Robust Controller Synthesis in Timed Automata

Pierre-Alain Reynier

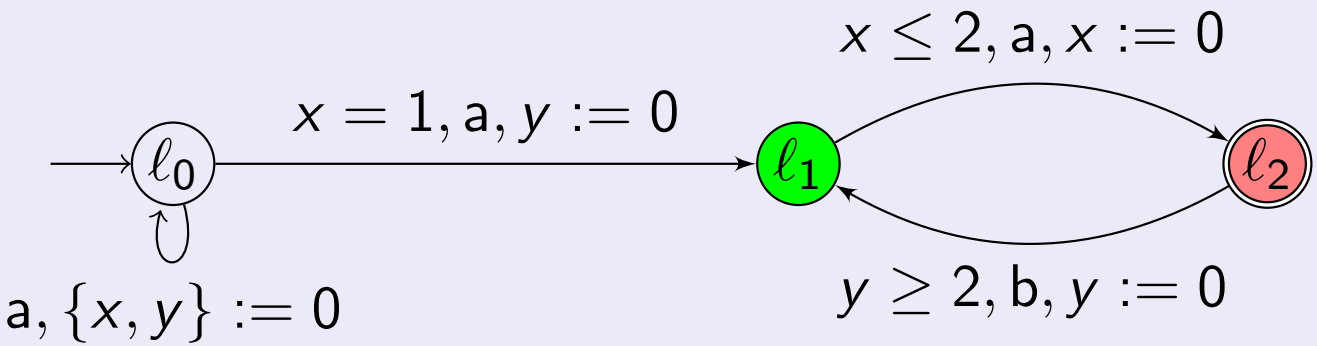
LIF, Aix-Marseille University & CNRS

Joint with Youssouf Oualhadj (UMons) and Ocan Sankur (ULB)
Acknowledgment to Ocan for slides

Overview

Timed Automata

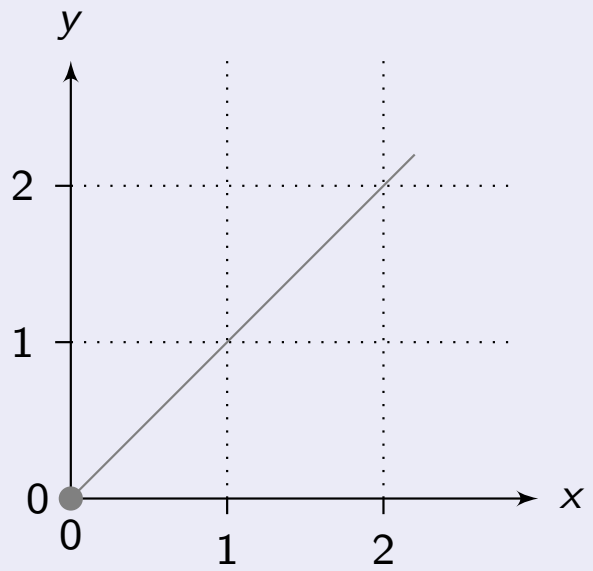
A timed automaton (TA)



Configurations, Runs

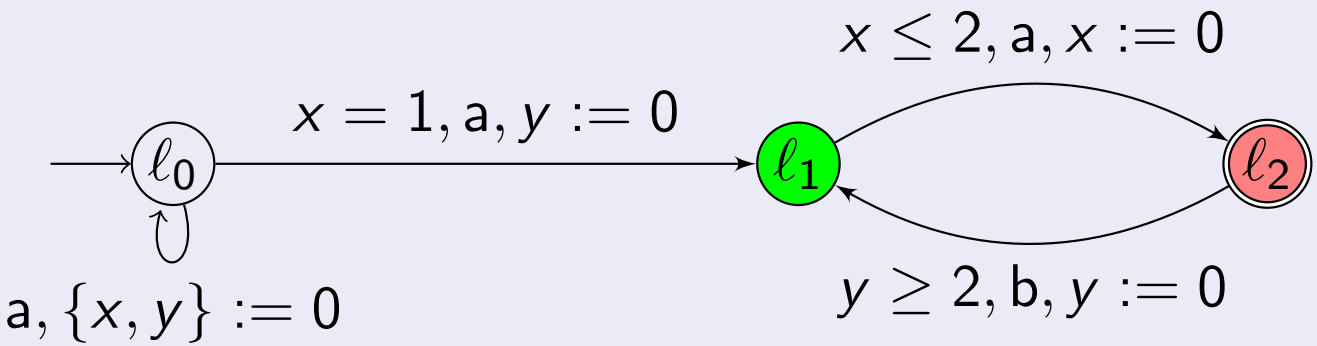
Configurations:

(ℓ, v)



Timed Automata

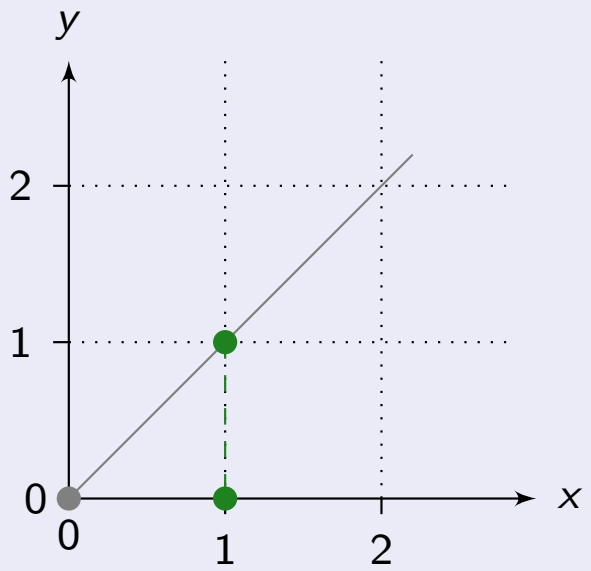
A timed automaton (TA)



Configurations, Runs

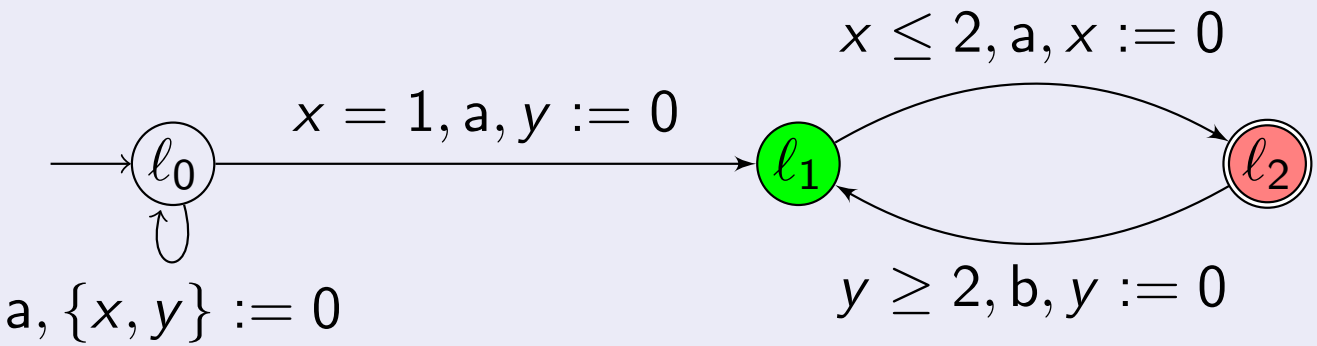
Configurations:

(ℓ, v)



Timed Automata

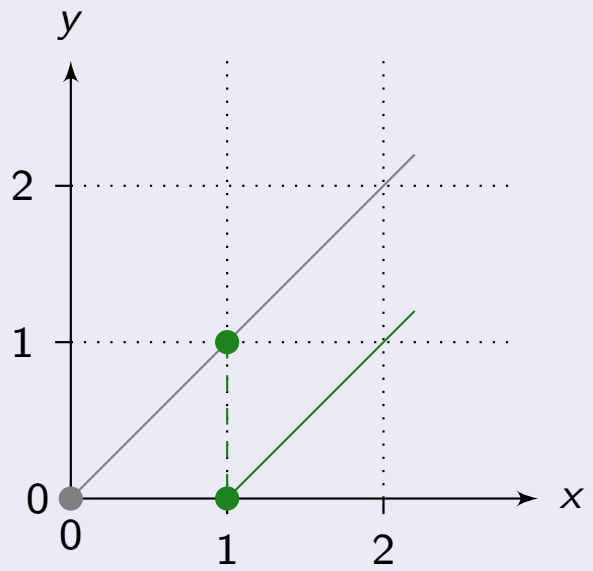
A timed automaton (TA)



Configurations, Runs

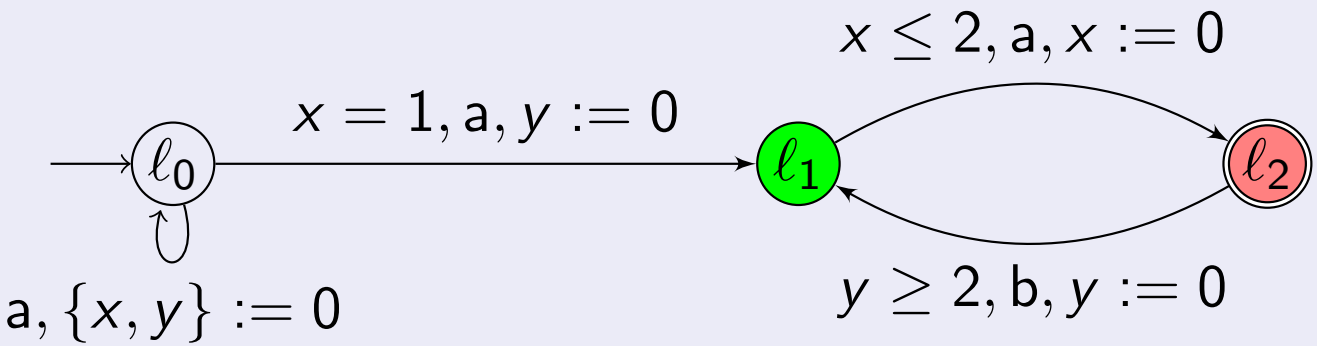
Configurations:

(ℓ, v)



