Robustness in Timed Systems

Pierre-Alain Reynier

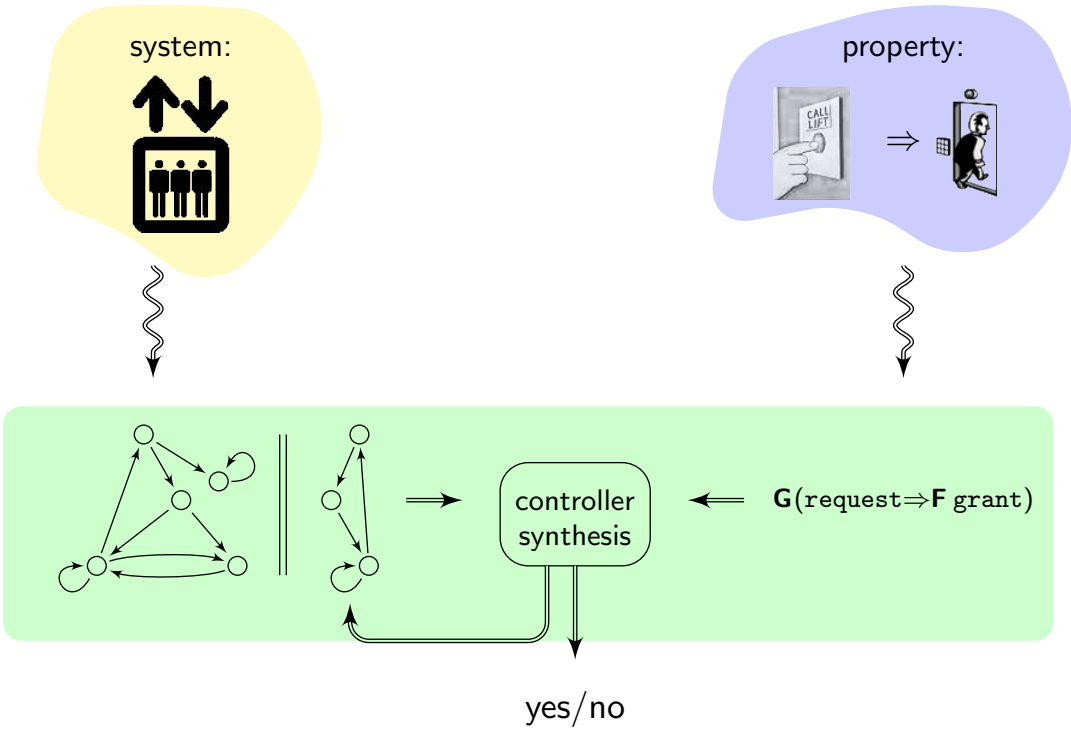
LIF, Aix-Marseille University & CNRS

Overview

Overview

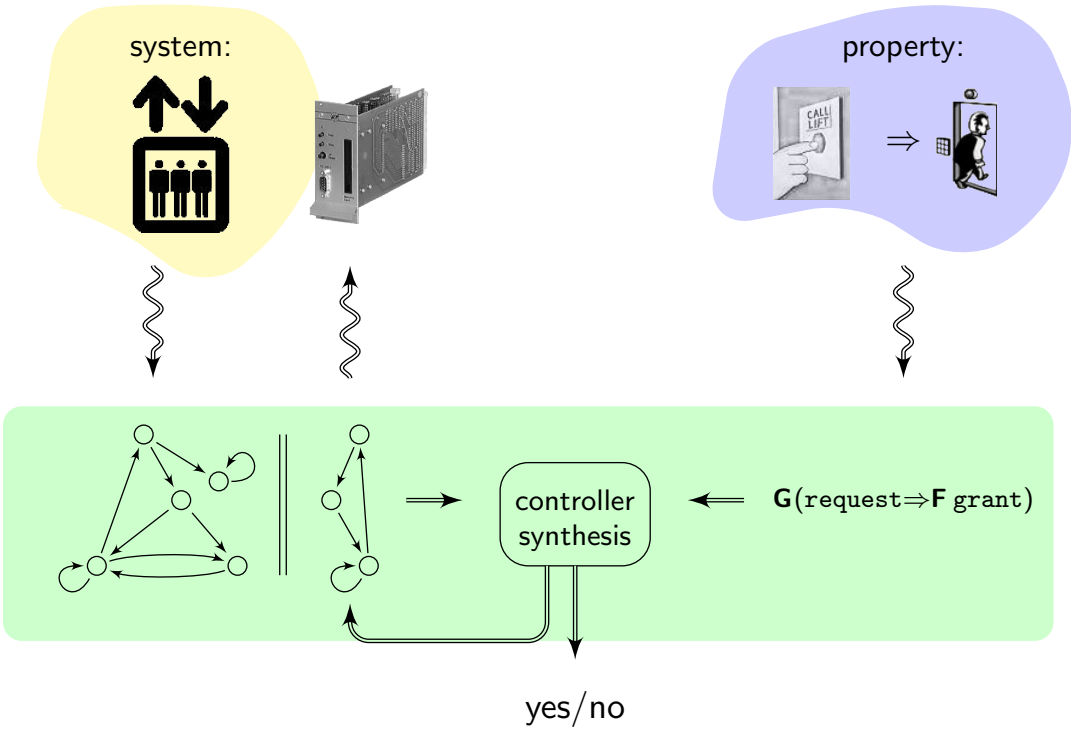
Model-Checking/Control for Timed Systems

