Robust Controller Synthesis in Timed Automata

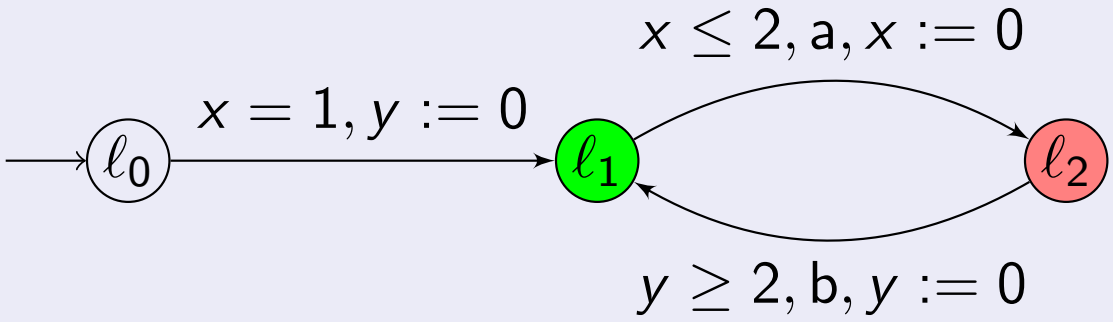
Ocan Sankur

LSV, ENS Cachan & CNRS

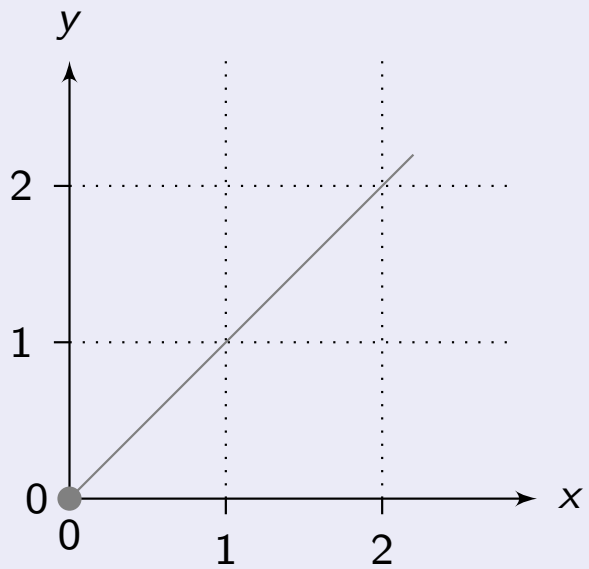
Joint with Patricia Bouyer, Nicolas Markey, Pierre-Alain Reynier.