Timed Automata

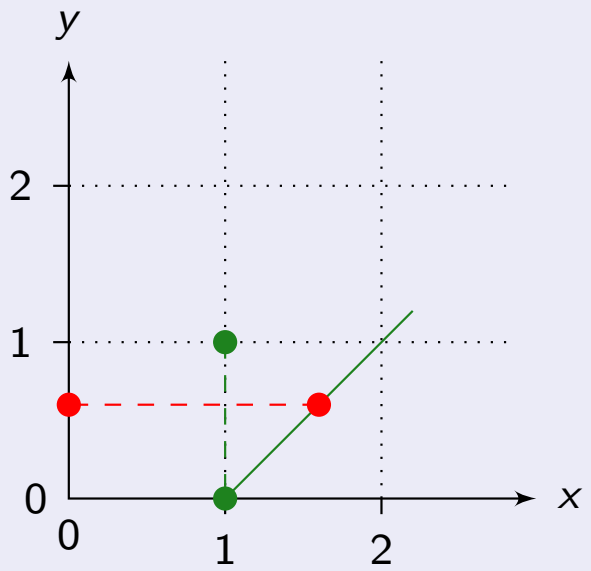
A timed automaton (TA)



Configurations, Runs

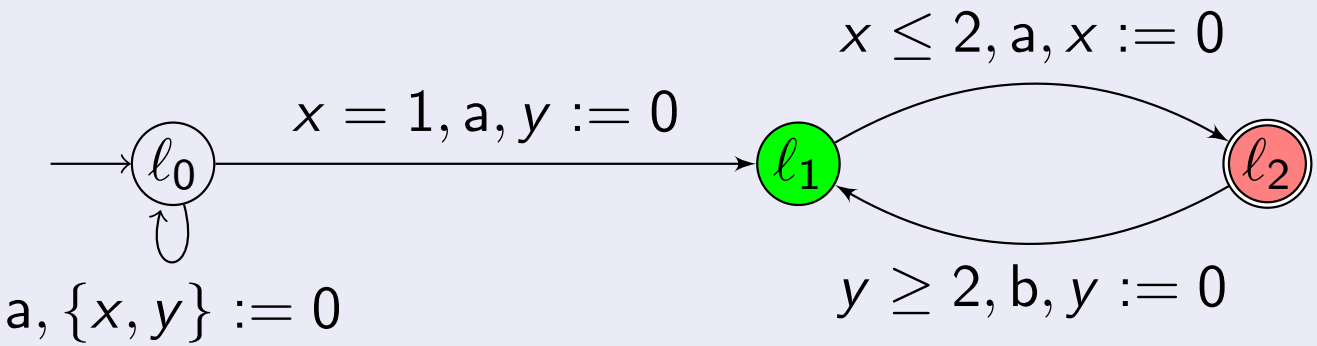
Configurations:

(ℓ, v)



Timed Automata

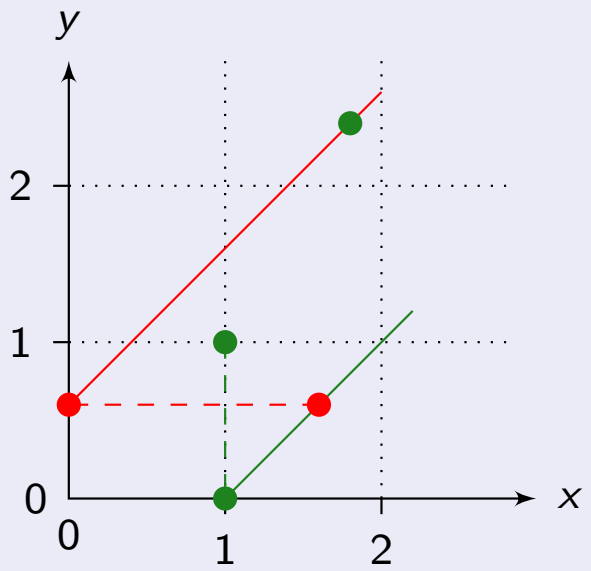
A timed automaton (TA)



Configurations, Runs

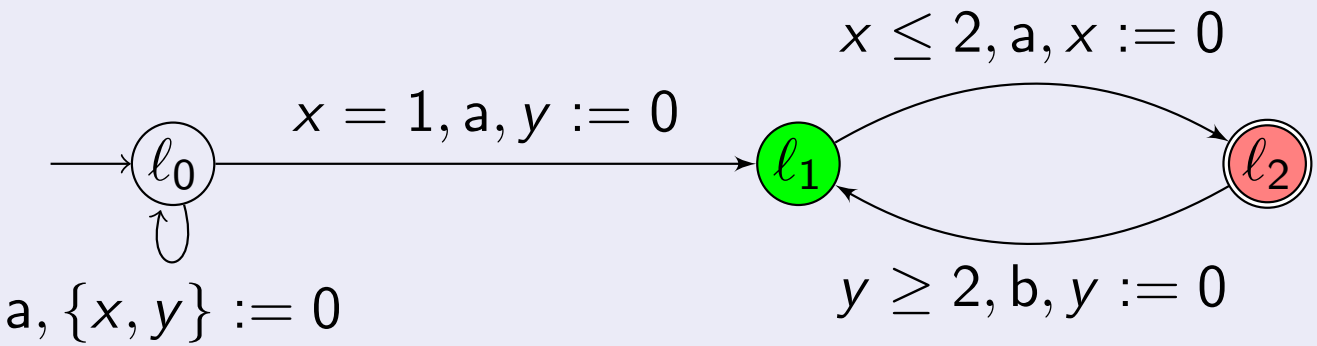
Configurations:

(ℓ, v)



Timed Automata

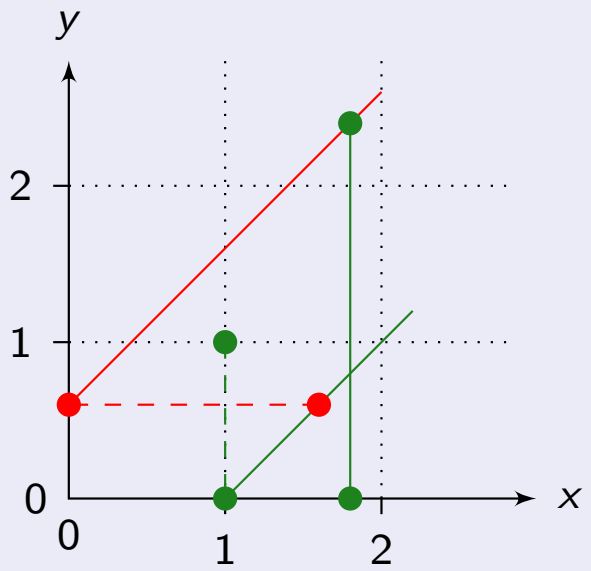
A timed automaton (TA)



Configurations, Runs

Configurations:

(ℓ, v)



Playing in Timed Automata

We consider a two-player turn-based game defined as follows:

- Controller suggests actions and delays
- Perturbator resolves non-determinism of actions

Playing in Timed Automata

We consider a two-player turn-based game defined as follows:

- Controller suggests actions and delays
- Perturbator resolves non-determinism of actions

➜ Notions of strategies for Controller and Perturbator, denoted σ and τ

Given two strategies σ and τ, they give rise to a single run in the TA.
This run is winning for Controller iff it verifies the Büchi condition.

Note: in deterministic timed automata, this amounts to a one-player game.

Playing in Timed Automata

We consider a two-player turn-based game defined as follows:

- Controller suggests actions and delays
- Perturbator resolves non-determinism of actions

➜ Notions of strategies for Controller and Perturbator, denoted σ and τ

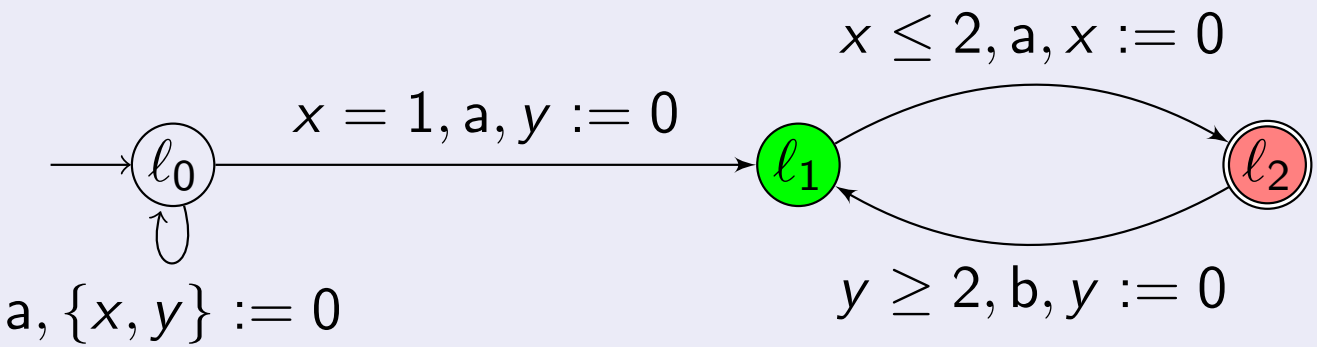
Given two strategies σ and τ, they give rise to a single run in the TA.
This run is winning for Controller iff it verifies the Büchi condition.

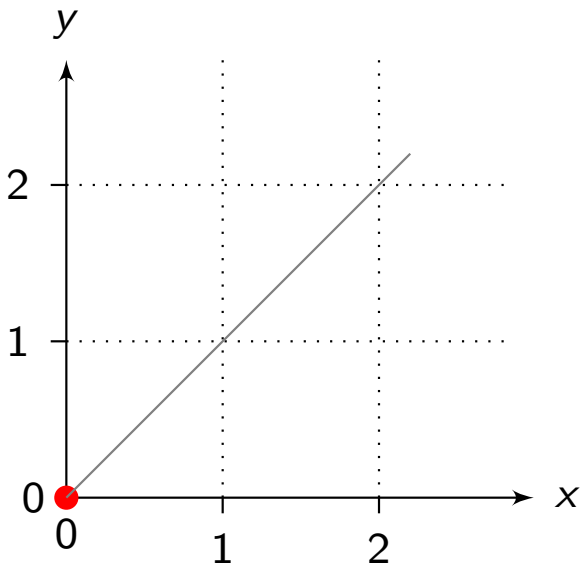
Note: in deterministic timed automata, this amounts to a one-player game.

Controller Synthesis

Given a timed automaton and a Büchi condition, decide whether there exists a **strategy** for Controller to win the game (and compute one).

