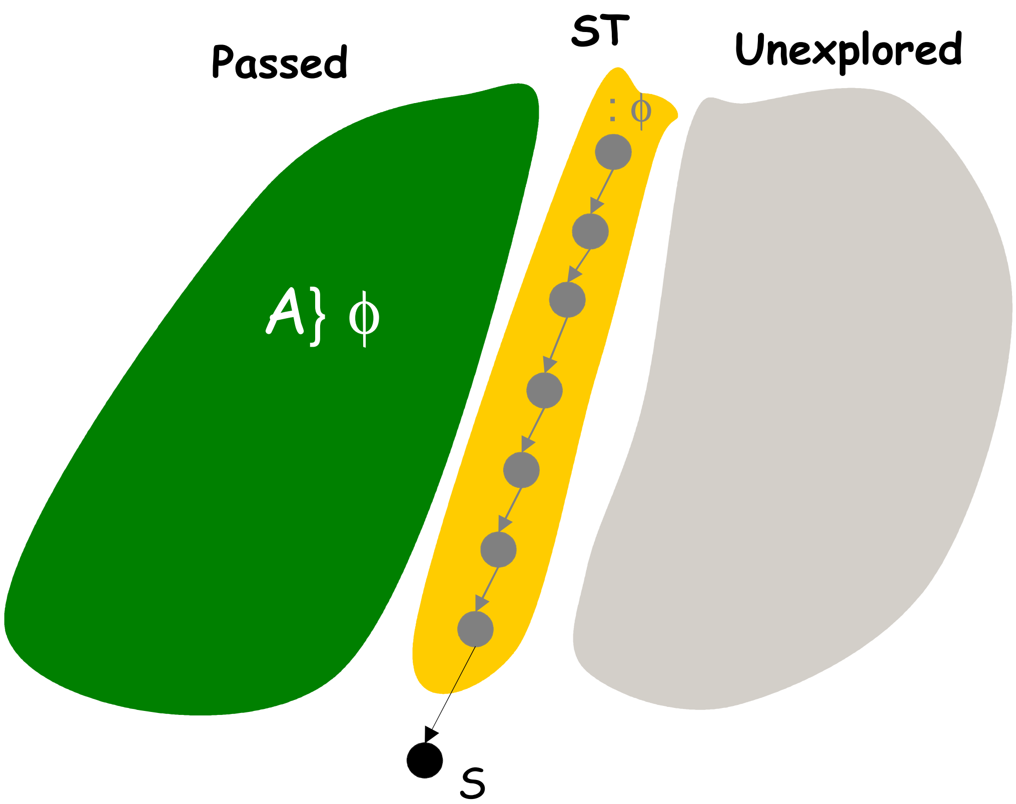
- Extensions
  - Priced Timed Automata
    (Worst Cost Execution)
  - Timed Games
    (Time-optimal Winning Strategies)
- Conclusions

Liveness Algorithm

Bouajjani, Tripakis, Yovine, 97

$\underline{\textbf{proc}}\ Eventually(S_0, \varphi)\ \equiv$
$\quad ST := \emptyset$
$\quad Passed := \emptyset$
$\quad Search(delay(S_0, \neg\varphi))$
$\quad \underline{\textbf{exit}}(true)$
$\underline{\textbf{end}}$

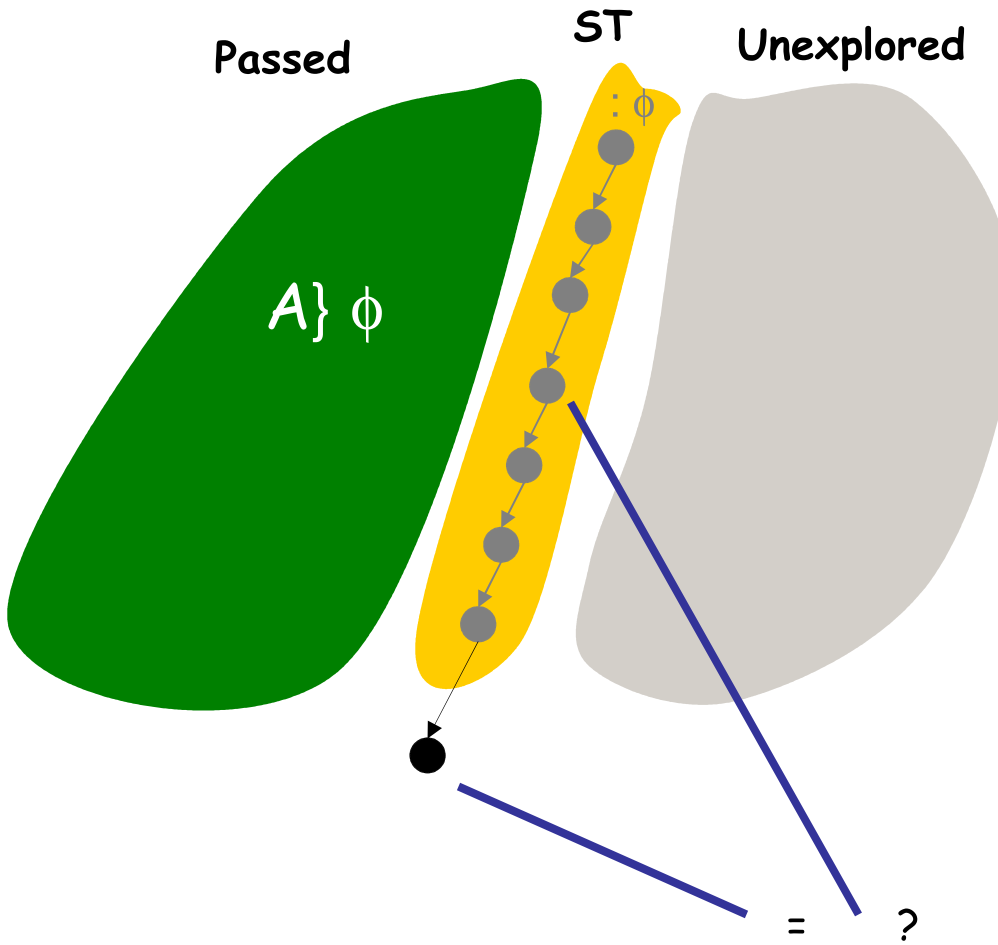
$\underline{\textbf{proc}}\ Search(S)\ \equiv$
$\quad \underline{\textbf{if}}\ loop(S, ST)\ \underline{\textbf{then}}\ \underline{\textbf{exit}}(false)\ \underline{\textbf{fi}}$
$\quad S := S \wedge \neg\varphi$
$\quad push(ST, S)$
$\quad \underline{\textbf{if}}\ unbounded(S) \vee deadlocked(S)\ \underline{\textbf{then}}$
$\qquad\qquad \underline{\textbf{exit}}(false)\ \underline{\textbf{fi}}$
$\quad \underline{\textbf{if}}\ \forall S' \in Passed : S \not\subseteq S'$
$\qquad \underline{\textbf{then}}\ \underline{\textbf{foreach}}\ S' : S \stackrel{a}{\Rightarrow} S'\ \underline{\textbf{do}}$
$\qquad\qquad Search(delay(S', \neg\varphi))$
$\qquad\quad \underline{\textbf{od}}$
$\quad \underline{\textbf{fi}}$
$\quad Passed := Passed \cup \{pop(ST)\}$
$\underline{\textbf{end}}$



Liveness Algorithm

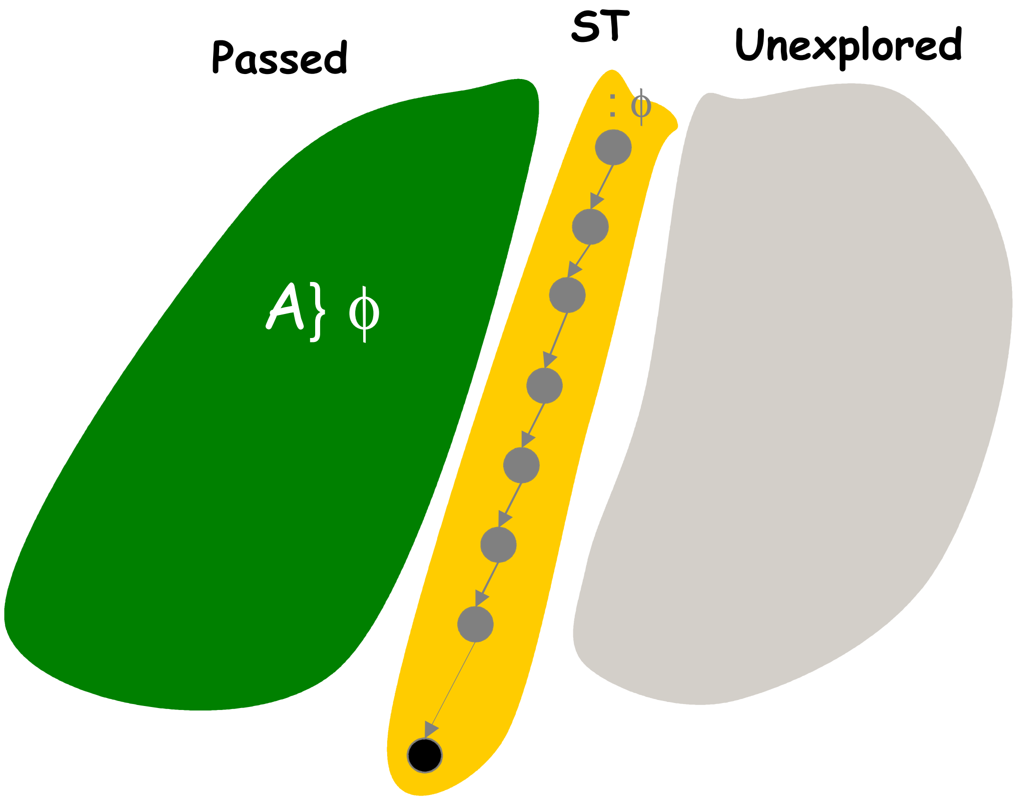
proc $Eventually(S_0, \varphi) \equiv$
  $ST := \emptyset$
  $Passed := \emptyset$
  $Search(delay(S_0, \neg\varphi))$
  exit$(true)$
end

proc $Search(S) \equiv$
  if $loop(S, ST)$ **then** **exit**$(false)$ **fi**
  $S := S \wedge \neg\varphi$
  $push(ST, S)$
  if $unbounded(S) \vee deadlocked(S)$ **then**
      exit$(false)$ **fi**
  if $\forall S' \in Passed : S \not\subseteq S'$
    then **foreach** $S' : S \overset{a}{\Rightarrow} S'$ **do**
      $Search(delay(S', \neg\varphi))$
    od
  fi
  $Passed := Passed \cup \{pop(ST)\}$
end