Timed Automata and Non-determinism

A timed automaton

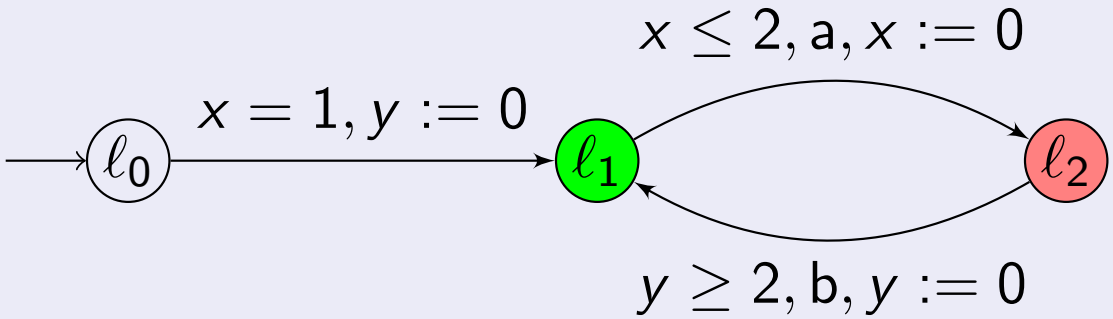


Runs

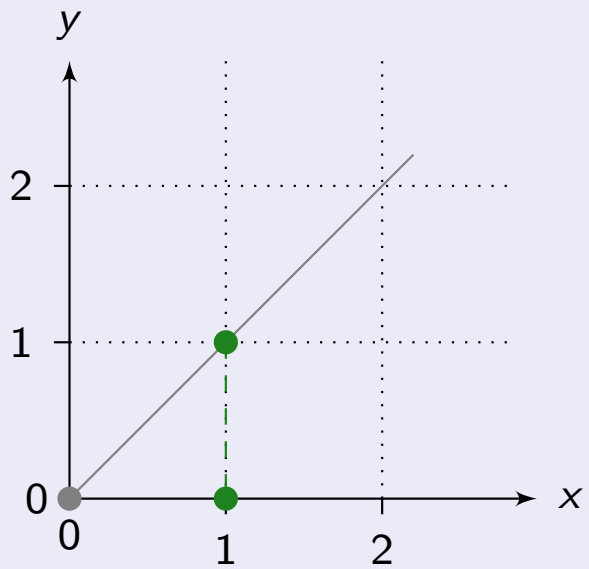


Timed Automata and Non-determinism

A timed automaton

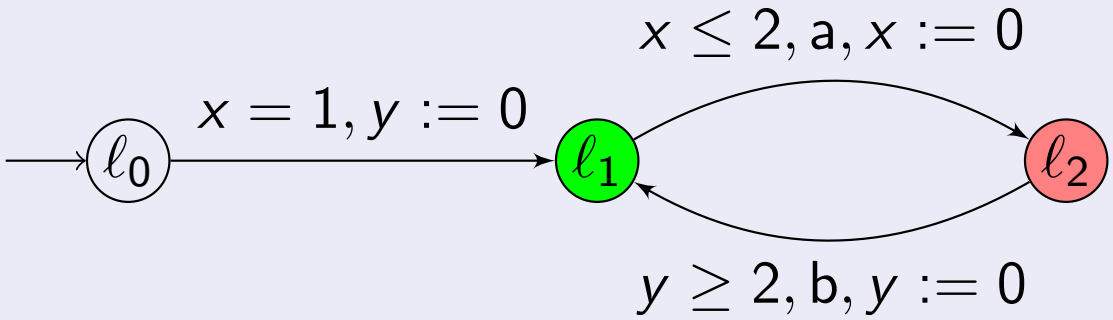


Runs

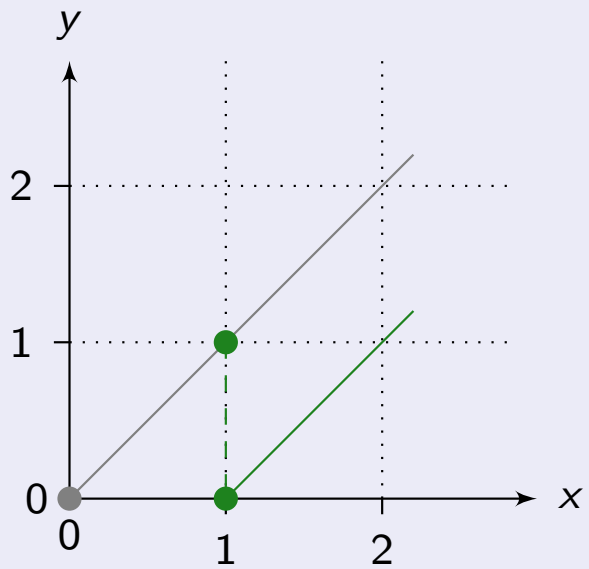


Timed Automata and Non-determinism

A timed automaton

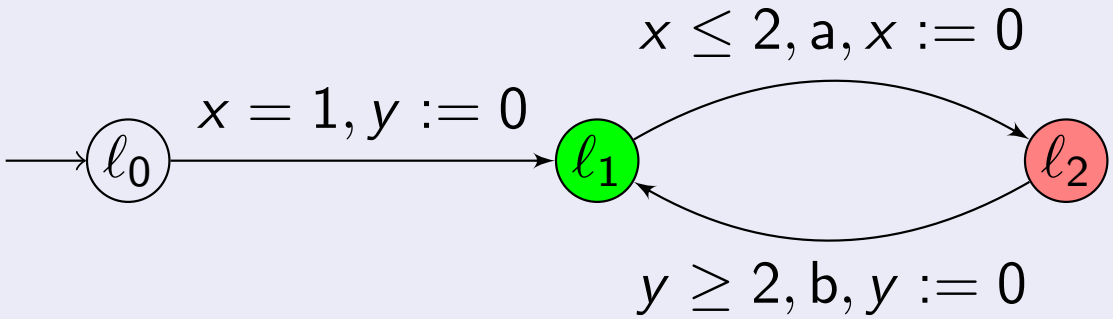


Runs

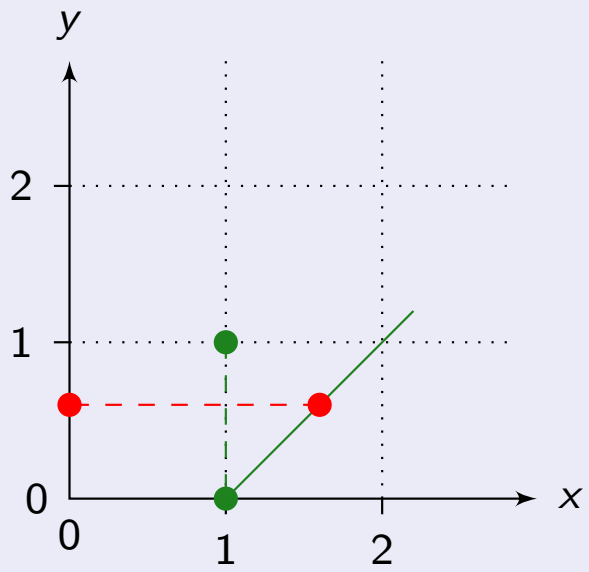


Timed Automata and Non-determinism

A timed automaton

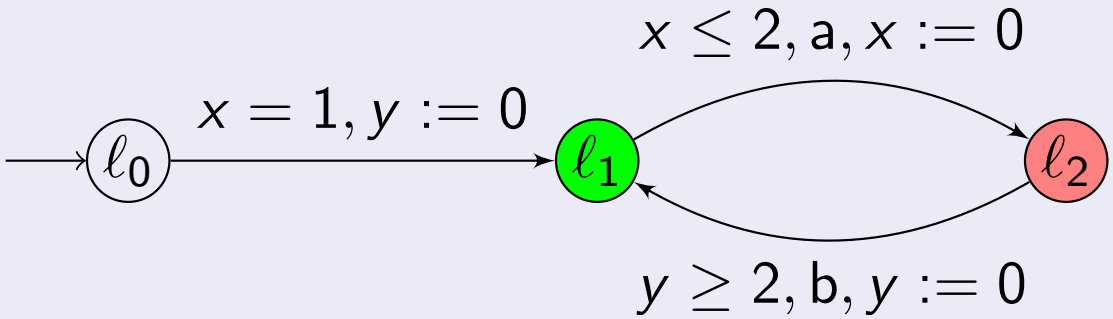


Runs

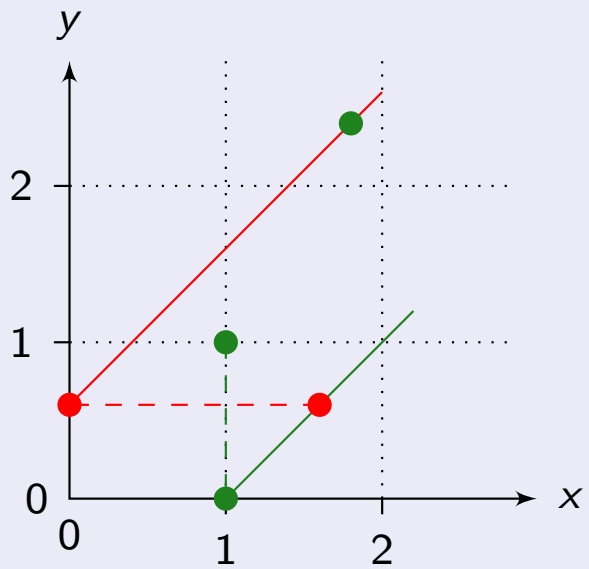


Timed Automata and Non-determinism

A timed automaton

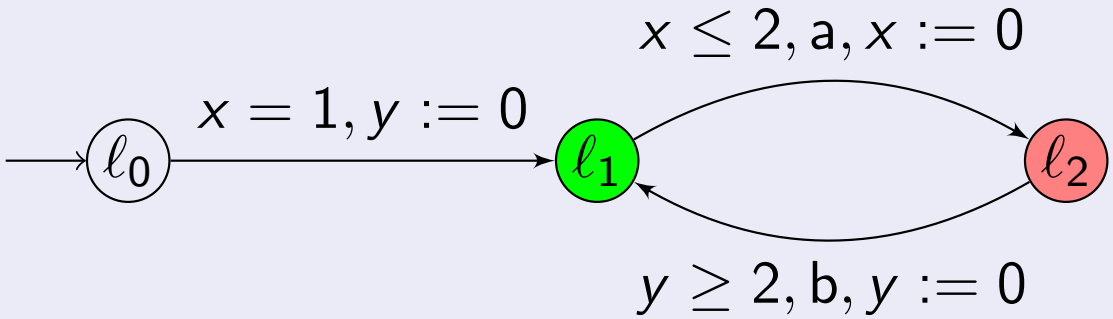


Runs

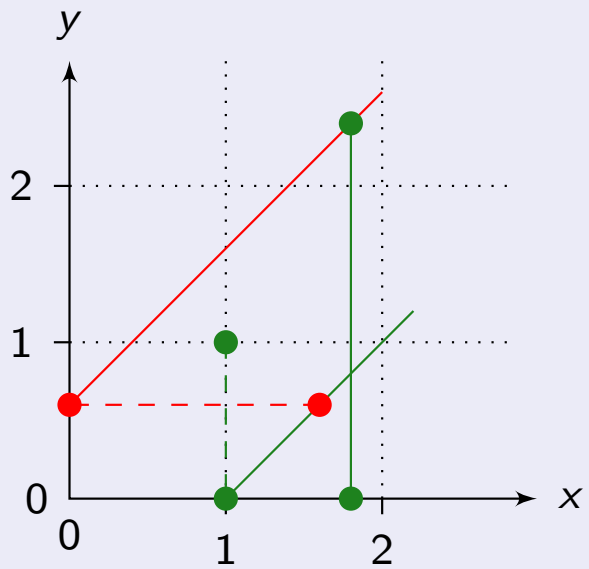


Timed Automata and Non-determinism

A timed automaton



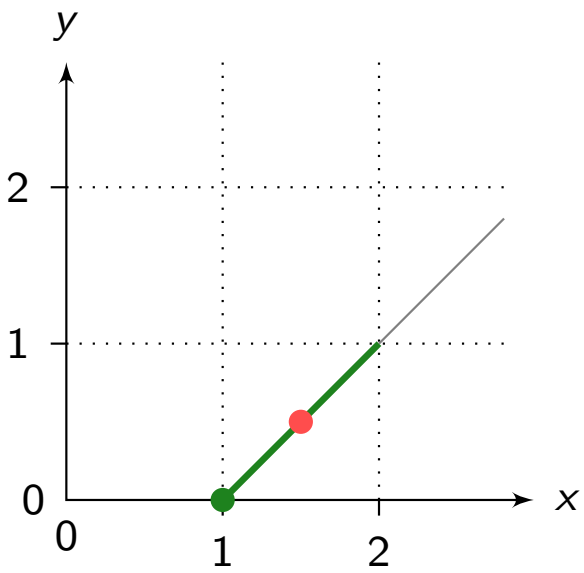
Runs



Controller Synthesis

Given a non-deterministic system, and a specification, compute a **strategy** to control the system.

The system under the strategy is deterministic. $\rightarrow$ an implementation.

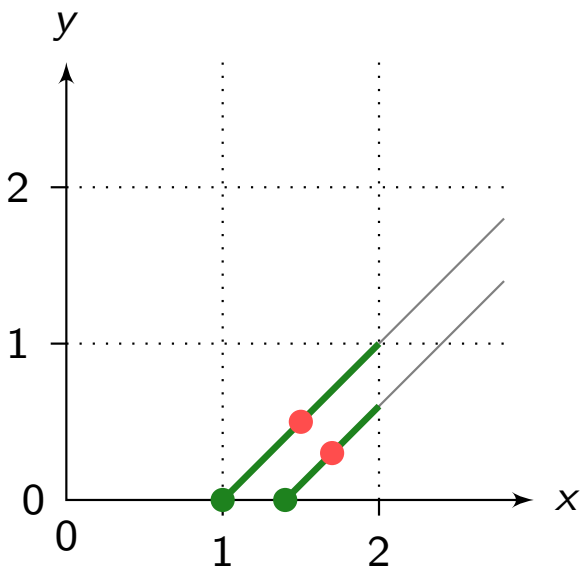


Example: at any state (ℓ_1, x, y), delay $\frac{2-x}{2}$ *i.e.* half way through the guard.

Controller Synthesis

Given a non-deterministic system, and a specification, compute a **strategy** to control the system.

The system under the strategy is deterministic. $\rightarrow$ an implementation.

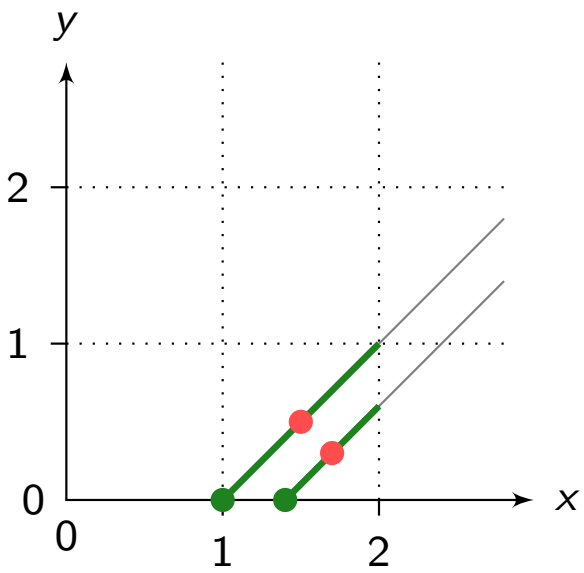


Example: at any state (ℓ_1, x, y), delay $\frac{2-x}{2}$ *i.e.* half way through the guard.

Controller Synthesis

Given a non-deterministic system, and a specification, compute a **strategy** to control the system.

The system under the strategy is deterministic. $\rightarrow$ an implementation.



Example: at any state (ℓ_1, x, y), delay $\frac{2-x}{2}$ *i.e.* half way through the guard.

Theorem [Alur & Dill 1994]

Computing strategies for Büchi objectives in timed automata is PSPACE-complete.

Robustness in Strategies

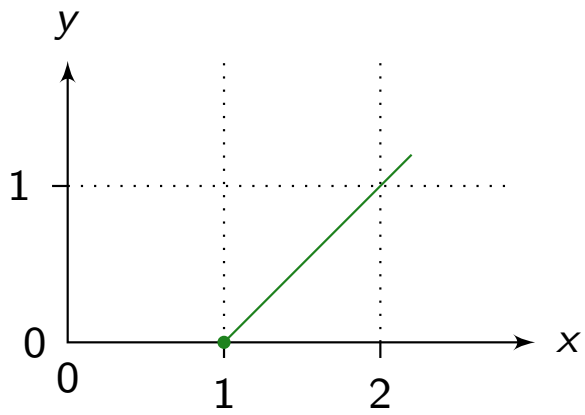
1. However exact timings cannot be ensured in real-time systems.
So the strategy $\left(x \mapsto \text{delay } \frac{2-x}{2}\right)$ cannot be implemented exactly.

Is any strategy valid under an error interval?

Robustness in Strategies

1. However 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.

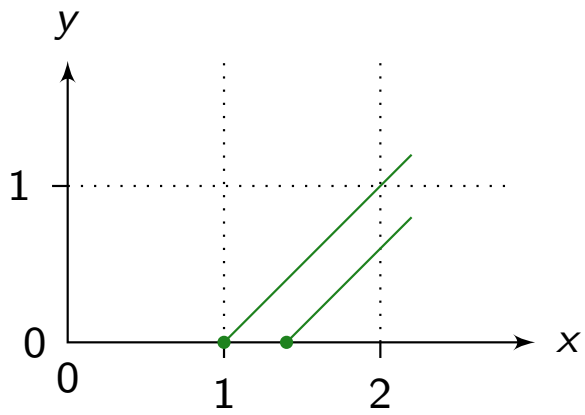
In our example, one can show that x is increasing during consecutive visits to ℓ_2, and the guard is $x \leq 2$.



Robustness in Strategies

1. However 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.

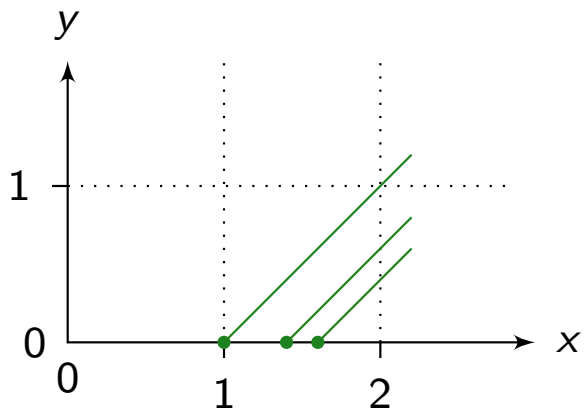
In our example, one can show that x is increasing during consecutive visits to ℓ_2, and the guard is $x \leq 2$.



Robustness in Strategies

1. However 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.

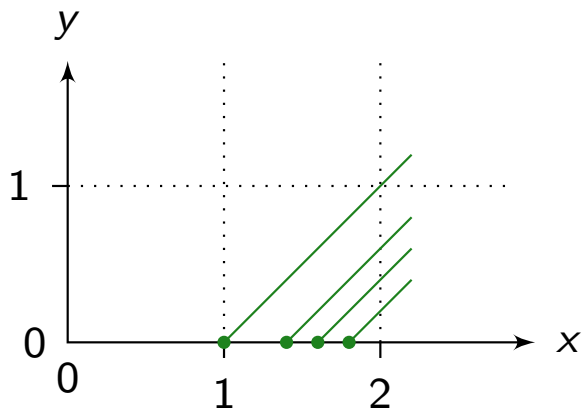
In our example, one can show that x is increasing during consecutive visits to ℓ_2, and the guard is $x \leq 2$.



Robustness in Strategies

1. However 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.

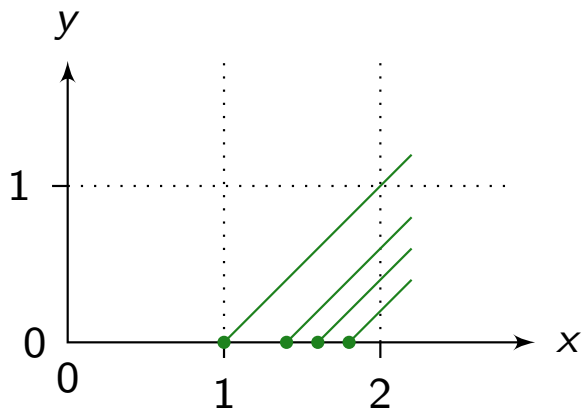
In our example, one can show that x is increasing during consecutive visits to ℓ_2, and the guard is $x \leq 2$.



Robustness in Strategies

1. However 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_2, and the guard is $x \leq 2$.



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 in Strategies (2)

Some strategies in timed automata are not realistic,
and may require high precision, and can cause convergence.
Goal: Develop a theory of *robust strategies* that tolerate error and avoid convergence.

Robustness in Strategies (2)

(Conservative) Perturbation Game: Controller vs Environment.

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

1. **Controller** chooses a delay $d \geq \delta$, and an edge $\ell \xrightarrow{g,R} \ell'$, such that $\nu + d + \epsilon \models g$ for all $\epsilon \in [-\delta, \delta]$.

Robustness in Strategies (2)

(Conservative) Perturbation Game: Controller vs Environment.

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

1. **Controller** chooses a delay $d \geq \delta$, and an edge $\ell \xrightarrow{g,R} \ell'$, such that $\nu + d + \epsilon \models g$ for all $\epsilon \in [-\delta, \delta]$.
2. **Environment** chooses $\epsilon \in [-\delta, +\delta]$,

Robustness in Strategies (2)

(Conservative) Perturbation Game: Controller vs Environment.

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

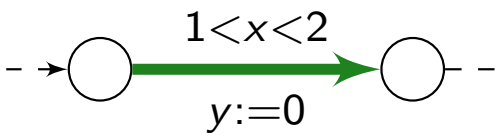
1. **Controller** chooses a delay $d \geq \delta$, and an edge $\ell \xrightarrow{g,R} \ell'$, such that $\nu + d + \epsilon \models g$ for all $\epsilon \in [-\delta, \delta]$.
2. **Environment** chooses $\epsilon \in [-\delta, +\delta]$,
3. New state is $(\ell', (\nu + d + \epsilon)[R \leftarrow 0])$.

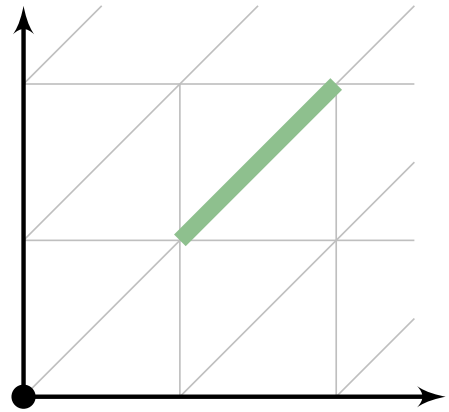
Robustness in Strategies (2)

(Conservative) Perturbation Game: Controller vs Environment.

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

1. **Controller** chooses a delay $d \geq \delta$, and an edge $\ell \xrightarrow{g,R} \ell'$, such that $\nu + d + \epsilon \models g$ for all $\epsilon \in [-\delta, \delta]$.
2. **Environment** chooses $\epsilon \in [-\delta, +\delta]$,
3. New state is $(\ell', (\nu + d + \epsilon)[R \leftarrow 0])$.



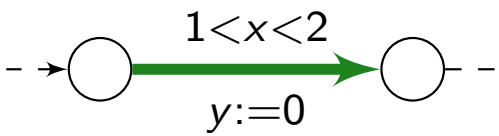


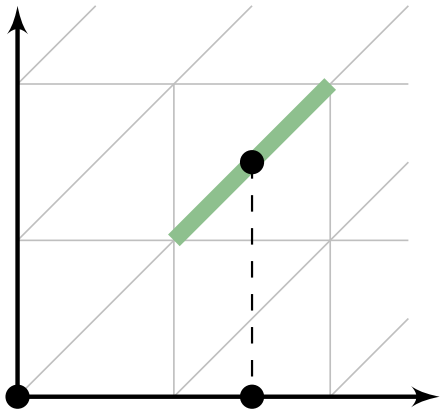
Robustness in Strategies (2)

(Conservative) Perturbation Game: Controller vs Environment.