Back to the example



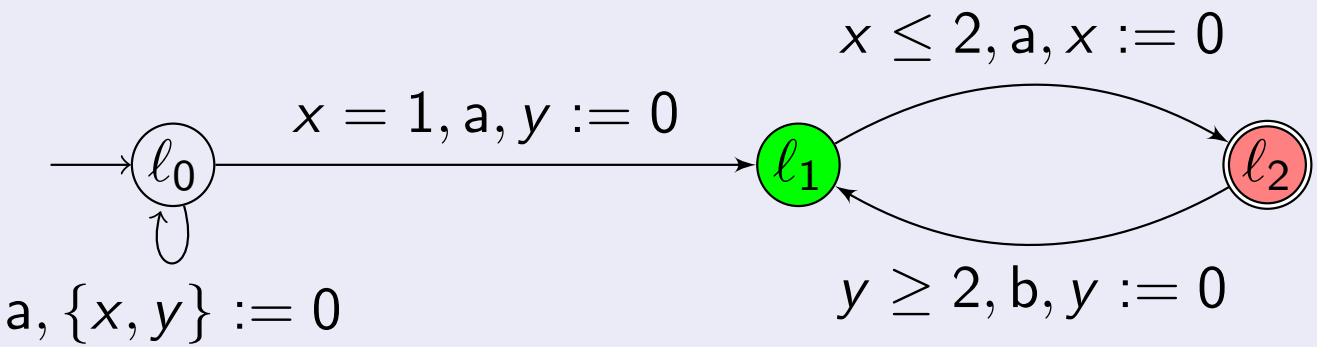


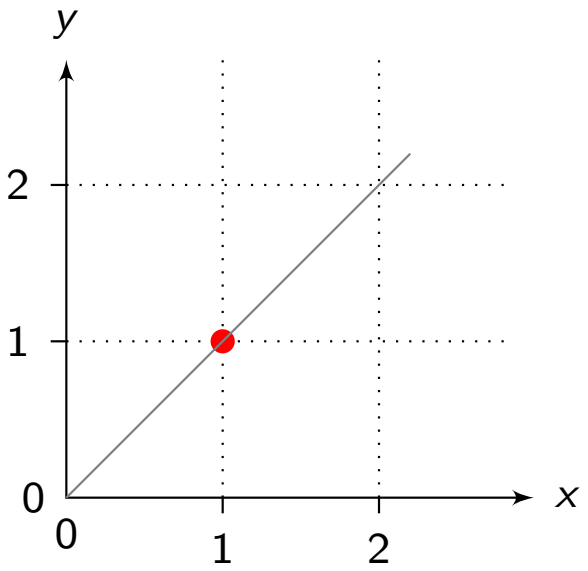
State $(\ell_0, 0, 0)$: fire action a after delay 1

States $(\ell_1, x, 0)$:
fire action a after delay $\frac{2-x}{2}$
(half-way to the max. delay)

States $(\ell_2, 0, y)$:
fire action b after delay $2 - y$

Back to the example



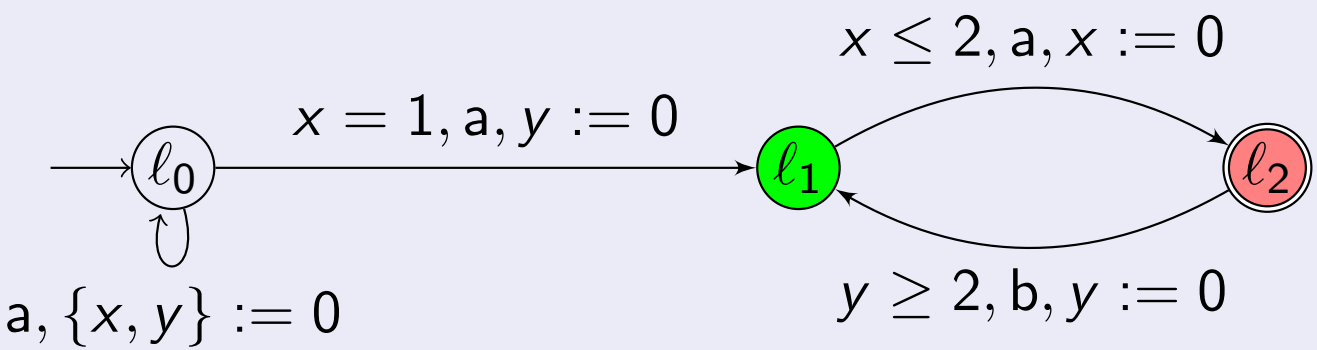


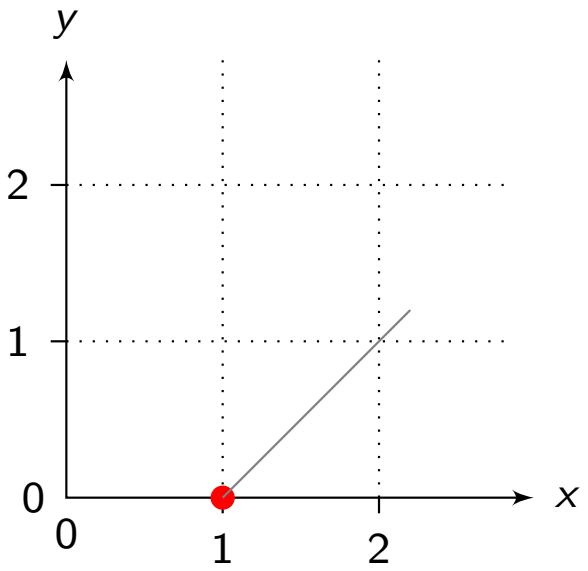
State $(\ell_0, 0, 0)$: fire action a after delay 1

States $(\ell_1, x, 0)$:
fire action a after delay $\frac{2-x}{2}$
(half-way to the max. delay)

States $(\ell_2, 0, y)$:
fire action b after delay $2 - y$

Back to the example



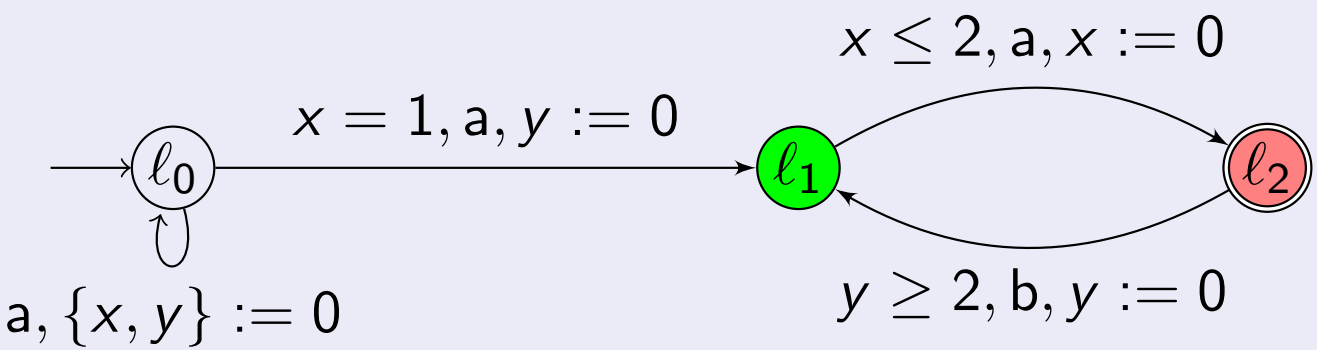


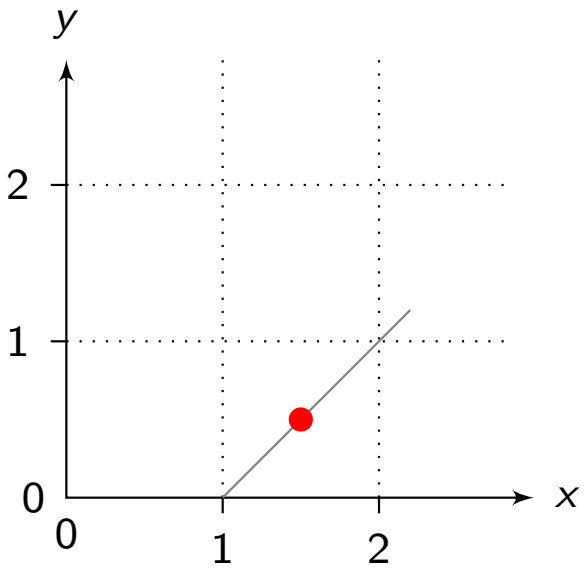
State $(\ell_0, 0, 0)$: fire action a after delay 1

States $(\ell_1, x, 0)$:
fire action a after delay $\frac{2-x}{2}$
(half-way to the max. delay)

States $(\ell_2, 0, y)$:
fire action b after delay $2 - y$

Back to the example



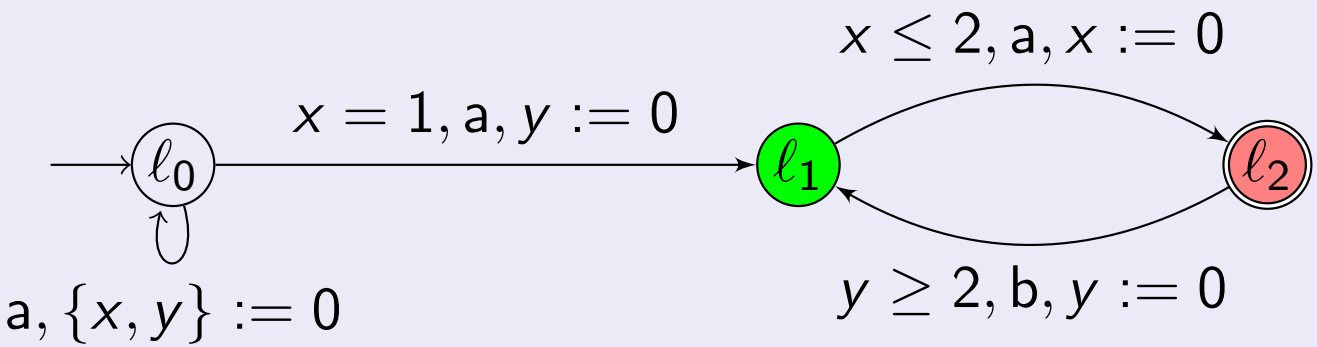


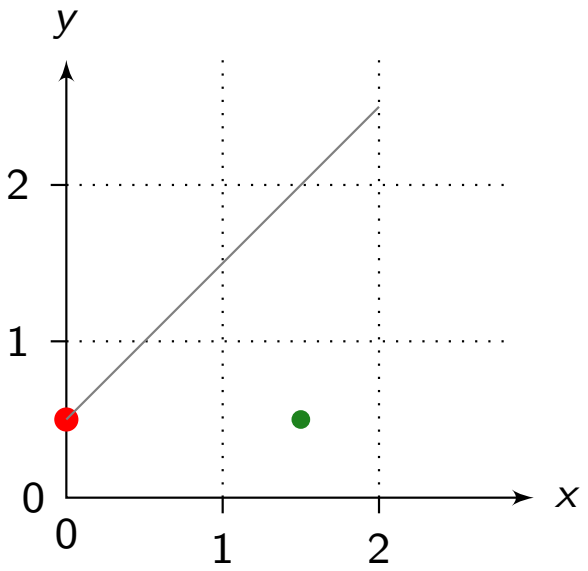
State $(\ell_0, 0, 0)$: fire action a after delay 1