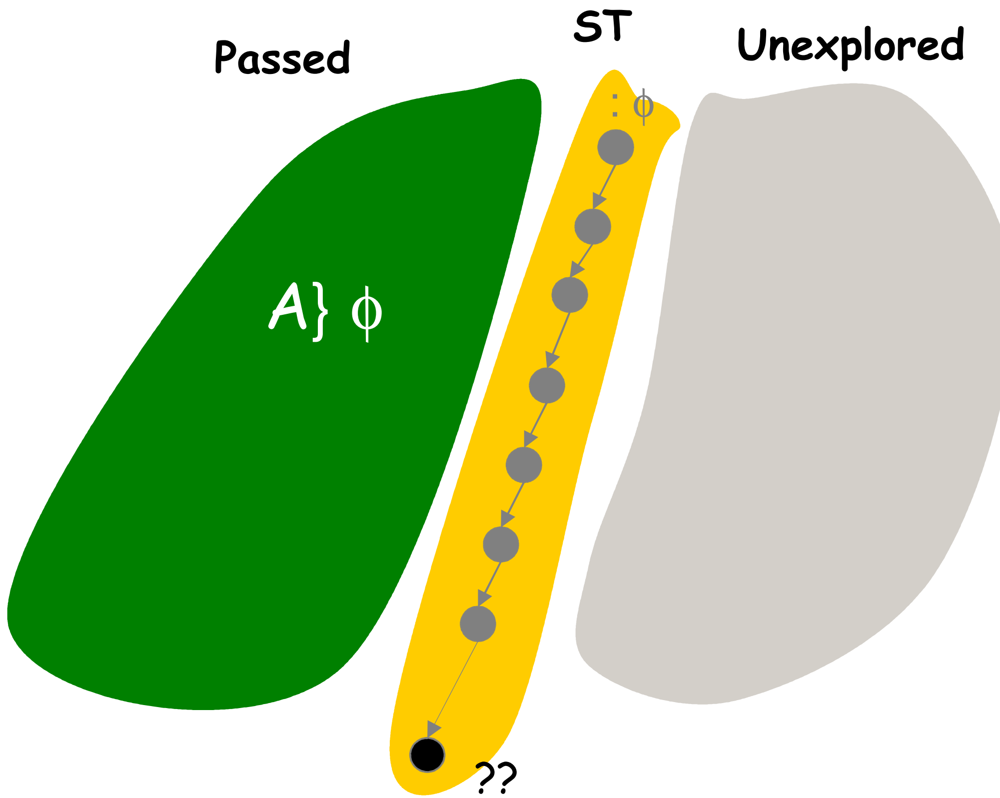
Liveness Algorithm

proc $Eventually(S_0, \varphi) \equiv$
  $ST := \emptyset$
  $Passed := \emptyset$
  $Search(delay(S_0, \neg\varphi))$
  exit$(true)$
end
proc $Search(S) \equiv$
  if $loop(S, ST)$ **then** **exit**$(false)$ **fi**
  $S := S \wedge \neg\varphi$
  $push(ST, S)$
  if $unbounded(S) \vee deadlocked(S)$ **then**
      exit$(false)$ **fi**
  if $\forall S' \in Passed : S \not\subseteq S'$
    then **foreach** $S' : S \stackrel{a}{\Rightarrow} S'$ **do**
        $Search(delay(S', \neg\varphi))$
      od
  fi
  $Passed := Passed \cup \{pop(ST)\}$
end



Liveness Algorithm

$\underline{\textbf{proc}}\ \mathit{Eventually}(S_0, \varphi)\ \equiv$
$\quad ST := \emptyset$
$\quad \mathit{Passed} := \emptyset$
$\quad \mathit{Search}(\mathit{delay}(S_0, \neg\varphi))$
$\quad \underline{\textbf{exit}}(\mathit{true})$
$\underline{\textbf{end}}$
$\underline{\textbf{proc}}\ \mathit{Search}(S)\ \equiv$
$\quad \underline{\textbf{if}}\ \mathit{loop}(S, ST)\ \underline{\textbf{then}}\ \underline{\textbf{exit}}(\mathit{false})\ \underline{\textbf{fi}}$
$\quad S := S \wedge \neg\varphi$
$\quad \mathit{push}(ST, S)$
$\quad \underline{\textbf{if}}\ \mathit{unbounded}(S) \vee \mathit{deadlocked}(S)\ \underline{\textbf{then}}$
$\qquad\qquad \underline{\textbf{exit}}(\mathit{false})\ \underline{\textbf{fi}}$
$\quad \underline{\textbf{if}}\ \forall S' \in \mathit{Passed} : S \not\subseteq S'$
$\qquad \underline{\textbf{then}}\ \underline{\textbf{foreach}}\ S' : S \stackrel{a}{\Rightarrow} S'\ \underline{\textbf{do}}$
$\qquad\qquad \mathit{Search}(\mathit{delay}(S', \neg\varphi))$
$\qquad\quad \underline{\textbf{od}}$
$\quad \underline{\textbf{fi}}$
$\quad \mathit{Passed} := \mathit{Passed} \cup \{\mathit{pop}(ST)\}$
$\underline{\textbf{end}}$



