Control of Timed Systems

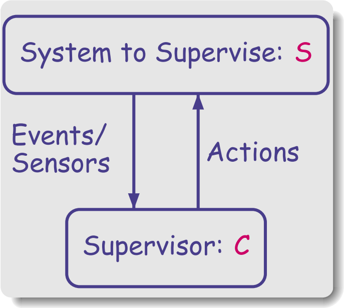
Franck Cassez

CNRS/IRCCyN
Nantes, France

Formalisation des Activités Concurrentes (FAC)
April 3–4, 2008

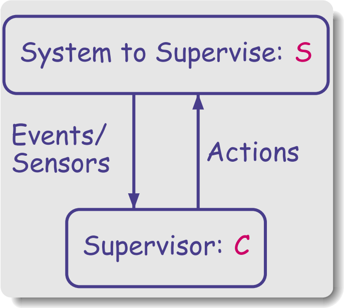
Toulouse, France

Context: Design of Real-Time Systems



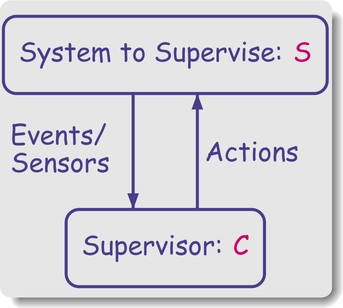
Context: Design of Real-Time Systems

Build Safe Systems



Context: Design of Real-Time Systems

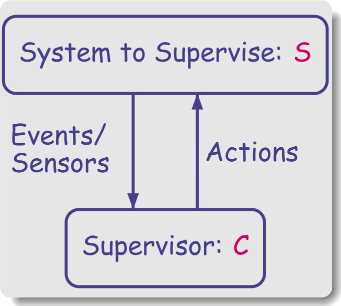
Build Safe Systems



Property φ

Build Safe Systems

Modeling
- Timed Automata
- Time Petri Nets
- Timed Logics



Property φ

Context: Design of Real-Time Systems

Modeling
- Timed Automata
- Time Petri Nets
- Timed Logics

Build Safe Systems

System to Supervise: S

Events/ Sensors

Actions

Supervisor: C

Verification
- Test
- Theorem Proving
- Model-Checking

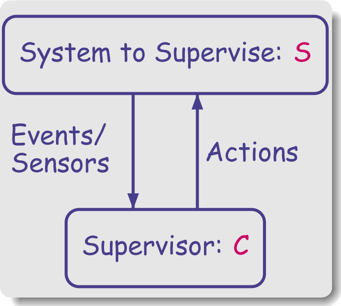
Property φ

Context: Design of Real-Time Systems

Build Safe Systems

Modeling
- Timed Automata
- Time Petri Nets
- Timed Logics

Diagnosis & Control
- Diagnosis
- Control
- Optimal Control



Verification
- Test
- Theorem Proving
- Model-Checking

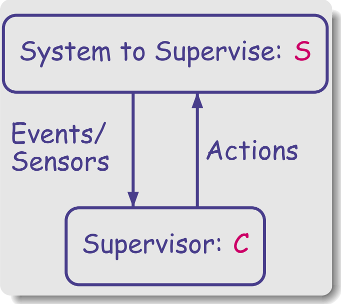
Property φ

Context: Design of Real-Time Systems

Modeling
- Timed Automata
- Time Petri Nets
- Timed Logics

Build Safe Systems

Diagnosis & Control
- Diagnosis
- Control
- Optimal Control



Verification
- Test
- Theorem Proving
- Model-Checking

Property φ

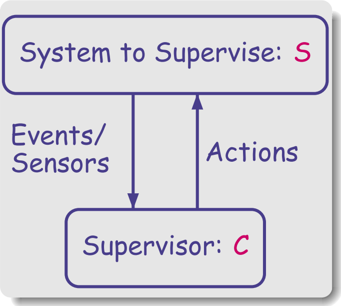
Implementation
- Digital Supervisors
- Continuous Systems

Context: Design of Real-Time Systems

Build Safe Systems

Modeling
- Timed Automata
- Time Petri Nets
- Timed Logics

Diagnosis & Control
- Diagnosis
- Control
- Optimal Control



Verification
- Test
- Theorem Proving
- Model-Checking

Property φ

Implementation
- Digital Supervisors
- Continuous Systems

Outline of the Talk

- Control of Timed Systems: Basics
  - Verification and Control
  - Control $=$ Game
- Control of Discrete Event Systems
  - Games, Strategies, Winning States
  - Controllable Predecessors
  - Results for Finite Games
- Control of Timed Systems
  - Timed Automata
  - Timed Game Automata
  - Symbolic Algorithms for Timed Game Automata
- Advanced Subjects
  - Implementable Controllers
  - Optimal Controllers
  - Efficient Algorithms for Controller Synthesis
- Conclusion

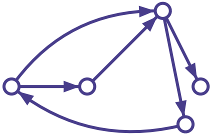
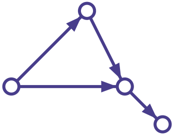
Next:

- Control of Timed Systems: Basics
  - Verification and Control
  - Control = Game
- Control of Discrete Event Systems
- Control of Timed Systems
- Advanced Subjects
- Conclusion

Verification and Control

Verification and Control

Modelling

 $\|$ 

Always (not bad)

$$S \;\|\; C \models \varphi$$

Verification and Control

Does the system meet the specification?

Modelling

Always (not bad)

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

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 $(S \parallel C)$ 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?

Modelling

Always (not bad)

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

Verification and Control

Can we enforce the system to meet the specification?

Modelling

$S \parallel C \models \varphi$

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?

Modelling

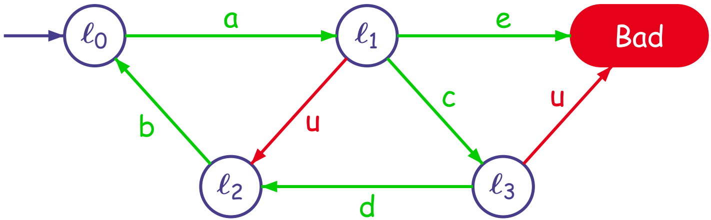
Always (not bad)

$S \parallel ?? \models \varphi$

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 of Discrete Event Systems

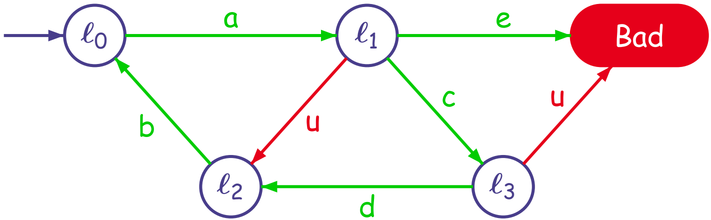


- Introduced by Ramadge & Wonham [Ramadge & Wonham'87]
- Discrete Event System = Finite Automaton with

> Controllable (Act_c) and Uncontrollable (Act_u) actions

- Specification = Control Objective = Language
  e.g. "avoid sequences of actions leading to state Bad"
- How to restrict: disable some controllable transitions
  [Ramadge & Wonham'89, Thistle & Wonham'94]

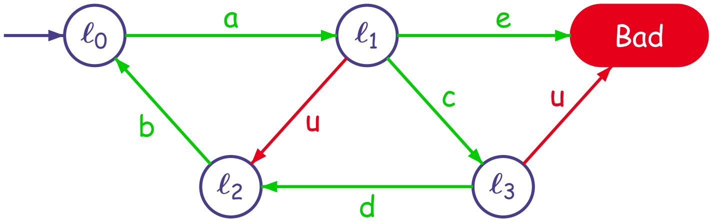
Control as Game



Open System = 2-player game, Controller (C) vs Environment (E)

- Controller plays Act_c moves, Environment plays Act_u moves
- Control Objective = Winning condition on the game
  "Avoid bad states" (safety) or "Enforce good states" (reachability)
- Control Problem: find a strategy (a controller) to win the game
- Various types of game models
  - Finite or pushdown or counter automata ...
  - Timed or hybrid automata

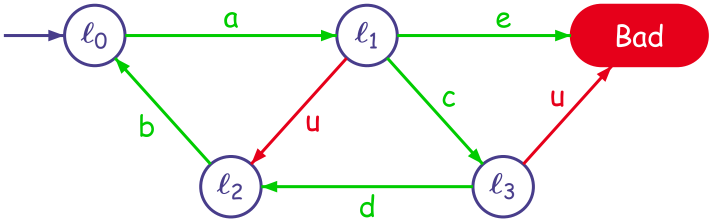
Control as Game



Open System = 2-player game, Controller (C) vs Environment (E)

- Controller plays Act_c moves, Environment plays Act_u moves
- Control Objective = Winning condition on the game
  "Avoid bad states" (safety) or "Enforce good states" (reachability)
- Control Problem: find a strategy (a controller) to win the game
- Various types of game models
  - Finite or pushdown or counter automata ...
  - Timed or hybrid automata

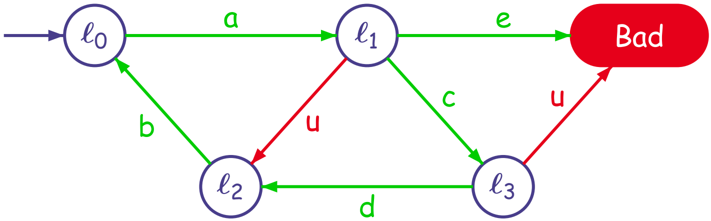
Control as Game



Open System = 2-player game, Controller (C) vs Environment (E)

- Controller plays Act_c moves, Environment plays Act_u moves
- Control Objective = Winning condition on the game
  "Avoid bad states" (safety) or "Enforce good states" (reachability)
- Control Problem: find a strategy (a controller) to win the game
- Various types of game models
  - Finite or pushdown or counter automata ...
  - Timed or hybrid automata

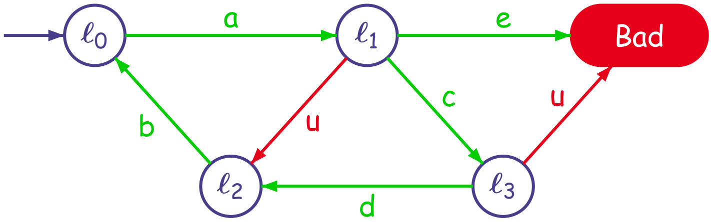
Control as Game



Open System = 2-player game, Controller (C) vs Environment (E)

- Controller plays Act_c moves, Environment plays Act_u moves
- Control Objective = Winning condition on the game
  "Avoid bad states" (safety) or "Enforce good states" (reachability)
- Control Problem: find a strategy (a controller) to win the game
- Various types of game models
  - Finite or pushdown or counter automata ...
  - Timed or hybrid automata

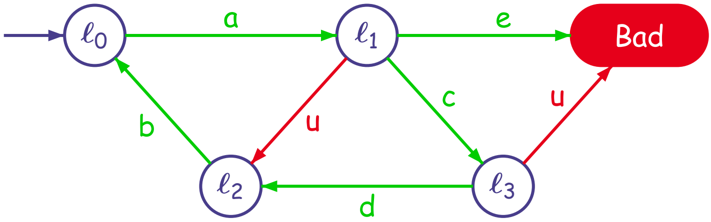
Control as Game



Open System = 2-player game, Controller (C) vs Environment (E)

- Controller plays Act_c moves, Environment plays Act_u moves
- Control Objective = Winning condition on the game
  "Avoid bad states" (safety) or "Enforce good states" (reachability)
- Control Problem: find a strategy (a controller) to win the game
- Various types of game models
  - Finite or pushdown or counter automata ...
  - Timed or hybrid automata

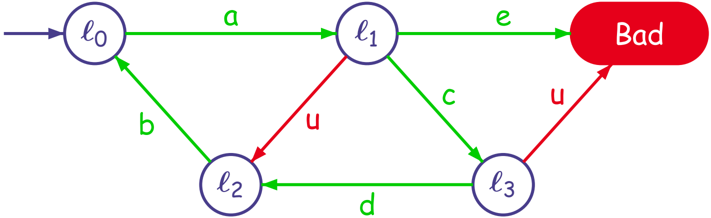
Control as Game



Open System = 2-player game, Controller (C) vs Environment (E)

- Controller plays Act_c moves, Environment plays Act_u moves
- Control Objective = Winning condition on the game
  "Avoid bad states" (safety) or "Enforce good states" (reachability)
- Control Problem: find a strategy (a controller) to win the game
- Various types of game models
  - Finite or pushdown or counter automata ...
  - Timed or hybrid automata

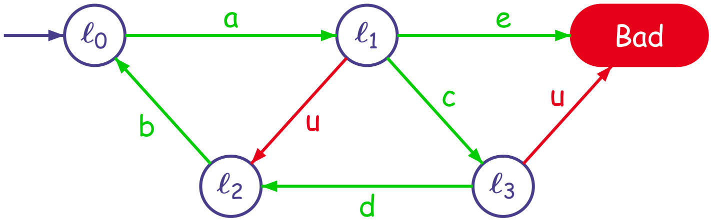
Control as Game



Open System = 2-player game, Controller (C) vs Environment (E)

- Controller plays Act_c moves, Environment plays Act_u moves
- Control Objective = Winning condition on the game
  "Avoid bad states" (safety) or "Enforce good states" (reachability)
- Control Problem: find a strategy (a controller) to win the game
- Various types of game models
  - Finite or pushdown or counter automata ...
  - Timed or hybrid automata

Control as Game



Open System = 2-player game, Controller (C) vs Environment (E)

- Controller plays Act_c moves, Environment plays Act_u moves
- Control Objective = Winning condition on the game
  "Avoid bad states" (safety) or "Enforce good states" (reachability)
- Control Problem: find a strategy (a controller) to win the game
- Various types of game models
  - Finite or pushdown or counter automata ...
  - Timed or hybrid automata

Problems of Interest

Verification Problem (or Model Checking Problem)

Input: a model of the closed system S and a property φ
Problem: Does S satisfy φ ?

Control Problem $CP(G, \varphi)$

Input: a model of the open system (game) G and a property φ
Problem: Is there a strategy (controller) C s.t. $(C \parallel G)$ satisfy φ ?

Control Synthesis Problem (CSP)

Input: a model of the open system (game) G and a property φ
Problem: If the answer to the $CP(G, \varphi)$ is "yes", can we effectively compute a witness controller ?

Problems of Interest

Verification Problem (or Model Checking Problem)

Input: a model of the closed system S and a property φ
Problem: Does S satisfy φ ?

Control Problem $CP(G, \varphi)$

Input: a model of the open system (game) G and a property φ
Problem: Is there a strategy (controller) C s.t. $(C \parallel G)$ satisfy φ ?

Control Synthesis Problem (CSP)

Input: a model of the open system (game) G and a property φ
Problem: If the answer to the $CP(G, \varphi)$ is "yes", can we effectively compute a witness controller ?

Problems of Interest

Verification Problem (or Model Checking Problem)

Input: a model of the closed system S and a property φ
Problem: Does S satisfy φ ?

Control Problem CP(G, φ)

Input: a model of the open system (game) G and a property φ
Problem: Is there a strategy (controller) C s.t. $(C \parallel G)$ satisfy φ ?

Control Synthesis Problem (CSP)

Input: a model of the open system (game) G and a property φ
Problem: If the answer to the CP(G, φ) is “yes”, can we effectively compute a witness controller ?

Problems of Interest

Verification Problem (or Model Checking Problem)

Input: a model of the closed system S and a property φ
Problem: Does S satisfy φ ?

Control Problem $CP(G, \varphi)$

Input: a model of the open system (game) G and a property φ
Problem: Is there a strategy (controller) C s.t. $(C \parallel G)$ satisfy φ ?

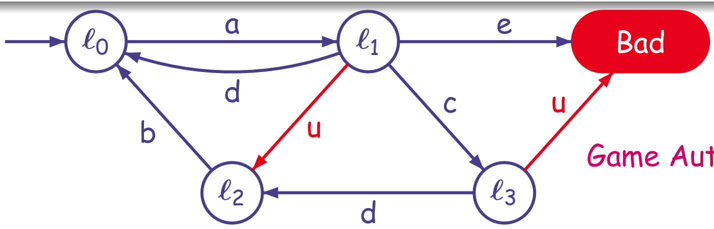
Control Synthesis Problem (CSP)

Input: a model of the open system (game) G and a property φ
Problem: If the answer to the $CP(G, \varphi)$ is "yes", can we effectively compute a witness controller ?

Next:

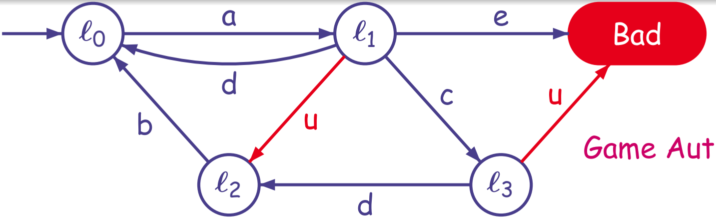
- Control of Timed Systems: Basics
- Control of Discrete Event Systems
  - Games, Strategies, Winning States
  - Controllable Predecessors
  - Results for Finite Games
- Control of Timed Systems
- Advanced Subjects
- Conclusion

Game Automata, Strategies & Winning States



Game Automaton

Game Automata, Strategies & Winning States



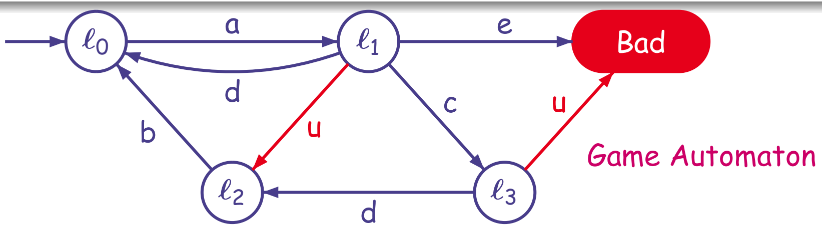
Runs

A run is a finite or infinite sequence of transitions $t_i = (s_i, \sigma_i, s_{i+1})$

$\ell_0 \xrightarrow{a} \ell_1 \xrightarrow{c} \ell_3 \xrightarrow{u} \text{Bad}$

$\ell_0 \xrightarrow{a} \ell_1 \xrightarrow{u} \ell_2 \xrightarrow{b} \ell_0 \xrightarrow{a} \ell_1 \xrightarrow{u} \ell_2 \xrightarrow{b} \quad \cdots$

Game Automata, Strategies & Winning States



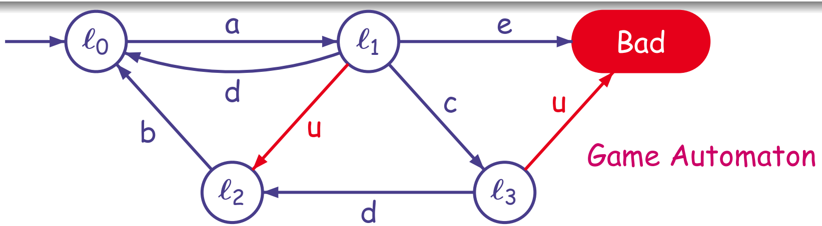
Runs

A run is a finite or infinite sequence of transitions $t_i = (s_i, \sigma_i, s_{i+1})$

$\ell_0 \xrightarrow{a} \ell_1 \xrightarrow{c} \ell_3 \xrightarrow{u} \text{Bad}$

$\ell_0 \xrightarrow{a} \ell_1 \xrightarrow{u} \ell_2 \xrightarrow{b} \ell_0 \xrightarrow{a} \ell_1 \xrightarrow{u} \ell_2 \xrightarrow{b} \quad \cdots$

Game Automata, Strategies & Winning States



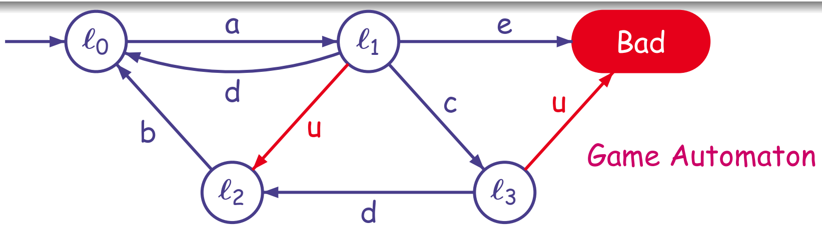
Runs

A run is a finite or infinite sequence of transitions $t_i = (s_i, \sigma_i, s_{i+1})$

$\ell_0 \xrightarrow{a} \ell_1 \xrightarrow{c} \ell_3 \xrightarrow{u} \text{Bad}$

$\ell_0 \xrightarrow{a} \ell_1 \xrightarrow{u} \ell_2 \xrightarrow{b} \ell_0 \xrightarrow{a} \ell_1 \xrightarrow{u} \ell_2 \xrightarrow{b} \quad \cdots$

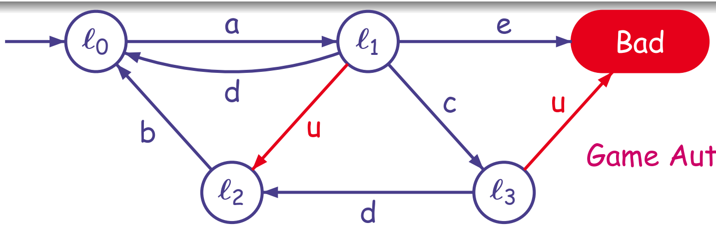
Game Automata, Strategies & Winning States



Strategy

- A strategy f gives for each finite run the controllable action to take. We assume full observability of the system

Game Automata, Strategies & Winning States



Game Automaton

Strategy

- A strategy f gives for each finite run the controllable action to take. We assume full observability of the system

Example of Strategies

$f(\ell_0) = a$

$f(\ell_0 \xrightarrow{a} \ell_1) = c$

$f(\ell_0 \xrightarrow{a} \ell_1 \xrightarrow{u} \ell_2) = b$

$f(\ell_0 \xrightarrow{a} \ell_1 \xrightarrow{u} \ell_2 \xrightarrow{b} \ell_0 \xrightarrow{a} \ell_1) = e$

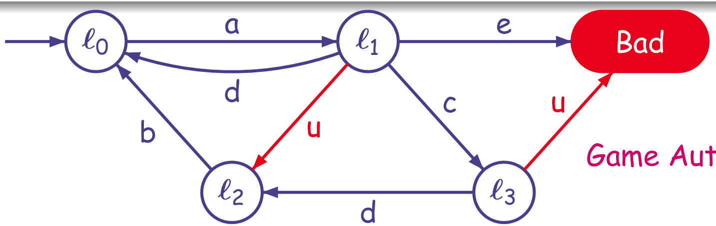
$f'(\rho \longrightarrow \ell_0) = a$

$f'(\rho \longrightarrow \ell_1) = c$

$f'(\rho \longrightarrow \ell_2) = b$

$f'(\rho \longrightarrow \ell_3) = d$

Game Automata, Strategies & Winning States



Strategy

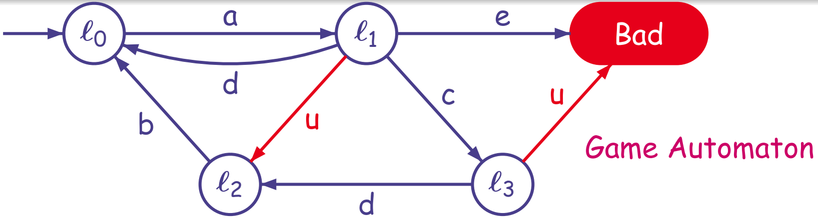
- A strategy f gives for each finite run the controllable action to take. We assume full observability of the system

Example of Strategies

$f(\ell_0) = a$
$f(\ell_0 \xrightarrow{a} \ell_1) = c$
$f(\ell_0 \xrightarrow{a} \ell_1 \xrightarrow{u} \ell_2) = b$
$f(\ell_0 \xrightarrow{a} \ell_1 \xrightarrow{u} \ell_2 \xrightarrow{b} \ell_0 \xrightarrow{a} \ell_1) = e$

$f'(\rho \rightarrow \ell_0) = a$
$f'(\rho \rightarrow \ell_1) = c$
$f'(\rho \rightarrow \ell_2) = b$
$f'(\rho \rightarrow \ell_3) = d$

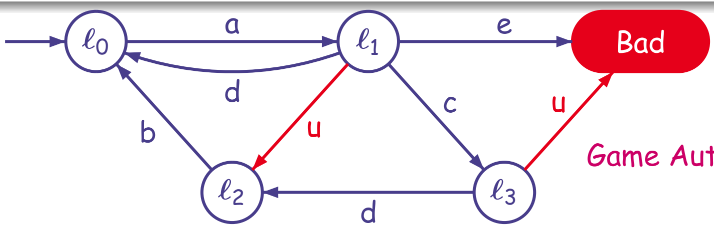
Game Automata, Strategies & Winning States



Strategy

- A strategy f gives for each finite run the controllable action to take. We assume full observability of the system
- A strategy restricts the set of runs of the system.
  from a state s it generates of subset of the runs of the initial game

Game Automata, Strategies & Winning States

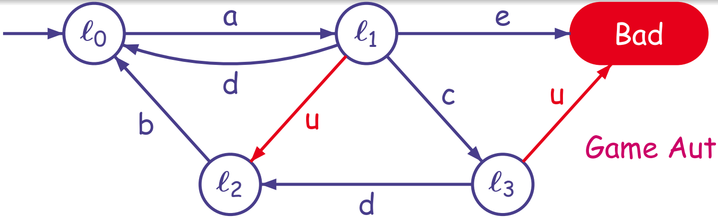


Game Automaton

Strategy

- A strategy f gives for each finite run the controllable action to take. We assume full observability of the system
- A strategy restricts the set of runs of the system.
  from a state s it generates of subset of the runs of the initial game
- A strategy is winning from s if it generates only good runs.

Game Automata, Strategies & Winning States



Game Automaton

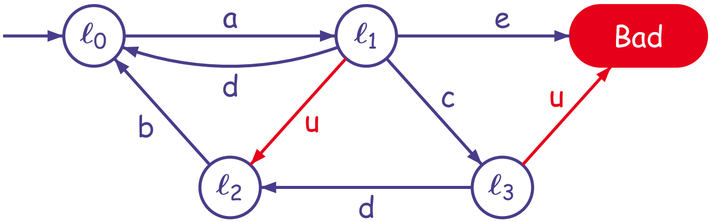
Strategy

- A strategy f gives for each finite run the controllable action to take. We assume full observability of the system
- A strategy restricts the set of runs of the system.
  from a state s it generates of subset of the runs of the initial game
- A strategy is winning from s if it generates only good runs.

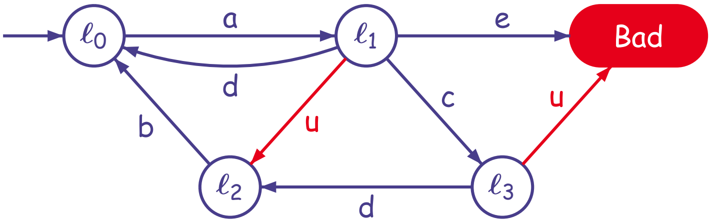
Winning States

A state s is winning if there exists a winning strategy from s.

Controllable Predecessors

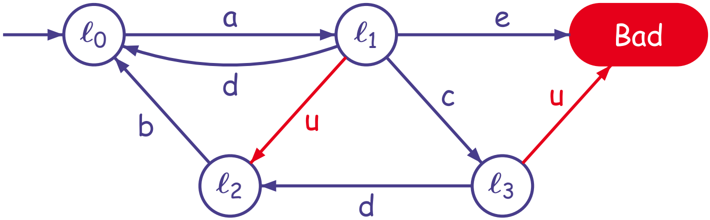


Controllable Predecessors

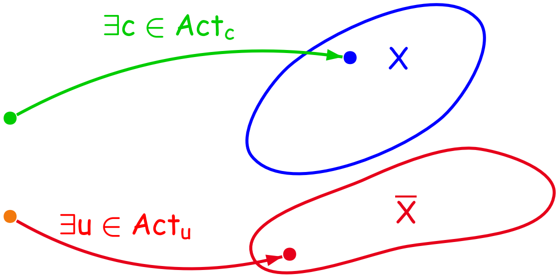


$\pi(X) =$ states from which one can enforce X with a controllable action

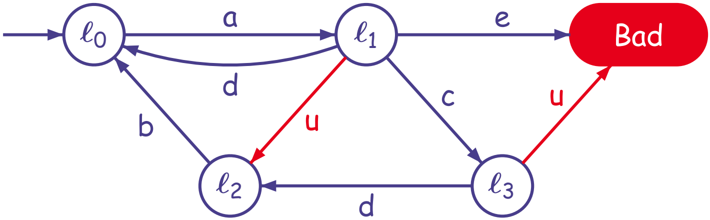
Controllable Predecessors



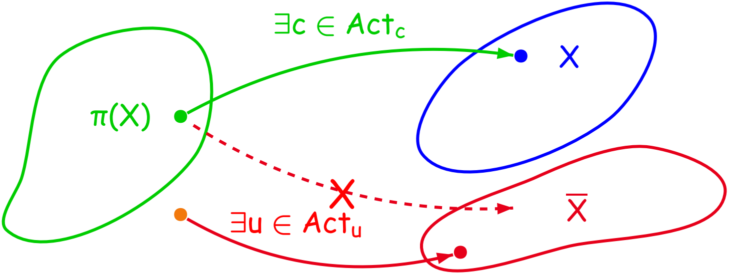
$\pi(X)$ = states from which one can enforce X with a controllable action



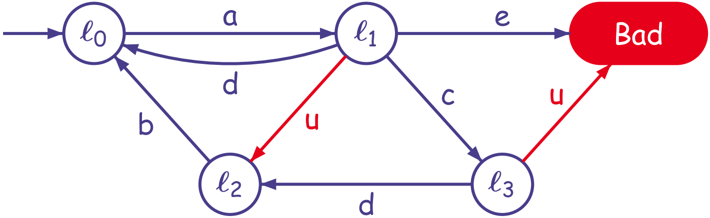
Controllable Predecessors



$\pi(X)$ = states from which one can enforce X with a controllable action



Controllable Predecessors



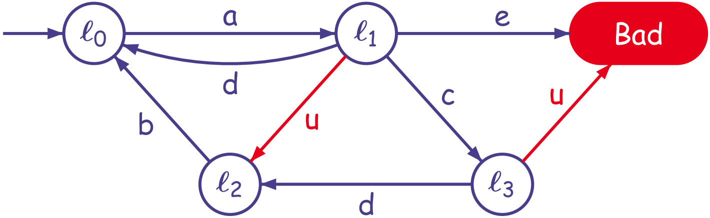
$\pi(X)$ = states from which one can enforce X with a controllable action

$$\pi(X) = \mathrm{Pred}^{Act_c}(X) \setminus \mathrm{Pred}^{Act_u}(\overline{X})$$

Fixpoint Characterization of Winning States for Safety Games:

1. Let φ be a set of safe (good) states and G a game
2. Let W^* be the greatest fixpoint of $h(X) = \varphi \cap \pi(X)$
3. W^* is the set of winning states for (G, φ)

Controllable Predecessors



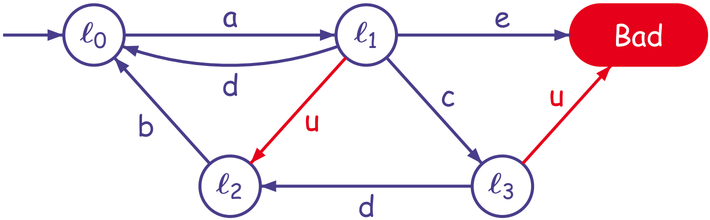
$\pi(X)$ = states from which one can enforce X with a controllable action

$$\pi(X) = \mathrm{Pred}^{Act_c}(X) \setminus \mathrm{Pred}^{Act_u}(\overline{X})$$

Fixpoint Characterization of Winning States for Safety Games:

1. Let φ be a set of safe (good) states and G a game
2. Let W^* be the greatest fixpoint of $h(X) = \varphi \cap \pi(X)$
3. W^* is the set of winning states for (G, φ)

Controllable Predecessors



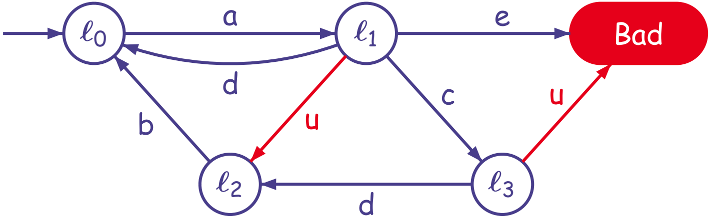
$\pi(X)$ = states from which one can enforce X with a controllable action

$$\pi(X) = \mathrm{Pred}^{Act_c}(X) \setminus \mathrm{Pred}^{Act_u}(\overline{X})$$

Fixpoint Characterization of Winning States for Safety Games:

1. Let φ be a set of safe (good) states and G a game
2. Let W^* be the greatest fixpoint of $h(X) = \varphi \cap \pi(X)$
3. W^* is the set of winning states for (G, φ)

Controllable Predecessors



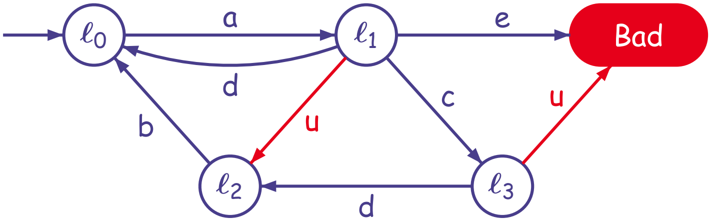
$\pi(X)$ = states from which one can enforce X with a controllable action

$$\pi(X) = Pred^{Act_c}(X) \setminus Pred^{Act_u}(\overline{X})$$

Fixpoint Characterization of Winning States for Safety Games:

1. Let φ be a set of safe (good) states and G a game
2. Let W^* be the greatest fixpoint of $h(X) = \varphi \cap \pi(X)$
3. W^* is the set of winning states for (G, φ)

Controllable Predecessors



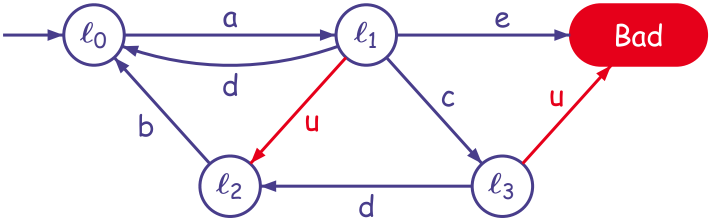
$\pi(X)$ = states from which one can enforce X with a controllable action

$$\pi(X) = \mathrm{Pred}^{Act_c}(X) \setminus \mathrm{Pred}^{Act_u}(\overline{X})$$

Fixpoint Characterization of Winning States for Safety Games:

1. Let φ be a set of safe (good) states and G a game
2. Let W^* be the greatest fixpoint of $h(X) = \varphi \cap \pi(X)$
3. W^* is the set of winning states for (G, φ)
   - Decide CP: check that $\ell_0 \in W^*$

Controllable Predecessors



$\pi(X)$ = states from which one can enforce X with a controllable action

$$\pi(X) = \mathrm{Pred}^{Act_c}(X) \setminus \mathrm{Pred}^{Act_u}(\overline{X})$$

Fixpoint Characterization of Winning States for Safety Games:

1. Let φ be a set of safe (good) states and G a game
2. Let W^* be the greatest fixpoint of $h(X) = \varphi \cap \pi(X)$
3. W^* is the set of winning states for (G, φ)

- Decide CP: check that $\ell_0 \in W^*$
- Synthesis Problem: Given W^* and G, by def. of π we can build a winning strategy

Results for Finite Games

Given G a finite game, φ a control objective

Results for Finite Games

Given G a finite game, φ a control objective

The fixpoint computation of W^* terminates

Results for Finite Games

Given G a finite game, φ a control objective

Theorem (CP is Decidable)

CP is decidable for ω-regular winning conditions.