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

1. **Controller** chooses a delay $d \geq \delta$, and an edge $\ell \xrightarrow{g,R} \ell'$, such that $\nu + d + \epsilon \models g$ for all $\epsilon \in [-\delta, \delta]$.
2. **Environment** chooses $\epsilon \in [-\delta, +\delta]$,
3. New state is $(\ell', (\nu + d + \epsilon)[R \leftarrow 0])$.



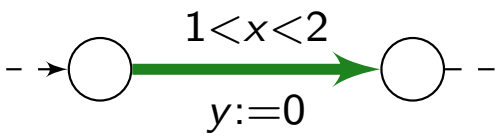


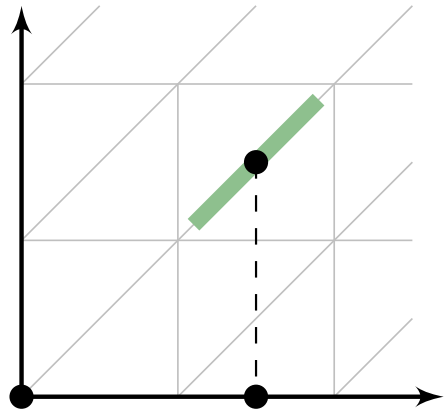
Robustness in Strategies (2)

(Conservative) Perturbation Game: Controller vs Environment.

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

1. **Controller** chooses a delay $d \geq \delta$, and an edge $\ell \xrightarrow{g,R} \ell'$, such that $\nu + d + \epsilon \models g$ for all $\epsilon \in [-\delta, \delta]$.
2. **Environment** chooses $\epsilon \in [-\delta, +\delta]$,
3. New state is $(\ell', (\nu + d + \epsilon)[R \leftarrow 0])$.



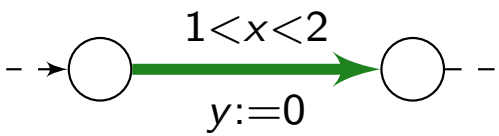


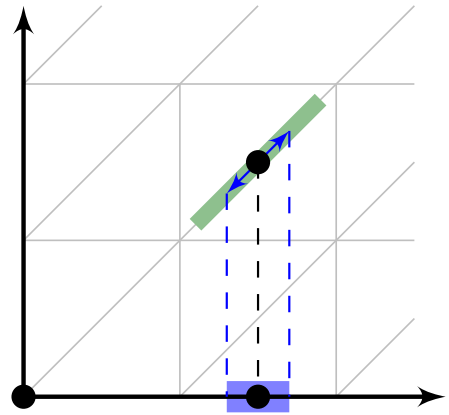
Robustness in Strategies (2)

(Conservative) Perturbation Game: Controller vs Environment.

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

1. **Controller** chooses a delay $d \geq \delta$, and an edge $\ell \xrightarrow{g,R} \ell'$, such that $\nu + d + \epsilon \models g$ for all $\epsilon \in [-\delta, \delta]$.
2. **Environment** chooses $\epsilon \in [-\delta, +\delta]$,
3. New state is $(\ell', (\nu + d + \epsilon)[R \leftarrow 0])$.





Previous work: Chatterjee, Henzinger, Prabhu 2008: for **fixed** $\delta > 0$.

Main Result: Parameterized Robust Control

Parameterized robust control

Given a timed automaton A, and a Büchi condition ϕ, decide whether for small enough $\delta > 0$, $\mathcal{G}(A)$ satisfies ϕ.

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

The problem consists in finding cycles that do not become blocked (converge).

Main Result: Parameterized Robust Control

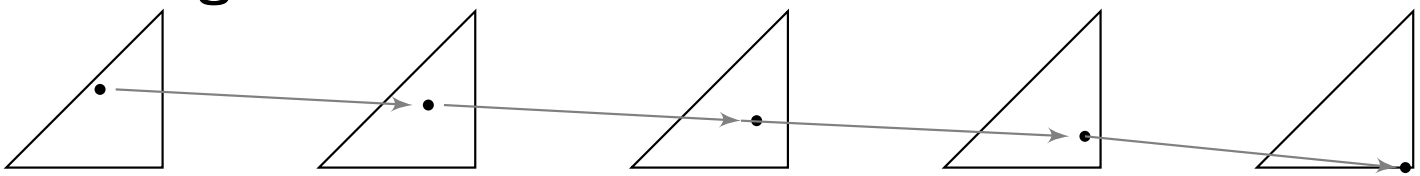
Parameterized robust control

Given a timed automaton A, and a Büchi condition ϕ, decide whether for small enough $\delta > 0$, $\mathcal{G}(A)$ satisfies ϕ.

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

The problem consists in finding cycles that do not become blocked (converge).

Convergence:



Main Result: Parameterized Robust Control

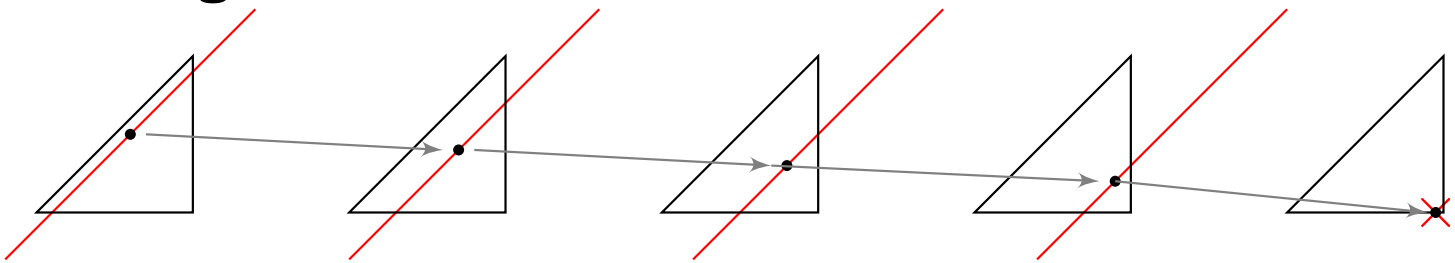
Parameterized robust control

Given a timed automaton A, and a Büchi condition ϕ, decide whether for small enough $\delta > 0$, $\mathcal{G}(A)$ satisfies ϕ.

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

The problem consists in finding cycles that do not become blocked (converge).

Convergence:



Environment can force the play inside a half-space.

Main Result: Parameterized Robust Control

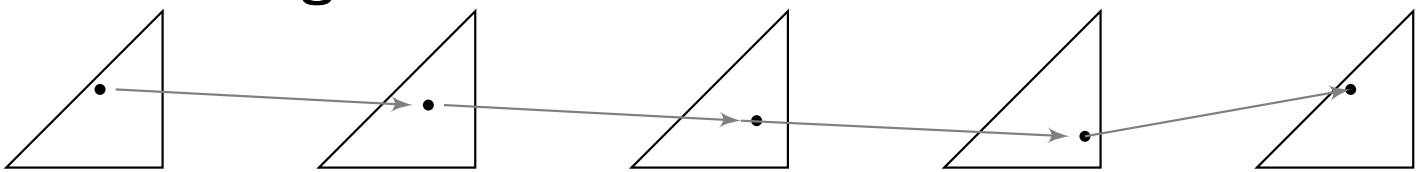
Parameterized robust control

Given a timed automaton A, and a Büchi condition ϕ, decide whether for small enough $\delta > 0$, $\mathcal{G}(A)$ satisfies ϕ.

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

The problem consists in finding cycles that do not become blocked (converge).

No Convergence:



No such constraining half-spaces.

Main Result: Parameterized Robust Control

Parameterized robust control

Given a timed automaton A, and a Büchi condition ϕ, decide whether for small enough $\delta > 0$, $\mathcal{G}(A)$ satisfies ϕ.

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

The problem consists in finding cycles that do not become blocked (converge).

Theorem

Parameterized robust control is PSPACE-complete.

Main Result: Parameterized Robust Control

Parameterized robust control

Given a timed automaton A, and a Büchi condition ϕ, decide whether for small enough $\delta > 0$, $\mathcal{G}(A)$ satisfies ϕ.

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

The problem consists in finding cycles that do not become blocked (converge).

Theorem

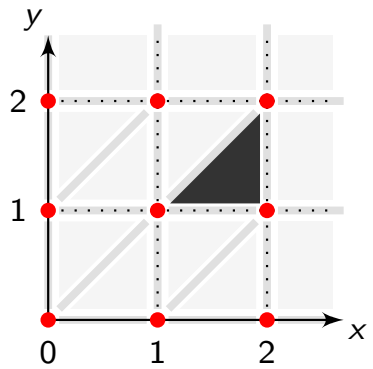
Parameterized robust control is PSPACE-complete.

Robustly controllable $\Leftrightarrow$ there exists a reachable "forgetful" cycle.

If all states are accepting, then
Thick timed automata are exactly those that are robustly controllable.

Reachability in Timed Automata

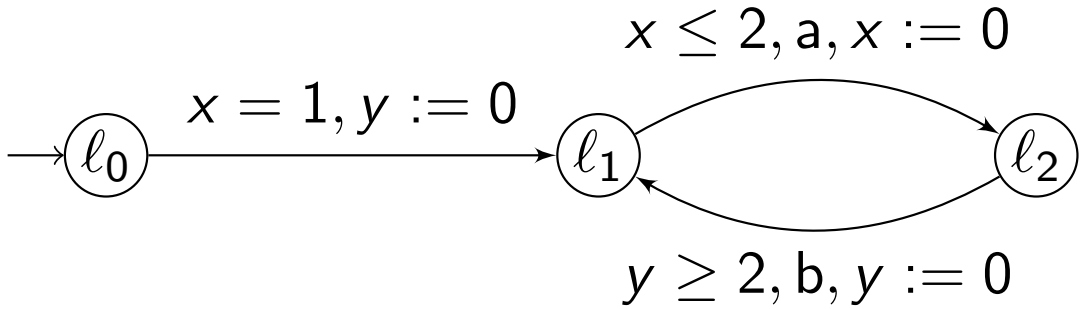
Runs of timed automata can be characterized by runs visiting only the **vertices** of regions. given the topological closure of the guards.

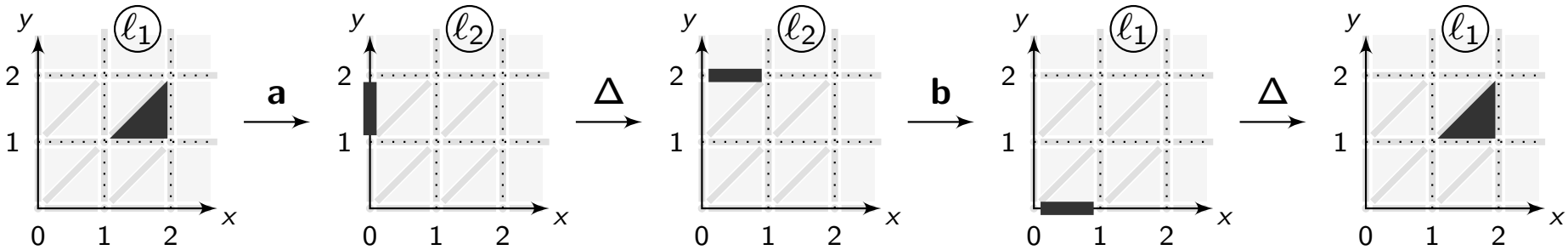


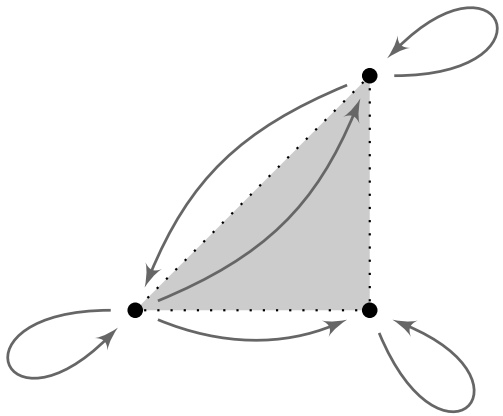
Orbit Graph

The **orbit graph** of a *cycle on regions* is a graph where:
- nodes are the vertices of the region.
- there is an edge $a \to b$, if b is reachable from a along the path.