- Model-Checking/Control:



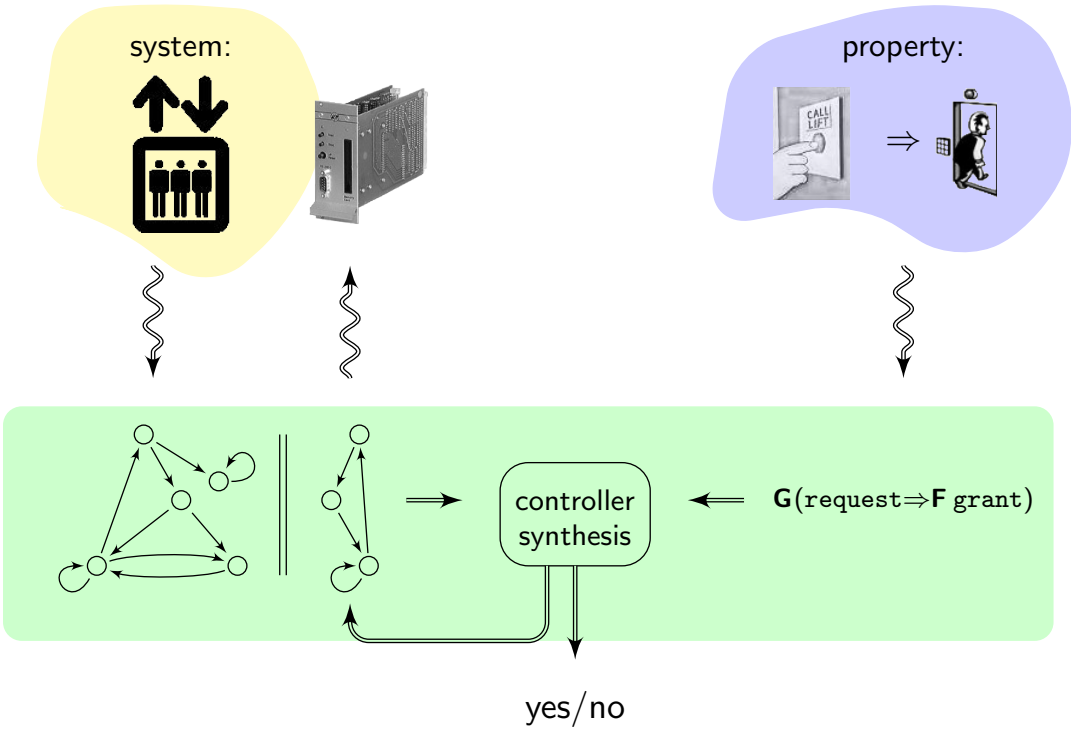
Model-Checking/Control for Timed Systems

- Model-Checking/Control:



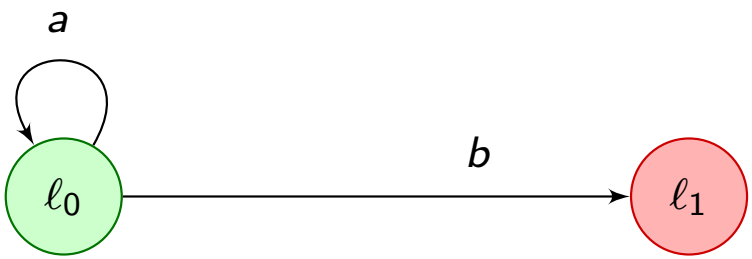
Model-Checking/Control for Timed Systems

- Model-Checking/Control:



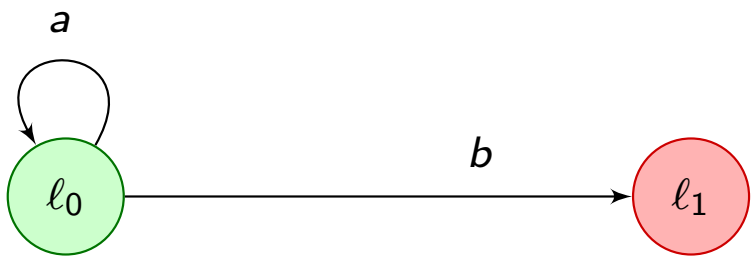
- "timed systems": explicit notion of time which allows a quantitative evaluation of time elapsing.
  "A signal is always followed by an alarm."
  "A signal is always followed by an alarm within 4 time units."