Results for Finite Games

Given G a finite game, φ a control objective

Theorem (CP is Decidable)

CP is decidable for ω-regular winning conditions.

Theorem (Effectiveness of CSP)

Strategy synthesis is effective. We can compute the most permissive winning strategy.

Results for Finite Games

Given G a finite game, φ a control objective

Theorem (CP is Decidable)

CP is decidable for ω-regular winning conditions.

Theorem (Effectiveness of CSP)

Strategy synthesis is effective. We can compute the most permissive winning strategy.

Theorem (Positional Strategies are Sufficient)

Positional (or memoryless) strategies suffice to win finite-state (turn-based) games with ω-regular winning conditions.
(The number of states of C is $\leq$ number of states of G.)

Time !

- Context : Real-Time Critical Systems
- Some expected properties are quantitative properties
  e.g. scheduling
  or "The system will answer within 10 t.u. after a request is issued"
- One solution: discrete time
  - Can be "expensive"
  - Not natural – Not accurate enough
- Real systems evolve in dense time

Use Dense Time models

Time !

- Context : Real-Time Critical Systems
- Some expected properties are quantitative properties
  e.g. scheduling
  or "The system will answer within 10 t.u. after a request is issued"
- One solution: discrete time
  - Can be "expensive"
  - Not natural – Not accurate enough
- Real systems evolve in dense time

Use Dense Time models

Time !

- Context : Real-Time Critical Systems
- Some expected properties are quantitative properties
  e.g. scheduling
  or "The system will answer within 10 t.u. after a request is issued"
- One solution: discrete time
  - Can be "expensive"
  - Not natural – Not accurate enough
- Real systems evolve in dense time

Use Dense Time models

Time !

- Context : Real-Time Critical Systems
- Some expected properties are quantitative properties
  e.g. scheduling
  or "The system will answer within 10 t.u. after a request is issued"
- One solution: discrete time
  - Can be "expensive"
  - Not natural – Not accurate enough
- Real systems evolve in dense time

Use Dense Time models

Time !

- Context : Real-Time Critical Systems
- Some expected properties are quantitative properties
  e.g. scheduling
  or "The system will answer within 10 t.u. after a request is issued"
- One solution: discrete time
  - Can be "expensive"
  - Not natural – Not accurate enough
- Real systems evolve in dense time

Use Dense Time models

Time !

- Context : Real-Time Critical Systems
- Some expected properties are quantitative properties
  e.g. scheduling
  or "The system will answer within 10 t.u. after a request is issued"
- One solution: discrete time
  - Can be "expensive"
  - Not natural – Not accurate enough
- Real systems evolve in dense time

Use Dense Time models

Time !

- Context : Real-Time Critical Systems
- Some expected properties are quantitative properties
  e.g. scheduling
  or "The system will answer within 10 t.u. after a request is issued"
- One solution: discrete time
  - Can be "expensive"
  - Not natural – Not accurate enough
- Real systems evolve in dense time

Use Dense Time models

Time !

- Context : Real-Time Critical Systems
- Some expected properties are quantitative properties
  e.g. scheduling
  or "The system will answer within 10 t.u. after a request is issued"
- One solution: discrete time
  - Can be "expensive"
  - Not natural – Not accurate enough
- Real systems evolve in dense time

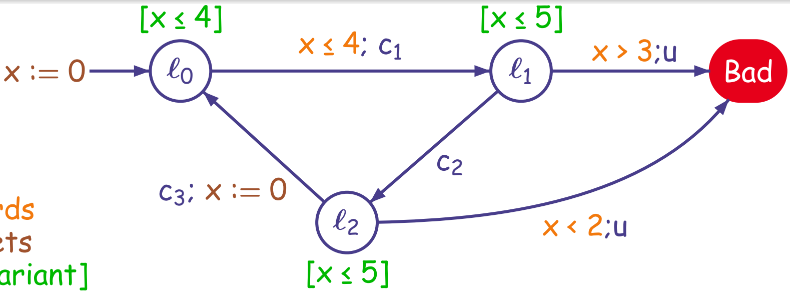
Use Dense Time models

Next:

- Control of Timed Systems: Basics
- Control of Discrete Event Systems
- **Control of Timed Systems**
  - Timed Automata
  - Timed Game Automata
  - Symbolic Algorithms for Timed Game Automata
- Advanced Subjects
- Conclusion

Timed Automaton

[Alur & Dill'94]



Guards
Resets
[Invariant]

Timed Automaton = Finite Automaton + clock variables

Run = sequence of discrete and time steps

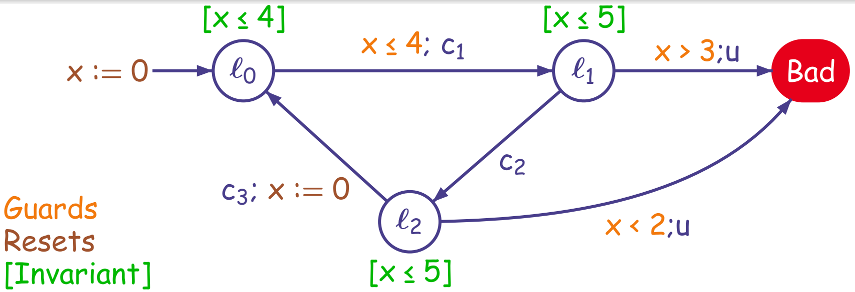
$\rho_1: \quad (\ell_0, 0) \xrightarrow{1.55} (\ell_0, 1.55) \xrightarrow{c_1} (\ell_1, 1.55) \xrightarrow{1.67} (\ell_1, 3.22) \xrightarrow{u} (\mathrm{Bad}, 3.22)$

$\rho_3: \quad (\ell_0, 0) \xrightarrow{c_1c_2c_3 \text{ in } \frac{1}{2}} (\ell_0, 0) \xrightarrow{c_1c_2c_3 \text{ in } \frac{1}{4}} (\ell_0, 0) \xrightarrow{c_1c_2c_3 \text{ in } \frac{1}{8}} \cdots$

Zeno behaviour

Timed Automaton

[Alur & Dill'94]



Timed Automaton = Finite Automaton + clock variables

Run = sequence of discrete and time steps

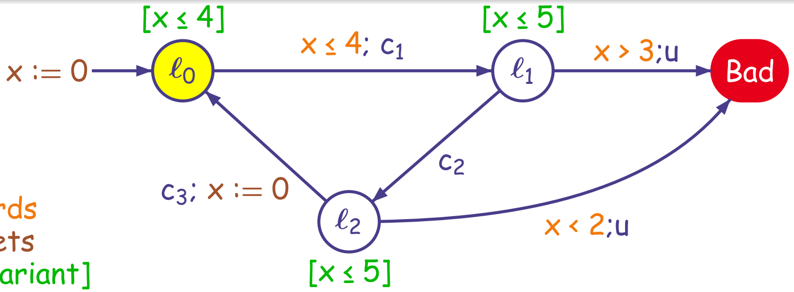
$\rho_1: \quad (\ell_0, 0) \xrightarrow{1.55} (\ell_0, 1.55) \xrightarrow{c_1} (\ell_1, 1.55) \xrightarrow{1.67} (\ell_1, 3.22) \xrightarrow{u} (Bad, 3.22)$

$\rho_3: \quad (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{2}} (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{4}} (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{8}} \cdots$

Zeno behaviour

Timed Automaton

[Alur & Dill'94]



Guards
Resets
[Invariant]

Timed Automaton = Finite Automaton + clock variables

Run = sequence of discrete and time steps

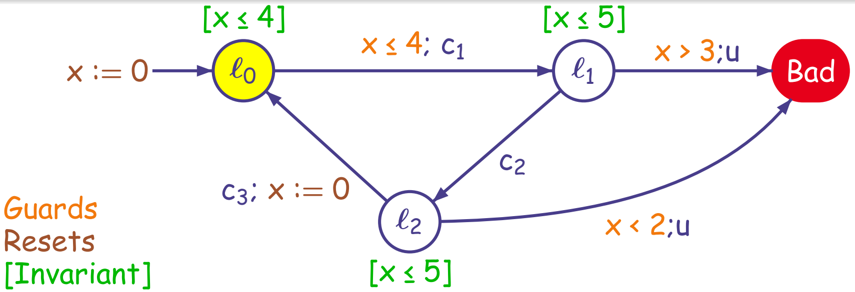
$\rho_1: \quad (\ell_0, 0) \xrightarrow{1.55} (\ell_0, 1.55) \xrightarrow{c_1} (\ell_1, 1.55) \xrightarrow{1.67} (\ell_1, 3.22) \xrightarrow{u} (Bad, 3.22)$

$\rho_3: \quad (\ell_0, 0) \xrightarrow{c_1c_2c_3 \text{ in } \frac{1}{2}} (\ell_0, 0) \xrightarrow{c_1c_2c_3 \text{ in } \frac{1}{4}} (\ell_0, 0) \xrightarrow{c_1c_2c_3 \text{ in } \frac{1}{8}} \cdots$

Zeno behaviour

Timed Automaton

[Alur & Dill'94]



Timed Automaton = Finite Automaton + clock variables

Run = sequence of discrete and time steps

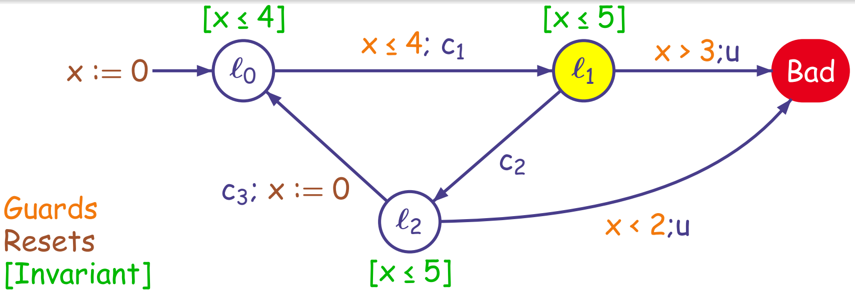
$\rho_1 : \quad (\ell_0, 0) \xrightarrow{1.55} (\ell_0, 1.55) \xrightarrow{c_1} (\ell_1, 1.55) \xrightarrow{1.67} (\ell_1, 3.22) \xrightarrow{u} (Bad, 3.22)$

$\rho_3 : \quad (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{2}} (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{4}} (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{8}} \cdots$

Zeno behaviour

Timed Automaton

[Alur & Dill'94]



Timed Automaton = Finite Automaton + clock variables

Run = sequence of discrete and time steps

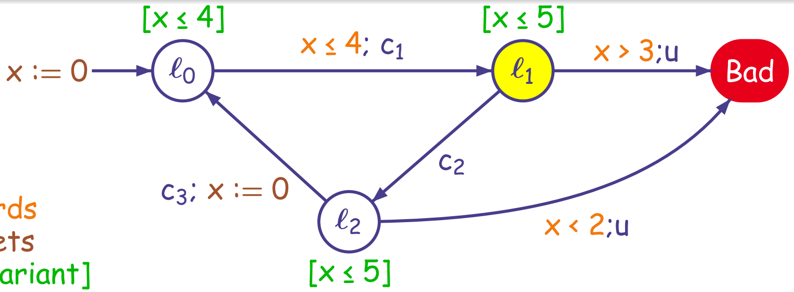
$\rho_1 : \quad (\ell_0, 0) \xrightarrow{1.55} (\ell_0, 1.55) \xrightarrow{c_1} (\ell_1, 1.55) \xrightarrow{1.67} (\ell_1, 3.22) \xrightarrow{u} (Bad, 3.22)$

$\rho_3 : \quad (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{2}} (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{4}} (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{8}} \cdots$

Zeno behaviour

Timed Automaton

[Alur & Dill'94]



Guards
Resets
[Invariant]

Timed Automaton = Finite Automaton + clock variables

Run = sequence of discrete and time steps

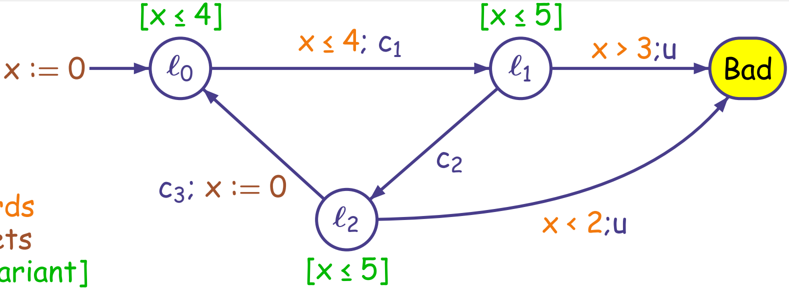
$\rho_1 : \quad (\ell_0, 0) \xrightarrow{1.55} (\ell_0, 1.55) \xrightarrow{c_1} (\ell_1, 1.55) \xrightarrow{1.67} (\ell_1, 3.22) \xrightarrow{u} (\mathrm{Bad}, 3.22)$

$\rho_3 : \quad (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{2}} (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{4}} (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{8}} \cdots$

Zeno behaviour

Timed Automaton

[Alur & Dill'94]



Guards
Resets
[Invariant]

Timed Automaton = Finite Automaton + clock variables

Run = sequence of discrete and time steps

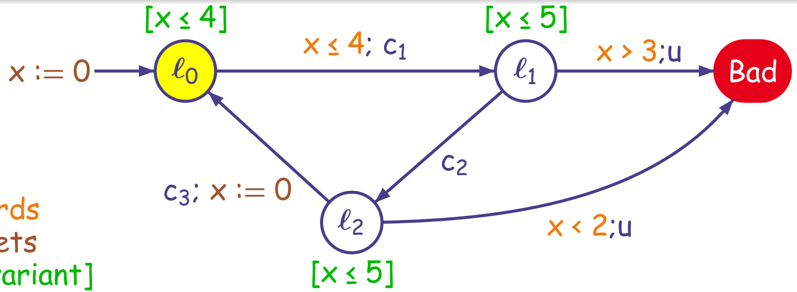
$$\rho_1 : \quad (\ell_0, 0) \xrightarrow{1.55} (\ell_0, 1.55) \xrightarrow{c_1} (\ell_1, 1.55) \xrightarrow{1.67} (\ell_1, 3.22) \xrightarrow{u} (Bad, 3.22)$$

$$\rho_3 : \quad (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{2}} (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{4}} (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{8}} \cdots$$

Zeno behaviour

Timed Automaton

[Alur & Dill'94]



Guards
Resets
[Invariant]

Timed Automaton = Finite Automaton + clock variables

Run = sequence of discrete and time steps

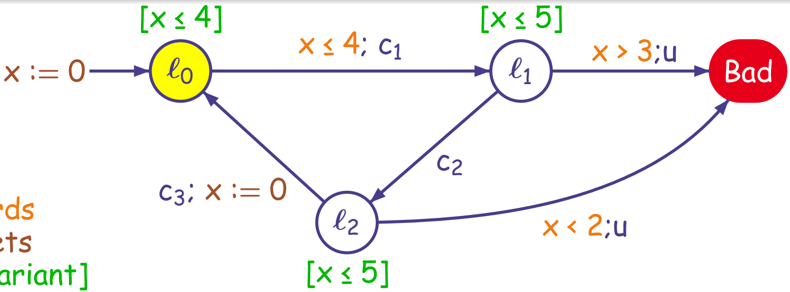
$$\rho_1 : \quad (\ell_0, 0) \xrightarrow{1.55} (\ell_0, 1.55) \xrightarrow{c_1} (\ell_1, 1.55) \xrightarrow{1.67} (\ell_1, 3.22) \xrightarrow{u} (\mathrm{Bad}, 3.22)$$

$$\rho_3 : \quad (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{2}} (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{4}} (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{8}} \cdots$$

Zeno behaviour

Timed Automaton

[Alur & Dill'94]



Guards
Resets
[Invariant]

Timed Automaton = Finite Automaton + clock variables

Run = sequence of discrete and time steps

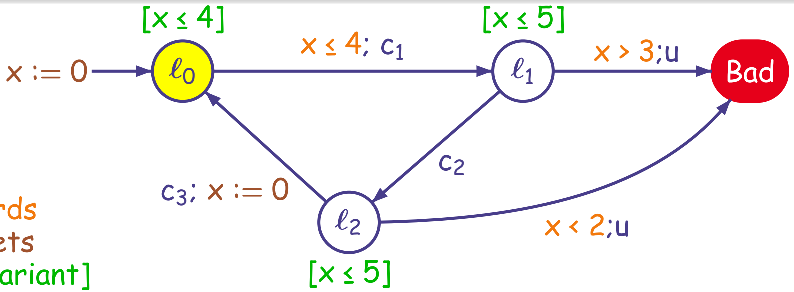
$\rho_1: \quad (\ell_0, 0) \xrightarrow{1.55} (\ell_0, 1.55) \xrightarrow{c_1} (\ell_1, 1.55) \xrightarrow{1.67} (\ell_1, 3.22) \xrightarrow{u} (\mathrm{Bad}, 3.22)$

$\rho_3: \quad (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{2}} (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{4}} (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{8}} \cdots$

Zeno behaviour

Timed Automaton

[Alur & Dill'94]



Guards
Resets
[Invariant]

Timed Automaton = Finite Automaton + clock variables

Run = sequence of discrete and time steps

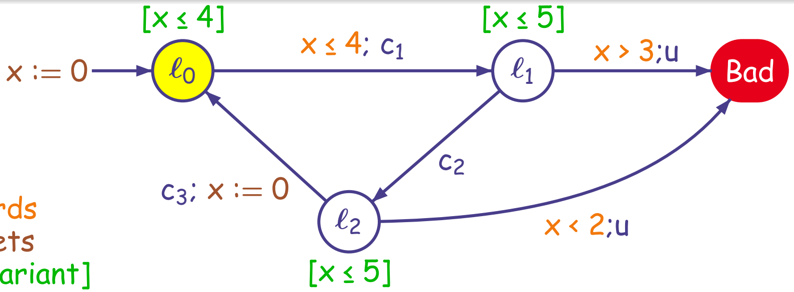
$\rho_1: \quad (\ell_0, 0) \xrightarrow{1.55} (\ell_0, 1.55) \xrightarrow{c_1} (\ell_1, 1.55) \xrightarrow{1.67} (\ell_1, 3.22) \xrightarrow{u} (Bad, 3.22)$

$\rho_3: \quad (\ell_0, 0) \xrightarrow{c_1c_2c_3 \text{ in } \frac{1}{2}} (\ell_0, 0) \xrightarrow{c_1c_2c_3 \text{ in } \frac{1}{4}} (\ell_0, 0) \xrightarrow{c_1c_2c_3 \text{ in } \frac{1}{8}} \cdots$

Zeno behaviour

Timed Automaton

[Alur & Dill'94]



Guards
Resets
[Invariant]

Timed Automaton = Finite Automaton + clock variables

Run = sequence of discrete and time steps

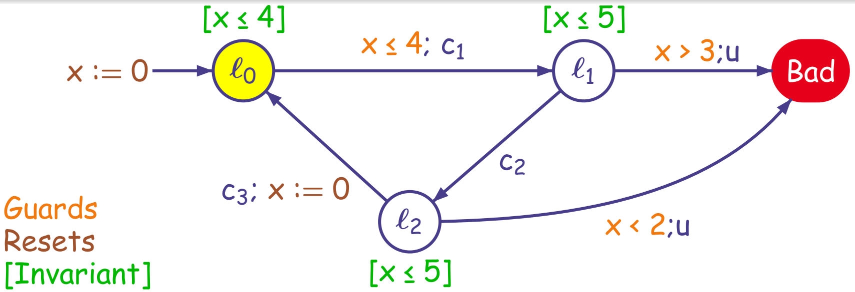
$$\rho_1 : \quad (\ell_0, 0) \xrightarrow{1.55} (\ell_0, 1.55) \xrightarrow{c_1} (\ell_1, 1.55) \xrightarrow{1.67} (\ell_1, 3.22) \xrightarrow{u} (\mathrm{Bad}, 3.22)$$

$$\rho_3 : \quad (\ell_0, 0) \xrightarrow{c_1c_2c_3 \text{ in } \frac{1}{2}} (\ell_0, 0) \xrightarrow{c_1c_2c_3 \text{ in } \frac{1}{4}} (\ell_0, 0) \xrightarrow{c_1c_2c_3 \text{ in } \frac{1}{8}} \cdots$$

Zeno behaviour

Timed Automaton

[Alur & Dill'94]



Timed Automaton = Finite Automaton + clock variables

Run = sequence of discrete and time steps

$$\rho_1 : \quad (\ell_0, 0) \xrightarrow{1.55} (\ell_0, 1.55) \xrightarrow{c_1} (\ell_1, 1.55) \xrightarrow{1.67} (\ell_1, 3.22) \xrightarrow{u} (\mathrm{Bad}, 3.22)$$

$$\rho_3 : \quad (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{2}} (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{4}} (\ell_0, 0) \xrightarrow{c_1 c_2 c_3 \text{ in } \frac{1}{8}} \cdots$$

Zeno behaviour

States & Symbolic States

- $Q = L \times \mathbb{R}_{\geq 0}^{Clock}$ is the set of states of the TA $\qquad q = (\ell, v) \in Q$
- Discrete Successors of $X \subseteq Q$ by an action a:

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

- Time Successors of $X \subseteq Q$:

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

- 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 $(\ell_0, x \leq 2) \cup (\ell_2, x > 0)$

States & Symbolic States

- $Q = L \times \mathbb{R}_{\geq 0}^{Clock}$ is the set of states of the TA $\qquad q = (\ell, v) \in Q$
- Discrete Successors of $X \subseteq Q$ by an action a:
$$Post^a(X) = \{q' \in Q \mid q \xrightarrow{a} q' \text{ and } q \in X\}$$
- Time Successors of $X \subseteq Q$:
$$Post^\delta(X) = \{q' \in Q \mid \exists t \geq 0 \mid q \xrightarrow{t} q' \text{ and } q \in X\}$$
- 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 $(\ell_0, x \leq 2) \cup (\ell_2, x > 0)$

States & Symbolic States

- $Q = L \times \mathbb{R}_{\geq 0}^{Clock}$ is the set of states of the TA $\qquad q = (\ell, v) \in Q$
- Discrete Successors of $X \subseteq Q$ by an action a:
$$Post^a(X) = \{q' \in Q \mid q \xrightarrow{a} q' \text{ and } q \in X\}$$
- Time Successors of $X \subseteq Q$:
$$Post^\delta(X) = \{q' \in Q \mid \exists t \geq 0 \mid q \xrightarrow{t} q' \text{ and } q \in X\}$$
- 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 $(\ell_0, x \leq 2) \cup (\ell_2, x > 0)$

States & Symbolic States

- $Q = L \times \mathbb{R}_{\geq 0}^{Clock}$ is the set of states of the TA $\qquad q = (\ell, v) \in Q$
- Discrete Successors of $X \subseteq Q$ by an action a:
$$Post^a(X) = \{q' \in Q \mid q \xrightarrow{a} q' \text{ and } q \in X\}$$
- Time Successors of $X \subseteq Q$:
$$Post^\delta(X) = \{q' \in Q \mid \exists t \geq 0 \mid q \xrightarrow{t} q' \text{ and } q \in X\}$$
- 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 $Post^a$ and $Post^\delta$

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

States & Symbolic States

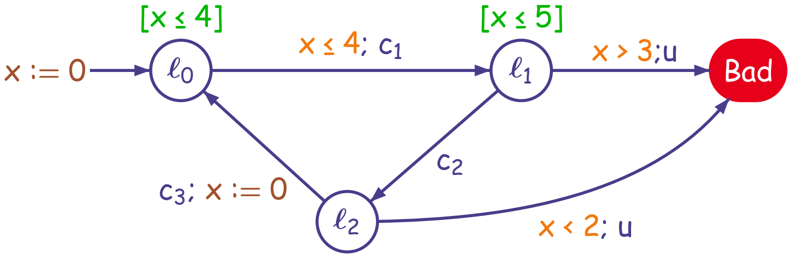
- $Q = L \times \mathbb{R}_{\geq 0}^{Clock}$ is the set of states of the TA $\qquad q = (\ell, v) \in Q$
- Discrete Successors of $X \subseteq Q$ by an action a:
$$Post^a(X) = \{q' \in Q \mid q \xrightarrow{a} q' \text{ and } q \in X\}$$
- Time Successors of $X \subseteq Q$:
$$Post^\delta(X) = \{q' \in Q \mid \exists t \geq 0 \mid q \xrightarrow{t} q' \text{ and } q \in X\}$$
- 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 $(\ell_0, x \leq 2) \cup (\ell_2, x > 0)$

Decidability Result for TA

Region Graph

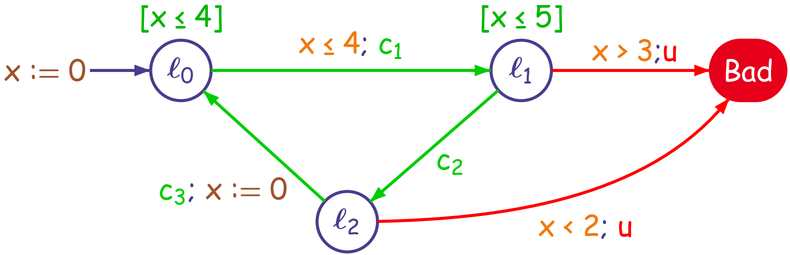
The Reachability Problem for TA is PSPACE-Complete.
Build a finite abstraction: region automaton

Timed Game Automata



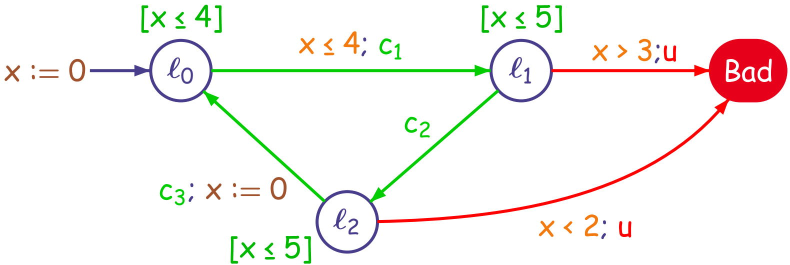
- Introduced by Maler, Pnueli, Sifakis [Maler et al.'95]
- The controller continuously observes the system
  time elapsing and discrete moves are observable
- The controller has the choice between two types of moves:
  - "do nothing" (delay action)
  - "do a controllable action" (among the ones that are possible)
- It can prevent time elapsing by taking a controllable move

Timed Game Automata

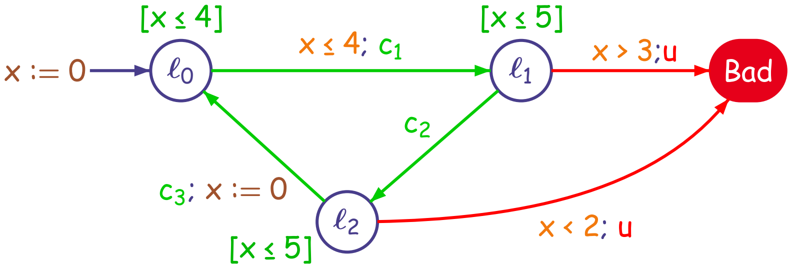


- Introduced by Maler, Pnueli, Sifakis [Maler et al.'95]
- The controller continuously observes the system
  time elapsing and discrete moves are observable
- The controller has the choice between two types of moves:
  - "do nothing" (delay action)
  - "do a controllable action" (among the ones that are possible)
- It can prevent time elapsing by taking a controllable move

Strategies and Winning States



Strategies and Winning States

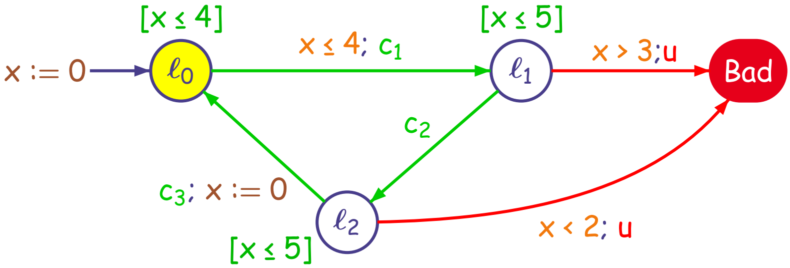


The strategy f: "Wait as long as the system permits"

ρ_1 : $(\ell_0, 0) \xrightarrow{4} (\ell_0, 4) \xrightarrow{c_1} (\ell_1, 4) \xrightarrow{0.5} (\ell_1, 4.5) \xrightarrow{u} (Bad, 4.5)$

ρ_2 : $(\ell_0, 0) \xrightarrow{4} (\ell_0, 4) \xrightarrow{c_1} (\ell_1, 4) \xrightarrow{1.0} (\ell_1, 5) \xrightarrow{c_2} (\ell_2, 5) \xrightarrow{c_3} (\ell_0, 0) \cdots$

Strategies and Winning States

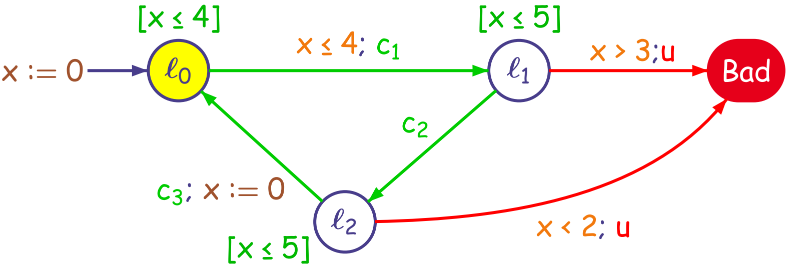


The strategy f: "Wait as long as the system permits"

ρ_1 : $(\ell_0, 0) \xrightarrow{4} (\ell_0, 4) \xrightarrow{c_1} (\ell_1, 4) \xrightarrow{0.5} (\ell_1, 4.5) \xrightarrow{u} (Bad, 4.5)$

ρ_2 : $(\ell_0, 0) \xrightarrow{4} (\ell_0, 4) \xrightarrow{c_1} (\ell_1, 4) \xrightarrow{1.0} (\ell_1, 5) \xrightarrow{c_2} (\ell_2, 5) \xrightarrow{c_3} (\ell_0, 0) \cdots$

Strategies and Winning States

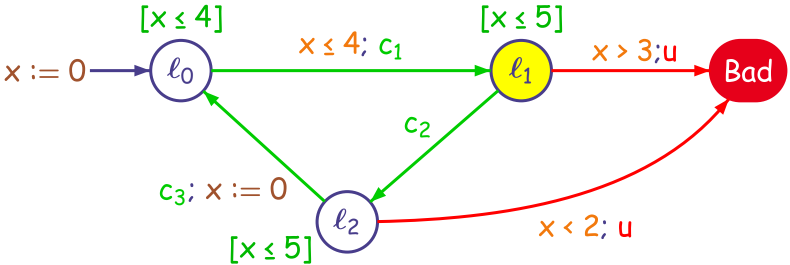


The strategy f: "Wait as long as the system permits"

ρ_1 : $(\ell_0, 0) \xrightarrow{4} (\ell_0, 4) \xrightarrow{c_1} (\ell_1, 4) \xrightarrow{0.5} (\ell_1, 4.5) \xrightarrow{u} (Bad, 4.5)$

ρ_2 : $(\ell_0, 0) \xrightarrow{4} (\ell_0, 4) \xrightarrow{c_1} (\ell_1, 4) \xrightarrow{1.0} (\ell_1, 5) \xrightarrow{c_2} (\ell_2, 5) \xrightarrow{c_3} (\ell_0, 0) \cdots$

Strategies and Winning States

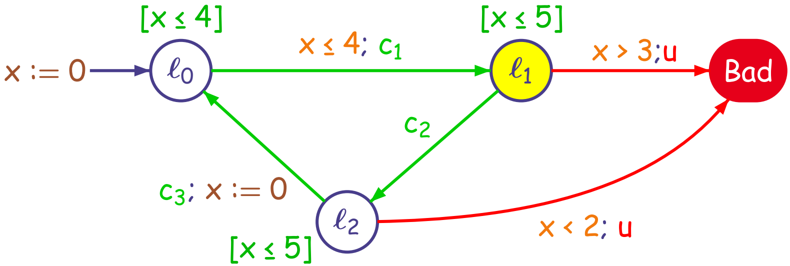


The strategy f: "Wait as long as the system permits"