Orbit Graph Example

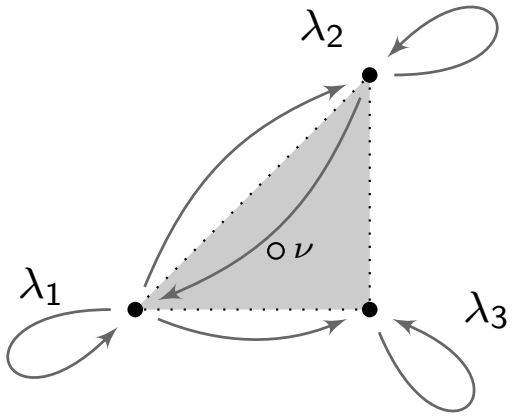






A cycle is called **forgetful** if it is strongly connected (Asarin & Basset 2011)

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]

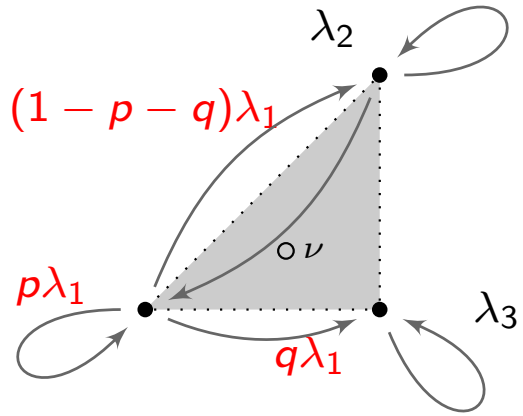
Given a path, and valuation $\vec{\lambda} \cdot \vec{v}$,

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

$$\Leftrightarrow$$

$\vec{\lambda'}$ 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]

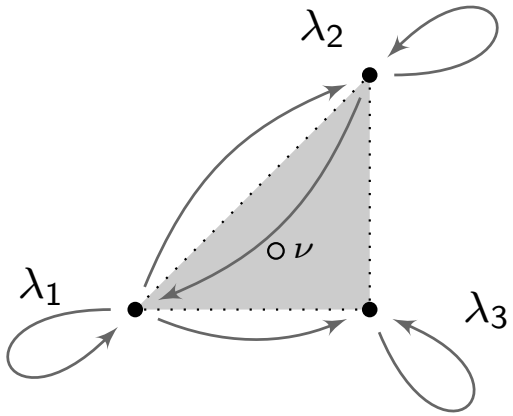
Given a path, and valuation $\vec{\lambda} \cdot \vec{v}$,

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

$$\Leftrightarrow$$

$\vec{\lambda'}$ 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]

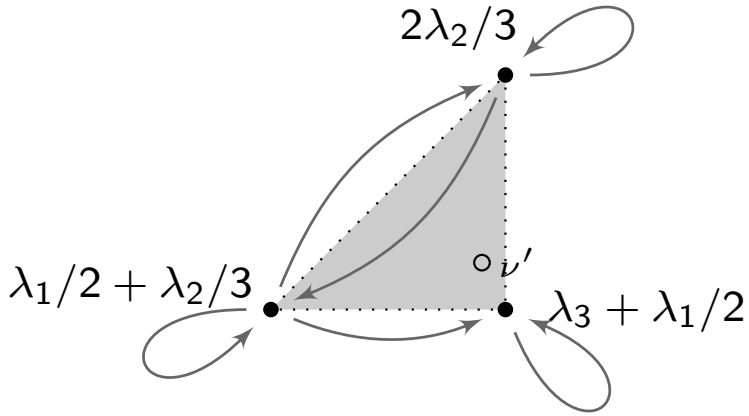
Given a path, and valuation $\vec{\lambda} \cdot \vec{v}$,

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

$$\Leftrightarrow$$

$\vec{\lambda'}$ 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.

Proof 1: When There is an Accepting Forgetful Cycle

Show that Controller can win.

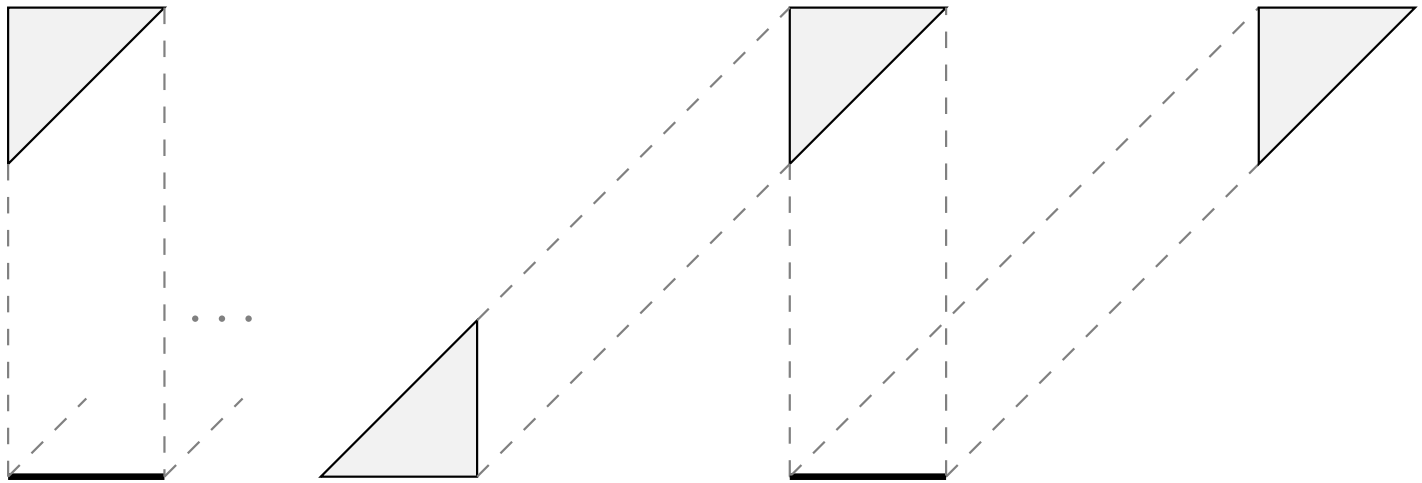
Claim: If π is a forgetful cycle, then the folded orbit graph of π^n is a complete graph for some $n \geq 1$.
In the rest, the graph of π is complete.

Proof 1: When There is an Accepting Forgetful Cycle

Show that Controller can win.

Claim: If π is a forgetful cycle, then the folded orbit graph of π^n is a complete graph for some $n \geq 1$.
In the rest, the graph of π is complete.

Claim: If π is has a complete f.o.g., then $\forall(\nu, \nu') \in r \times s,\ \nu \to \nu'$.

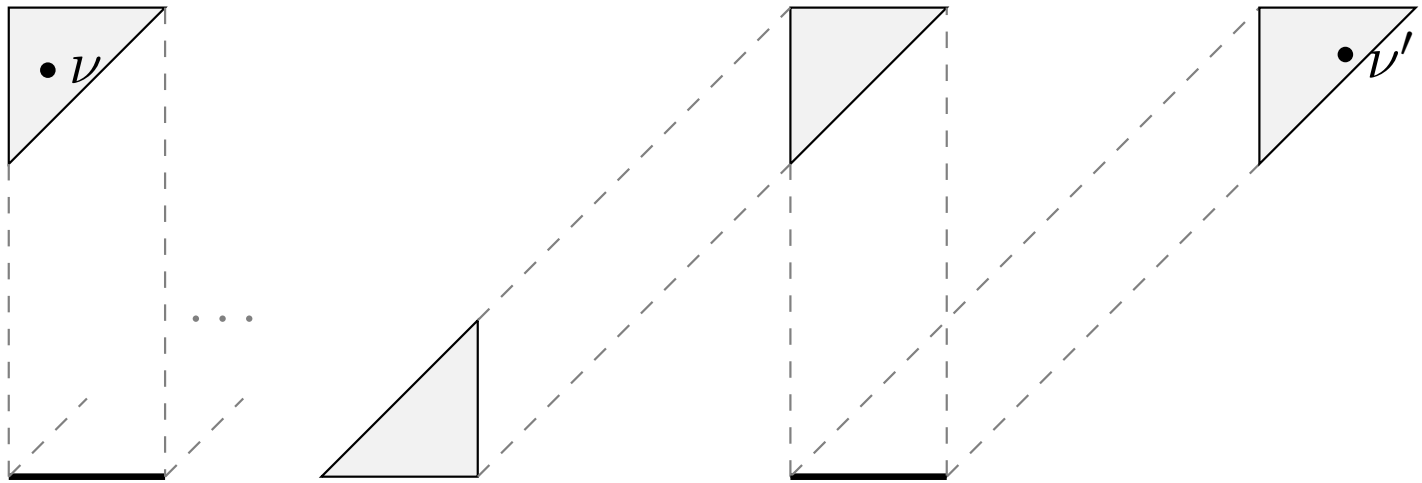


Proof 1: When There is an Accepting Forgetful Cycle

Show that Controller can win.

Claim: If π is a forgetful cycle, then the folded orbit graph of π^n is a complete graph for some $n \geq 1$.
In the rest, the graph of π is complete.

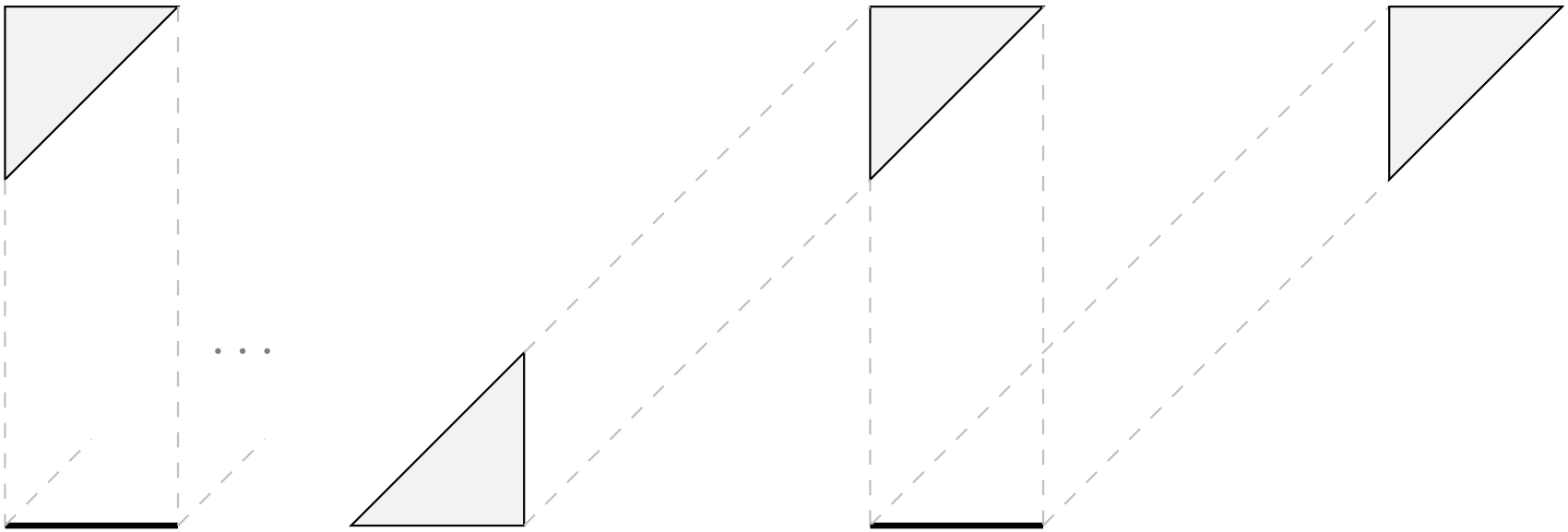
Claim: If π is has a complete f.o.g., then $\forall(\nu, \nu') \in r \times s$, $\nu \to \nu'$.



Proof 1: There is an Accepting Forgetful Cycle

- Controller's strategy: try to come back at the middle of the region.

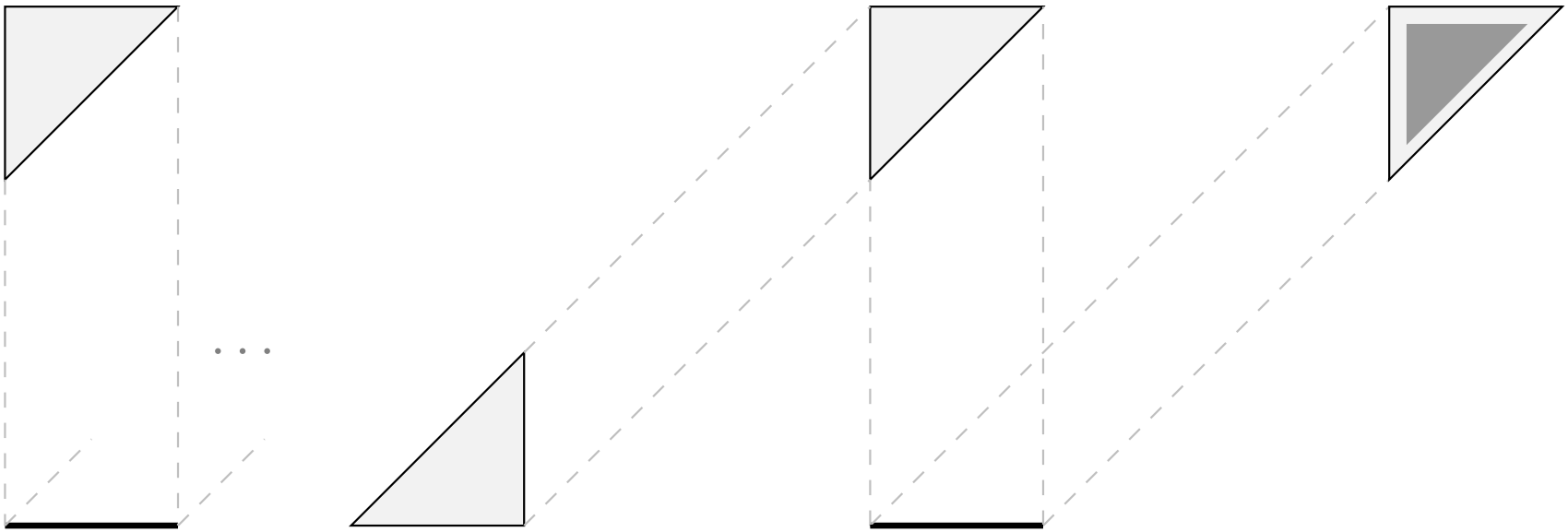
Without perturbations, this is possible by hypothesis:



Proof 1: There is an Accepting Forgetful Cycle

- Controller's strategy: try to come back at the middle of the region.