Timed Automata [AD90]



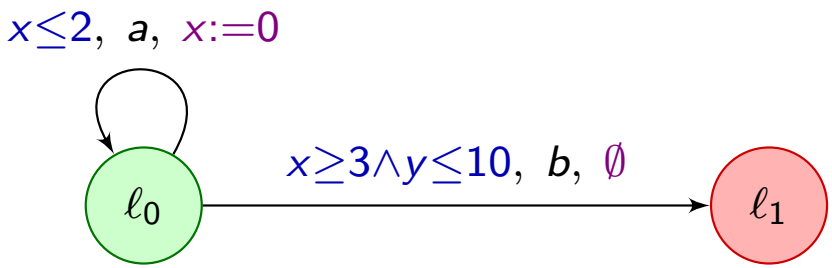
- A finite control structure

Timed Automata [AD90]



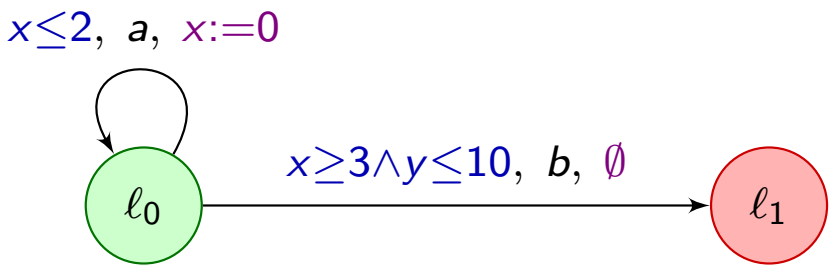
- A finite control structure
- A finit set of clocks (dense-time variables)

Timed Automata [AD90]



- A finite control structure
- A finit set of clocks (dense-time variables)
- Clock constraints + updates on transitions

Timed Automata [AD90]



- A finite control structure
- A finit set of clocks (dense-time variables)
- Clock constraints + updates on transitions
- Configurations are pairs $(\ell, \nu) \in \text{Loc} \times \mathbb{R}^X_{\geq 0}$
- Two kinds of moves:

  Delay: given $d \in \mathbb{R}_{\geq 0}$, we have $(\ell, \nu) \xrightarrow{d} (\ell, \nu + d)$

  Discrete: $\ell \xrightarrow{g,a,r} \ell'$ and $\nu \models g$ implies $(\ell, \nu) \xrightarrow{a} (\ell', \nu[r \leftarrow 0])$

Verification of timed systems

Other models: (priced) timed games, time Petri nets...

Symbolic techniques :

- region graph
- zones, represented by Difference Bound Matrices (DBMs)

Widely used, several case studies, advanced tools (Uppaal, Uppaal TiGa)

Key issues:

- develop finite time-abstract bisimulations (e.g. corner point abstraction)
- develop symbolic algorithms

Implementation of timed systems

Semantics of models are mathematical idealizations

Infinitely punctual : Exact synchronization is required when composing several TAs.

Implementation of timed systems

Semantics of models are mathematical idealizations

Infinitely punctual : Exact synchronization is required when composing several TAs.

Infinitely precise : Different clocks are assumed to increase at the same rate in both the controller and the system.

Implementation of timed systems

Semantics of models are mathematical idealizations

Infinitely punctual : Exact synchronization is required when composing several TAs.

Infinitely precise : Different clocks are assumed to increase at the same rate in both the controller and the system.

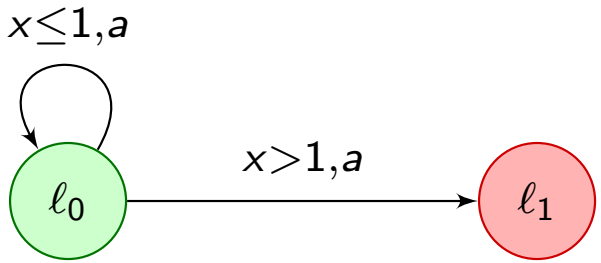
Infinitely fast : It may happen, for instance, that a TA will have to perform actions at time n and $n + 1/n$, for all n.

In practice, a processor is digital and imprecise.

Once the correction of the model is proven, what do we know on its implementability?