ρ_1 : $(\ell_0, 0) \xrightarrow{4} (\ell_0, 4) \xrightarrow{c_1} (\ell_1, 4) \xrightarrow{0.5} (\ell_1, 4.5) \xrightarrow{u} (\mathrm{Bad}, 4.5)$

ρ_2 : $(\ell_0, 0) \xrightarrow{4} (\ell_0, 4) \xrightarrow{c_1} (\ell_1, 4) \xrightarrow{1.0} (\ell_1, 5) \xrightarrow{c_2} (\ell_2, 5) \xrightarrow{c_3} (\ell_0, 0) \cdots$

Strategies and Winning States

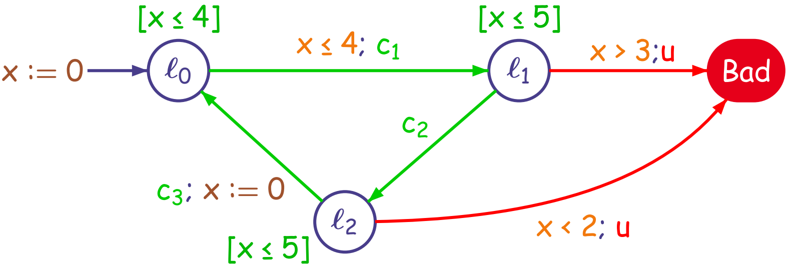


The strategy f: "Wait as long as the system permits"

$\rho_1: \quad (\ell_0, 0) \xrightarrow{4} (\ell_0, 4) \xrightarrow{c_1} (\ell_1, 4) \xrightarrow{0.5} (\ell_1, 4.5) \xrightarrow{u} (Bad, 4.5)$

$\rho_2: \quad (\ell_0, 0) \xrightarrow{4} (\ell_0, 4) \xrightarrow{c_1} (\ell_1, 4) \xrightarrow{1.0} (\ell_1, 5) \xrightarrow{c_2} (\ell_2, 5) \xrightarrow{c_3} (\ell_0, 0) \cdots$

Strategies and Winning States

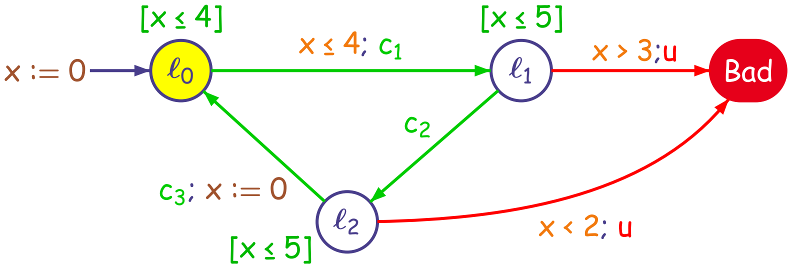


The strategy f: "Wait as long as the system permits"

$\rho_1: \quad (\ell_0, 0) \xrightarrow{4} (\ell_0, 4) \xrightarrow{c_1} (\ell_1, 4) \xrightarrow{0.5} (\ell_1, 4.5) \xrightarrow{u} (\mathrm{Bad}, 4.5)$

$\rho_2: \quad (\ell_0, 0) \xrightarrow{4} (\ell_0, 4) \xrightarrow{c_1} (\ell_1, 4) \xrightarrow{1.0} (\ell_1, 5) \xrightarrow{c_2} (\ell_2, 5) \xrightarrow{c_3} (\ell_0, 0) \cdots$

Strategies and Winning States

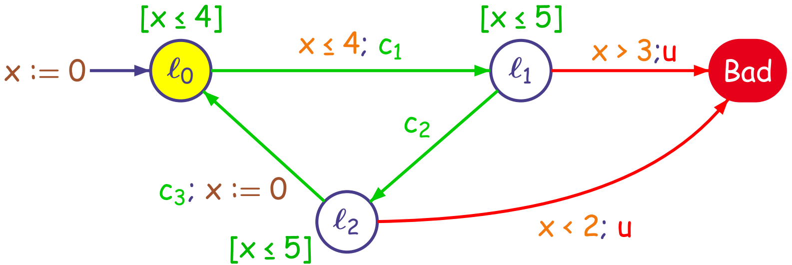


The strategy f: "Wait as long as the system permits"

ρ_1 : $(\ell_0, 0) \xrightarrow{4} (\ell_0, 4) \xrightarrow{c_1} (\ell_1, 4) \xrightarrow{0.5} (\ell_1, 4.5) \xrightarrow{u} (Bad, 4.5)$

ρ_2 : $(\ell_0, 0) \xrightarrow{4} (\ell_0, 4) \xrightarrow{c_1} (\ell_1, 4) \xrightarrow{1.0} (\ell_1, 5) \xrightarrow{c_2} (\ell_2, 5) \xrightarrow{c_3} (\ell_0, 0) \cdots$

Strategies and Winning States

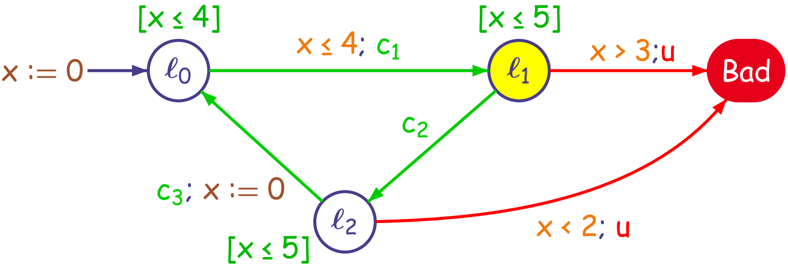


The strategy f: "Wait as long as the system permits"

ρ_1 : $(\ell_0, 0) \xrightarrow{4} (\ell_0, 4) \xrightarrow{c_1} (\ell_1, 4) \xrightarrow{0.5} (\ell_1, 4.5) \xrightarrow{u} (\mathrm{Bad}, 4.5)$

ρ_2 : $(\ell_0, 0) \xrightarrow{4} (\ell_0, 4) \xrightarrow{c_1} (\ell_1, 4) \xrightarrow{1.0} (\ell_1, 5) \xrightarrow{c_2} (\ell_2, 5) \xrightarrow{c_3} (\ell_0, 0) \cdots$

Strategies and Winning States

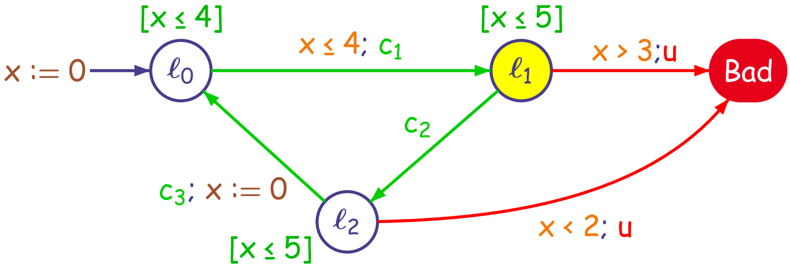


The strategy f: "Wait as long as the system permits"

$\rho_1: \quad (\ell_0,0) \xrightarrow{4} (\ell_0,4) \xrightarrow{c_1} (\ell_1,4) \xrightarrow{0.5} (\ell_1,4.5) \xrightarrow{u} (\mathrm{Bad},4.5)$

$\rho_2: \quad (\ell_0,0) \xrightarrow{4} (\ell_0,4) \xrightarrow{c_1} (\ell_1,4) \xrightarrow{1.0} (\ell_1,5) \xrightarrow{c_2} (\ell_2,5) \xrightarrow{c_3} (\ell_0,0) \cdots$

Strategies and Winning States

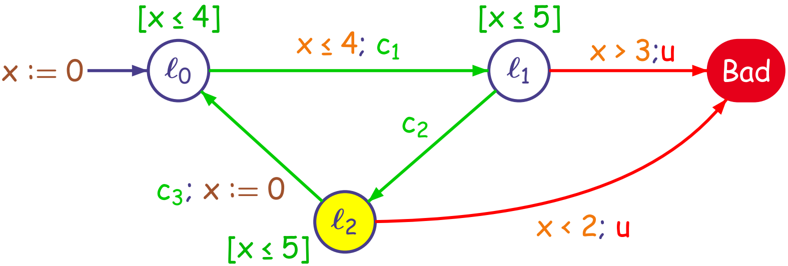


The strategy f: "Wait as long as the system permits"

ρ_1 : $(\ell_0,0) \xrightarrow{4} (\ell_0,4) \xrightarrow{c_1} (\ell_1,4) \xrightarrow{0.5} (\ell_1,4.5) \xrightarrow{u} (Bad,4.5)$

ρ_2 : $(\ell_0,0) \xrightarrow{4} (\ell_0,4) \xrightarrow{c_1} (\ell_1,4) \xrightarrow{1.0} (\ell_1,5) \xrightarrow{c_2} (\ell_2,5) \xrightarrow{c_3} (\ell_0,0) \cdots$

Strategies and Winning States

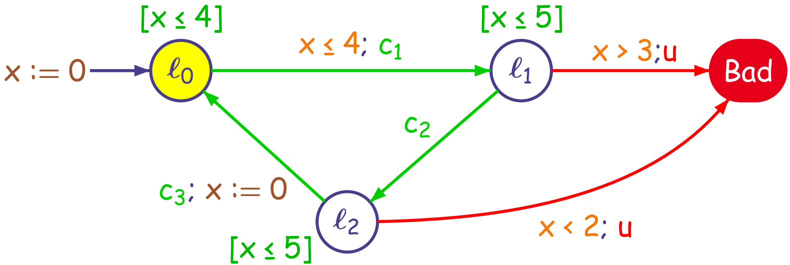


The strategy f: "Wait as long as the system permits"

$\rho_1: \quad (\ell_0,0) \xrightarrow{4} (\ell_0,4) \xrightarrow{c_1} (\ell_1,4) \xrightarrow{0.5} (\ell_1,4.5) \xrightarrow{u} (\mathrm{Bad},4.5)$

$\rho_2: \quad (\ell_0,0) \xrightarrow{4} (\ell_0,4) \xrightarrow{c_1} (\ell_1,4) \xrightarrow{1.0} (\ell_1,5) \xrightarrow{c_2} (\ell_2,5) \xrightarrow{c_3} (\ell_0,0) \cdots$

Strategies and Winning States

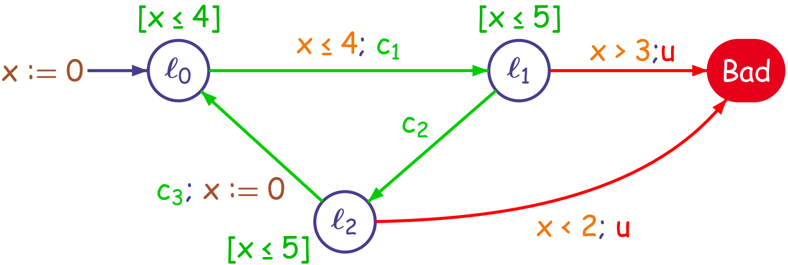


The strategy f: "Wait as long as the system permits"

$\rho_1:\quad (\ell_0,0) \xrightarrow{4} (\ell_0,4) \xrightarrow{c_1} (\ell_1,4) \xrightarrow{0.5} (\ell_1,4.5) \xrightarrow{u} (\text{Bad},4.5)$

$\rho_2:\quad (\ell_0,0) \xrightarrow{4} (\ell_0,4) \xrightarrow{c_1} (\ell_1,4) \xrightarrow{1.0} (\ell_1,5) \xrightarrow{c_2} (\ell_2,5) \xrightarrow{c_3} (\ell_0,0) \cdots$

Strategies and Winning States

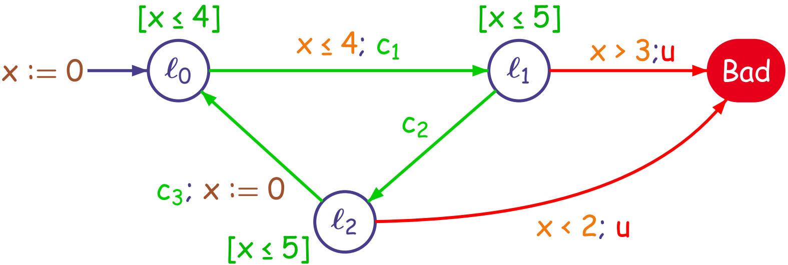


The strategy f: "Wait as long as the system permits"

$\rho_1: \quad (\ell_0,0) \xrightarrow{4} (\ell_0,4) \xrightarrow{c_1} (\ell_1,4) \xrightarrow{0.5} (\ell_1,4.5) \xrightarrow{u} (Bad,4.5)$

$\rho_2: \quad (\ell_0,0) \xrightarrow{4} (\ell_0,4) \xrightarrow{c_1} (\ell_1,4) \xrightarrow{1.0} (\ell_1,5) \xrightarrow{c_2} (\ell_2,5) \xrightarrow{c_3} (\ell_0,0) \cdots$

Strategies and Winning States

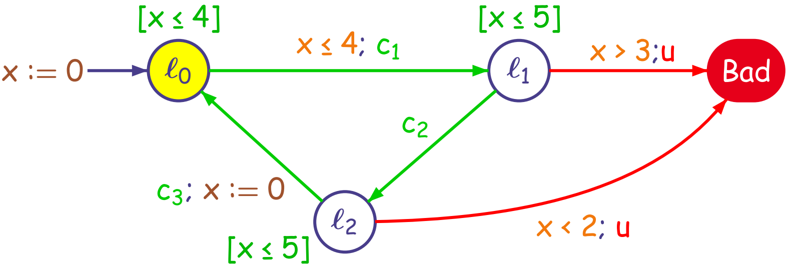


A winning strategy f'

in ℓ_0 at $x = 2$ do c_1; in ℓ_1 at $x = 2.5$ do c_2; in ℓ_2 at $x = 4$ do c_3

$\rho : \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2)$

Strategies and Winning States

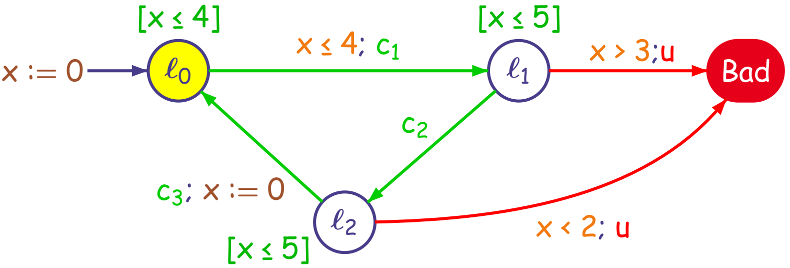


A winning strategy f'

in ℓ_0 at $x = 2$ do c_1; in ℓ_1 at $x = 2.5$ do c_2; in ℓ_2 at $x = 4$ do c_3

$\rho: \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2)$

Strategies and Winning States

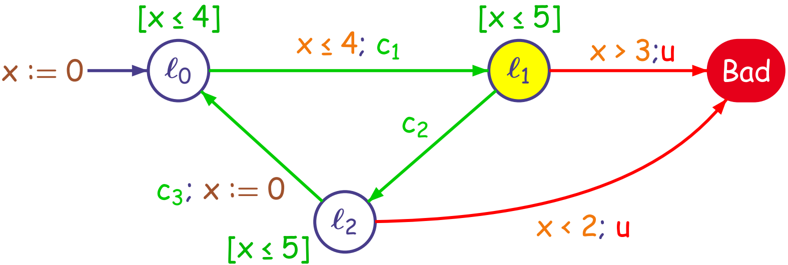


A winning strategy f'

in ℓ_0 at $x = 2$ do c_1; in ℓ_1 at $x = 2.5$ do c_2; in ℓ_2 at $x = 4$ do c_3

$\rho: \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2)$

Strategies and Winning States

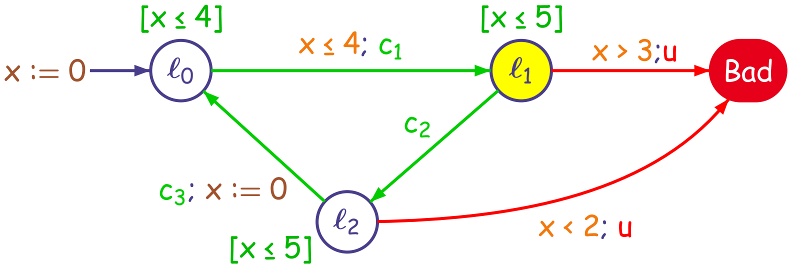


A winning strategy f'

in ℓ_0 at $x = 2$ do c_1; in ℓ_1 at $x = 2.5$ do c_2; in ℓ_2 at $x = 4$ do c_3

$\rho: \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2)$

Strategies and Winning States

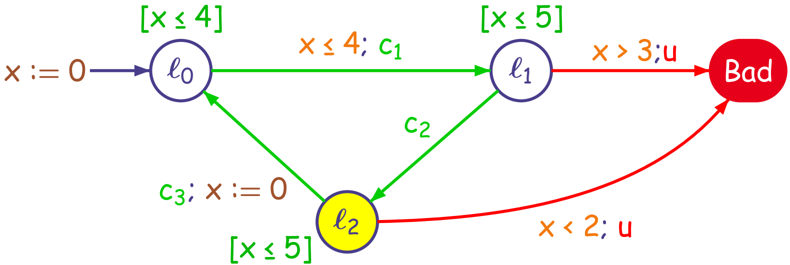


A winning strategy f'

in ℓ_0 at $x = 2$ do c_1; in ℓ_1 at $x = 2.5$ do c_2; in ℓ_2 at $x = 4$ do c_3

$\rho : \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2) \xrightarrow{0.5} (\ell_1, 2.5)$

Strategies and Winning States

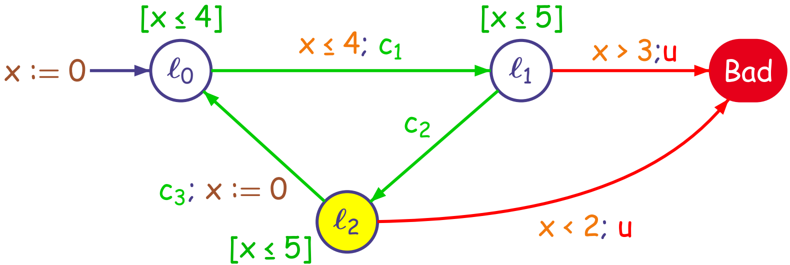


A winning strategy f'

in ℓ_0 at $x = 2$ do c_1; in ℓ_1 at $x = 2.5$ do c_2; in ℓ_2 at $x = 4$ do c_3

$\rho: \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2) \xrightarrow{0.5} (\ell_1, 2.5) \xrightarrow{c_2} (\ell_2, 2.5)$

Strategies and Winning States

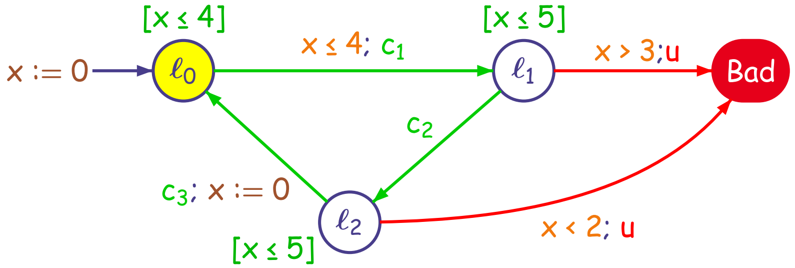


A winning strategy f'

in ℓ_0 at $x = 2$ do c_1; in ℓ_1 at $x = 2.5$ do c_2; in ℓ_2 at $x = 4$ do c_3

$$\rho : \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2) \xrightarrow{0.5} (\ell_1, 2.5) \xrightarrow{c_2} (\ell_2, 2.5) \xrightarrow{1.5} (\ell_2, 4)$$

Strategies and Winning States

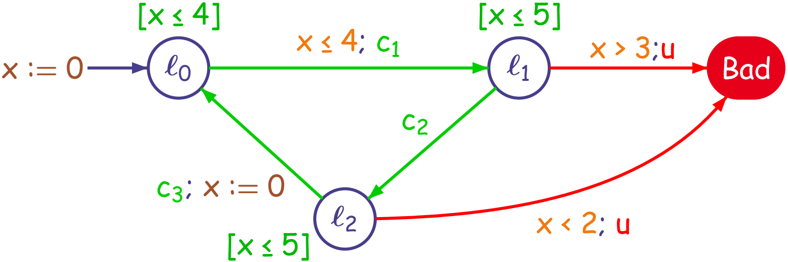


A winning strategy f'

in ℓ_0 at $x = 2$ do c_1; in ℓ_1 at $x = 2.5$ do c_2; in ℓ_2 at $x = 4$ do c_3

$\rho:\ (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2) \xrightarrow{0.5} (\ell_1, 2.5) \xrightarrow{c_2} (\ell_2, 2.5) \xrightarrow{1.5} (\ell_2, 4) \xrightarrow{c_3} (\ell_0, 0)$

Strategies and Winning States

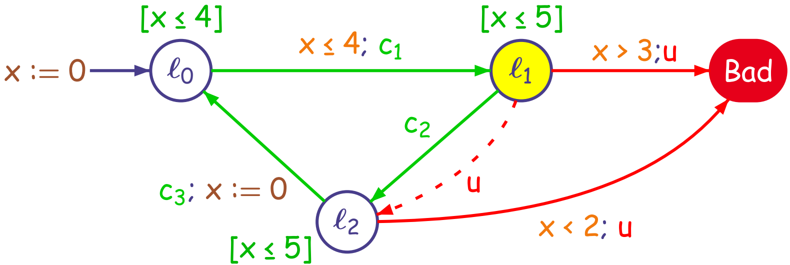


A winning strategy f'

in ℓ_0 at $x = 2$ do c_1; in ℓ_1 at $x = 2.5$ do c_2; in ℓ_2 at $x = 4$ do c_3

$$\rho: \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2) \xrightarrow{0.5} (\ell_1, 2.5) \xrightarrow{c_2} (\ell_2, 2.5) \xrightarrow{1.5} (\ell_2, 4) \xrightarrow{c_3} (\ell_0, 0) \cdots$$

Strategies and Winning States

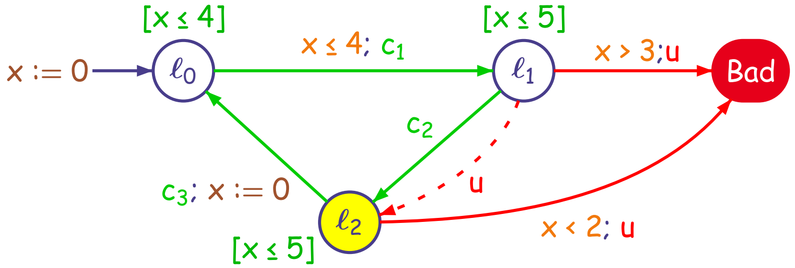


A winning strategy f'

in ℓ_0 at $x = 2$ do c_1; in ℓ_1 at $x = 2.5$ do c_2; in ℓ_2 at $x = 4$ do c_3

$\rho : \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2)$

Strategies and Winning States

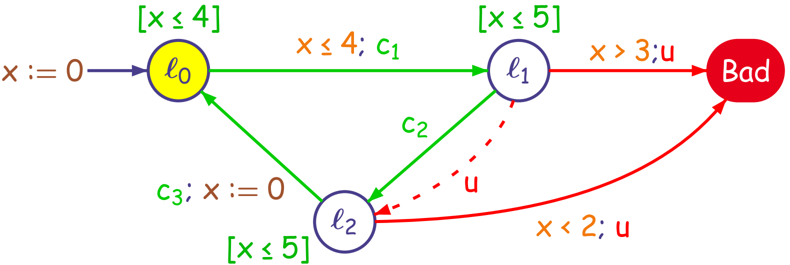


A winning strategy f'

in ℓ_0 at $x = 2$ do c_1; in ℓ_1 at $x = 2.5$ do c_2; in ℓ_2 at $x = 4$ do c_3

$\rho : \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2) \xrightarrow{u \text{ at } \delta \le 0.5} (\ell_2, 2 + \delta)$

Strategies and Winning States

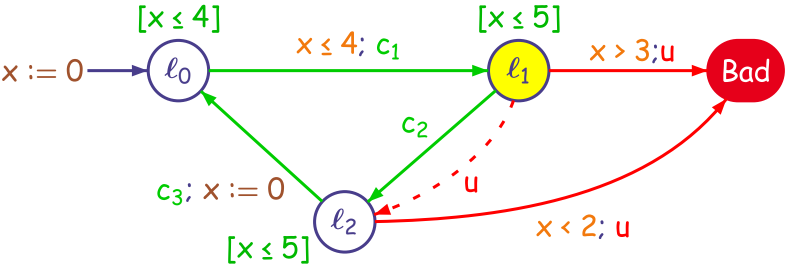


A winning strategy f'

in ℓ_0 at $x = 2$ do c_1; in ℓ_1 at $x = 2.5$ do c_2; in ℓ_2 at $x = 4$ do c_3

$\rho: \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2) \xrightarrow{u \text{ at } \delta \leq 0.5} (\ell_2, 2+\delta) \xrightarrow{c_3 \text{ at } 2-\delta} (\ell_0, 0)$

Strategies and Winning States



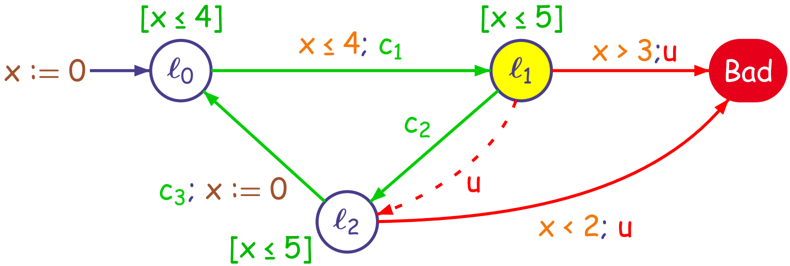
A winning strategy f'

in ℓ_0 at $x = 2$ do c_1; in ℓ_1 at $x = 2.5$ do c_2; in ℓ_2 at $x = 4$ do c_3

$\rho : \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2) \xrightarrow{u \text{ at } \delta \leq 0.5} (\ell_2, 2 + \delta) \xrightarrow{c_3 \text{ at } 2-\delta} (\ell_0, 0)$

$\rho' : \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2)$

Strategies and Winning States



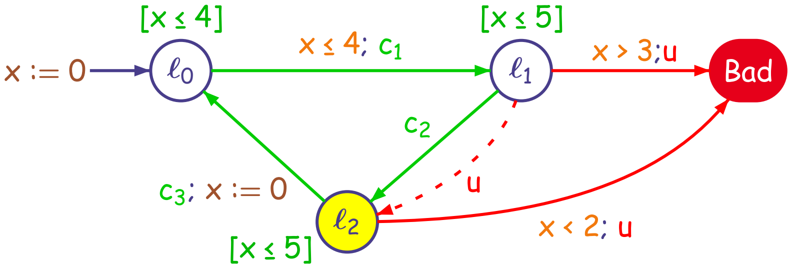
A winning strategy f'

in ℓ_0 at $x = 2$ do c_1; in ℓ_1 at $x = 2.5$ do c_2; in ℓ_2 at $x = 4$ do c_3

$\rho: \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2) \xrightarrow{u \text{ at } \delta \leq 0.5} (\ell_2, 2+\delta) \xrightarrow{c_3 \text{ at } 2-\delta} (\ell_0, 0)$

$\rho': \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2) \xrightarrow{0.5} (\ell_1, 2.5)$

Strategies and Winning States



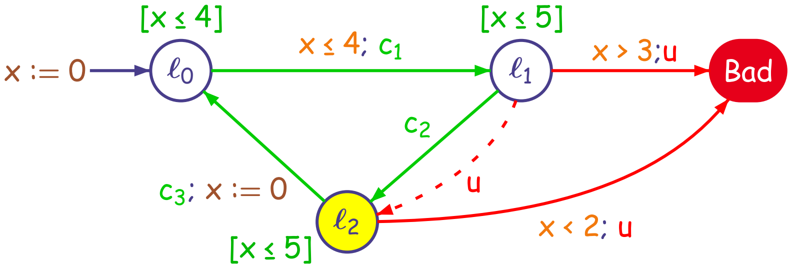
A winning strategy f'

in ℓ_0 at $x = 2$ do c_1; in ℓ_1 at $x = 2.5$ do c_2; in ℓ_2 at $x = 4$ do c_3

$\rho: \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2) \xrightarrow{u \text{ at } \delta \leq 0.5} (\ell_2, 2+\delta) \xrightarrow{c_3 \text{ at } 2-\delta} (\ell_0, 0)$

$\rho': \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2) \xrightarrow{0.5} (\ell_1, 2.5) \xrightarrow{c_2} (\ell_2, 2.5)$

Strategies and Winning States



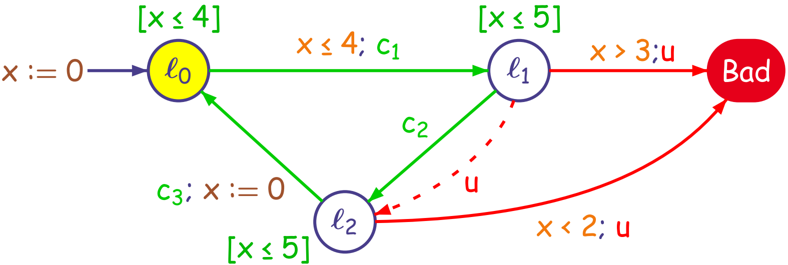
A winning strategy f'

in ℓ_0 at $x = 2$ do c_1; in ℓ_1 at $x = 2.5$ do c_2; in ℓ_2 at $x = 4$ do c_3

$\rho: \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2) \xrightarrow{u \text{ at } \delta \leq 0.5} (\ell_2, 2+\delta) \xrightarrow{c_3 \text{ at } 2-\delta} (\ell_0, 0)$

$\rho': \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2) \xrightarrow{0.5} (\ell_1, 2.5) \xrightarrow{c_2} (\ell_2, 2.5) \xrightarrow{1.5} (\ell_2, 4)$

Strategies and Winning States



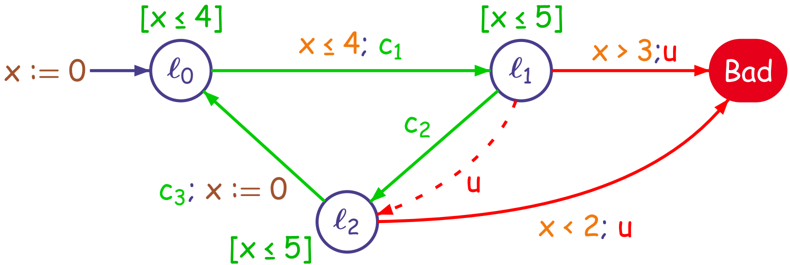
A winning strategy f'

in ℓ_0 at $x = 2$ do c_1; in ℓ_1 at $x = 2.5$ do c_2; in ℓ_2 at $x = 4$ do c_3

$\rho: \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2) \xrightarrow{u \text{ at } \delta \leq 0.5} (\ell_2, 2+\delta) \xrightarrow{c_3 \text{ at } 2-\delta} (\ell_0, 0)$

$\rho': \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2) \xrightarrow{0.5} (\ell_1, 2.5) \xrightarrow{c_2} (\ell_2, 2.5) \xrightarrow{1.5} (\ell_2, 4) \xrightarrow{c_3} (\ell_0, 0) \cdots$

Strategies and Winning States



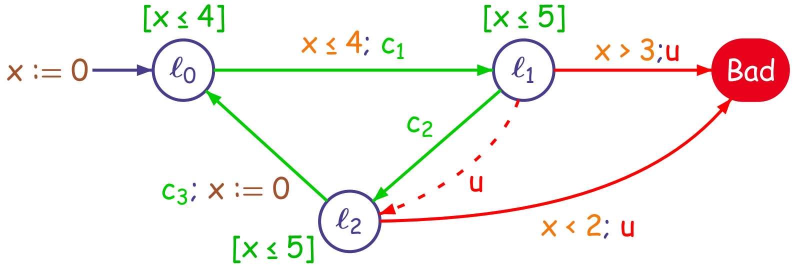
A winning strategy f'

in ℓ_0 at $x = 2$ do c_1; in ℓ_1 at $x = 2.5$ do c_2; in ℓ_2 at $x = 4$ do c_3

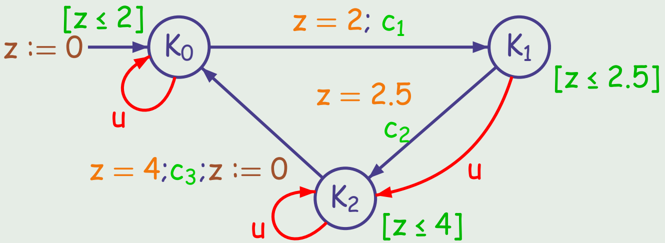
$\rho: \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2) \xrightarrow{u \text{ at } \delta \leq 0.5} (\ell_2, 2+\delta) \xrightarrow{c_3 \text{ at } 2-\delta} (\ell_0, 0)$

$\rho': \quad (\ell_0, 0) \xrightarrow{2} (\ell_0, 2) \xrightarrow{c_1} (\ell_1, 2) \xrightarrow{0.5} (\ell_1, 2.5) \xrightarrow{c_2} (\ell_2, 2.5) \xrightarrow{1.5} (\ell_2, 4) \xrightarrow{c_3} (\ell_0, 0) \cdots$

Strategies and Winning States



The Strategy f' as a Timed Automaton



Controllable Predecessors

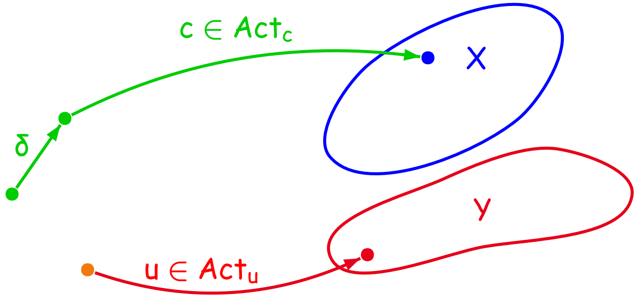
$\pi(X, Y)$ = states from which one can enforce X and avoid Y by:
time elapsing followed by a controllable action

Fixpoint Characterization of Winning States for Safety Games:

1. Let φ be a set of safe (good) states and G a game
2. Let W^* be the greatest fixpoint of $h(X) = \varphi \cap \pi(X, \overline{X})$
3. W^* is the set of winning states for (G, φ)

Controllable Predecessors

$\pi(X, Y)$ = states from which one can enforce X and avoid Y by:
time elapsing followed by a controllable action

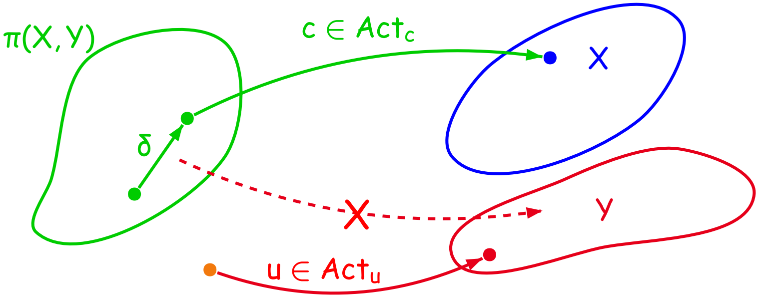


Fixpoint Characterization of Winning States for Safety Games:

1. Let φ be a set of safe (good) states and G a game
2. Let W^* be the greatest fixpoint of $h(X) = \varphi \cap \pi(X, \overline{X})$
3. W^* is the set of winning states for (G, φ)

Controllable Predecessors

$\pi(X, Y)$ = states from which one can enforce X and avoid Y by: time elapsing followed by a controllable action

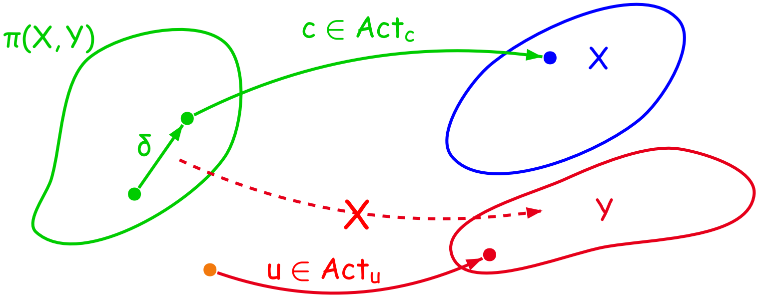


Fixpoint Characterization of Winning States for Safety Games:

1. Let φ be a set of safe (good) states and G a game
2. Let W^* be the greatest fixpoint of $h(X) = \varphi \cap \pi(X, \overline{X})$
3. W^* is the set of winning states for (G, φ)

Controllable Predecessors

$\pi(X, Y)$ = states from which one can enforce X and avoid Y by:
time elapsing followed by a controllable action



Fixpoint Characterization of Winning States for Safety Games:

1. Let φ be a set of safe (good) states and G a game
2. Let W^* be the greatest fixpoint of $h(X) = \varphi \cap \pi(X, \overline{X})$
3. W^* is the set of winning states for (G, φ)

Symbolic Algorithms for Safety Control

[Maler et al.'95, De Alfaro et al.'01]

Details & Example

1. There is a symbolic version for $\pi(X, Y)$
2. $\Longrightarrow$ there is a symbolic version for $h(X)$

Symbolic Algorithms for Safety Control

[Maler et al.'95, De Alfaro et al.'01]

Details & Example

1. There is a symbolic version for $\pi(X,Y)$
2. $\Longrightarrow$ there is a symbolic version for $h(X)$

- Control Problem (CP): check that $(\ell_0, 0) \in W^*$
- Control Synthesis Problem (CSP): by definition of π there is a strategy

Symbolic Algorithms for Safety Control

[Maler et al.'95, De Alfaro et al.'01]

▸ Details & Example

1. There is a symbolic version for $\pi(X, Y)$
2. $\Longrightarrow$ there is a symbolic version for $h(X)$

Theorem (Termination)

The iterative computation of W^* terminates for (G, φ) with G a timed game automaton φ a ω-regular winning condition.