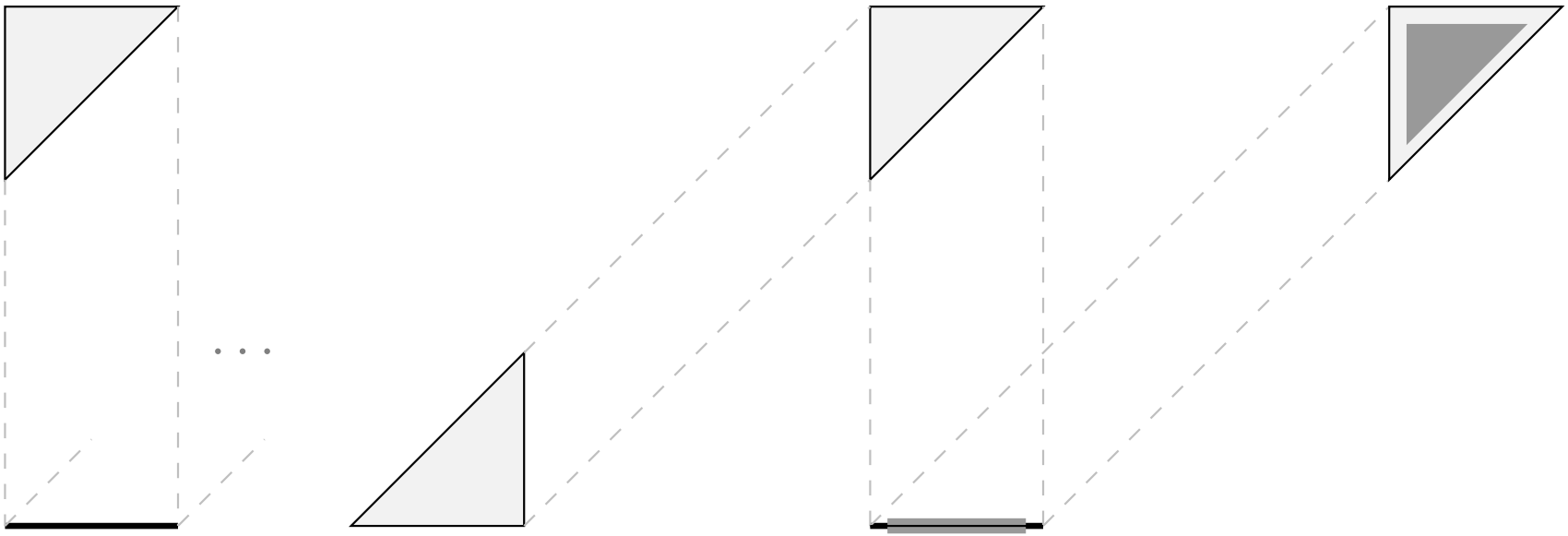
Without perturbations, this is possible by hypothesis:



Proof 1: There is an Accepting Forgetful Cycle

- Controller's strategy: try to come back at the middle of the region.

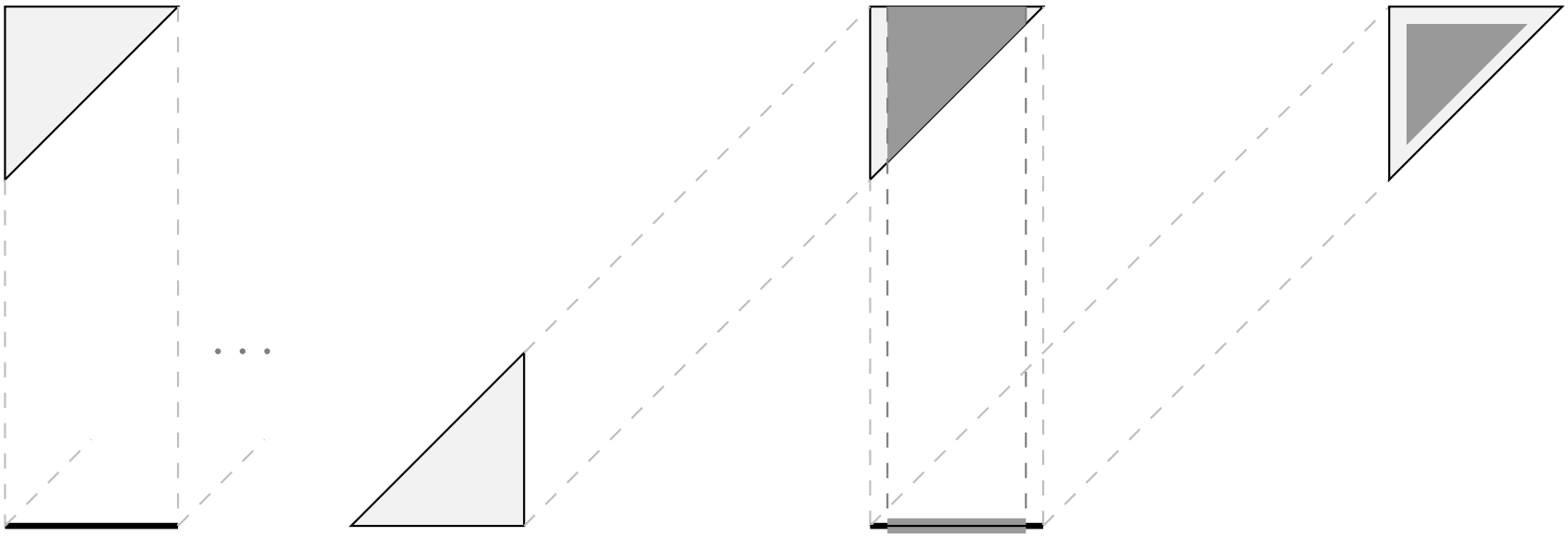
Without perturbations, this is possible by hypothesis:



Proof 1: There is an Accepting Forgetful Cycle

- Controller's strategy: try to come back at the middle of the region.

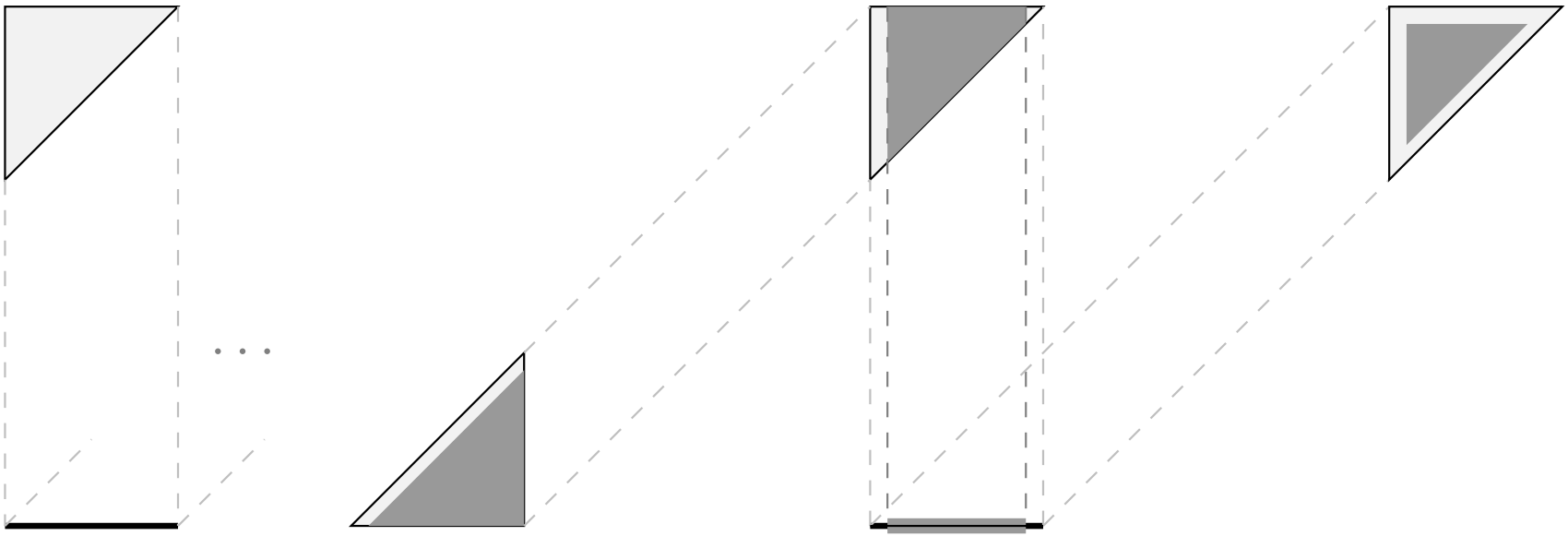
Without perturbations, this is possible by hypothesis:



Proof 1: There is an Accepting Forgetful Cycle

- Controller's strategy: try to come back at the middle of the region.

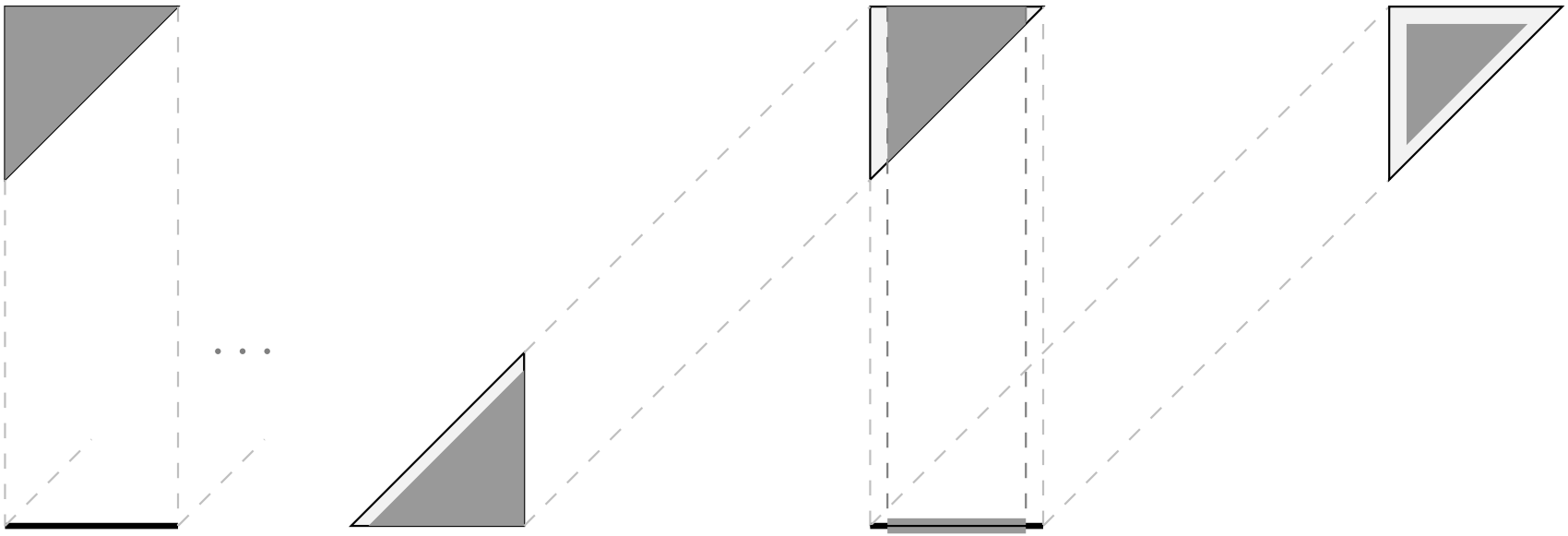
Without perturbations, this is possible by hypothesis:



Proof 1: There is an Accepting Forgetful Cycle

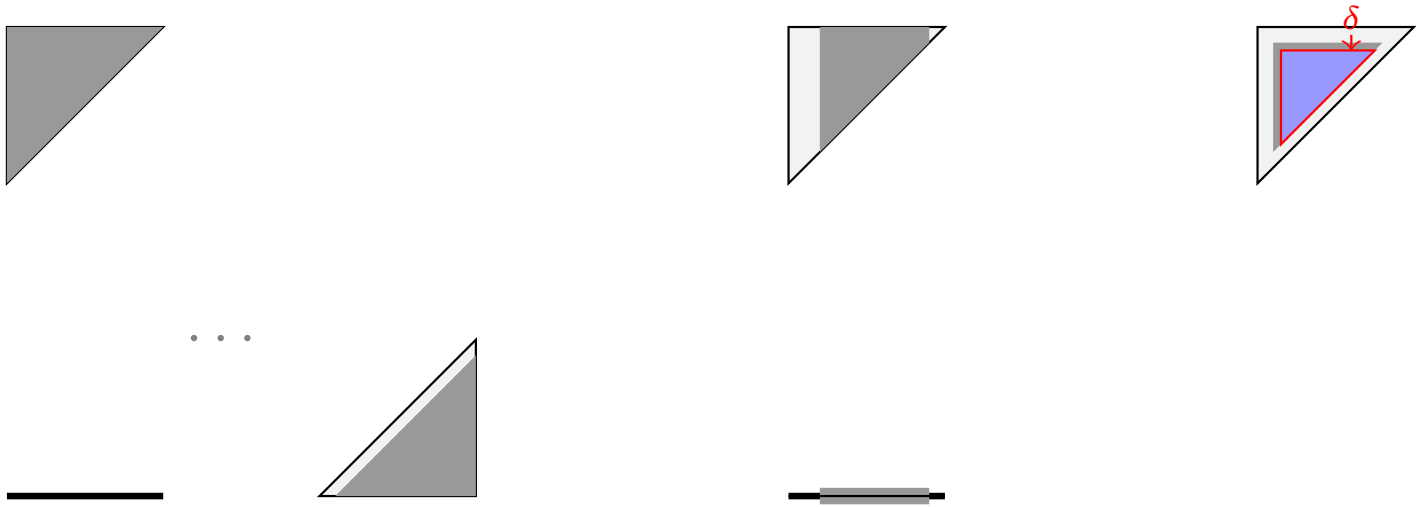
- Controller's strategy: try to come back at the middle of the region.

