Efficient On-the-Fly Algorithms for Partially Observable Timed Games

Franck Cassez
CNRS/IRCCyN

Nantes, France

Joint work with: A. David, E. Fleury, Kim G. Larsen,

D. Lime and J.-F. Raskin

October 5th, 2007

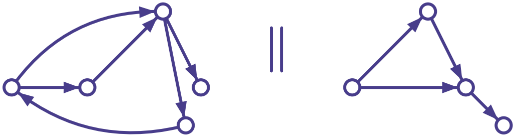
5th International Conference on Formal Modelling
and Analysis of Timed Systems
FORMATS'07

Salzburg, Austria

Verification and Control

Verification and Control

Modelling



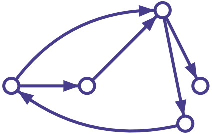
$$S \parallel C \models \varphi$$

Always (not bad)

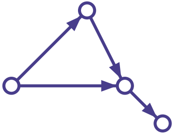
Verification and Control

Does the system meet the specification?

Modelling



$\|$



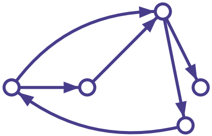
Always (not bad)

$$S \quad \| \quad C \quad \models \quad \varphi$$

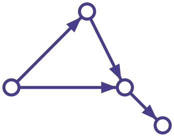
Verification and Control

Does the system meet the specification?

Modelling



||



Always (not bad)

$S \quad || \quad C \quad \models \quad \varphi$

Model Checking Problem

Does the closed system $(S \mid\mid C)$ satisfy φ ?

Verification and Control

Can we enforce the system to meet the specification?

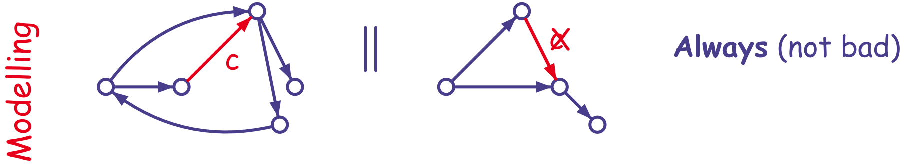
Modelling

Always (not bad)

$$S \parallel C \models \varphi$$

Verification and Control

Can we enforce the system to meet the specification?

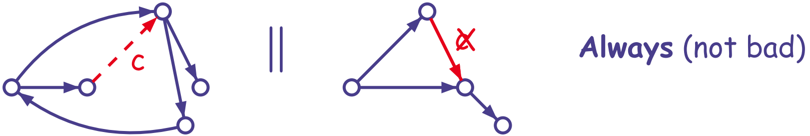


$S \parallel C \models \varphi$

Verification and Control

Can we enforce the system to meet the specification?

Modelling



$S \quad \| \quad C \quad \models \quad \varphi$

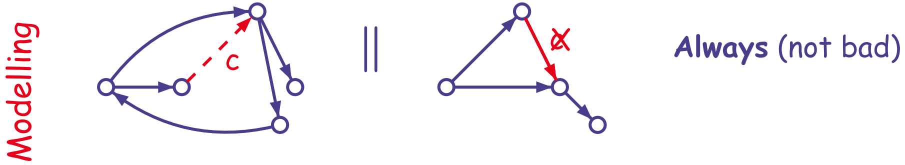
Control Problem

Can the open system S be restricted to satisfy φ ?

Is there a Controller C such that $(S \parallel C) \models \varphi$?

Verification and Control

Can we enforce the system to meet the specification?

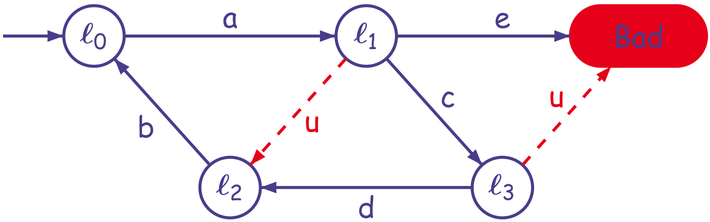


$S \quad || \quad ?? \quad \models \quad \varphi$

Control Problem

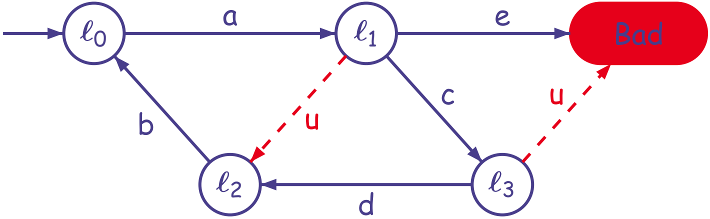
Can the open system S be restricted to satisfy φ ?
Is there a Controller C such that $(S \parallel C) \models \varphi$?

Control Problems as Games



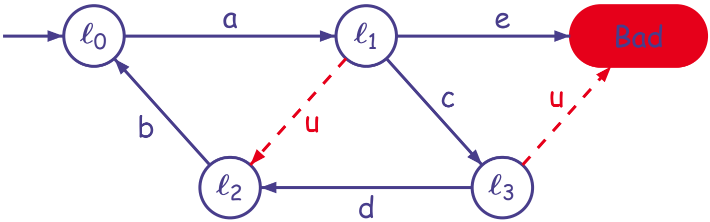
- Control problem = game = controller (C) vs environment (E)
- Various types of game models for C and E
  - Finite or pushdown or counter automata ...
  - Timed (or hybrid) automata
- Goal: find a strategy for the controller to win
  Avoid bad states: safety control
  Enforce good states: reachability control

Control Problems as Games



- Control problem = game = controller (C) vs environment (E)
- Various types of game models for C and E
  - Finite or pushdown or counter automata ...
  - Timed (or hybrid) automata
- Goal: find a strategy for the controller to win
  Avoid bad states: safety control
  Enforce good states: reachability control

Control Problems as Games



- Control problem = game = controller (C) vs environment (E)
- Various types of game models for C and E
  - Finite or pushdown or counter automata ...
  - Timed (or hybrid) automata
- Goal: find a strategy for the controller to win
  Avoid bad states: safety control
  Enforce good states: reachability control

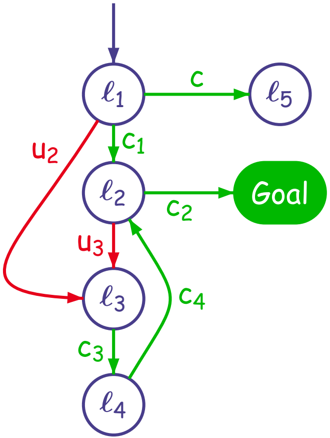
Outline of the Talk

- Reachability Control
  - Finite-State Games
  - Backward Algorithm for Finite State Games
  - Backward Algorithm for Timed Games
  - Summary of the Results for (Reachability) Control
- On-the-fly Algorithms for Reachability Control
  - Finite State Games
  - Timed Games
- Implementation, Optimizations, Time Optimality
- Partial Observation
  - Results About Partial Observation for Timed Games
  - Stuttering Free Observations
  - Knowledge Based Subset Construction

Next:

- Reachability Control
  - Finite-State Games
  - Backward Algorithm for Finite State Games
  - Backward Algorithm for Timed Games
  - Summary of the Results for (Reachability) Control
- On-the-fly Algorithms for Reachability Control
- Implementation, Optimizations, Time Optimality
- Partial Observation

Reachability Control for Finite Games

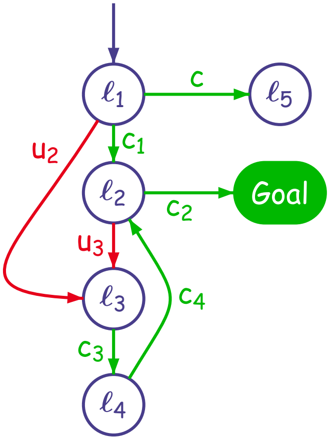


$\rightarrow$ Uncontrollable
$\rightarrow$ Controllable

Aim: enforce Goal

- Semantics: no priority
  Cont. must take a controllable action
- Winning run = a run containing Goal
- Strategy: based on the full history tells which controllable action to fire
  It restricts the set of behaviors of the open system
- Winning strategy: all the runs in the controlled system are winning
- Winning state = a state from which there is winning strategy

Reachability Control for Finite Games

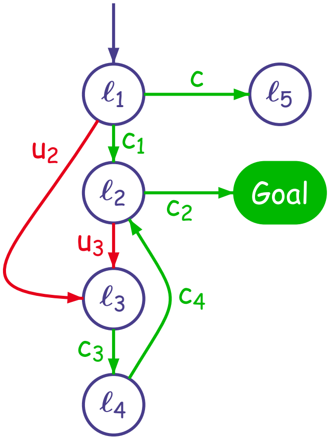


$\rightarrow$ Uncontrollable
$\rightarrow$ Controllable

Aim: enforce Goal

- Semantics: no priority
  Cont. must take a controllable action
- Winning run = a run containing Goal
- Strategy: based on the full history tells which controllable action to fire
  It restricts the set of behaviors of the open system
- Winning strategy: all the runs in the controlled system are winning
- Winning state = a state from which there is winning strategy

Reachability Control for Finite Games



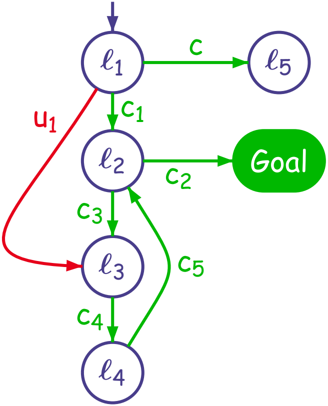
→ Uncontrollable
→ Controllable

Aim: enforce Goal

- Semantics: no priority
  Cont. must take a controllable action
- Winning run = a run containing Goal
- Strategy: based on the full history tells which controllable action to fire
  It restricts the set of behaviors of the open system
- Winning strategy: all the runs in the controlled system are winning
- Winning state = a state from which there is winning strategy

How to Solve Reachability Games?

Backward Computation of Winning States



$\overline{X}$ = complement of X

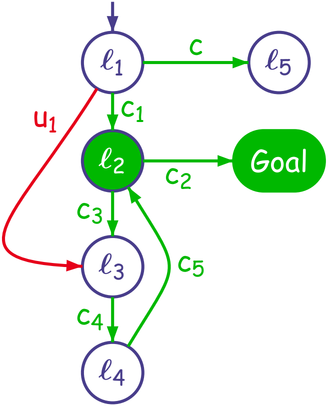
Controllable Predecessors:

$$\pi(X) = \big(cPred(X) \setminus uPred(\overline{X})\big)$$

Iterate π: $X_{i+1} = X_i \cup \pi(X_i)$

1. $X_0 = \{Goal\}$
2. $X_1 = \{Goal, \ell_2\}$
3. $X_2 = \{Goal, \ell_2, \ell_4\}$
4. $X_3 = \{Goal, \ell_2, \ell_4, \ell_3\}$
5. $X_4 = \{Goal, \ell_2, \ell_4, \ell_3, \ell_1\}$

Backward Computation of Winning States



$\overline{X}$ = complement of X

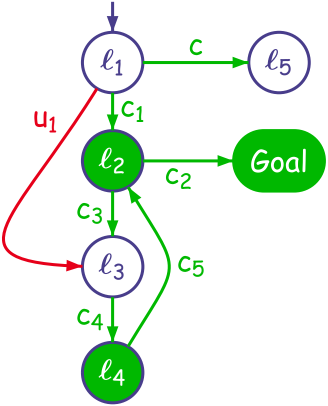
Controllable Predecessors:

$$\pi(X) = \left(cPred(X) \setminus uPred(\overline{X})\right)$$

Iterate π: $X_{i+1} = X_i \cup \pi(X_i)$

1. $X_0 = \{Goal\}$
2. $X_1 = \{Goal, \ell_2\}$
3. $X_2 = \{Goal, \ell_2, \ell_4\}$
4. $X_3 = \{Goal, \ell_2, \ell_4, \ell_3\}$
5. $X_4 = \{Goal, \ell_2, \ell_4, \ell_3, \ell_1\}$

Backward Computation of Winning States



$\overline{X}$ = complement of X

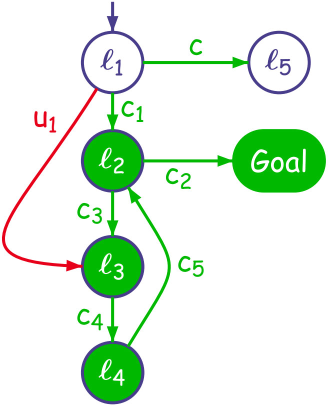
Controllable Predecessors:

$$\pi(X) = \big(cPred(X) \setminus uPred(\overline{X})\big)$$

Iterate π: $X_{i+1} = X_i \cup \pi(X_i)$

1. $X_0 = \{Goal\}$
2. $X_1 = \{Goal, \ell_2\}$
3. $X_2 = \{Goal, \ell_2, \ell_4\}$
4. $X_3 = \{Goal, \ell_2, \ell_4, \ell_3\}$
5. $X_4 = \{Goal, \ell_2, \ell_4, \ell_3, \ell_1\}$

Backward Computation of Winning States



$\overline{X}$ = complement of X

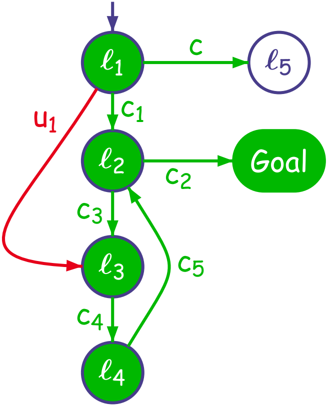
Controllable Predecessors:

$$\pi(X) = \big(cPred(X) \setminus uPred(\overline{X})\big)$$

Iterate π: $X_{i+1} = X_i \cup \pi(X_i)$

1. $X_0 = \{Goal\}$
2. $X_1 = \{Goal, \ell_2\}$
3. $X_2 = \{Goal, \ell_2, \ell_4\}$
4. $X_3 = \{Goal, \ell_2, \ell_4, \ell_3\}$
5. $X_4 = \{Goal, \ell_2, \ell_4, \ell_3, \ell_1\}$

Backward Computation of Winning States



$\overline{X}$ = complement of X

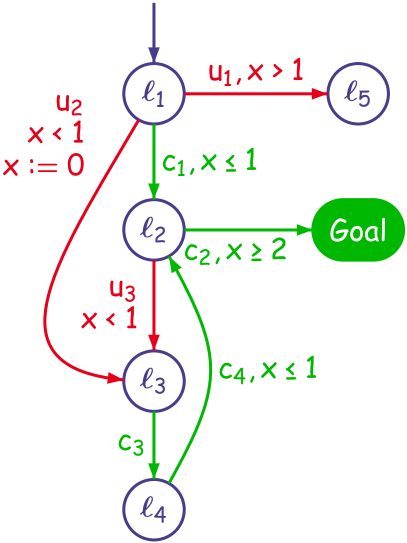
Controllable Predecessors:

$$\pi(X) = \left(cPred(X) \setminus uPred(\overline{X})\right)$$

Iterate π: $X_{i+1} = X_i \cup \pi(X_i)$

1. $X_0 = \{Goal\}$
2. $X_1 = \{Goal, \ell_2\}$
3. $X_2 = \{Goal, \ell_2, \ell_4\}$
4. $X_3 = \{Goal, \ell_2, \ell_4, \ell_3\}$
5. $X_4 = \{Goal, \ell_2, \ell_4, \ell_3, \ell_1\}$

Reachability Control for Timed Games



Safe Time Elapsing:
When is it safe to let time elapse from q to q′ ?

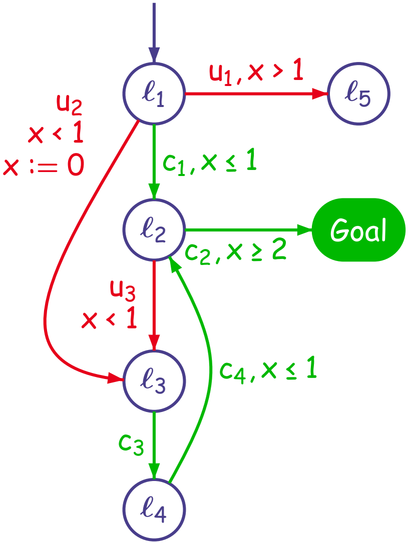
q $\qquad$ $q' \in X$

Controllable Predecessors:

$$\pi(X) = \mathrm{Pred}_t\big(X \cup \mathrm{cPred}(X), \mathrm{uPred}(\overline{X})\big)$$

▸ Timed Auto

Reachability Control for Timed Games



Safe Time Elapsing:
When is it safe to let time elapse from q to q′ ?

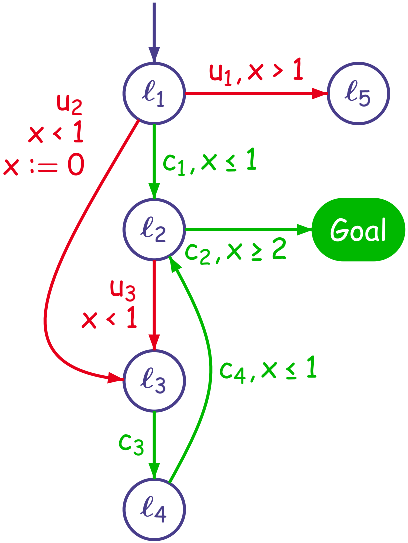
$$q \xrightarrow{\quad t \quad} q' \in X$$

Controllable Predecessors:

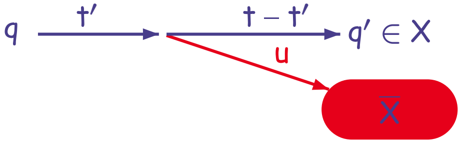
$$\pi(X) = \mathrm{Pred}_t\left(X \cup \mathrm{cPred}(X), \mathrm{uPred}(\overline{X})\right)$$

Timed Auto

Reachability Control for Timed Games



Safe Time Elapsing:
When is it safe to let time elapse from q to q' ?

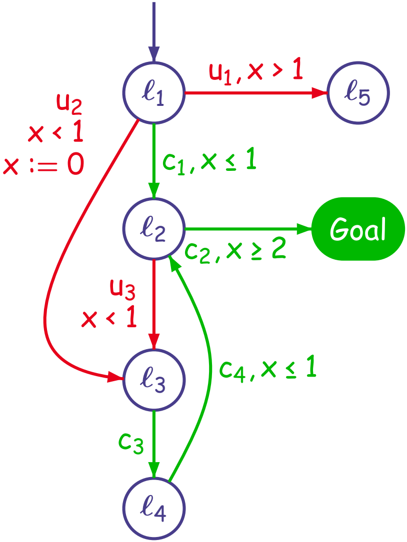


Controllable Predecessors:

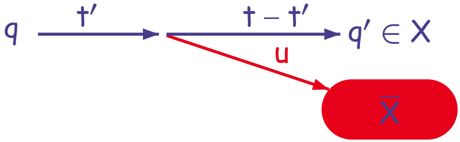
$$\pi(X) = \mathrm{Pred}_t\left(X \cup \mathrm{cPred}(X), \mathrm{uPred}(\overline{X})\right)$$

Timed Auto

Reachability Control for Timed Games



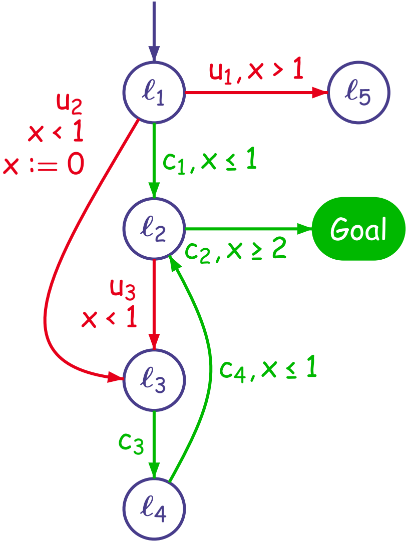
Safe Time Elapsing:
When is it safe to let time elapse from q to q' ?

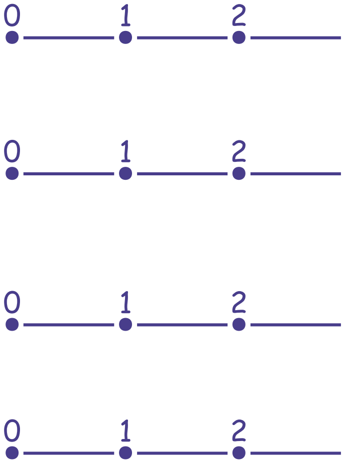


Controllable Predecessors:

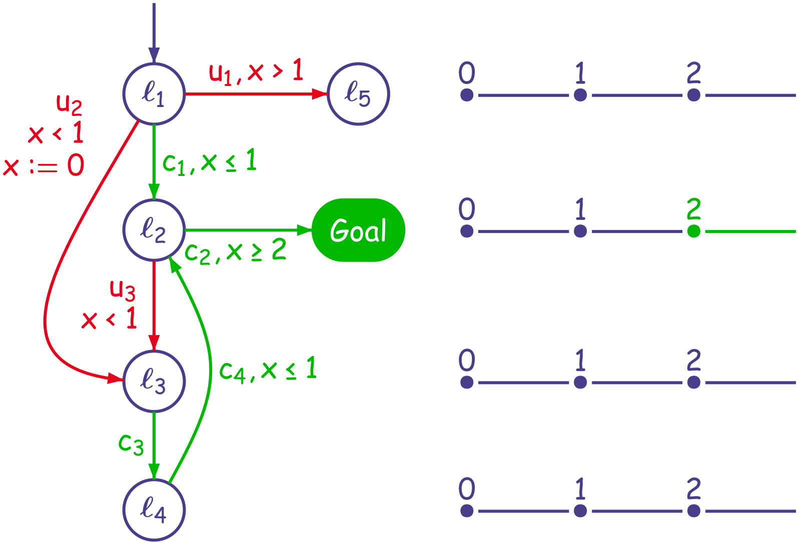
$$\pi(X) = \mathrm{Pred}_t\big(X \cup \mathrm{cPred}(X), \mathrm{uPred}(\overline{X})\big)$$

Reachability Control for Timed Games

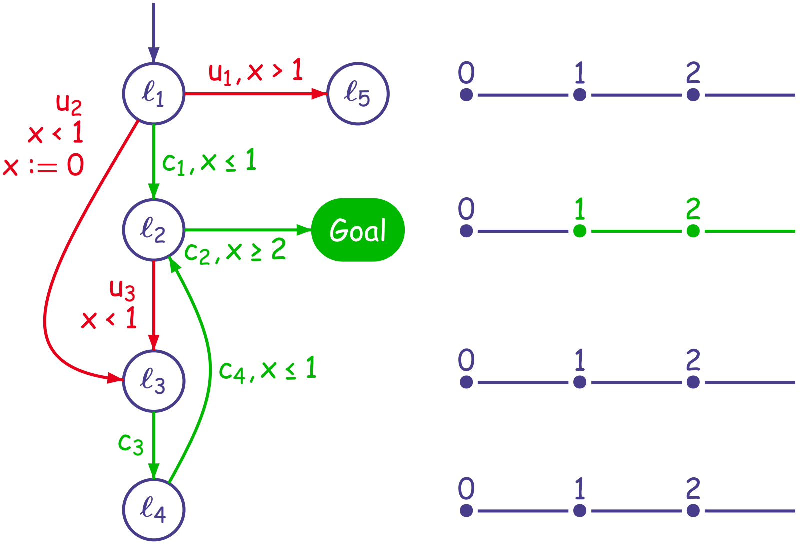




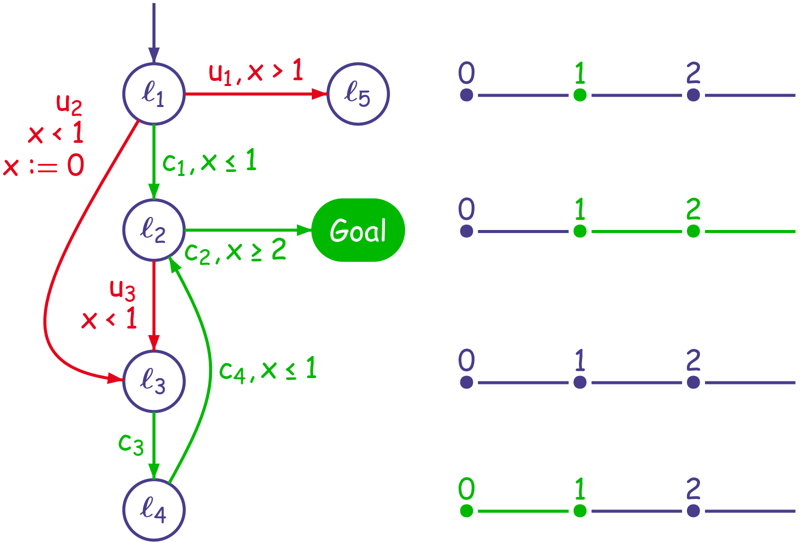
Reachability Control for Timed Games



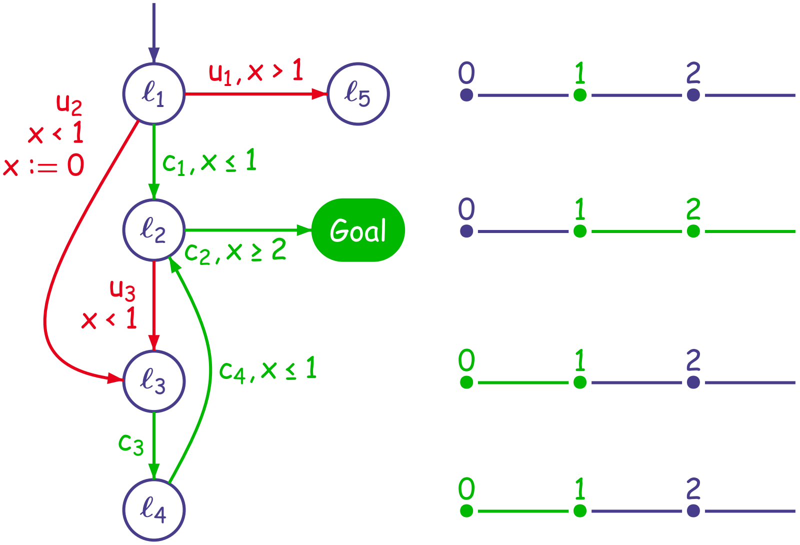
Reachability Control for Timed Games



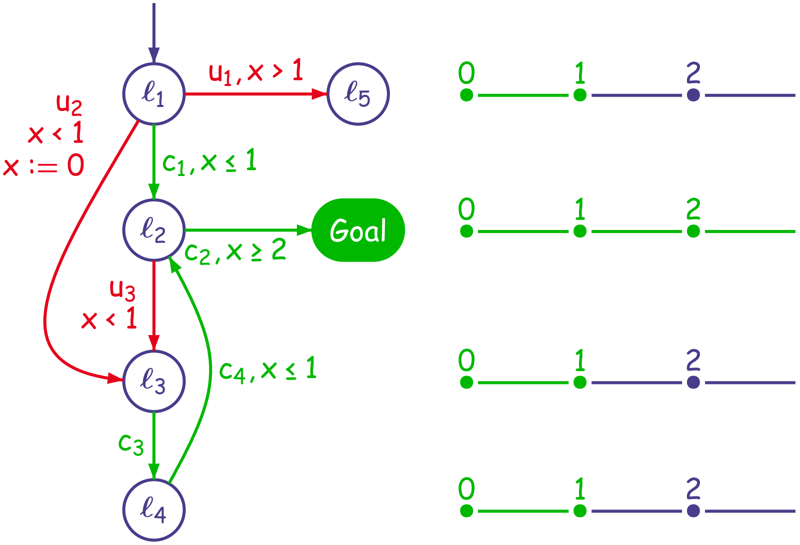
Reachability Control for Timed Games



Reachability Control for Timed Games



Reachability Control for Timed Games



State-of-the-art

Known Results for Timed (Game) Automata:

- Reachability in Timed Automata [Alur & Dill'94]
- Büchi Control for Timed Game Automata [Maler et al.'95]
- Time Optimal Control [Asarin & Maler'99]
- Optimal Control for Priced Timed Game Automata [Bouyer et al.'04]
- Half on-the-fly algorithm [Altisen & Tripakis'99, Altisen & Tripakis'02]