Liveness Algorithm

proc $Eventually(S_0, \varphi) \equiv$

$ST := \emptyset$

$Passed := \emptyset$

$Search(delay(S_0, \neg\varphi))$

exit$(true)$

end

proc $Search(S) \equiv$

if $loop(S, ST)$ **then** **exit**$(false)$ **fi**

$S := S \wedge \neg\varphi$

$push(ST, S)$

if $unbounded(S) \vee deadlocked(S)$ **then** **exit**$(false)$ **fi**

if $\forall S' \in Passed : S \not\subseteq S'$

then **foreach** $S' : S \overset{a}{\Rightarrow} S'$ **do**

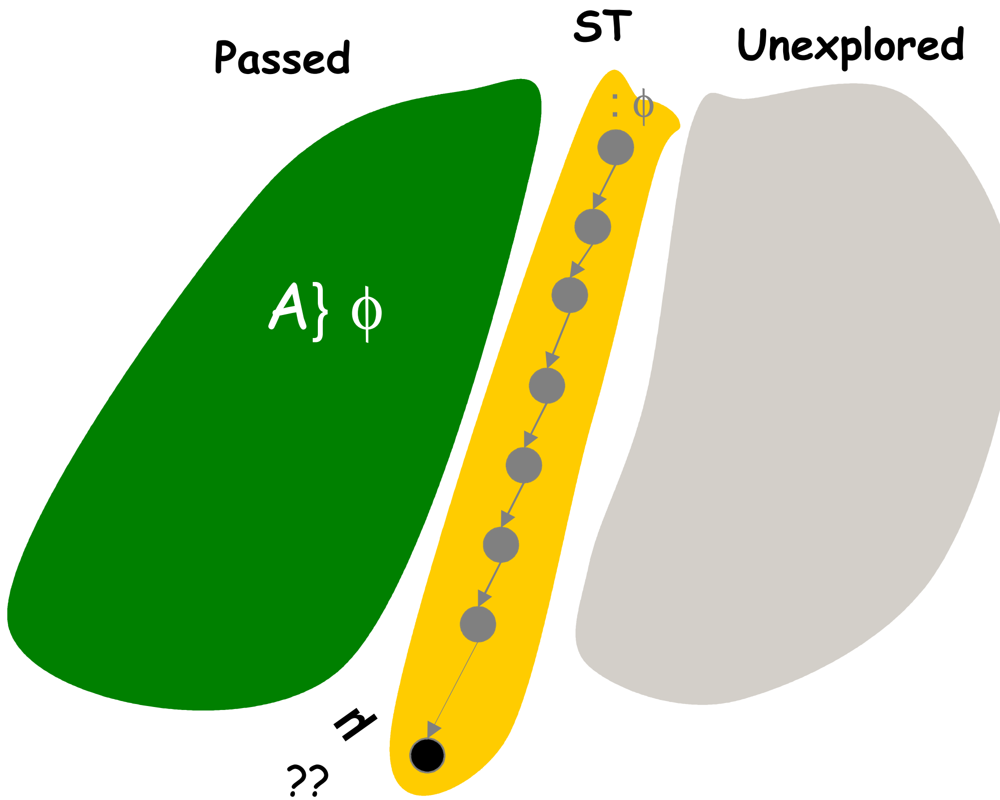
$Search(delay(S', \neg\varphi))$

od

fi

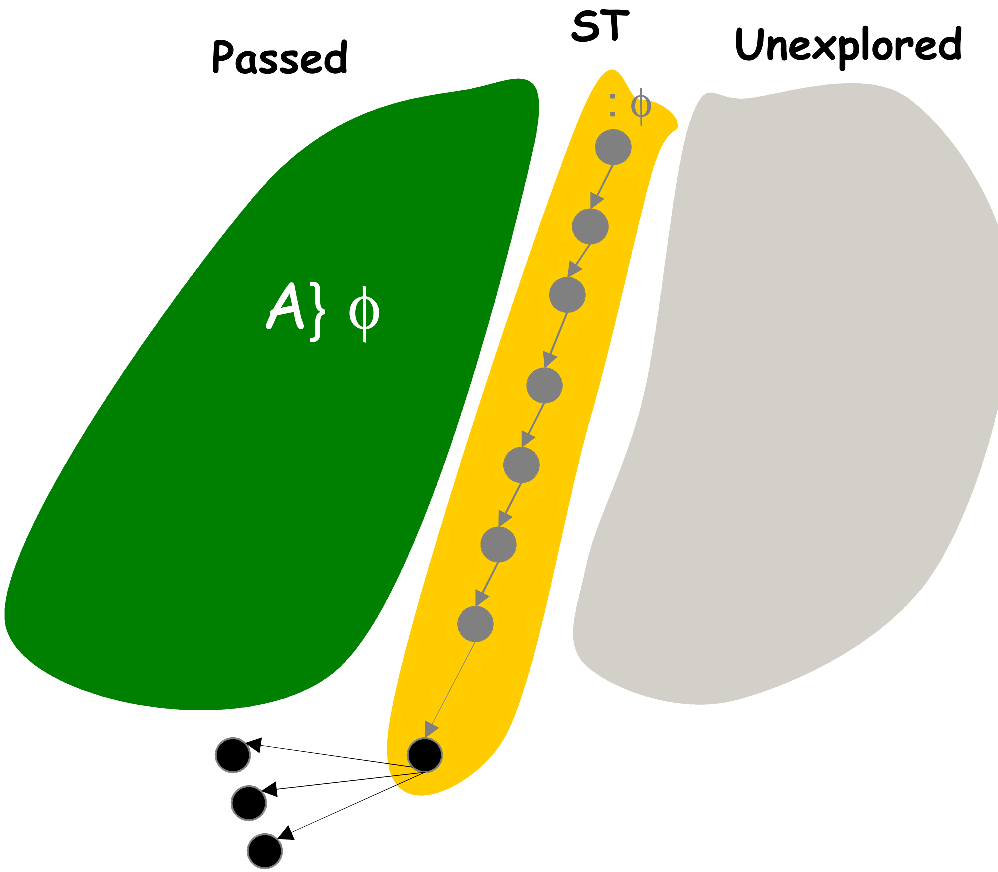
$Passed := Passed \cup \{pop(ST)\}$

end



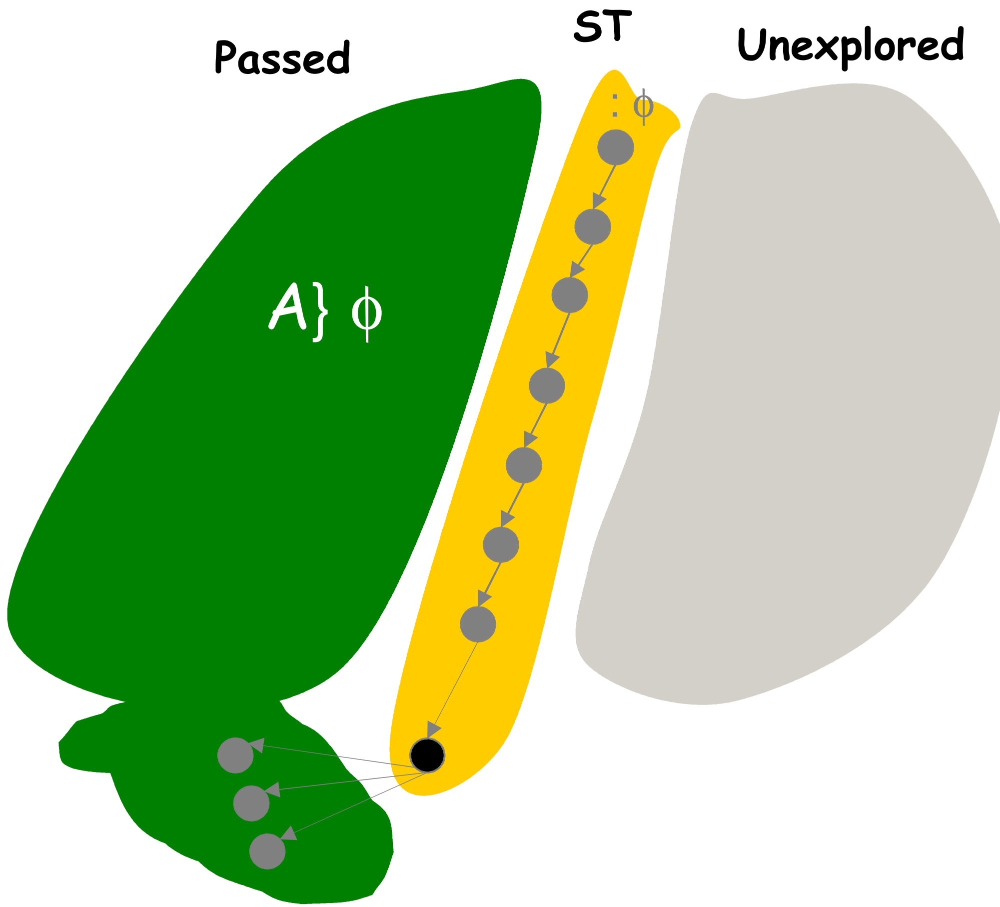
Liveness Algorithm

```
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
```



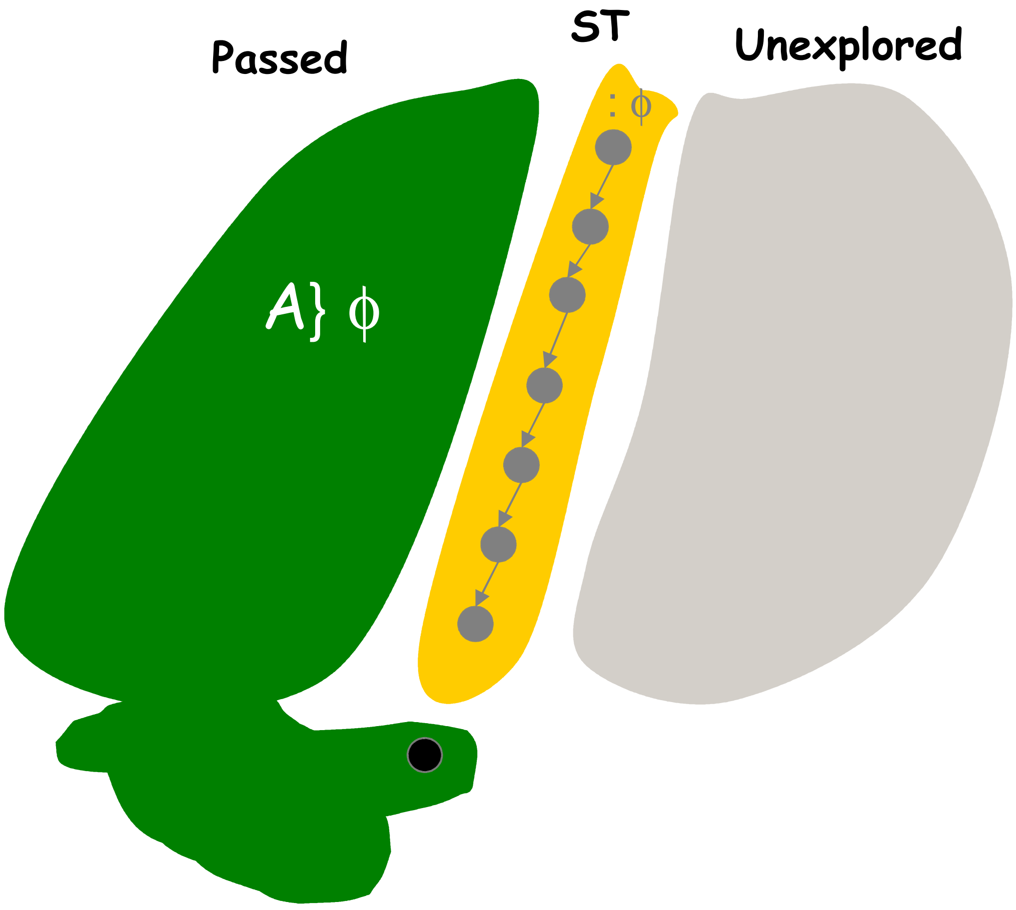
Liveness Algorithm

proc $Eventually(S_0, \varphi) \equiv$
  $ST := \emptyset$
  $Passed := \emptyset$
  $Search(delay(S_0, \neg\varphi))$
  exit$(true)$
end
proc $Search(S) \equiv$
  if $loop(S, ST)$ **then** **exit**$(false)$ **fi**
  $S := S \wedge \neg\varphi$
  $push(ST, S)$
  if $unbounded(S) \vee deadlocked(S)$ **then**
      exit$(false)$ **fi**
  if $\forall S' \in Passed : S \not\subseteq S'$
    then **foreach** $S' : S \overset{a}{\Rightarrow} S'$ **do**
        $Search(delay(S', \neg\varphi))$
      od
  fi
  $Passed := Passed \cup \{pop(ST)\}$
end



Liveness Algorithm

proc $Eventually(S_0, \varphi) \equiv$
$\quad ST := \emptyset$
$\quad Passed := \emptyset$
$\quad Search(delay(S_0, \neg\varphi))$
$\quad$ **exit**$(true)$
end
proc $Search(S) \equiv$
$\quad$ **if** $loop(S, ST)$ **then** **exit**$(false)$ **fi**
$\quad S := S \wedge \neg\varphi$
$\quad push(ST, S)$
$\quad$ **if** $unbounded(S) \vee deadlocked(S)$ **then**
$\qquad\qquad$ **exit**$(false)$ **fi**
$\quad$ **if** $\forall S' \in Passed : S \not\subseteq S'$
$\qquad$ **then** **foreach** $S' : S \overset{a}{\Rightarrow} S'$ **do**
$\qquad\qquad Search(delay(S', \neg\varphi))$
$\qquad\quad$ **od**
$\quad$ **fi**
● $\quad Passed := Passed \cup \{pop(ST)\}$
end