Symbolic Algorithms for Safety Control

[Maler et al.'95, De Alfaro et al.'01]

Details & Example

1. There is a symbolic version for $\pi(X, Y)$
2. $\Longrightarrow$ there is a symbolic version for $h(X)$

Theorem (Termination)

The iterative computation of W^* terminates for (G, φ) with G a timed game automaton φ a ω-regular winning condition.

Theorem (Decidability of CP for Timed Game Automata)

The (Safety) Control Problem is decidable.

Symbolic Algorithms for Safety Control

[Maler et al.'95, De Alfaro et al.'01]

Details & Example

1. There is a symbolic version for $\pi(X, Y)$
2. $\Longrightarrow$ there is a symbolic version for $h(X)$

Theorem (Termination)

The iterative computation of W^* terminates for (G, φ) with G a timed game automaton φ a ω-regular winning condition.

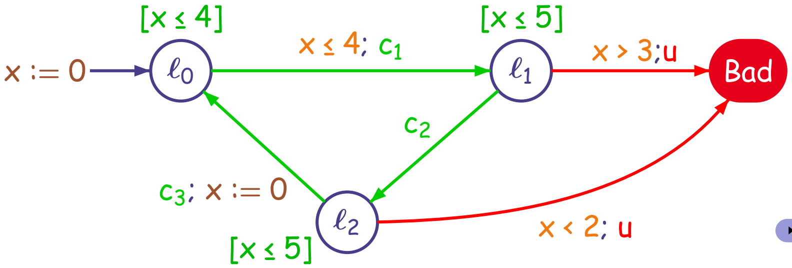
Theorem (Decidability of CP for Timed Game Automata)

The (Safety) Control Problem is decidable.

Theorem (Effectiveness of CSP)

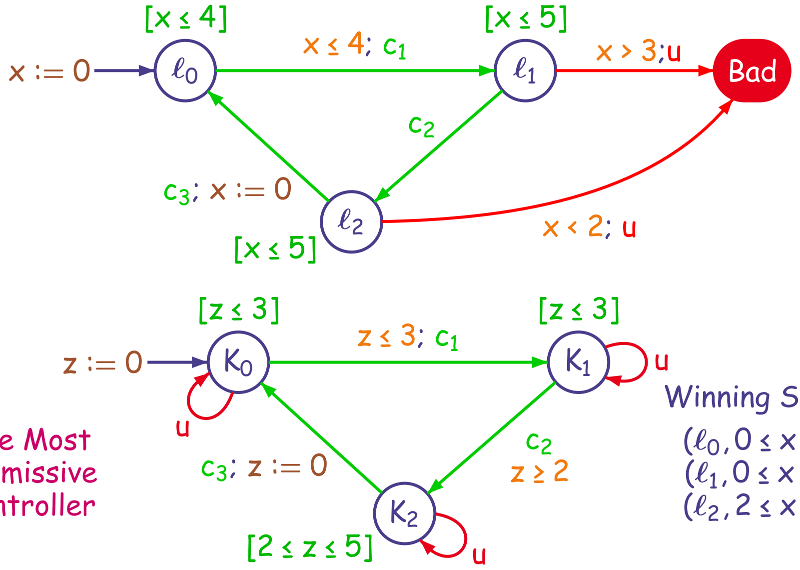
If $(\ell_0, 0) \in W^*$ we can compute the most permissive positional winning strategy.

Result of the Computation for the Example



Skip

Result of the Computation for the Example



The Most Permissive Controller

Winning States

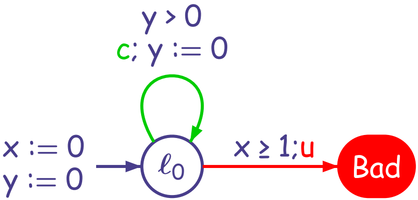
$(\ell_0, 0 \leq x \leq 3)$
$(\ell_1, 0 \leq x \leq 3)$
$(\ell_2, 2 \leq x \leq 5)$

Next:

- Control of Timed Systems: Basics
- Control of Discrete Event Systems
- Control of Timed Systems
- **Advanced Subjects**
  - Implementable Controllers
  - Optimal Controllers
  - Efficient Algorithms for Controller Synthesis
- Conclusion

Implementable Controllers

Problems with Dense-Time Control (1) [C. et al.'02]

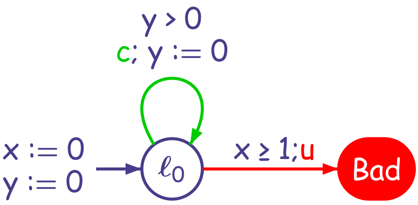


The System

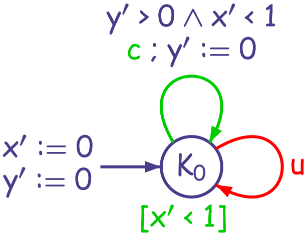
The Controller is Zeno !!!

Solution: add non-Zenoness in the control objective

Problems with Dense-Time Control (1) [C. et al.'02]



The System

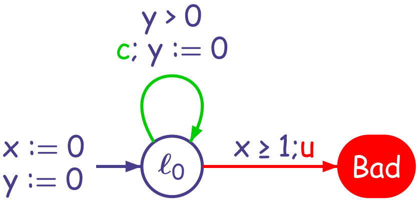


The Controller

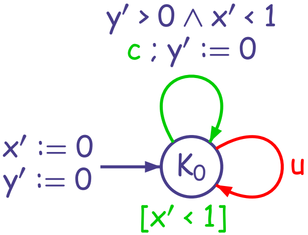
The Controller is Zeno !!!

Solution: add non-Zenoness in the control objective

Problems with Dense-Time Control (1) [C. et al.'02]



The System

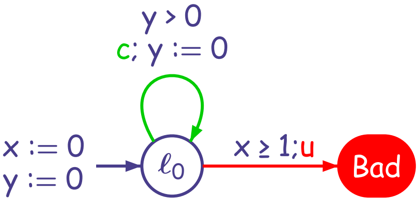


The Controller

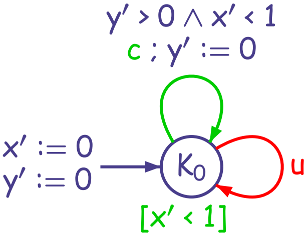
The Controller is Zeno !!!

Solution: add non-Zenoness in the control objective

Problems with Dense-Time Control (1) [C. et al.'02]



The System

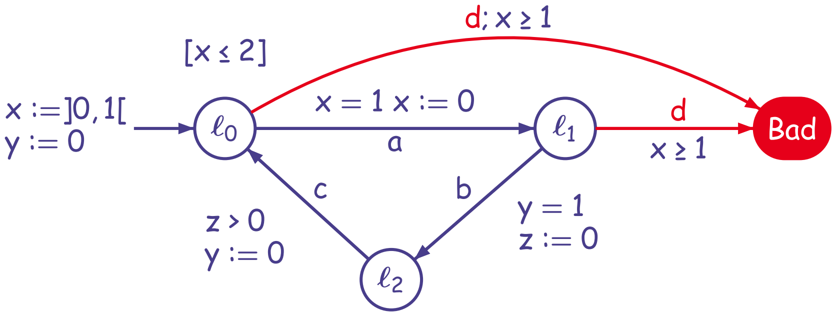


The Controller

The Controller is Zeno !!!

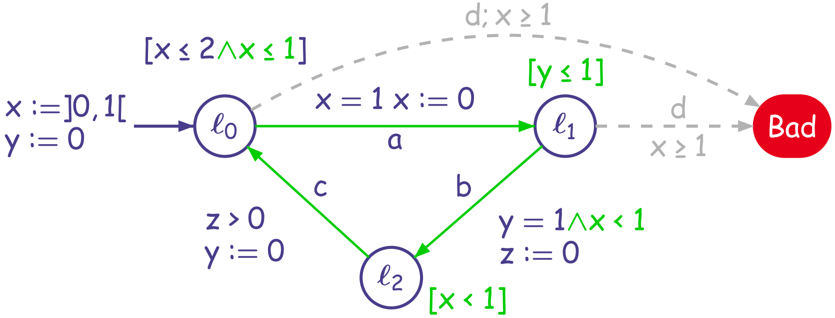
Solution: add non-Zenoness in the control objective

Problems with Dense-Time Control (2) [C. et al.'02]



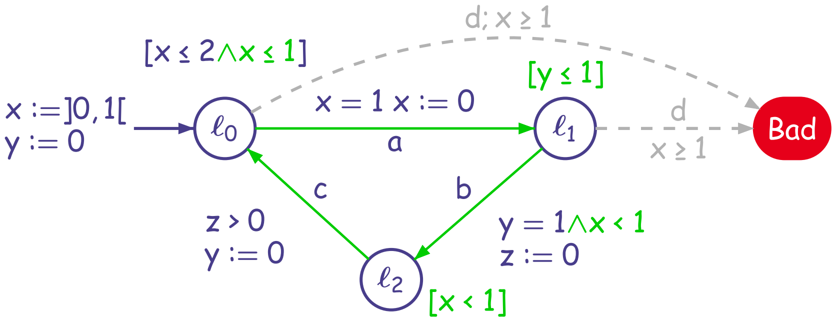
- The controller is Non-Zeno; One untimed behavior: $(\ell_0\ell_1\ell_2)^\omega$
- Let $\Delta_k > 0$ be the time spent in ℓ_2 in the k-th loop from ℓ_0 to ℓ_0
- It implies: $\forall k, \sum_{i=1}^{k} \Delta_i < 1 - x_0$, with $\forall i, \Delta_i > 0$

Problems with Dense-Time Control (2) [C. et al.'02]



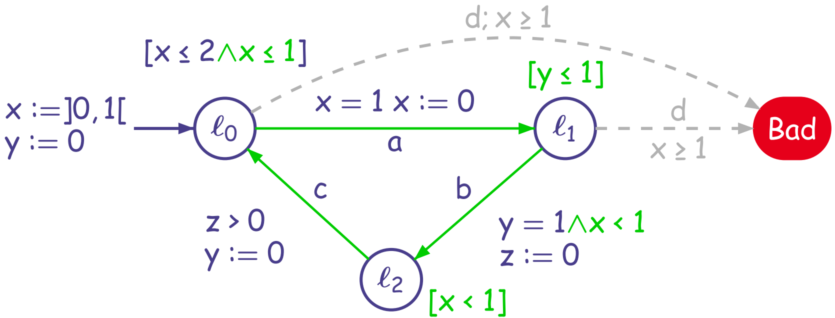
- The controller is Non-Zeno; One untimed behavior: $(\ell_0\ell_1\ell_2)^\omega$
- Let $\Delta_k > 0$ be the time spent in ℓ_2 in the k-th loop from ℓ_0 to ℓ_0
- It implies: $\forall k, \sum_{i=1}^{k} \Delta_i < 1 - x_0$, with $\forall i, \Delta_i > 0$

Problems with Dense-Time Control (2) [C. et al.'02]



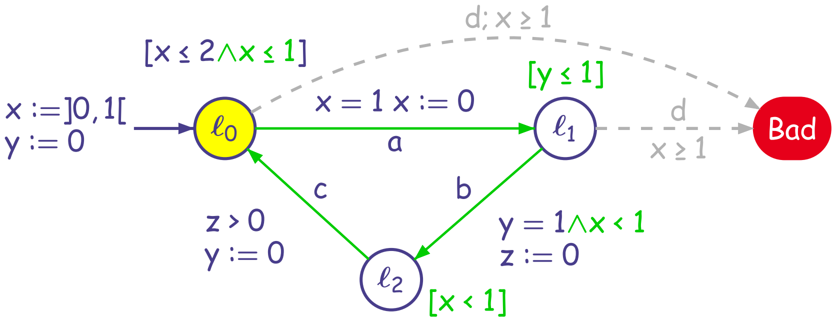
- The controller is Non-Zeno; One untimed behavior: $(\ell_0 \ell_1 \ell_2)^\omega$
- Let $\Delta_k > 0$ be the time spent in ℓ_2 in the k-th loop from ℓ_0 to ℓ_0
- It implies: $\forall k, \sum_{i=1}^{k} \Delta_i < 1 - x_0$, with $\forall i, \Delta_i > 0$

Problems with Dense-Time Control (2) [C. et al.'02]



- The controller is Non-Zeno; One untimed behavior: $(\ell_0\ell_1\ell_2)^\omega$
- Let $\Delta_k > 0$ be the time spent in ℓ_2 in the k-th loop from ℓ_0 to ℓ_0
- It implies: $\forall k, \sum_{i=1}^{k} \Delta_i < 1 - x_0$, with $\forall i, \Delta_i > 0$

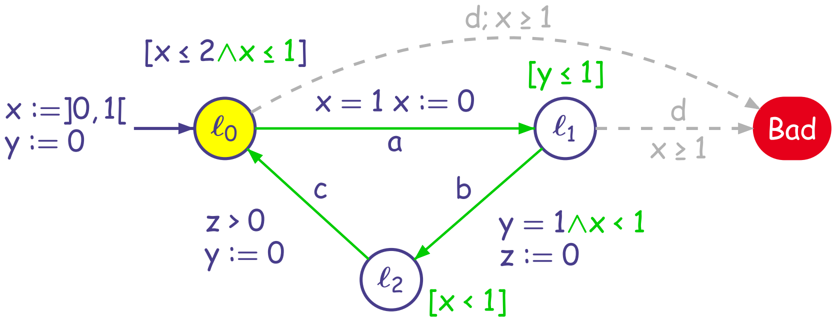
Problems with Dense-Time Control (2) [C. et al.'02]



- The controller is Non-Zeno; One untimed behavior: $(\ell_0\ell_1\ell_2)^\omega$
- Let $\Delta_k > 0$ be the time spent in ℓ_2 in the k-th loop from ℓ_0 to ℓ_0
- It implies: $\forall k, \sum_{i=1}^{k} \Delta_i < 1 - x_0$, with $\forall i, \Delta_i > 0$

	ℓ_0
x:	x_0
y:	0

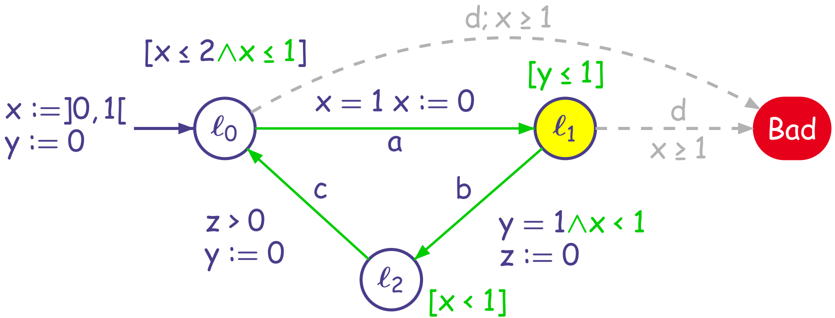
Problems with Dense-Time Control (2) [C. et al.'02]



- The controller is Non-Zeno; One untimed behavior: $(\ell_0\ell_1\ell_2)^\omega$
- Let $\Delta_k > 0$ be the time spent in ℓ_2 in the k-th loop from ℓ_0 to ℓ_0
- It implies: $\forall k, \sum_{i=1}^{k} \Delta_i < 1 - x_0$, with $\forall i, \Delta_i > 0$

	ℓ_0		
x:	x_0	$\rightsquigarrow$	1
y:	0		$1 - x_0$

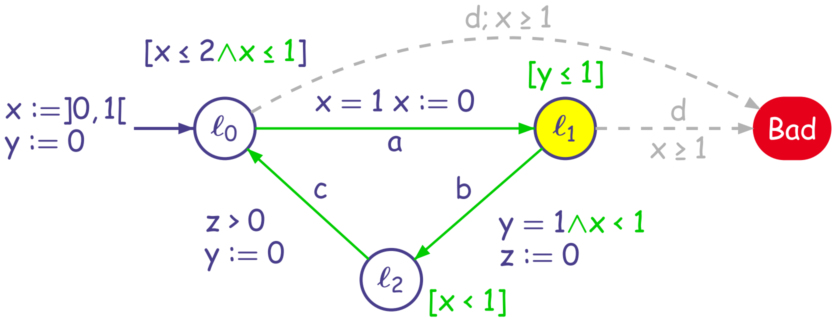
Problems with Dense-Time Control (2) [C. et al.'02]



- The controller is Non-Zeno; One untimed behavior: $(\ell_0\ell_1\ell_2)^\omega$
- Let $\Delta_k > 0$ be the time spent in ℓ_2 in the k-th loop from ℓ_0 to ℓ_0
- It implies: $\forall k, \sum_{i=1}^{k} \Delta_i < 1 - x_0$, with $\forall i, \Delta_i > 0$

$$
\begin{array}{lcccc}
 & \ell_0 & & & \ell_1 \\
x: & x_0 & \rightsquigarrow & 1 & \xrightarrow{a} 0 \\
y: & 0 & & 1 - x_0 & 1 - x_0
\end{array}
$$

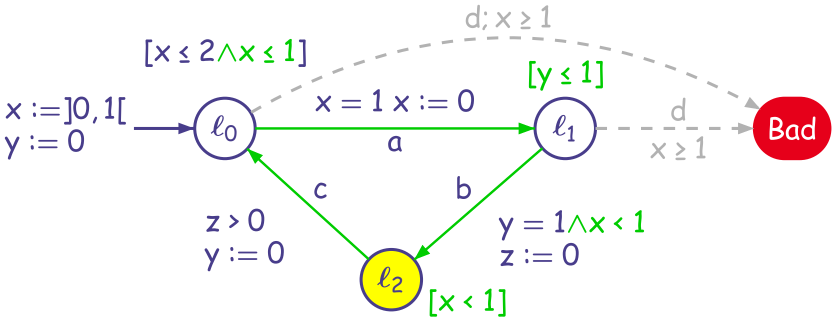
Problems with Dense-Time Control (2) [C. et al.'02]



- The controller is Non-Zeno; One untimed behavior: $(\ell_0\ell_1\ell_2)^\omega$
- Let $\Delta_k > 0$ be the time spent in ℓ_2 in the k-th loop from ℓ_0 to ℓ_0
- It implies: $\forall k, \sum_{i=1}^{k} \Delta_i < 1 - x_0$, with $\forall i, \Delta_i > 0$

$$\begin{array}{llllll} & \ell_0 & & & \ell_1 & \\ x: & x_0 & \rightsquigarrow & 1 & \xrightarrow{a} & 0 & \rightsquigarrow & x_0 \\ y: & 0 & & 1 - x_0 & & 1 - x_0 & & 1 \end{array}$$

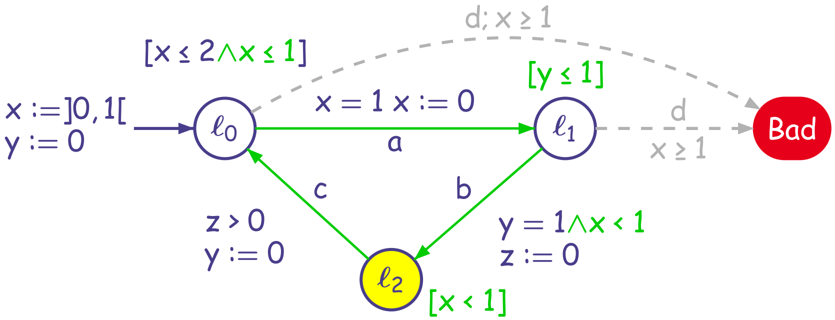
Problems with Dense-Time Control (2) [C. et al.'02]



- The controller is Non-Zeno; One untimed behavior: $(\ell_0\ell_1\ell_2)^\omega$
- Let $\Delta_k > 0$ be the time spent in ℓ_2 in the k-th loop from ℓ_0 to ℓ_0
- It implies: $\forall k, \sum_{i=1}^{k} \Delta_i < 1 - x_0$, with $\forall i, \Delta_i > 0$

$$
\begin{array}{lccccccc}
 & \ell_0 & & & & \ell_1 & & & & \ell_2 \\
x: & x_0 & \rightsquigarrow & 1 & \xrightarrow{a} & 0 & \rightsquigarrow & x_0 & \xrightarrow{b} & x_0 \\
y: & 0 & & 1 - x_0 & & 1 - x_0 & & 1 & & 1
\end{array}
$$

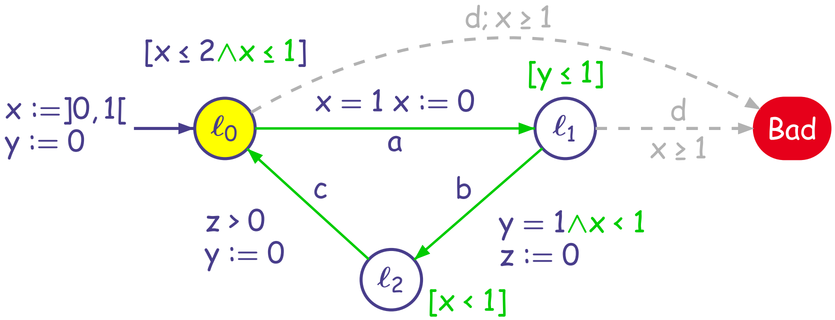
Problems with Dense-Time Control (2) [C. et al.'02]



- The controller is Non-Zeno; One untimed behavior: $(\ell_0\ell_1\ell_2)^\omega$
- Let $\Delta_k > 0$ be the time spent in ℓ_2 in the k-th loop from ℓ_0 to ℓ_0
- It implies: $\forall k, \sum_{i=1}^{k} \Delta_i < 1 - x_0$, with $\forall i, \Delta_i > 0$

$$
\begin{array}{lccccccccc}
 & \ell_0 & & & & \ell_1 & & & & \ell_2 & & \\
x: & x_0 & \rightsquigarrow & 1 & \xrightarrow{a} & 0 & \rightsquigarrow & x_0 & \xrightarrow{b} & x_0 & \overset{\Delta_1}{\rightsquigarrow} & x_0 + \Delta_1 \\
y: & 0 & & 1 - x_0 & & 1 - x_0 & & 1 & & 1 & & 1 + \Delta_1
\end{array}
$$

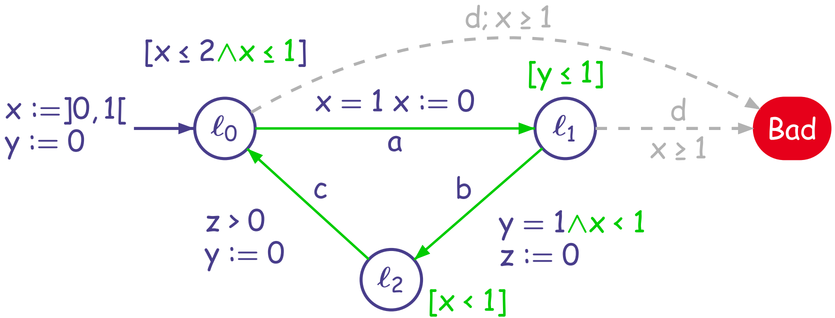
Problems with Dense-Time Control (2) [C. et al.'02]



- The controller is Non-Zeno; One untimed behavior: $(\ell_0\ell_1\ell_2)^\omega$
- Let $\Delta_k > 0$ be the time spent in ℓ_2 in the k-th loop from ℓ_0 to ℓ_0
- It implies: $\forall k, \sum_{i=1}^{k} \Delta_i < 1 - x_0$, with $\forall i, \Delta_i > 0$

$$\begin{array}{lccccccccccc} & \ell_0 & & & & \ell_1 & & & & \ell_2 & & & & \ell_0 \\ x: & x_0 & \rightsquigarrow & 1 & \xrightarrow{a} & 0 & \rightsquigarrow & x_0 & \xrightarrow{b} & x_0 & \overset{\Delta_1}{\rightsquigarrow} & x_0 + \Delta_1 & \xrightarrow{c} & x_0 + \Delta_1 \\ y: & 0 & & 1 - x_0 & & 1 - x_0 & & 1 & & 1 & & 1 + \Delta_1 & & 0 \end{array}$$

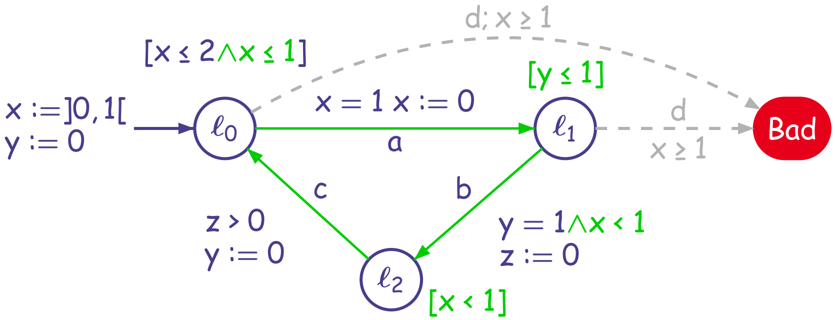
Problems with Dense-Time Control (2) [C. et al.'02]



- The controller is Non-Zeno; One untimed behavior: $(\ell_0\ell_1\ell_2)^\omega$
- Let $\Delta_k > 0$ be the time spent in ℓ_2 in the k-th loop from ℓ_0 to ℓ_0
- It implies: $\forall k, \sum_{i=1}^{k} \Delta_i < 1 - x_0$, with $\forall i, \Delta_i > 0$

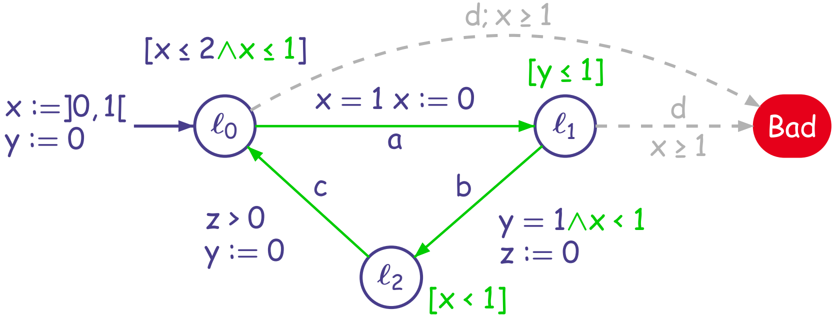
$$\begin{array}{lcccccccccc}
 & \ell_0 & & & \ell_1 & & & & \ell_2 & & & \ell_0 \\
x: & x_0 & \rightsquigarrow & 1 & \xrightarrow{a} 0 & \rightsquigarrow & x_0 & \xrightarrow{b} & x_0 & \overset{\Delta_1}{\rightsquigarrow} & x_0+\Delta_1 & \xrightarrow{c} & x_0+\Delta_1 \\
y: & 0 & & 1-x_0 & 1-x_0 & & 1 & & 1 & & 1+\Delta_1 & & 0
\end{array}$$

Problems with Dense-Time Control (2) [C. et al.'02]



- The controller is Non-Zeno; One untimed behavior: $(\ell_0\ell_1\ell_2)^\omega$
- Let $\Delta_k > 0$ be the time spent in ℓ_2 in the k-th loop from ℓ_0 to ℓ_0
- It implies: $\forall k, \sum_{i=1}^{k} \Delta_i < 1 - x_0$, with $\forall i, \Delta_i > 0$
- Must hold for ever: $\sum_{k=1}^{k=+\infty} \Delta_k < 1 - x_0$ with $\forall k, \Delta_k > 0$

Problems with Dense-Time Control (2) [C. et al.'02]



- The controller is Non-Zeno; One untimed behavior: $(\ell_0 \ell_1 \ell_2)^\omega$
- Let $\Delta_k > 0$ be the time spent in ℓ_2 in the k-th loop from ℓ_0 to ℓ_0
- It implies: $\forall k, \sum_{i=1}^{k} \Delta_i < 1 - x_0$, with $\forall i, \Delta_i > 0$
- Must hold for ever: $\sum_{k=1}^{k=+\infty} \Delta_k < 1 - x_0$ with $\forall k, \Delta_k > 0$

The Controller is Non-Zeno but not Implementable !!!

Sampling Control

- Let $a \in \mathbb{Q}^*$ be a sampling rate
- An a-controller is a controller that can do actions only at $k \cdot a, k \geq 1$ and $k \in \mathbb{N}$

Sampling Control

- Let $a \in \mathbb{Q}^*$ be a sampling rate
- An a-controller is a controller that can do actions only at $k \cdot a, k \geq 1$ and $k \in \mathbb{N}$

Known Sampling Rate Control Problem (KSR)

Input: $a \in \mathbb{Q}^*$, Bad (states), G a TGA
Problem: Is there a a-controller for G that avoids Bad ?

Sampling Control

- Let $a \in \mathbb{Q}^*$ be a sampling rate
- An a-controller is a controller that can do actions only at $k \cdot a, k \geq 1$ and $k \in \mathbb{N}$

Known Sampling Rate Control Problem (KSR)

Input: $a \in \mathbb{Q}^*$, Bad (states), G a TGA
Problem: Is there a a-controller for G that avoids Bad ?

Theorem ([Henzinger & Kopke'99])

The Known Sampling Rate Control Problem is decidable.

Sampling Control

- Let $a \in \mathbb{Q}^*$ be a sampling rate
- An a-controller is a controller that can do actions only at $k \cdot a, k \geq 1$ and $k \in \mathbb{N}$

Unknown Sampling Rate Control Problem (USR)

Input: Bad (states), G a TGA
Problem: Is there a sampling rate $a \in \mathbb{Q}^*$ such that there is a a-controller for G that avoids Bad ?

Sampling Control

- Let $\alpha \in \mathbb{Q}^*$ be a sampling rate
- An α-controller is a controller that can do actions only at $k \cdot \alpha, k \geq 1$ and $k \in \mathbb{N}$

Unknown Sampling Rate Control Problem (USR)

Input: Bad (states), G a TGA
Problem: Is there a sampling rate $\alpha \in \mathbb{Q}^*$ such that there is a α-controller for G that avoids Bad ?