Our Contribution: True On-the-fly algorithm for reachability games

- Advantages: [Concur'05]
  - avoid constructing all backward & forward reachable states
  - allows for use of discrete variables (e.g. $i := i + 1$)
- Extends to Time-Optimal Control
- Extends to Partially Observable Games [ATVA'07]
- Efficient implementation in UPPAAL-TiGA [UPPAAL-TiGA'07]

State-of-the-art

Known Results for Timed (Game) Automata:

- Reachability in Timed Automata [Alur & Dill'94]
- Büchi Control for Timed Game Automata [Maler et al.'95]
- Time Optimal Control [Asarin & Maler'99]
- Optimal Control for Priced Timed Game Automata [Bouyer et al.'04]
- Half on-the-fly algorithm [Altisen & Tripakis'99, Altisen & Tripakis'02]

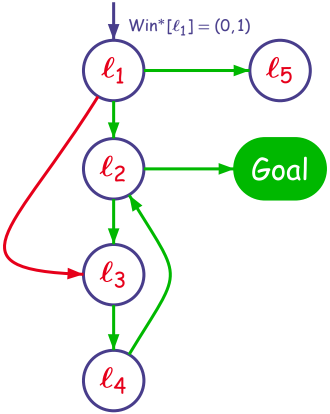
Our Contribution: True On-the-fly algorithm for reachability games

- Advantages: [Concur'05]
  - avoid constructing all backward & forward reachable states
  - allows for use of discrete variables (e.g. $i := i + 1$)
- Extends to Time-Optimal Control
- Extends to Partially Observable Games [ATVA'07]
- Efficient implementation in **UPPAAL-TiGA** [UPPAAL-TiGA'07]

Next:

- Reachability Control
- On-the-fly Algorithms for Reachability Control
  - Finite State Games
  - Timed Games
- Implementation, Optimizations, Time Optimality
- Partial Observation

Liu & Smolka Algorithm [Liu & Smolka'98]

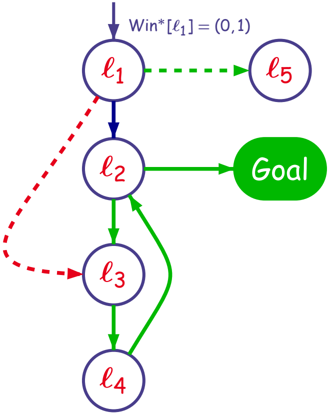


$\longrightarrow$ Current
$\longrightarrow$ Explored Forwards
Dashed = in Waiting List

Initialization:
  Passed $\leftarrow \{q_0\}$;
  Waiting $\leftarrow \{(q_0, \alpha, q') \mid \alpha \in Act\ q \xrightarrow{\alpha} q'\}$;
  Win$[q_0] \leftarrow (q_0 \in Goal\ ?\ 1 : 0)$;
  Depend$[q_0] \leftarrow \emptyset$;

Main:
while ((Waiting $\neq \emptyset$) $\wedge$ Win$[q_0] \neq 1$)) **do**
  $e = (q, \alpha, q') \leftarrow$ pop(Waiting);
  if $q' \notin$ Passed **then** {
    Passed $\leftarrow$ Passed $\cup \{q'\}$;
    Depend$[q'] \leftarrow \{(q, \alpha, q')\}$;
    Win$[q'] \leftarrow (q' \in Goal\ ?\ 1 : 0)$;
    Waiting $\leftarrow$ Waiting $\cup \{(q', \alpha, q'') \mid q' \xrightarrow{\alpha} q''\}$;
    Win$^*[q] \leftarrow (0, \#\{q \xrightarrow{u}\})$;
    if Win$[q']$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  }
  else (* reevaluate *)
    Win$^*[q] \leftarrow$ **Update**(Win$^*[q]$) ;
    if (Win$^*[q] = (k, 0) \wedge k \geq 1$) **then** {
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[q]$;
      Win$[q] \leftarrow 1$;
    }
    if Win$[q'] = 0$ **then** Depend$[q'] \leftarrow$ Depend$[q'] \cup \{e\}$;
  endif
endwhile

Liu & Smolka Algorithm [Liu & Smolka'98]



→ Current

→ Explored Forwards

Dashed = in Waiting List

$$e = (\ell_1, c_1, \ell_2)$$

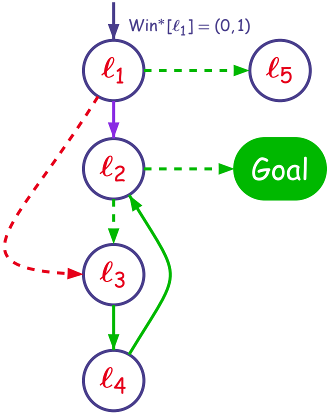
Initialization:

Passed $\leftarrow \{q_0\}$;
Waiting $\leftarrow \{(q_0, \alpha, q') \mid \alpha \in Act\ q \xrightarrow{\alpha} q'\}$;
Win$[q_0] \leftarrow (q_0 \in Goal\ ?\ 1 : 0)$;
Depend$[q_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ Win$[q_0] \neq 1$)) **do**
  $e = (q, \alpha, q') \leftarrow$ pop(Waiting);
  if $q' \notin$ Passed **then {**
    Passed $\leftarrow$ Passed $\cup \{q'\}$;
    Depend$[q'] \leftarrow \{(q, \alpha, q')\}$;
    Win$[q'] \leftarrow (q' \in Goal\ ?\ 1 : 0)$;
    Waiting $\leftarrow$ Waiting $\cup \{(q', \alpha, q'') \mid q' \xrightarrow{\alpha} q''\}$;
    Win$^*[q] \leftarrow (0, \#\{q \xrightarrow{u}\})$;
    if Win$[q']$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  }
  else (* reevaluate *)
    Win$^*[q] \leftarrow$ **Update**(Win$^*[q]$) ;
    if (Win$^*[q] = (k, 0) \wedge k \geq 1$) **then {**
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[q]$;
      Win$[q] \leftarrow 1$;
    }
    if Win$[q'] = 0$ **then** Depend$[q'] \leftarrow$ Depend$[q'] \cup \{e\}$;
  endif
endwhile

Liu & Smolka Algorithm [Liu & Smolka'98]



$\longrightarrow$ Current
$\longrightarrow$ Explored Forwards
Dashed = in Waiting List

$$e = (\ell_1, c_1, \ell_2)$$

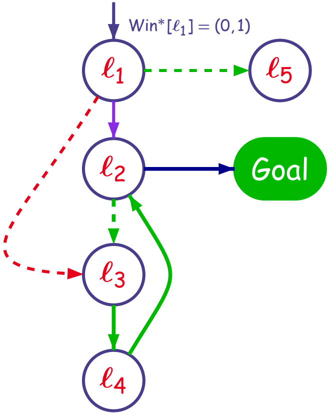
Initialization:

Passed $\leftarrow \{q_0\}$;
Waiting $\leftarrow \{(q_0, \alpha, q') \mid \alpha \in Act\ q \xrightarrow{\alpha} q'\}$;
Win$[q_0] \leftarrow (q_0 \in Goal\ ?\ 1 : 0)$;
Depend$[q_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ Win$[q_0] \neq 1$)) **do**
  $e = (q, \alpha, q') \leftarrow$ pop(Waiting);
  if $q' \notin$ Passed **then** {
    Passed $\leftarrow$ Passed $\cup \{q'\}$;
    Depend$[q'] \leftarrow \{(q, \alpha, q')\}$;
    Win$[q'] \leftarrow (q' \in Goal\ ?\ 1 : 0)$;
    Waiting $\leftarrow$ Waiting $\cup \{(q', \alpha, q'') \mid q' \xrightarrow{\alpha} q''\}$;
    Win$^*[q] \leftarrow (0, \#\{q \xrightarrow{u}\})$;
    if Win$[q']$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  }
  else (* reevaluate *)
    Win$^*[q] \leftarrow$ **Update**(Win$^*[q]$) ;
    if (Win$^*[q] = (k, 0) \wedge k \geq 1$) **then** {
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[q]$;
      Win$[q] \leftarrow 1$;
    }
    if Win$[q'] = 0$ **then** Depend$[q'] \leftarrow$ Depend$[q'] \cup \{e\}$;
  endif
endwhile

Liu & Smolka Algorithm [Liu & Smolka'98]



$\longrightarrow$ Current
$\longrightarrow$ Explored Forwards
Dashed = in Waiting List

$$e = (\ell_2, c_2, Goal)$$

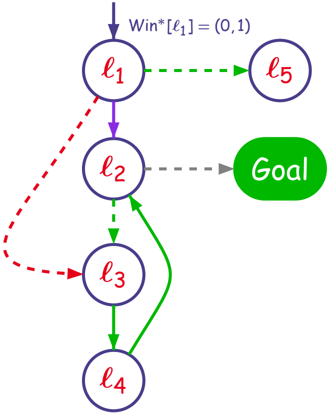
Initialization:

Passed $\leftarrow \{q_0\}$;
Waiting $\leftarrow \{(q_0, a, q') \mid a \in Act\ q \xrightarrow{a} q'\}$;
Win$[q_0] \leftarrow (q_0 \in Goal\ ?\ 1 : 0)$;
Depend$[q_0] \leftarrow \emptyset$;

Main:

```
while ((Waiting ≠ ∅) ∧ Win[q0] ≠ 1)) do
    e = (q, a, q') ← pop(Waiting);
    if q' ∉ Passed then {
        Passed ← Passed ∪ {q'};
        Depend[q'] ← {(q, a, q')};
        Win[q'] ← (q' ∈ Goal ? 1 : 0);
        Waiting ← Waiting ∪ {(q', a, q'') | q' -a-> q''};
        Win*[q] ← (0, #{q -u->});
        if Win[q'] then Waiting ← Waiting ∪ {e};
    }
    else (* reevaluate *)
        Win*[q] ← Update(Win*[q]) ;
        if (Win*[q] = (k, 0) ∧ k ≥ 1) then {
            Waiting ← Waiting ∪ Depend[q];
            Win[q] ← 1;
        }
        if Win[q'] = 0 then Depend[q'] ← Depend[q'] ∪ {e};
    endif
endwhile
```

Liu & Smolka Algorithm [Liu & Smolka'98]



$\longrightarrow$ Current

$\longrightarrow$ Explored Forwards

Dashed = in Waiting List

$$e = (\ell_2, c_2, Goal)$$

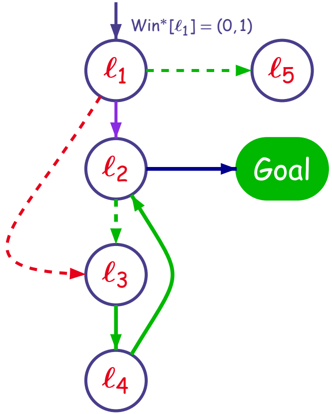
Initialization:

Passed $\leftarrow \{q_0\}$;
Waiting $\leftarrow \{(q_0, \alpha, q') \mid \alpha \in Act\ q \xrightarrow{\alpha} q'\}$;
Win$[q_0] \leftarrow (q_0 \in Goal\ ?\ 1 : 0)$;
Depend$[q_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ Win$[q_0] \neq 1$)) **do**
  $e = (q, \alpha, q') \leftarrow$ pop(Waiting);
  if $q' \notin$ Passed **then** {
    Passed $\leftarrow$ Passed $\cup \{q'\}$;
    Depend$[q'] \leftarrow \{(q, \alpha, q')\}$;
    Win$[q'] \leftarrow (q' \in Goal\ ?\ 1 : 0)$;
    Waiting $\leftarrow$ Waiting $\cup \{(q', \alpha, q'') \mid q' \xrightarrow{\alpha} q''\}$;
    Win$^*[q] \leftarrow (0, \#\{q \xrightarrow{u}\})$;
    if Win$[q']$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  }
  else (* reevaluate *)
    Win$^*[q] \leftarrow$ **Update**(Win$^*[q]$) ;
    if (Win$^*[q] = (k, 0) \wedge k \geq 1$) **then** {
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[q]$;
      Win$[q] \leftarrow 1$;
    }
    if Win$[q'] = 0$ **then** Depend$[q'] \leftarrow$ Depend$[q'] \cup \{e\}$;
  endif
endwhile

Liu & Smolka Algorithm [Liu & Smolka'98]



$\longrightarrow$ Current

$\longrightarrow$ Explored Forwards

Dashed = in Waiting List

$$e = (\ell_2, c_2, Goal)$$

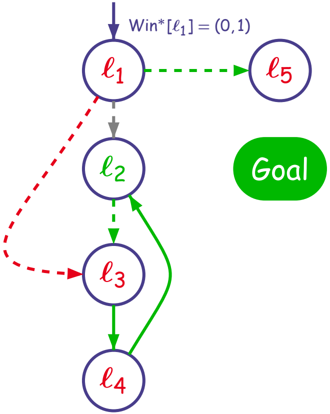
Initialization:

Passed $\leftarrow \{q_0\}$;
Waiting $\leftarrow \{(q_0, a, q') \mid a \in Act\ q \xrightarrow{a} q'\}$;
Win$[q_0] \leftarrow (q_0 \in Goal\ ?\ 1 : 0)$;
Depend$[q_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ Win$[q_0] \neq 1$)) **do**
  $e = (q, a, q') \leftarrow$ pop(Waiting);
  if $q' \notin$ Passed **then** {
    Passed $\leftarrow$ Passed $\cup \{q'\}$;
    Depend$[q'] \leftarrow \{(q, a, q')\}$;
    Win$[q'] \leftarrow (q' \in Goal\ ?\ 1 : 0)$;
    Waiting $\leftarrow$ Waiting $\cup \{(q', a, q'') \mid q' \xrightarrow{a} q''\}$;
    Win$^*[q] \leftarrow (0, \#\{q \xrightarrow{u}\})$;
    if Win$[q']$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  }
  else (* reevaluate *)
    Win$^*[q] \leftarrow$ **Update**(Win$^*[q]$) ;
    if (Win$^*[q] = (k, 0) \wedge k \geq 1$) **then** {
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[q]$;
      Win$[q] \leftarrow 1$;
    }
    if Win$[q'] = 0$ **then** Depend$[q'] \leftarrow$ Depend$[q'] \cup \{e\}$;
  endif
endwhile

Liu & Smolka Algorithm [Liu & Smolka'98]



$\longrightarrow$ Current

$\longrightarrow$ Explored Forwards

Dashed = in Waiting List

$$e = (\ell_2, c_2, Goal)$$

Initialization:

$\quad$ Passed $\leftarrow \{q_0\}$;

$\quad$ Waiting $\leftarrow \{(q_0, \alpha, q') \mid \alpha \in Act\ q \xrightarrow{\alpha} q'\}$;

$\quad$ Win$[q_0] \leftarrow (q_0 \in Goal\ ?\ 1 : 0)$;

$\quad$ Depend$[q_0] \leftarrow \emptyset$;

Main:

while $((\text{Waiting} \neq \emptyset) \wedge \text{Win}[q_0] \neq 1))$ **do**

$\quad e = (q, \alpha, q') \leftarrow$ pop(Waiting);

$\quad$ **if** $q' \notin$ Passed **then** {

$\qquad$ Passed $\leftarrow$ Passed $\cup \{q'\}$;

$\qquad$ Depend$[q'] \leftarrow \{(q, \alpha, q')\}$;

$\qquad$ Win$[q'] \leftarrow (q' \in Goal\ ?\ 1 : 0)$;

$\qquad$ Waiting $\leftarrow$ Waiting $\cup \{(q', \alpha, q'') \mid q' \xrightarrow{\alpha} q''\}$;

$\qquad$ Win$^*[q] \leftarrow (0, \#\{q \xrightarrow{u}\})$;

$\qquad$ **if** Win$[q']$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;

$\quad$ }

$\quad$ **else (* reevaluate *)**

$\qquad$ Win$^*[q] \leftarrow$ **Update**(Win$^*[q]$) ;

$\qquad$ **if** (Win$^*[q] = (k, 0) \wedge k \geq 1$) **then** {

$\qquad\quad$ Waiting $\leftarrow$ Waiting $\cup$ Depend$[q]$;

$\qquad\quad$ Win$[q] \leftarrow 1$;

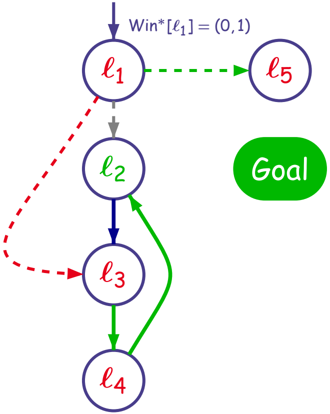
$\qquad$ }

$\qquad$ **if** Win$[q'] = 0$ **then** Depend$[q'] \leftarrow$ Depend$[q'] \cup \{e\}$;

$\quad$ **endif**

endwhile

Liu & Smolka Algorithm [Liu & Smolka'98]



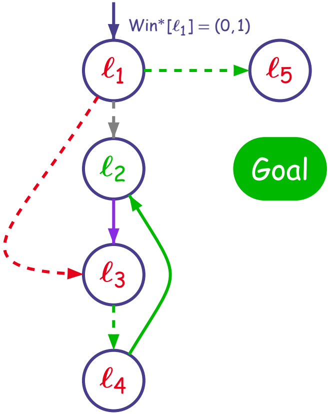
→ Current
→ Explored Forwards
Dashed = in Waiting List

$$e = (\ell_2, c_3, \ell_3)$$

Initialization:
Passed ← $\{q_0\}$;
Waiting ← $\{(q_0, a, q') \mid a \in Act\ q \xrightarrow{a} q'\}$;
$Win[q_0] \leftarrow (q_0 \in Goal\ ?\ 1 : 0)$;
$Depend[q_0] \leftarrow \emptyset$;

Main:
while $((Waiting \neq \emptyset) \wedge Win[q_0] \neq 1)$ **do**
  $e = (q, a, q') \leftarrow pop(Waiting)$;
  if $q' \notin Passed$ **then {**
    $Passed \leftarrow Passed \cup \{q'\}$;
    $Depend[q'] \leftarrow \{(q, a, q')\}$;
    $Win[q'] \leftarrow (q' \in Goal\ ?\ 1 : 0)$;
    $Waiting \leftarrow Waiting \cup \{(q', a, q'') \mid q' \xrightarrow{a} q''\}$;
    $Win^*[q] \leftarrow (0, \#\{q \xrightarrow{u}\})$;
    if $Win[q']$ **then** $Waiting \leftarrow Waiting \cup \{e\}$;
  }
  else (* reevaluate *)
    $Win^*[q] \leftarrow$ **Update**$(Win^*[q])$;
    if $(Win^*[q] = (k, 0) \wedge k \geq 1)$ **then {**
      $Waiting \leftarrow Waiting \cup Depend[q]$;
      $Win[q] \leftarrow 1$;
    }
    if $Win[q'] = 0$ **then** $Depend[q'] \leftarrow Depend[q'] \cup \{e\}$;
  endif
endwhile

Liu & Smolka Algorithm [Liu & Smolka'98]



→ Current
→ Explored Forwards
Dashed = in Waiting List

$$e = (\ell_2, c_3, \ell_3)$$

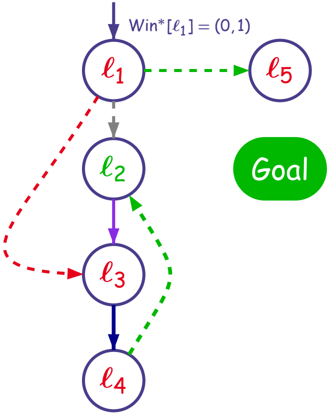
Initialization:

Passed $\leftarrow \{q_0\}$;
Waiting $\leftarrow \{(q_0, \alpha, q') \mid \alpha \in Act\ q \xrightarrow{\alpha} q'\}$;
Win$[q_0] \leftarrow (q_0 \in Goal\ ?\ 1 : 0)$;
Depend$[q_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ Win$[q_0] \neq 1$)) **do**
  $e = (q, \alpha, q') \leftarrow$ pop(Waiting);
  if $q' \notin$ Passed **then {**
    Passed $\leftarrow$ Passed $\cup \{q'\}$;
    Depend$[q'] \leftarrow \{(q, \alpha, q')\}$;
    Win$[q'] \leftarrow (q' \in Goal\ ?\ 1 : 0)$;
    Waiting $\leftarrow$ Waiting $\cup \{(q', \alpha, q'') \mid q' \xrightarrow{\alpha} q''\}$;
    Win*$[q] \leftarrow (0, \#\{q \xrightarrow{u}\})$;
    if Win$[q']$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  }
  else (* reevaluate *)
    Win*$[q] \leftarrow$ **Update**(Win*$[q]$) ;
    if (Win*$[q] = (k, 0) \wedge k \geq 1$) **then {**
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[q]$;
      Win$[q] \leftarrow 1$;
    }
    if Win$[q'] = 0$ **then** Depend$[q'] \leftarrow$ Depend$[q'] \cup \{e\}$;
  endif
endwhile

Liu & Smolka Algorithm [Liu & Smolka'98]



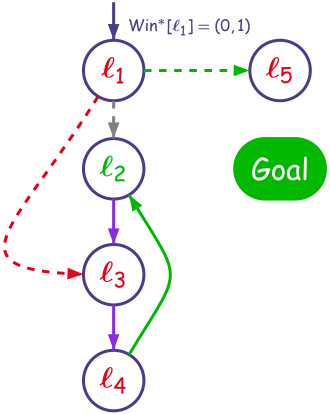
$\longrightarrow$ Current
$\longrightarrow$ Explored Forwards
Dashed = in Waiting List

$$e = (\ell_2, c_3, \ell_3)$$

Initialization:
Passed $\leftarrow \{q_0\}$;
Waiting $\leftarrow \{(q_0, a, q') \mid a \in Act\ q \xrightarrow{a} q'\}$;
Win[q_0] $\leftarrow$ ($q_0 \in$ Goal ? 1 : 0);
Depend[q_0] $\leftarrow \emptyset$;

Main:
while ((Waiting $\neq \emptyset$) $\wedge$ Win[q_0] $\neq$ 1)) **do**
  $e = (q, a, q') \leftarrow$ pop(Waiting);
  if $q' \notin$ Passed **then** {
    Passed $\leftarrow$ Passed $\cup \{q'\}$;
    Depend[q'] $\leftarrow \{(q, a, q')\}$;
    Win[q'] $\leftarrow$ ($q' \in$ Goal ? 1 : 0);
    Waiting $\leftarrow$ Waiting $\cup \{(q', a, q'') \mid q' \xrightarrow{a} q''\}$;
    Win*[q] $\leftarrow$ (0, #{$q \xrightarrow{u}$});
    if Win[q'] **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  }
  else (* **reevaluate** *)
    Win*[q] $\leftarrow$ **Update**(Win*[q]) ;
    if (Win*[q] = $(k, 0) \wedge k \geq 1$) **then** {
      Waiting $\leftarrow$ Waiting $\cup$ Depend[q];
      Win[q] $\leftarrow$ 1;
    }
    if Win[q'] = 0 **then** Depend[q'] $\leftarrow$ Depend[q'] $\cup \{e\}$;
  endif
endwhile

Liu & Smolka Algorithm [Liu & Smolka'98]



$\longrightarrow$ Current
$\longrightarrow$ Explored Forwards
Dashed = in Waiting List

$$e = (\ell_2, c_3, \ell_3)$$

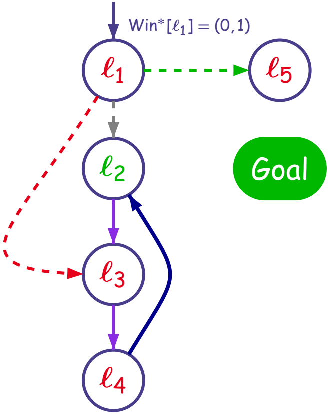
Initialization:

Passed $\leftarrow \{q_0\}$;
Waiting $\leftarrow \{(q_0, \alpha, q') \mid \alpha \in Act\ q \xrightarrow{\alpha} q'\}$;
Win$[q_0] \leftarrow (q_0 \in Goal\ ?\ 1 : 0)$;
Depend$[q_0] \leftarrow \emptyset$;

Main:

while $((Waiting \neq \emptyset) \wedge Win[q_0] \neq 1))$ **do**
  $e = (q, \alpha, q') \leftarrow$ pop(Waiting);
  if $q' \notin$ Passed **then** {
    Passed $\leftarrow$ Passed $\cup \{q'\}$;
    Depend$[q'] \leftarrow \{(q, \alpha, q')\}$;
    Win$[q'] \leftarrow (q' \in Goal\ ?\ 1 : 0)$;
    Waiting $\leftarrow$ Waiting $\cup \{(q', \alpha, q'') \mid q' \xrightarrow{\alpha} q''\}$;
    Win$^*[q] \leftarrow (0, \#\{q \xrightarrow{u}\})$;
    if Win$[q']$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  }
  else (* reevaluate *)
    Win$^*[q] \leftarrow$ **Update**(Win$^*[q]$) ;
    if (Win$^*[q] = (k, 0) \wedge k \geq 1$) **then** {
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[q]$;
      Win$[q] \leftarrow 1$;
    }
    if Win$[q'] = 0$ **then** Depend$[q'] \leftarrow$ Depend$[q'] \cup \{e\}$;
  endif
endwhile

Liu & Smolka Algorithm [Liu & Smolka'98]



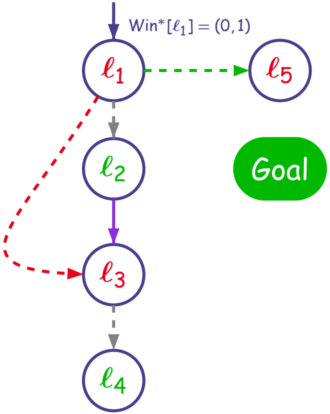
$\longrightarrow$ Current
$\longrightarrow$ Explored Forwards
Dashed = in Waiting List

$$e = (\ell_4, c_5, \ell_2)$$

Initialization:
Passed $\leftarrow \{q_0\}$;
Waiting $\leftarrow \{(q_0, \alpha, q') \mid \alpha \in Act\ q \xrightarrow{\alpha} q'\}$;
Win$[q_0] \leftarrow (q_0 \in Goal\ ?\ 1 : 0)$;
Depend$[q_0] \leftarrow \emptyset$;

Main:
while ((Waiting $\neq \emptyset) \wedge$ Win$[q_0] \neq 1$)) **do**
  $e = (q, \alpha, q') \leftarrow$ pop(Waiting);
  if $q' \notin$ Passed **then** {
    Passed $\leftarrow$ Passed $\cup \{q'\}$;
    Depend$[q'] \leftarrow \{(q, \alpha, q')\}$;
    Win$[q'] \leftarrow (q' \in Goal\ ?\ 1 : 0)$;
    Waiting $\leftarrow$ Waiting $\cup \{(q', \alpha, q'') \mid q' \xrightarrow{\alpha} q''\}$;
    Win$^*[q] \leftarrow (0, \#\{q \xrightarrow{u}\})$;
    if Win$[q']$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  }
  else (* reevaluate *)
    Win$^*[q] \leftarrow$ **Update**(Win$^*[q]$) ;
    if (Win$^*[q] = (k, 0) \wedge k \geq 1$) **then** {
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[q]$;
      Win$[q] \leftarrow 1$;
    }
    if Win$[q'] = 0$ **then** Depend$[q'] \leftarrow$ Depend$[q'] \cup \{e\}$;
  endif
endwhile

Liu & Smolka Algorithm [Liu & Smolka'98]



→ Current
→ Explored Forwards
Dashed = in Waiting List

$e = (\ell_4, c_5, \ell_2)$

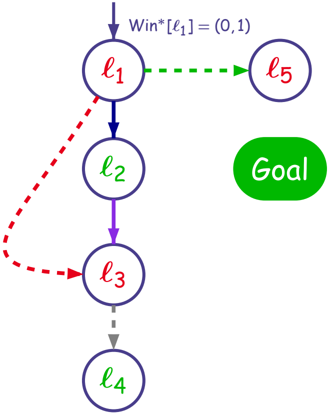
Initialization:

Passed $\leftarrow \{q_0\}$;
Waiting $\leftarrow \{(q_0, a, q') \mid a \in Act\ q \xrightarrow{a} q'\}$;
Win$[q_0] \leftarrow (q_0 \in Goal\ ?\ 1 : 0)$;
Depend$[q_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ Win$[q_0] \neq 1$)) **do**
  $e = (q, a, q') \leftarrow$ pop(Waiting);
  if $q' \notin$ Passed **then** {
    Passed $\leftarrow$ Passed $\cup \{q'\}$;
    Depend$[q'] \leftarrow \{(q, a, q')\}$;
    Win$[q'] \leftarrow (q' \in Goal\ ?\ 1 : 0)$;
    Waiting $\leftarrow$ Waiting $\cup \{(q', a, q'') \mid q' \xrightarrow{a} q''\}$;
    Win*$[q] \leftarrow (0, \#\{q \xrightarrow{u}\})$;
    if Win$[q']$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  }
  else (* reevaluate *)
    Win*$[q] \leftarrow$ **Update**(Win*$[q]$) ;
    if (Win*$[q] = (k, 0) \wedge k \geq 1$) **then** {
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[q]$;
      Win$[q] \leftarrow 1$;
    }
    if Win$[q'] = 0$ **then** Depend$[q'] \leftarrow$ Depend$[q'] \cup \{e\}$;
  endif
endwhile

Liu & Smolka Algorithm [Liu & Smolka'98]



$\longrightarrow$ Current

$\longrightarrow$ Explored Forwards

Dashed = in Waiting List

$$e = (\ell_1, c_1, \ell_2)$$

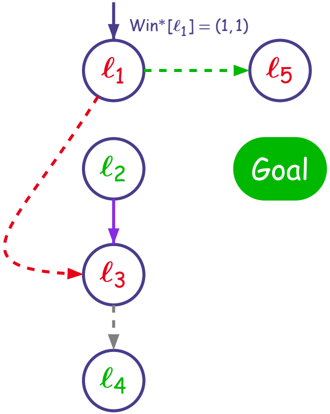
Initialization:

Passed $\leftarrow \{q_0\}$;
Waiting $\leftarrow \{(q_0, a, q') \mid a \in Act\ q \xrightarrow{a} q'\}$;
$Win[q_0] \leftarrow (q_0 \in Goal\ ?\ 1 : 0)$;
$Depend[q_0] \leftarrow \emptyset$;

Main:

while $((Waiting \neq \emptyset) \wedge Win[q_0] \neq 1))$ **do**
$e = (q, a, q') \leftarrow pop(Waiting)$;
if $q' \notin Passed$ **then** {
Passed $\leftarrow$ Passed $\cup \{q'\}$;
$Depend[q'] \leftarrow \{(q, a, q')\}$;
$Win[q'] \leftarrow (q' \in Goal\ ?\ 1 : 0)$;
Waiting $\leftarrow$ Waiting $\cup \{(q', a, q'') \mid q' \xrightarrow{a} q''\}$;
$Win^*[q] \leftarrow (0, \#\{q \xrightarrow{u}\})$;
if $Win[q']$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
}
else (* reevaluate *)
$Win^*[q] \leftarrow$ **Update**$(Win^*[q])$;
if $(Win^*[q] = (k, 0) \wedge k \geq 1)$ **then** {
Waiting $\leftarrow$ Waiting $\cup$ Depend[q];
$Win[q] \leftarrow 1$;
}
if $Win[q'] = 0$ **then** $Depend[q'] \leftarrow Depend[q'] \cup \{e\}$;
endif
endwhile

Liu & Smolka Algorithm [Liu & Smolka'98]



$\longrightarrow$ Current
$\longrightarrow$ Explored Forwards
Dashed = in Waiting List

$$e = (\ell_1, c_1, \ell_2)$$

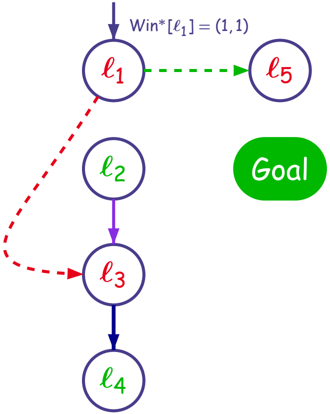
Initialization:

Passed $\leftarrow \{q_0\}$;
Waiting $\leftarrow \{(q_0, a, q') \mid a \in Act\ q \xrightarrow{a} q'\}$;
Win$[q_0] \leftarrow (q_0 \in Goal\ ?\ 1 : 0)$;
Depend$[q_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ Win$[q_0] \neq 1$)) **do**
 $e = (q, a, q') \leftarrow$ pop(Waiting);
 if $q' \notin$ Passed **then** {
  Passed $\leftarrow$ Passed $\cup \{q'\}$;
  Depend$[q'] \leftarrow \{(q, a, q')\}$;
  Win$[q'] \leftarrow (q' \in Goal\ ?\ 1 : 0)$;
  Waiting $\leftarrow$ Waiting $\cup \{(q', a, q'') \mid q' \xrightarrow{a} q''\}$;
  Win$^*[q] \leftarrow (0, \#\{q \xrightarrow{u}\})$;
  if Win$[q']$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
 }
 else (* reevaluate *)
  Win$^*[q] \leftarrow$ **Update**(Win$^*[q]$) ;
  if (Win$^*[q] = (k, 0) \wedge k \geq 1$) **then** {
   Waiting $\leftarrow$ Waiting $\cup$ Depend$[q]$;
   Win$[q] \leftarrow 1$;
  }
  if Win$[q'] = 0$ **then** Depend$[q'] \leftarrow$ Depend$[q'] \cup \{e\}$;
 endif
endwhile

Liu & Smolka Algorithm [Liu & Smolka'98]



$\longrightarrow$ Current
$\longrightarrow$ Explored Forwards
Dashed = in Waiting List

$$e = (\ell_3, c_4, \ell_4)$$

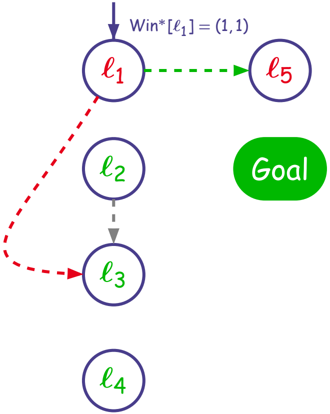
Initialization:

Passed $\leftarrow \{q_0\}$;
Waiting $\leftarrow \{(q_0, \alpha, q') \mid \alpha \in Act\ q \xrightarrow{\alpha} q'\}$;
Win$[q_0] \leftarrow (q_0 \in Goal\ ?\ 1 : 0)$;
Depend$[q_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ Win$[q_0] \neq 1$)) **do**
  $e = (q, \alpha, q') \leftarrow$ pop(Waiting);
  if $q' \notin$ Passed **then** {
    Passed $\leftarrow$ Passed $\cup \{q'\}$;
    Depend$[q'] \leftarrow \{(q, \alpha, q')\}$;
    Win$[q'] \leftarrow (q' \in Goal\ ?\ 1 : 0)$;
    Waiting $\leftarrow$ Waiting $\cup \{(q', \alpha, q'') \mid q' \xrightarrow{\alpha} q''\}$;
    Win$^*[q] \leftarrow (0, \#\{q \xrightarrow{u}\})$;
    if Win$[q']$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  }
  else (* reevaluate *)
    Win$^*[q] \leftarrow$ **Update**(Win$^*[q]$) ;
    if (Win$^*[q] = (k, 0) \wedge k \geq 1$) **then** {
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[q]$;
      Win$[q] \leftarrow 1$;
    }
    if Win$[q'] = 0$ **then** Depend$[q'] \leftarrow$ Depend$[q'] \cup \{e\}$;
  endif
endwhile

Liu & Smolka Algorithm [Liu & Smolka'98]



$\longrightarrow$ Current
$\longrightarrow$ Explored Forwards
Dashed = in Waiting List

$$e = (\ell_3, c_4, \ell_4)$$

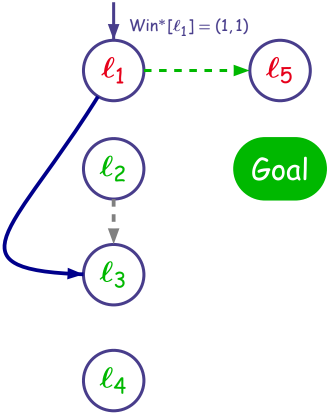
Initialization:

Passed $\leftarrow \{q_0\}$;
Waiting $\leftarrow \{(q_0, \alpha, q') \mid \alpha \in Act\ q \xrightarrow{\alpha} q'\}$;
Win$[q_0] \leftarrow (q_0 \in Goal\ ?\ 1 : 0)$;
Depend$[q_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ Win$[q_0] \neq 1$)) **do**
  $e = (q, \alpha, q') \leftarrow$ pop(Waiting);
  if $q' \notin$ Passed **then** {
    Passed $\leftarrow$ Passed $\cup \{q'\}$;
    Depend$[q'] \leftarrow \{(q, \alpha, q')\}$;
    Win$[q'] \leftarrow (q' \in Goal\ ?\ 1 : 0)$;
    Waiting $\leftarrow$ Waiting $\cup \{(q', \alpha, q'') \mid q' \xrightarrow{\alpha} q''\}$;
    Win*$[q] \leftarrow (0, \#\{q \xrightarrow{u}\})$;
    if Win$[q']$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  }
  else (* reevaluate *)
    Win*$[q] \leftarrow$ **Update**(Win*$[q]$) ;
    if (Win*$[q] = (k, 0) \wedge k \geq 1$) **then** {
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[q]$;
      Win$[q] \leftarrow 1$;
    }
    if Win$[q'] = 0$ **then** Depend$[q'] \leftarrow$ Depend$[q'] \cup \{e\}$;
  endif
endwhile

Liu & Smolka Algorithm [Liu & Smolka'98]



$\longrightarrow$ Current
$\longrightarrow$ Explored Forwards
Dashed = in Waiting List

$$e = (\ell_3, c_4, \ell_4)$$

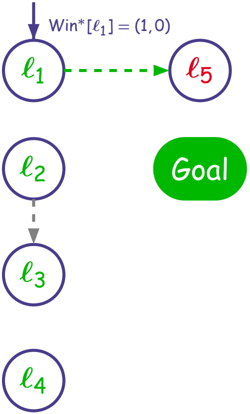
Initialization:

Passed $\leftarrow \{q_0\}$;
Waiting $\leftarrow \{(q_0, \alpha, q') \mid \alpha \in Act\ q \xrightarrow{\alpha} q'\}$;
$Win[q_0] \leftarrow (q_0 \in Goal\ ?\ 1 : 0)$;
$Depend[q_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ $Win[q_0] \neq 1$)) **do**
  $e = (q, \alpha, q') \leftarrow$ pop(Waiting);
  if $q' \not\in$ Passed **then** {
    Passed $\leftarrow$ Passed $\cup \{q'\}$;
    $Depend[q'] \leftarrow \{(q, \alpha, q')\}$;
    $Win[q'] \leftarrow (q' \in Goal\ ?\ 1 : 0)$;
    Waiting $\leftarrow$ Waiting $\cup \{(q', \alpha, q'') \mid q' \xrightarrow{\alpha} q''\}$;
    $Win^*[q] \leftarrow (0, \#\{q \xrightarrow{u}\})$;
    if $Win[q']$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  }
  else (* reevaluate *)
    $Win^*[q] \leftarrow$ **Update**$(Win^*[q])$;
    if $(Win^*[q] = (k, 0) \wedge k \geq 1)$ **then** {
      Waiting $\leftarrow$ Waiting $\cup$ $Depend[q]$;
      $Win[q] \leftarrow 1$;
    }
    if $Win[q'] = 0$ **then** $Depend[q'] \leftarrow Depend[q'] \cup \{e\}$;
  endif
endwhile

Liu & Smolka Algorithm [Liu & Smolka'98]



$\longrightarrow$ Current

$\longrightarrow$ Explored Forwards

Dashed = in Waiting List

$$e = (\ell_3, c_4, \ell_4)$$

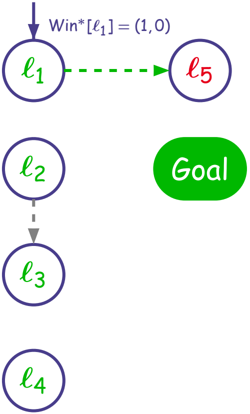
Initialization:

Passed $\leftarrow \{q_0\}$;
Waiting $\leftarrow \{(q_0, a, q') \mid a \in Act\ q \xrightarrow{a} q'\}$;
Win$[q_0] \leftarrow (q_0 \in Goal\ ?\ 1 : 0)$;
Depend$[q_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ Win$[q_0] \neq 1$)) **do**
  $e = (q, a, q') \leftarrow$ pop(Waiting);
  if $q' \notin$ Passed **then** {
    Passed $\leftarrow$ Passed $\cup \{q'\}$;
    Depend$[q'] \leftarrow \{(q, a, q')\}$;
    Win$[q'] \leftarrow (q' \in Goal\ ?\ 1 : 0)$;
    Waiting $\leftarrow$ Waiting $\cup \{(q', a, q'') \mid q' \xrightarrow{a} q''\}$;
    Win$^*[q] \leftarrow (0, \#\{q \xrightarrow{u}\})$;
    if Win$[q']$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  }
  else (* reevaluate *)
    Win$^*[q] \leftarrow$ **Update**(Win$^*[q]$) ;
    if (Win$^*[q] = (k, 0) \wedge k \geq 1$) **then** {
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[q]$;
      Win$[q] \leftarrow 1$;
    }
    if Win$[q'] = 0$ **then** Depend$[q'] \leftarrow$ Depend$[q'] \cup \{e\}$;
  endif
endwhile

Liu & Smolka Algorithm [Liu & Smolka'98]



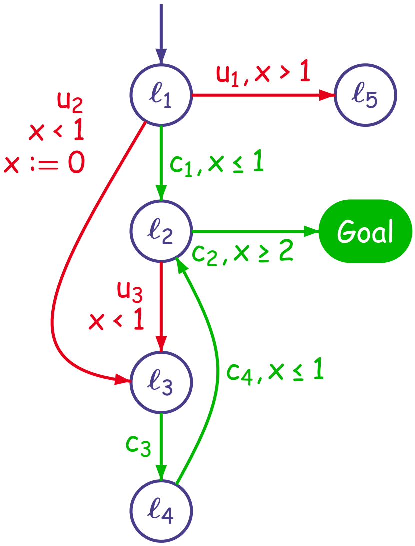
$\longrightarrow$ Current
$\longrightarrow$ Explored Forwards
Dashed = in Waiting List

$$e = (\ell_3, c_4, \ell_4)$$

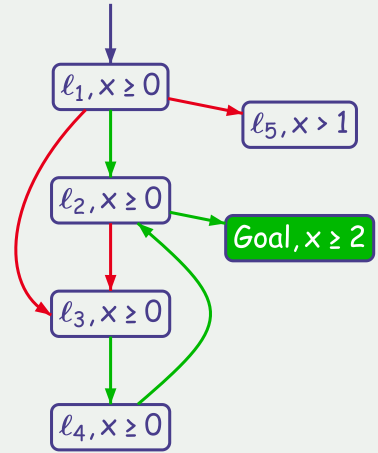
Initialization:
Passed $\leftarrow \{q_0\}$;
Waiting $\leftarrow \{(q_0, \alpha, q') \mid \alpha \in Act\ q \xrightarrow{\alpha} q'\}$;
$Win[q_0] \leftarrow (q_0 \in Goal\ ?\ 1 : 0)$;
$Depend[q_0] \leftarrow \emptyset$;

Main:
while ((Waiting $\neq \emptyset$) $\wedge$ $Win[q_0] \neq 1$)) **do**
  $e = (q, \alpha, q') \leftarrow$ pop(Waiting);
  if $q' \notin$ Passed **then** {
    Passed $\leftarrow$ Passed $\cup \{q'\}$;
    $Depend[q'] \leftarrow \{(q, \alpha, q')\}$;
    $Win[q'] \leftarrow (q' \in Goal\ ?\ 1 : 0)$;
    Waiting $\leftarrow$ Waiting $\cup \{(q', \alpha, q'') \mid q' \xrightarrow{\alpha} q''\}$;
    $Win^*[q] \leftarrow (0, \#\{q \xrightarrow{u}\})$;
    if $Win[q']$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  }
  else (* reevaluate *)
    $Win^*[q] \leftarrow$ **Update**($Win^*[q]$) ;
    if ($Win^*[q] = (k, 0) \wedge k \geq 1$) **then** {
      Waiting $\leftarrow$ Waiting $\cup$ $Depend[q]$;
      $Win[q] \leftarrow 1$;
    }
    if $Win[q'] = 0$ **then** $Depend[q'] \leftarrow Depend[q'] \cup \{e\}$;
  endif
endwhile

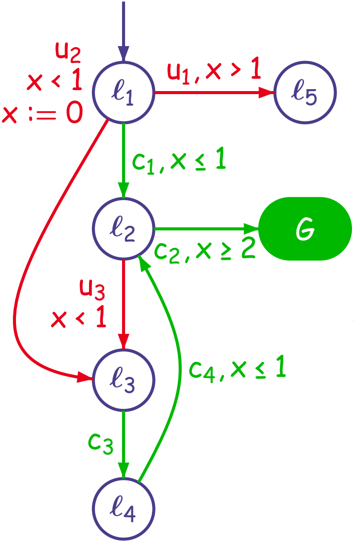
On-The-Fly algorithm for Timed Games (1)



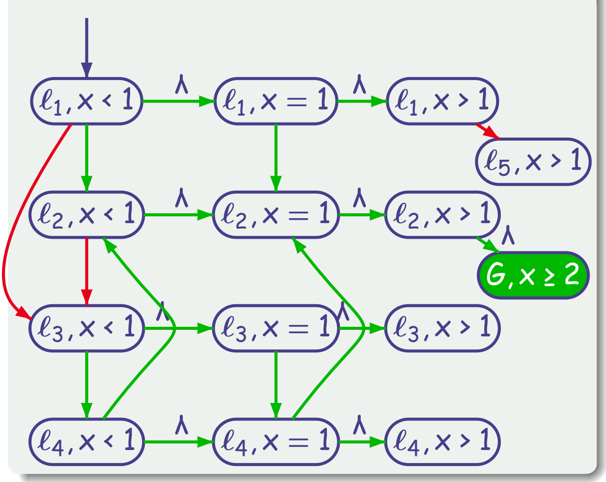
Using the Simulation Graph



Second Try (2) [Altisen & Tripakis'99, Altisen & Tripakis'02]



Stable Partitionning



Towards a True On-The-Fly Algorithm

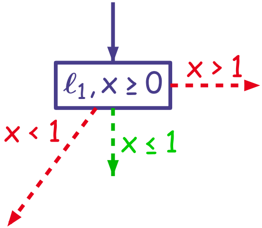
Our Approach:

- Write a Symbolic version of Liu & Smolka
- Use Symbolic states and Transitions
- Apply this to Timed Games

Key issues to be adressed:

- Symbolic States can be partially winning
  compared to finite state games where 0 or 1
- When to propagate backwards ?
- Termination, Complexity ?

Liu & Smolka for Timed Games



Initialization:

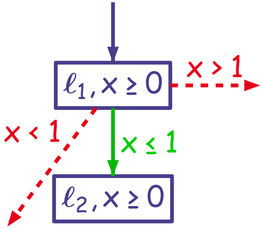
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, a, S') \mid S' = Post_a(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ (($\ell_0, 0) \notin$ Win$[S_0]$)) **do**
  $e = (S, a, S') \leftarrow$ pop(Waiting);
  if $S' \notin$ Passed **then**
    Passed $\leftarrow$ Passed $\cup \{S'\}$;
    Depend$[S'] \leftarrow \{(S, a, S')\}$;
    Win$[S'] \leftarrow S' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
    Waiting $\leftarrow$ Waiting $\cup \{(S', a, S'') \mid S'' = Post_a(S')^{\nearrow}\}$;
    if Win$[S'] \neq \emptyset$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  else **(* reevaluate *)**
    Win$^* \leftarrow$ Pred$_t$(Win$[S] \cup \bigcup_{S \xrightarrow{c} T}$ cPred(Win$[T]$),
      $\bigcup_{S \xrightarrow{u} T}$ uPred($T \setminus$ Win$[T]$)) $\cap S$;
    if (Win$[S] \subsetneq$ Win*) **then**
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[S]$; Win$[S] \leftarrow$ Win*;
    Depend$[S'] \leftarrow$ Depend$[S'] \cup \{e\}$;
  endif
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Initialization:

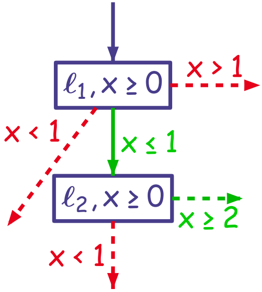
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ ((ℓ_0, 0) $\notin$ Win$[S_0]$)) **do**
 e = $(S, \alpha, S') \leftarrow$ pop(Waiting);
 if $S' \notin$ Passed **then**
  Passed $\leftarrow$ Passed $\cup \{S'\}$;
  Depend$[S'] \leftarrow \{(S, \alpha, S')\}$;
  Win$[S'] \leftarrow S' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
  Waiting $\leftarrow$ Waiting $\cup \{(S', \alpha, S'') \mid S'' = Post_\alpha(S')^{\nearrow}\}$;
  if Win$[S'] \neq \emptyset$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
 else (* reevaluate *)
  Win$^* \leftarrow$Pred$_t$(Win$[S] \cup \bigcup_{S \xrightarrow{c} T}$ cPred(Win$[T]$),
   $\bigcup_{S \xrightarrow{u} T}$ uPred($T \setminus$ Win$[T]$)) $\cap S$;
  if (Win$[S] \subsetneq$ Win*) **then**
   Waiting $\leftarrow$ Waiting $\cup$ Depend$[S]$; Win$[S] \leftarrow$ Win*;
  Depend$[S'] \leftarrow$ Depend$[S'] \cup \{e\}$;
 endif
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Initialization:

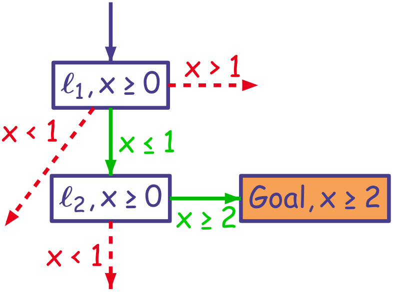
$\quad$ Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
$\quad$ Waiting $\leftarrow \{(S_0, a, S') \mid S' = Post_a(S_0)^{\nearrow}\}$;
$\quad$ Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
$\quad$ Depend$[S_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ ((ℓ_0, 0) $\notin$ Win$[S_0]$)) **do**
$\quad$ $e = (S, a, S') \leftarrow$ pop(Waiting);
$\quad$ **if** $S' \notin$ Passed **then**
$\quad\quad$ Passed $\leftarrow$ Passed $\cup \{S'\}$;
$\quad\quad$ Depend$[S'] \leftarrow \{(S, a, S')\}$;
$\quad\quad$ Win$[S'] \leftarrow S' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
$\quad\quad$ Waiting $\leftarrow$ Waiting $\cup \{(S', a, S'') \mid S'' = Post_a(S')^{\nearrow}\}$;
$\quad\quad$ **if** Win$[S'] \neq \emptyset$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
$\quad$ **else (* reevaluate *)**
$\quad\quad$ Win$^* \leftarrow$ Pred$_t$(Win$[S] \cup \bigcup_{S \xrightarrow{c} T}$ cPred(Win$[T]$),
$\quad\quad\quad \bigcup_{S \xrightarrow{u} T}$ uPred($T \setminus$ Win$[T]$)) $\cap S$;
$\quad\quad$ **if** (Win$[S] \subsetneq$ Win*) **then**
$\quad\quad\quad$ Waiting $\leftarrow$ Waiting $\cup$ Depend$[S]$; Win$[S] \leftarrow$ Win*;
$\quad\quad$ Depend$[S'] \leftarrow$ Depend$[S'] \cup \{e\}$;
$\quad$ **endif**
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Initialization:

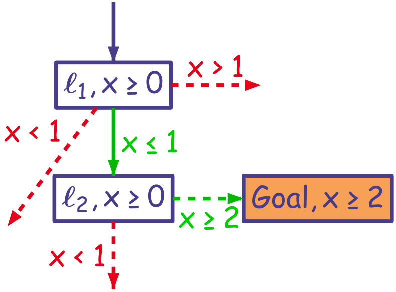
$Passed \leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
$Waiting \leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
$Win[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
$Depend[S_0] \leftarrow \emptyset$;

Main:

while $((Waiting \neq \emptyset) \wedge ((\ell_0, 0) \notin Win[S_0]))$ **do**
  $e = (S, \alpha, S') \leftarrow pop(Waiting)$;
  if $S' \notin Passed$ **then**
    $Passed \leftarrow Passed \cup \{S'\}$;
    $Depend[S'] \leftarrow \{(S, \alpha, S')\}$;
    $Win[S'] \leftarrow S' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
    $Waiting \leftarrow Waiting \cup \{(S', \alpha, S'') \mid S'' = Post_\alpha(S')^{\nearrow}\}$;
    if $Win[S'] \neq \emptyset$ **then** $Waiting \leftarrow Waiting \cup \{e\}$;
  else (* reevaluate *)
    $Win^* \leftarrow Pred_t(Win[S] \cup \bigcup_{S \xrightarrow{c} T} cPred(Win[T]),$
      $\bigcup_{S \xrightarrow{u} T} uPred(T \setminus Win[T])) \cap S$;
    if $(Win[S] \subsetneq Win^*)$ **then**
      $Waiting \leftarrow Waiting \cup Depend[S]$; $Win[S] \leftarrow Win^*$;
    $Depend[S'] \leftarrow Depend[S'] \cup \{e\}$;
  endif
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Initialization:

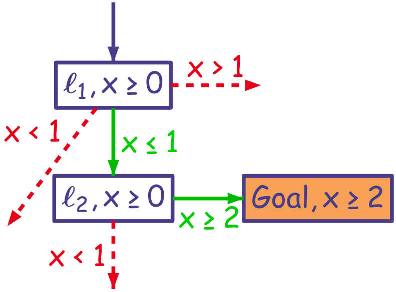
$\quad$ Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
$\quad$ Waiting $\leftarrow \{(S_0, a, S') \mid S' = Post_a(S_0)^{\nearrow}\}$;
$\quad$ Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
$\quad$ Depend$[S_0] \leftarrow \emptyset$;

Main:

while $((\text{Waiting} \neq \emptyset) \wedge ((\ell_0, 0) \not\in \text{Win}[S_0]))$ **do**
$\quad e = (S, a, S') \leftarrow$ pop(Waiting);
$\quad$ **if** $S' \not\in$ Passed **then**
$\quad\quad$ Passed $\leftarrow$ Passed $\cup \{S'\}$;
$\quad\quad$ Depend$[S'] \leftarrow \{(S, a, S')\}$;
$\quad\quad$ Win$[S'] \leftarrow S' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
$\quad\quad$ Waiting $\leftarrow$ Waiting $\cup \{(S', a, S'') \mid S'' = Post_a(S')^{\nearrow}\}$;
$\quad\quad$ **if** Win$[S'] \neq \emptyset$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
$\quad$ **else (* reevaluate *)**
$\quad\quad$ $\text{Win}^* \leftarrow \text{Pred}_t(\text{Win}[S] \cup \bigcup_{S \xrightarrow{c} T} \text{cPred}(\text{Win}[T]), \bigcup_{S \xrightarrow{u} T} \text{uPred}(T \setminus \text{Win}[T])) \cap S$;
$\quad\quad$ **if** $(\text{Win}[S] \subsetneq \text{Win}^*)$ **then**
$\quad\quad\quad$ Waiting $\leftarrow$ Waiting $\cup$ Depend$[S]$; Win$[S] \leftarrow$ Win*;
$\quad\quad$ Depend$[S'] \leftarrow$ Depend$[S'] \cup \{e\}$;
$\quad$ **endif**
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Initialization:

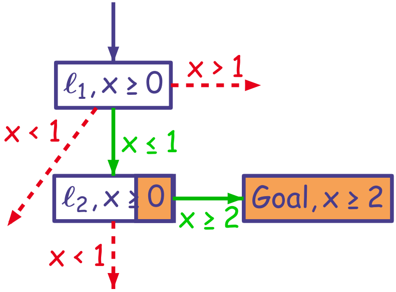
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, a, S') \mid S' = Post_a(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ (($\ell_0, 0$) $\notin$ Win$[S_0]$)) **do**
  $e = (S, a, S') \leftarrow$ pop(Waiting);
  if $S' \notin$ Passed **then**
    Passed $\leftarrow$ Passed $\cup \{S'\}$;
    Depend$[S'] \leftarrow \{(S, a, S')\}$;
    Win$[S'] \leftarrow S' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
    Waiting $\leftarrow$ Waiting $\cup \{(S', a, S'') \mid S'' = Post_a(S')^{\nearrow}\}$;
    if Win$[S'] \neq \emptyset$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  else (* reevaluate *)
    Win$^* \leftarrow$Pred$_t$(Win$[S] \cup \bigcup_{S \xrightarrow{c} T}$ cPred(Win$[T]$),
      $\bigcup_{S \xrightarrow{u} T}$ uPred($T \setminus$ Win$[T]$)) $\cap S$;
    if (Win$[S] \subsetneq$ Win*) **then**
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[S]$; Win$[S] \leftarrow$ Win*;
    Depend$[S'] \leftarrow$ Depend$[S'] \cup \{e\}$;
  endif
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Initialization:

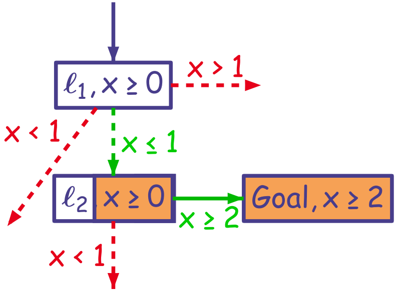
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, a, S') \mid S' = Post_a(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ (($\ell_0, 0$) $\notin$ Win$[S_0]$)) **do**
  $e = (S, a, S') \leftarrow$ pop(Waiting);
  if $S' \notin$ Passed **then**
    Passed $\leftarrow$ Passed $\cup \{S'\}$;
    Depend$[S'] \leftarrow \{(S, a, S')\}$;
    Win$[S'] \leftarrow S' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
    Waiting $\leftarrow$ Waiting $\cup \{(S', a, S'') \mid S'' = Post_a(S')^{\nearrow}\}$;
    if Win$[S'] \neq \emptyset$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  else (* reevaluate *)
    Win$^* \leftarrow$ Pred$_t$(Win$[S] \cup \bigcup_{S \xrightarrow{c} T}$ cPred(Win$[T]$), $\bigcup_{S \xrightarrow{u} T}$ uPred($T \setminus$ Win$[T]$)) $\cap S$;
    if (Win$[S] \subsetneq$ Win*) **then**
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[S]$; Win$[S] \leftarrow$ Win*;
    Depend$[S'] \leftarrow$ Depend$[S'] \cup \{e\}$;
  endif
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Initialization:

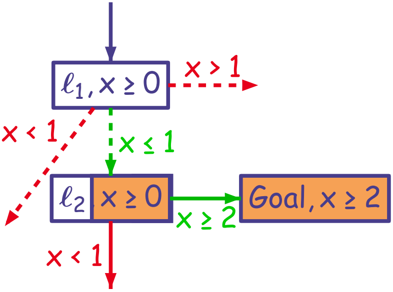
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, a, S') \mid S' = Post_a(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ ((ℓ_0, 0) $\notin$ Win$[S_0]$)) **do**
  $e = (S, a, S') \leftarrow$ pop(Waiting);
  if $S' \notin$ Passed **then**
    Passed $\leftarrow$ Passed $\cup \{S'\}$;
    Depend$[S'] \leftarrow \{(S, a, S')\}$;
    Win$[S'] \leftarrow S' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
    Waiting $\leftarrow$ Waiting $\cup \{(S', a, S'') \mid S'' = Post_a(S')^{\nearrow}\}$;
    if Win$[S'] \neq \emptyset$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  else (* reevaluate *)
    Win$^* \leftarrow$ Pred$_t$(Win$[S] \cup \bigcup_{S \xrightarrow{c} T}$ cPred(Win$[T]$),
      $\bigcup_{S \xrightarrow{u} T}$ uPred($T \setminus$ Win$[T]$)) $\cap S$;
    if (Win$[S] \subsetneq$ Win*) **then**
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[S]$; Win$[S] \leftarrow$ Win*;
    Depend$[S'] \leftarrow$ Depend$[S'] \cup \{e\}$;
  endif
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Initialization:

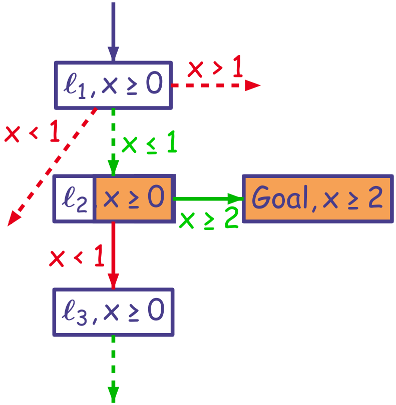
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, a, S') \mid S' = Post_a(S_0)^{\nearrow}\}$;
$\mathrm{Win}[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
$\mathrm{Depend}[S_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ ($(\ell_0, 0) \notin \mathrm{Win}[S_0]$)) **do**
 $e = (S, a, S') \leftarrow$ pop(Waiting);
 if $S' \notin$ Passed **then**
  Passed $\leftarrow$ Passed $\cup \{S'\}$;
  $\mathrm{Depend}[S'] \leftarrow \{(S, a, S')\}$;
  $\mathrm{Win}[S'] \leftarrow S' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
  Waiting $\leftarrow$ Waiting $\cup \{(S', a, S'') \mid S'' = Post_a(S')^{\nearrow}\}$;
  if $\mathrm{Win}[S'] \neq \emptyset$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
 else (* reevaluate *)
  $\mathrm{Win}^* \leftarrow \mathrm{Pred}_t(\mathrm{Win}[S] \cup \bigcup_{S \xrightarrow{c} T} \mathrm{cPred}(\mathrm{Win}[T]),$
   $\bigcup_{S \xrightarrow{u} T} \mathrm{uPred}(T \setminus \mathrm{Win}[T])) \cap S$;
  if $(\mathrm{Win}[S] \subsetneq \mathrm{Win}^*)$ **then**
   Waiting $\leftarrow$ Waiting $\cup$ Depend[S]; $\mathrm{Win}[S] \leftarrow \mathrm{Win}^*$;
  $\mathrm{Depend}[S'] \leftarrow \mathrm{Depend}[S'] \cup \{e\}$;
 endif
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Initialization:

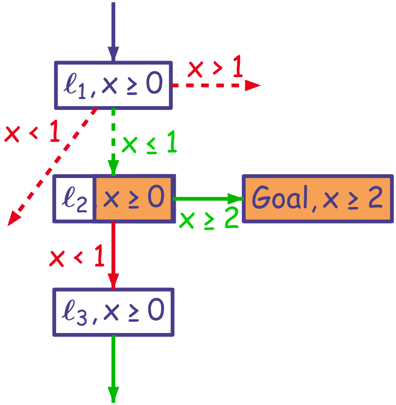
$\mathrm{Passed} \leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
$\mathrm{Waiting} \leftarrow \{(S_0, \alpha, S') \mid S' = \mathit{Post}_\alpha(S_0)^{\nearrow}\}$;
$\mathrm{Win}[S_0] \leftarrow S_0 \cap (\{\mathrm{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
$\mathrm{Depend}[S_0] \leftarrow \emptyset$;

Main:

while $((\mathrm{Waiting} \neq \emptyset) \wedge ((\ell_0, 0) \not\in \mathrm{Win}[S_0]))$ **do**
$e = (S, \alpha, S') \leftarrow \mathrm{pop}(\mathrm{Waiting})$;
if $S' \not\in \mathrm{Passed}$ **then**
$\mathrm{Passed} \leftarrow \mathrm{Passed} \cup \{S'\}$;
$\mathrm{Depend}[S'] \leftarrow \{(S, \alpha, S')\}$;
$\mathrm{Win}[S'] \leftarrow S' \cap (\{\mathrm{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
$\mathrm{Waiting} \leftarrow \mathrm{Waiting} \cup \{(S', \alpha, S'') \mid S'' = \mathit{Post}_\alpha(S')^{\nearrow}\}$;
if $\mathrm{Win}[S'] \neq \emptyset$ **then** $\mathrm{Waiting} \leftarrow \mathrm{Waiting} \cup \{e\}$;
else (* reevaluate *)
$\mathrm{Win}^* \leftarrow \mathrm{Pred}_t(\mathrm{Win}[S] \cup \bigcup_{S \xrightarrow{c} T} \mathrm{cPred}(\mathrm{Win}[T]), \bigcup_{S \xrightarrow{u} T} \mathrm{uPred}(T \setminus \mathrm{Win}[T])) \cap S$;
if $(\mathrm{Win}[S] \subsetneq \mathrm{Win}^*)$ **then**
$\mathrm{Waiting} \leftarrow \mathrm{Waiting} \cup \mathrm{Depend}[S]$; $\mathrm{Win}[S] \leftarrow \mathrm{Win}^*$;
$\mathrm{Depend}[S'] \leftarrow \mathrm{Depend}[S'] \cup \{e\}$;
endif
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Initialization:

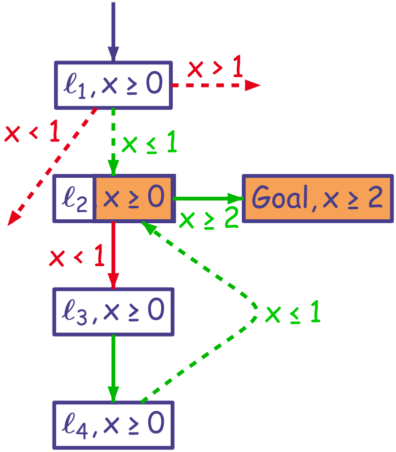
$\quad Passed \leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
$\quad Waiting \leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
$\quad Win[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
$\quad Depend[S_0] \leftarrow \emptyset$;

Main:

while $((Waiting \neq \emptyset) \wedge ((\ell_0, 0) \notin Win[S_0]))$ **do**
$\quad e = (S, \alpha, S') \leftarrow pop(Waiting)$;
$\quad$ **if** $S' \notin Passed$ **then**
$\quad\quad Passed \leftarrow Passed \cup \{S'\}$;
$\quad\quad Depend[S'] \leftarrow \{(S, \alpha, S')\}$;
$\quad\quad Win[S'] \leftarrow S' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
$\quad\quad Waiting \leftarrow Waiting \cup \{(S', \alpha, S'') \mid S'' = Post_\alpha(S')^{\nearrow}\}$;
$\quad\quad$ **if** $Win[S'] \neq \emptyset$ **then** $Waiting \leftarrow Waiting \cup \{e\}$;
$\quad$ **else (* reevaluate *)**
$\quad\quad Win^* \leftarrow Pred_t(Win[S] \cup \bigcup_{S \xrightarrow{c} T} cPred(Win[T]),$
$\quad\quad\quad \bigcup_{S \xrightarrow{u} T} uPred(T \setminus Win[T])) \cap S$;
$\quad\quad$ **if** $(Win[S] \subsetneq Win^*)$ **then**
$\quad\quad\quad Waiting \leftarrow Waiting \cup Depend[S]$; $Win[S] \leftarrow Win^*$;
$\quad\quad Depend[S'] \leftarrow Depend[S'] \cup \{e\}$;
$\quad$ **endif**
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Initialization:

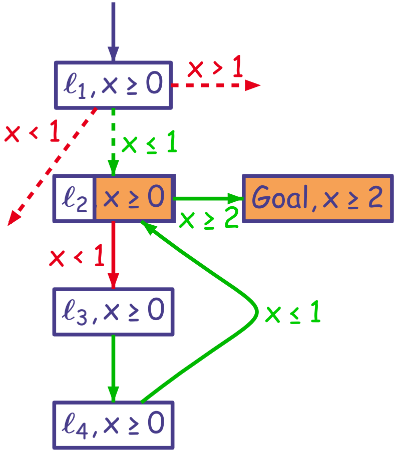
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{\mathrm{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

Main:

while $((\mathrm{Waiting} \neq \emptyset) \wedge ((\ell_0, 0) \notin \mathrm{Win}[S_0]))$ **do**
  $e = (S, \alpha, S') \leftarrow \mathrm{pop}(\mathrm{Waiting})$;
  if $S' \notin$ Passed **then**
    Passed $\leftarrow$ Passed $\cup \{S'\}$;
    Depend$[S'] \leftarrow \{(S, \alpha, S')\}$;
    Win$[S'] \leftarrow S' \cap (\{\mathrm{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
    Waiting $\leftarrow$ Waiting $\cup \{(S', \alpha, S'') \mid S'' = Post_\alpha(S')^{\nearrow}\}$;
    if Win$[S'] \neq \emptyset$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  else (* reevaluate *)
    $\mathrm{Win}^* \leftarrow \mathrm{Pred}_t(\mathrm{Win}[S] \cup \bigcup_{S \xrightarrow{c} T} \mathrm{cPred}(\mathrm{Win}[T]), \bigcup_{S \xrightarrow{u} T} \mathrm{uPred}(T \setminus \mathrm{Win}[T])) \cap S$;
    if $(\mathrm{Win}[S] \subsetneq \mathrm{Win}^*)$ **then**
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[S]$; Win$[S] \leftarrow \mathrm{Win}^*$;
    Depend$[S'] \leftarrow$ Depend$[S'] \cup \{e\}$;
  endif
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Initialization:

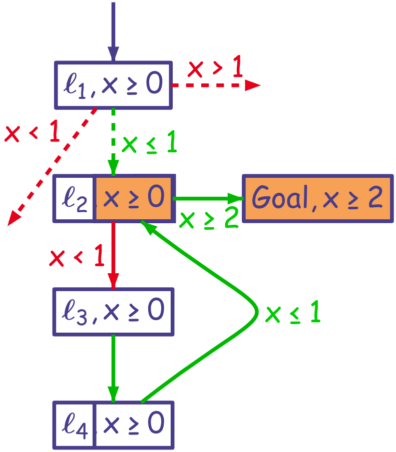
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ (($\ell_0, 0) \notin$ Win$[S_0]$)) **do**
  $e = (S, \alpha, S') \leftarrow$ pop(Waiting);
  if $S' \notin$ Passed **then**
    Passed $\leftarrow$ Passed $\cup \{S'\}$;
    Depend$[S'] \leftarrow \{(S, \alpha, S')\}$;
    Win$[S'] \leftarrow S' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
    Waiting $\leftarrow$ Waiting $\cup \{(S', \alpha, S'') \mid S'' = Post_\alpha(S')^{\nearrow}\}$;
    if Win$[S'] \neq \emptyset$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  else (* reevaluate *)
    Win* $\leftarrow$Pred$_t$(Win$[S] \cup \bigcup_{S \xrightarrow{c} T}$ cPred(Win$[T]$),
      $\bigcup_{S \xrightarrow{u} T}$ uPred($T \setminus$ Win$[T]$)) $\cap S$;
    if (Win$[S] \subsetneq$ Win*) **then**
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[S]$; Win$[S] \leftarrow$ Win*;
    Depend$[S'] \leftarrow$ Depend$[S'] \cup \{e\}$;
  endif
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Initialization:

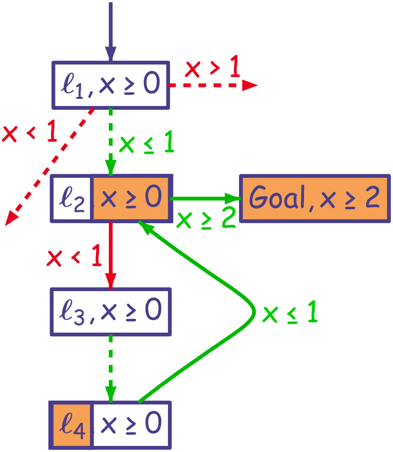
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ (($\ell_0, 0) \not\in$ Win$[S_0]$)) **do**
  $e = (S, \alpha, S') \leftarrow$ pop(Waiting);
  if $S' \not\in$ Passed **then**
    Passed $\leftarrow$ Passed $\cup \{S'\}$;
    Depend$[S'] \leftarrow \{(S, \alpha, S')\}$;
    Win$[S'] \leftarrow S' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
    Waiting $\leftarrow$ Waiting $\cup \{(S', \alpha, S'') \mid S'' = Post_\alpha(S')^{\nearrow}\}$;
    if Win$[S'] \neq \emptyset$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  else (* reevaluate *)
    Win$^* \leftarrow$Pred$_t$(Win$[S] \cup \bigcup_{S \xrightarrow{c} T}$ cPred(Win$[T]$),
        $\bigcup_{S \xrightarrow{u} T}$ uPred($T \setminus$ Win$[T]$)) $\cap S$;
    if (Win$[S] \subsetneq$ Win*) **then**
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[S]$; Win$[S] \leftarrow$ Win*;
    Depend$[S'] \leftarrow$ Depend$[S'] \cup \{e\}$;
  endif
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Initialization:

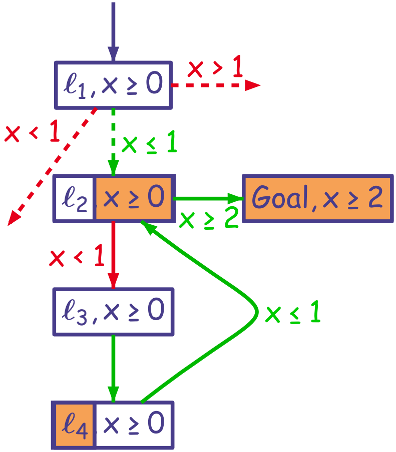
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, a, S') \mid S' = Post_a(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ ((ℓ_0, 0) $\notin$ Win$[S_0]$)) **do**
  $e = (S, a, S') \leftarrow$ pop(Waiting);
  if $S' \notin$ Passed **then**
    Passed $\leftarrow$ Passed $\cup \{S'\}$;
    Depend$[S'] \leftarrow \{(S, a, S')\}$;
    Win$[S'] \leftarrow S' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
    Waiting $\leftarrow$ Waiting $\cup \{(S', a, S'') \mid S'' = Post_a(S')^{\nearrow}\}$;
    if Win$[S'] \neq \emptyset$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  else (* reevaluate *)
    Win$^* \leftarrow$ Pred$_t$(Win$[S] \cup \bigcup_{S \xrightarrow{c} T}$ cPred(Win$[T]$),
      $\bigcup_{S \xrightarrow{u} T}$ uPred($T \setminus$ Win$[T]$)) $\cap S$;
    if (Win$[S] \subsetneq$ Win*) **then**
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[S]$; Win$[S] \leftarrow$ Win*;
    Depend$[S'] \leftarrow$ Depend$[S'] \cup \{e\}$;
  endif
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Skip algorithm

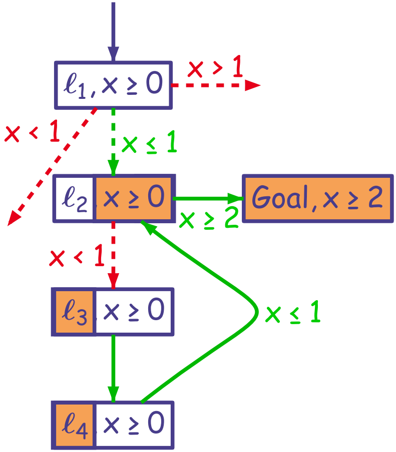
Initialization:

Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ $((\ell_0, 0) \notin$ Win$[S_0]$)) **do**
  $e = (S, \alpha, S') \leftarrow$ pop(Waiting);
  if $S' \notin$ Passed **then**
    Passed $\leftarrow$ Passed $\cup \{S'\}$;
    Depend$[S'] \leftarrow \{(S, \alpha, S')\}$;
    Win$[S'] \leftarrow S' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
    Waiting $\leftarrow$ Waiting $\cup \{(S', \alpha, S'') \mid S'' = Post_\alpha(S')^{\nearrow}\}$;
    if Win$[S'] \neq \emptyset$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  else (* reevaluate *)
    Win$^* \leftarrow$ Pred$_t$(Win$[S] \cup \bigcup_{S \xrightarrow{c} T}$ cPred(Win$[T]$),
      $\bigcup_{S \xrightarrow{u} T}$ uPred($T \setminus$ Win$[T]$)) $\cap S$;
    if (Win$[S] \subsetneq$ Win*) **then**
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[S]$; Win$[S] \leftarrow$ Win*;
    Depend$[S'] \leftarrow$ Depend$[S'] \cup \{e\}$;
  endif
endwhile

Liu & Smolka for Timed Games



Initialization:

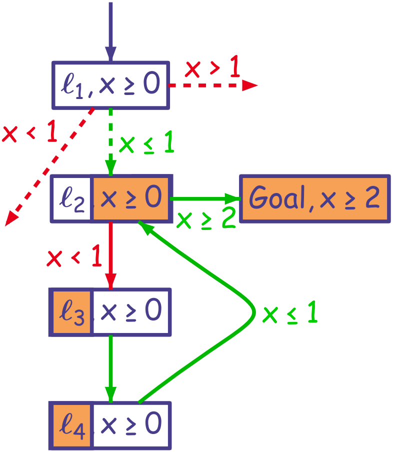
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

Main:

while $((\text{Waiting} \neq \emptyset) \wedge ((\ell_0, 0) \notin \text{Win}[S_0]))$ **do**
  $e = (S, \alpha, S') \leftarrow$ pop(Waiting);
  if $S' \notin$ Passed **then**
    Passed $\leftarrow$ Passed $\cup \{S'\}$;
    Depend$[S'] \leftarrow \{(S, \alpha, S')\}$;
    Win$[S'] \leftarrow S' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
    Waiting $\leftarrow$ Waiting $\cup \{(S', \alpha, S'') \mid S'' = Post_\alpha(S')^{\nearrow}\}$;
    if Win$[S'] \neq \emptyset$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  else (* reevaluate *)
    $\text{Win}^* \leftarrow \text{Pred}_t(\text{Win}[S] \cup \bigcup_{S \xrightarrow{c} T} \text{cPred}(\text{Win}[T]),$
    $\bigcup_{S \xrightarrow{u} T} \text{uPred}(T \setminus \text{Win}[T])) \cap S$;
    if $(\text{Win}[S] \subsetneq \text{Win}^*)$ **then**
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[S]$; Win$[S] \leftarrow$ Win*;
    Depend$[S'] \leftarrow$ Depend$[S'] \cup \{e\}$;
  endif
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Initialization:

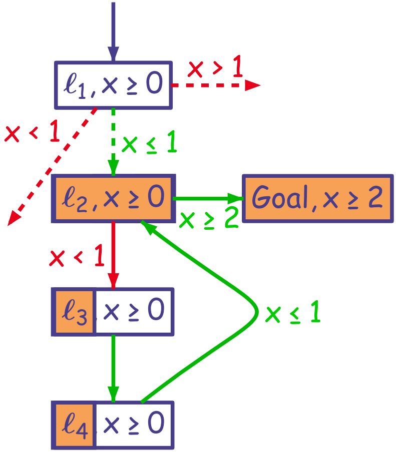
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, a, S') \mid S' = Post_a(S_0)^{\nearrow}\}$;
Win[S_0] $\leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Depend[S_0] $\leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ (($\ell_0, 0) \not\in$ Win[S_0])) **do**
  $e = (S, a, S') \leftarrow$ pop(Waiting);
  if $S' \not\in$ Passed **then**
    Passed $\leftarrow$ Passed $\cup \{S'\}$;
    Depend[S'] $\leftarrow \{(S, a, S')\}$;
    Win[S'] $\leftarrow S' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
    Waiting $\leftarrow$ Waiting $\cup \{(S', a, S'') \mid S'' = Post_a(S')^{\nearrow}\}$;
    if Win[S'] $\neq \emptyset$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  else (* reevaluate *)
    Win* $\leftarrow \mathrm{Pred}_t(\mathrm{Win}[S] \cup \bigcup_{S \xrightarrow{c} T} \mathrm{cPred}(\mathrm{Win}[T]),$
      $\bigcup_{S \xrightarrow{u} T} \mathrm{uPred}(T \setminus \mathrm{Win}[T])) \cap S$;
    if (Win[S] $\subsetneq$ Win*) **then**
      Waiting $\leftarrow$ Waiting $\cup$ Depend[S]; Win[S] $\leftarrow$ Win*;
    Depend[S'] $\leftarrow$ Depend[S'] $\cup \{e\}$;
  endif
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Initialization:

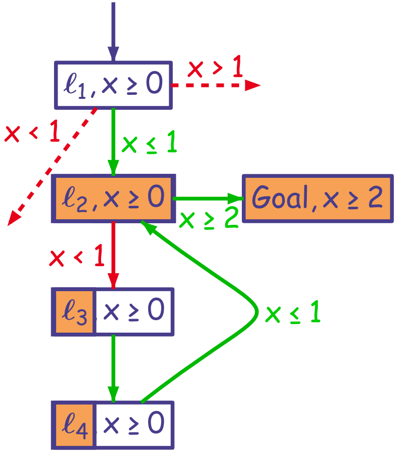
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, a, S') \mid S' = Post_a(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

Main:

```
while ((Waiting ≠ ∅) ∧ ((ℓ0, 0) ∉ Win[S0])) do
    e = (S, a, S') ← pop(Waiting);
    if S' ∉ Passed then
      Passed ← Passed ∪ {S'};
      Depend[S'] ← {(S, a, S')};
      Win[S'] ← S' ∩ ({Goal} × ℝ^X_≥0);
      Waiting ← Waiting ∪ {(S', a, S'') | S'' = Post_a(S')↗};
      if Win[S'] ≠ ∅ then Waiting ← Waiting ∪ {e};
    else (* reevaluate *)
      Win* ←Pred_t(Win[S] ∪ ⋃_{S -c-> T} cPred(Win[T]),
                            ⋃_{S -u-> T} uPred(T \ Win[T])) ∩ S;
      if (Win[S] ⊊ Win*) then
        Waiting ← Waiting ∪ Depend[S]; Win[S] ← Win*;
      Depend[S'] ← Depend[S'] ∪ {e};
    endif
endwhile
```

Skip algorithm

Liu & Smolka for Timed Games



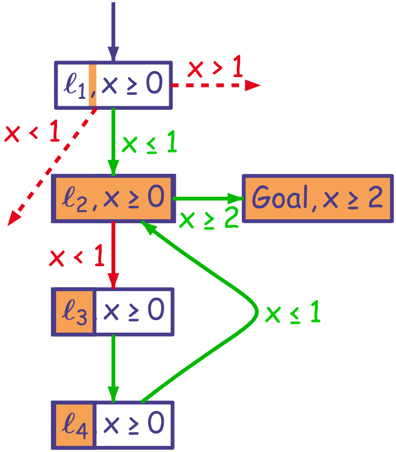
Initialization:

Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, a, S') \mid S' = Post_a(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

Main:
while ((Waiting $\neq \emptyset$) $\wedge$ (($\ell_0, 0) \notin$ Win$[S_0]$)) **do**
$e = (S, a, S') \leftarrow$ pop(Waiting);
if $S' \notin$ Passed **then**
Passed $\leftarrow$ Passed $\cup \{S'\}$;
Depend$[S'] \leftarrow \{(S, a, S')\}$;
Win$[S'] \leftarrow S' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Waiting $\leftarrow$ Waiting $\cup \{(S', a, S'') \mid S'' = Post_a(S')^{\nearrow}\}$;
if Win$[S'] \neq \emptyset$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
else (* reevaluate *)
Win$^* \leftarrow$Pred$_t$(Win$[S] \cup \bigcup_{S \xrightarrow{c} T}$ cPred(Win$[T]$),
$\bigcup_{S \xrightarrow{u} T}$ uPred($T \setminus$ Win$[T]$)) $\cap S$;
if (Win$[S] \subsetneq$ Win*) **then**
Waiting $\leftarrow$ Waiting $\cup$ Depend$[S]$; Win$[S] \leftarrow$ Win*;
Depend$[S'] \leftarrow$ Depend$[S'] \cup \{e\}$;
endif
endwhile

▸▸ Skip algorithm

Liu & Smolka for Timed Games



Initialization:

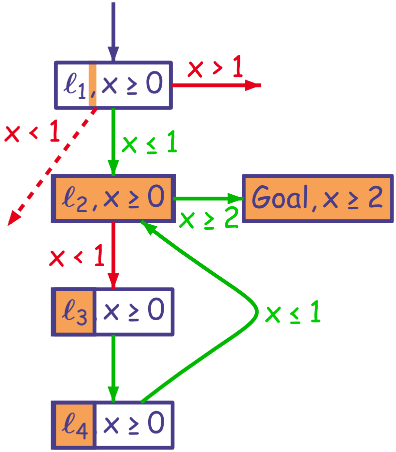
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ (($\ell_0, 0) \notin$ Win$[S_0]$)) **do**
  $e = (S, \alpha, S') \leftarrow$ pop(Waiting);
  if $S' \notin$ Passed **then**
    Passed $\leftarrow$ Passed $\cup \{S'\}$;
    Depend$[S'] \leftarrow \{(S, \alpha, S')\}$;
    Win$[S'] \leftarrow S' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
    Waiting $\leftarrow$ Waiting $\cup \{(S', \alpha, S'') \mid S'' = Post_\alpha(S')^{\nearrow}\}$;
    if Win$[S'] \neq \emptyset$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  else (* reevaluate *)
    Win* $\leftarrow \mathrm{Pred}_t(\mathrm{Win}[S] \cup \bigcup_{S \xrightarrow{c} T} \mathrm{cPred}(\mathrm{Win}[T]),$
      $\bigcup_{S \xrightarrow{u} T} \mathrm{uPred}(T \setminus \mathrm{Win}[T])) \cap S$;
    if (Win$[S] \subsetneq$ Win*) **then**
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[S]$; Win$[S] \leftarrow$ Win*;
    Depend$[S'] \leftarrow$ Depend$[S'] \cup \{e\}$;
  endif
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Initialization:

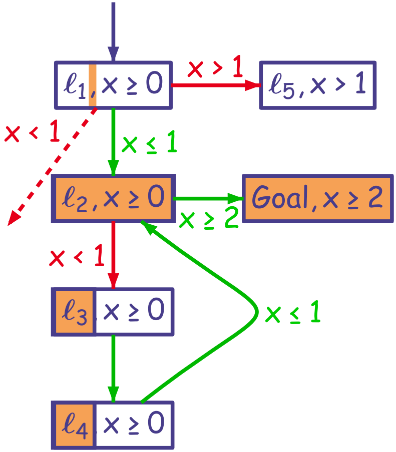
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ ((ℓ_0, 0) $\notin$ Win$[S_0]$)) **do**
  $e = (S, \alpha, S') \leftarrow$ pop(Waiting);
  if $S' \notin$ Passed **then**
    Passed $\leftarrow$ Passed $\cup \{S'\}$;
    Depend$[S'] \leftarrow \{(S, \alpha, S')\}$;
    Win$[S'] \leftarrow S' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
    Waiting $\leftarrow$ Waiting $\cup \{(S', \alpha, S'') \mid S'' = Post_\alpha(S')^{\nearrow}\}$;
    if Win$[S'] \neq \emptyset$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  else (* reevaluate *)
    Win$^* \leftarrow$Pred$_t$(Win$[S] \cup \bigcup_{S \xrightarrow{c} T}$ cPred(Win$[T]$),
      $\bigcup_{S \xrightarrow{u} T}$ uPred($T \setminus$ Win$[T]$)) $\cap S$;
    if (Win$[S] \subsetneq$ Win*) **then**
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[S]$; Win$[S] \leftarrow$ Win*;
    Depend$[S'] \leftarrow$ Depend$[S'] \cup \{e\}$;
  endif
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Initialization:

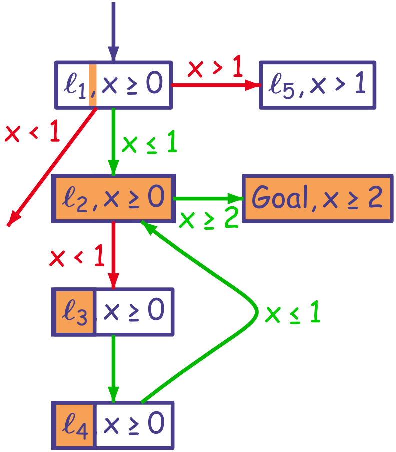
$Passed \leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
$Waiting \leftarrow \{(S_0, a, S') \mid S' = Post_a(S_0)^{\nearrow}\}$;
$Win[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
$Depend[S_0] \leftarrow \emptyset$;

Main:

while $((Waiting \neq \emptyset) \wedge ((\ell_0, 0) \notin Win[S_0]))$ **do**
$e = (S, a, S') \leftarrow pop(Waiting)$;
if $S' \notin Passed$ **then**
$Passed \leftarrow Passed \cup \{S'\}$;
$Depend[S'] \leftarrow \{(S, a, S')\}$;
$Win[S'] \leftarrow S' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
$Waiting \leftarrow Waiting \cup \{(S', a, S'') \mid S'' = Post_a(S')^{\nearrow}\}$;
if $Win[S'] \neq \emptyset$ **then** $Waiting \leftarrow Waiting \cup \{e\}$;
else (* reevaluate *)
$Win^* \leftarrow Pred_t(Win[S] \cup \bigcup_{S \xrightarrow{c} T} cPred(Win[T]),$
$\bigcup_{S \xrightarrow{u} T} uPred(T \setminus Win[T])) \cap S$;
if $(Win[S] \subsetneq Win^*)$ **then**
$Waiting \leftarrow Waiting \cup Depend[S]$; $Win[S] \leftarrow Win^*$;
$Depend[S'] \leftarrow Depend[S'] \cup \{e\}$;
endif
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Initialization:

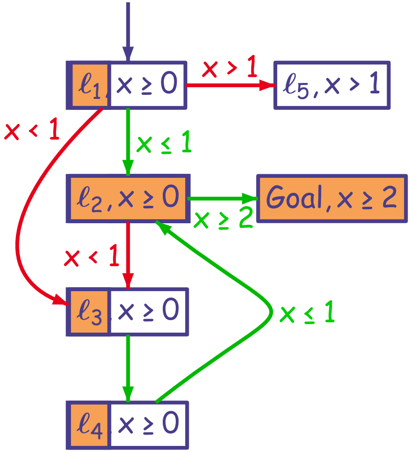
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
Win[S_0] $\leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Depend[S_0] $\leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ ((ℓ_0, 0) $\notin$ Win[S_0])) **do**
  $e = (S, \alpha, S') \leftarrow$ pop(Waiting);
  if $S' \notin$ Passed **then**
    Passed $\leftarrow$ Passed $\cup \{S'\}$;
    Depend[S'] $\leftarrow \{(S, \alpha, S')\}$;
    Win[S'] $\leftarrow S' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
    Waiting $\leftarrow$ Waiting $\cup \{(S', \alpha, S'') \mid S'' = Post_\alpha(S')^{\nearrow}\}$;
    if Win[S'] $\neq \emptyset$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  else (* reevaluate *)
    Win* $\leftarrow$Pred$_t$(Win[S]$\cup$ $\bigcup_{S \xrightarrow{c} T}$ cPred(Win[T]),
      $\bigcup_{S \xrightarrow{u} T}$ uPred(T \ Win[T])) $\cap$ S;
    if (Win[S] $\subsetneq$ Win*) **then**
      Waiting $\leftarrow$ Waiting $\cup$ Depend[S]; Win[S] $\leftarrow$ Win*;
    Depend[S'] $\leftarrow$ Depend[S'] $\cup \{e\}$;
  endif
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Initialization:

Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, a, S') \mid S' = Post_a(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ (($\ell_0, 0) \notin$ Win$[S_0]$)) **do**
  $e = (S, a, S') \leftarrow$ pop(Waiting);
  if $S' \notin$ Passed **then**
    Passed $\leftarrow$ Passed $\cup \{S'\}$;
    Depend$[S'] \leftarrow \{(S, a, S')\}$;
    Win$[S'] \leftarrow S' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
    Waiting $\leftarrow$ Waiting $\cup \{(S', a, S'') \mid S'' = Post_a(S')^{\nearrow}\}$;
    if Win$[S'] \neq \emptyset$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
  else (* reevaluate *)
    Win$^* \leftarrow$ Pred$_t$(Win$[S] \cup \bigcup_{S \xrightarrow{c} T}$ cPred(Win$[T]$),
      $\bigcup_{S \xrightarrow{u} T}$ uPred($T \setminus$ Win$[T]$)) $\cap S$;
    if (Win$[S] \subsetneq$ Win*) **then**
      Waiting $\leftarrow$ Waiting $\cup$ Depend$[S]$; Win$[S] \leftarrow$ Win*;
    Depend$[S'] \leftarrow$ Depend$[S'] \cup \{e\}$;
  endif
endwhile

Skip algorithm

Summary of the Results [Concur'05]

- A True on-the-fly algorithm for reachability control
- Winning Strategies can be computed
- Termination
  A symbolic edge (S, a, T) will be at most (1+ # regions(T)) times in *Waiting* list
- Complexity
  A region may be in many symbolic states
  Our algorithm: Not linear in the size of the region graph hence not theoretically optimal
- However ... seems good in practice with **UPPAAL-TiGA**

Download at **`http://www.cs.aau.dk/~adavid/tiga/`**

Next:

- Reachability Control
- On-the-fly Algorithms for Reachability Control
- **Implementation, Optimizations, Time Optimality**
- Partial Observation

Efficient Implementation of Pred_t

Theorem

The following distribution law holds:

$$\mathsf{Pred}_t(\bigcup_i G_i, \bigcup_j B_j) = \bigcup_i \bigcap_j \mathsf{Pred}_t(G_i, B)$$

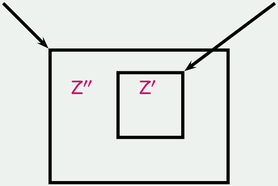
Theorem

If B is a convex set, then:

$$\mathsf{Pred}_t(G, B) = (G^{\swarrow} \setminus B^{\swarrow}) \cup ((G \cap B^{\swarrow}) \setminus B)^{\swarrow}$$

Inclusion Checking & Losing States

Inclusion checking



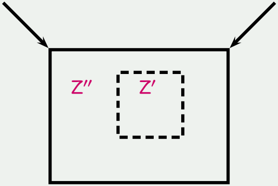
Losing States

Pruning

Main:
while ((Waiting $\neq \emptyset$) $\wedge$ ($s_0 \notin$ Win[S_0])) **do**
    $e = (S, a, S') \longleftarrow$ pop(Waiting);
    if Win[S] $\subsetneq$ S **then**
        if $S' \notin$ Passed **then**
            (...)
        else (* reevaluate *)
            (...)
        endif
    endif
endwhile

Inclusion Checking & Losing States

Inclusion checking



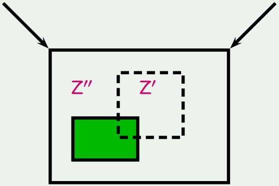
Losing States

Pruning

Main:
while ((Waiting $\neq \emptyset$) $\wedge$ ($s_0 \not\in$ Win[S_0])) do
    $e = (S, \alpha, S') \leftarrow$ pop(Waiting);
    if Win[S] $\subsetneq$ S then
        if $S' \not\in$ Passed then
            (...)
        else (* reevaluate *)
            (...)
        endif
    endif
endwhile

Inclusion Checking & Losing States

Inclusion checking



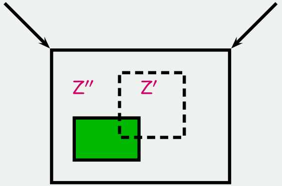
Losing States

Pruning

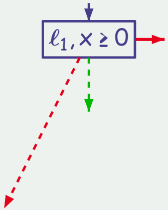
Main:
while ((Waiting $\neq \emptyset$) $\wedge$ ($s_0 \not\in$ Win[S_0])) **do**
    $e = (S, \alpha, S') \longleftarrow$ pop(Waiting);
    if Win[S] $\subsetneq$ S **then**
        if $S' \not\in$ Passed **then**
            (...)
        else (* **reevaluate** *)
            (...)
        endif
    endif
endwhile

Inclusion Checking & Losing States

Inclusion checking



Losing States

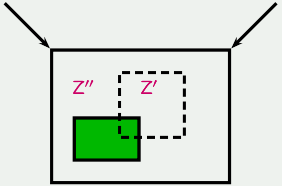


Pruning

<u>Main:</u>
while ((Waiting $\neq \emptyset$) $\wedge$ ($s_0 \not\in$ Win[S_0])) **do**
    $e = (S, a, S') \longleftarrow$ pop(Waiting);
    if Win[S] $\subsetneq$ S **then**
        if $S' \not\in$ Passed **then**
            (...)
        else (* **reevaluate** *)
            (...)
        endif
    endif
endwhile

Inclusion Checking & Losing States

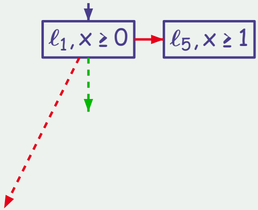
Inclusion checking



Pruning

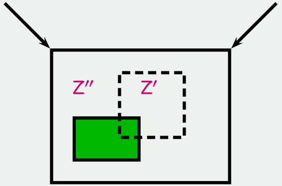
Main:
while ((Waiting $\neq \emptyset$) $\wedge$ ($s_0 \notin$ Win[S_0])) **do**
  $e = (S, a, S') \longleftarrow$ pop(Waiting);
  if Win[S] $\subsetneq$ S **then**
    if $S' \notin$ Passed **then**
      (...)
    else (* **reevaluate** *)
      (...)
    endif
  endif
endwhile

Losing States



Inclusion Checking & Losing States

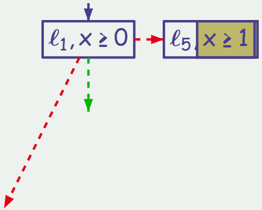
Inclusion checking



Pruning

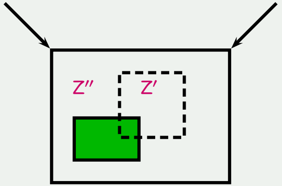
Main:
while ((Waiting $\neq \emptyset$) $\wedge$ ($s_0 \notin$ Win[S_0])) **do**
  $e = (S, a, S') \longleftarrow$ pop(Waiting);
  if Win[S] $\subsetneq$ S **then**
    if $S' \not\subseteq$ Passed **then**
      (...)
    else (* reevaluate *)
      (...)
    endif
  endif
endwhile

Losing States



Inclusion Checking & Losing States

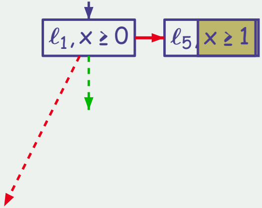
Inclusion checking



Pruning

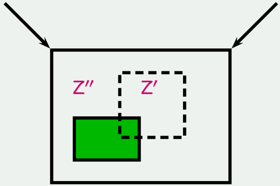
Main:
while ((Waiting $\neq \emptyset$) $\wedge$ ($s_0 \not\in$ Win[S_0])) **do**
  $e = (S, a, S') \longleftarrow$ pop(Waiting);
  if Win[S] $\subsetneq$ S **then**
    if $S' \not\in$ Passed **then**
      $(\ldots)$
    else (* **reevaluate** *)
      $(\ldots)$
    endif
  endif
endwhile

Losing States



Inclusion Checking & Losing States

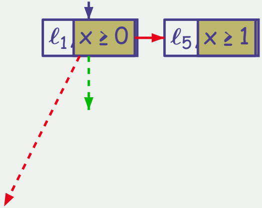
Inclusion checking



Pruning

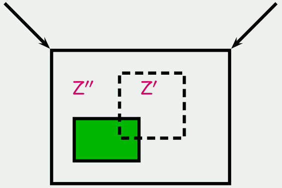
<u>Main:</u>
while ((Waiting $\neq \emptyset$) $\wedge$ ($s_0 \notin$ Win[S_0])) **do**
  $e = (S, a, S') \leftarrow$ pop(Waiting);
  if Win[S] $\subsetneq$ S **then**
    if $S' \notin$ Passed **then**
      (...)
    else (* **reevaluate** *)
      (...)
    endif
  endif
endwhile

Losing States

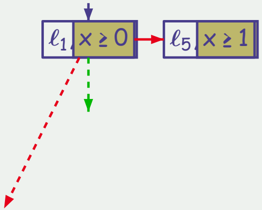


Inclusion Checking & Losing States

Inclusion checking



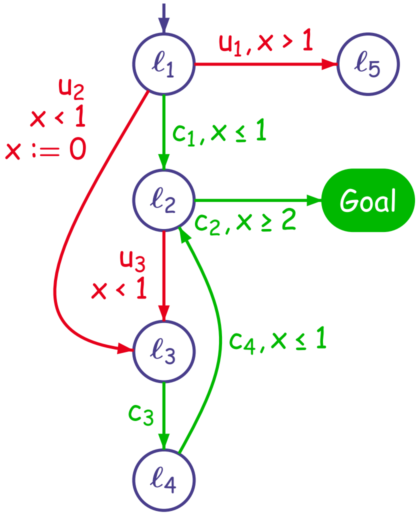
Losing States



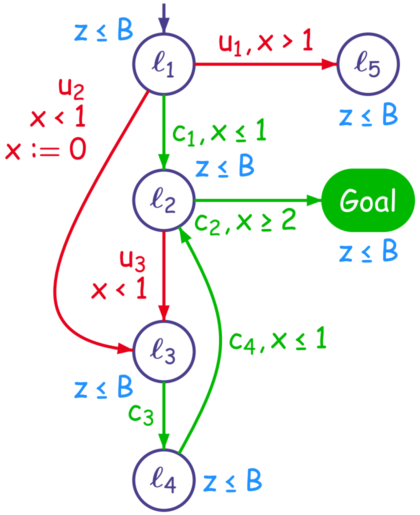
Pruning

Main:
while ((Waiting $\neq \emptyset$) $\wedge$ ($s_0 \notin$ Win[S_0])) **do**
    e = (S, a, S′) ← pop(Waiting);
    if Win[S] $\subsetneq$ S **then**
        if S′ $\not\subseteq$ Passed **then**
            (. . .)
        else (* **reevaluate** *)
            (. . .)
        endif
    endif
endwhile

Time Optimality for Free



Time Optimality for Free

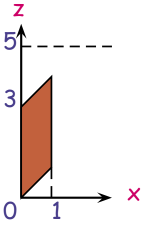


Assume:

- The initial state is winning
- We know an upper bound B of the (optimal) time needed to reach Goal

To compute the optimal time:

- Add a clock z (unconstrained at the beginning)
- Add a global invariant $z \leq B$



Next:

- Reachability Control
- On-the-fly Algorithms for Reachability Control
- Implementation, Optimizations, Time Optimality
- Partial Observation
  - Results About Partial Observation for Timed Games
  - Stuttering Free Observations
  - Knowledge Based Subset Construction

Results for Partial Observation

Partial Observation of Events:

- Discrete Event Systems:
  Partial Obs. = Hiding + Determinization + Full Observation
  [Kupferman & Vardi'99, Reif'84, Arnold et al.'03]
- Timed Automata:
  "Invisible" ε-transition cannot be removed [Bérard et al.'98]
  Timed Automata cannot be determinized [Alur & Dill'94]

Theorem ([Bouyer et al.'03])

Safety and reachability control under partial observation are undecidable.

Theorem ([Bouyer et al.'03])

Control under partial observation with bounded resources is 2EXPTIME-complete.

Results for Partial Observation

Partial Observation of Events:

- Discrete Event Systems:
  Partial Obs. = Hiding + Determinization + Full Observation
  [Kupferman & Vardi'99, Reif'84, Arnold et al.'03]
- Timed Automata:
  "Invisible" ε-transition cannot be removed [Bérard et al.'98]
  Timed Automata cannot be determinized [Alur & Dill'94]

Theorem ([Bouyer et al.'03])

Safety and reachability control under partial observation are undecidable.

Theorem ([Bouyer et al.'03])

Control under partial observation with bounded resources is 2EXPTIME-complete.

Results for Partial Observation

Partial Observation of Events:

- Discrete Event Systems:
  Partial Obs. = Hiding + Determinization + Full Observation
  [Kupferman & Vardi'99, Reif'84, Arnold et al.'03]
- Timed Automata:
  "Invisible" ε-transition cannot be removed [Bérard et al.'98]
  Timed Automata cannot be determinized [Alur & Dill'94]

Theorem ([Bouyer et al.'03])

Safety and reachability control under partial observation are undecidable.

Theorem ([Bouyer et al.'03])

Control under partial observation with bounded resources is 2EXPTIME-complete.

Results for Partial Observation

Partial Observation of Events:

- Discrete Event Systems:
  Partial Obs. = Hiding + Determinization + Full Observation
  [Kupferman & Vardi'99, Reif'84, Arnold et al.'03]
- Timed Automata:
  "Invisible" ε-transition cannot be removed [Bérard et al.'98]
  Timed Automata cannot be determinized [Alur & Dill'94]

Theorem ([Bouyer et al.'03])

Safety and reachability control under partial observation are undecidable.

Theorem ([Bouyer et al.'03])

Control under partial observation with bounded resources is 2EXPTIME-complete.

Results for Partial Observation

Partial Observation of Events:

- Discrete Event Systems:
  Partial Obs. = Hiding + Determinization + Full Observation
  [Kupferman & Vardi'99, Reif'84, Arnold et al.'03]
- Timed Automata:
  "Invisible" ε-transition cannot be removed [Bérard et al.'98]
  Timed Automata cannot be determinized [Alur & Dill'94]

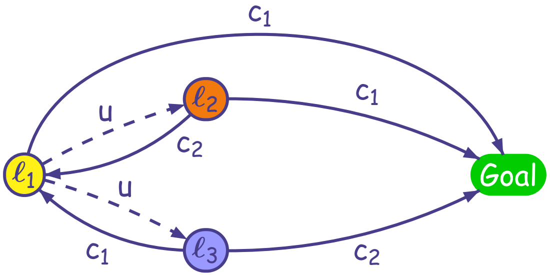
Theorem ([Bouyer et al.'03])

Safety and reachability control under partial observation are undecidable.

Theorem ([Bouyer et al.'03])

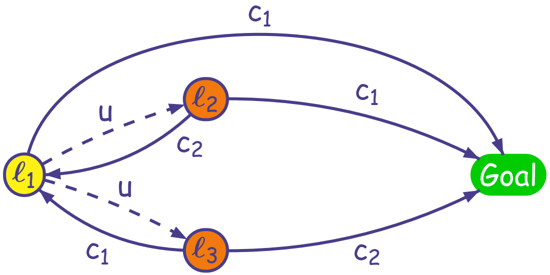
Control under partial observation with bounded resources is 2EXPTIME-complete.

State-Based Partial Observation



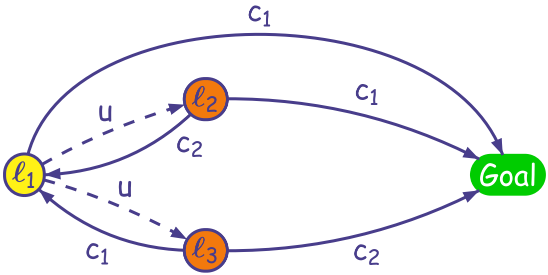
- **Full Observation**: in ℓ_2 do c_1, in ℓ_3 do c_2
- Partial Observation: Partition of the state space e.g. $\ell_2 \equiv \ell_3$ [Chatterjee et al.'06, De Wulf et al.'06]

State-Based Partial Observation



- Full Observation: in ℓ_2 do c_1, in ℓ_3 do c_2
- Partial Observation: Partition of the state space e.g. $\ell_2 \equiv \ell_3$ [Chatterjee et al.'06, De Wulf et al.'06]

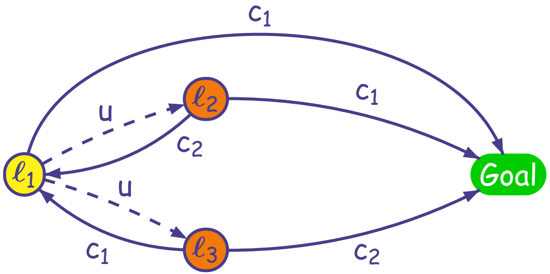
State-Based Partial Observation



- Full Observation: in ℓ_2 do c_1, in ℓ_3 do c_2
- Partial Observation: Partition of the state space e.g. $\ell_2 \equiv \ell_3$ [Chatterjee et al.'06, De Wulf et al.'06]

Impossible to Win

State-Based Partial Observation

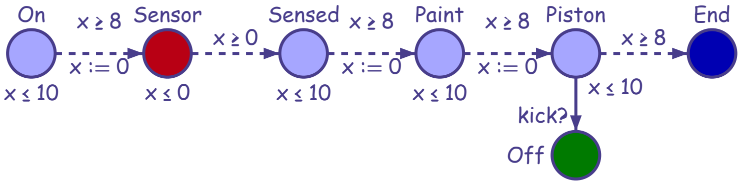


- Full Observation: in ℓ_2 do c_1, in ℓ_3 do c_2
- Partial Observation: Partition of the state space e.g. $\ell_2 \equiv \ell_3$ [Chatterjee et al.'06, De Wulf et al.'06]

Our Aim:

- Extend this framework to timed systems
- Design efficient algorithms for solving this type of games

Example



$$(On,0) \xrightarrow{8} (Sensor,0) \longrightarrow (Sensed,0) \xrightarrow{9} (Paint,0) \xrightarrow{8.7} (Piston,0) \xrightarrow{kick?} Off$$

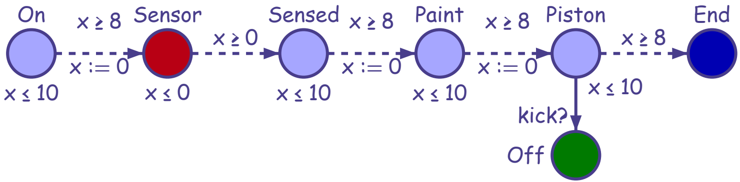
$$\bullet \xrightarrow{8} \bullet \longrightarrow \bullet \xrightarrow{9} \bullet \xrightarrow{8.7} \bullet \xrightarrow{kick?} \bullet$$

Assumption: the controller can only see changes of observations

Stuttering-Free observation: $\bullet \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \bullet$

Must play based on stuttering-free observations

Example



$$(\text{On}, 0) \xrightarrow{8} (\text{Sensor}, 0) \longrightarrow (\text{Sensed}, 0) \xrightarrow{9} (\text{Paint}, 0) \xrightarrow{8.7} (\text{Piston}, 0) \xrightarrow{kick?} \text{Off}$$

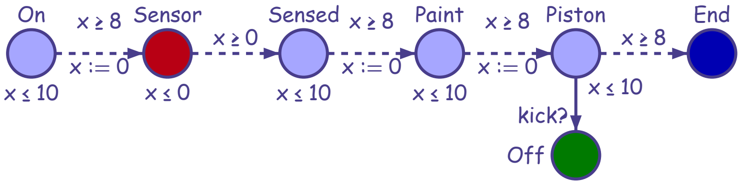
$$\bullet \xrightarrow{8} \bullet \longrightarrow \bullet \xrightarrow{9} \bullet \xrightarrow{8.7} \bullet \xrightarrow{kick?} \bullet$$

Assumption: the controller can only see changes of observations

Stuttering-Free observation: $\bullet \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \bullet$

Must play based on stuttering-free observations

Example



$$(\mathrm{On},0) \xrightarrow{8} (\mathrm{Sensor},0) \longrightarrow (\mathrm{Sensed},0) \xrightarrow{9} (\mathrm{Paint},0) \xrightarrow{8.7} (\mathrm{Piston},0) \xrightarrow{kick?} \mathrm{Off}$$

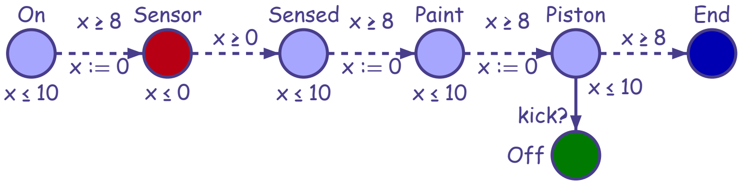
$$\bullet \xrightarrow{8} \bullet \longrightarrow \bullet \xrightarrow{9} \bullet \xrightarrow{8.7} \bullet \xrightarrow{kick?} \bullet$$

Assumption: the controller can only see changes of observations

Stuttering-Free observation: $\bullet \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \bullet$

Must play based on stuttering-free observations

Example



$$(On,0) \xrightarrow{8} (Sensor,0) \longrightarrow (Sensed,0) \xrightarrow{9} (Paint,0) \xrightarrow{8.7} (Piston,0) \xrightarrow{kick?} Off$$

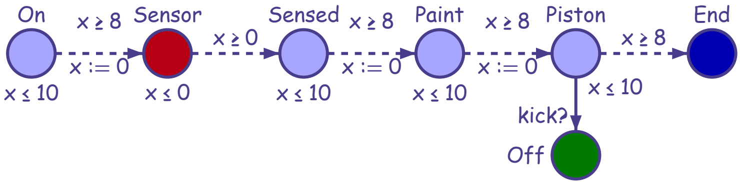
$$\bullet \xrightarrow{8} \bullet \longrightarrow \bullet \xrightarrow{9} \bullet \xrightarrow{8.7} \bullet \xrightarrow{kick?} \bullet$$

Assumption: the controller can only see changes of observations

Stuttering-Free observation: $\bullet \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \bullet$

Must play based on stuttering-free observations

Example



$(On, 0) \xrightarrow{8} (Sensor, 0) \longrightarrow (Sensed, 0) \xrightarrow{9} (Paint, 0) \xrightarrow{8.7} (Piston, 0) \xrightarrow{kick?} Off$

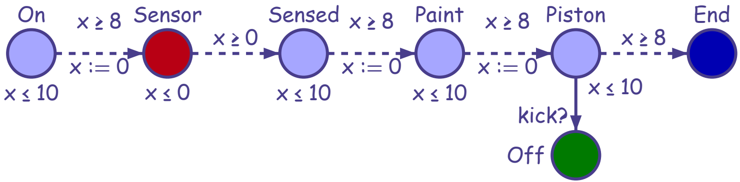
● $\xrightarrow{8}$ ● $\longrightarrow$ ● $\xrightarrow{9}$ ● $\xrightarrow{8.7}$ ● $\xrightarrow{kick?}$ ●

Assumption: the controller can only see **changes** of observations

Stuttering-Free observation: ● $\longrightarrow$ ● $\longrightarrow$ ● $\longrightarrow$ ●

Must play based on **stuttering-free** observations

Example



$$(\mathrm{On},0) \xrightarrow{8} (\mathrm{Sensor},0) \longrightarrow (\mathrm{Sensed},0) \xrightarrow{9} (\mathrm{Paint},0) \xrightarrow{8.7} (\mathrm{Piston},0) \xrightarrow{kick?} \mathrm{Off}$$

$$\bullet \xrightarrow{8} \bullet \longrightarrow \bullet \xrightarrow{9} \bullet \xrightarrow{8.7} \bullet \xrightarrow{kick?} \bullet$$

> Assumption: the controller can only see **changes** of observations

Stuttering-Free observation: $\bullet \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \bullet$

> Must play based on **stuttering-free** observations

Rules of the Game

Given:

- a TGA automaton G
- a finite set of observations $\mathcal{O}$ (maps states to observations)
- a control objective $\Phi \subseteq \mathcal{O}^\omega$

Observation-Based Stuttering Invariant Strategies

f is a OBSI strategy if:

$$\text{Observation}(\rho) \equiv_{\text{stutt}} \text{Observation}(\rho') \text{ implies } f(\rho) = f(\rho')$$

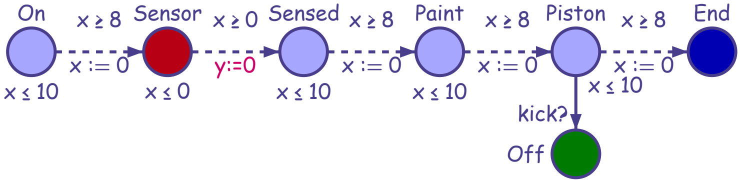
Control under Partial Observation:

Is there an OBSI winning strategy ?