Implementability of Timed Automata

- Zeno behaviors

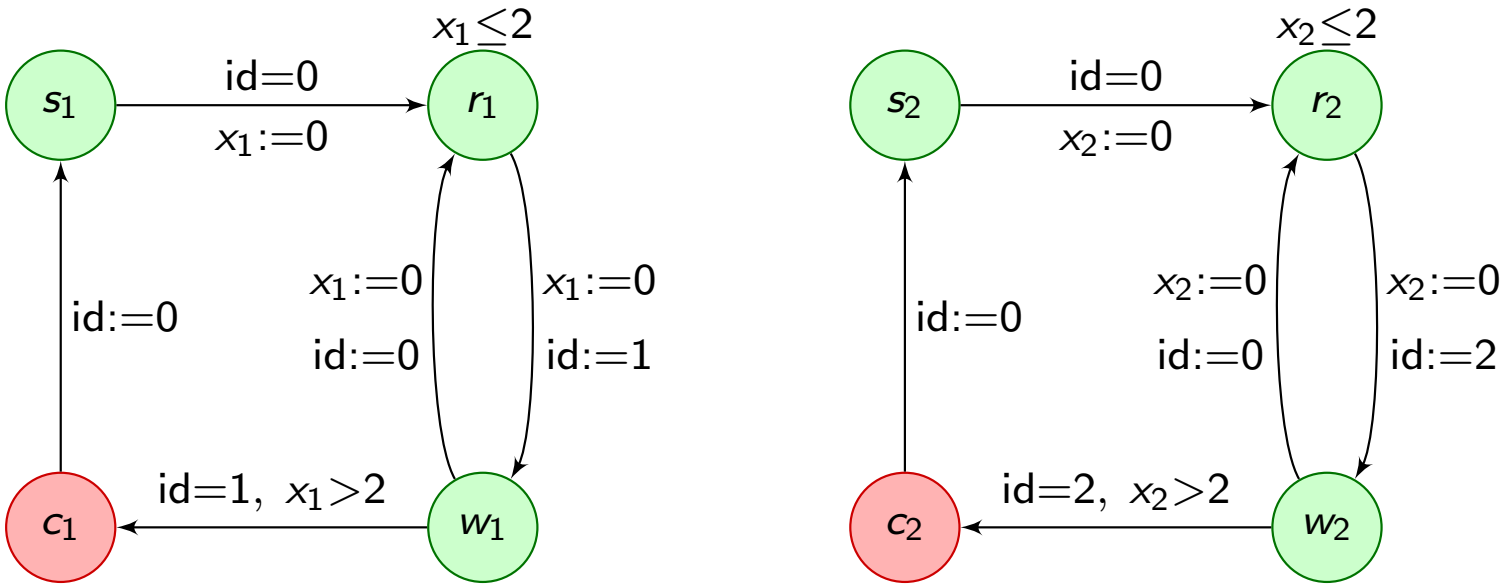


Executions of the form $a, d_1, a, d_2, a, d_3 \ldots$

Stays in green state iff delays $(d_i)_i$ verify $\sum_i d_i \leq 1$

Implementability of Timed Automata

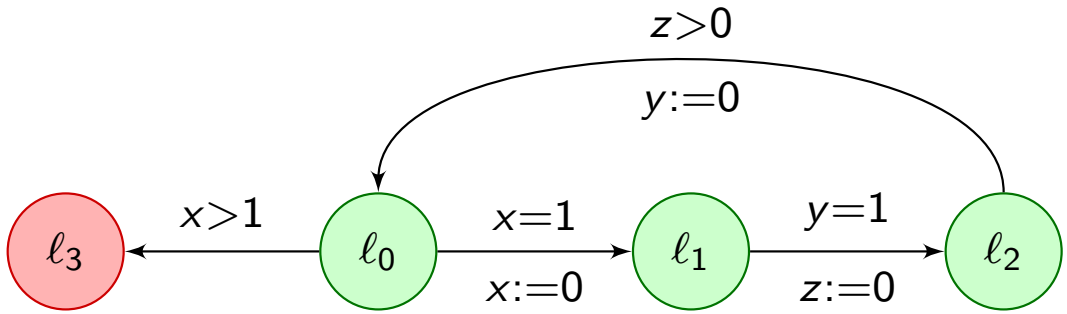
- Zeno behaviors
- Strict guards: Fischer's protocol [KLL+97]



Mutual exclusion can be proven formally
Becomes false if guards $x_i > 2$ become non-strict

Implementability of Timed Automata

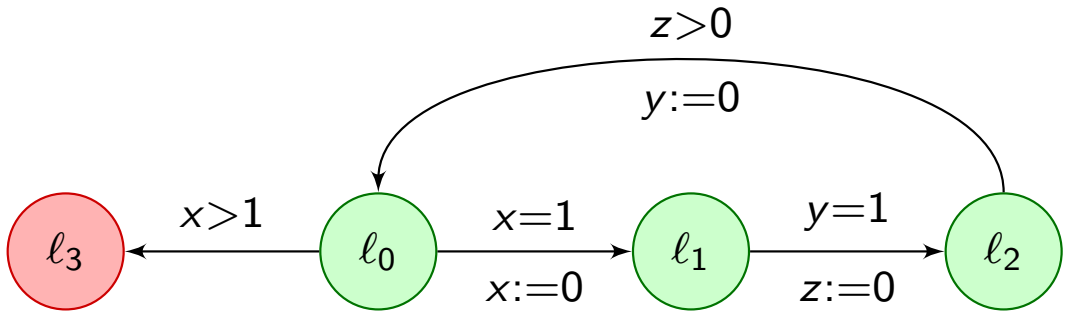
- Zeno behaviors
- Strict guards: Fischer's protocol [KLL+97]
- "Fragile" controllers [CHR02]



In order to loop forever in green states, the global time of the n-th entry in location ℓ_0 is of the form $n + \delta_n$ such that $\sum_i \delta_i < 1$.

Implementability of Timed Automata

- Zeno behaviors
- Strict guards: Fischer's protocol [KLL+97]
- "Fragile" controllers [CHR02]



In order to loop forever in green states, the global time of the n-th entry in location ℓ_0 is of the form $n + \delta_n$ such that $\sum_i \delta_i < 1$.

➜ Implementability is related to the tolerance to clock imprecisions.