Theorem ([C. et al.'02])

The Unknown Sampling Rate Control Problem is undecidable.

Summary of the Results

Decidability results for the safety control problem on LHA:

	Known Switch Cond.	Unknown Switch Cond.
Timed Auto.	√ [Maler et al.'95]	√ [Maler et al.'95]
Init. Rect. Auto	√ [Henzinger et al.'99]	× [Henzinger et al.'95]
Rect. Auto.	× [Henzinger et al.'99]	× [Henzinger et al.'99]

	Known Sampling Rate	Unknown SR
Timed Auto.	√ [Hoffmann & Wong-Toi'92]	× [C. et al.'02]
Init. Rect. Auto.	√ [Henzinger & Kopke'97]	× [C. et al.'02]
Rect. Auto.	√ [Henzinger & Kopke'97]	× [C. et al.'02]

√: Decidable ×: Undecidable

Recent result [Bouyer et al.'06]

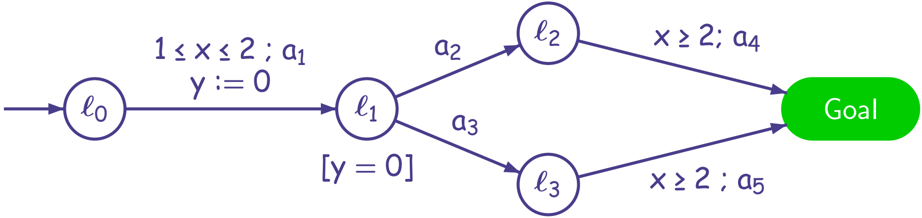
The reachability USR-CP is decidable for o-minimal automata.

Results on implementation of Timed Automata

[De Wulf et al.'04b, De Wulf et al.'04a, De Wulf et al.'05b]

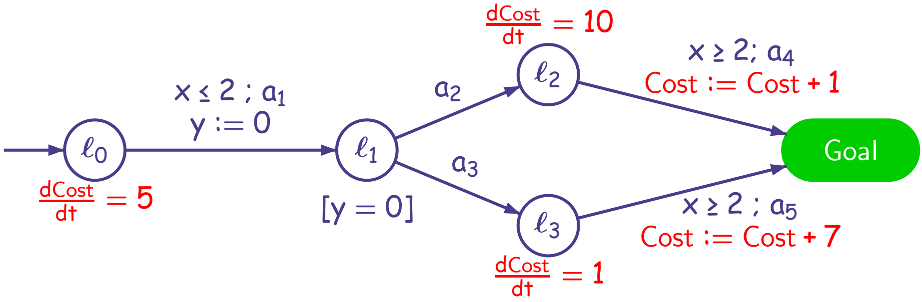
Optimal Controllers

Optimal Reachability for Timed Automata



- Reachability for Timed Automata [Alur & Dill'94]

Optimal Reachability for Timed Automata

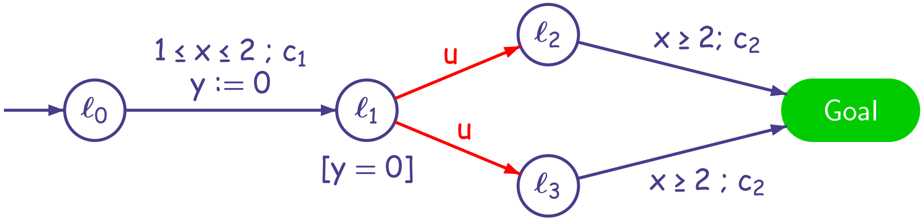


- Reachability for Timed Automata [Alur & Dill'94]
- Optimal Reachability for Priced (or Weighted) Timed Automata [Larsen et al.'01, Alur et al.'01]

$$(\ell_0,0,0) \xrightarrow{1} (\ell_0,1,1) \xrightarrow{a_1\ a_2} (\ell_2,1,0) \xrightarrow{3} (\ell_2,4,3) \xrightarrow{a_4} (Goal,4,3)$$

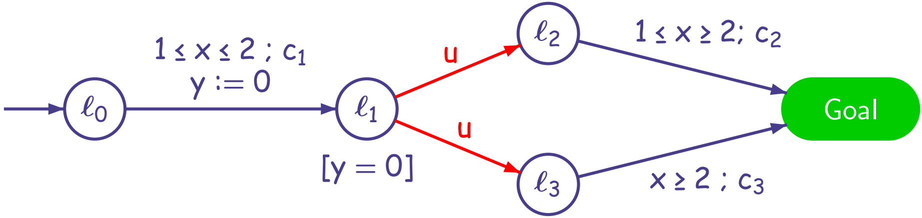
$$Cost = 1 \cdot 5 + 3 \cdot 10 + 1 = 36$$

Optimal Reachability for Timed Automata



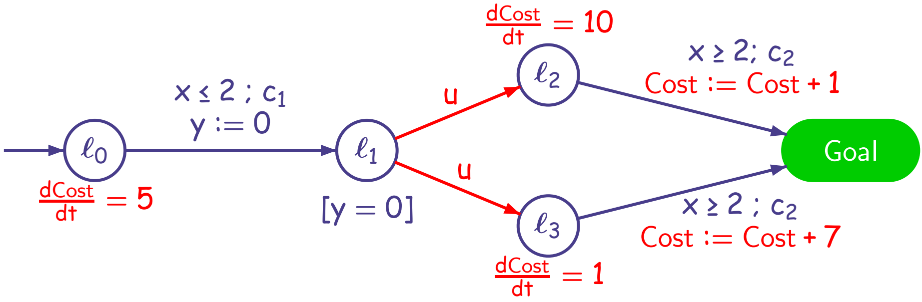
- Reachability for Timed Automata [Alur & Dill'94]
- Optimal Reachability for Priced (or Weighted) Timed Automata [Larsen et al.'01, Alur et al.'01]
- Control for Timed Game Automata [Maler et al.'95]

Optimal Reachability for Timed Automata



- Reachability for Timed Automata [Alur & Dill'94]
- Optimal Reachability for Priced (or Weighted) Timed Automata [Larsen et al.'01, Alur et al.'01]
- Control for Timed Game Automata [Maler et al.'95]
- Time Optimal Control (Reachability) [Asarin & Maler'99]

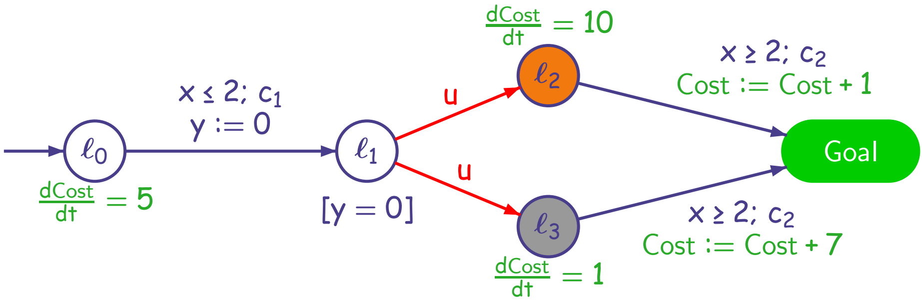
Optimal Reachability for Timed Automata



- Reachability for Timed Automata [Alur & Dill'94]
- Optimal Reachability for Priced (or Weighted) Timed Automata [Larsen et al.'01, Alur et al.'01]
- Control for Timed Game Automata [Maler et al.'95]
- Time Optimal Control (Reachability) [Asarin & Maler'99]

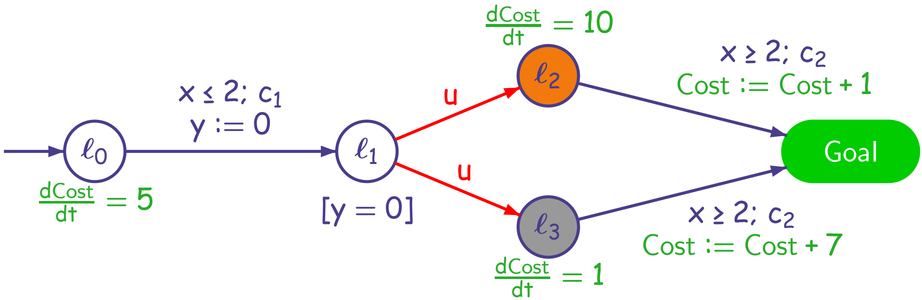
Optimal Control for Priced Timed Game Automata ?

A Small Example



- What is the best cost whatever the environment does ?

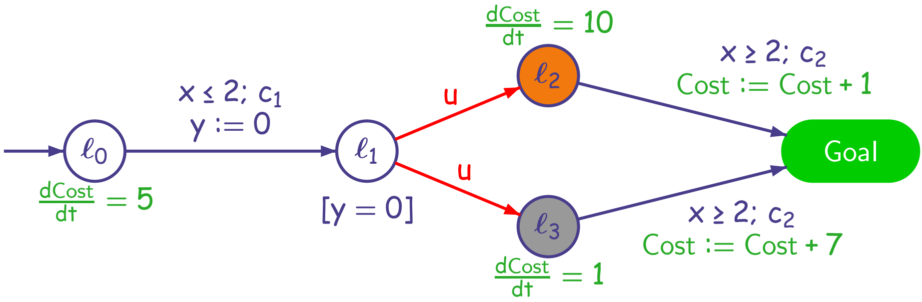
A Small Example



- What is the best cost whatever the environment does ?

$$\inf_{0 \leq t \leq 2} \max\{5t + 10(2-t) + 1, 5t + (2-t) + 7\} = 14 + \frac{1}{3}$$

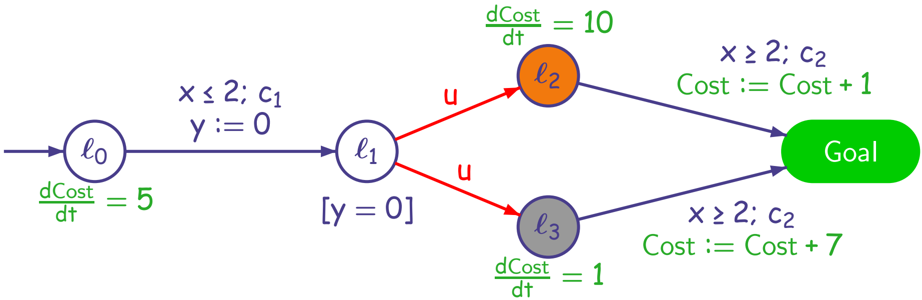
A Small Example



- What is the best cost whatever the environment does ?

$$\inf_{0 \leq t \leq 2} \max\{5t + 10(2-t) + 1, 5t + (2-t) + 7\} = 14 + \frac{1}{3}$$

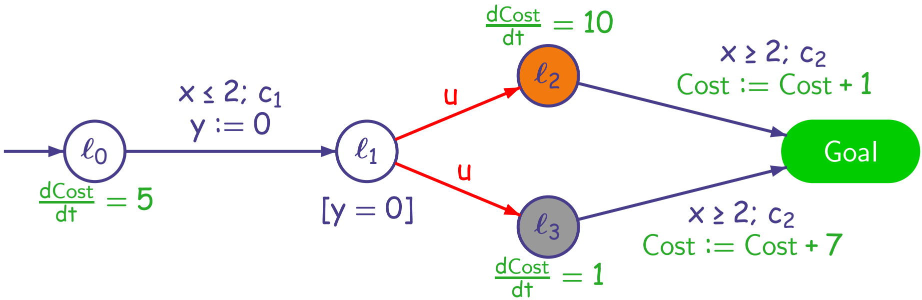
A Small Example



- What is the best cost whatever the environment does ?

$$\inf_{0 \leq t \leq 2} \max\{5t + 10(2-t) + 1, 5t + (2-t) + 7\} = 14 + \frac{1}{3}$$

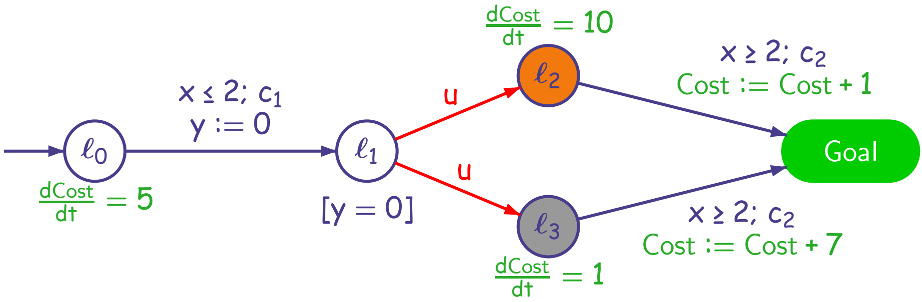
A Small Example



- What is the best cost whatever the environment does ?

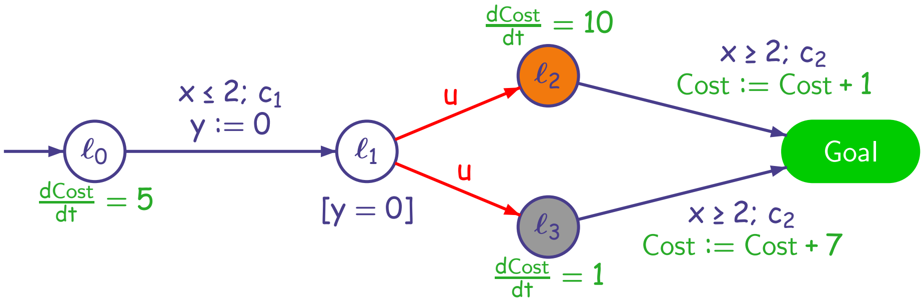
$$\inf_{0 \leq t \leq 2} \max\{5t + 10(2-t) + 1, 5t + (2-t) + 7\} = 14 + \frac{1}{3}$$

A Small Example



- What is the best cost whatever the environment does ?
- Is there a strategy to achieve this optimal cost ?

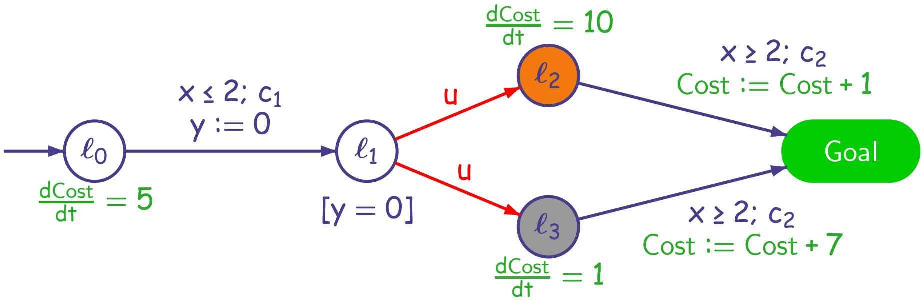
A Small Example



- What is the best cost whatever the environment does ?
- Is there a strategy to achieve this optimal cost ?

  Yes: wait in ℓ_0 until $t = \frac{4}{3}$ and then fire c_1

A Small Example

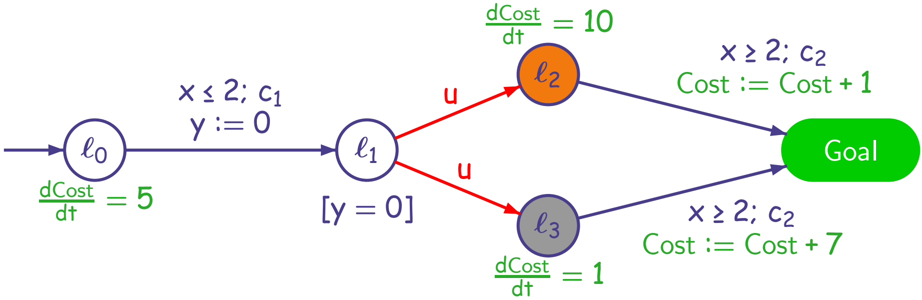


- What is the best cost whatever the environment does ?
- Is there a strategy to achieve this optimal cost ?

  Yes: wait in ℓ_0 until $t = \frac{4}{3}$ and then fire c_1
- Can we compute such a strategy ?

  Yes: but need memory sometimes

Optimal Control Problems

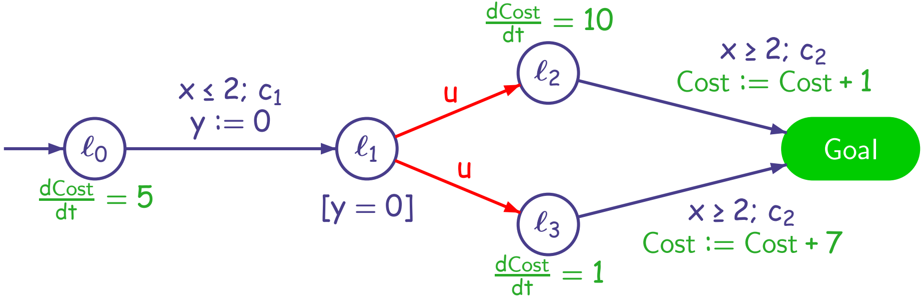


Can we find algorithms for these problems on PTGA ?

1. Compute the optimal cost
2. Decide if there is an optimal strategy
3. Compute an optimal strategy (if one exists)

From Optimal Control to Control

A Reachability TGA $\mathcal{A}$

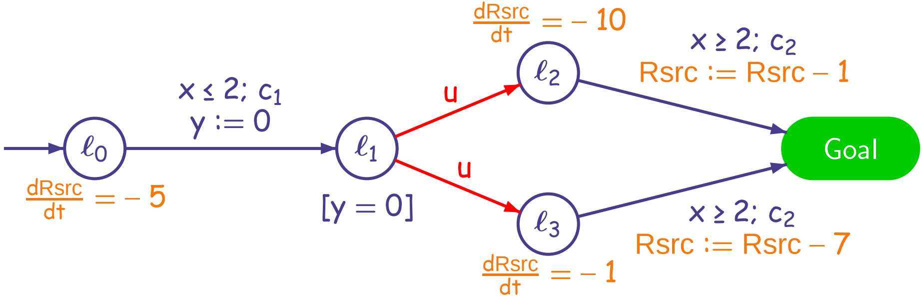


- Transform $\mathcal{A}$ in Linear Hybrid Game Automaton $H(\mathcal{A})$
- Define the reachability game for $H(\mathcal{A})$ with goal: $\text{Goal} \wedge \text{Rsrc} \geq 0$

Optimal Control for $\mathcal{A}$ $\iff$ Reachability Control for $H(\mathcal{A})$

From Optimal Control to Control

A Linear Hybrid Game $H(\mathcal{A})$

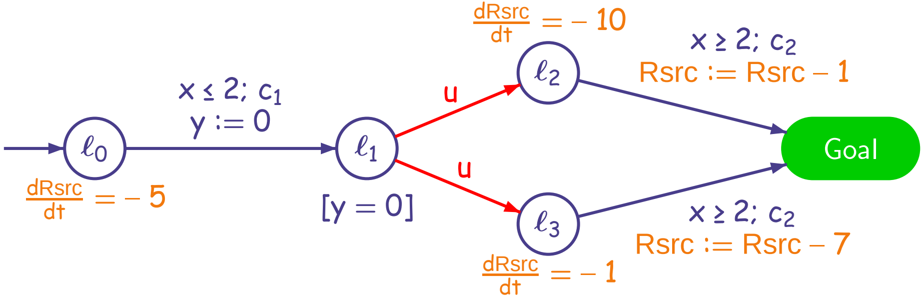


- Transform $\mathcal{A}$ in Linear Hybrid Game Automaton $H(\mathcal{A})$
- Define the reachability game for $H(\mathcal{A})$ with goal: $\text{Goal} \wedge Rsrc \geq 0$

Optimal Control for $\mathcal{A}$ $\iff$ Reachability Control for $H(\mathcal{A})$

From Optimal Control to Control

A Linear Hybrid Game $H(\mathcal{A})$

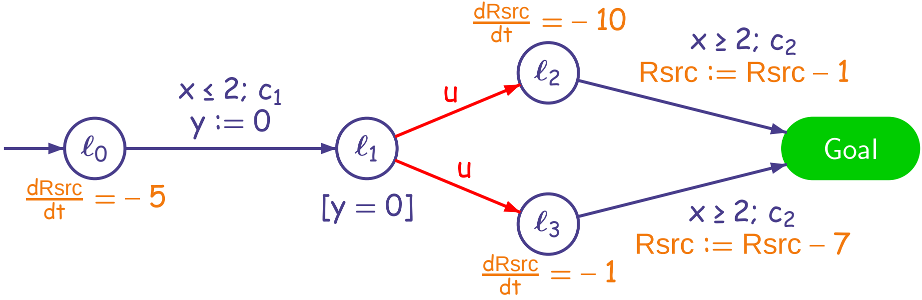


- Transform $\mathcal{A}$ in Linear Hybrid Game Automaton $H(\mathcal{A})$
- Define the reachability game for $H(\mathcal{A})$ with goal: $Goal \wedge Rsrc \geq 0$

Optimal Control for $\mathcal{A}$ $\Longleftrightarrow$ Reachability Control for $H(\mathcal{A})$

From Optimal Control to Control

A Linear Hybrid Game $H(\mathcal{A})$



- Transform $\mathcal{A}$ in Linear Hybrid Game Automaton $H(\mathcal{A})$
- Define the reachability game for $H(\mathcal{A})$ with goal: $Goal \wedge Rsrc \geq 0$

Optimal Control for $\mathcal{A}$ $\iff$ Reachability Control for $H(\mathcal{A})$

Results [Bouyer et al.'04a, Bouyer et al.'04b]

Theorem (Reachability Control for LHA)

There is a semi-algorithm CompWin that computes the set of winning states for LHA.
Uses polyhedra instead of zones.

Results

[Bouyer et al.'04a, Bouyer et al.'04b]

Let $\mathcal{A}$ be a Reachability Priced Timed Game Automaton such that:

- $\mathcal{A}$ is cost non-zeno *i.e.* $\exists\kappa$ s.t. every cycle in the region automaton of $\mathcal{A}$ has cost at least κ
- $\mathcal{A}$ is bounded *i.e.* all clocks in $\mathcal{A}$ are bounded

Results [Bouyer et al.'04a, Bouyer et al.'04b]

Let A be a Reachability Priced Timed Game Automaton such that:

- A is cost non-zeno *i.e.* $\exists\kappa$ s.t. every cycle in the region automaton of A has cost at least κ
- A is bounded *i.e.* all clocks in A are bounded

Theorem (Non-Zeno Cost [Bouyer et al.'04a])

The algorithm CompWin terminates for $H(A)$.

Results

[Bouyer et al.'04a, Bouyer et al.'04b]

Let A be a Reachability Priced Timed Game Automaton such that:

- A is cost non-zeno *i.e.* $\exists \kappa$ s.t. every cycle in the region automaton of A has cost at least κ
- A is bounded *i.e.* all clocks in A are bounded

Theorem (Non-Zeno Cost [Bouyer et al.'04a])

The algorithm CompWin terminates for $H(A)$.

Theorem (Optimal Cost Computation [Bouyer et al.'04a])

1. Optimal Cost is computable.
2. Optimal Strategy Existence Problem is decidable.

Results [Bouyer et al.'04a, Bouyer et al.'04b]

Let A be a Reachability Priced Timed Game Automaton such that:

- A is cost non-zeno *i.e.* $\exists\kappa$ s.t. every cycle in the region automaton of A has cost at least κ
- A is bounded *i.e.* all clocks in A are bounded

Theorem (Non-Zeno Cost [Bouyer et al.'04a])

The algorithm CompWin terminates for $H(A)$.

Theorem (Optimal Cost Computation [Bouyer et al.'04a])

1. Optimal Cost is computable.
2. Optimal Strategy Existence Problem is decidable.

Theorem ([Brihaye et al.'05])

Non-Zeno Cost is a necessary assumption.

Summary of the Results

What's decidable about optimal reachability control?

- Non-Zeno Cost [Bouyer et al.'04a]
- O-minimal automata [Bouyer et al.'07]
- 1-clock PTGA (3EXPTIME) [Bouyer et al.'06a]

What's UNdecidable about optimal control?

- 5-clock Zeno PTGA [Brihaye et al.'05]
- 3-clock Zeno PTGA [Bouyer et al.'06b]

What's decidable for infinite schedules (safety) ?

- Mean Cost decidable for 1-player PTA [Bouyer et al.'04c]

What's open?

Optimal Mean Cost for PTGA

Summary of the Results

What's decidable about optimal reachability control?

- Non-Zeno Cost [Bouyer et al.'04a]
- O-minimal automata [Bouyer et al.'07]
- 1-clock PTGA (3EXPTIME) [Bouyer et al.'06a]

What's UNdecidable about optimal control?

- 5-clock Zeno PTGA [Brihaye et al.'05]
- 3-clock Zeno PTGA [Bouyer et al.'06b]

What's decidable for infinite schedules (safety) ?

- Mean Cost decidable for 1-player PTA [Bouyer et al.'04c]

What's open?

Optimal Mean Cost for PTGA

Summary of the Results

What's decidable about optimal reachability control?

- Non-Zeno Cost [Bouyer et al.'04a]
- O-minimal automata [Bouyer et al.'07]
- 1-clock PTGA (3EXPTIME) [Bouyer et al.'06a]

What's UNdecidable about optimal control?

- 5-clock Zeno PTGA [Brihaye et al.'05]
- 3-clock Zeno PTGA [Bouyer et al.'06b]

What's decidable for infinite schedules (safety) ?

- Mean Cost decidable for 1-player PTA [Bouyer et al.'04c]

What's open?

Optimal Mean Cost for PTGA

Summary of the Results

What's decidable about optimal reachability control?

- Non-Zeno Cost [Bouyer et al.'04a]
- O-minimal automata [Bouyer et al.'07]
- 1-clock PTGA (3EXPTIME) [Bouyer et al.'06a]

What's UNdecidable about optimal control?

- 5-clock Zeno PTGA [Brihaye et al.'05]
- 3-clock Zeno PTGA [Bouyer et al.'06b]

What's decidable for infinite schedules (safety) ?

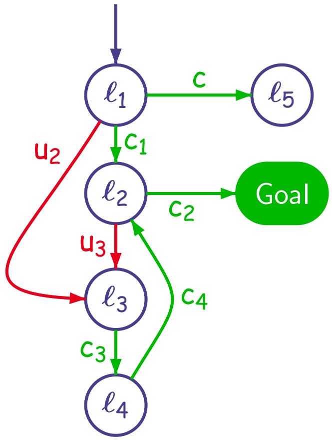
- Mean Cost decidable for 1-player PTA [Bouyer et al.'04c]

What's open?

Optimal Mean Cost for PTGA

Efficient Controller Synthesis

Reachability Control for Finite Games

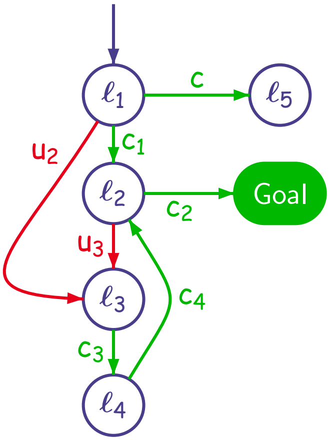


$\longrightarrow$ Uncontrollable
$\longrightarrow$ Controllable

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

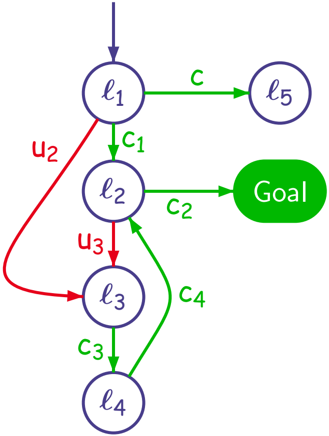


$\longrightarrow$ Uncontrollable
$\longrightarrow$ Controllable

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



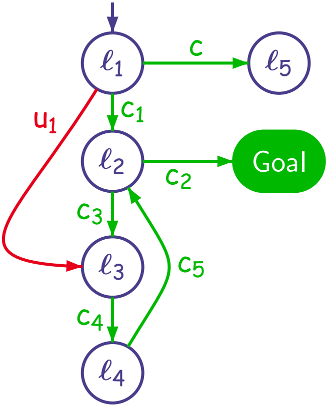
$\longrightarrow$ Uncontrollable
$\longrightarrow$ Controllable

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

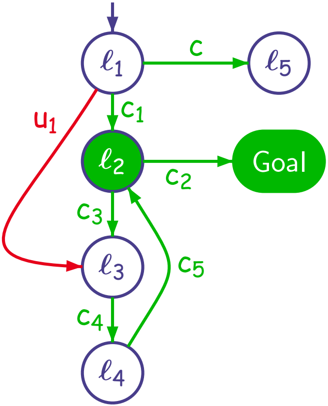
Controllable Predecessors:

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

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

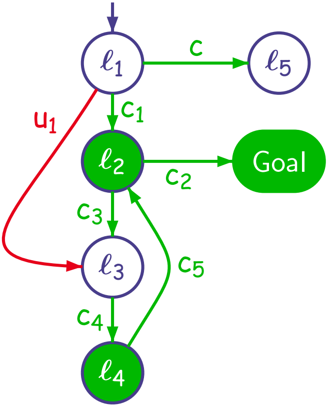
Controllable Predecessors:

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

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

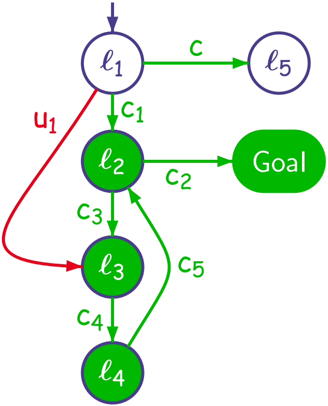
Controllable Predecessors:

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

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

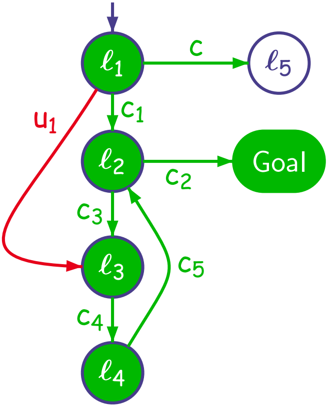
Controllable Predecessors:

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

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

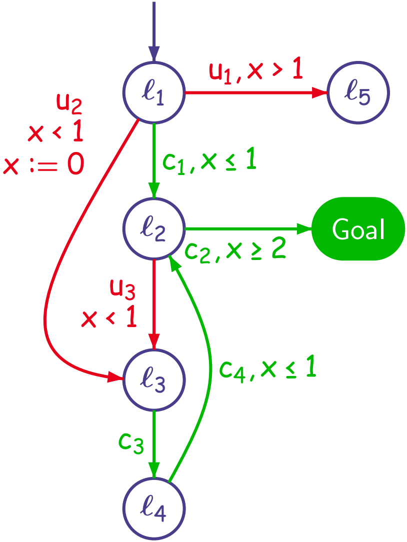
Controllable Predecessors:

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

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. $X_4 = \{Goal, \ell_2, \ell_4, \ell_3, \ell_1\}$

Reachability Control for Timed Games



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

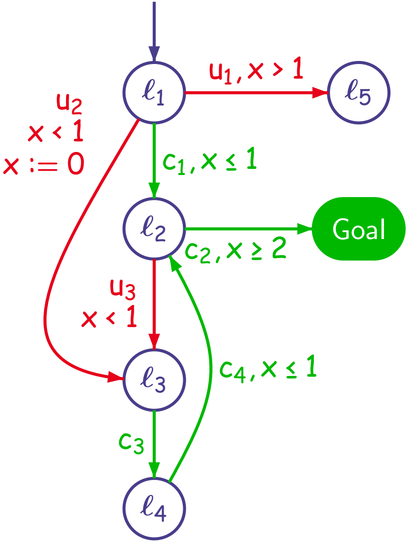
q $\qquad$ $q' \in X$

Controllable Predecessors:

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

▸ Timed Auto

Reachability Control for Timed Games



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

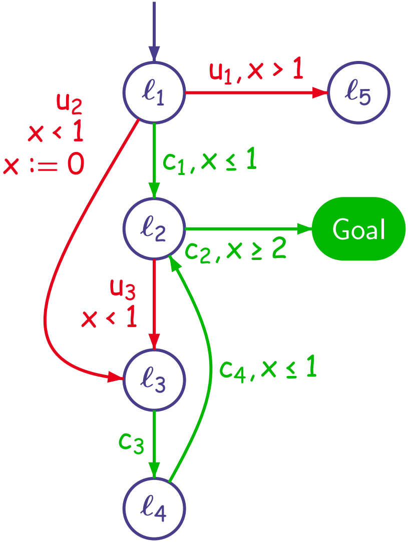
$$q \xrightarrow{t} q' \in X$$

Controllable Predecessors:

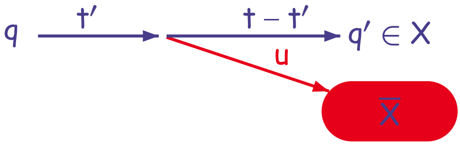
$$\pi(X) = \mathrm{Pred}_t\left(X \cup \mathrm{cPred}(X), \mathrm{uPred}(\overline{X})\right)$$

Timed Auto

Reachability Control for Timed Games



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

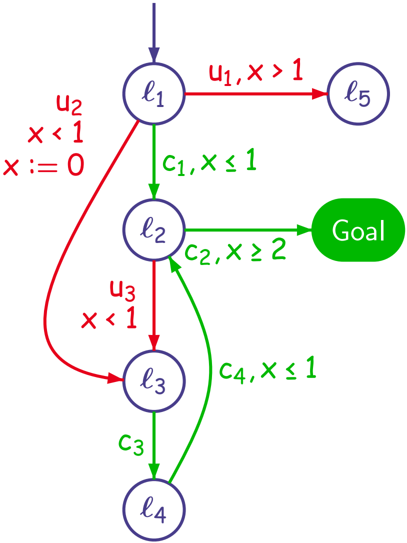


Controllable Predecessors:

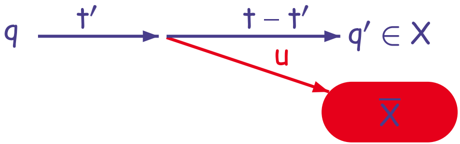
$$\pi(X) = \mathrm{Pred}_t\left(X \cup \mathrm{cPred}(X), \mathrm{uPred}(\overline{X})\right)$$

▸ Timed Auto

Reachability Control for Timed Games



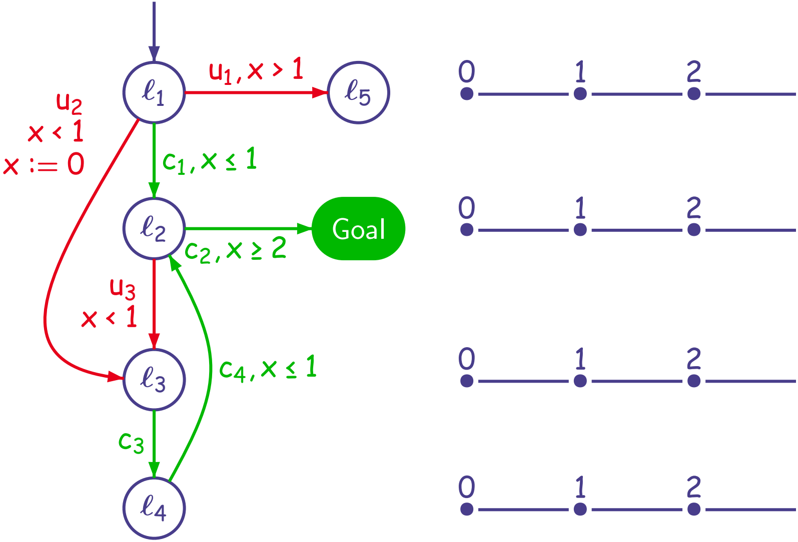
Safe Time Elapsing:
When is it safe to let time elapse from q to q′ ?