States $(\ell_1, x, 0)$:
fire action a after delay $\frac{2-x}{2}$
(half-way to the max. delay)

States $(\ell_2, 0, y)$:
fire action b after delay $2 - y$

Back to the example



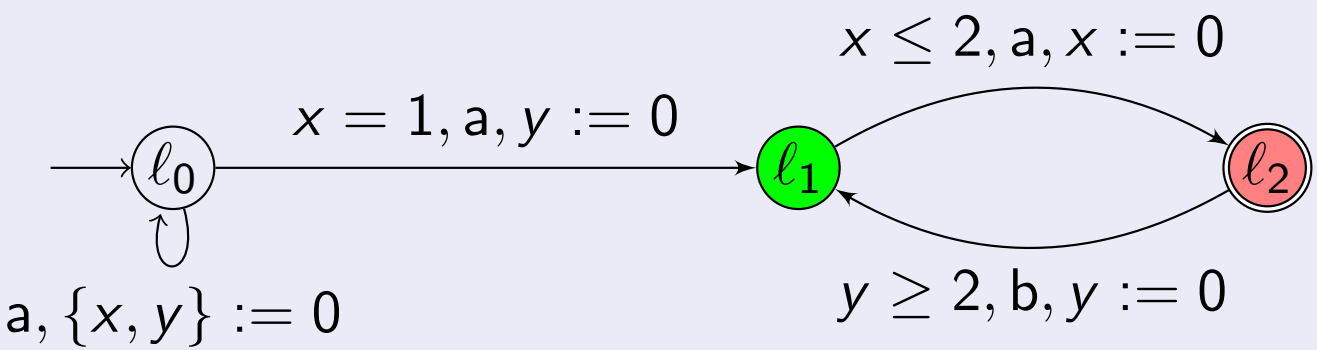


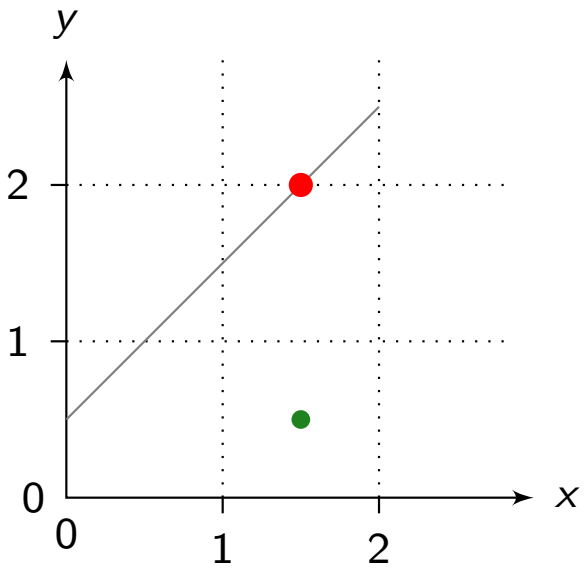
State $(\ell_0, 0, 0)$: fire action a after delay 1

States $(\ell_1, x, 0)$:
fire action a after delay $\frac{2-x}{2}$
(half-way to the max. delay)

States $(\ell_2, 0, y)$:
fire action b after delay $2 - y$

Back to the example



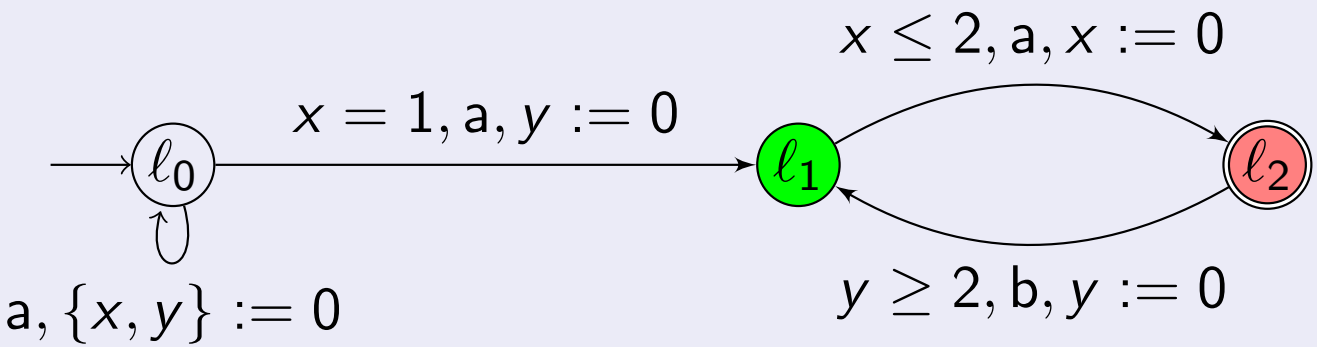


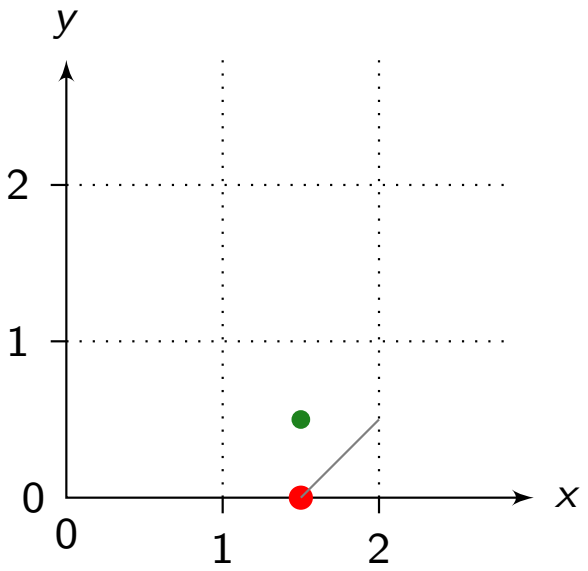
State $(\ell_0, 0, 0)$: fire action a after delay 1

States $(\ell_1, x, 0)$:
fire action a after delay $\frac{2-x}{2}$
(half-way to the max. delay)

States $(\ell_2, 0, y)$:
fire action b after delay $2 - y$

Back to the example



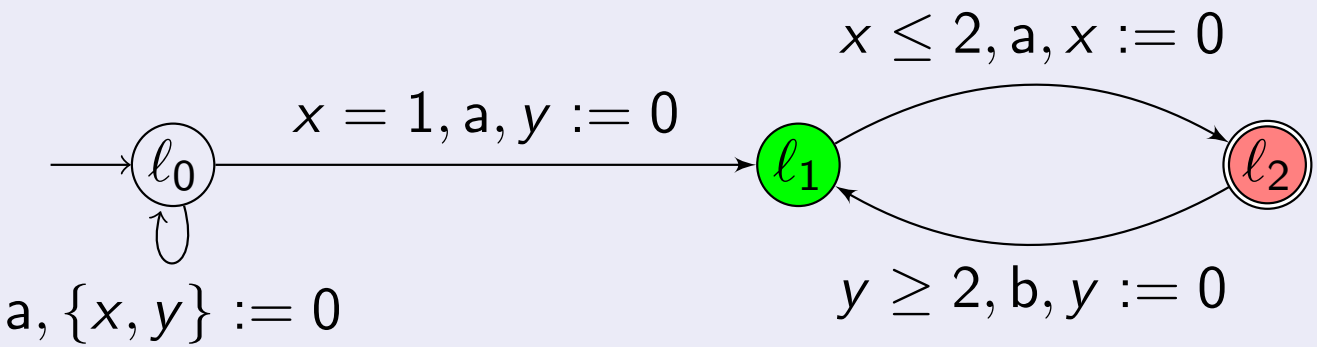


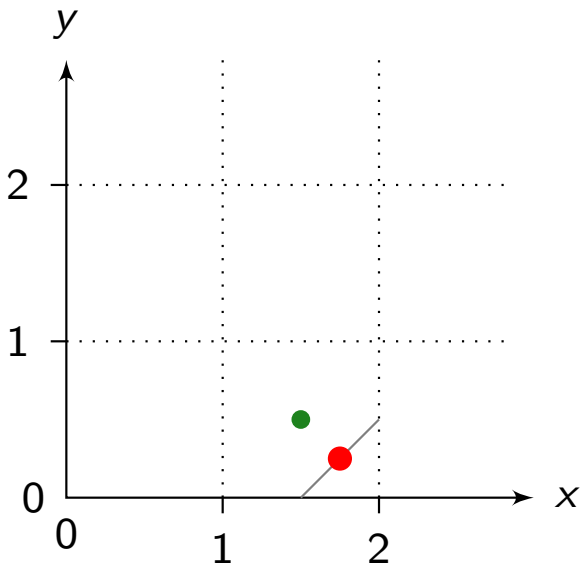
State $(\ell_0, 0, 0)$: fire action a after delay 1

States $(\ell_1, x, 0)$:
fire action a after delay $\frac{2-x}{2}$
(half-way to the max. delay)