Our goal

Add robustness to the theory of timed automata

- Understand the "real" system behind the mathematical model
- Provide tools to automatically develop robust correct systems
- Notion of robustness may depend on the application area

➜ We present results on a framework developed recently

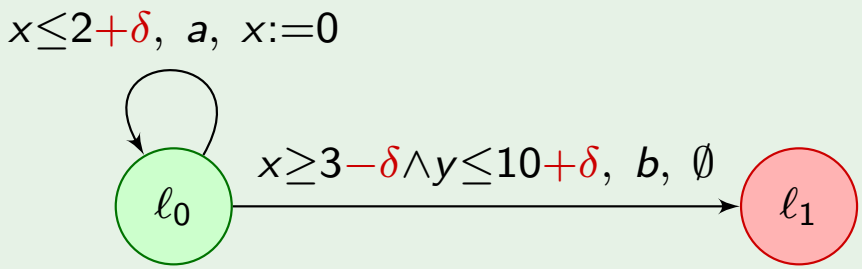
➜ Focus on perturbations on time measurements and jitter

Guard enlargment in Timed Automata

Enlarge guards

$a \leq x \leq b \quad \leadsto \quad a-\delta \leq x \leq b+\delta$

Example



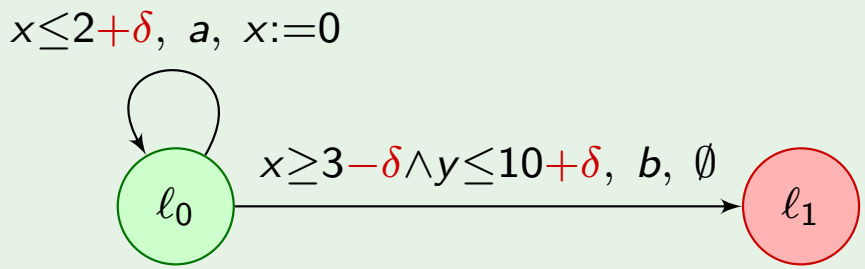
➜ This yields a parameterized model $\mathcal{A}_\delta$

Guard enlargment in Timed Automata

Enlarge guards

$a \leq x \leq b \quad \leadsto \quad a-\delta \leq x \leq b+\delta$

Example



➜ This yields a parameterized model $\mathcal{A}_\delta$

Problem considered: Does there exist $\delta > 0$ such that $\mathcal{A}_\delta$ is "correct"?

Relevance of this approach

- This is a worst-case approach
- Direct link with implementability [DDR05]
  One can define a program semantics such that whenever $3\delta_L + 4\delta_P < \delta$,

$$\mathcal{A} \subseteq \text{program}_{\delta_L,\delta_P}(\mathcal{A}) \subseteq \mathcal{A}_\delta$$

  where δ_L, δ_P are characteristics of the implementation platform

Relevance of this approach

- This is a worst-case approach
- Direct link with implementability [DDR05]
  One can define a program semantics such that whenever $3\delta_L + 4\delta_P < \delta$,

$$\mathcal{A} \subseteq \text{program}_{\delta_L,\delta_P}(\mathcal{A}) \subseteq \mathcal{A}_\delta$$

  where δ_L, δ_P are characteristics of the implementation platform

Lemma (Monotony)

Given $\delta_1 \leq \delta_2$, $\mathcal{A}_{\delta_2}$ simulates $\mathcal{A}_{\delta_1}$.

➜ Faster is better: if every behaviour of $\mathcal{A}_{\delta_2}$ is correct, then so are those of $\mathcal{A}_{\delta_1}$

Relevance of this approach (2)

- Clock drift is simulated by guard enlargement
- Timed automata with one parameter:
  - undecidable in general (even emptiness)
  - in our case, decidable problems
- For all the problems we have studied, the theoretical complexity of the "robust problem" is the same as that of the "non-robust problem".
- Fixed parameter value
  - reduces to a problem on standard timed automata
  - choice of δ depends on the implementation platform
  - $\delta << $ constants in $\mathcal{A} \Rightarrow$ state explosion

Guard enlargment in Timed Automata: assumptions

In the sequel:

- all guards and invariants only involve non-strict inequalities.

This is not a restriction since

$$[a - \Delta/2; b + \Delta/2] \subseteq (a - \Delta, b + \Delta).$$

$\rightsquigarrow$ we consider only closed regions.

Guard enlargment in Timed Automata: assumptions

In the sequel:

- all guards and invariants only involve non-strict inequalities.

This is not a restriction since

$$[a - \Delta/2; b + \Delta/2] \subseteq (a - \Delta, b + \Delta).$$

$\rightsquigarrow$ we consider only closed regions.

- we may assume progress cycles:
  along any cycle of the region graph, all the clocks are reset.

This is a real restriction, but it is weaker than the "strongly non-Zeno" restriction.

Overview

Robustness and bounded horizon

Guard enlargement matters only for unbounded executions.

Lemma ([DDMR04])

For any $n \in \mathbb{N}$ and any $\epsilon > 0$, there exists δ_0 such that $\forall \delta \leq \delta_0$,

$$\forall \rho \in \mathit{Runs}(\mathcal{A}_\delta). \exists \rho' \in \mathit{Runs}(\mathcal{A}) \mid d_\infty(\rho, \rho') \leq \epsilon$$

d_∞: ρ and ρ' follow the same locations, and at each step valuations are close w.r.t. infinite norm.