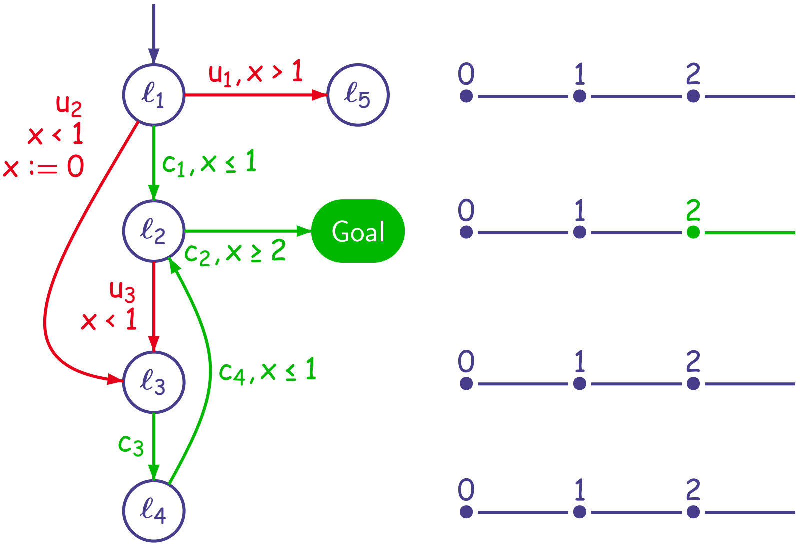
Controllable Predecessors:

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

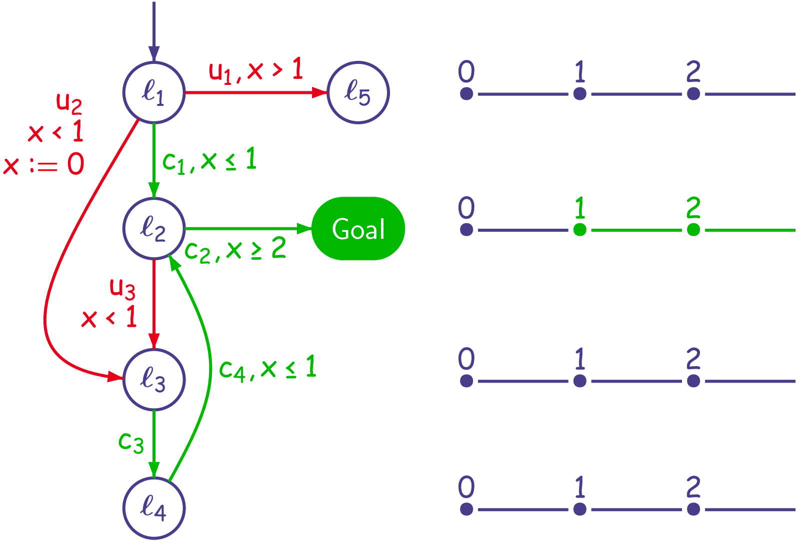
Reachability Control for Timed Games



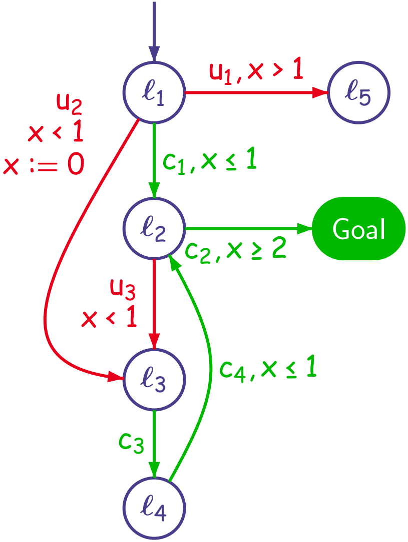
Reachability Control for Timed Games

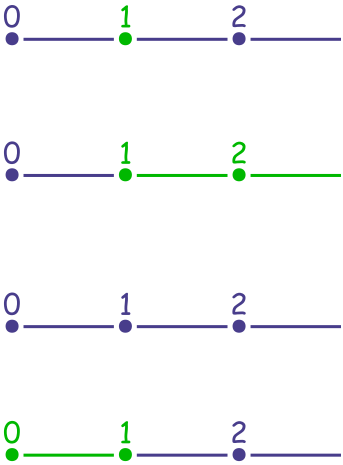


Reachability Control for Timed Games

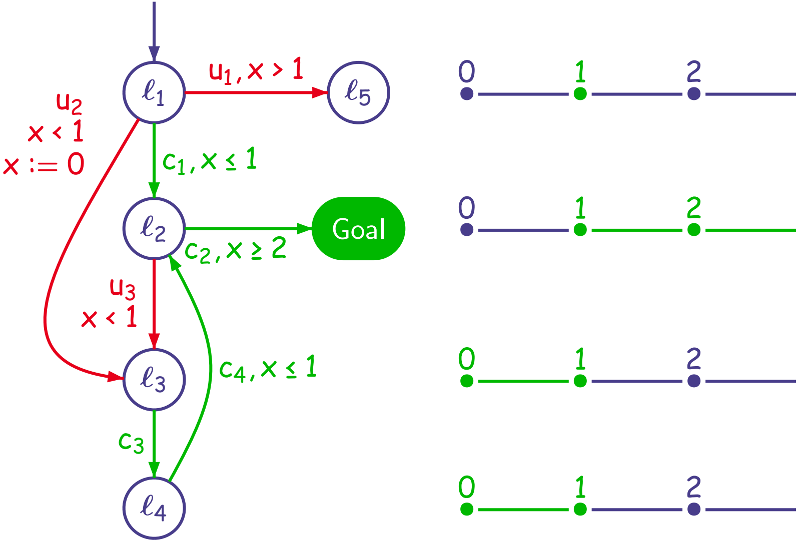


Reachability Control for Timed Games

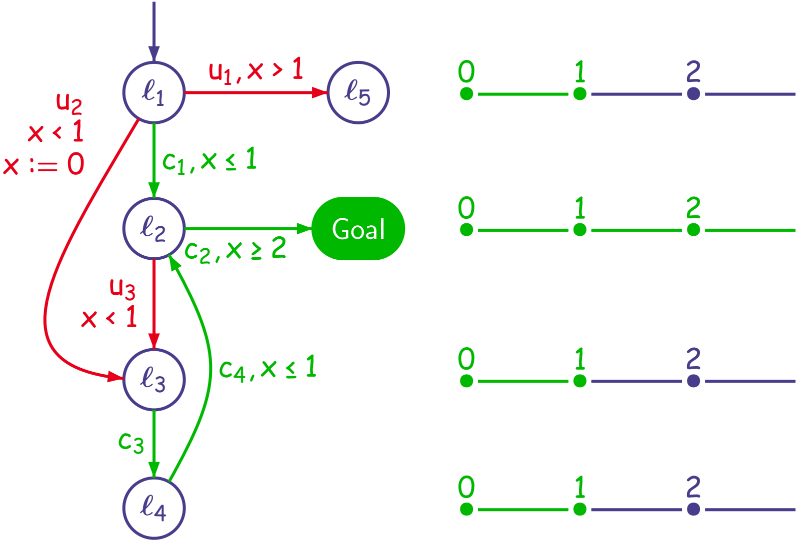




Reachability Control for Timed Games



Reachability Control for Timed Games



Summary of the Results for Reachability Control

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.'04a]
- Half on-the-fly algorithm [Altisen & Tripakis'99, Altisen & Tripakis'02]

New Results: 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 the tool UPPAAL-TiGA [UPPAAL-TiGA'07]

Summary of the Results for Reachability Control

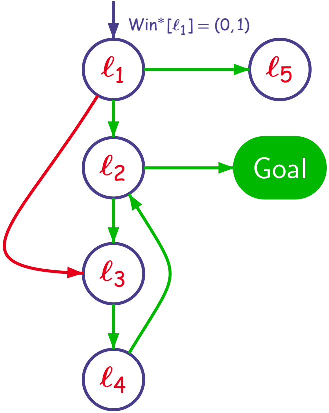
Known Results for Timed (Game) Automata:

- 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.'04a]
- Half on-the-fly algorithm [Altisen & Tripakis'99, Altisen & Tripakis'02]

New Results: 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 the tool **UPPAAL-TiGA** [UPPAAL-TiGA'07]

Liu & Smolka Algorithm [Liu & Smolka'98]



→ Current
→ Explored Forwards
Dashed = in Waiting List

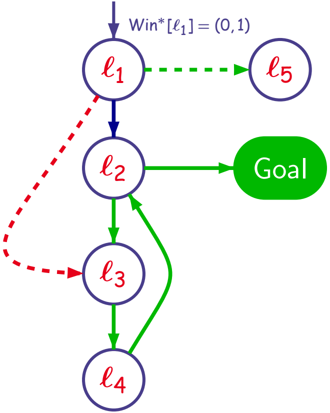
Initialization:

```
    Passed ← {q0};
    Waiting ← {(q0, a, q') | a ∈ Act q --a--> q'};
    Win[q0] ← (q0 ∈ Goal ? 1 : 0);
    Depend[q0] ← ∅;
```

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]



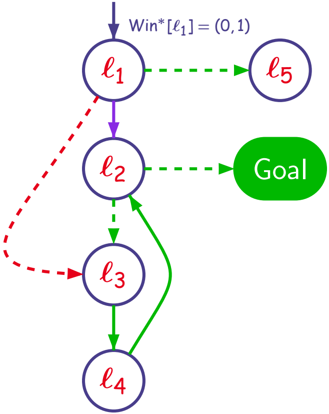
$\longrightarrow$ Current
$\longrightarrow$ Explored Forwards
Dashed = in Waiting List

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

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 $q' \not\in$ 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

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

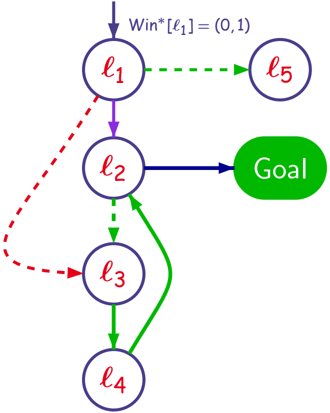
Initialization:

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

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

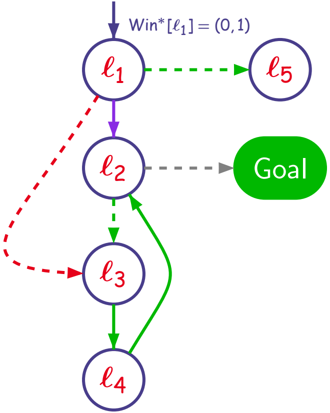
Initialization:

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 $q' \not\in$ 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

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

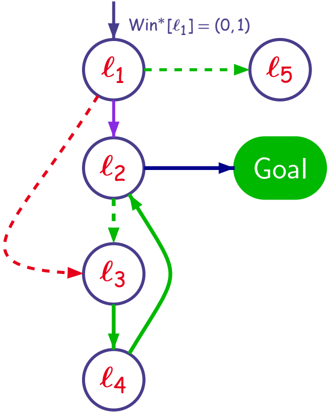
Initialization:

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

Main:

while $((Waiting \neq \emptyset) \wedge Win[q_0] \neq 1))$ **do**
$e = (q, a, q') \leftarrow pop(Waiting)$;
if $q' \not\in Passed$ **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

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

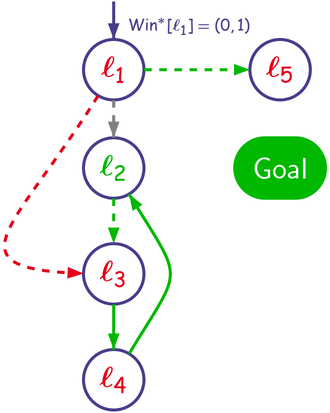
Initialization:

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 ≠ ∅) ∧ Win[q0] ≠ 1)) do
    e = (q, α, q') ← pop(Waiting);
    if q' ∉ Passed then {
        Passed ← Passed ∪ {q'};
        Depend[q'] ← {(q, α, q')};
        Win[q'] ← (q' ∈ Goal ? 1 : 0);
        Waiting ← Waiting ∪ {(q', α, q'') | q' -α-> 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]



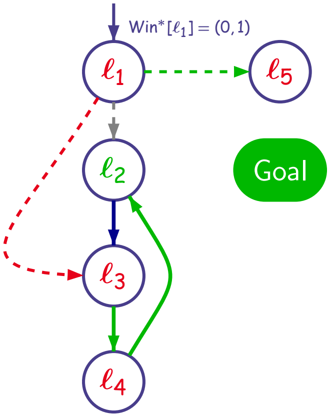
→ Current
→ Explored Forwards
Dashed = in Waiting List

$$e = (\ell_2, c_2, \mathsf{Goal})$$

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

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

Liu & Smolka Algorithm [Liu & Smolka'98]



$\longrightarrow$ Current

$\longrightarrow$ Explored Forwards

Dashed = in Waiting List

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

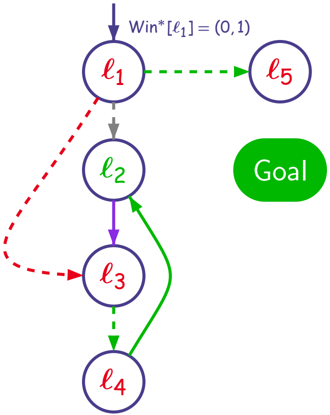
Initialization:

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 $q' \not\in$ 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

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

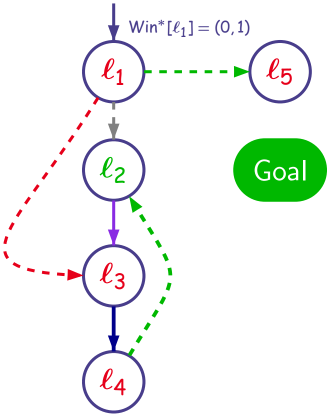
Initialization:

Passed $\leftarrow \{q_0\}$;
Waiting $\leftarrow \{(q_0, a, q') \mid a \in Act\ q \xrightarrow{a} q'\}$;
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 $q' \not\in$ 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

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

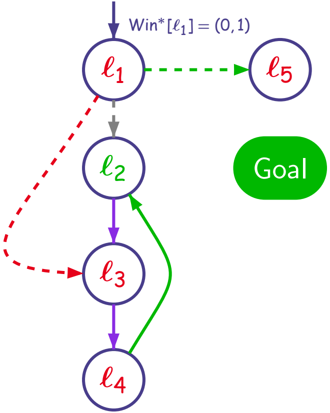
Initialization:

Passed $\leftarrow \{q_0\}$;
Waiting $\leftarrow \{(q_0, a, q') \mid a \in Act\ q \xrightarrow{a} q'\}$;
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]



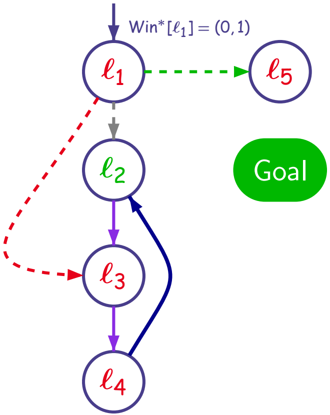
→ Current
→ Explored Forwards
Dashed = in Waiting List

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

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 $q' \not\in$ 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

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

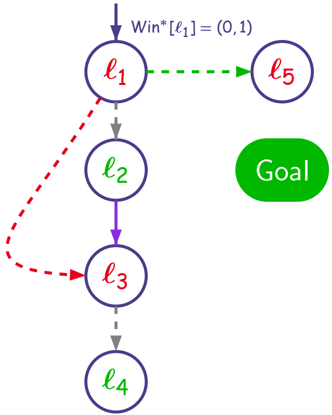
Initialization:

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

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

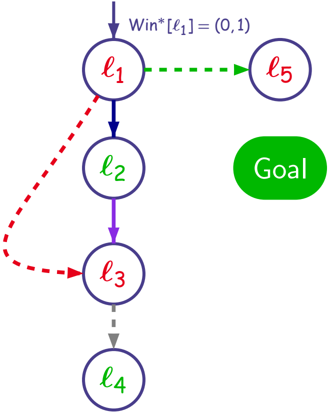
Initialization:

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

Main:

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

Liu & Smolka Algorithm [Liu & Smolka'98]



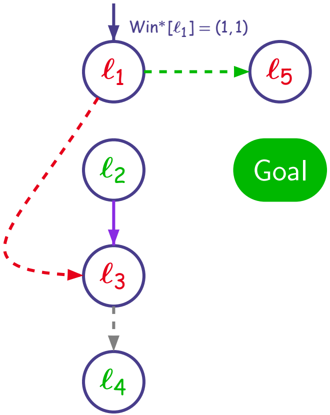
$\longrightarrow$ Current
$\longrightarrow$ Explored Forwards
Dashed = in Waiting List

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

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

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

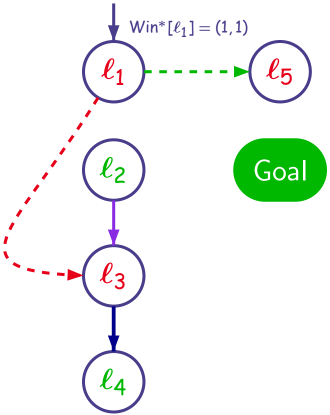
Initialization:

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

Main:

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

Liu & Smolka Algorithm [Liu & Smolka'98]



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

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

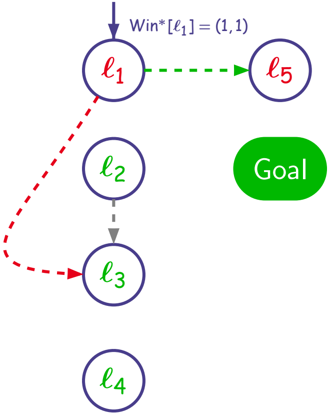
Initialization:

```
    Passed ← {q0};
    Waiting ← {(q0, a, q') | a ∈ Act q -a-> q'};
    Win[q0] ← (q0 ∈ Goal ? 1 : 0);
    Depend[q0] ← ∅;
```

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]



→ Current
→ Explored Forwards
Dashed = in Waiting List

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

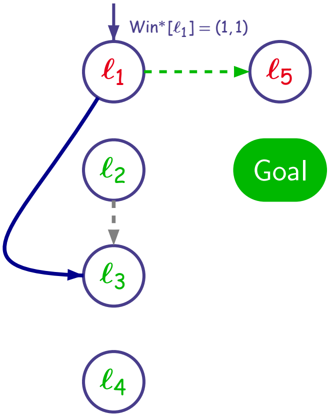
Initialization:

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

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

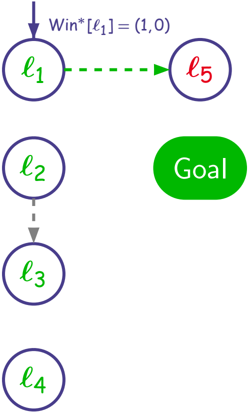
Initialization:

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 $((\text{Waiting} \neq \emptyset) \wedge \text{Win}[q_0] \neq 1))$ **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 $\cup \{(q', \alpha, q'') \mid q' \xrightarrow{\alpha} q''\}$;
  $\text{Win}^*[q] \leftarrow (0, \#\{q \xrightarrow{u}\})$;
  if Win$[q']$ **then** Waiting $\leftarrow$ Waiting $\cup \{e\}$;
 }
 else (* reevaluate *)
  $\text{Win}^*[q] \leftarrow$ **Update**$(\text{Win}^*[q])$;
  if $(\text{Win}^*[q] = (k, 0) \wedge k \geq 1)$ **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]



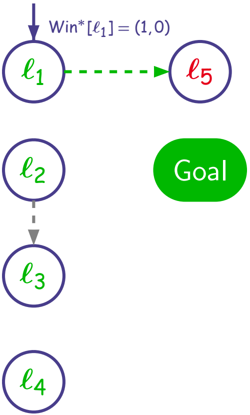
$\longrightarrow$ Current
$\longrightarrow$ Explored Forwards
Dashed = in Waiting List

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

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]



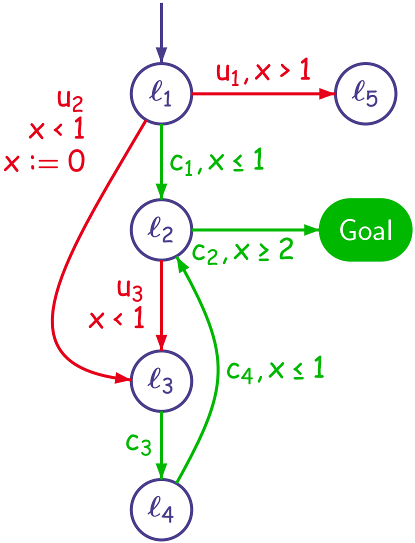
$\longrightarrow$ Current
$\longrightarrow$ Explored Forwards
Dashed = in Waiting List

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

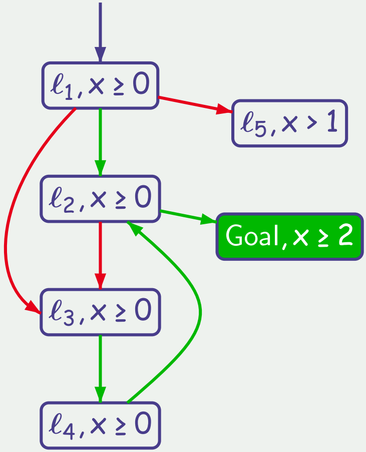
Initialization:
 Passed $\leftarrow \{q_0\}$;
 Waiting $\leftarrow \{(q_0, a, q') \mid a \in Act\ q \xrightarrow{a} q'\}$;
 Win$[q_0] \leftarrow (q_0 \in$ Goal ? 1 : 0);
 Depend$[q_0] \leftarrow \emptyset$;

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

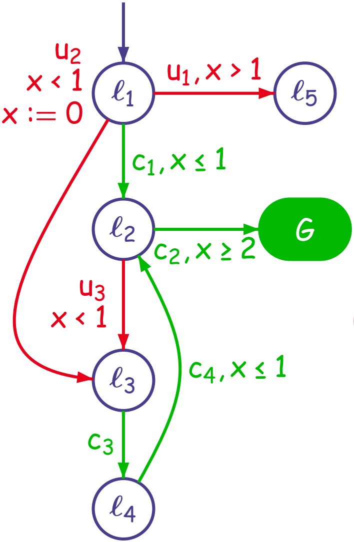
On-The-Fly Algorithm for Timed Games (1)



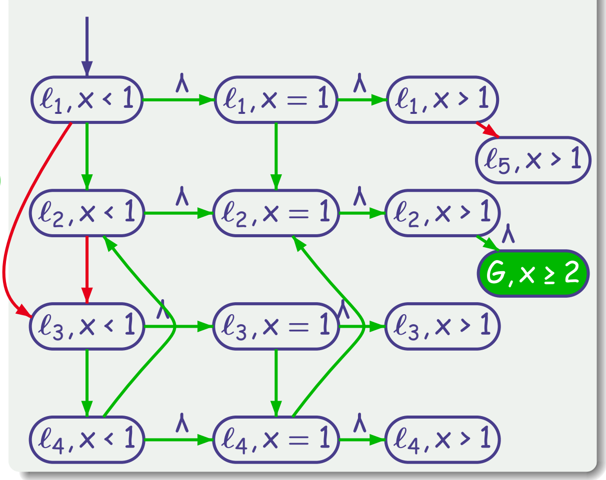
Using the Simulation Graph



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



Stable Partitionning



Towards a True On-The-Fly Algorithm

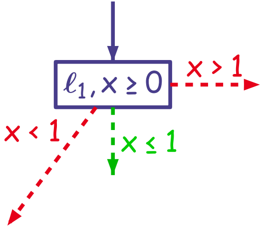
To Do:

- 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:

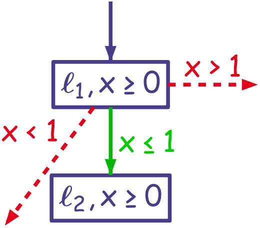
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, a, S') \mid S' = Post_a(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{\text{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

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$[S'] \leftarrow S' \cap (\{\text{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
  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:

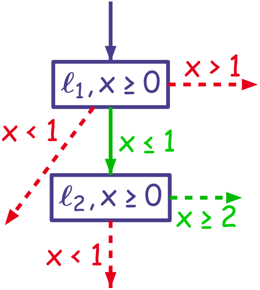
  Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
  Waiting $\leftarrow \{(S_0, a, S') \mid S' = Post_a(S_0)^{\nearrow}\}$;
  Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
  Depend$[S_0] \leftarrow \emptyset$;

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:

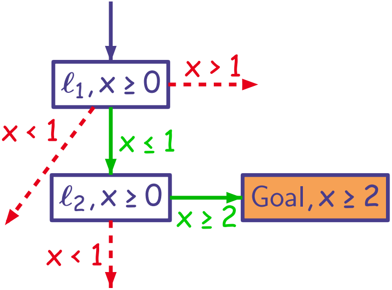
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, a, S') \mid S' = Post_a(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ (($\ell_0, 0) \notin$ 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:

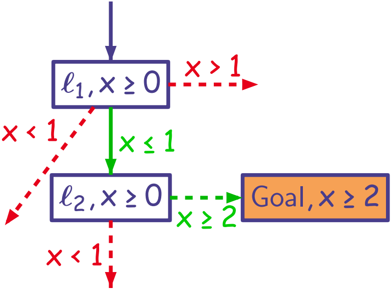
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
Win[S_0] $\leftarrow S_0 \cap (\{\text{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
Depend[S_0] $\leftarrow \emptyset$;

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'] $\leftarrow S' \cap (\{\text{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
    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:

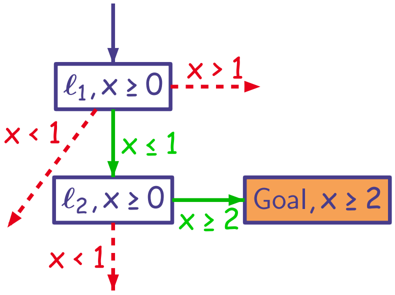
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

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:

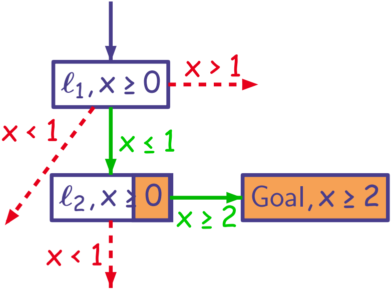
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ (($\ell_0, 0$) $\notin$ 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:

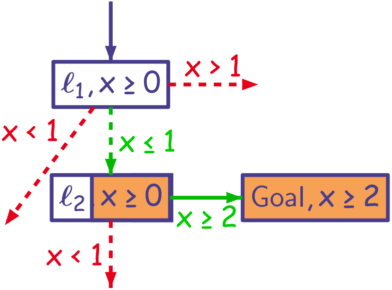
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, a, S') \mid S' = Post_a(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{\text{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

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$[S'] \leftarrow S' \cap (\{\text{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
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 *)
$\text{Win}^* \leftarrow \text{Pred}_t(\text{Win}[S] \cup \bigcup_{S \xrightarrow{c} T} \text{cPred}(\text{Win}[T]), \bigcup_{S \xrightarrow{u} T} \text{uPred}(T \setminus \text{Win}[T])) \cap S$;
if (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:

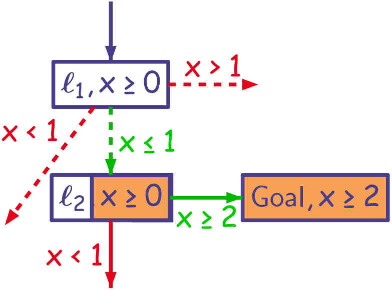
$\quad$ Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
$\quad$ Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
$\quad$ Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
$\quad$ Depend$[S_0] \leftarrow \emptyset$;

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$[T]$),
$\quad\quad\quad\quad \bigcup_{S \xrightarrow{u} T}$ 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:

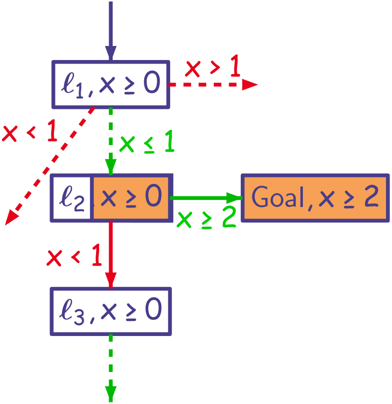
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
$Win[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
$Depend[S_0] \leftarrow \emptyset$;

Main:

while $((Waiting \neq \emptyset) \wedge ((\ell_0, 0) \not\sqsubseteq Win[S_0]))$ **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:

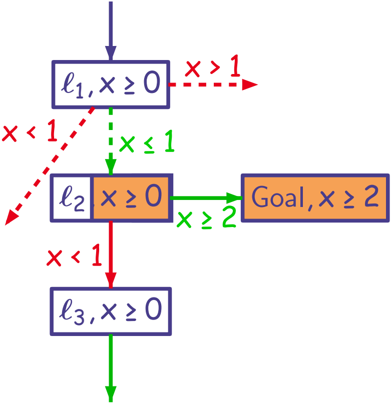
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
$\text{Win}[S_0] \leftarrow S_0 \cap (\{\text{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
$\text{Depend}[S_0] \leftarrow \emptyset$;

Main:

while $((\text{Waiting} \neq \emptyset) \wedge ((\ell_0, 0) \notin \text{Win}[S_0]))$ **do**
  $e = (S, \alpha, S') \leftarrow \text{pop(Waiting)}$;
  if $S' \notin$ Passed **then**
    Passed $\leftarrow$ Passed $\cup \{S'\}$;
    $\text{Depend}[S'] \leftarrow \{(S, \alpha, S')\}$;
    $\text{Win}[S'] \leftarrow S' \cap (\{\text{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
    Waiting $\leftarrow$ Waiting $\cup \{(S', \alpha, S'') \mid S'' = Post_\alpha(S')^{\nearrow}\}$;
    if $\text{Win}[S'] \neq \emptyset$ **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]; $\text{Win}[S] \leftarrow \text{Win}^*$;
    $\text{Depend}[S'] \leftarrow \text{Depend}[S'] \cup \{e\}$;
  endif
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Initialization:

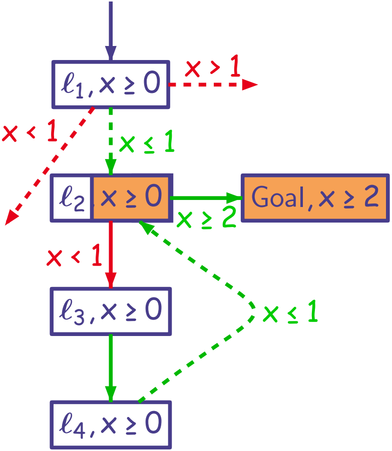
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, a, S') \mid S' = Post_a(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{\text{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

Main:

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

▸▸ Skip algorithm

Liu & Smolka for Timed Games



Initialization:

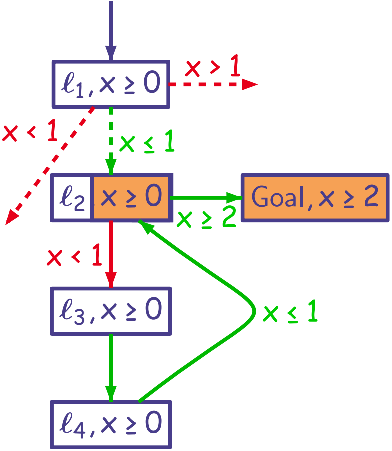
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, a, S') \mid S' = Post_a(S_0)^{\nearrow}\}$;
$\mathrm{Win}[S_0] \leftarrow S_0 \cap (\{\mathrm{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
$\mathrm{Depend}[S_0] \leftarrow \emptyset$;