States $(\ell_2, 0, y)$:
fire action b after delay $2 - y$

Back to the example



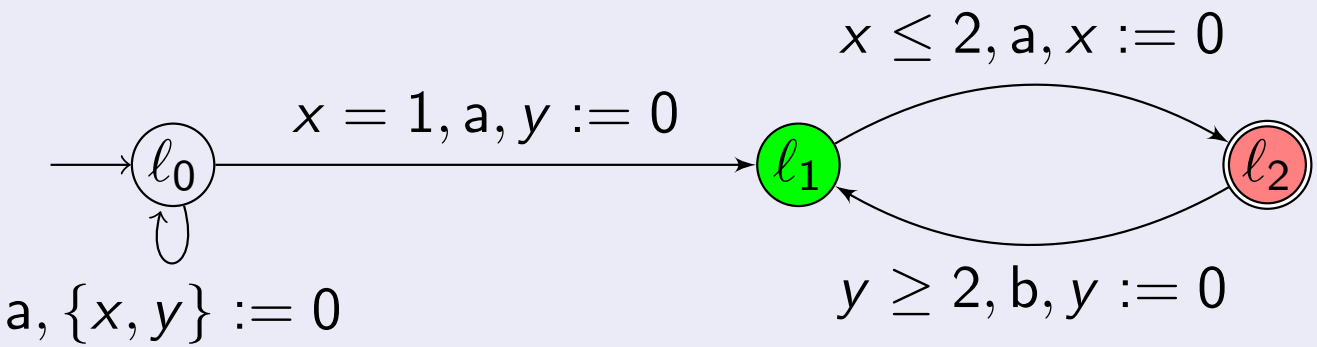


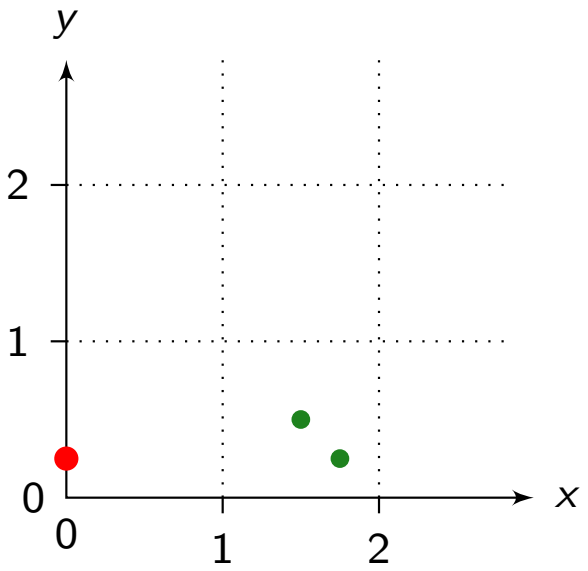
State $(\ell_0, 0, 0)$: fire action a after delay 1

States $(\ell_1, x, 0)$:
fire action a after delay $\frac{2-x}{2}$
(half-way to the max. delay)

States $(\ell_2, 0, y)$:
fire action b after delay $2 - y$

Back to the example



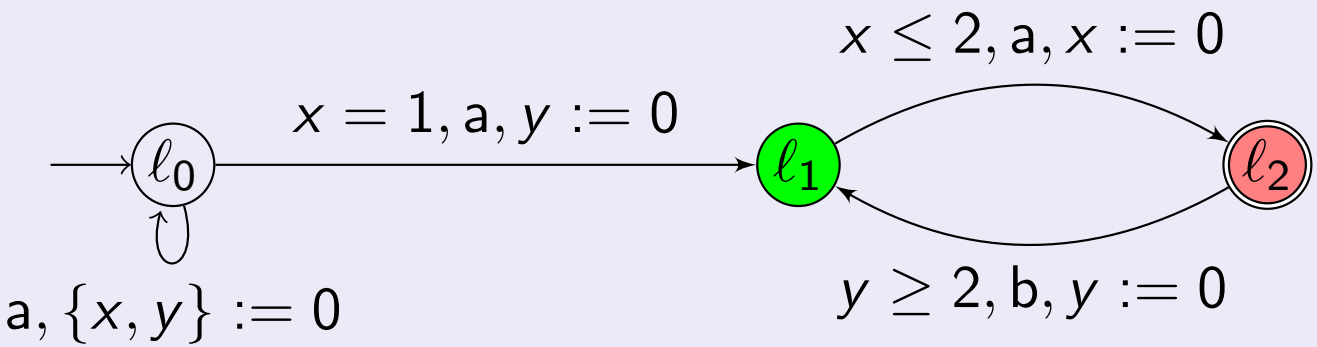


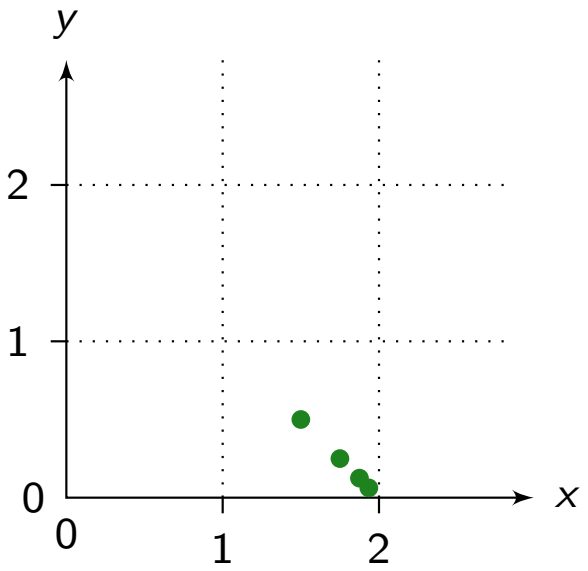
State $(\ell_0, 0, 0)$: fire action a after delay 1

States $(\ell_1, x, 0)$:
fire action a after delay $\frac{2-x}{2}$
(half-way to the max. delay)

States $(\ell_2, 0, y)$:
fire action b after delay $2 - y$

Back to the example





State $(\ell_0, 0, 0)$: fire action a after delay 1

States $(\ell_1, x, 0)$:
fire action a after delay $\frac{2-x}{2}$
(half-way to the max. delay)

States $(\ell_2, 0, y)$:
fire action b after delay $2 - y$

Controller Synthesis

Note: For deterministic TA, the problem amounts to find an accepting run.

Theorem

Controller synthesis for Büchi objectives in deterministic timed automata is PSPACE-complete.

The general case induces a complexity blow-up:

Theorem

Controller synthesis for Büchi objectives in non-deterministic timed automata is EXPTIME-complete.

➜ This setting is well-known with symbolic algorithms, tools (UppAal-TiGa) and several case studies.

Controller Synthesis

Note: For deterministic TA, the problem amounts to find an accepting run.

Theorem

Controller synthesis for Büchi objectives in deterministic timed automata is PSPACE-complete.

The general case induces a complexity blow-up:

Theorem

Controller synthesis for Büchi objectives in non-deterministic timed automata is EXPTIME-complete.

➜ This setting is well-known with symbolic algorithms, tools (UppAal-TiGa) and several case studies.

Is that the end of the story?

Robustness Issues

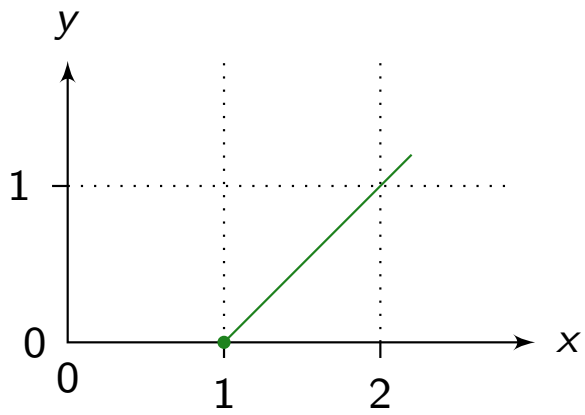
1. Exact timings cannot be ensured in real-time systems.
So the strategy $\left(x \mapsto \text{delay } \frac{2-x}{2}\right)$ cannot be implemented exactly.

Is any strategy valid under imprecise delays?

Robustness Issues

1. Exact timings cannot be ensured in real-time systems.
So the strategy $\left(x \mapsto \text{delay } \frac{2-x}{2}\right)$ cannot be implemented exactly.

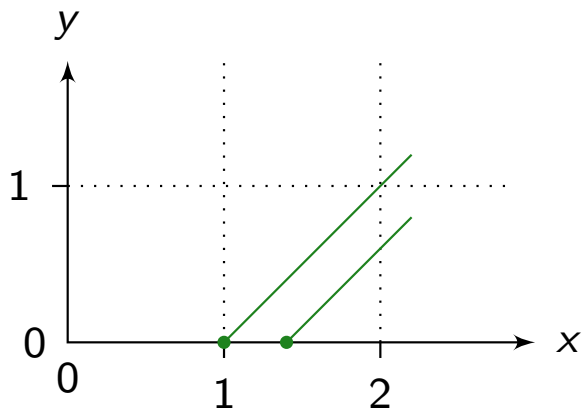
In our example, one can show that x is increasing during consecutive visits to ℓ_1, and the guard is $x \leq 2$.



Robustness Issues

1. Exact timings cannot be ensured in real-time systems.
So the strategy $\left(x \mapsto \text{delay } \frac{2-x}{2}\right)$ cannot be implemented exactly.

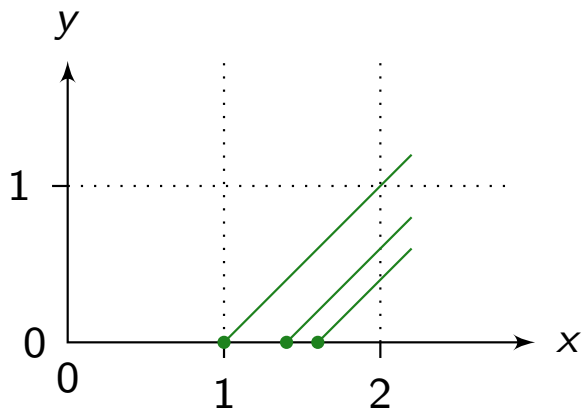
In our example, one can show that x is increasing during consecutive visits to ℓ_1, and the guard is $x \leq 2$.



Robustness Issues

1. Exact timings cannot be ensured in real-time systems.
So the strategy $\left(x \mapsto \text{delay } \frac{2-x}{2}\right)$ cannot be implemented exactly.

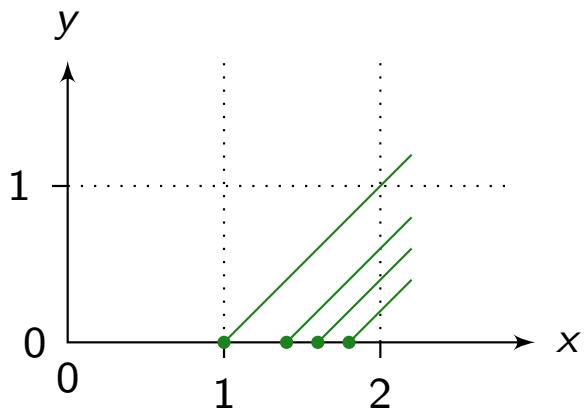
In our example, one can show that x is increasing during consecutive visits to ℓ_1, and the guard is $x \leq 2$.



Robustness Issues

1. Exact timings cannot be ensured in real-time systems.
So the strategy $\left(x \mapsto \text{delay } \frac{2-x}{2}\right)$ cannot be implemented exactly.

In our example, one can show that x is increasing during consecutive visits to ℓ_1, and the guard is $x \leq 2$.



Robustness Issues

1. Exact timings cannot be ensured in real-time systems.
So the strategy $\left(x \mapsto \text{delay } \frac{2-x}{2}\right)$ cannot be implemented exactly.