Outline of Talk

- Simulation Graph for Timed Automata

- Reduction to Reachability Analysis
- Parameterized Liveness Analysis
- Experimental Results

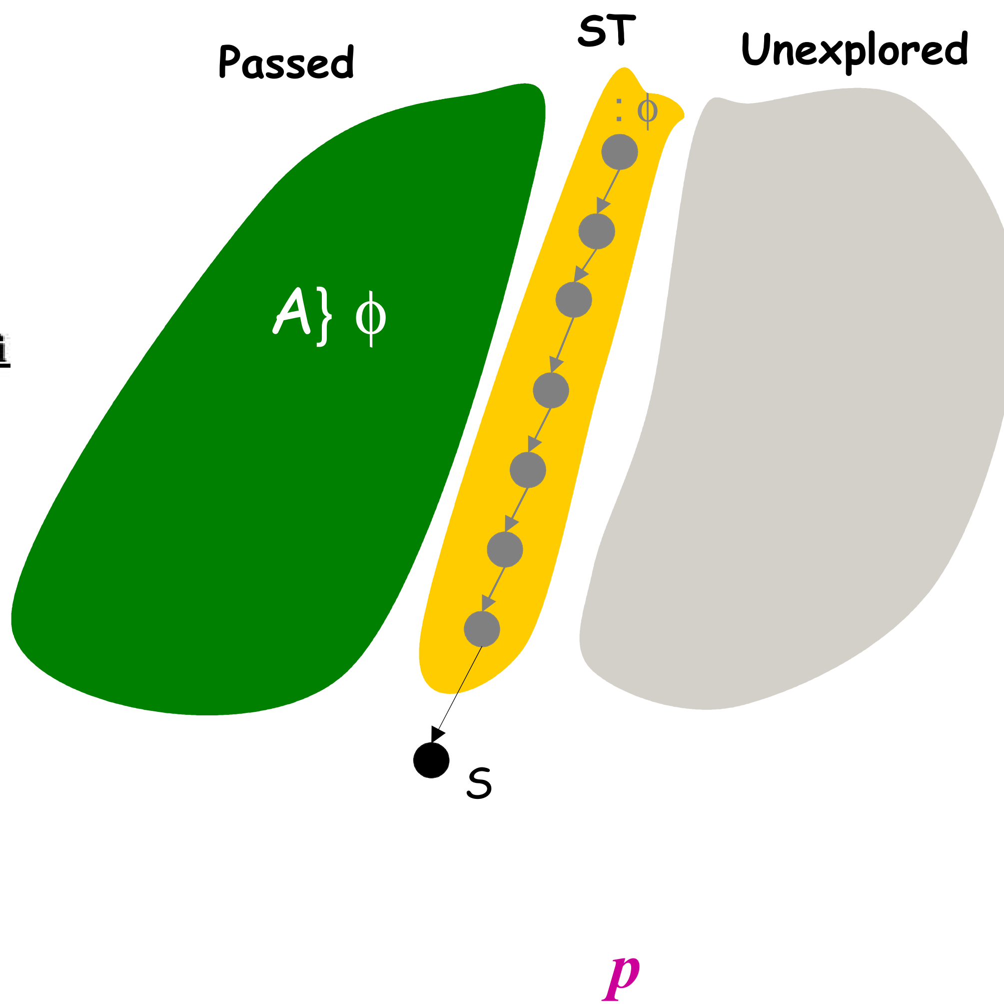
- Extensions
  - Priced Timed Automata
    (Worst Cost Execution)
  - Timed Games
    (Time-optimal Winning Strategies)
- Conclusions

Parameterized Liveness Algorithm

proc *Eventually'*$(S_0, \varphi) \;\equiv$
  $p := 0$
  $ST := \emptyset$
  $Passed := \emptyset$
  Search'$(delay(S_0[c \mapsto 0], \neg\varphi))$
  exit(p)
end

proc *Search'*$(S) \;\equiv$
  if $loop(\pi_X(S), \pi_X(ST))$ **then** **exit**$(false)$ **fi**
  $p := \max(p, \max_c(S))$
  $push(ST, S^{\ddagger})$
  $S := S \wedge \neg\varphi$
  if $unbounded(S) \vee deadlocked(S)$
    then **exit**$(false)$ **fi**

  if $\forall S' \in Passed : S \not\subseteq S'$
    then **foreach** $S' : S \overset{a}{\Rightarrow} S'$ **do**
      Search'$(delay(S', \neg\varphi))$
    od
  fi
  $Passed := Passed \cup \{pop(ST)\}$
end



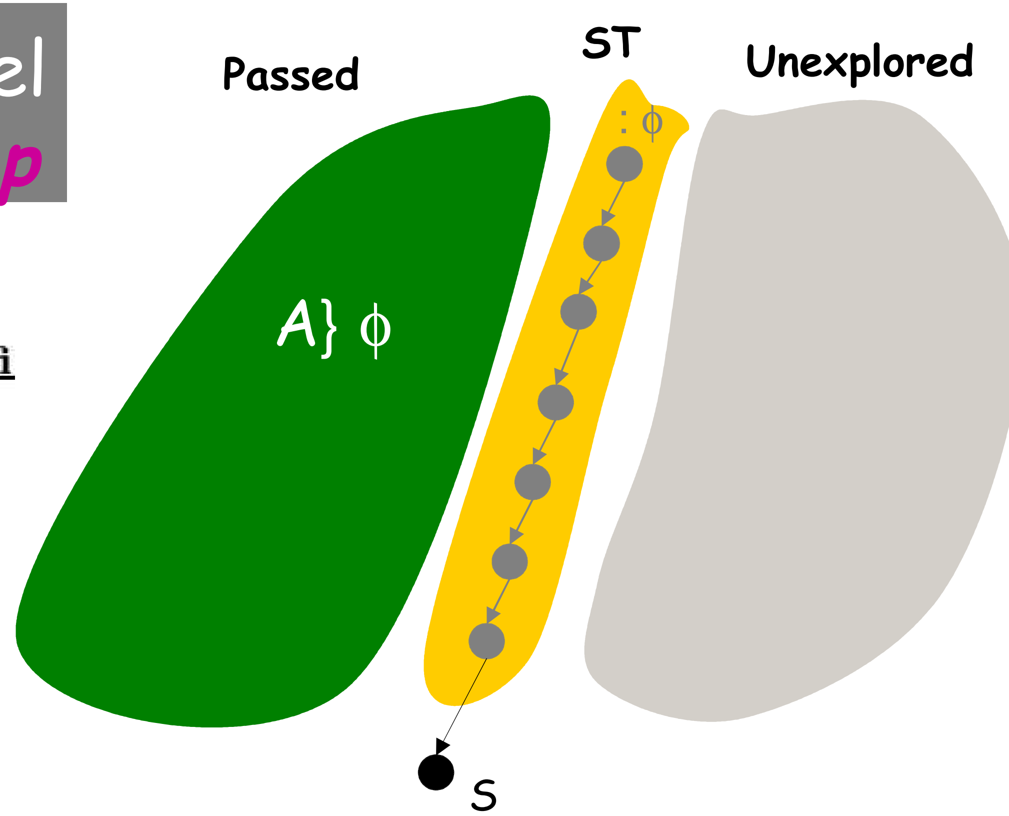
Parameterized Liveness Algorithm

proc *Eventually'*$(S_0, \varphi) \equiv$
  p :
  ST
  Pa
  Se
  exit(p)
end

Add clock c to model
Add global variable p

proc *Search'*$(S) \equiv$
  if $loop(\pi_X(S), \pi_X(ST))$ **then** **exit**$(false)$ **fi**
  $p := \max(p, \max_c(S))$
  $push(ST, S^{\ddagger})$
  $S := S \wedge \neg\varphi$
  if $unbounded(S) \vee deadlocked(S)$
      then **exit**$(false)$ **fi**

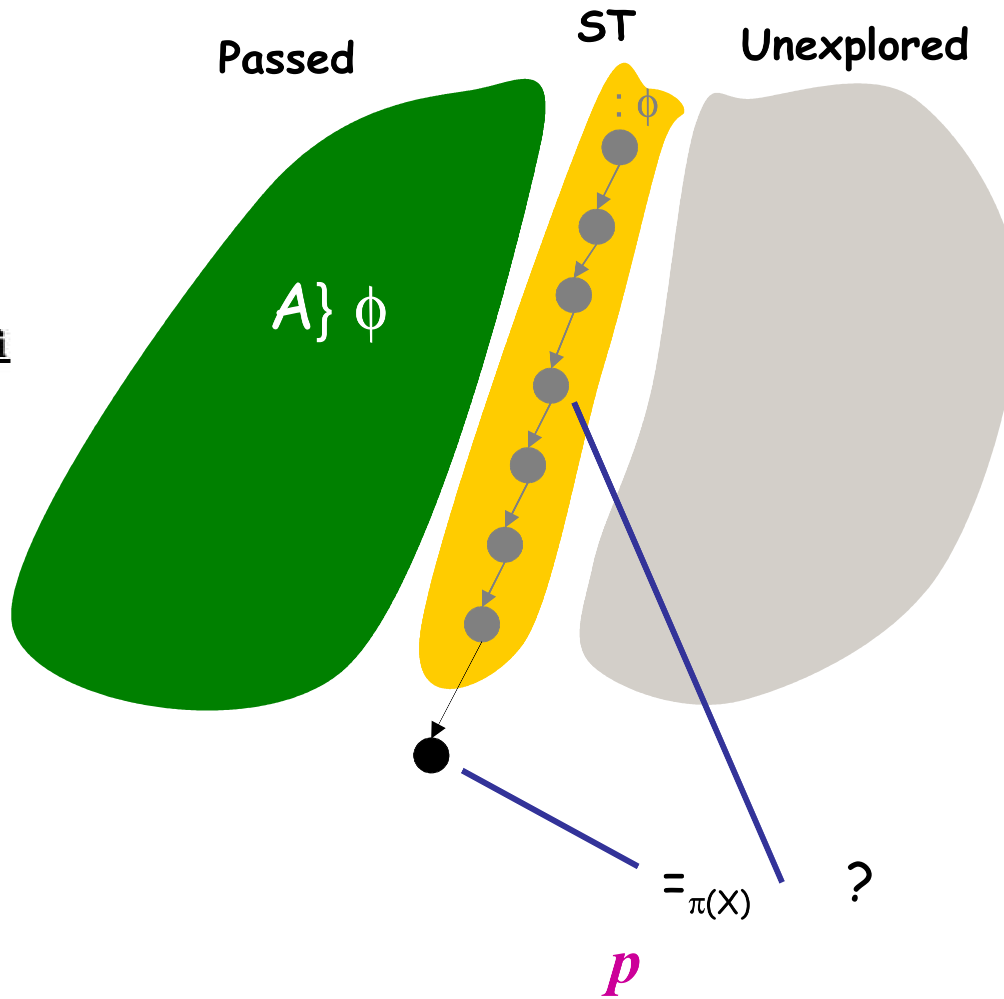
  if $\forall S' \in Passed : S \not\subseteq S'$
    then **foreach** $S' : S \overset{a}{\Rightarrow} S'$ **do**
        Search'$(delay(S', \neg\varphi))$
      od
  fi
  $Passed := Passed \cup \{pop(ST)\}$
end