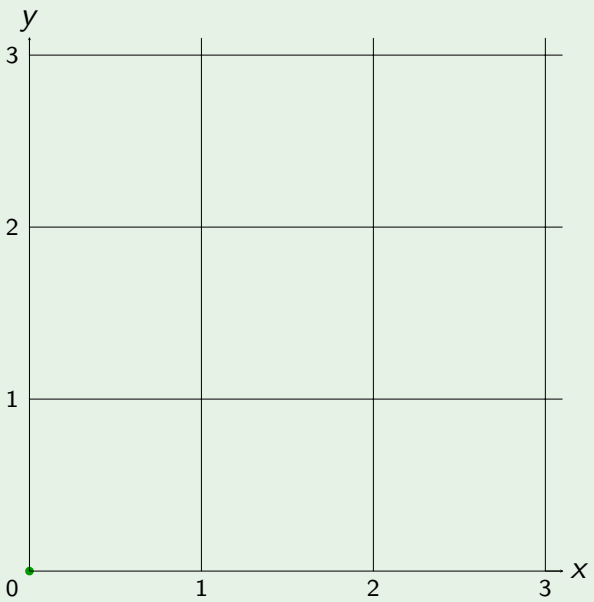
Proof based on Parametric DBMs = DBMs with parametric entries.

$$\begin{array}{rrcl} & x_1 & \geq & 3 - 2\delta \\ \wedge & x_2 & \leq & 1 + 4\delta \\ \wedge & x_1 - x_2 & \leq & 2 + \delta \end{array} \qquad \begin{array}{c} \\ x_0 \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \begin{pmatrix} 0 & -3+2\delta & +\infty \\ +\infty & 0 & 2+\delta \\ 1+4\delta & +\infty & 0 \end{pmatrix} \end{array}$$

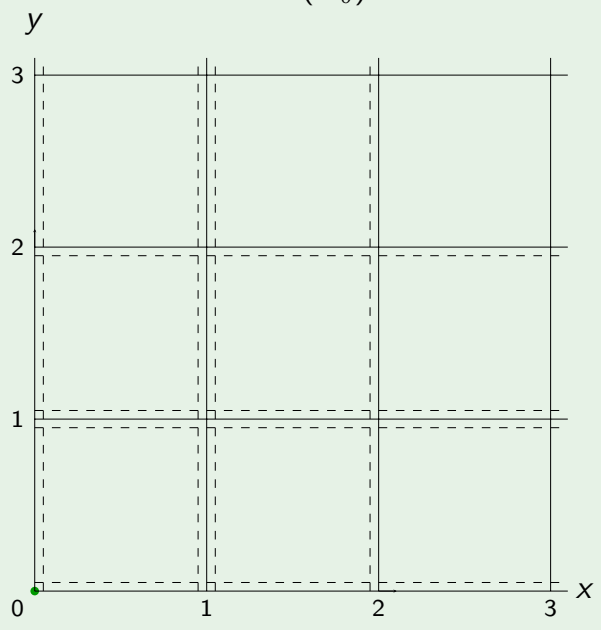
New behaviours on an example

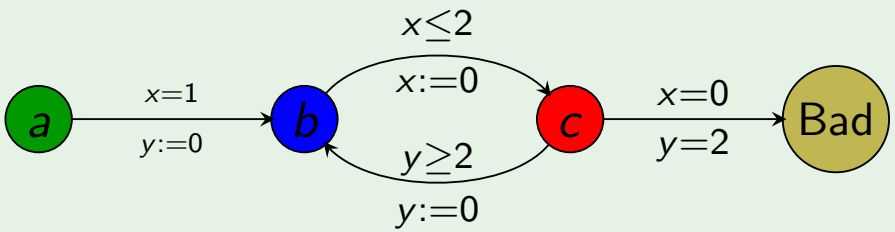
Example ([Puri00])

Reach($\mathcal{A}$)



Reach($\mathcal{A}_\delta$)