2. Strategies may require arbitrary precision.
Required delays **converge** here.

When x is closed to 2, no additional delay is supported.
Run is theoretically infinite, but it is actually **blocking**.

Robustness Issues

1. Exact timings cannot be ensured in real-time systems.
So the strategy $\left(x \mapsto \text{delay } \frac{2-x}{2}\right)$ cannot be implemented exactly.

2. Strategies may require arbitrary precision.
Required delays **converge** here.

When x is closed to 2, no additional delay is supported.
Run is theoretically infinite, but it is actually **blocking**.

Goal: Revisit Controller Synthesis for timed automata
and suggest a robust alternative: only accept realizable strategies, avoid convergent ones.

Perturbation Game Semantics

Game Semantics: Controller vs Perturbator

Given A and $\delta > 0$, define $\mathcal{G}_\delta(A)$ as a game as follows. At any state (ℓ, ν),

1. **Controller** chooses a delay $d \geq \delta$ and an action a s.t. for every edge $\ell \xrightarrow{g,R} \ell'$, if $\nu + d \models g$ then $\nu + d + \epsilon \models g$ for all $\epsilon \in [-\delta, \delta]$.

Perturbation Game Semantics

Game Semantics: Controller vs Perturbator

Given A and $\delta > 0$, define $\mathcal{G}_\delta(A)$ as a game as follows. At any state (ℓ, ν),

1. **Controller** chooses a delay $d \geq \delta$ and an action a s.t. for every edge $\ell \xrightarrow{g,R} \ell'$, if $\nu + d \models g$ then $\nu + d + \epsilon \models g$ for all $\epsilon \in [-\delta, \delta]$.
2. **Perturbator** resolves non-determinism and chooses $\epsilon \in [-\delta, +\delta]$,

Perturbation Game Semantics

Game Semantics: Controller vs Perturbator

Given A and $\delta > 0$, define $\mathcal{G}_\delta(A)$ as a game as follows. At any state (ℓ, ν),

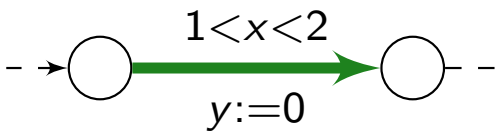
1. **Controller** chooses a delay $d \geq \delta$ and an action a s.t. for every edge $\ell \xrightarrow{g,R} \ell'$, if $\nu + d \models g$ then $\nu + d + \epsilon \models g$ for all $\epsilon \in [-\delta, \delta]$.
2. **Perturbator** resolves non-determinism and chooses $\epsilon \in [-\delta, +\delta]$,
3. New state is $(\ell', (\nu + d + \epsilon)[R \leftarrow 0])$.

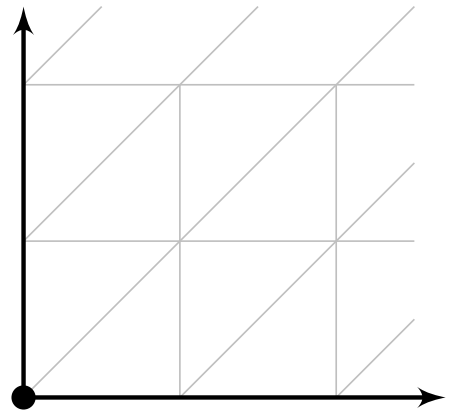
Perturbation Game Semantics

Game Semantics: Controller vs Perturbator

Given A and $\delta > 0$, define $\mathcal{G}_\delta(A)$ as a game as follows. At any state (ℓ, ν),

1. **Controller** chooses a delay $d \geq \delta$ and an action a s.t. for every edge $\ell \xrightarrow{g,R} \ell'$, if $\nu + d \models g$ then $\nu + d + \epsilon \models g$ for all $\epsilon \in [-\delta, \delta]$.
2. **Perturbator** resolves non-determinism and chooses $\epsilon \in [-\delta, +\delta]$,
3. New state is $(\ell', (\nu + d + \epsilon)[R \leftarrow 0])$.



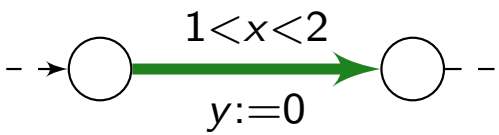


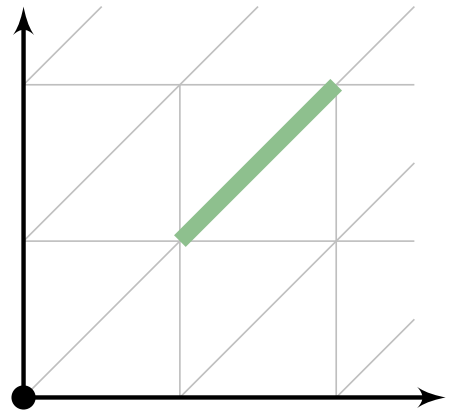
Perturbation Game Semantics

Game Semantics: Controller vs Perturbator

Given A and $\delta > 0$, define $\mathcal{G}_\delta(A)$ as a game as follows. At any state (ℓ, ν),

1. **Controller** chooses a delay $d \geq \delta$ and an action a s.t. for every edge $\ell \xrightarrow{g,R} \ell'$, if $\nu + d \models g$ then $\nu + d + \epsilon \models g$ for all $\epsilon \in [-\delta, \delta]$.
2. **Perturbator** resolves non-determinism and chooses $\epsilon \in [-\delta, +\delta]$,
3. New state is $(\ell', (\nu + d + \epsilon)[R \leftarrow 0])$.



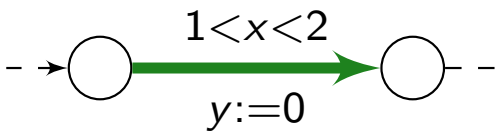


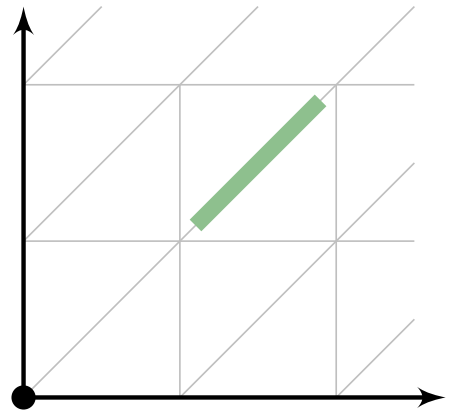
Perturbation Game Semantics

Game Semantics: Controller vs Perturbator

Given A and $\delta > 0$, define $\mathcal{G}_\delta(A)$ as a game as follows. At any state (ℓ, ν),

1. **Controller** chooses a delay $d \geq \delta$ and an action a s.t. for every edge $\ell \xrightarrow{g,R} \ell'$, if $\nu + d \models g$ then $\nu + d + \epsilon \models g$ for all $\epsilon \in [-\delta, \delta]$.
2. **Perturbator** resolves non-determinism and chooses $\epsilon \in [-\delta, +\delta]$,
3. New state is $(\ell', (\nu + d + \epsilon)[R \leftarrow 0])$.



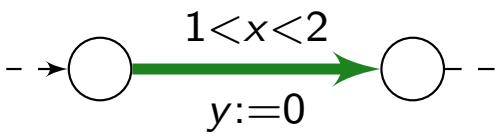


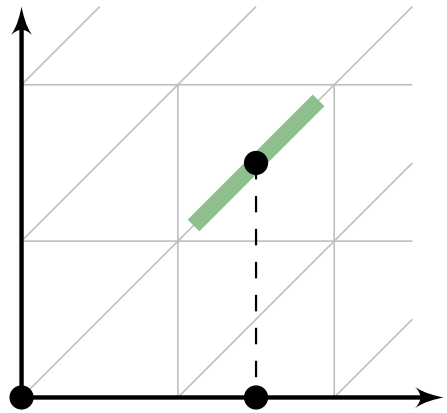
Perturbation Game Semantics

Game Semantics: Controller vs Perturbator

Given A and $\delta > 0$, define $\mathcal{G}_\delta(A)$ as a game as follows. At any state (ℓ, ν),

1. **Controller** chooses a delay $d \geq \delta$ and an action a s.t. for every edge $\ell \xrightarrow{g,R} \ell'$, if $\nu + d \models g$ then $\nu + d + \epsilon \models g$ for all $\epsilon \in [-\delta, \delta]$.
2. **Perturbator** resolves non-determinism and chooses $\epsilon \in [-\delta, +\delta]$,
3. New state is $(\ell', (\nu + d + \epsilon)[R \leftarrow 0])$.



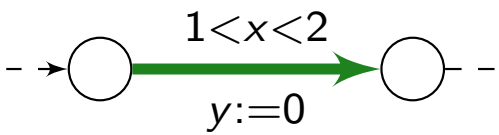


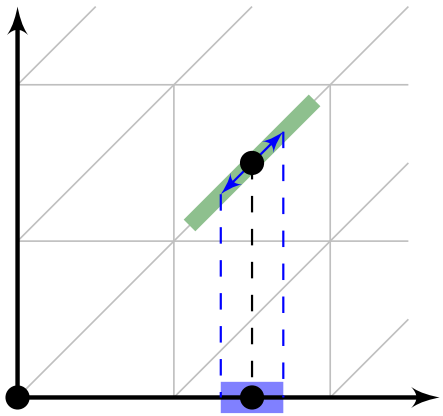
Perturbation Game Semantics

Game Semantics: Controller vs Perturbator

Given A and $\delta > 0$, define $\mathcal{G}_\delta(A)$ as a game as follows. At any state (ℓ, ν),

1. **Controller** chooses a delay $d \geq \delta$ and an action a s.t. for every edge $\ell \xrightarrow{g,R} \ell'$, if $\nu + d \models g$ then $\nu + d + \epsilon \models g$ for all $\epsilon \in [-\delta, \delta]$.
2. **Perturbator** resolves non-determinism and chooses $\epsilon \in [-\delta, +\delta]$,
3. New state is $(\ell', (\nu + d + \epsilon)[R \leftarrow 0])$.





Perturbation Game Semantics

Game Semantics: Controller vs Perturbator

Given A and $\delta > 0$, define $\mathcal{G}_\delta(A)$ as a game as follows. At any state (ℓ, ν),

1. **Controller** chooses a delay $d \geq \delta$ and an action a s.t. for every edge $\ell \xrightarrow{g,R} \ell'$, if $\nu + d \models g$ then $\nu + d + \epsilon \models g$ for all $\epsilon \in [-\delta, \delta]$.
2. **Perturbator** resolves non-determinism and chooses $\epsilon \in [-\delta, +\delta]$,
3. New state is $(\ell', (\nu + d + \epsilon)[R \leftarrow 0])$.