p

Parameterized Liveness Algorithm

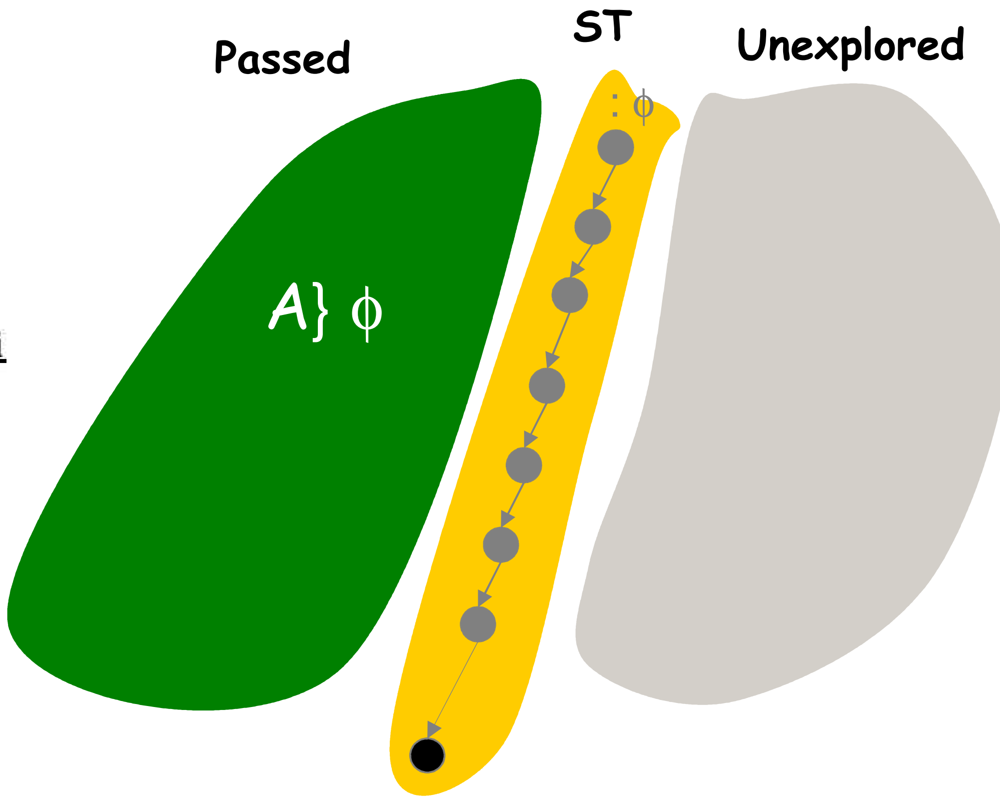
proc *Eventually'*$(S_0, \varphi) \equiv$
  $p := 0$
  $ST := \emptyset$
  $Passed := \emptyset$
  Search'$(delay(S_0[c \mapsto 0], \neg\varphi))$
  exit(p)
end
proc *Search'*$(S) \equiv$
  if $loop(\pi_X(S), \pi_X(ST))$ **then** **exit**$(false)$ **fi**
  $p := \max(p, \max_c(S))$
  $push(ST, S^{\ddagger})$
  $S := S \wedge \neg\varphi$
  if $unbounded(S) \vee deadlocked(S)$
    then **exit**$(false)$ **fi**
  if $\forall S' \in Passed : S \not\subseteq S'$
    then **foreach** $S' : S \overset{a}{\Rightarrow} S'$ **do**
      Search'$(delay(S', \neg\varphi))$
    od
  fi
  $Passed := Passed \cup \{pop(ST)\}$
end



Parameterized Liveness Algorithm

```
proc Eventually'(S₀, φ) ≡
   p := 0
   ST := ∅
   Passed := ∅
   Search'(delay(S₀[c ↦ 0], ¬φ))
   exit(p)
end
proc Search'(S) ≡
   if loop(π_X(S), π_X(ST)) then exit(false) fi
   p := max(p, max_c(S))
   push(ST, S^‡)
   S := S ∧ ¬φ
   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
```



$p{:=}\max(p, \max_c(S))$

Experimental Results

Task Graph Scheduling

	Liveness	Reachability					
Instance	$A\Diamond_{\leq p}$	$A\Diamond$	$E\Diamond$	Total	Binary	DF	w/o Extrap.
rand0000	23,1s	2.0s	0.5s	2.5s	3.5s	10.7	1.8s
rand0010	31.0s	2.8s	0.7s	3.5s	5.1s	14.0s	3.2s
rand0020	31.5s	2.4s	0.5s	2.9s	4.1s	14.9s	0.9s
rand0030	19.6s	1.4s	0.4s	1.8s	2.6s	9.1s	0.9s
rand0040	22.6s	2.0s	0.6s	2.6s	4.3s	10.4s	2.9s
rand0050	24.6s	1.5s	0.3s	1.8s	2.4s	11.4s	0.7s
rand0060	24.2s	1.6s	1.8s	3.4s	14.2s	11.3s	1.9s
rand0070	2.8	0.5s	0.6s	1.1s	4.4s	1.3s	1.3s
rand0080	29.6s	1.9s	0.4s	2.3s	3.2s	14.0s	1.0s
rand0090	20.6s	1.7s	0.4s	2.1s	2.9s	9.4s	1.2s
rand0100	17.1s	1.3s	0.3s	1.6s	2.6s	7.7s	1.3s

11 random problems. (100 Tasks, 2 Processes)

Train Gate

Trains	$\rightsquigarrow_{\leq p}$	$\rightsquigarrow$
4	0.2s	0.06s
5	0.4s	0.2s
6	3.8s	1.3s
7	25.7s	9.3s
8	268.1s	88.7s