Without perturbations, this is possible by hypothesis:



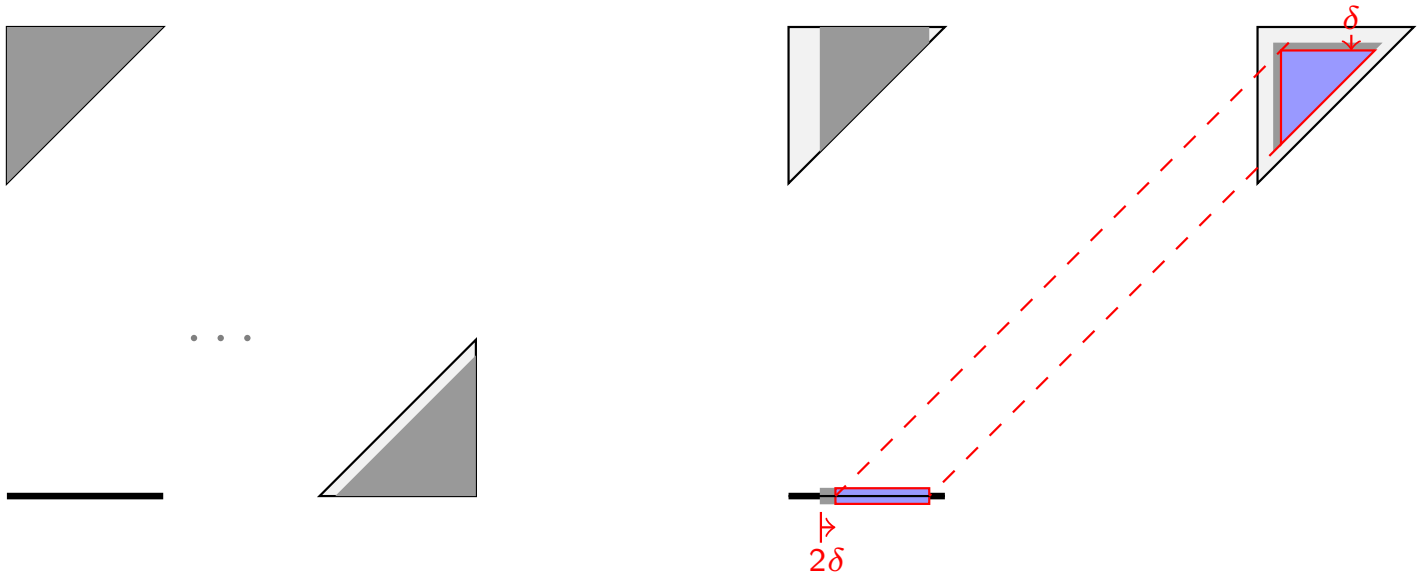
Proof 1: There is an Accepting Forgetful Cycle

Under perturbations, we need the *controllable predecessors* of a subset.



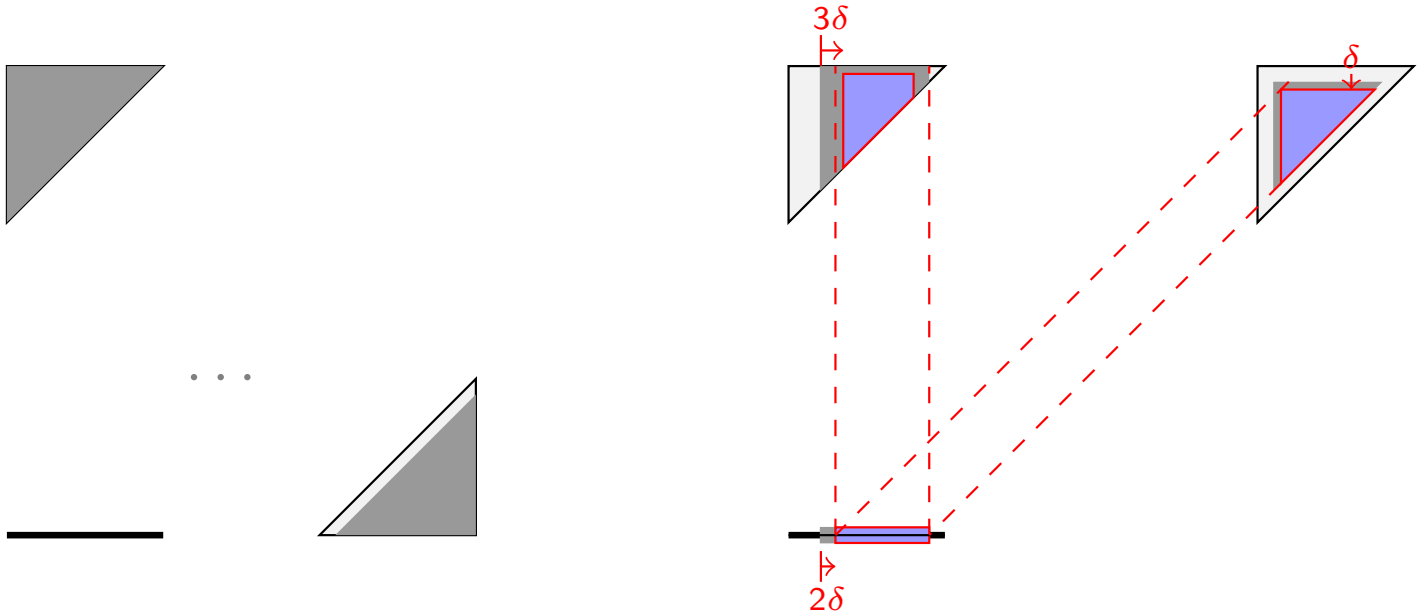
Proof 1: There is an Accepting Forgetful Cycle

Under perturbations, we need the *controllable predecessors* of a subset.



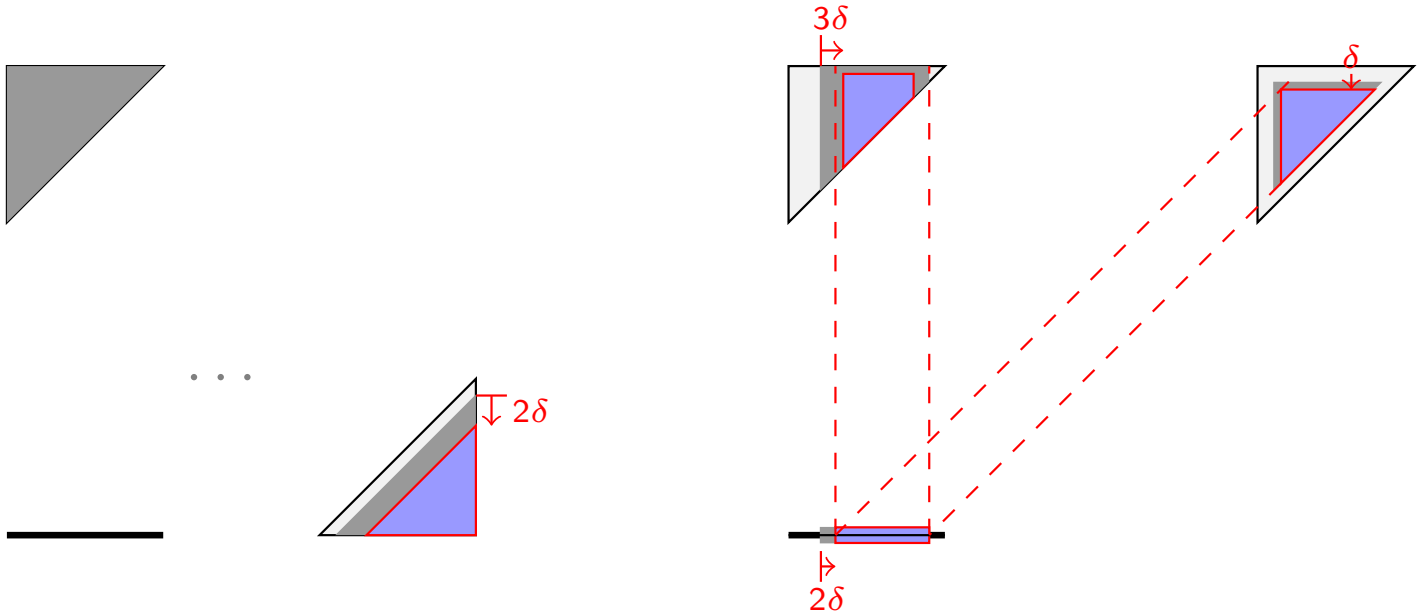
Proof 1: There is an Accepting Forgetful Cycle

Under perturbations, we need the *controllable predecessors* of a subset.



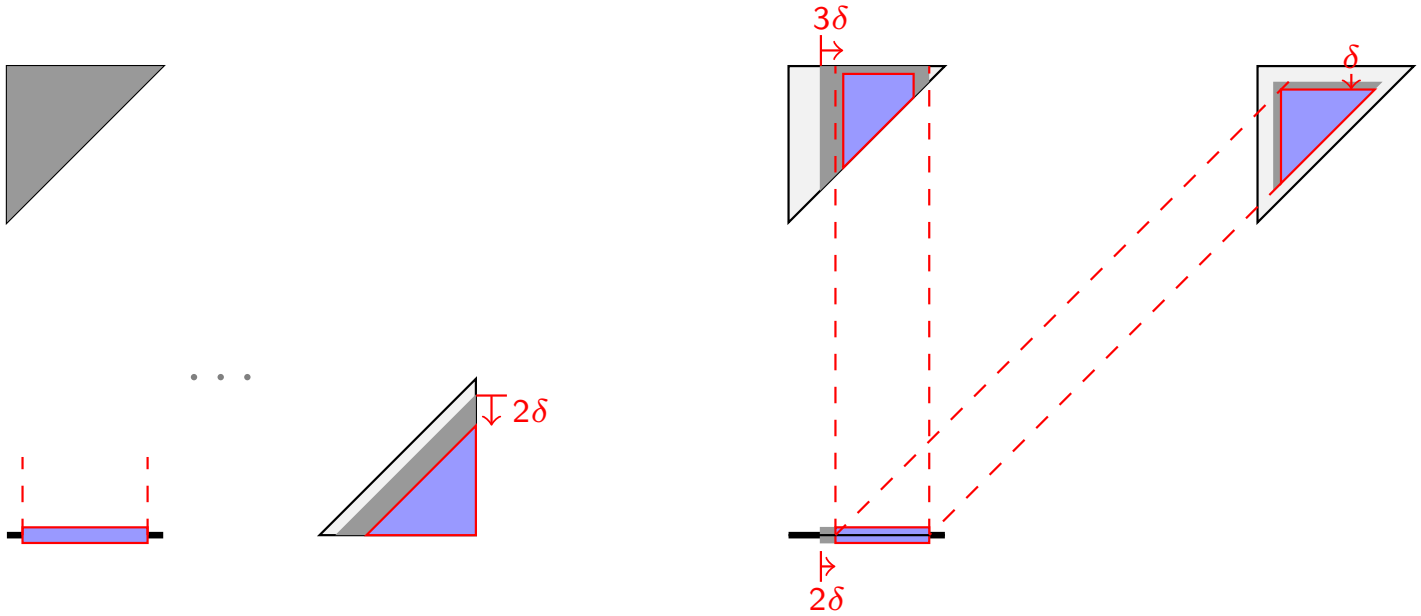
Proof 1: There is an Accepting Forgetful Cycle

Under perturbations, we need the *controllable predecessors* of a subset.



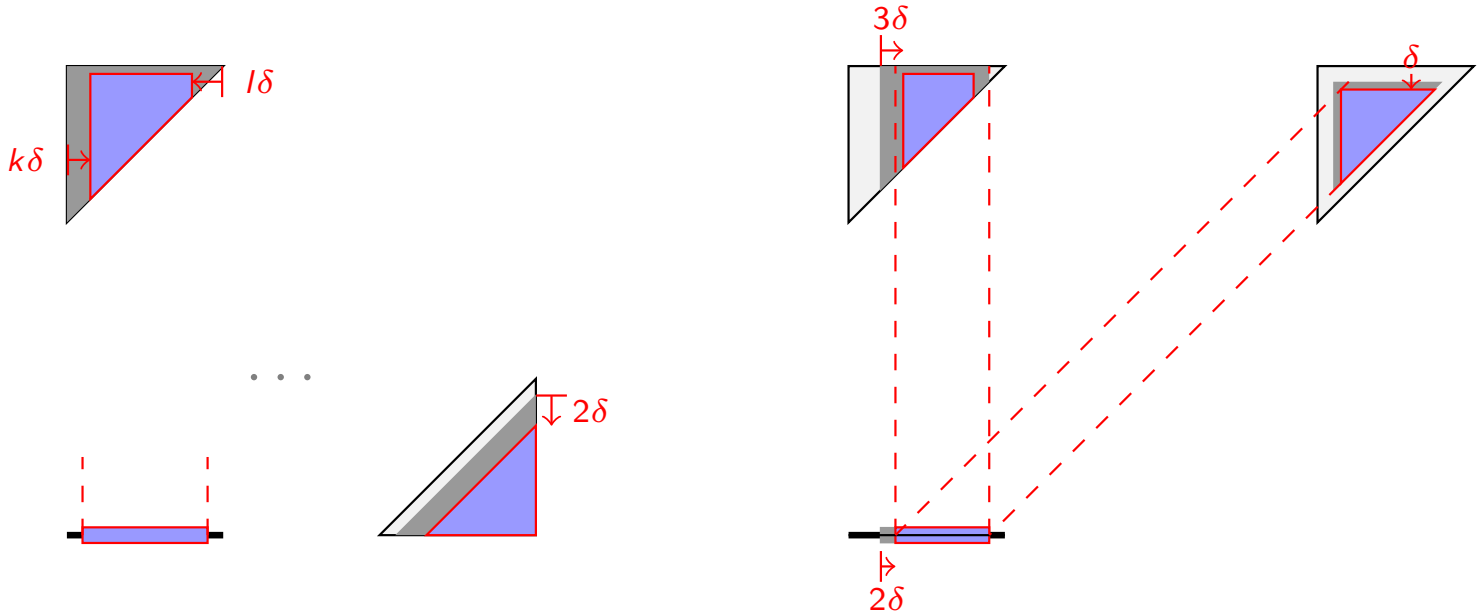
Proof 1: There is an Accepting Forgetful Cycle

Under perturbations, we need the *controllable predecessors* of a subset.