Robust Controller Synthesis

Given a timed automaton A, and a Büchi condition ϕ, decide whether there exists $\delta > 0$ s.t. Controller wins in $\mathcal{G}_\delta(A)$ for ϕ.

A strategy for δ is valid for all $0 < \delta' < \delta$.

Existing results

Model Checking: many results

Controller Synthesis

- Chatterjee, Henzinger, Prabhu 2008: for **fixed** $\delta > 0$.
- Bouyer, Markey, Sankur 2012: different semantics, reachability objectives
- Sankur, Bouyer, Markey, Reynier 2013: case of deterministic TA

In this paper:

- General case
- Stochastic Perturbator

Overview

Region Graph [Alur&Dill 1990]

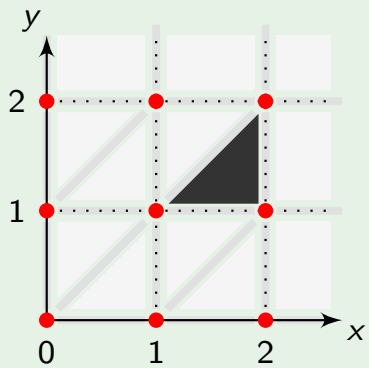
Region Graph $\mathcal{R}(\mathcal{A})$

Finite partitioning of the state-space compatible with guards, resets and time-elapsing.

Time-abstract bisimulation

Exponential size

Example



Two types of transitions: (abstract) delays, and discrete actions.

Region Graph [Alur&Dill 1990]

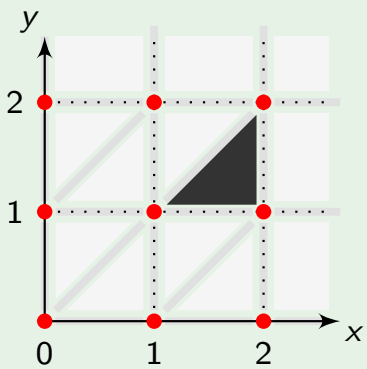
Region Graph $\mathcal{R}(\mathcal{A})$

Finite partitioning of the state-space compatible with guards, resets and time-elapsing.

Time-abstract bisimulation

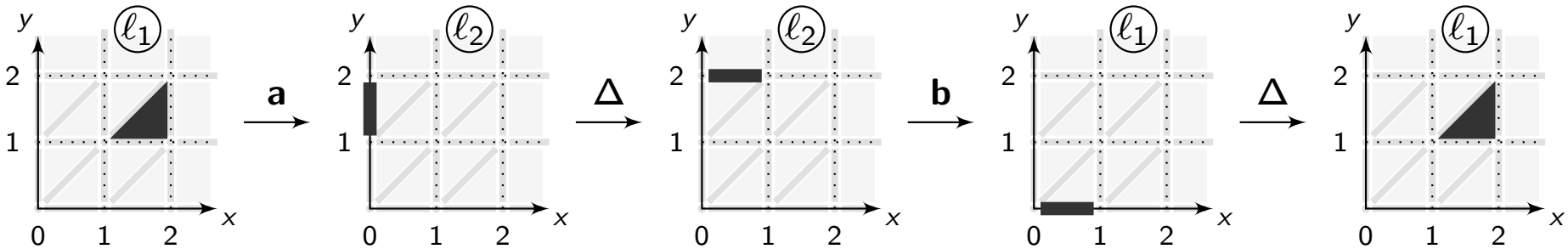
Exponential size

Example



Two types of transitions: (abstract) delays, and discrete actions.

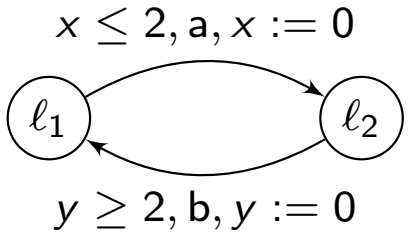
➜ Lack of precision: convergence phenomena are not captured.

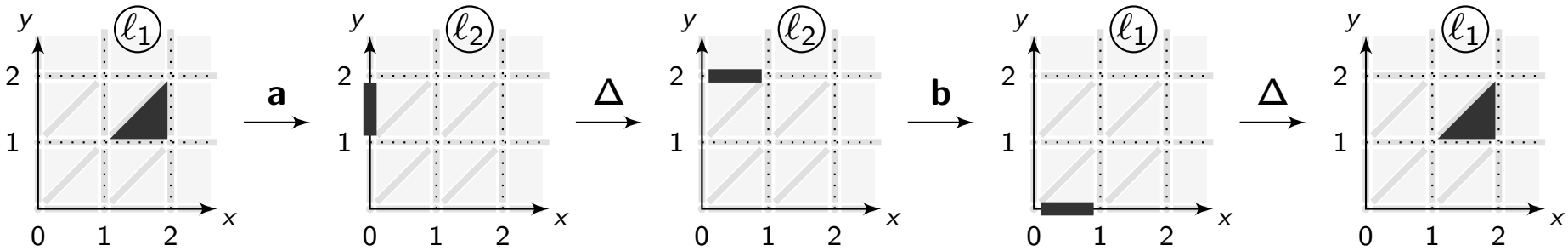


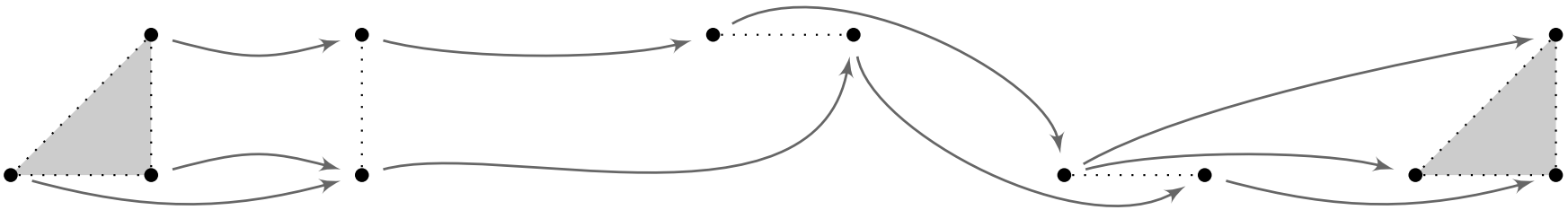
Orbit Graph [Puri 2000]

Orbit Graph

Stores the reachability relation between vertices of the regions



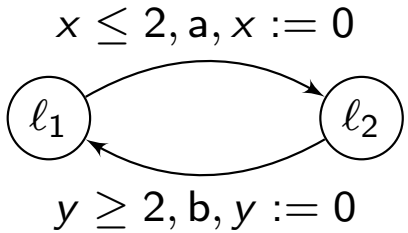


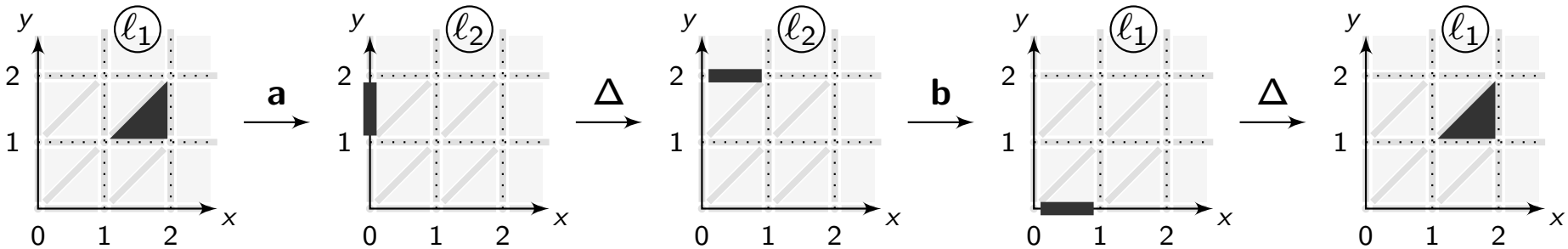


Orbit Graph [Puri 2000]

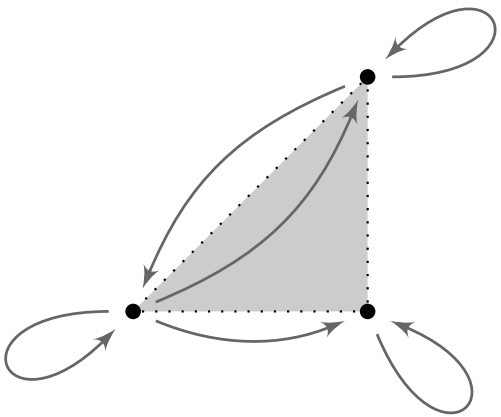
Orbit Graph

Stores the reachability relation between vertices of the regions





Case of cycles: Folded Orbit Graph (FOG)

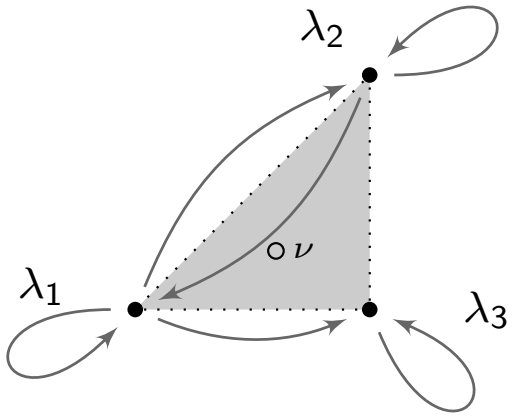


A cycle is:

- forgetful iff the FOG is strongly connected
- aperiodic iff every power is forgetful

[Asarin&Basset 2011], [Stainer 2012]

Reachability with Orbit Graphs



For any valuation ν, write $\nu = \vec{\lambda}\vec{v}$, a convex combination of the vertices.

Theorem [Puri 2000], [Asarin & Basset 2011]

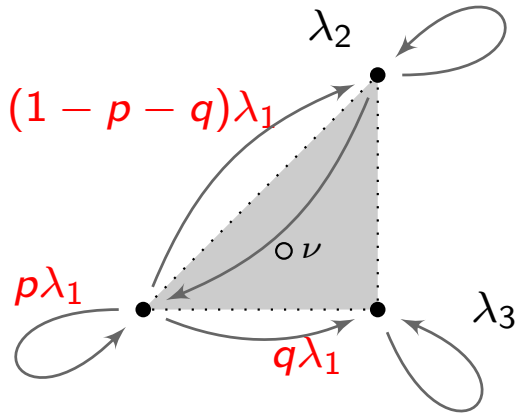
Given a path, and valuation $\vec{\lambda} \cdot \vec{v}$,

$$\vec{\lambda} \cdot \vec{v} \to \vec{\lambda'}\vec{v}$$

$$\Leftrightarrow$$

$\vec{\lambda'}$ is computed by distributing each λ_ν to its successors following a probability distribution.