Main:

while $((\mathrm{Waiting} \neq \emptyset) \wedge ((\ell_0, 0) \notin \mathrm{Win}[S_0]))$ **do**
  $e = (S, a, S') \leftarrow \mathrm{pop}(\mathrm{Waiting})$;
  if $S' \notin$ Passed **then**
    Passed $\leftarrow$ Passed $\cup \{S'\}$;
    $\mathrm{Depend}[S'] \leftarrow \{(S, a, S')\}$;
    $\mathrm{Win}[S'] \leftarrow S' \cap (\{\mathrm{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
    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 $\cup\ \mathrm{Depend}[S]$; $\mathrm{Win}[S] \leftarrow \mathrm{Win}^*$;
    $\mathrm{Depend}[S'] \leftarrow \mathrm{Depend}[S'] \cup \{e\}$;
  endif
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Initialization:

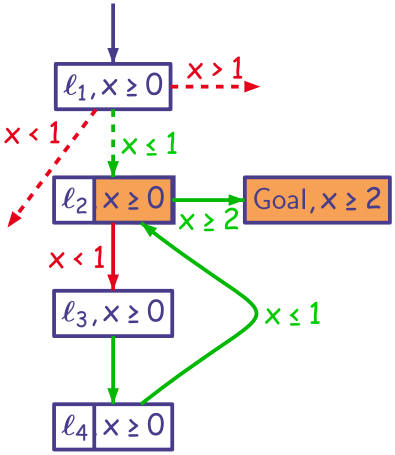
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
Win[S_0] $\leftarrow S_0 \cap (\{\text{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
Depend[S_0] $\leftarrow \emptyset$;

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'] $\leftarrow S' \cap (\{\text{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
    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* $\leftarrow \text{Pred}_t(\text{Win}[S] \cup \bigcup_{S \xrightarrow{c} T} \text{cPred}(\text{Win}[T]),$
    $\bigcup_{S \xrightarrow{u} T} \text{uPred}(T \setminus \text{Win}[T])) \cap S$;
    if (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:

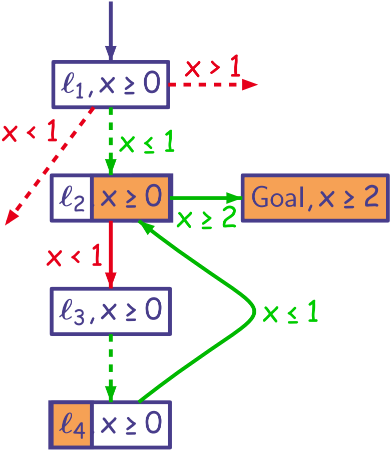
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{\mathrm{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

Main:

while $((\mathrm{Waiting} \neq \emptyset) \wedge ((\ell_0, 0) \notin \mathrm{Win}[S_0]))$ **do**
  $e = (S, \alpha, S') \leftarrow$ pop(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:

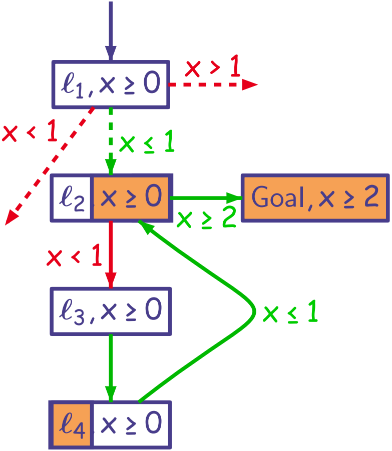
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
$Win[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
$Depend[S_0] \leftarrow \emptyset$;

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:

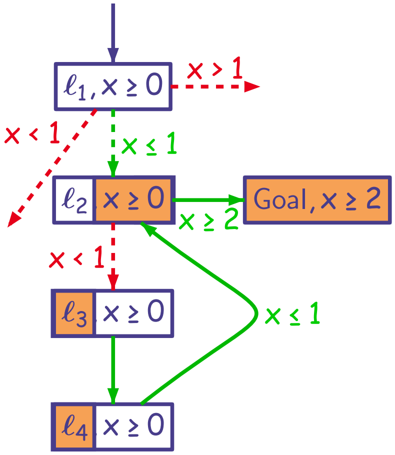
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
$\mathrm{Win}[S_0] \leftarrow S_0 \cap (\{\mathrm{Goal}\} \times \mathbb{R}_{\geq 0}^X)$;
$\mathrm{Depend}[S_0] \leftarrow \emptyset$;

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 $\cup \{S'\}$;
    $\mathrm{Depend}[S'] \leftarrow \{(S, \alpha, S')\}$;
    $\mathrm{Win}[S'] \leftarrow S' \cap (\{\mathrm{Goal}\} \times \mathbb{R}_{\geq 0}^X)$;
    Waiting $\leftarrow$ Waiting $\cup \{(S', \alpha, S'') \mid S'' = Post_\alpha(S')^{\nearrow}\}$;
    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 $\cup\, \mathrm{Depend}[S]$; $\mathrm{Win}[S] \leftarrow \mathrm{Win}^*$;
    $\mathrm{Depend}[S'] \leftarrow \mathrm{Depend}[S'] \cup \{e\}$;
  endif
endwhile

Skip algorithm

Liu & Smolka for Timed Games



Skip algorithm

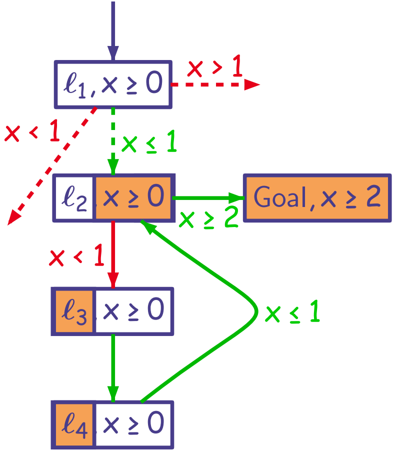
Initialization:

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 $\neq \emptyset$) $\wedge$ ((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

Liu & Smolka for Timed Games



Initialization:

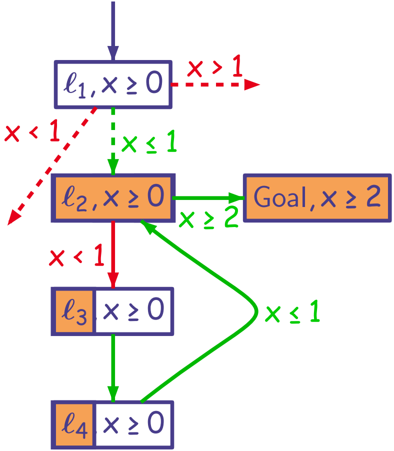
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
$Win[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
$Depend[S_0] \leftarrow \emptyset$;

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:

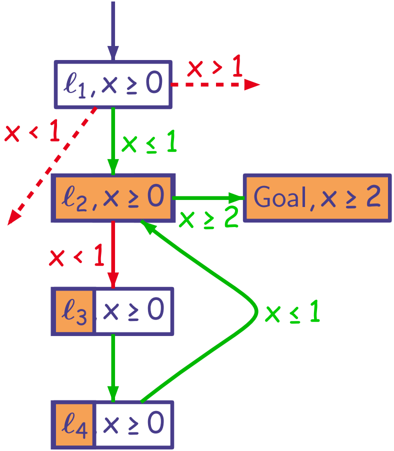
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
$\mathrm{Win}[S_0] \leftarrow S_0 \cap (\{\mathrm{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
$\mathrm{Depend}[S_0] \leftarrow \emptyset$;

Main:

while $((\mathrm{Waiting} \neq \emptyset) \wedge ((\ell_0, 0) \notin \mathrm{Win}[S_0]))$ **do**
  $e = (S, \alpha, S') \leftarrow \mathrm{pop}(\mathrm{Waiting})$;
  if $S' \notin$ Passed **then**
    Passed $\leftarrow$ Passed $\cup \{S'\}$;
    $\mathrm{Depend}[S'] \leftarrow \{(S, \alpha, S')\}$;
    $\mathrm{Win}[S'] \leftarrow S' \cap (\{\mathrm{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
    Waiting $\leftarrow$ Waiting $\cup \{(S', \alpha, S'') \mid S'' = Post_\alpha(S')^{\nearrow}\}$;
    if $\mathrm{Win}[S'] \neq \emptyset$ **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:

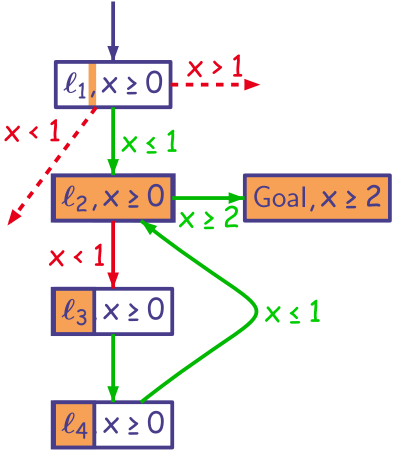
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