Proof 1: There is an Accepting Forgetful Cycle

Under perturbations, we need the *controllable predecessors* of a subset.

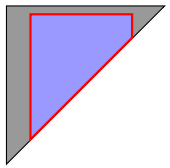


Each red-blue set is a **shrinking** of a gray zone.
The **shrunk DBM** data structure can be used to compute / represent these sets, where computations hold for all small enough $\delta > 0$.

S., Bouyer, Markey. Shrinking Timed Automata. FSTTCS'11.

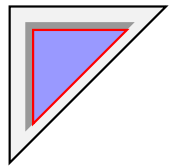
Bouyer, Markey, S. Robust Reachability in Timed Automata: A Game-Based Approach. ICALP'12.

Case: There is an Accepting Forgetful Cycle



A_δ

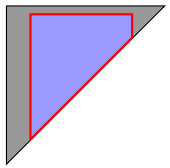
$\supseteq$



B_δ

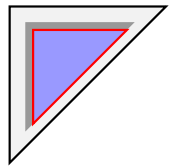
Computations with shrunk DBMs hold for all $\delta \in [0, \delta_0]$, for some $\delta_0 > 0$. Here, for any $\delta > 0$, from A_δ, Controller can ensure reaching B_δ.

Case: There is an Accepting Forgetful Cycle



A_δ

$\supseteq$

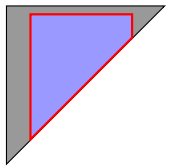


B_δ

Computations with shrunk DBMs hold for all $\delta \in [0, \delta_0]$, for some $\delta_0 > 0$. Here, for any $\delta > 0$, from A_δ, Controller can ensure reaching B_δ.

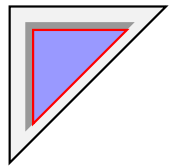
Claim 1: For (computable) small enough $\delta > 0$, $A_\delta, B_\delta \neq \emptyset$.
Claim 2: For (computable) small enough $\delta > 0$, $B_\delta \subseteq A_\delta$.

Case: There is an Accepting Forgetful Cycle



A_δ

$\supseteq$



B_δ

Computations with shrunk DBMs hold for all $\delta \in [0, \delta_0]$, for some $\delta_0 > 0$. Here, for any $\delta > 0$, from A_δ, Controller can ensure reaching B_δ.

Claim 1: For (computable) small enough $\delta > 0$, $A_\delta, B_\delta \neq \emptyset$.
Claim 2: For (computable) small enough $\delta > 0$, $B_\delta \subseteq A_\delta$.

Controller ends in A_δ at each iteration, so it can repeat its strategy.

Controller wins!

Proof 2: Only Non-Forgetful Cycles

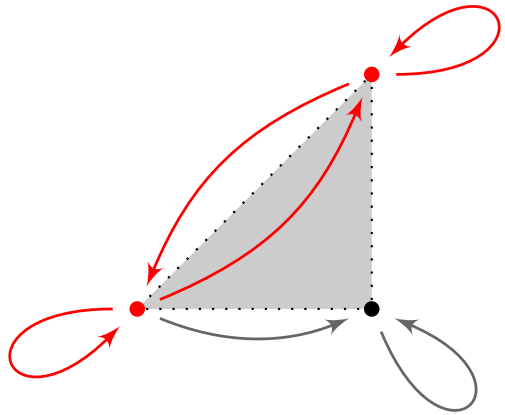
Show that if all cycles are non-forgetful, Controller looses.

If accepting for Büchi, then for some strategy, some cycle is repeated infinitely often.

Assume there are only non-forgetful cycles. We will show a contradiction (proof idea).

Proof 2: Only Non-Forgetful Cycles

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



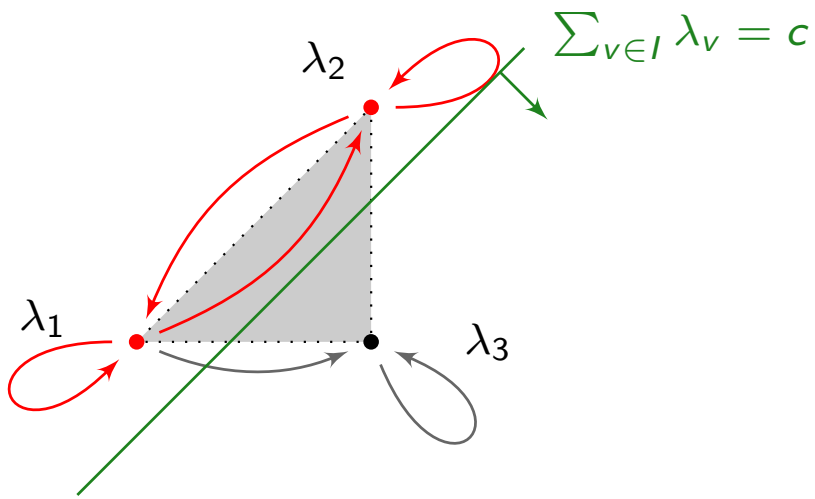
Lemma [Asarin & Basset 2011]

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 quantitiy is **nonincreasing** along an infinite repetition of the cycle.

Proof 2: Only Non-Forgetful Cycles

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



Lemma [Asarin & Basset 2011]

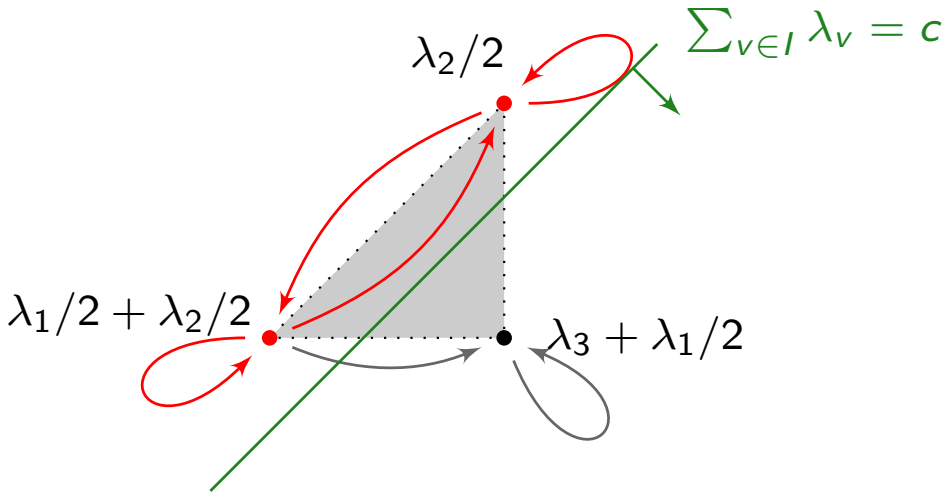
If $\vec{\lambda} \cdot \vec{v} \to \vec{\lambda'} \vec{v'}$, then $\sum_{v \in I} \lambda'_v \leq \sum_{v \in I} \lambda_v$.

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

Proof: Each λ'_v is the sum of the predecessors of v multiplied by a probability.

Proof 2: Only Non-Forgetful Cycles

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 $\sum_{v \in I} \lambda'_v \leq \sum_{v \in I} \lambda_v$.

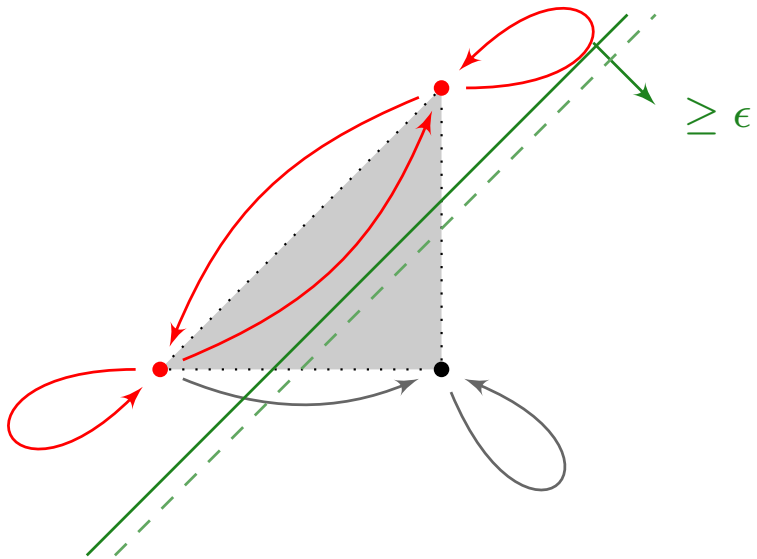
This quantitiy is **nonincreasing** along an infinite repetition of the cycle.

Proof: Each λ'_v is the sum of the predecessors of v multiplied by a probability.

Case: Only Non-Forgetful Cycles

Lemma

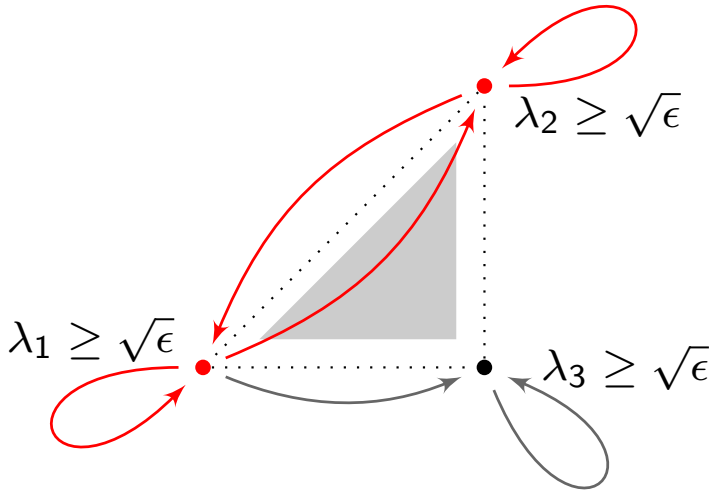
Environment has a strategy to ensure, for any $\vec{\lambda} \cdot \vec{v} \to \vec{\lambda'} \vec{v'}$, $\sum_{v \in I} \lambda'_v \leq \sum_{v \in I} \lambda_v - \epsilon$.



Case: Only Non-Forgetful Cycles

Lemma

Environment has a strategy to ensure, for any $\vec{\lambda} \cdot \vec{v} \to \vec{\lambda'} \vec{v'}$, $\sum_{v \in I} \lambda'_v \leq \sum_{v \in I} \lambda_v - \epsilon$.

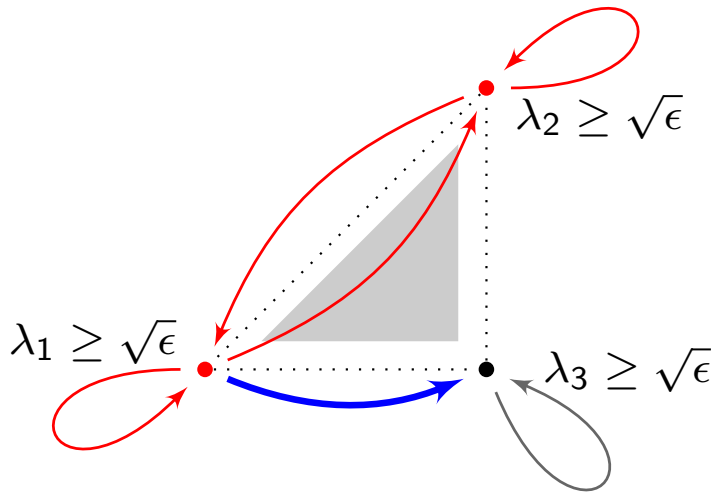


Claim 1: Environment can enforce $\lambda_v \geq \sqrt{\epsilon}$ for all v.

Case: Only Non-Forgetful Cycles

Lemma

Environment has a strategy to ensure, for any $\vec{\lambda} \cdot \vec{v} \to \vec{\lambda'} \vec{v'}$, $\sum_{v \in I} \lambda'_v \leq \sum_{v \in I} \lambda_v - \epsilon$.