Requirements:

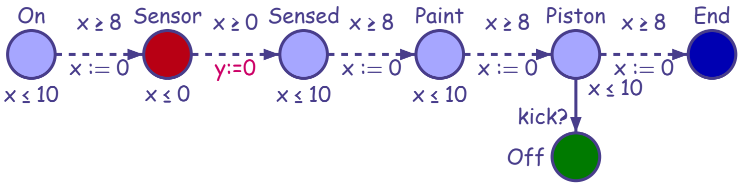
- Observations have special shape:
  must become true at some first instant / partition the state space
  e.g. $20 \leq y < 24$
- Φ is stuttering closed

Knowledge Based Subset Construction

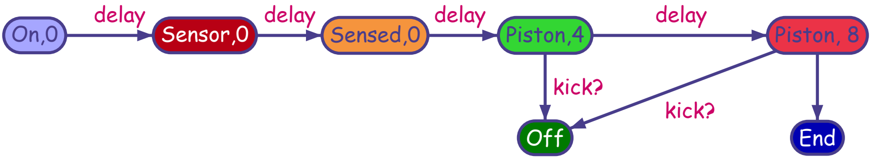


Observations = colors + $y < 20$, $20 \leq y < 24$ and $24 \leq y$

Knowledge Based Subset Construction



Observations = colors + $y < 20$, $20 \leq y < 24$ and $24 \leq y$



Result for Partial Observation

Input:

- a TGA automaton G
- a stuttering closed control objective Φ

Knowledge Game:

1. Build a Knowledge Game Know(G)
   + take care of infinite single-observation runs
2. if G is bounded Know(G) is finite

Theorem

The controller has a winning strategy in (Know(G), Φ) iff it has an OBSI winning strategy in (G, Φ).

Holds for any control objective

Result for Partial Observation

Input:

- a TGA automaton G
- a stuttering closed control objective Φ

Knowledge Game:

1. Build a Knowledge Game Know(G)
   + take care of infinite single-observation runs
2. if G is bounded Know(G) is finite

Theorem

The controller has a winning strategy in (Know(G), Φ) iff it has an OBSI winning strategy in (G, Φ).

Holds for any control objective

Result for Partial Observation

Input:

- a TGA automaton G
- a stuttering closed control objective Φ

Knowledge Game:

1. Build a Knowledge Game Know(G)
   + take care of infinite single-observation runs
2. if G is bounded Know(G) is finite

Theorem

The controller has a winning strategy in (Know(G), Φ) iff it has an OBSI winning strategy in (G, Φ).

Holds for any control objective

Efficient Algorithm for Reachability Game

Initialization:
Passed $\leftarrow \{q_0\}$;
Waiting $\leftarrow \{(q_0, a, q') \mid a \in Act\ q \xrightarrow{a} q'\}$;
Win$[q_0] \leftarrow (q_0 \in Goal\ ?\ 1 : 0)$;
Depend$[q_0] \leftarrow \emptyset$;

Main:
while ((Waiting $\neq \emptyset$) $\wedge$ Win$[q_0] \neq 1$)) **do**
$e = (q, a, q') \leftarrow$ pop(Waiting);
if $q' \notin$ Passed **then** {
Passed $\leftarrow$ Passed $\cup \{q'\}$;
Depend$[q'] \leftarrow \{(q, a, q')\}$;
Win$[q'] \leftarrow (q' \in Goal\ ?\ 1 : 0)$;
Waiting $\leftarrow$ Waiting $\cup \{(q', a, q'') \mid q' \xrightarrow{a} q''\}$;
Win$^*[q] \leftarrow (0, \#\{q \xrightarrow{u}\})$;
if Win$[q']$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
}
else (* **reevaluate** *)
Win$^*[q] \leftarrow$ **Update**(Win$^*[q]$) ;
if (Win$^*[q] = (k, 0) \wedge k \geq 1$) **then** {
Waiting $\leftarrow$ Waiting $\cup$ Depend$[q]$;
Win$[q] \leftarrow 1$;
}
if Win$[q'] = 0$ **then**
Depend$[q'] \leftarrow$ Depend$[q'] \cup \{e\}$;
endif
endwhile

Initialization:
Passed $\leftarrow \{\{s_0\}\}$;
Waiting $\leftarrow \{(\{s_0\}, a, W') \mid a \in \Sigma_1,\ o \in \mathcal{O},\ \ W' = \text{Next}_a(\{s_0\}) \cap o \wedge W' \neq \emptyset\}$;
Win$[\{s_0\}] \leftarrow (\{s_0\} \subseteq \gamma(\text{Goal})\ ?\ 1 : 0)$;
Losing$[\{s_0\}] \leftarrow (\{s_0\} \not\subseteq \gamma(\text{Goal}) \wedge (\text{Waiting} = \emptyset \vee \forall a \in \Sigma_1, \text{Sink}_a(s_0) \neq \emptyset)\ ?\ 1 : 0)$;
Depend$[\{s_0\}] \leftarrow \emptyset$;

Main:
while ((Waiting $\neq \emptyset$) $\wedge$ Win$[\{s_0\}] \neq 1 \wedge$ Losing$[\{s_0\}] \neq 1$)) **do**
$e = (W, a, W') \leftarrow$ pop(Waiting);
if $s' \notin$ Passed **then**
Passed $\leftarrow$ Passed $\cup \{W'\}$;
Depend$[W'] \leftarrow \{(W, a, W')\}$;
Win$[W'] \leftarrow (W' \subseteq \gamma(\text{Goal})\ ?\ 1 : 0)$;
Losing$[W'] \leftarrow (W' \not\subseteq \gamma(\text{Goal}) \wedge \text{Sink}_a(W') \neq \emptyset\ ?\ 1 : 0)$;
if (Losing$[W'] \neq 1$) **then** (* **if losing it is a deadlock state** *)
NewTrans $\leftarrow \{(W', a, W'') \mid a \in \Sigma,\ o \in \mathcal{O},\ \ W' = \text{Next}_a(W) \cap o \wedge W' \neq \emptyset\}$;
if NewTrans $= \emptyset \wedge$ Win$[W'] = 0$ **then** Losing$[W'] \leftarrow 1$;
if (Win$[W'] \vee$ Losing$[W']$) **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
Waiting $\leftarrow$ Waiting $\cup$ NewTrans;
else (* **reevaluate** *)
Win$^* \leftarrow \bigvee_{c \in \text{Enabled}(W)} \bigwedge_{W \xrightarrow{c} W''}$ Win$[W'']$;
if Win* **then**
Waiting $\leftarrow$ Waiting $\cup$ Depend$[W]$; Win$[W] \leftarrow 1$;
Losing$^* \leftarrow \bigwedge_{c \in \text{Enabled}(W)} \bigvee_{W \xrightarrow{c} W''}$ Losing$[W'']$;
if Losing* **then**
Waiting $\leftarrow$ Waiting $\cup$ Depend$[W]$; Losing$[W] \leftarrow 1$;
if (Win$[W'] = 0 \wedge$ Losing$[W'] = 0$) **then** Depend$[W'] \leftarrow$ Depend$[W'] \cup \{e\}$;
endif
endwhile

Conclusion & Ongoing Work

Conclusion

- Efficient algorithm for reachability timed games
  Extends to safety games
- Implementation in UPPAAL-TiGA
  http://www.cs.aau.dk/~adavid/tiga/
- Experiments: Pigs Factory + Jobshop scheduling

Ongoing Work

- On-the-fly algorithm for Büchi objectives
- Compact representation of winning strategies
- Extension to optimal cost computation (reachability)
- Simulation of strategies in UPPAAL-TiGA

References

[Altisen & Tripakis'99] Karine Altisen and Stavros Tripakis.
On-the-fly controller synthesis for discrete and dense-time systems.
In *World Congress on Formal Methods (FM'99)*, volume 1708 of *Lecture Notes in Computer Science*, pages 233–252. Springer, 1999.

[Altisen & Tripakis'02] Karine Altisen and Stavros Tripakis.
Tools for controller synthesis of timed systems.
In *Proc. 2nd Workshop on Real-Time Tools (RT-TOOLS'02)*, 2002.
Proc. published as Technical Report 2002-025, Uppsala University, Sweden.

[Alur & Dill'94] R. Alur and D. Dill.
A theory of timed automata.
Theoretical Computer Science (TCS), 126(2):183–235, 1994.

[Arnold et al.'03] André Arnold, Aymeric Vincent, and Igor Walukiewicz.
Games for synthesis of controllers with partial observation.
Theoretical Computer Science, 303(1):7–34,2003.

[Asarin & Maler'99] E. Asarin and O. Maler.
As soon as possible: Time optimal control for timed automata.
In *Proc. 2nd Int. Work. Hybrid Systems: Computation and Control (HSCC'99)*, volume 1569 of *LNCS*, pages 19–30. Springer, 1999.

[UPPAAL-TiGA'07] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. Larsen, and D. Lime.
Uppaal-tiga: Time for playing games!
In *Proceedings of 19th International Conference on Computer Aided Verification (CAV'07)*, volume 4590 of *LNCS*, pages 121–125, Berlin, Germany, 2007. Springer.

[Bérard et al.'98] B. Berard, A. Petit, V. Diekert and P. Gastin.
Characterization of the Expressive Power of Silent Transitions in Timed Automata.
Fundamenta Informaticae, 36(2-3):145–182, 1998.

References (cont.)

[Bouyer et al.'03] P. Bouyer, D. D'Souza, P. Madhusudan, and A. Petit.
Timed control with partial observability.
In W. A. Hunt, Jr and F. Somenzi, editors, *Proceedings of the 15th International Conference on Computer Aided Verification (CAV'03)*, volume 2725 of *LNCS*, pages 180–192, Boulder, Colorado, USA, July 2003. Springer.

[Bouyer et al.'04] Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim G. Larsen.
Optimal strategies in priced timed game automata.
In *Proc. of the 24th Int. Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'04)*, volume 3328 of *LNCS*, pages 148–160. Springer, 2004.

[Concur'05] F. Cassez, A. David, E. Fleury, K. Larsen, and D. Lime.
Efficient on-the-fly algorithms for the analysis of timed games.
In M. Abadi and L. de Alfaro, editors, *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR'05)*, volume 3653 of *LNCS*, pages 66–80, San Francisco, CA, USA, Aug. 2005. Springer.

[ATVA'07] F. Cassez, A. David, K. Larsen, D. Lime, and J.-F. Raskin.
Timed Control with Observation Based and Stuttering Invariant Strategies.
In *Proc. of the 5th Int. Symp. on Automated Technology for Verif ication and Analysis (ATVA'2007)*, *LNCS*, Tokyo, Oct. 2007. Springer-Verlag.
Forthcoming.

[Chatterjee et al.'06] K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin.
Algorithms for omega-regular games with imperfect information .
In *CSL*, pages 287–302, 2006.

[De Wulf et al.'06] M. De Wulf, L. Doyen, and J.-F. Raskin.
A lattice theory for solving games of imperfect information.
In *HSCC*, pages 153–168, 2006.

References (cont.)

[Kupferman & Vardi'99] O. Kupferman and M. Vardi
Synthesis with incomplete informatio.
In Advances in Temporal Logic (H. Barringer et al., eds.), pages 109–127. Kluwer, 1999.

[Liu & Smolka'98] X. Liu and S. A. Smolka.
Simple Linear-Time Algorithm for Minimal Fixed Points.
In *Proc. 26th Int. Conf. on Automata, Languages and Programming (ICALP'98)*, volume 1443 of *LNCS*, pages 53–66, Aalborg, Denmark, 1998. Springer.

[Maler et al.'95] Oded Maler, Amir Pnueli, and Joseph Sifakis.
On the synthesis of discrete controllers for timed systems.
In *Proc. 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS'95)*, volume 900, pages 229–242. Springer, 1995.

[Reif'84] John H. Reif.
The Complexity of Two-Player Games of Incomplete Information.
J. Comput. Syst. Sci. 29(2): 274-301 (1984).

Timed Automata

[Alur & Dill'94]

A Timed Automaton $\mathcal{A}$ is a tuple $(L, \ell_0, Act, X, \mathrm{inv}, \longrightarrow)$ where:

- L is a finite set of locations
- ℓ_0 is the initial location
- X is a finite set of clocks
- Act is a finite set of actions
- $\longrightarrow$ is a set of transitions of the form $\ell \xrightarrow{g,a,R} \ell'$ with:
  - $\ell, \ell' \in L$,
  - $a \in Act$
  - a guard g which is a clock constraint over X
  - a reset set R which is the set of clocks to be reset to 0

Clock constraints are boolean combinations of $x \sim k$ with $x \in C$ and $k \in \mathbb{Z}$ and $\sim \in \{\leq, <\}$.

Semantics of Timed Automata

Let $\mathcal{A} = (L, \ell_0, Act, X, inv, \longrightarrow)$ be a Timed Automaton.

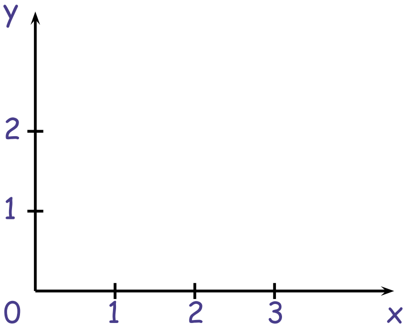
A **state** (ℓ, v) of $\mathcal{A}$ is in $L \times \mathbb{R}_{\geq 0}^X$

The semantics of $\mathcal{A}$ is a **Timed Transition System** $S_{\mathcal{A}} = (Q, q_0, Act \cup \mathbb{R}_{\geq 0}, \longrightarrow)$ with:

- $Q = L \times \mathbb{R}_{\geq 0}^X$
- $q_0 = (\ell_0, \overline{0})$
- $\longrightarrow$ consists in:

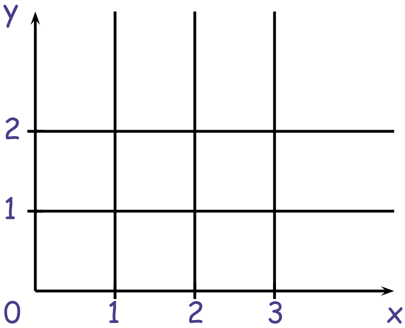
discrete transition: $(\ell, v) \xrightarrow{a} (\ell', v') \iff \begin{cases} \exists\, \ell \xrightarrow{g,a,r} \ell' \in \mathcal{A} \\ v \models g \\ v' = v[r \leftarrow 0] \\ v' \models inv(\ell') \end{cases}$

delay transition: $(\ell, v) \xrightarrow{d} (\ell, v + d) \iff d \in \mathbb{R}_{\geq 0} \wedge v + d \models inv(\ell)$


y
2
1
0
1
2
3
x

The Region Abstraction

[Alur & Dill'94]

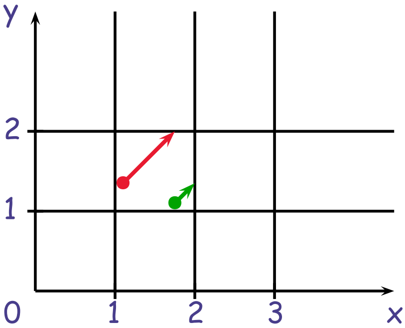


Build an equivalence relation which is of finite index and is:

- "compatible" with clock constraints $(g ::= x \sim c \quad g \wedge g)$

  $r, r' \in R \implies \forall$ constraints $g, \quad r \models g \iff r' \models g$

The Region Abstraction

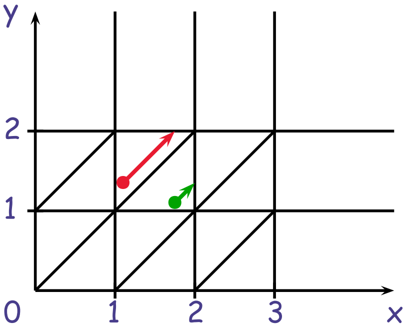


Build an equivalence relation which is of finite index and is:

- "compatible" with clock constraints $(g ::= x \sim c \quad g \wedge g)$
  $r, r' \in R \implies \forall$ constraints $g, \quad r \models g \iff r' \models g$
- "compatible" with time elapsing
  $r, r' \in R \implies$ same delay successor regions

The Region Abstraction

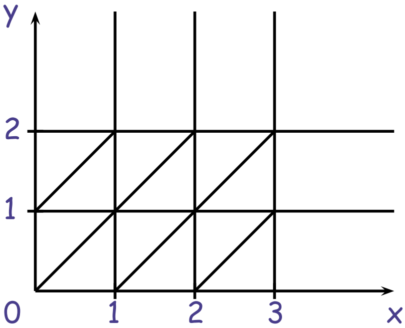
[Alur & Dill'94]



Build an equivalence relation which is of finite index and is:

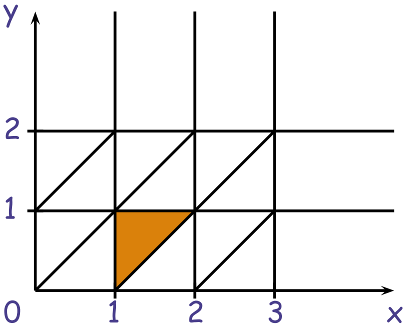
- "compatible" with clock constraints $(g ::= x \sim c \quad g \wedge g)$
  $r, r' \in R \implies \forall$ constraints $g, \quad r \models g \iff r' \models g$
- "compatible" with time elapsing
  $r, r' \in R \implies$ same delay successor regions

The Region Abstraction [Alur & Dill'94]



Build an equivalence relation which is of finite index and is:

- "compatible" with clock constraints ($g ::= x \sim c \quad g \wedge g$)
  $r, r' \in R \implies \forall$ constraints g, $\quad r \models g \iff r' \models g$
- "compatible" with time elapsing
  $r, r' \in R \implies$ same delay successor regions



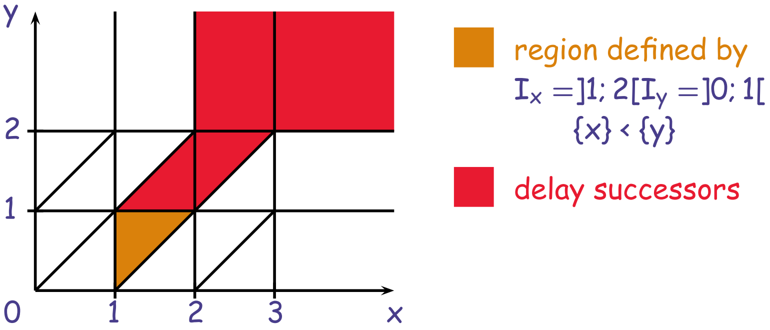
region defined by
$I_x =]1;2[\; I_y =]0;1[$
$\{x\} < \{y\}$

Build an equivalence relation which is of finite index and is:

- "compatible" with clock constraints ($g ::= x \sim c \quad g \wedge g$)
  $r, r' \in R \implies \forall$ constraints g, $\quad r \models g \iff r' \models g$
- "compatible" with time elapsing
  $r, r' \in R \implies$ same delay successor regions

The Region Abstraction

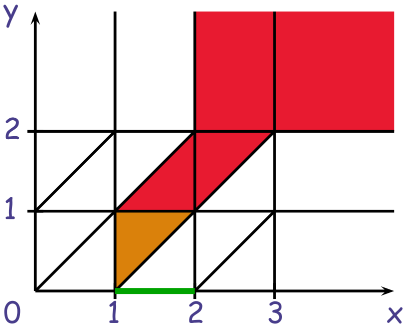
[Alur & Dill'94]



Build an equivalence relation which is of finite index and is:

- "compatible" with clock constraints $(g ::= x \sim c \quad g \wedge g)$
  $r, r' \in R \implies \forall$ constraints $g, \quad r \models g \iff r' \models g$
- "compatible" with time elapsing
  $r, r' \in R \implies$ same delay successor regions

The Region Abstraction [Alur & Dill'94]



region defined by
$I_x =]1;2[\; I_y =]0;1[$
$\{x\} < \{y\}$

delay successors

successor by reset

Build an equivalence relation which is of finite index and is:

- "compatible" with clock constraints ($g ::= x \sim c \quad g \wedge g$)
  $r, r' \in R \implies \forall$ constraints $g, \quad r \models g \iff r' \models g$
- "compatible" with time elapsing
  $r, r' \in R \implies$ same delay successor regions

The Region Automaton

- For each transition $\ell \xrightarrow{g,a,C:=0} \ell'$ of the TA
- Build transitions in the region automaton RA: $(\ell, R) \xrightarrow{a} (\ell', R')$ if:
  - there exists R'' a delay successor of R s.t.
  - R'' satisfies the guard g ($R'' \subseteq [\![g]\!]$)
  - $R''[C \leftarrow 0]$ is included in R'

a TA and its region automaton RA are time-abstract bisimilar

- The region automaton is finite
- Language accepted by the RA = untimed language accepted by the TA

  a timed word $w = (a, 1.2)(b, 3.4)(a, 6.256)$; $untimed(w) = aba$
- Language Emptyness can be decided on the RA

The Region Automaton

- For each transition $\ell \xrightarrow{g,a,C:=0} \ell'$ of the TA
- Build transitions in the region automaton RA: $(\ell, R) \xrightarrow{a} (\ell', R')$ if:
  - there exists R'' a delay successor of R s.t.
  - R'' satisfies the guard g ($R'' \subseteq [\![g]\!]$)
  - $R''[C \leftarrow 0]$ is included in R'

a TA and its region automaton RA are time-abstract bisimilar

- The region automaton is finite
- Language accepted by the RA = untimed language accepted by the TA
  a timed word $w = (a, 1.2)(b, 3.4)(a, 6.256)$; $untimed(w) = aba$
- Language Emptyness can be decided on the RA

The Region Automaton

- For each transition $\ell \xrightarrow{g,a,C:=0} \ell'$ of the TA
- Build transitions in the region automaton RA: $(\ell, R) \xrightarrow{a} (\ell', R')$ if:
  - there exists R'' a delay successor of R s.t.
  - R'' satisfies the guard g ($R'' \subseteq [\![g]\!]$)
  - $R''[C \leftarrow 0]$ is included in R'

a TA and its region automaton RA are time-abstract bisimilar

- The region automaton is finite
- Language accepted by the RA = untimed language accepted by the TA
  a timed word $w = (a, 1.2)(b, 3.4)(a, 6.256)$; $untimed(w) = aba$
- Language Emptyness can be decided on the RA

The Region Automaton

- For each transition $\ell \xrightarrow{g,a,C:=0} \ell'$ of the TA
- Build transitions in the region automaton RA: $(\ell, R) \xrightarrow{a} (\ell', R')$ if:
  - there exists R'' a delay successor of R s.t.
  - R'' satisfies the guard g ($R'' \subseteq [\![g]\!]$)
  - $R''[C \leftarrow 0]$ is included in R'

a TA and its region automaton RA are time-abstract bisimilar

- The region automaton is finite
- Language accepted by the RA = untimed language accepted by the TA
  a timed word $w = (a, 1.2)(b, 3.4)(a, 6.256)$; untimed$(w) = aba$
- Language Emptyness can be decided on the RA

The Region Automaton

- For each transition $\ell \xrightarrow{g,a,C:=0} \ell'$ of the TA
- Build transitions in the region automaton RA: $(\ell, R) \xrightarrow{a} (\ell', R')$ if:
  - there exists R'' a delay successor of R s.t.
  - R'' satisfies the guard g ($R'' \subseteq [\![g]\!]$)
  - $R''[C \leftarrow 0]$ is included in R'

a TA and its region automaton RA are time-abstract bisimilar

- The region automaton is finite
- Language accepted by the RA = untimed language accepted by the TA

  a timed word $w = (a, 1.2)(b, 3.4)(a, 6.256)$; $untimed(w) = aba$
- Language Emptyness can be decided on the RA

The Region Automaton

- For each transition $\ell \xrightarrow{g,a,C:=0} \ell'$ of the TA
- Build transitions in the region automaton RA: $(\ell, R) \xrightarrow{a} (\ell', R')$ if:
  - there exists R'' a delay successor of R s.t.
  - R'' satisfies the guard g ($R'' \subseteq [\![g]\!]$)
  - $R''[C \leftarrow 0]$ is included in R'

a TA and its region automaton RA are time-abstract bisimilar

- The region automaton is finite
- Language accepted by the RA = untimed language accepted by the TA

  a timed word $w = (a, 1.2)(b, 3.4)(a, 6.256)$; $\text{untimed}(w) = aba$
- Language Emptyness can be decided on the RA

The Region Automaton

- For each transition $\ell \xrightarrow{g,a,C:=0} \ell'$ of the TA
- Build transitions in the region automaton RA: $(\ell, R) \xrightarrow{a} (\ell', R')$ if:
  - there exists R'' a delay successor of R s.t.
  - R'' satisfies the guard g ($R'' \subseteq [\![g]\!]$)
  - $R''[C \leftarrow 0]$ is included in R'

a TA and its region automaton RA are time-abstract bisimilar

- The region automaton is finite
- Language accepted by the RA = untimed language accepted by the TA

  a timed word $w = (a, 1.2)(b, 3.4)(a, 6.256)$; $untimed(w) = aba$
- Language Emptyness can be decided on the RA

The Region Automaton

- For each transition $\ell \xrightarrow{g,a,C:=0} \ell'$ of the TA
- Build transitions in the region automaton RA: $(\ell,R) \xrightarrow{a} (\ell',R')$ if:
  - there exists R'' a delay successor of R s.t.
  - R'' satisfies the guard g ($R'' \subseteq [\![g]\!]$)
  - $R''[C \leftarrow 0]$ is included in R'

a TA and its region automaton RA are time-abstract bisimilar

- The region automaton is finite
- Language accepted by the RA = untimed language accepted by the TA

  a timed word $w = (a,1.2)(b,3.4)(a,6.256)$; $\text{untimed}(w) = aba$
- Language Emptyness can be decided on the RA

The Region Automaton

- For each transition $\ell \xrightarrow{g,a,C:=0} \ell'$ of the TA
- Build transitions in the region automaton RA: $(\ell, R) \xrightarrow{a} (\ell', R')$ if:
  - there exists R'' a delay successor of R s.t.
  - R'' satisfies the guard g ($R'' \subseteq [\![g]\!]$)
  - $R''[C \leftarrow 0]$ is included in R'

a TA and its region automaton RA are time-abstract bisimilar

- The region automaton is finite
- Language accepted by the RA = untimed language accepted by the TA

  a timed word $w = (a, 1.2)(b, 3.4)(a, 6.256)$; $\text{untimed}(w) = aba$
- Language Emptyness can be decided on the RA

The Region Automaton

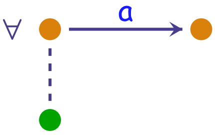
- For each transition $\ell \xrightarrow{g,a,C:=0} \ell'$ of the TA
- Build transitions in the region automaton RA: $(\ell, R) \xrightarrow{a} (\ell', R')$ if:
  - there exists R'' a delay successor of R s.t.
  - R'' satisfies the guard g ($R'' \subseteq [\![g]\!]$)
  - $R''[C \leftarrow 0]$ is included in R'

a TA and its region automaton RA are time-abstract bisimilar

- The region automaton is finite
- Language accepted by the RA = untimed language accepted by the TA

  a timed word $w = (a, 1.2)(b, 3.4)(a, 6.256)$; $\text{untimed}(w) = aba$
- Language Emptyness can be decided on the RA

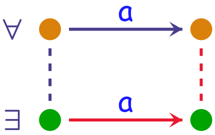
Time-abstract bisimulation



$$
\begin{array}{ccccccc}
(\ell_0, v_0) & \xrightarrow{a_1, t_1} & (\ell_1, v_1) & \xrightarrow{a_2, t_2} & (\ell_2, v_2) & \xrightarrow{a_3, t_3} & \dots \\
\updownarrow & & \updownarrow & & \updownarrow & & \\
(\ell_0, R_0) & \xrightarrow{a_1} & (\ell_1, R_1) & \xrightarrow{a_2} & (\ell_2, R_2) & \xrightarrow{a_3} & \dots
\end{array}
$$

with $v_i \in R_i$ for all i.