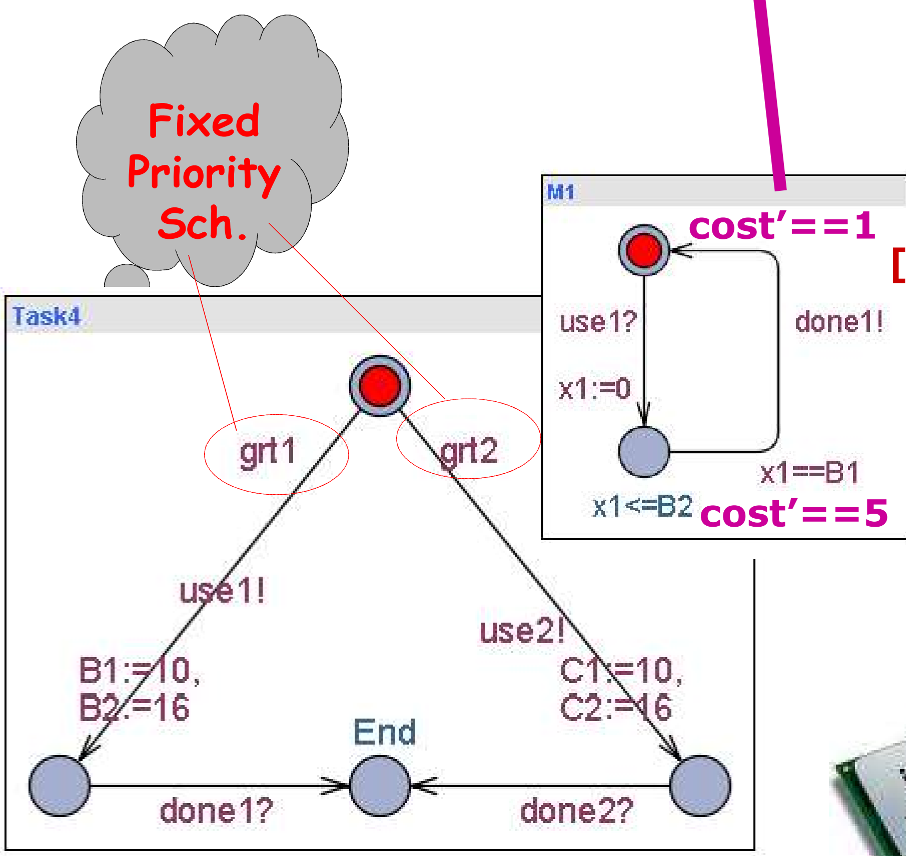
Outline of Talk

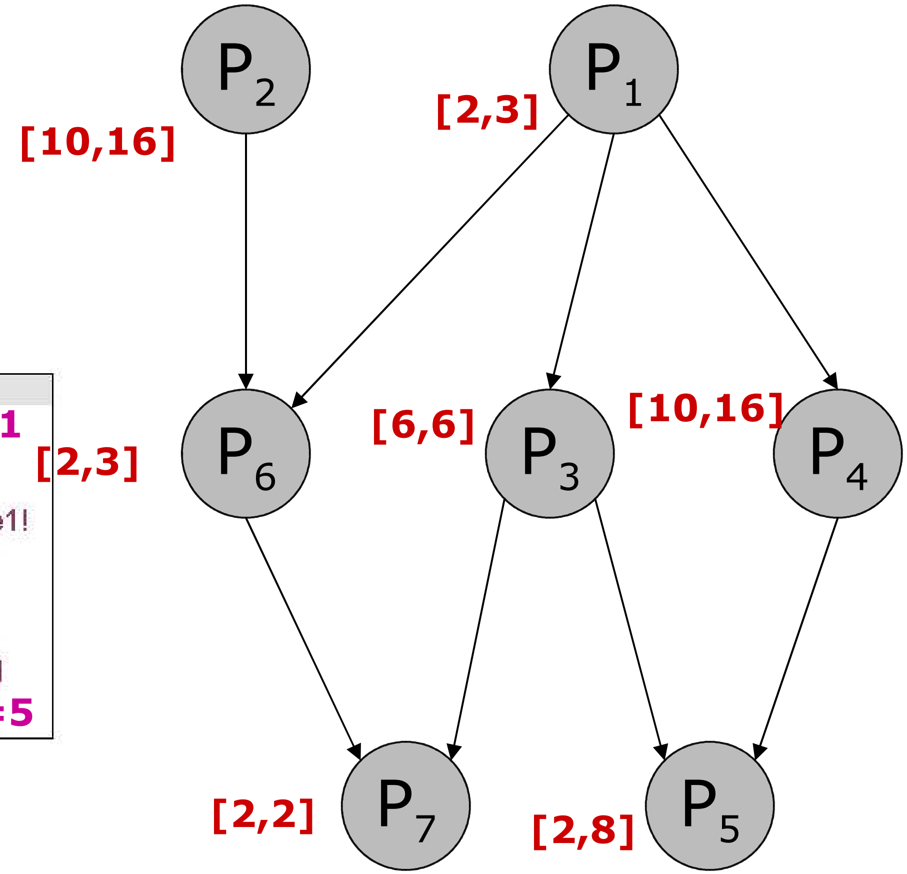
- Simulation Graph for Timed Automata
- Reduction to Reachability Analysis
- Parameterized Liveness Analysis
- Experimental Results
- Extensions
  - Priced Timed Automata (Worst Cost Execution)
  - Timed Games (Time-optimal Winning Strategies)
- Conclusions

Cost-Bounded Liveness

Task Graph Scheduling w Uncertainty,

- Energy-rates **C**:**M** ! **N**x**N**



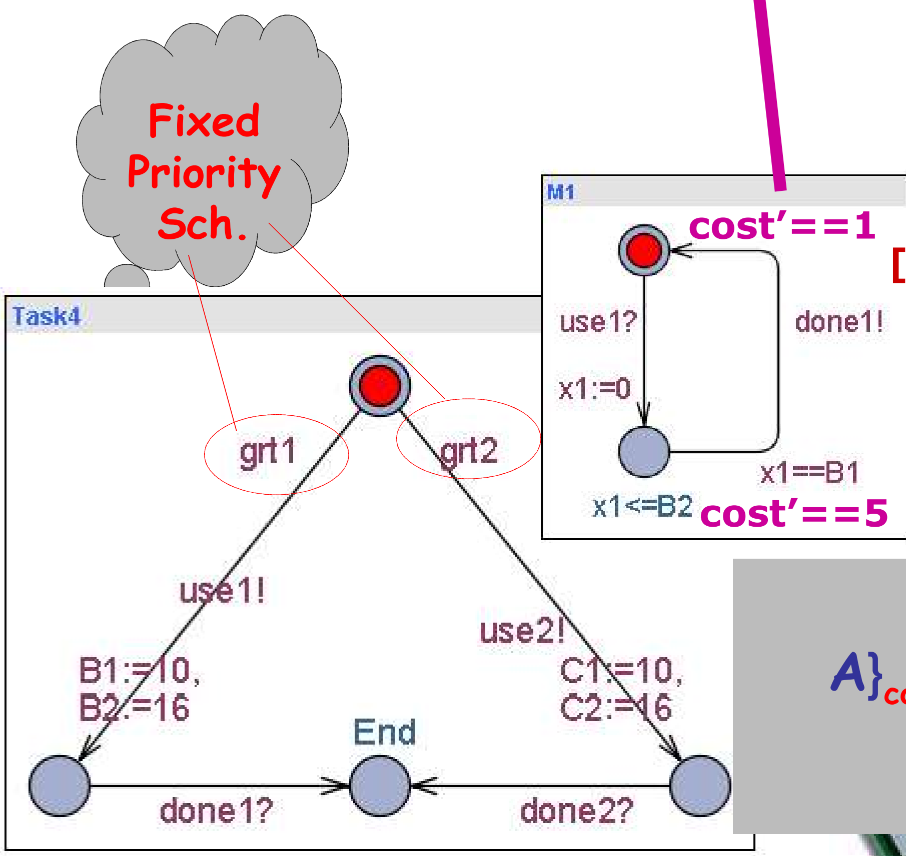


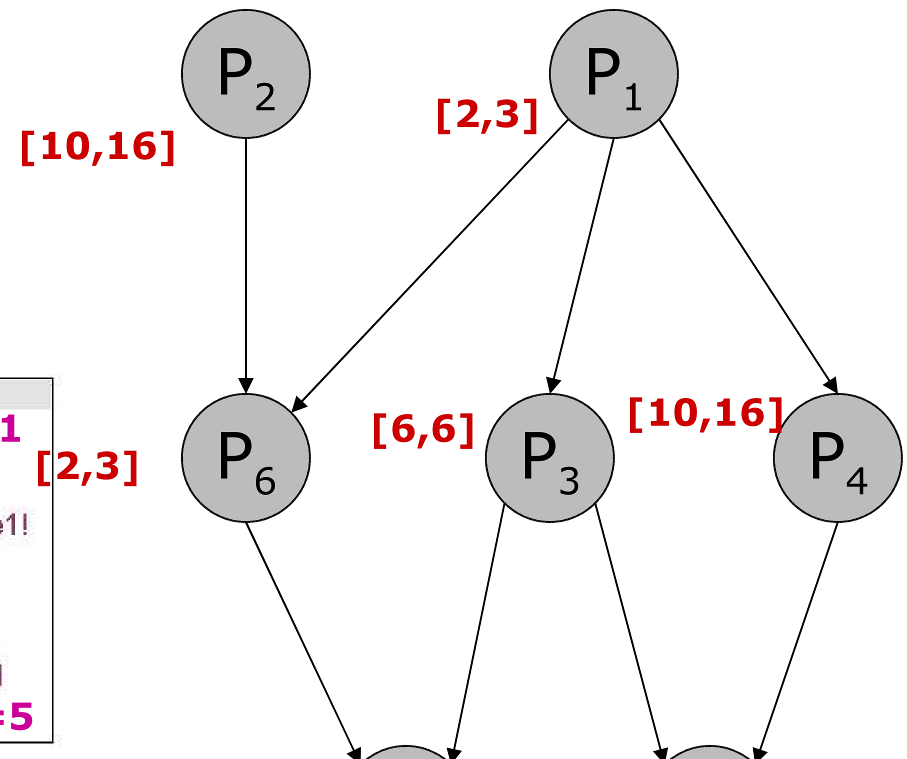
M = $\{M_1, M_2\}$

Cost-Bounded Liveness

Task Graph Scheduling w Uncertainty,

- Energy-rates **C**:**M** ! **N**x**N**



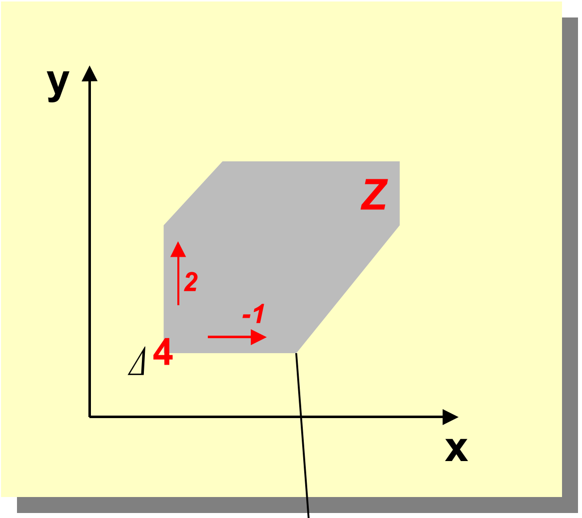


A}$_{cost\cdot\ 60}$ (Task$_1$.End Æ … Æ Task$_7$.End)

???

Priced Zone

CAV'01



Definition

A priced zone P is a tuple (Z, c, r), where:

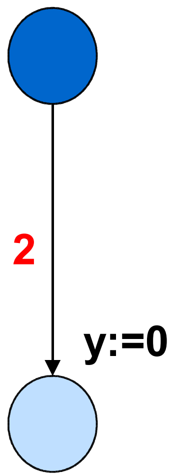
- Z is a zone
- $c \in \mathbb{N}$ describes the cost of Δ_Z
- $r : C \to \mathbb{Z}$ gives a *rate* for any clock $x \in C$.