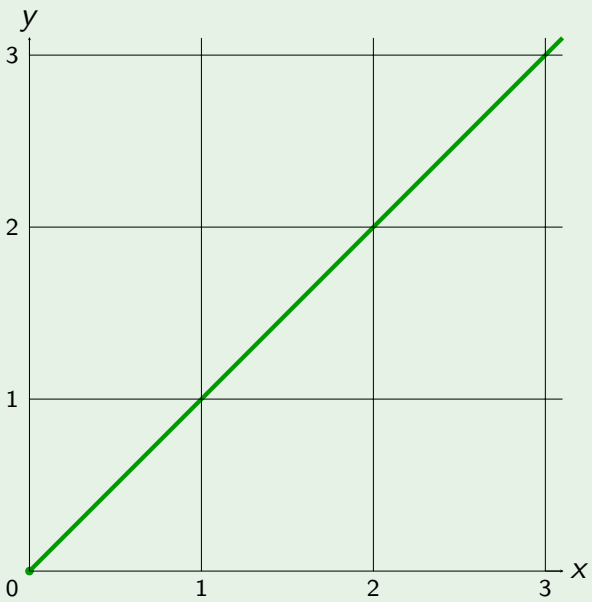




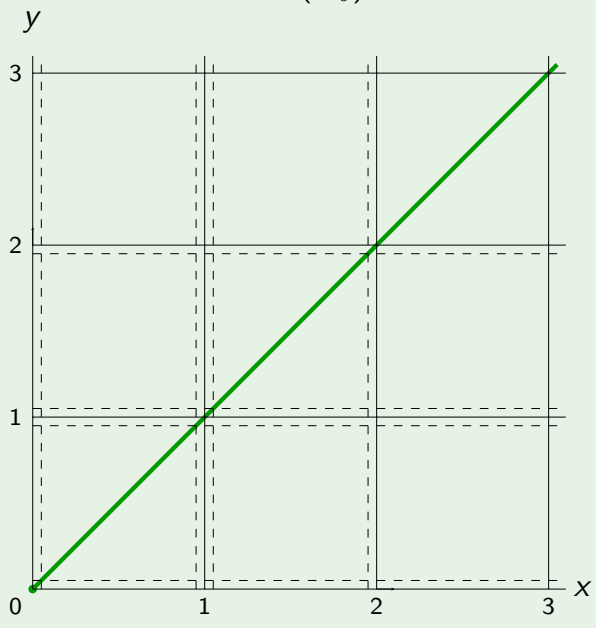
New behaviours on an example

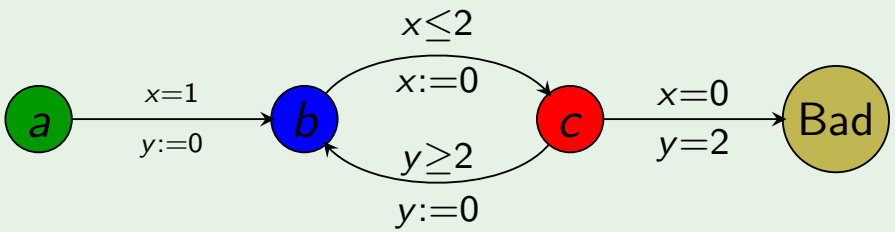
Example ([Puri00])

Reach($\mathcal{A}$)



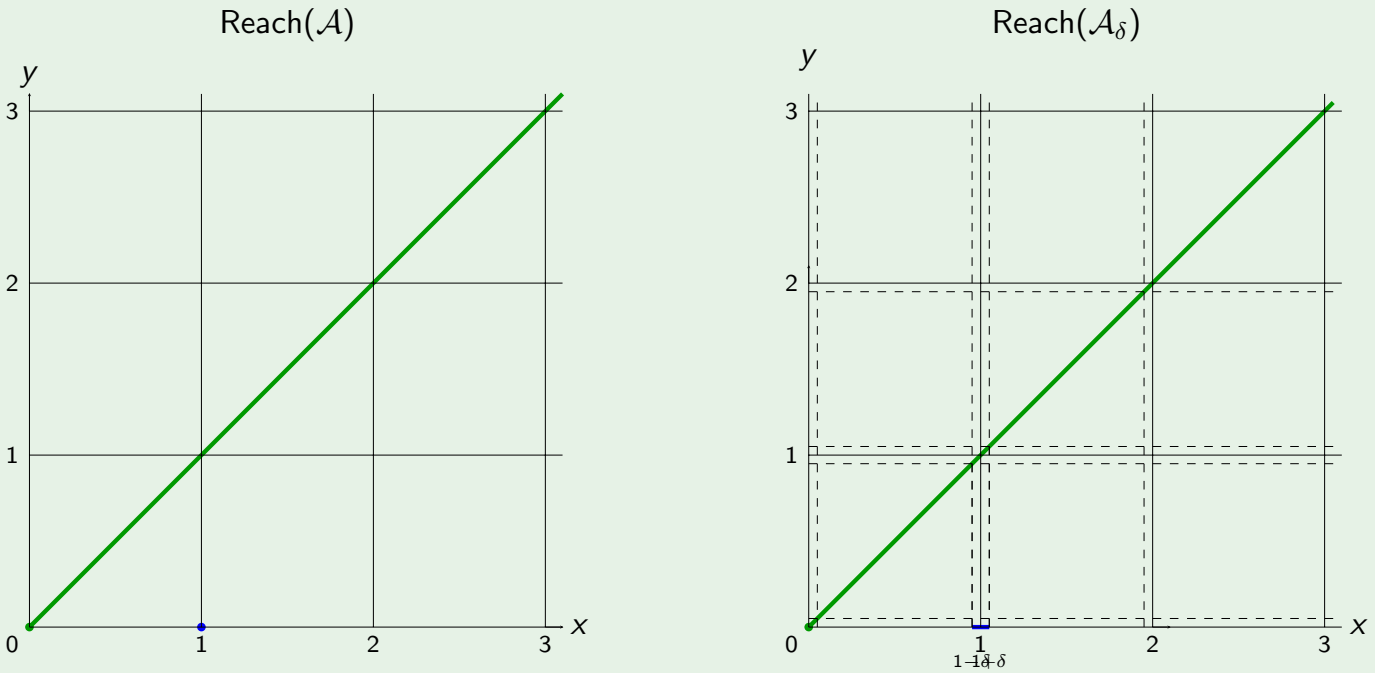
Reach($\mathcal{A}_\delta$)