Time-abstract bisimulation



$$
\begin{array}{ccccccc}
(\ell_0, v_0) & \xrightarrow{a_1, t_1} & (\ell_1, v_1) & \xrightarrow{a_2, t_2} & (\ell_2, v_2) & \xrightarrow{a_3, t_3} & \ldots \\
\updownarrow & & \updownarrow & & \updownarrow & & \\
(\ell_0, R_0) & \xrightarrow{a_1} & (\ell_1, R_1) & \xrightarrow{a_2} & (\ell_2, R_2) & \xrightarrow{a_3} & \ldots
\end{array}
$$

with $v_i \in R_i$ for all i.

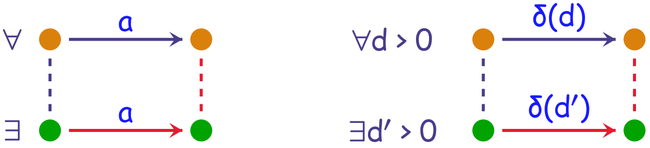
Time-abstract bisimulation

$$(\ell_0, v_0) \xrightarrow{a_1, t_1} (\ell_1, v_1) \xrightarrow{a_2, t_2} (\ell_2, v_2) \xrightarrow{a_3, t_3} \dots$$

$$(\ell_0, R_0) \xrightarrow{a_1} (\ell_1, R_1) \xrightarrow{a_2} (\ell_2, R_2) \xrightarrow{a_3} \dots$$

with $v_i \in R_i$ for all i.

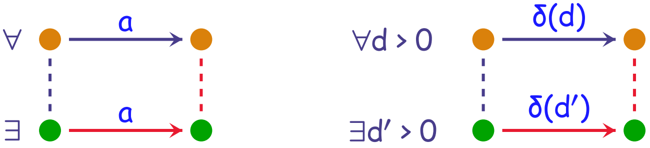
Time-abstract bisimulation



$$\begin{array}{ccccccc}
(\ell_0, v_0) & \xrightarrow{a_1, t_1} & (\ell_1, v_1) & \xrightarrow{a_2, t_2} & (\ell_2, v_2) & \xrightarrow{a_3, t_3} & \ldots \\
\updownarrow & & \updownarrow & & \updownarrow & & \\
(\ell_0, R_0) & \xrightarrow{a_1} & (\ell_1, R_1) & \xrightarrow{a_2} & (\ell_2, R_2) & \xrightarrow{a_3} & \ldots
\end{array}$$

with $v_i \in R_i$ for all i.

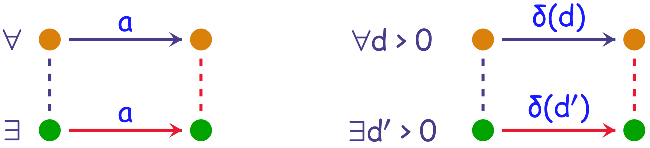
Time-abstract bisimulation



$$
\begin{array}{ccccccc}
(\ell_0, v_0) & \xrightarrow{a_1, t_1} & (\ell_1, v_1) & \xrightarrow{a_2, t_2} & (\ell_2, v_2) & \xrightarrow{a_3, t_3} & \dots \\
\updownarrow & & \updownarrow & & \updownarrow & & \\
(\ell_0, R_0) & \xrightarrow{a_1} & (\ell_1, R_1) & \xrightarrow{a_2} & (\ell_2, R_2) & \xrightarrow{a_3} & \dots
\end{array}
$$

with $v_i \in R_i$ for all i.

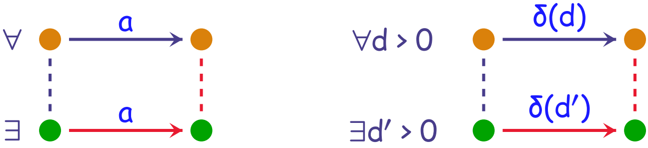
Time-abstract bisimulation



$$\begin{array}{ccccccc}
(\ell_0, v_0) & \xrightarrow{a_1, t_1} & (\ell_1, v_1) & \xrightarrow{a_2, t_2} & (\ell_2, v_2) & \xrightarrow{a_3, t_3} & \dots \\
\downarrow & & \downarrow & & \downarrow & & \\
\updownarrow & & \updownarrow & & \updownarrow & & \\
(\ell_0, R_0) & \xrightarrow{a_1} & (\ell_1, R_1) & \xrightarrow{a_2} & (\ell_2, R_2) & \xrightarrow{a_3} & \dots
\end{array}$$

with $v_i \in R_i$ for all i.

Time-abstract bisimulation



$$
\begin{tikzcd}
(\ell_0, v_0) \arrow[r, "a_1, t_1"] \arrow[d, leftrightarrow] & (\ell_1, v_1) \arrow[r, "a_2, t_2"] \arrow[d, leftrightarrow] & (\ell_2, v_2) \arrow[r, "a_3, t_3"] \arrow[d, leftrightarrow] & \ldots \\
(\ell_0, R_0) \arrow[r, "a_1"] & (\ell_1, R_1) \arrow[r, "a_2"] & (\ell_2, R_2) \arrow[r, "a_3"] & \ldots
\end{tikzcd}
$$

with $v_i \in R_i$ for all i.

Definition (Outcome in Timed Games)

Let $G = (L, \ell_0, Act, X, E, inv)$ be a TGA and f a strategy over G. The **outcome** $\mathrm{Outcome}((\ell, v), f)$ of f from configuration (ℓ, v) in G is the subset of $\mathrm{Runs}((\ell, v), G)$ defined inductively by:

- $(\ell, v) \in \mathrm{Outcome}((\ell, v), f)$,
- if $\rho \in \mathrm{Outcome}((\ell, v), f)$ then $\rho' = \rho \xrightarrow{e} (\ell', v') \in \mathrm{Outcome}((\ell, v), f)$ if $\rho' \in \mathrm{Runs}((\ell, v), G)$ and one of the following three conditions hold:
  1. $e \in Act_u$,
  2. $e \in Act_c$ and $e = f(\rho)$,
  3. $e \in \mathbb{R}_{\geq 0}$ and $\forall 0 \leq e' < e, \exists (\ell'', v'') \in (L \times \mathbb{R}^X_{\geq 0})$ s.t. $last(\rho) \xrightarrow{e'} (\ell'', v'') \wedge f(\rho \xrightarrow{e'} (\ell'', v'')) = \lambda$.
- an infinite run ρ is in $\in \mathrm{Outcome}((\ell, v), f)$ if all the finite prefixes of ρ are in $\mathrm{Outcome}((\ell, v), f)$.

States & Symbolic States

- $Q = L \times \mathbb{R}_{\geq 0}^{Clock}$ is the set of states of the TGA
  $$q = (\ell, v) \in Q$$
- Discrete predecessors of $X \subseteq Q$ by an action a:
  $$\mathrm{Pred}^a(X) = \{q \in Q \mid q \xrightarrow{a} q' \text{ and } q' \in X\}$$
- Time predecessors of $X \subseteq Q$:
  $$\mathrm{Pred}^\delta(X) = \{q \in Q \mid \exists t \geq 0 \mid q \xrightarrow{t} q' \text{ and } q' \in X\}$$
- Zone = conjunction of triangular constraints
  $x - y < 3, x \geq 2 \wedge 1 < y - x < 2$
- Symbolic State is defined by a State predicate (SP)
  $P = \cup_{i \in [1..n]}(\ell_{j_i}, Z_i)$, $\ell_{j_i} \in L$, Z_i is a zone
  $(\ell_1, 2 \leq x < 4)$ or $(\ell_0, x < 1 \wedge y - x \geq 2)$ or $(\ell_0, x \leq 2) \cup (\ell_2, x > 0)$

Effectiveness of Pred^a and Pred^δ

If P is a SP then $\mathrm{Pred}^a(P), \mathrm{Pred}^\delta(P)$ are SP and can be computed effectively. (There is a symbolic version for Pred^a and Pred^δ.)

States & Symbolic States

- $Q = L \times \mathbb{R}_{\geq 0}^{Clock}$ is the set of states of the TGA
$$q = (\ell, v) \in Q$$
- Discrete predecessors of $X \subseteq Q$ by an action a:
$$\mathrm{Pred}^a(X) = \{q \in Q \mid q \xrightarrow{a} q' \text{ and } q' \in X\}$$
- Time predecessors of $X \subseteq Q$:
$$\mathrm{Pred}^\delta(X) = \{q \in Q \mid \exists t \geq 0 \mid q \xrightarrow{t} q' \text{ and } q' \in X\}$$
- Zone = conjunction of triangular constraints
$x - y < 3, x \geq 2 \wedge 1 < y - x < 2$
- Symbolic State is defined by a State predicate (SP)
$P = \cup_{i \in [1..n]}(\ell_{j_i}, Z_i)$, $\ell_{j_i} \in L$, Z_i is a zone
$(\ell_1, 2 \leq x < 4)$ or $(\ell_0, x < 1 \wedge y - x \geq 2)$ or $(\ell_0, x \leq 2) \cup (\ell_2, x > 0)$

Effectiveness of Pred^a and Pred^δ

If P is a SP then $\mathrm{Pred}^a(P), \mathrm{Pred}^\delta(P)$ are SP and can be computed effectively. (There is a symbolic version for Pred^a and Pred^δ.)

States & Symbolic States

- $Q = L \times \mathbb{R}_{\geq 0}^{Clock}$ is the set of states of the TGA
  $q = (\ell, v) \in Q$
- Discrete predecessors of $X \subseteq Q$ by an action a:
  $$\mathrm{Pred}^a(X) = \{q \in Q \mid q \xrightarrow{a} q' \text{ and } q' \in X\}$$
- Time predecessors of $X \subseteq Q$:
  $$\mathrm{Pred}^\delta(X) = \{q \in Q \mid \exists t \geq 0 \mid q \xrightarrow{t} q' \text{ and } q' \in X\}$$
- Zone = conjunction of triangular constraints
  $x - y < 3, x \geq 2 \wedge 1 < y - x < 2$
- Symbolic State is defined by a State predicate (SP)
  $P = \cup_{i \in [1..n]}(\ell_{j_i}, Z_i)$, $\ell_{j_i} \in L$, Z_i is a zone
  $(\ell_1, 2 \leq x < 4)$ or $(\ell_0, x < 1 \wedge y - x \geq 2)$ or $(\ell_0, x \leq 2) \cup (\ell_2, x > 0)$

Effectiveness of Pred^a and Pred^δ

If P is a SP then $\mathrm{Pred}^a(P), \mathrm{Pred}^\delta(P)$ are SP and can be computed effectively. (There is a symbolic version for Pred^a and Pred^δ.)

States & Symbolic States

- $Q = L \times \mathbb{R}_{\geq 0}^{Clock}$ is the set of states of the TGA

  $$q = (\ell, v) \in Q$$

- Discrete predecessors of $X \subseteq Q$ by an action a:

  $$\mathrm{Pred}^a(X) = \{q \in Q \mid q \xrightarrow{a} q' \text{ and } q' \in X\}$$

- Time predecessors of $X \subseteq Q$:

  $$\mathrm{Pred}^\delta(X) = \{q \in Q \mid \exists t \geq 0 \mid q \xrightarrow{t} q' \text{ and } q' \in X\}$$

- Zone = conjunction of triangular constraints

  $x - y < 3, x \geq 2 \wedge 1 < y - x < 2$

- Symbolic State is defined by a State predicate (SP)

  $P = \cup_{i \in [1..n]}(\ell_{j_i}, Z_i)$, $\ell_{j_i} \in L$, Z_i is a zone

  $(\ell_1, 2 \leq x < 4)$ or $(\ell_0, x < 1 \wedge y - x \geq 2)$ or $(\ell_0, x \leq 2) \cup (\ell_2, x > 0)$

Effectiveness of Pred^a and Pred^δ

If P is a SP then $\mathrm{Pred}^a(P), \mathrm{Pred}^\delta(P)$ are SP and can be computed effectively. (There is a symbolic version for Pred^a and Pred^δ.)

States & Symbolic States

- $Q = L \times \mathbb{R}_{\geq 0}^{Clock}$ is the set of states of the TGA
  $$q = (\ell, v) \in Q$$
- Discrete predecessors of $X \subseteq Q$ by an action a:
  $$\mathrm{Pred}^a(X) = \{q \in Q \mid q \xrightarrow{a} q' \text{ and } q' \in X\}$$
- Time predecessors of $X \subseteq Q$:
  $$\mathrm{Pred}^\delta(X) = \{q \in Q \mid \exists t \geq 0 \mid q \xrightarrow{t} q' \text{ and } q' \in X\}$$
- Zone = conjunction of triangular constraints
  $x - y < 3, x \geq 2 \wedge 1 < y - x < 2$
- Symbolic State is defined by a State predicate (SP)
  $P = \cup_{i \in [1..n]}(\ell_{j_i}, Z_i)$, $\ell_{j_i} \in L$, Z_i is a zone
  $(\ell_1, 2 \leq x < 4)$ or $(\ell_0, x < 1 \wedge y - x \geq 2)$ or $(\ell_0, x \leq 2) \cup (\ell_2, x > 0)$

Effectiveness of Pred^a and Pred^δ

If P is a SP then $\mathrm{Pred}^a(P), \mathrm{Pred}^\delta(P)$ are SP and can be computed effectively. (There is a symbolic version for Pred^a and Pred^δ.)

States & Symbolic States

- $Q = L \times \mathbb{R}_{\geq 0}^{Clock}$ is the set of states of the TGA
  $$q = (\ell, v) \in Q$$
- Discrete predecessors of $X \subseteq Q$ by an action a:
  $$\mathrm{Pred}^a(X) = \{q \in Q \mid q \xrightarrow{a} q' \text{ and } q' \in X\}$$
- Time predecessors of $X \subseteq Q$:
  $$\mathrm{Pred}^\delta(X) = \{q \in Q \mid \exists t \geq 0 \mid q \xrightarrow{t} q' \text{ and } q' \in X\}$$
- Zone = conjunction of triangular constraints
  $x - y < 3, x \geq 2 \wedge 1 < y - x < 2$
- Symbolic State is defined by a State predicate (SP)
  $P = \cup_{i \in [1..n]} (\ell_{j_i}, Z_i)$, $\ell_{j_i} \in L$, Z_i is a zone
  $(\ell_1, 2 \leq x < 4)$ or $(\ell_0, x < 1 \wedge y - x \geq 2)$ or $(\ell_0, x \leq 2) \cup (\ell_2, x > 0)$

Effectiveness of Pred^a and Pred^δ

If P is a SP then $\mathrm{Pred}^a(P), \mathrm{Pred}^\delta(P)$ are SP and can be computed effectively. (There is a symbolic version for Pred^a and Pred^δ.)

States & Symbolic States

- $Q = L \times \mathbb{R}_{\geq 0}^{Clock}$ is the set of states of the TGA
  $q = (\ell, v) \in Q$
- Discrete predecessors of $X \subseteq Q$ by an action a:
  $$\mathrm{Pred}^a(X) = \{q \in Q \mid q \xrightarrow{a} q' \text{ and } q' \in X\}$$
- Time predecessors of $X \subseteq Q$:
  $$\mathrm{Pred}^\delta(X) = \{q \in Q \mid \exists t \geq 0 \mid q \xrightarrow{t} q' \text{ and } q' \in X\}$$
- Zone = conjunction of triangular constraints
  $x - y < 3,\ x \geq 2 \wedge 1 < y - x < 2$
- Symbolic State is defined by a State predicate (SP)
  $P = \cup_{i \in [1..n]}(\ell_{j_i}, Z_i)$, $\ell_{j_i} \in L$, Z_i is a zone
  $(\ell_1, 2 \leq x < 4)$ or $(\ell_0, x < 1 \wedge y - x \geq 2)$ or $(\ell_0, x \leq 2) \cup (\ell_2, x > 0)$

Effectiveness of Pred^a and Pred^δ

If P is a SP then $\mathrm{Pred}^a(P), \mathrm{Pred}^\delta(P)$ are SP and can be computed effectively. (There is a symbolic version for Pred^a and Pred^δ.)

States & Symbolic States

- $Q = L \times \mathbb{R}_{\geq 0}^{Clock}$ is the set of states of the TGA
  $q = (\ell, v) \in Q$
- Discrete predecessors of $X \subseteq Q$ by an action a:
  $$\mathrm{Pred}^a(X) = \{q \in Q \mid q \xrightarrow{a} q' \text{ and } q' \in X\}$$
- Time predecessors of $X \subseteq Q$:
  $$\mathrm{Pred}^\delta(X) = \{q \in Q \mid \exists t \geq 0 \mid q \xrightarrow{t} q' \text{ and } q' \in X\}$$
- Zone = conjunction of triangular constraints
  $x - y < 3, x \geq 2 \wedge 1 < y - x < 2$
- Symbolic State is defined by a State predicate (SP)
  $P = \cup_{i \in [1..n]}(\ell_{j_i}, Z_i)$, $\ell_{j_i} \in L$, Z_i is a zone
  $(\ell_1, 2 \leq x < 4)$ or $(\ell_0, x < 1 \wedge y - x \geq 2)$ or $(\ell_0, x \leq 2) \cup (\ell_2, x > 0)$

Effectiveness of Pred^a and Pred^δ

If P is a SP then $\mathrm{Pred}^a(P), \mathrm{Pred}^\delta(P)$ are SP and can be computed effectively. (There is a symbolic version for Pred^a and Pred^δ.)

Symbolic Computation For Timed Games

X is a state predicate

- $cPred(X) = \bigcup_{c \in Act_c} Pred^c(X)$ $uPred(X) = \bigcup_{u \in Act_u} Pred^u(X)$
  cPred and uPred are effectively computable
- $Pred_\delta(X, Y)$: Time controllable predecessors of X avoiding Y:

q $q' \in X$

$Pred_\delta(X, Y)$ is effectively computable for state predicates X, Y

- Controllable Predecessors Operator for Timed Games

$$\pi_\delta(X) = Pred_\delta\left(cPred(X), uPred(\overline{X})\right)$$

$\pi_\delta(X)$ is effectively computable for state predicate X.

Symbolic Computation For Timed Games

X is a state predicate

- $\mathrm{cPred}(X) = \bigcup_{c \in Act_c} \mathrm{Pred}^c(X)$ $\mathrm{uPred}(X) = \bigcup_{u \in Act_u} \mathrm{Pred}^u(X)$
  cPred and uPred are effectively computable
- $\mathrm{Pred}_\delta(X, Y)$: Time controllable predecessors of X avoiding Y:

q $q' \in X$

$\mathrm{Pred}_\delta(X, Y)$ is effectively computable for state predicates X, Y

- Controllable Predecessors Operator for Timed Games

$$\pi_\delta(X) = \mathrm{Pred}_\delta\left(\mathrm{cPred}(X), \mathrm{uPred}(\overline{X})\right)$$

$\pi_\delta(X)$ is effectively computable for state predicate X.

Symbolic Computation For Timed Games

X is a state predicate

- $cPred(X) = \bigcup_{c \in Act_c} Pred^c(X)$ $\qquad$ $uPred(X) = \bigcup_{u \in Act_u} Pred^u(X)$
  cPred and uPred are effectively computable
- $Pred_\delta(X, Y)$: Time controllable predecessors of X avoiding Y:

$q \qquad\qquad q' \in X$

$Pred_\delta(X, Y)$ is effectively computable for state predicates X, Y

- Controllable Predecessors Operator for Timed Games

$$\pi_\delta(X) = Pred_\delta\left(cPred(X), uPred(\overline{X})\right)$$

$\pi_\delta(X)$ is effectively computable for state predicate X.

Symbolic Computation For Timed Games

X is a state predicate

- $\mathrm{cPred}(X) = \bigcup_{c \in Act_c} \mathrm{Pred}^c(X)$ $\qquad$ $\mathrm{uPred}(X) = \bigcup_{u \in Act_u} \mathrm{Pred}^u(X)$
  cPred and uPred are effectively computable
- $\mathrm{Pred}_\delta(X, Y)$: Time controllable predecessors of X avoiding Y:

$$q \xrightarrow{\quad t \quad} q' \in X$$

$\mathrm{Pred}_\delta(X, Y)$ is effectively computable for state predicates X, Y

- Controllable Predecessors Operator for Timed Games

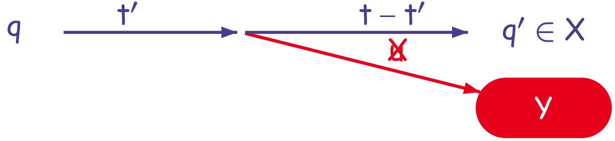
$$\pi_\delta(X) = \mathrm{Pred}_\delta\left(\mathrm{cPred}(X), \mathrm{uPred}(\overline{X})\right)$$

$\pi_\delta(X)$ is effectively computable for state predicate X.

Symbolic Computation For Timed Games

X is a state predicate

- $\text{cPred}(X) = \bigcup_{c \in Act_c} \text{Pred}^c(X)$ $\quad$ $\text{uPred}(X) = \bigcup_{u \in Act_u} \text{Pred}^u(X)$
  cPred and uPred are effectively computable
- $\text{Pred}_\delta(X, Y)$: Time controllable predecessors of X avoiding Y:



$\text{Pred}_\delta(X, Y)$ is effectively computable for state predicates X, Y

- Controllable Predecessors Operator for Timed Games

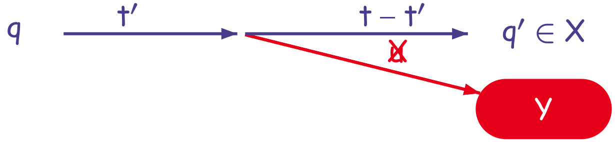
$$\pi_\delta(X) = \text{Pred}_\delta\left(\text{cPred}(X), \text{uPred}(\overline{X})\right)$$

$\pi_\delta(X)$ is effectively computable for state predicate X.

Symbolic Computation For Timed Games

X is a state predicate

- $cPred(X) = \bigcup_{c \in Act_c} Pred^c(X)$ $\quad$ $uPred(X) = \bigcup_{u \in Act_u} Pred^u(X)$
  cPred and uPred are effectively computable
- $Pred_\delta(X, Y)$: Time controllable predecessors of X avoiding Y:



$Pred_\delta(X, Y)$ is effectively computable for state predicates X, Y

- Controllable Predecessors Operator for Timed Games

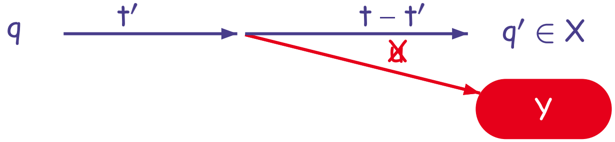
$$\pi_\delta(X) = Pred_\delta\left(cPred(X), uPred(\overline{X})\right)$$

$\pi_\delta(X)$ is effectively computable for state predicate X.

Symbolic Computation For Timed Games

X is a state predicate

- $cPred(X) = \bigcup_{c \in Act_c} Pred^c(X)$ $\qquad uPred(X) = \bigcup_{u \in Act_u} Pred^u(X)$
  cPred and uPred are effectively computable
- $Pred_\delta(X, Y)$: Time controllable predecessors of X avoiding Y:



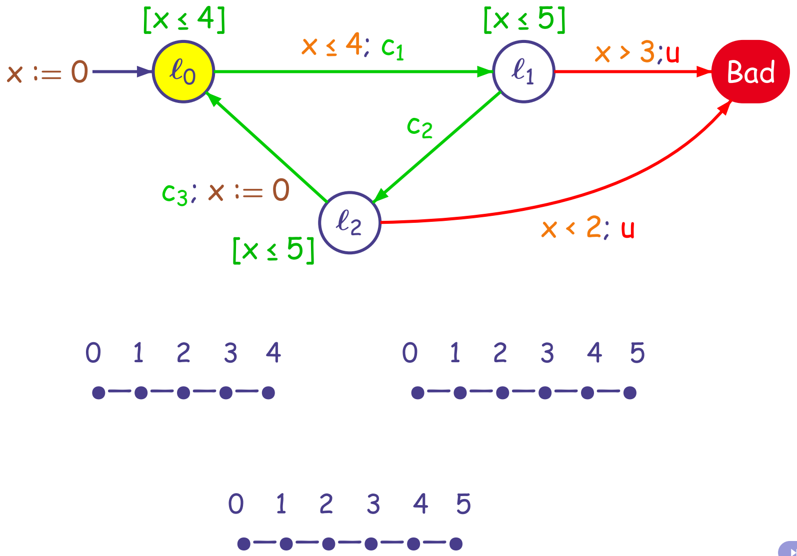
$Pred_\delta(X, Y)$ is effectively computable for state predicates X, Y

- Controllable Predecessors Operator for Timed Games

$$\pi_\delta(X) = Pred_\delta\left(cPred(X), uPred(\overline{X})\right)$$

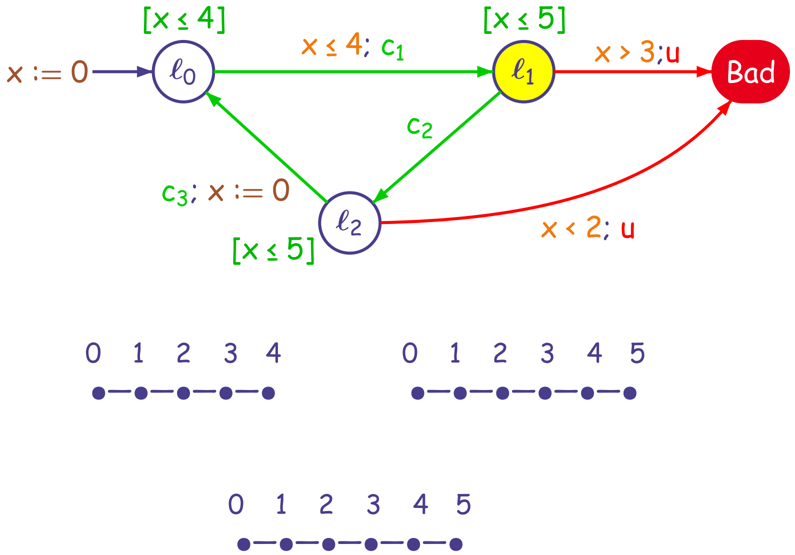
$\pi_\delta(X)$ is effectively computable for state predicate X.