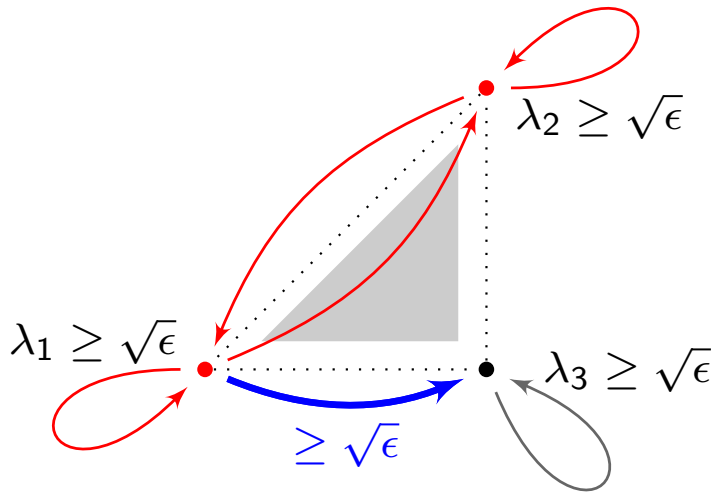


Claim 1: Environment can enforce $\lambda_v \geq \sqrt{\epsilon}$ for all v.
Claim 2: There exists some **edge** leaving I.

Case: Only Non-Forgetful Cycles

Lemma

Environment has a strategy to ensure, for any $\vec{\lambda} \cdot \vec{v} \to \vec{\lambda'} \vec{v'}$, $\sum_{v \in I} \lambda'_v \leq \sum_{v \in I} \lambda_v - \epsilon$.

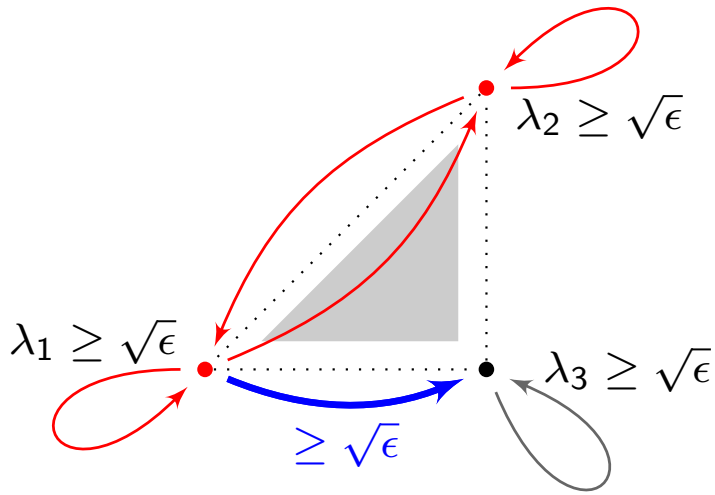


Claim 1: Environment can enforce $\lambda_v \geq \sqrt{\epsilon}$ for all v.
Claim 2: There exists some **edge** leaving I.
Claim 3: Environment can enforce edge probabilities $\geq \sqrt{\epsilon}$.

Case: Only Non-Forgetful Cycles

Lemma

Environment has a strategy to ensure, for any $\vec{\lambda} \cdot \vec{v} \to \vec{\lambda'} \vec{v'}$, $\sum_{v \in I} \lambda'_v \leq \sum_{v \in I} \lambda_v - \epsilon$.



It follows: The sum $\sum_{v \in I}$ is decreased by at least ϵ at each iteration. But $0 \leq \sum_{v \in I} \leq 1$, contradiction (no infinite run along this region).

Algorithm

Robust Controller Synthesis Algorithm

Given a timed automaton A,
- Guess a cycle of length at most exponential in the region automaton,
- Compute the folded orbit graph (on-the-fly) and check whether it is forgetful.
- Accept if it is, reject otherwise.

(If there is a forgetful cycle, there is one of at most exponential length.)

Once a forgetful lasso is found, one can compute the maximal δ and a winning strategy in time polynomial in the length of the lasso, using results from the shrinking papers.

Algorithm

Robust Controller Synthesis Algorithm

Given a timed automaton A,
- Guess a cycle of length at most exponential in the region automaton,
- Compute the folded orbit graph (on-the-fly) and check whether it is forgetful.
- Accept if it is, reject otherwise.

(If there is a forgetful cycle, there is one of at most exponential length.)

Once a forgetful lasso is found, one can compute the maximal δ and a winning strategy in time polynomial in the length of the lasso, using results from the shrinking papers.

The problem is PSPACE-complete.

Same complexity as non-emptiness in the standard semantics.

Conclusion

- A Game semantics for modelling perturbations
  disables punctual moves, and too precise strategies.
- **Thin timed automata** do become blocking under some perturbation strategies.
- Robust control $\Leftrightarrow$ **thickness** (along accepting lassos).
- Winning strategies can be computed by δ-parameterized data structures. One can compute symbolically, and adjust δ later.

- Timed games

Excess-Perturbation Game

Let $\mathcal{A}$ be a timed automaton and $\delta > 0$.

"Excess" Perturbation Game: Controller vs Environment.

At any state (ℓ, ν),

Excess-Perturbation Game

Let $\mathcal{A}$ be a timed automaton and $\delta > 0$.

"Excess" Perturbation Game: Controller vs Environment.

At any state (ℓ, ν),

1. Controller chooses a delay $d \geq \delta$, and an edge $\ell \xrightarrow{g,R} \ell'$, such that $\nu + d \models g$,

Excess-Perturbation Game

Let $\mathcal{A}$ be a timed automaton and $\delta > 0$.

"Excess" Perturbation Game: Controller vs Environment.

At any state (ℓ, ν),

1. Controller chooses a delay $d \geq \delta$, and an edge $\ell \xrightarrow{g,R} \ell'$, such that $\nu + d \models g$,
2. Environment chooses $d' \in [d - \delta, d + \delta]$, (we can have $d' \not\models g$)

Excess-Perturbation Game

Let $\mathcal{A}$ be a timed automaton and $\delta > 0$.

"Excess" Perturbation Game: Controller vs Environment.

At any state (ℓ, ν),

1. Controller chooses a delay $d \geq \delta$, and an edge $\ell \xrightarrow{g,R} \ell'$, such that $\nu + d \models g$,
2. Environment chooses $d' \in [d - \delta, d + \delta]$, (we can have $d' \not\models g$)
3. New state is $(\ell', (\nu + d')[R \leftarrow 0])$.

Excess-Perturbation Game

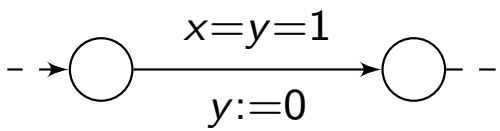
Let $\mathcal{A}$ be a timed automaton and $\delta > 0$.

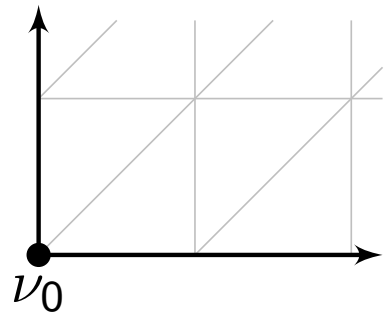
"Excess" Perturbation Game: Controller vs Environment.

At any state (ℓ, ν),

1. Controller chooses a delay $d \geq \delta$, and an edge $\ell \xrightarrow{g,R} \ell'$, such that $\nu + d \models g$,
2. Environment chooses $d' \in [d - \delta, d + \delta]$, (we can have $d' \not\models g$)
3. New state is $(\ell', (\nu + d')[R \leftarrow 0])$.

For $\delta > 0$,





Excess-Perturbation Game

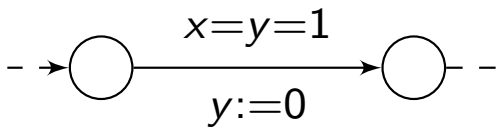
Let $\mathcal{A}$ be a timed automaton and $\delta > 0$.

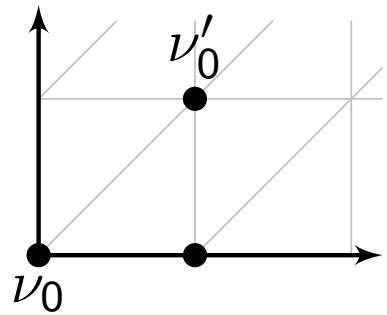
"Excess" Perturbation Game: Controller vs Environment.

At any state (ℓ, ν),

1. Controller chooses a delay $d \geq \delta$, and an edge $\ell \xrightarrow{g,R} \ell'$, such that $\nu + d \models g$,
2. Environment chooses $d' \in [d - \delta, d + \delta]$, (we can have $d' \not\models g$)
3. New state is $(\ell', (\nu + d')[R \leftarrow 0])$.

For $\delta > 0$,





Excess-Perturbation Game

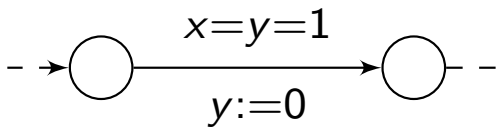
Let $\mathcal{A}$ be a timed automaton and $\delta > 0$.

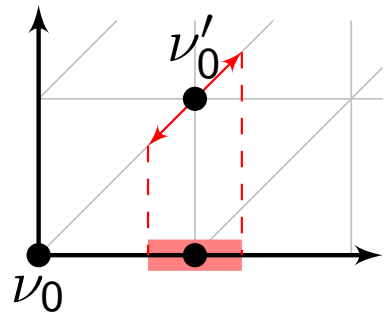
"Excess" Perturbation Game: Controller vs Environment.

At any state (ℓ, ν),

1. Controller chooses a delay $d \geq \delta$, and an edge $\ell \xrightarrow{g,R} \ell'$, such that $\nu + d \models g$,
2. Environment chooses $d' \in [d - \delta, d + \delta]$, (we can have $d' \not\models g$)
3. New state is $(\ell', (\nu + d')[R \leftarrow 0])$.

For $\delta > 0$,





Excess-Perturbation Game - 2

Excess Perturbation Semantics

The model is often **simpler**: one can use equalities, not think about error intervals.

A synthesized strategy takes into account **additional** behaviors.

VS

Conservative Perturbation Semantics

The model already contains **intervals** for timing constraints.

No additional behavior is expected.

The problem is **convergence**: the synthesized strategy must ensure liveness.

Pick one...

Reachability in Excess-Perturbation Game

(Parameterized) Robust Reachability

Given a timed automaton $\mathcal{A}$ and target location ℓ,
Does there exist $\delta_0 > 0$, such that Controller has a strategy reaching ℓ in the excess semantics for all $\delta \in [0, \delta_0)$?

Main result

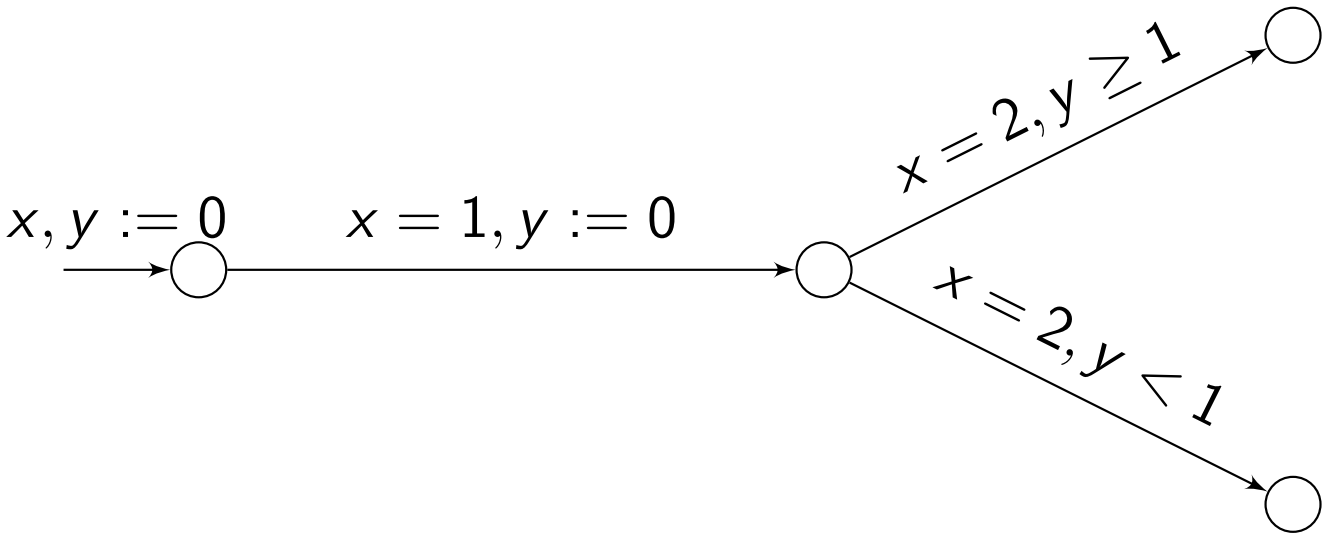
Robust reachability in excess semantics is EXPTIME-complete.

EXPTIME-hardness

Usual semantics in TA can encode reachability in linearly bounded Turing machines (PSPACE-complete).

Robust semantics in TA can encode reachability in **alternating** linearly bounded Turing machines (EXPTIME-complete).

The encoding is similar as in the PSPACE-hardness proofs for TA.
Alternation: simulated by the perturbating player



Conclusion

- The excess-perturbation game semantics is harder: reachability is EXPTIME-complete.
- NB: Convergence is not a problem for reachability (finite runs).

- General timed games in both semantics.
- Zone-based algorithms, efficient implementations.