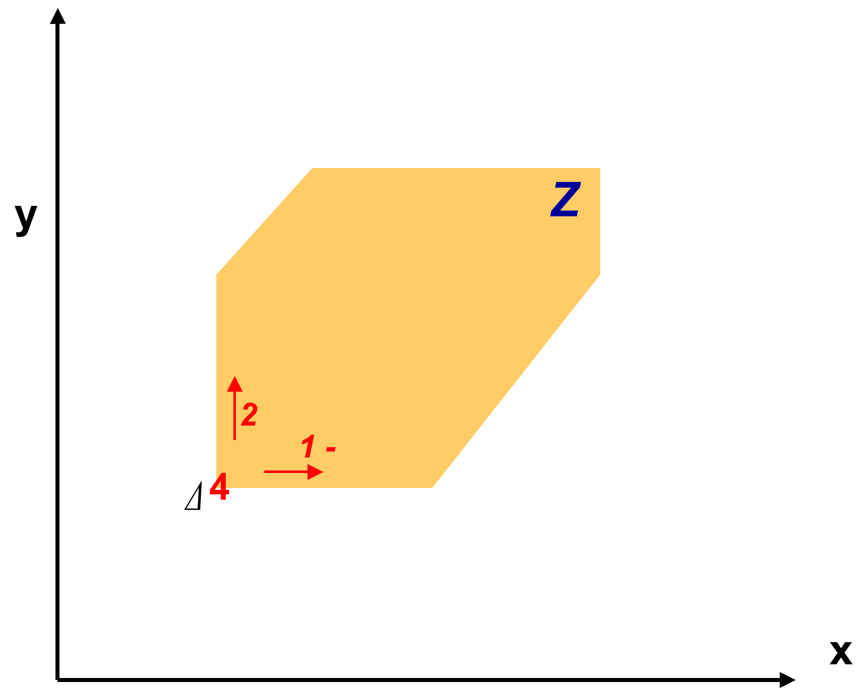
We write $u \models P$ whenever $u \models Z$. For $u \models P$ we define $\mathrm{Cost}(u, P)$ as follows:

$$\mathrm{Cost}(u, P) = c + \sum_{x \in C} r(x) \cdot (u(x) - \Delta_Z(x))$$

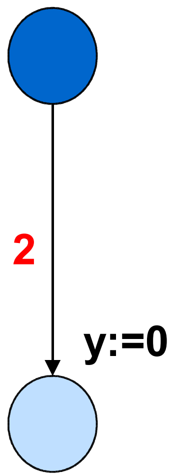
$$Cost(x, y) = 2y - x + 2$$

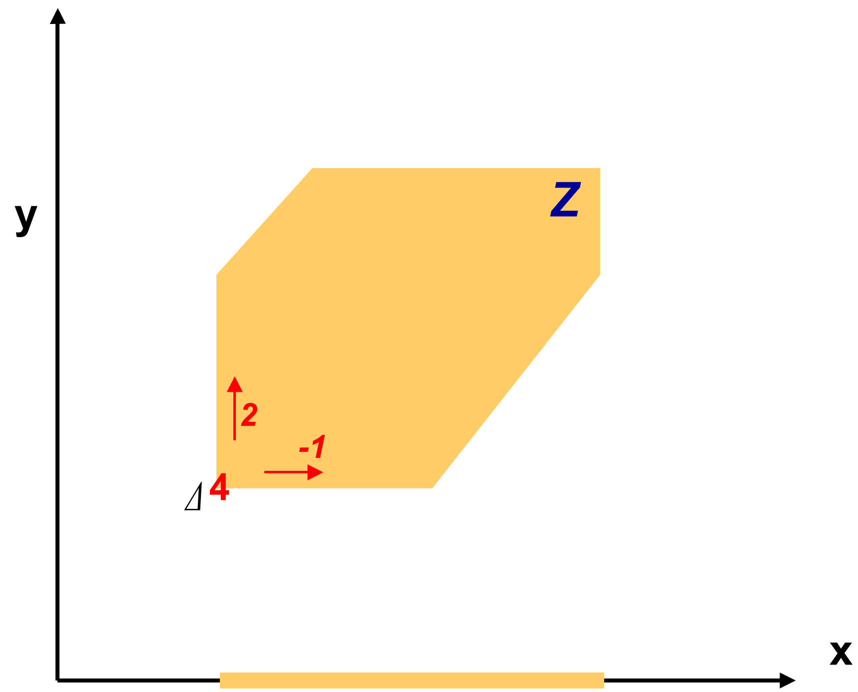
Reset



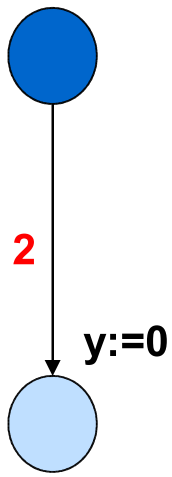


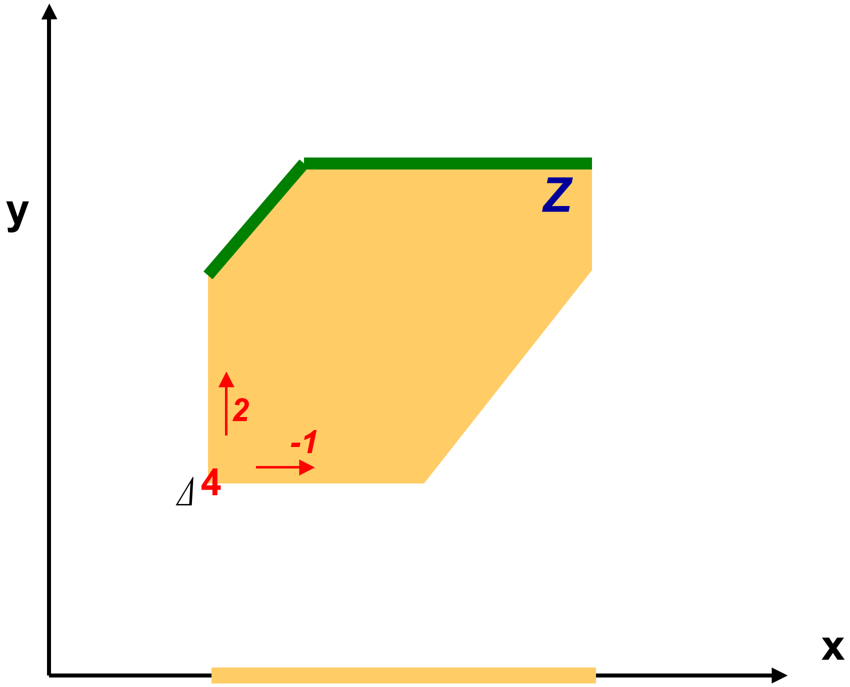
Reset



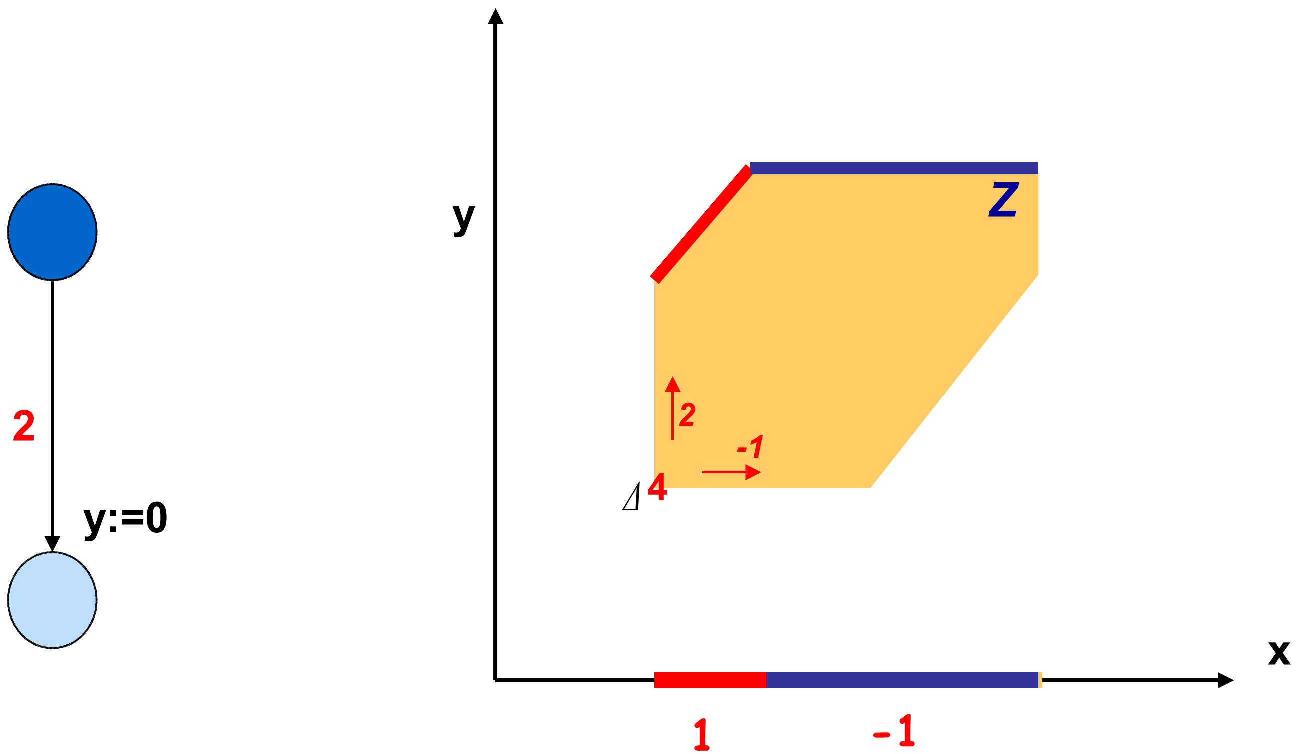


Reset





Reset



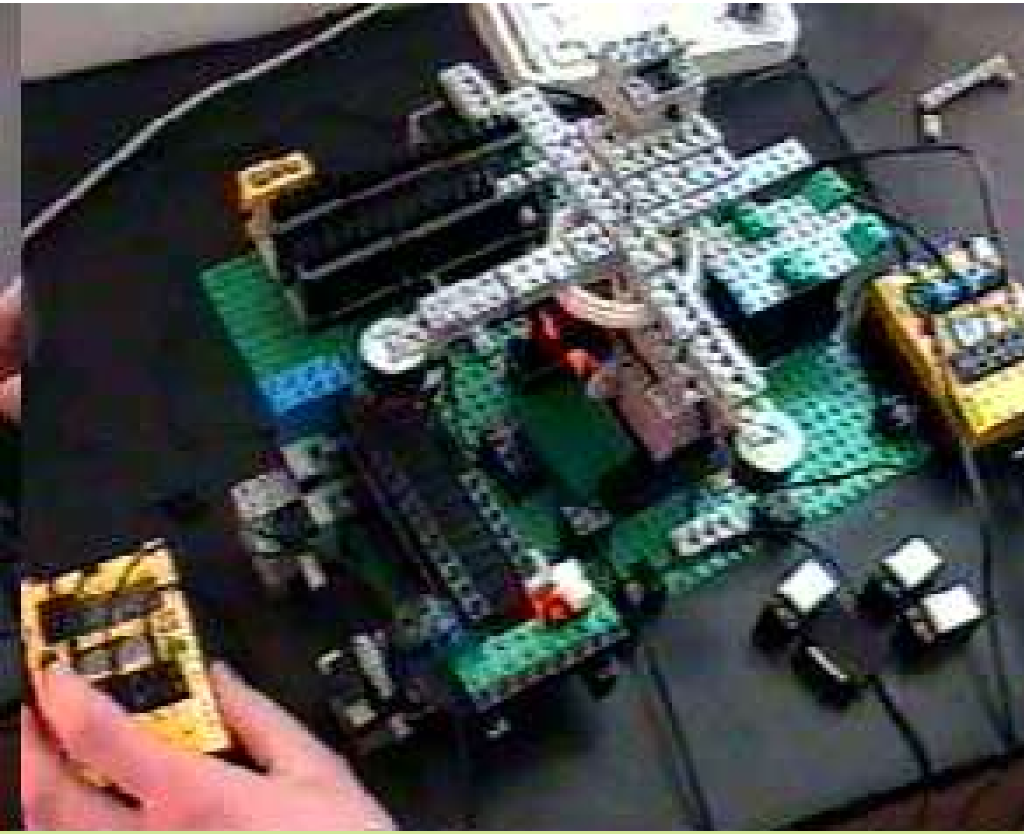
TACAS'2005: For dual-priced TA we use dual-priced zones:

$$(Z, \{(c_1 , d_1), \dots ,(c_k , d_k)\})$$

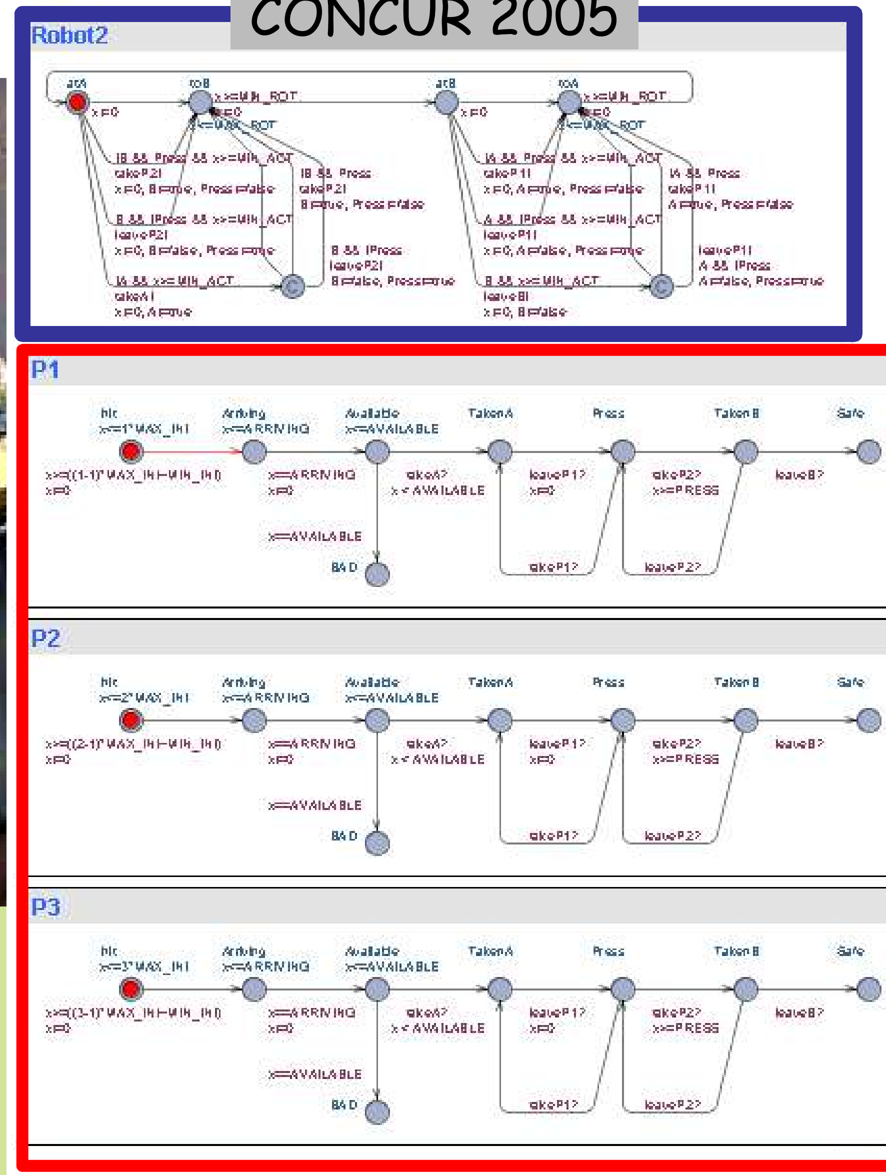
characterizing **ALL** cost-pair by which states may be reached !

Controller Synthesis and Timed Games

Production Cell



CONCUR 2005

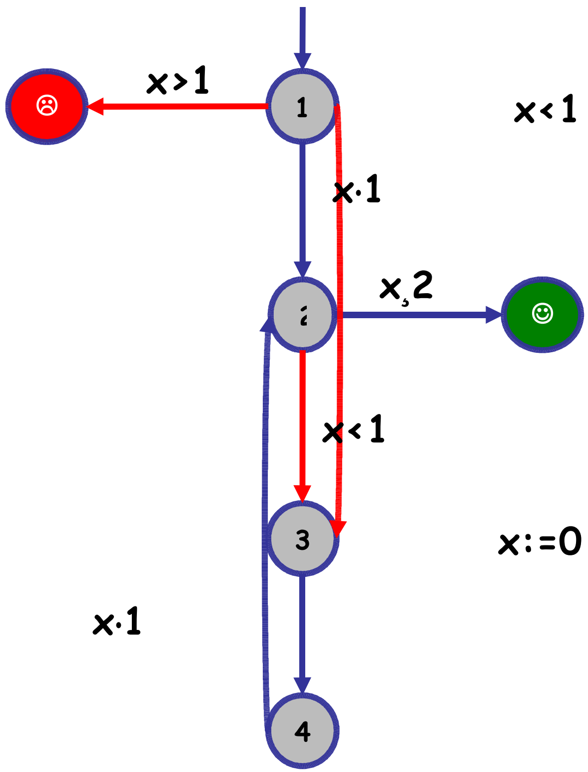


GIVEN System moves S,
Controller moves C, and property ϕ
FIND strategy s_C such that $s_C||S$ ² ϕ

$\rightarrow$

A Two-Player Game

Time Optimality Winning Strategy



Assumption:

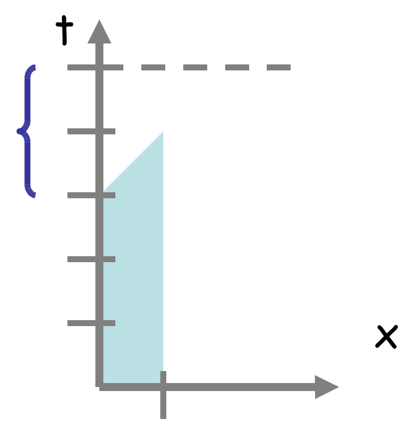
Known upper bound B
-- here 5 (say)

Technique:

Add new clock t
Add invariant
t·B
to all locations
t unconstrained in initial state(s)

Result:

Minimum time required = 2



Experimental Results

Plates		Basic		Basic +inc		Basic +inc +pruning		Basic+lose +inc +pruning		Basic+lose +inc +topt	
		time	mem	time	mem	time	mem	time	mem	time	mem
2	win	0.0s	1M	0.0s	1M	0.0s	1M	0.0s	1M	0.04s	1M
	lose	0.0s	1M	0.0s	1M	0.0s	1M	0.0s	1M	n/a	n/a
3	win	0.5s	19M	0.0s	1M	0.0s	1M	0.1s	1M	0.27s	4M
	lose	1.1s	45M	0.1s	1M	0.0s	1M	0.2s	3M	n/a	n/a
4	win	33.9s	1395M	0.2s	8M	0.1s	6M	0.4s	5M	1.88s	13M
	lose	-	-	0.5s	11M	0.4s	10M	0.9s	9M	n/a	n/a
5	win	-	-	3.0s	31M	1.5s	22M	2.0s	16M	13.35s	59M
	lose	-	-	11.1s	61M	5.9s	46M	7.0s	41M	n/a	n/a
6	win	-	-	89.1s	179M	38.9s	121M	12.0s	63M	220.3s	369M
	lose	-	-	699s	480M	317s	346M	135.1s	273M	n/a	n/a
7	win	-	-	3256s	1183M	1181s	786M	124s	319M	6188s	2457M
	lose	-	-	-	-	16791s	2981M	4075s	2090M	n/a	n/a

Conclusion & Future Work

- Improvements of algorithms:
  - Pruning: give upper bounds on the remaining time for reaching goal condition.
  - Guiding: towards most expensive goal state.
- Alternative algorithms for parameterized liveness:
  - Breadth-first algorithm
  - Forward on-the-fly algorithm
- Extensions to Priced Timed Automata
- Implementation in UPPAAL & UPPAAL Cora

☺ END ☺