Main:

```
while ((Waiting ≠ ∅) ∧ ((ℓ0, 0) ∉ Win[S0])) do
    e = (S, α, S') ← pop(Waiting);
    if S' ∉ Passed then
        Passed ← Passed ∪ {S'};
        Depend[S'] ← {(S, α, S')};
        Win[S'] ← S' ∩ ({Goal} × R^X_{≥0});
        Waiting ← Waiting ∪ {(S', α, S'') | S'' = Post_α(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:

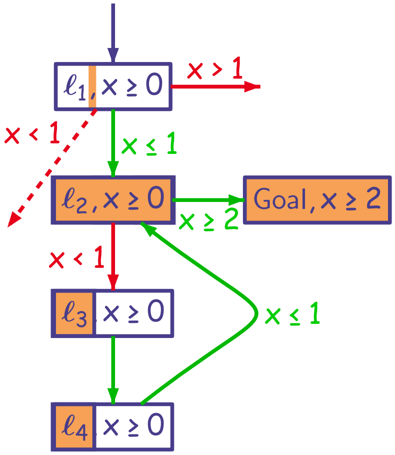
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
$\mathrm{Win}[S_0] \leftarrow S_0 \cap (\{\mathrm{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
$\mathrm{Depend}[S_0] \leftarrow \emptyset$;

Main:

while ((Waiting $\neq \emptyset$) $\wedge$ ($(\ell_0, 0) \notin \mathrm{Win}[S_0]$)) **do**
  $e = (S, \alpha, S') \leftarrow$ pop(Waiting);
  if $S' \notin$ Passed **then**
    Passed $\leftarrow$ Passed $\cup \{S'\}$;
    $\mathrm{Depend}[S'] \leftarrow \{(S, \alpha, S')\}$;
    $\mathrm{Win}[S'] \leftarrow S' \cap (\{\mathrm{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
    Waiting $\leftarrow$ Waiting $\cup \{(S', \alpha, S'') \mid S'' = Post_\alpha(S')^{\nearrow}\}$;
    if $\mathrm{Win}[S'] \neq \emptyset$ **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:

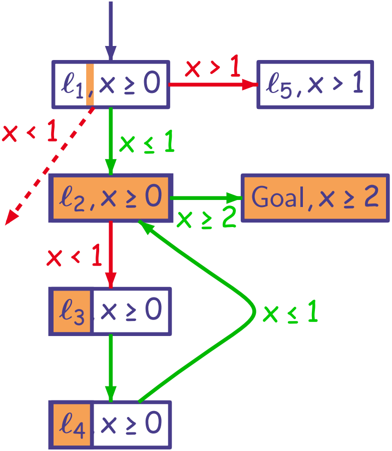
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, \alpha, S') \mid S' = Post_\alpha(S_0)^{\nearrow}\}$;
$Win[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
$Depend[S_0] \leftarrow \emptyset$;

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:

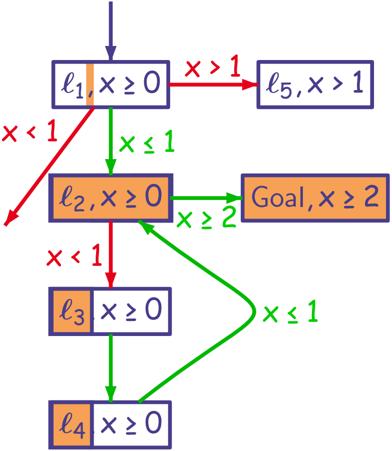
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, a, S') \mid S' = Post_a(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{\text{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

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$[S'] \leftarrow S' \cap (\{\text{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
  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:

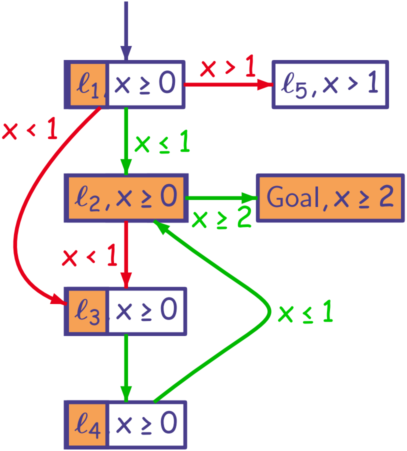
Passed $\leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, 0)\}^{\nearrow}$;
Waiting $\leftarrow \{(S_0, a, S') \mid S' = Post_a(S_0)^{\nearrow}\}$;
Win$[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
Depend$[S_0] \leftarrow \emptyset$;

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:

$\quad$ 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$[S_0] \leftarrow S_0 \cap (\{\text{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
$\quad$ 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$[S'] \leftarrow S' \cap (\{\text{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
$\quad\quad$ 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$[T]$),
$\quad\quad\quad\quad\quad\quad\quad\quad$ $\bigcup_{S \xrightarrow{u} T}$ 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

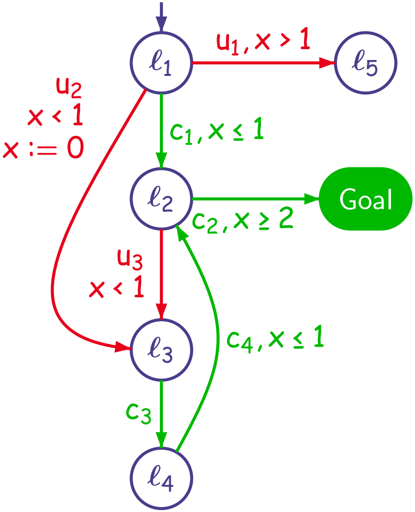
Skip algorithm

Summary of the Results [Concur'05]

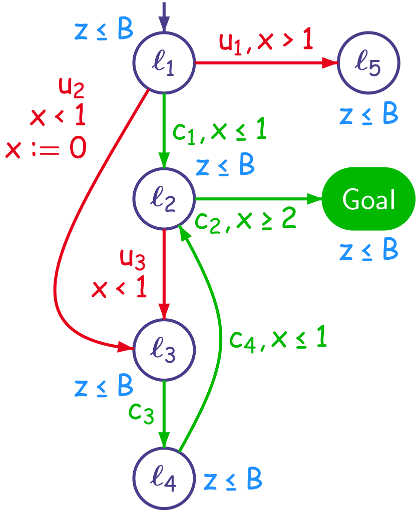
- A True on-the-fly algorithm for reachability control
- Winning Strategies can be computed
- Termination
  A symbolic edge (S, a, T) will be at most (1+ # regions(T)) times in *Waiting* list
- Complexity
  A region may be in many symbolic states
  Our algorithm: Not linear in the size of the region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

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

Time Optimality for Free



Time Optimality for Free

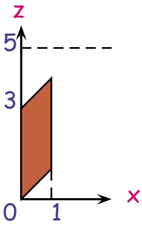


Assume:

- 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:

- Control of Timed Systems: Basics
- Control of Discrete Event Systems
- Control of Timed Systems
- Advanced Subjects
- **Conclusion**

Conclusion

- Recent Research Results:
  - Implementability of controllers
  - Optimality of controllers
  - Efficient algorithms for solving Timed Games
  - Control under Partial-Observation
- Ongoing work:
  - Efficient Algorithms for Safety, Büchi games
  - Data Structures for optimal control
  - Optimal control for infinite schedules
  - Synthesis of robust controllers
  - Abstraction/Refinement for synthesis of controllers

Merci !

Conclusion

- Recent Research Results:
  - Implementability of controllers
  - Optimality of controllers
  - Efficient algorithms for solving Timed Games
  - Control under Partial-Observation
- Ongoing work:
  - Efficient Algorithms for Safety, Büchi games
  - Data Structures for optimal control
  - Optimal control for infinite schedules
  - Synthesis of robust controllers
  - Abstraction/Refinement for synthesis of controllers

Merci !

Conclusion

- Recent Research Results:
  - Implementability of controllers
  - Optimality of controllers
  - Efficient algorithms for solving Timed Games
  - Control under Partial-Observation
- Ongoing work:
  - Efficient Algorithms for Safety, Büchi games
  - Data Structures for optimal control
  - Optimal control for infinite schedules
  - Synthesis of robust controllers
  - Abstraction/Refinement for synthesis of controllers

Merci !

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 et al.'01] R. Alur, S. La Torre, and G. J. Pappas.
Optimal paths in weighted timed automata.
In *Proc. 4th Int. Work. Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *LNCS*, pages 49–62. Springer, 2001.

[Alur et al.'04] R. Alur, M, Bernadsky, and P. Madhusudan.
Optimal reachability in weighted timed games.
In *Proc. 31st International Colloquium on Automata, Languages and Programming (ICALP'04)*, Lecture Notes in Computer Science. Springer, 2004.

[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.

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

References (cont.)

[De Alfaro et al.'01] Luca de Alfaro, Thomas A. Henzinger, and Rupak Majumdar. **Symbolic algorithms for infinite-state games.** In *Proc. 12th International Conference on Concurrency Theory (CONCUR'01)*, volume 2154 of *LNCS*, pages 536–550. Springer, 2001.

[Asarin et al.'98] Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. **Controller synthesis for timed automata.** In *Proc. IFAC Symposium on System Structure and Control*, pages 469–474. Elsevier Science, 1998.

[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.

[Larsen et al.'01] Kim G. Larsen, Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Paul Pettersson, Judi Romijn, and Frits Vaandrager. **Minimum-cost reachability for priced timed automata.** In *Proc. 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2001.

[Bouyer et al.'06] Patricia Bouyer, Thomas Brihaye, and Fabrice Chevalier. **Control in o-minimal hybrid systems.** In *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science (LICS'06)*, pages 367–378, Seattle, Washington, USA, August 2006. IEEE Computer Society Press.

[Büchi & Landweber'69] J.R. Büchi and L.H. Landweber. **Solving sequential conditions by finite-state operators.** *Trans. of the AMS*; 138:295–311.

References (cont.)

[Bouyer et al.'04a] 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.

[Bouyer et al.'04b] Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim G. Larsen.
Synthesis of optimal strategies using HyTech.
In *Proc. of the Workshop on Games in Design and Verification (GDV'04)*, volume 119 of *Elec. Notes in Theo. Comp. Science*, pages 11–31. Elsevier, 2005.

[Bouyer et al.'07] Patricia Bouyer, Thomas Brihaye, and Fabrice Chevalier.
Weighted o-minimal hybrid systems are more decidable than weighted timed automata!
In Sergei N. Artemov, editor, *Proceedings of the Symposium on Logical Foundations of Computer Science (LFCS'07)*, Lecture Notes in Computer Science, New-York, NY, USA, June 2007. Springer.
To appear.

[Bouyer et al.'06a] Patricia Bouyer, Kim G. Larsen, Nicolas Markey, and Jacob Illum Rasmussen.
Almost optimal strategies in one-clock priced timed automata.
In Naveen Garg and S. Arun-Kumar, editors, *Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337 of *Lecture Notes in Computer Science*, pages 345–356, Kolkata, India, December 2006. Springer.

[Bouyer et al.'06b] Patricia Bouyer, Thomas Brihaye, and Nicolas Markey.
Improved undecidability results on weighted timed automata.
Information Processing Letters, 98(5):188–194, June 2006.

References (cont.)

[Bouyer et al.'04c] Patricia Bouyer, Ed Brinksma, and Kim G. Larsen.
Staying alive as cheaply as possible.
In Rajeev Alur and George J. Pappas, editors, *Proceedings of the 7th International Conference on Hybrid Systems: Computation and Control (HSCC'04)*, volume 2993 of *Lecture Notes in Computer Science*, pages 203–218, Philadelphia, Pennsylvania, USA, March 2004. Springer.

[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.

[De Wulf et al.'04a] Martin De Wulf, Laurent Doyen, Nicoals Markey, and Jean-François Raskin.
Robustness and implementability of timed automata.
In *Proceedings of FORMATS-FTRTFT 2004: Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, Lecture Notes in Computer Science 3253, pages 118–133. Springer-Verlag, 2004.

[De Wulf et al.'04b] Martin De Wulf, Laurent Doyen, and Jean-François Raskin.
Almost ASAP semantics: From timed models to timed implementations.
In *Proceedings of HSCC 2004: Hybrid SystemsŠComputation and Control*, Lecture Notes in Computer Science 2993, pages 296–310. Springer-Verlag, 2004.

References (cont.)

[De Wulf et al.'05a] Martin De Wulf, Laurent Doyen, and Jean-François Raskin.
Almost ASAP semantics: From timed models to timed implementations.
Formal Aspects of Computing, 17(3):319–341, 2005.

[De Wulf et al.'05b] Martin De Wulf, Laurent Doyen, and Jean-François Raskin.
Systematic implementation of real-time models.
In *Proceedings of FM 2005: Formal Methods*, Lecture Notes in Computer Science 3582, pages 139–156. Springer-Verlag, 2005.

[C. et al.'02] Franck Cassez, Thomas A. Henzinger, and Jean-François Raskin.
A comparison of control problems for timed and hybrid systems.
In *Proc. 5th Int. Workshop on Hybrid Systems: Computation and Control (HSCC'02)*, volume 2289 of *LNCS*, pages 134–148. Springer, 2002.

[Henzinger & Kopke'99] T.A. Henzinger and P.W. Kopke.
Discrete-time control for rectangular hybrid automata.
Theoretical Computer Science, 221:369–392, 1999.

[Henzinger et al.'99] Thomas A. Henzinger, Benjamin Horowitz, and Rupak Majumdar.
Rectangular hybrid games.
In *Proc. 10th International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 1999.

[Henzinger et al.'95] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya.
What's decidable about hybrid automata?
Journal of Computer and System Sciences, 57:94–124, 1998.

[Henzinger & Kopke'97] Thomas A. Henzinger and Peter W. Kopke.
Discrete-time control for rectangular hybrid automata.
Theoretical Computer Science, 221:369–392, 1999.

References (cont.)

[Hoffmann & Wong-Toi'92] G. Hoffmann and Howard Wong-Toi.
The input-output control of real-time discrete-event systems.
In *Proceedings of the 13th Annual Real-time Systems Symposium*, pages 256–265. IEEE Computer Society Press, 1992.

[La Torre et al.'02] Salvatore La Torre, Supratik Mukhopadhyay, and Aniello Murano.
Optimal-reachability and control for acyclic weighted timed automata.
In *Proc. 2nd IFIP International Conference on Theoretical Computer Science (TCS 2002)*, volume 223 of *IFIP Conference Proceedings*, pages 485–497. Kluwer, 2002.

[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.

[Ramadge & Wonham'87] P.J. Ramadge and W.M. Wonham.
Supervisory control of a class of discrete event processes.
SIAM J. of Control and Optimization, 25:206–230, 1987

[Ramadge & Wonham'89] P.J. Ramadge and W.M. Wonham.
The control of discrete event processes.
Proc. of IEEE, 77:81–98, 1989

[Brihaye et al.'05] Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin.
On optimal timed strategies.
In *FORMATS*, pages 49–64, 2005.

References (cont.)

[Thistle & Wonham'94] J.G. Thistle and W.M. Wonham.
Control of infinite behavior of finite automata.
SIAM J. of Control and Optimization, 32:1075–1097, 1994

[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.

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:

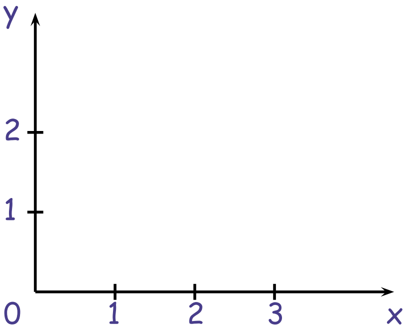
- $Q = L \times \mathbb{R}_{\geq 0}^X$
- $q_0 = (\ell_0, \overline{0})$
- $\longrightarrow$ consists in:

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

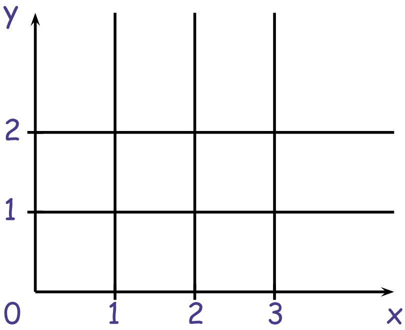
delay transition: $(\ell, v) \xrightarrow{d} (\ell, v + d) \iff d \in \mathbb{R}_{\geq 0} \wedge v + d \models inv(\ell)$

The Region Abstraction

[Alur & Dill'94]



The Region Abstraction

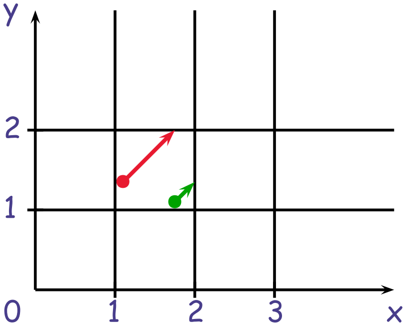


Build an equivalence relation which is of finite index and is:

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

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

The Region Abstraction



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

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

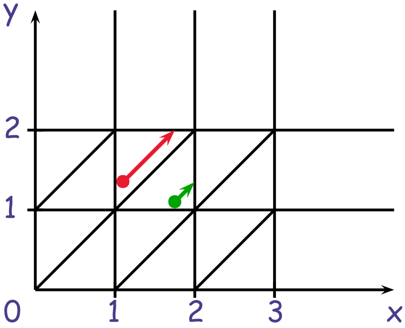
  $r, r' \in R \implies \forall$ constraints $g,\quad r \models g \iff r' \models g$

- "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:

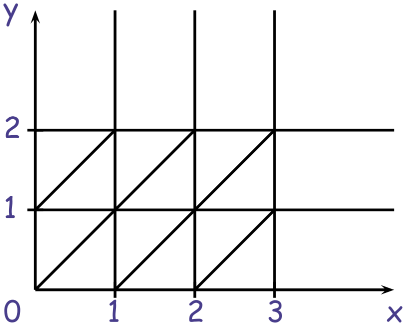
- "compatible" with clock constraints ($g ::= x \sim c \quad g \wedge g$)

  $r, r' \in R \implies \forall$ 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



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

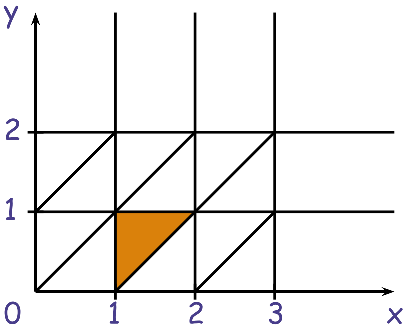
- "compatible" with clock constraints ($g ::= x \sim c \quad g \wedge g$)

  $r, r' \in R \implies \forall$ 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



region defined by

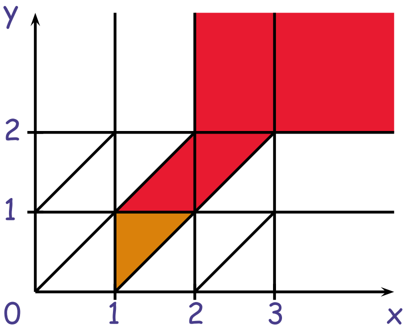
$I_x =]1;2[$ $I_y =]0;1[$

$\{x\} < \{y\}$

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]



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

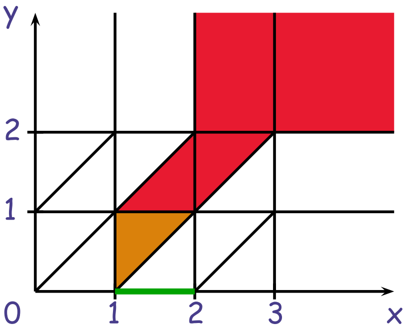
delay successors

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 $w = (a, 1.2)(b, 3.4)(a, 6.256)$; $\mathrm{untimed}(w) = aba$
- 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

The Region Automaton

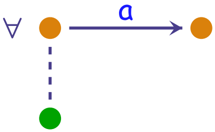
- For each transition $\ell \xrightarrow{g,a,C:=0} \ell'$ of the TA
- Build transitions in the region automaton RA: $(\ell, R) \xrightarrow{a} (\ell', R')$ if:
  - there exists R'' a delay successor of R s.t.
  - R'' satisfies the guard g ($R'' \subseteq [\![g]\!]$)
  - $R''[C \leftarrow 0]$ is included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

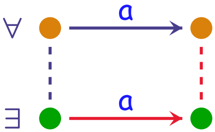
Time-abstract bisimulation



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

with $v_i \in R_i$ for all i.

Time-abstract bisimulation



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

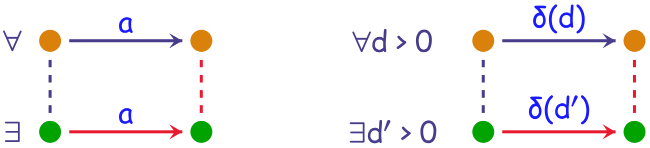
with $v_i \in R_i$ for all i.

Time-abstract bisimulation

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

with $v_i \in R_i$ for all i.

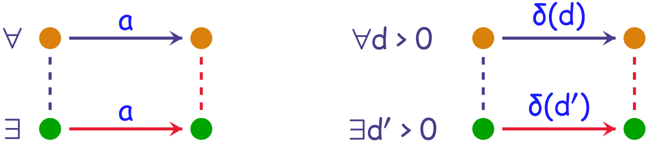
Time-abstract bisimulation



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

with $v_i \in R_i$ for all i.

Time-abstract bisimulation



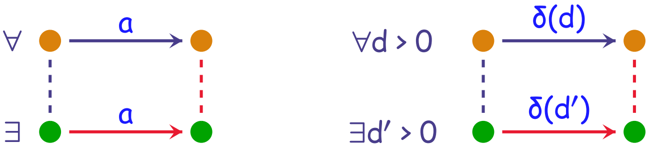
$$(\ell_0, v_0) \xrightarrow{a_1, t_1} (\ell_1, v_1) \xrightarrow{a_2, t_2} (\ell_2, v_2) \xrightarrow{a_3, t_3} \cdots$$

$$\updownarrow \qquad\qquad \updownarrow \qquad\qquad \updownarrow$$

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

with $v_i \in R_i$ for all i.

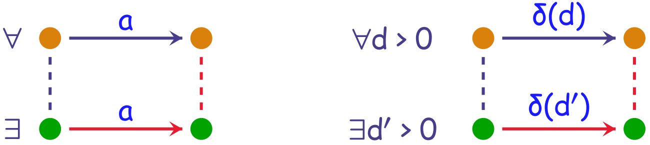
Time-abstract bisimulation



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

with $v_i \in R_i$ for all i.

Time-abstract bisimulation



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

with $v_i \in R_i$ for all i.

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 $\forall 0 \leq e' < e, \exists (\ell'', v'') \in (L \times \mathbb{R}_{\geq 0}^X)$ 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

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

q $q' \in X$

$\text{Pred}_\delta(X, Y)$ 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

- $\mathrm{cPred}(X) = \bigcup_{c \in Act_c} \mathrm{Pred}^c(X)$ $\mathrm{uPred}(X) = \bigcup_{u \in Act_u} \mathrm{Pred}^u(X)$
  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

- $\mathrm{cPred}(X) = \bigcup_{c \in Act_c} \mathrm{Pred}^c(X)$ $\quad$ $\mathrm{uPred}(X) = \bigcup_{u \in Act_u} \mathrm{Pred}^u(X)$
  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

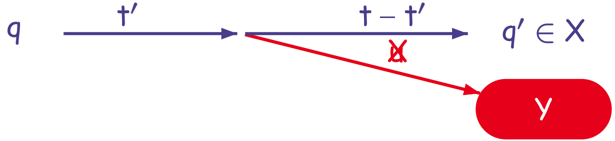
$$\pi_\delta(X) = \mathrm{Pred}_\delta\left(\mathrm{cPred}(X), \mathrm{uPred}(\overline{X})\right)$$

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

Symbolic Computation For Timed Games

X is a state predicate

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



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

- Controllable Predecessors Operator for Timed Games

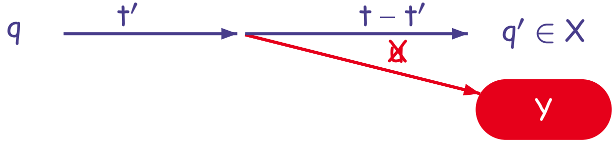
$$\pi_\delta(X) = \mathsf{Pred}_\delta\left(\mathsf{cPred}(X), \mathsf{uPred}(\overline{X})\right)$$

$\pi_\delta(X)$ 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:



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

- **Controllable Predecessors Operator** for Timed Games

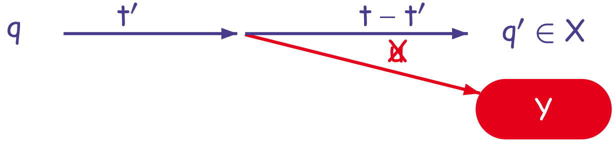
$$\pi_\delta(X) = \mathrm{Pred}_\delta\left(\mathrm{cPred}(X), \mathrm{uPred}(\overline{X})\right)$$

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

Symbolic Computation For Timed Games

X is a state predicate

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



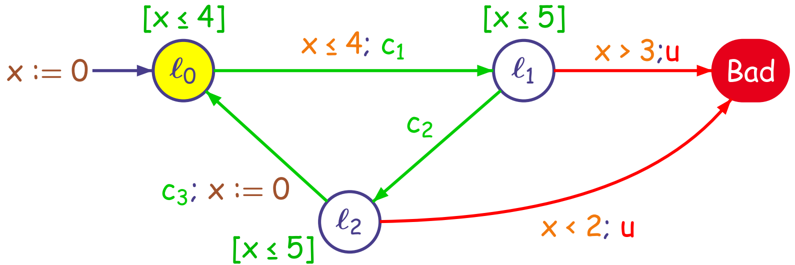
$\mathrm{Pred}_\delta(X, Y)$ is effectively computable for state predicates X, Y

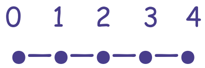
- Controllable Predecessors Operator for Timed Games

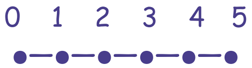
$$\pi_\delta(X) = \mathrm{Pred}_\delta\left(\mathrm{cPred}(X), \mathrm{uPred}(\overline{X})\right)$$

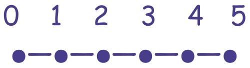
$\pi_\delta(X)$ is effectively computable for state predicate X.

Example of Computation for Safety Games



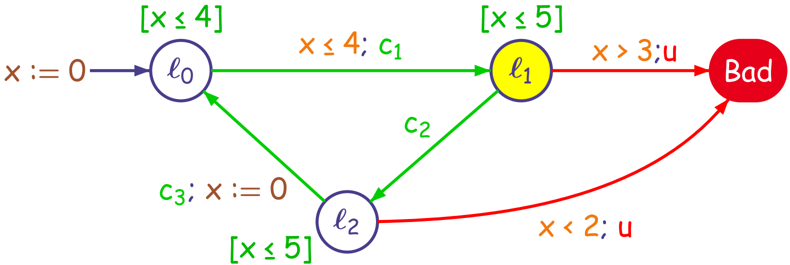


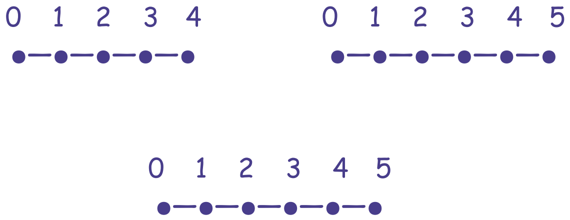




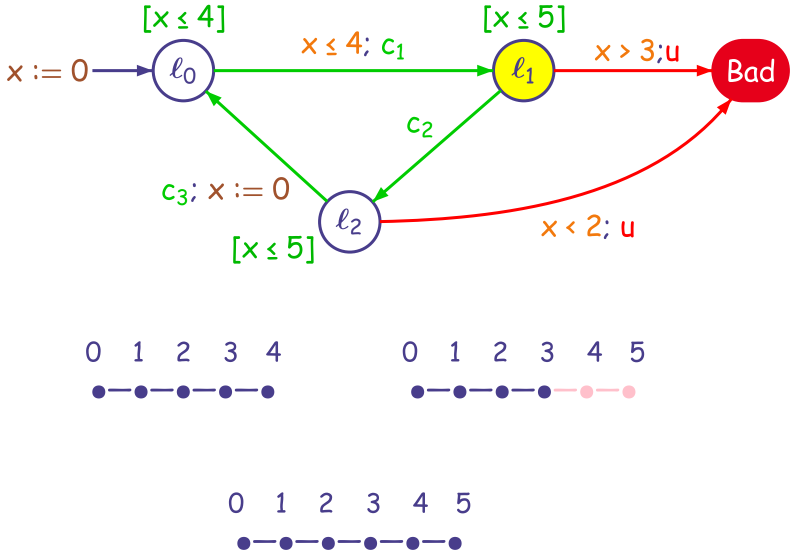
Skip

Example of Computation for Safety Games

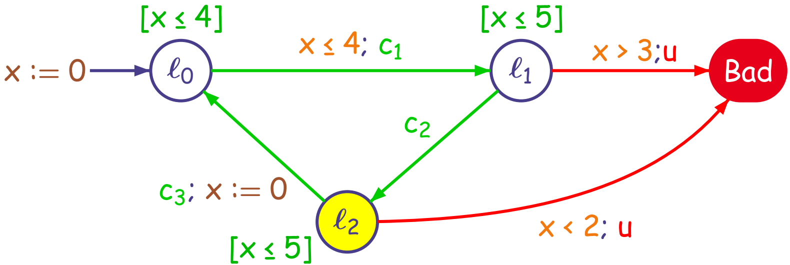


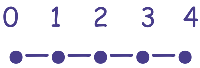


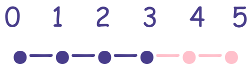
Example of Computation for Safety Games

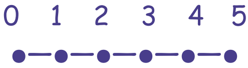


Example of Computation for Safety Games

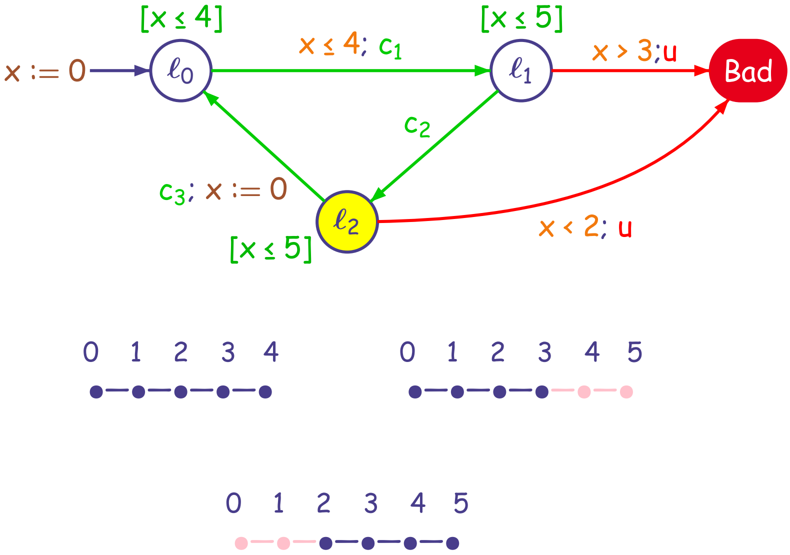




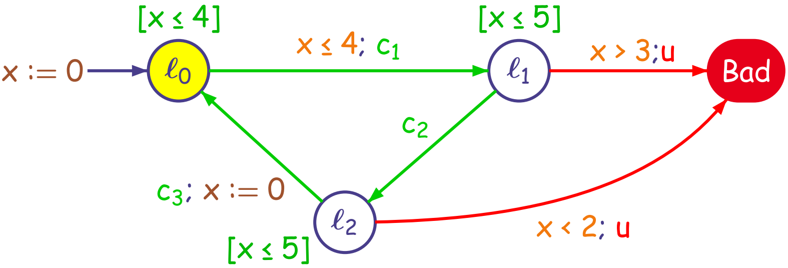


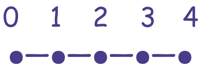


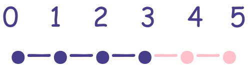
Example of Computation for Safety Games

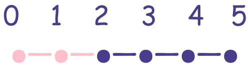


Example of Computation for Safety Games

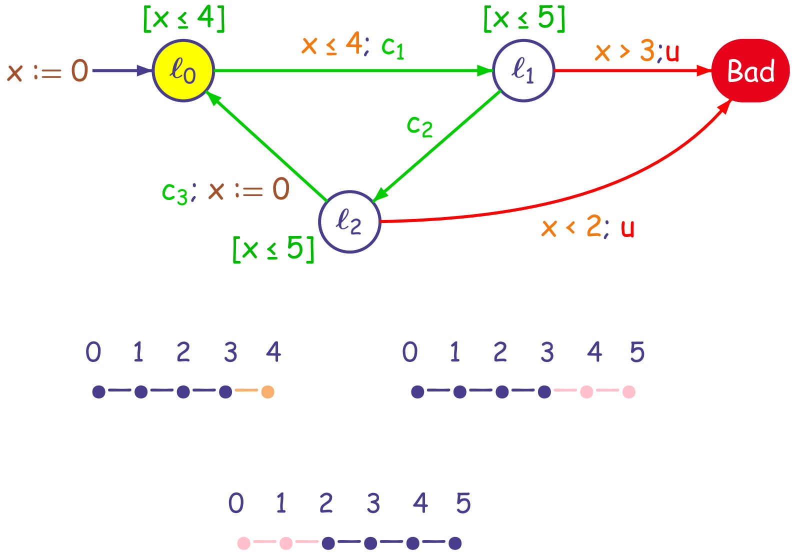




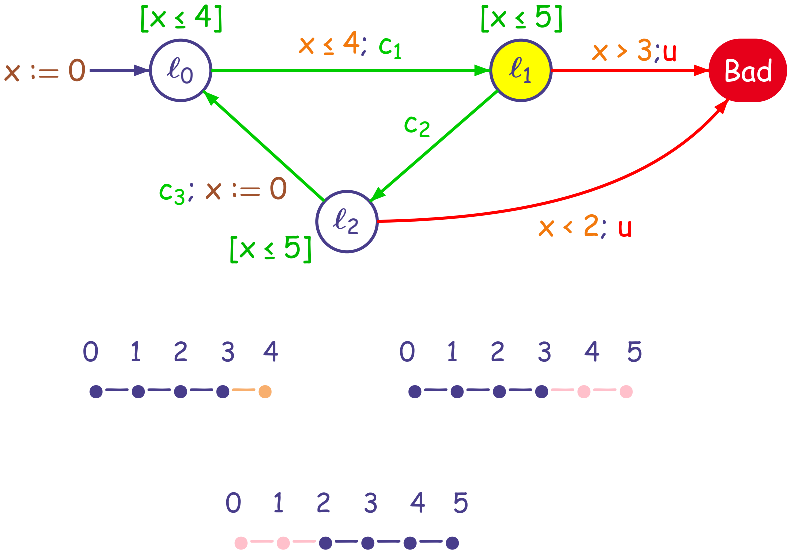




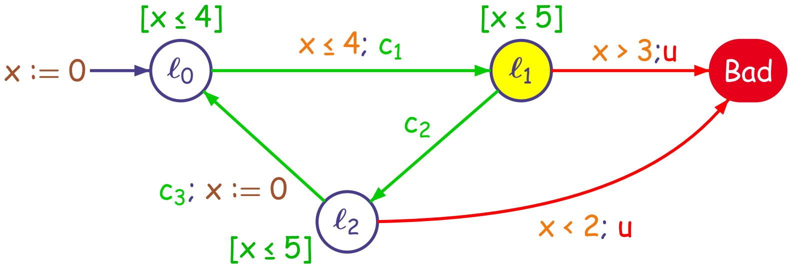
Example of Computation for Safety Games

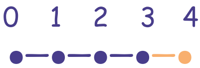


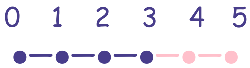
Example of Computation for Safety Games

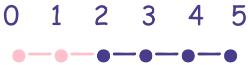


Example of Computation for Safety Games

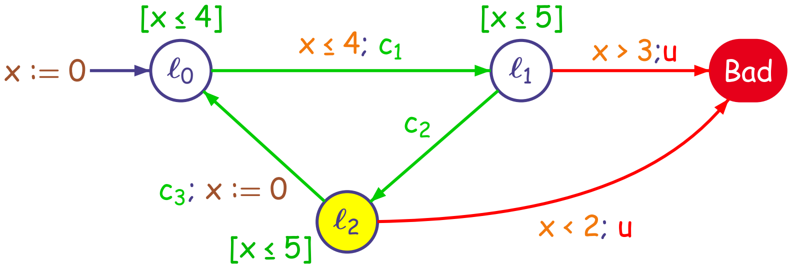


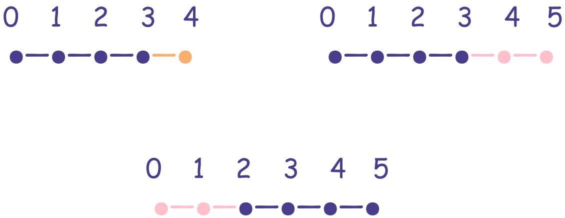




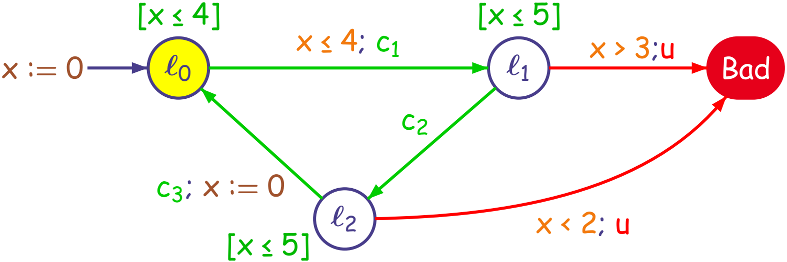


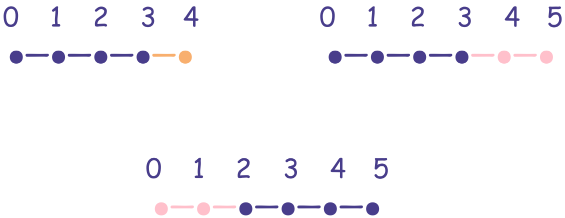
Example of Computation for Safety Games



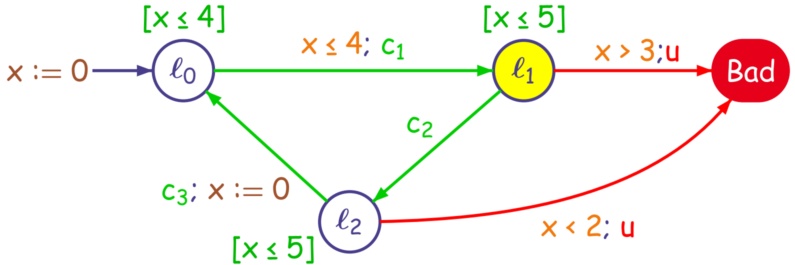


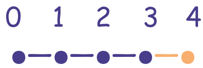
Example of Computation for Safety Games

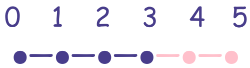


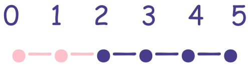


Example of Computation for Safety Games

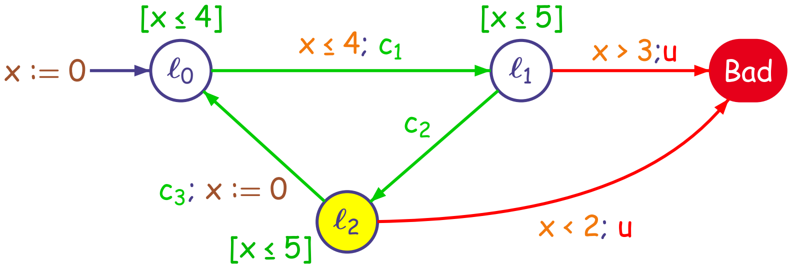


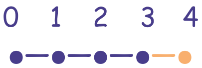


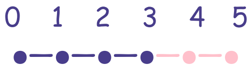


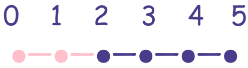


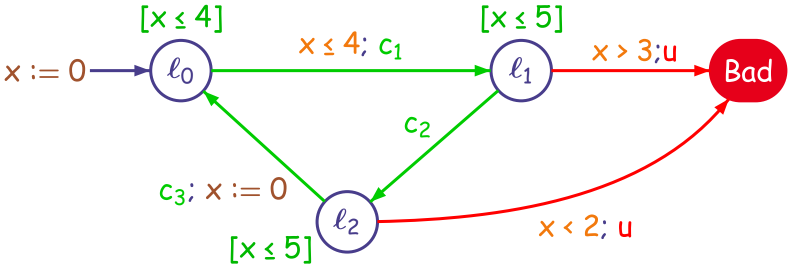
Example of Computation for Safety Games







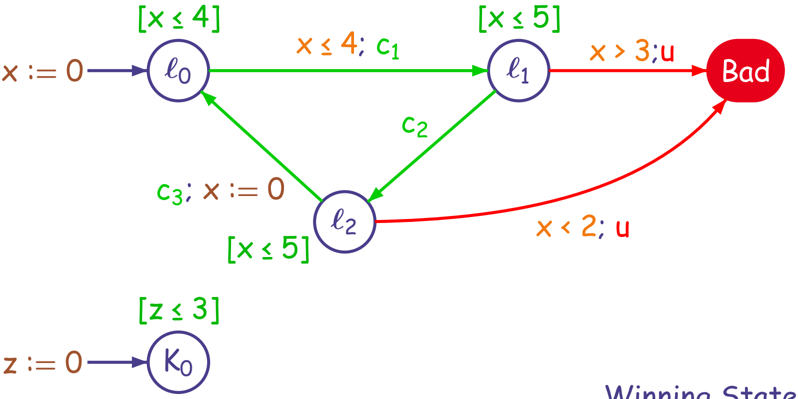




Winning States

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

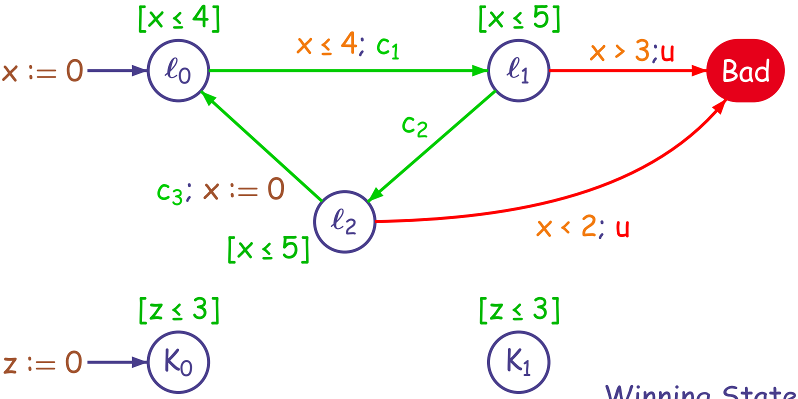
Example of Computation for Safety Games



Winning States

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

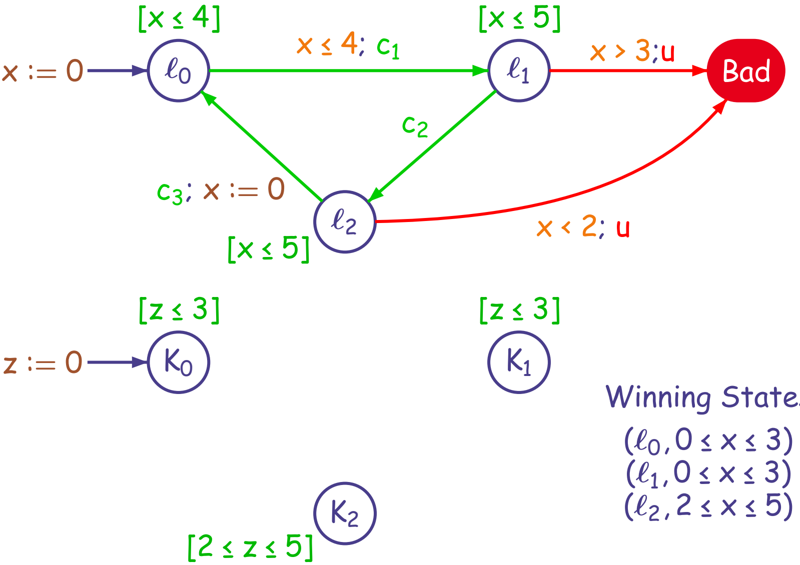
Example of Computation for Safety Games



Winning States

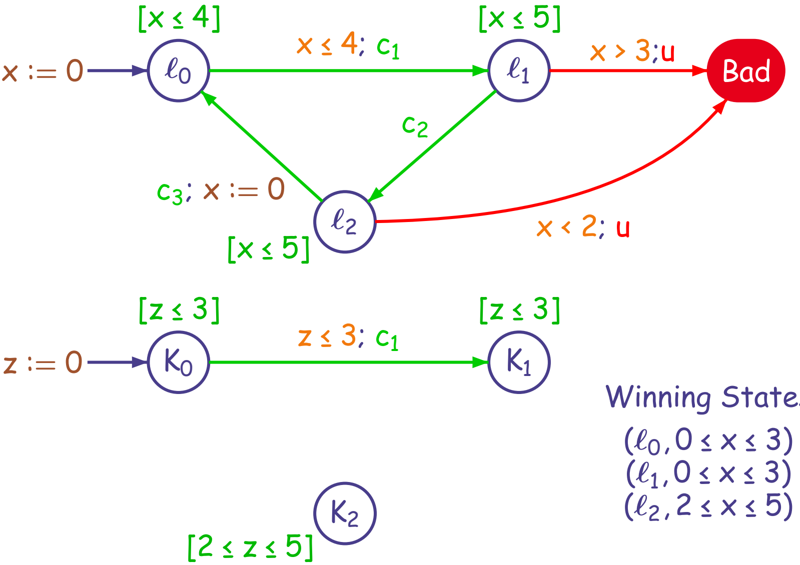
$(\ell_0, 0 \leq x \leq 3)$
$(\ell_1, 0 \leq x \leq 3)$
$(\ell_2, 2 \leq x \leq 5)$

Example of Computation for Safety Games



Winning States

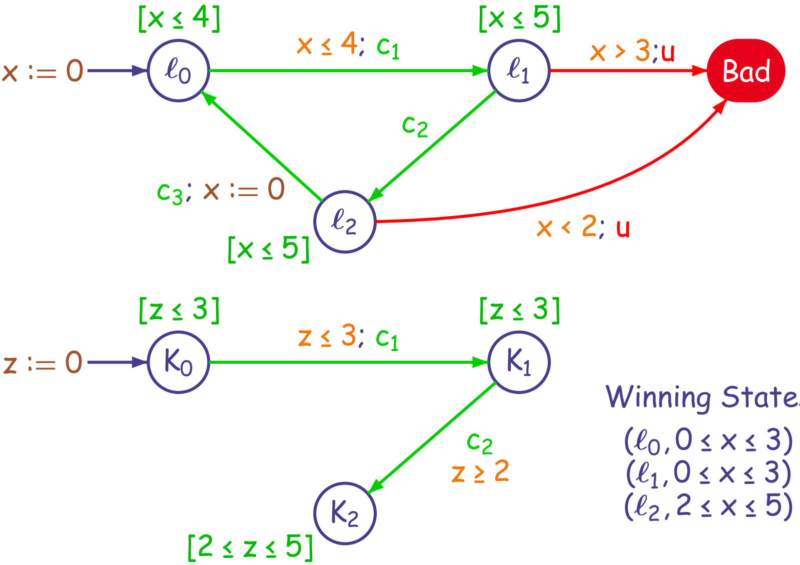
$(\ell_0, 0 \leq x \leq 3)$
$(\ell_1, 0 \leq x \leq 3)$
$(\ell_2, 2 \leq x \leq 5)$



Winning States

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

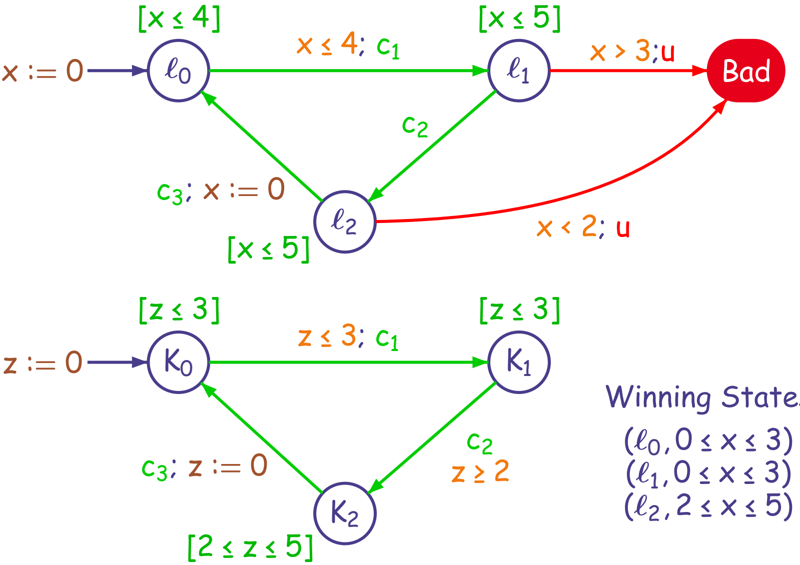
Example of Computation for Safety Games



Winning States

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

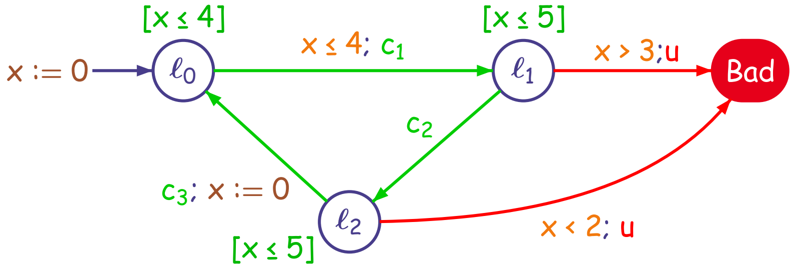
Example of Computation for Safety Games

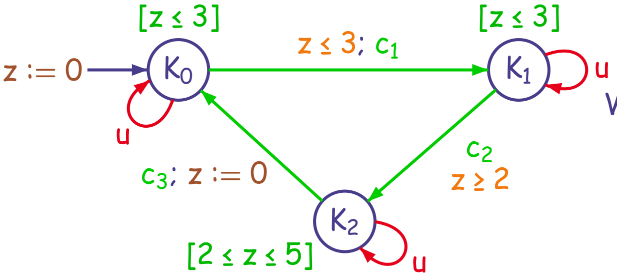


Winning States

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

Example of Computation for Safety Games

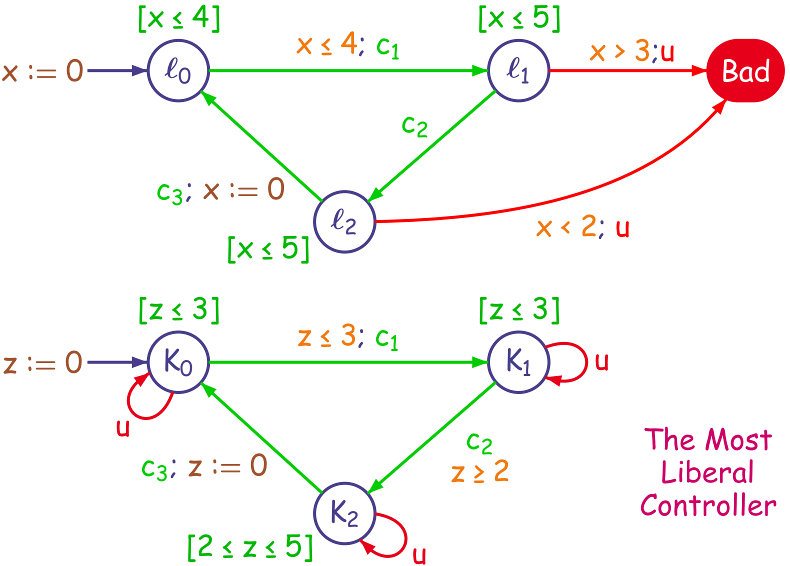




Winning States

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

Example of Computation for Safety Games



The Most Liberal Controller

Existence of Cost Independent Strategies

Let A be a RPTGA such that:

- guards of u actions are strict
- guards on c actions are large

There is an optimal cost independent strategy

Is it necessary ?

Existence of Cost Independent Strategies

Let A be a RPTGA such that:

- guards of u actions are strict
- guards on c actions are large

There is an optimal cost independent strategy

Is it necessary ?

Existence of Cost Independent Strategies

Let A be a RPTGA such that:

- guards of u actions are strict
- guards on c actions are large

There is an optimal cost independent strategy

Is it necessary ?

Existence of Cost Independent Strategies

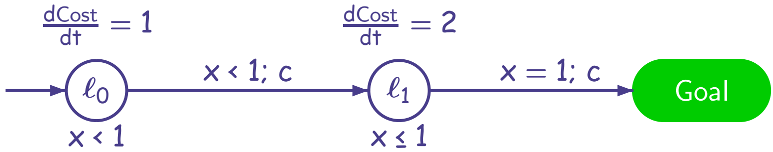
Let A be a RPTGA such that:

- guards of u actions are strict
- guards on c actions are large

There is an optimal cost independent strategy

Is it necessary ?

No Optimal Strategy

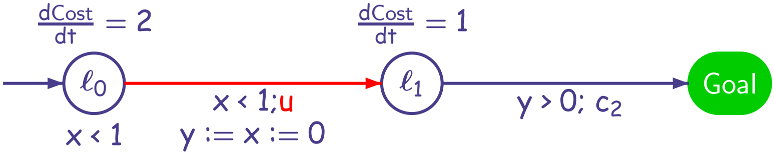


- define f_ε with $0 < \varepsilon < 1$ by:
  in ℓ_0: $f(\ell_0, x < 1 - \varepsilon) = \lambda$, $f(\ell_0, 1 - \varepsilon \leq x < 1) = c$
  in ℓ_1: $f(\ell_1, x < 1) = \lambda$, $f(\ell_1, x = 1) = c$
  $Cost(f_\varepsilon) = (1 - \varepsilon) + 2.\varepsilon = 1 + \varepsilon$ and $OptCost = 1$.
- given $\varepsilon > 0$, there is a sub-optimal strategy f_ε such that

$$|Cost((\ell_0, \vec{0}), f_\varepsilon) - OptCost((\ell_0, \vec{0}), G)| < \varepsilon$$

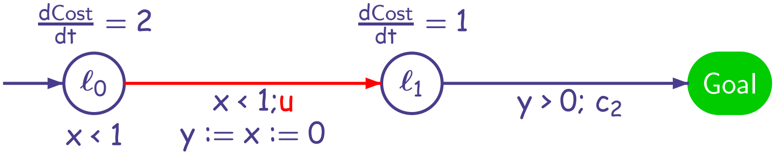
- New problem: given ε, compute such an f_ε strategy.

No Optimal Cost-Independent Strategy



- Optimal cost is 2

No Optimal Cost-Independent Strategy

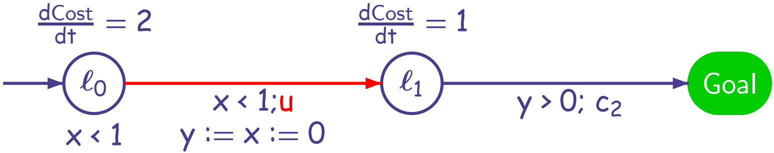


- Optimal cost is 2
- An optimal winning cost-dependent strategy f:
  $f(\ell_1, -, cost < 2) = \lambda$ and $f(\ell_1, -, cost = 2) = c_2$
  assume u taken at time $(1 - \delta_0)$:

$$Cost(f, (\ell_0, 0)) = 2 \cdot (1 - \delta_0) + \delta_1 = 2$$

because according to f we have $\delta_1 = 2 \cdot \delta_0$

No Optimal Cost-Independent Strategy



- Optimal cost is 2
- assume $\exists$ f^* cost-independent: f^* must wait in ℓ_1 at least ε
  assume u taken at time $(1 - \delta)$:

$$\mathrm{Cost}(f^*, (\ell_0, 0)) = 2 \cdot (1 - \delta) + \varepsilon$$

Take $\delta = \frac{\varepsilon}{4}$: $\mathrm{Cost}(f^*, (\ell_0, 0)) = 2 + \frac{\varepsilon}{2}$ and $\mathrm{OptCost}(f^*) = 2 + \varepsilon$

Related Work for Optimal Control

- [La Torre et al.'02]
  - Acyclic Priced Timed Game Automata
  - Recursive definition of optimal cost
  - Computation of the infimum of the optimal cost
    i.e. OptCost = 2 could mean that it is 2 or $2 + \varepsilon$
  - No strategy synthesis
- [Alur et al.'04] (ICALP'04)
  - Bounded optimality: optimal cost within k steps
  - Complexity bound: exponential in k and #states of the PTGA
  - No bound for the more general optimal problem
  - Computation of the infimum of the optimal cost
  - No strategy synthesis
- Our work [Bouyer et al.'04a]:
  - Run-based definition of optimal cost
  - We can decide whether $\exists$ an optimal strategy
  - We can effectively synthesize an optimal strategy (if one exists)
  - We can prove structural properties of optimal strategies
  - Applies to Linear Hybrid Game (Automata)

Related Work for Optimal Control

- [La Torre et al.'02] Acyclic Games, infimum, no strategy synthesis

- [Alur et al.'04] (ICALP'04)
  - Bounded optimality: optimal cost within k steps
  - Complexity bound: exponential in k and #states of the PTGA
  - No bound for the more general optimal problem
  - Computation of the infimum of the optimal cost
  - No strategy synthesis

- Our work [Bouyer et al.'04a]:
  - Run-based definition of optimal cost
  - We can decide whether ∃ an optimal strategy
  - We can effectively synthesize an optimal strategy (if one exists)
  - We can prove structural properties of optimal strategies
  - Applies to Linear Hybrid Game (Automata)

Related Work for Optimal Control

- [La Torre et al.'02] Acyclic Games, infimum, no strategy synthesis
- [Alur et al.'04] (ICALP'04)
  - Bounded optimality: optimal cost within k steps
  - Complexity bound: exponential in k and #states of the PTGA
  - No bound for the more general optimal problem
  - Computation of the infimum of the optimal cost
  - No strategy synthesis

  Bounded optimality, complexity bound, infimum, no strategy synthesis
- Our work [Bouyer et al.'04a]:
  - Run-based definition of optimal cost
  - We can decide whether ∃ an optimal strategy
  - We can effectively synthesize an optimal strategy (if one exists)
  - We can prove structural properties of optimal strategies
  - Applies to Linear Hybrid Game (Automata)

Related Work for Optimal Control

- [La Torre et al.'02] Acyclic Games, infimum, no strategy synthesis
- [Alur et al.'04] (ICALP'04)
  - Bounded optimality: optimal cost within k steps
  - Complexity bound: exponential in k and #states of the PTGA
  - No bound for the more general optimal problem
  - Computation of the infimum of the optimal cost
  - No strategy synthesis

  Bounded optimality, complexity bound, infimum, no strategy synthesis
- Our work [Bouyer et al.'04a]:
  - Run-based definition of optimal cost
  - We can decide whether ∃ an optimal strategy
  - We can effectively synthesize an optimal strategy (if one exists)
  - We can prove structural properties of optimal strategies
  - Applies to Linear Hybrid Game (Automata)