Reachability with Orbit Graphs



For any valuation ν, write $\nu = \vec{\lambda}\vec{v}$, a convex combination of the vertices.

Theorem [Puri 2000], [Asarin & Basset 2011]

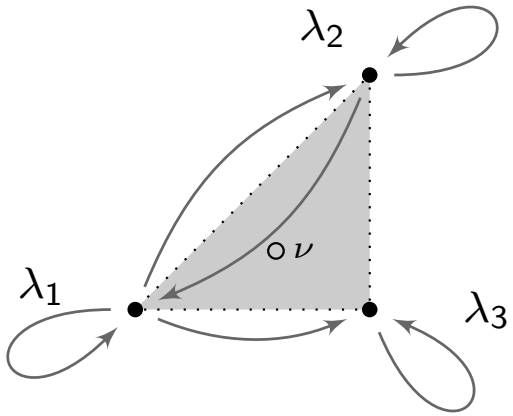
Given a path, and valuation $\vec{\lambda} \cdot \vec{v}$,

$$\vec{\lambda} \cdot \vec{v} \to \vec{\lambda'}\vec{v}$$

$$\Leftrightarrow$$

$\vec{\lambda'}$ is computed by distributing each λ_ν to its successors following a probability distribution.

Reachability with Orbit Graphs



For any valuation ν, write $\nu = \vec{\lambda}\vec{v}$, a convex combination of the vertices.

Theorem [Puri 2000], [Asarin & Basset 2011]

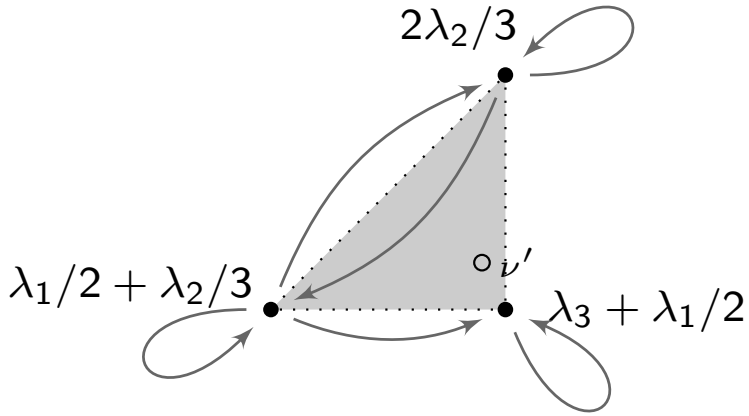
Given a path, and valuation $\vec{\lambda} \cdot \vec{v}$,

$$\vec{\lambda} \cdot \vec{v} \to \vec{\lambda'}\vec{v}$$

$$\Leftrightarrow$$

$\vec{\lambda'}$ is computed by distributing each λ_ν to its successors following a probability distribution.

Reachability with Orbit Graphs



For any valuation ν, write $\nu = \vec{\lambda}\vec{v}$, a convex combination of the vertices.

Theorem [Puri 2000], [Asarin & Basset 2011]

Given a path, and valuation $\vec{\lambda} \cdot \vec{v}$,

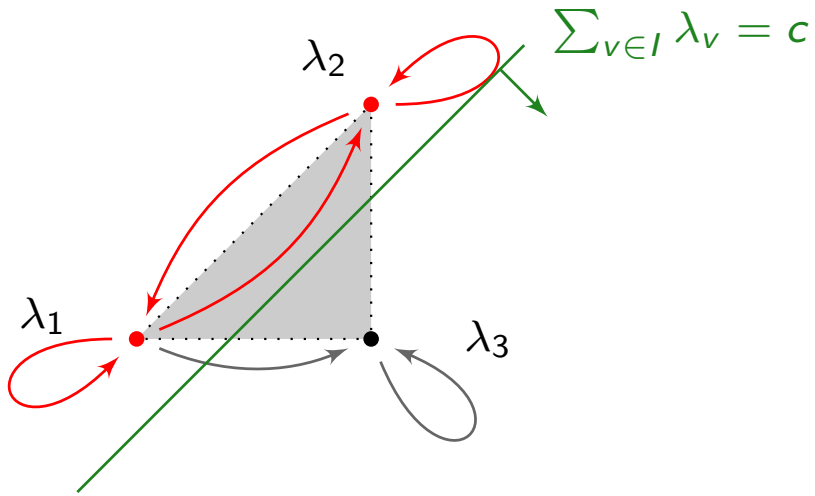
$$\vec{\lambda} \cdot \vec{v} \to \vec{\lambda'}\vec{v}$$

$$\Leftrightarrow$$

$\vec{\lambda'}$ is computed by distributing each λ_v to its successors following a probability distribution.

Non-forgetful implies convergence

A non-forgetful cycle always has an initial component I.



Lemma [Asarin & Basset 2011]

If $\vec{\lambda} \cdot \vec{v} \to \vec{\lambda'} \vec{v'}$, then $\sum_{v \in I} \lambda'_v \leq \sum_{v \in I} \lambda_v$.

This quantity is **nonincreasing** along an infinite repetition of the cycle.

Characterization of convergence

Consider a cycle ρ in the region graph of $\mathcal{A}$.

Lemma

There is no convergence phenomena when the cycle ρ is iterated iff the graph FOG(ρ) is aperiodic.

Hints:

- FOG(ρ) is not aperiodic: there exists k s.t. $FOG(\rho^k)$ is not forgetful
  ➜ convergence
- FOG(ρ) is aperiodic: there exists k s.t. FOG(ρ^k) is complete
  ➜ the reachability relation is complete: no convergence

Overview

Case of deterministic timed automata

Controller chooses the path followed in the region automaton.

Theorem (Sankur et al 2013)

The Robust Controller Synthesis has a solution iff there exists a reachable cycle in $\mathcal{R}(\mathcal{A})$ which is both aperiodic and accepting.

Corollary

The Robust Controller Synthesis is PSPACE-complete.

Non-determinism:

Controller cannot guarantee to follow a given cycle ➜ execution is a tree

➜ lift the aperiodicity condition (on cycles) to trees

A Game on the Region Graph

Two-player turn based game played on the region graph $\mathcal{R}(\mathcal{A})$:

- Controller suggests a (time-successor) region and an action
- Perturbator resolves non-determinism

Winning Condition $\mathcal{W}$

A Controller strategy is winning iff it verifies the two following conditions:

- every outcome satisfies the Büchi condition (acceptance)
- every outcome contains infinitely many factors whose FOG is complete

A Game on the Region Graph

Two-player turn based game played on the region graph $\mathcal{R}(\mathcal{A})$:

- Controller suggests a (time-successor) region and an action
- Perturbator resolves non-determinism

Winning Condition $\mathcal{W}$

A Controller strategy is winning iff it verifies the two following conditions:

- every outcome satisfies the Büchi condition (acceptance)
- every outcome contains infinitely many factors whose FOG is complete

Condition $\mathcal{W}$ can be expressed as a Büchi condition.
➜ the game $(\mathcal{R}(\mathcal{A}), \mathcal{W})$ is determined, and can be solved in EXPTIME.

A Game on the Region Graph (2)

Thanks to the determinacy of the game $(\mathcal{R}(\mathcal{A}), \mathcal{W})$, we prove:

- Controller **wins:** there exists a **finite memory** strategy σ such that every reachable cycle in $\mathcal{R}(\mathcal{A})[\sigma]$ is aperiodic and accepting.
- Perturbator **wins:** there exists a **finite memory** strategy τ such that no reachable cycle in $\mathcal{R}(\mathcal{A})[\tau]$ is aperiodic and accepting.

If Controller wins the abstract game on $\mathcal{R}(\mathcal{A})$, we build $\delta > 0$ and a concrete strategy in $\mathcal{G}_\delta(\mathcal{A})$

If Perturbator wins, we can combine strategy τ with results of [Sankur et al, 2013].

A Game on the Region Graph (2)

Thanks to the determinacy of the game $(\mathcal{R}(\mathcal{A}), \mathcal{W})$, we prove:

- Controller **wins:** there exists a **finite memory** strategy σ such that every reachable cycle in $\mathcal{R}(\mathcal{A})[\sigma]$ is aperiodic and accepting.
- Perturbator **wins:** there exists a **finite memory** strategy τ such that no reachable cycle in $\mathcal{R}(\mathcal{A})[\tau]$ is aperiodic and accepting.

As a consequence, we have:

Theorem

The Robust Controller Synthesis has a solution iff Controller *wins the game* $(\mathcal{R}(\mathcal{A}), \mathcal{W})$.

Corollary

The Robust Controller Synthesis is EXPTIME-complete.

Overview

Stochastic Perturbations

Perturbations of delays are chosen randomly in $[-\delta, \delta]$

➜ Yields a probabilistic setting

Almost-sure winning

Given a timed automaton and a Büchi condition B, does there exist some $\delta > 0$ and a strategy σ for Controller s.t. $\mathbb{P}^{\sigma}(F^{\infty}B) = 1$?

Remember the game $(\mathcal{R}(\mathcal{A}), \mathcal{W})$

Theorem

Controller *wins almost surely iff he wins the game* $(\mathcal{R}(\mathcal{A}), \mathcal{W})$. *In addition, if he loses the game, then he loses almost surely for every* $\delta > 0$.

Markov Decision Process

Non-determinism associated with actions is replaced by a probability distribution on the outgoing transitions.

➜ Yields a Markov Decision Process

Winning Condition $\mathcal{W}'$

A Controller strategy is winning iff it verifies the two following conditions:

- the Büchi condition is verified with probability 1
- every outcome contains infinitely many factors whose FOG is complete

➜ This game with mixed objectives can be solved in EXPTIME, and finite memory strategies are sufficient for Controller.

Theorem

Controller *wins almost surely iff he wins the game* $(\mathcal{R}(\mathcal{A}), \mathcal{W}')$.

Overview

Conclusion

- A Game semantics for modelling perturbations
  disables punctual moves, and too precise strategies.
- Characterization of convergence phenomena
- Synthesis of robust strategies
- Winning strategies can be computed by δ-parameterized data structures. One can compute symbolically, and adjust δ later.

- Increase power of Environment: uncontrollable edges
- Concurrent timed games
- Symbolic approaches
- Priced extensions