Example of Computation for Safety Games

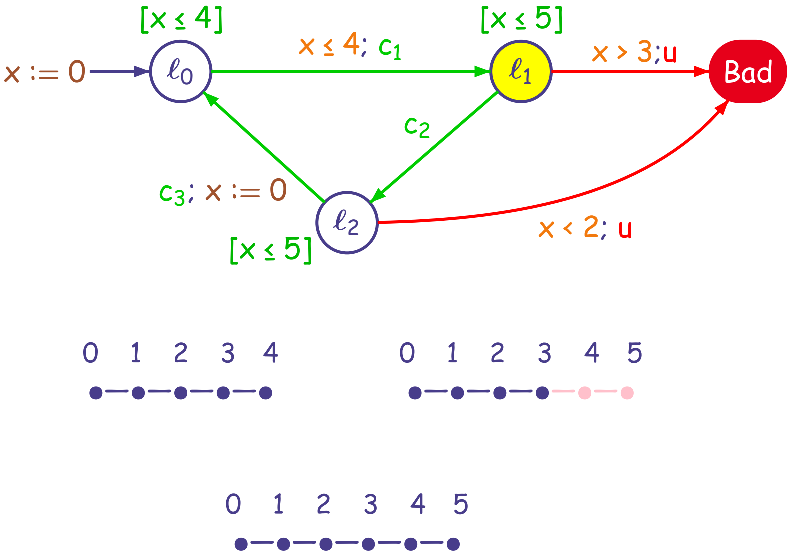


Skip

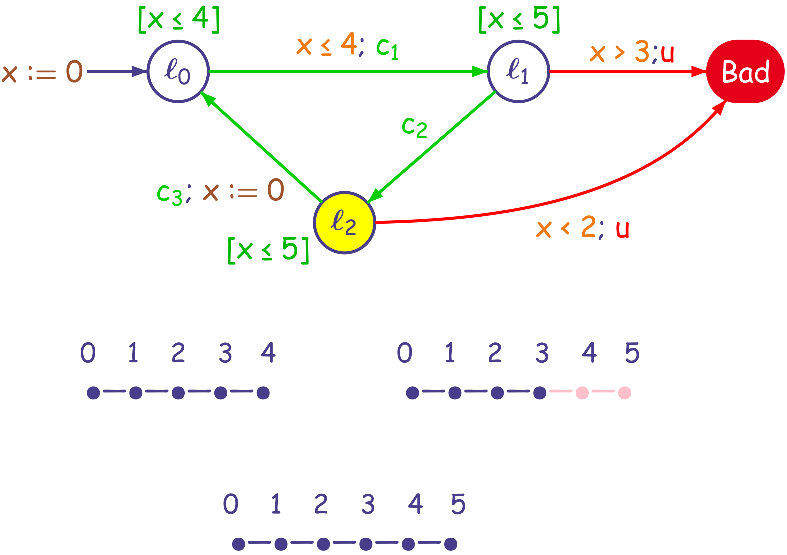
Example of Computation for Safety Games



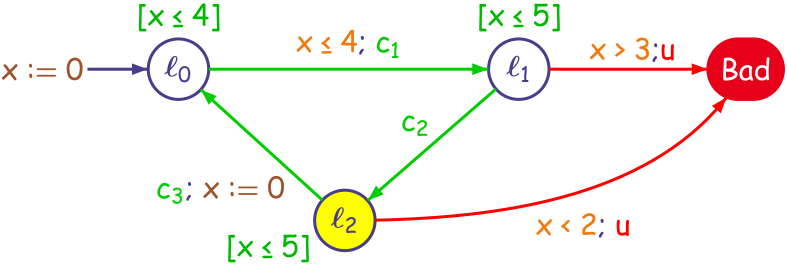
Example of Computation for Safety Games

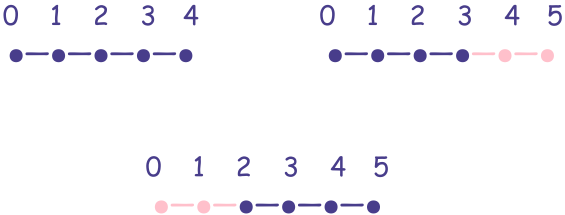


Example of Computation for Safety Games

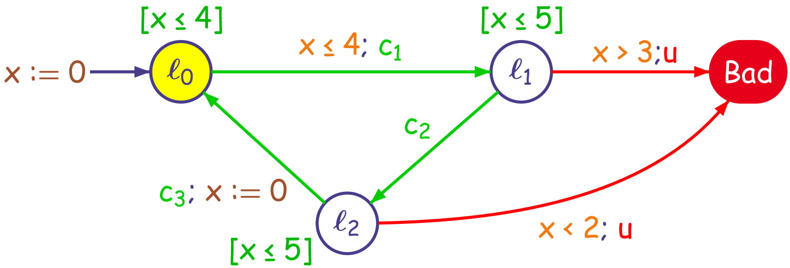


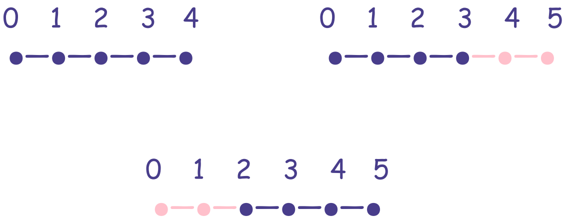
Example of Computation for Safety Games



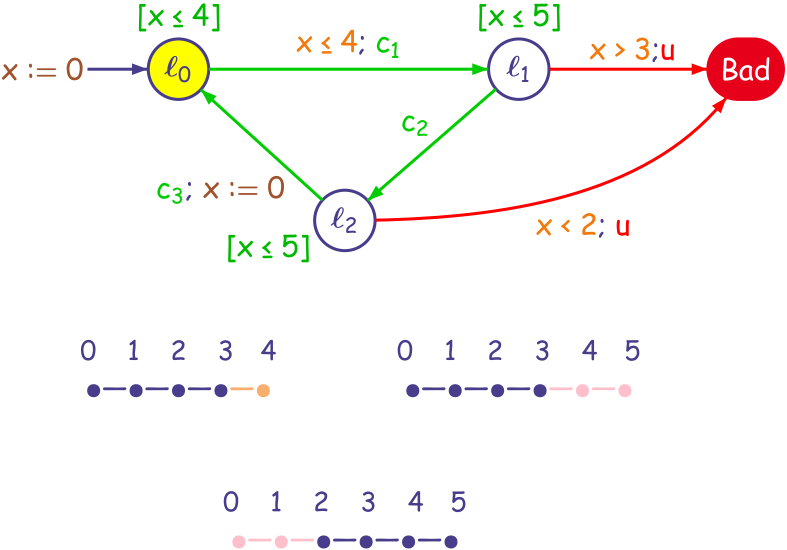


Example of Computation for Safety Games

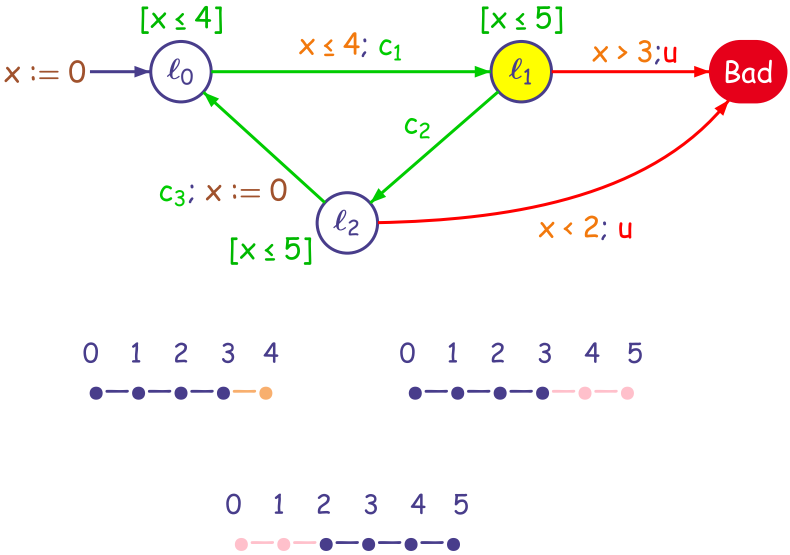




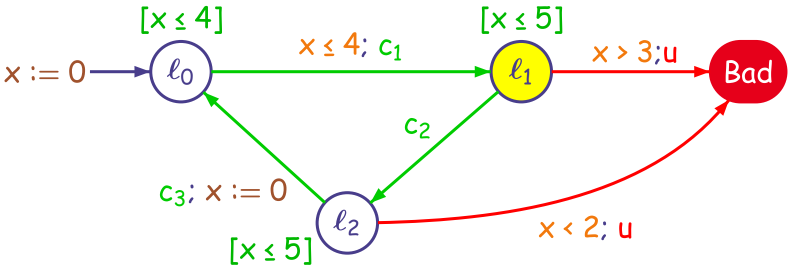
Example of Computation for Safety Games

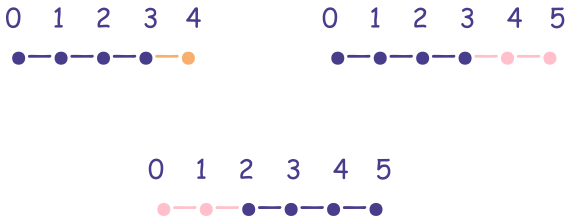


Example of Computation for Safety Games

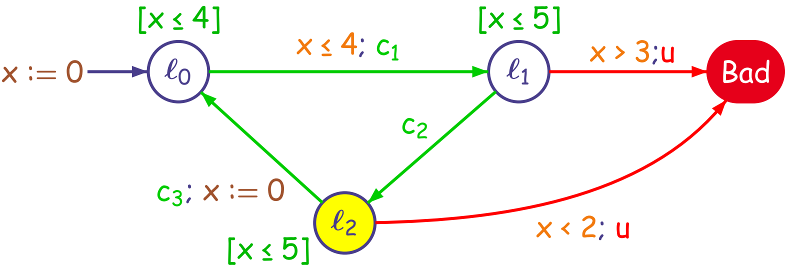


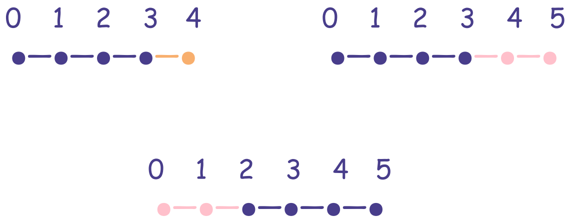
Example of Computation for Safety Games



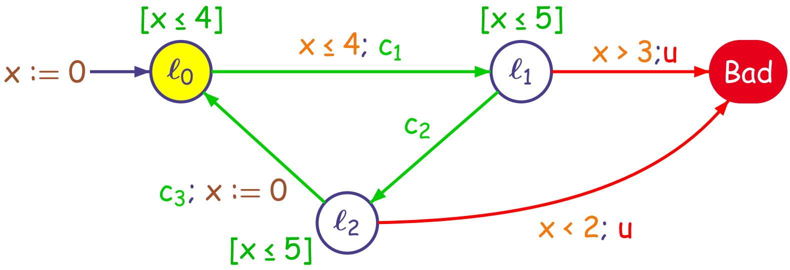


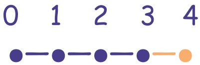
Example of Computation for Safety Games

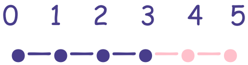


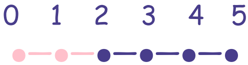


Example of Computation for Safety Games

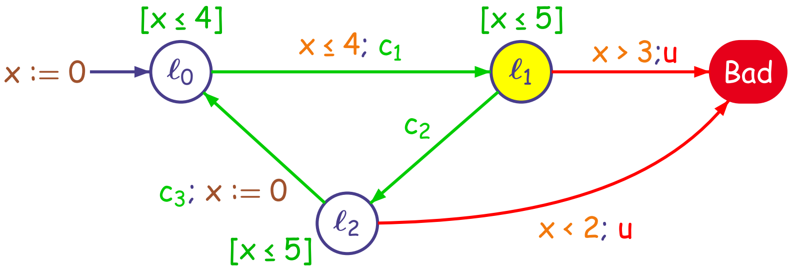


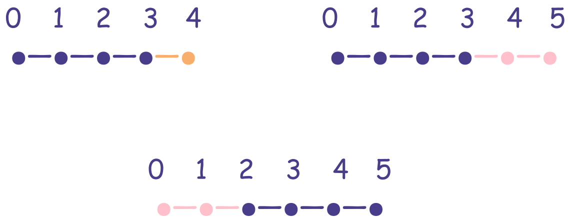




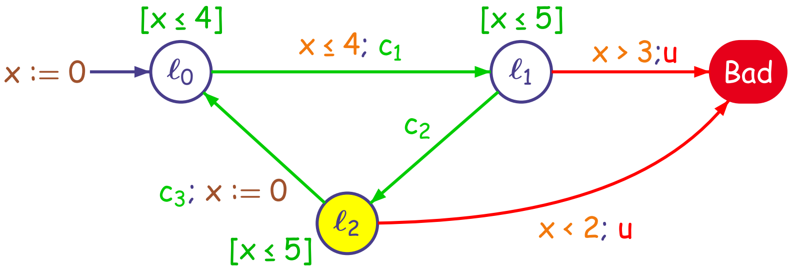


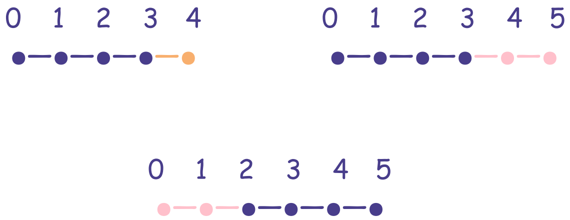
Example of Computation for Safety Games



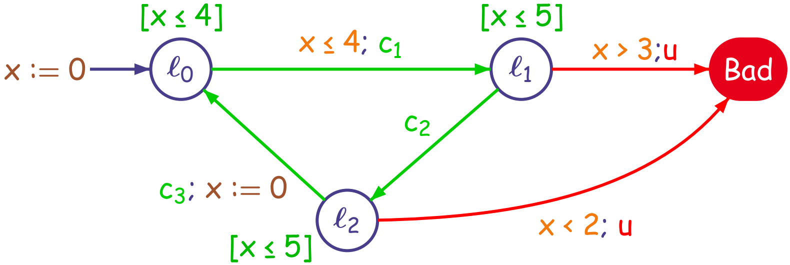


Example of Computation for Safety Games





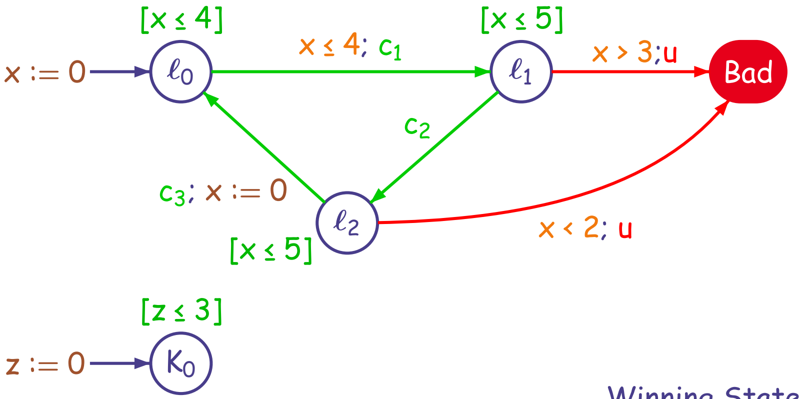
Example of Computation for Safety Games



Winning States

$(\ell_0, 0 \leq x \leq 3)$
$(\ell_1, 0 \leq x \leq 3)$
$(\ell_2, 2 \leq x \leq 5)$

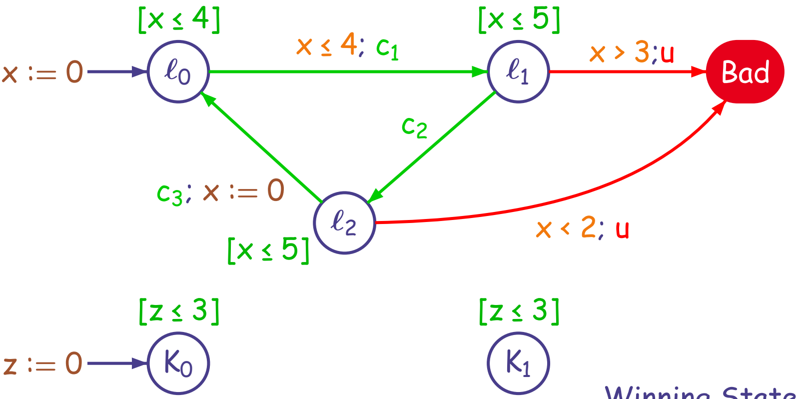
Example of Computation for Safety Games



Winning States

$(\ell_0, 0 \leq x \leq 3)$
$(\ell_1, 0 \leq x \leq 3)$
$(\ell_2, 2 \leq x \leq 5)$

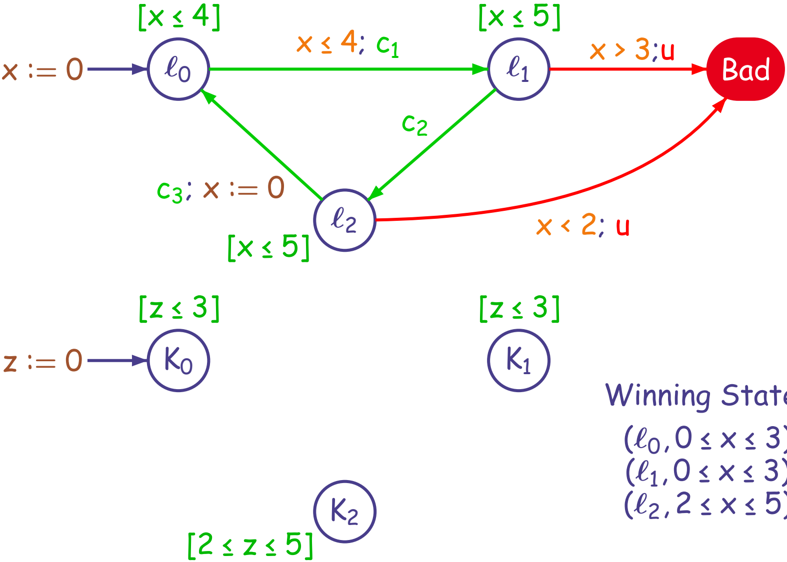
Example of Computation for Safety Games



Winning States

$(\ell_0, 0 \leq x \leq 3)$
$(\ell_1, 0 \leq x \leq 3)$
$(\ell_2, 2 \leq x \leq 5)$

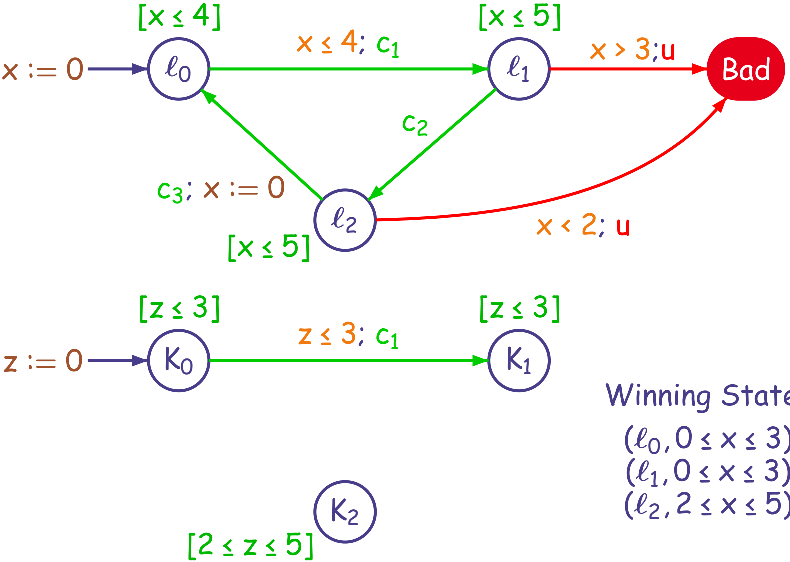
Example of Computation for Safety Games



Winning States

$(\ell_0, 0 \leq x \leq 3)$
$(\ell_1, 0 \leq x \leq 3)$
$(\ell_2, 2 \leq x \leq 5)$

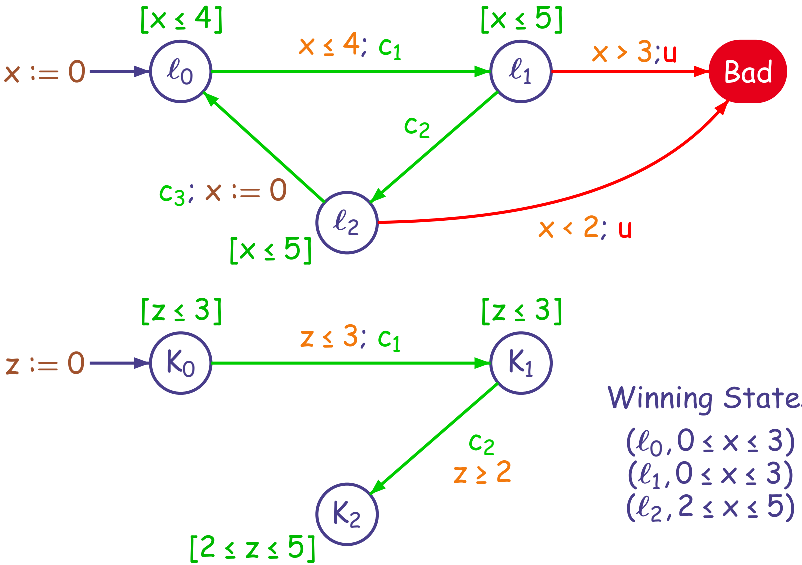
Example of Computation for Safety Games



Winning States

$(\ell_0, 0 \leq x \leq 3)$
$(\ell_1, 0 \leq x \leq 3)$
$(\ell_2, 2 \leq x \leq 5)$

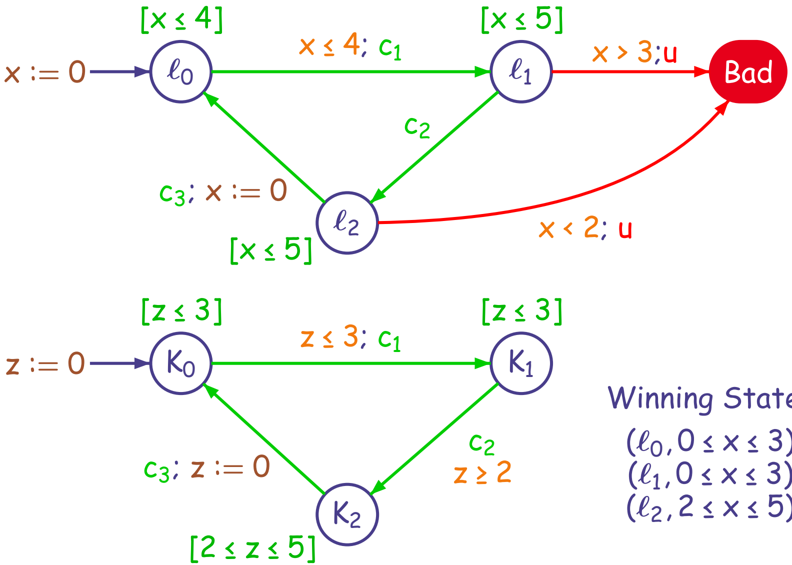
Example of Computation for Safety Games



Winning States

$(\ell_0, 0 \leq x \leq 3)$
$(\ell_1, 0 \leq x \leq 3)$
$(\ell_2, 2 \leq x \leq 5)$

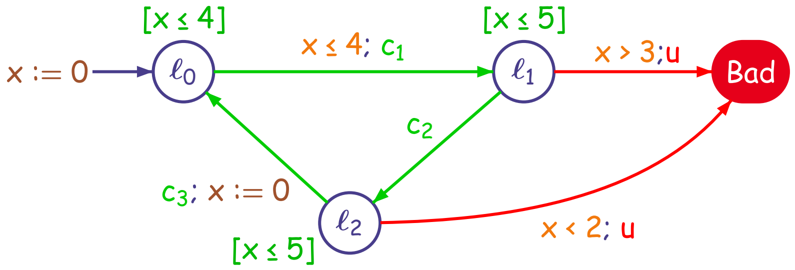
Example of Computation for Safety Games

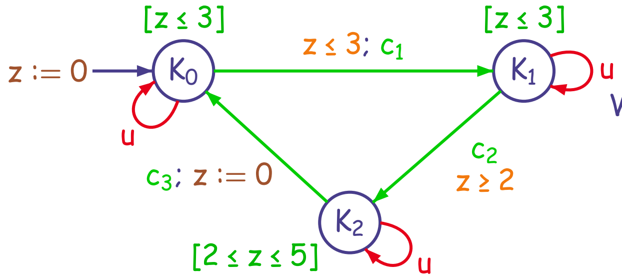


Winning States

$(\ell_0, 0 \le x \le 3)$
$(\ell_1, 0 \le x \le 3)$
$(\ell_2, 2 \le x \le 5)$

Example of Computation for Safety Games

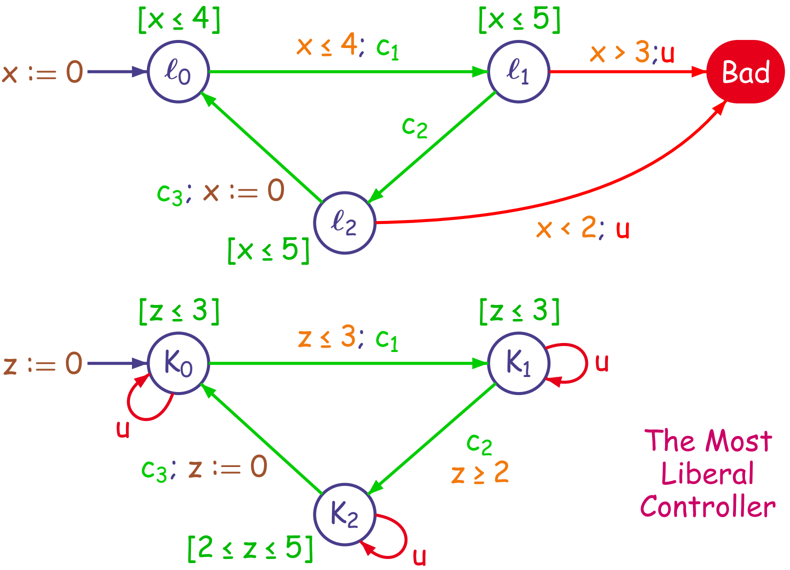




Winning States

$(\ell_0, 0 \leq x \leq 3)$
$(\ell_1, 0 \leq x \leq 3)$
$(\ell_2, 2 \leq x \leq 5)$

Example of Computation for Safety Games



The Most Liberal Controller