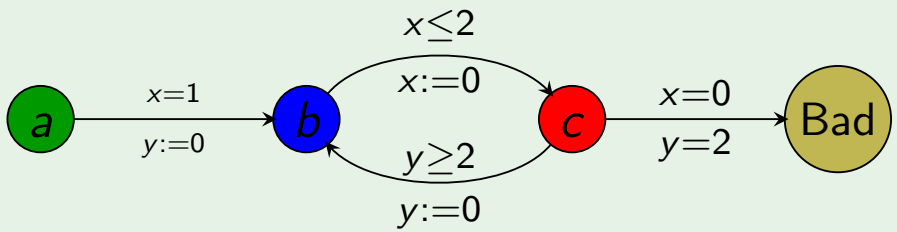




New behaviours on an example

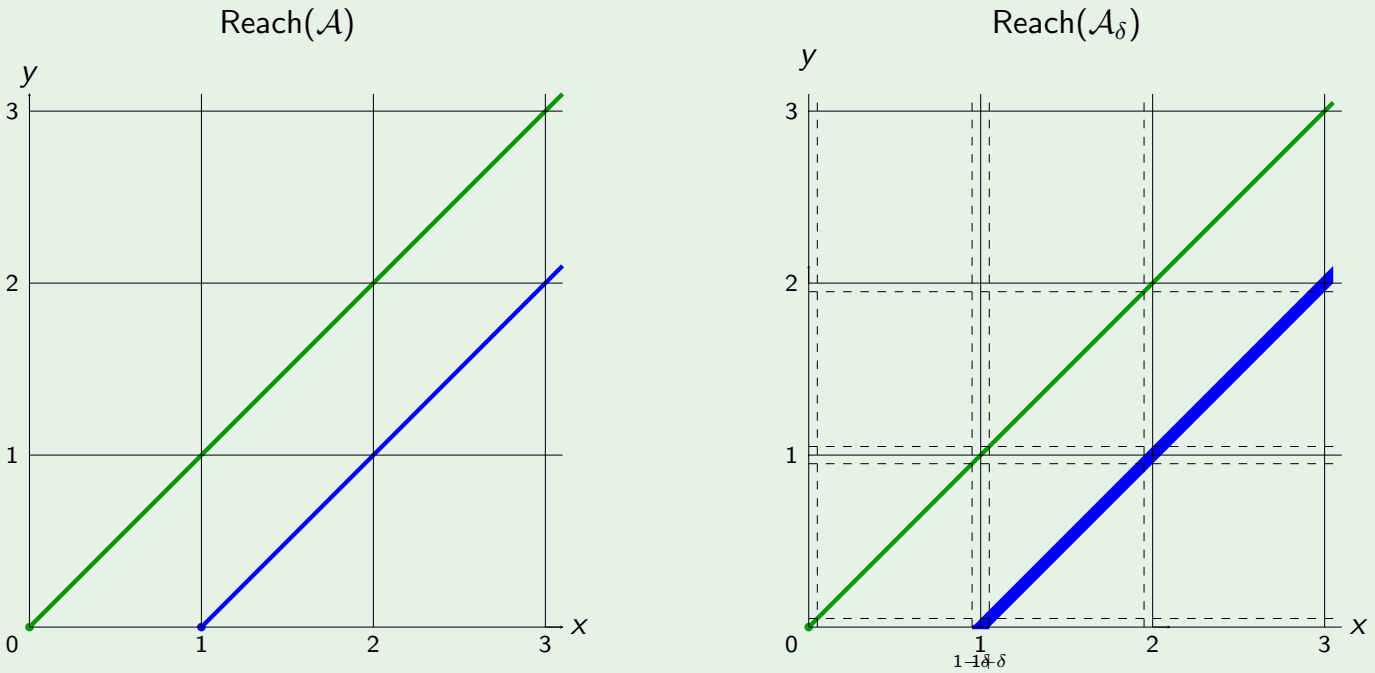
Example ([Puri00])

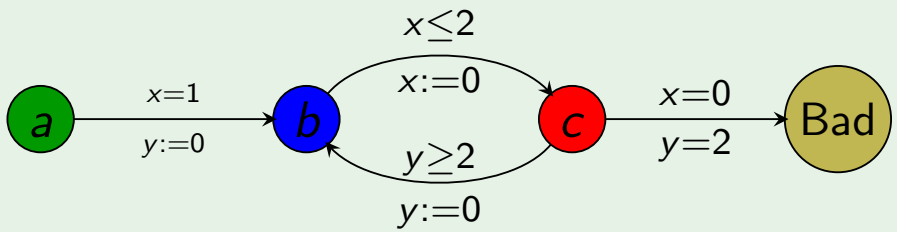




New behaviours on an example

Example ([Puri00])

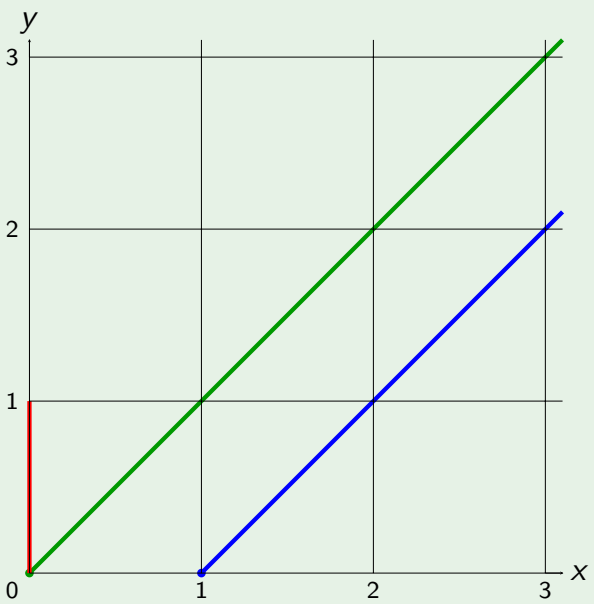




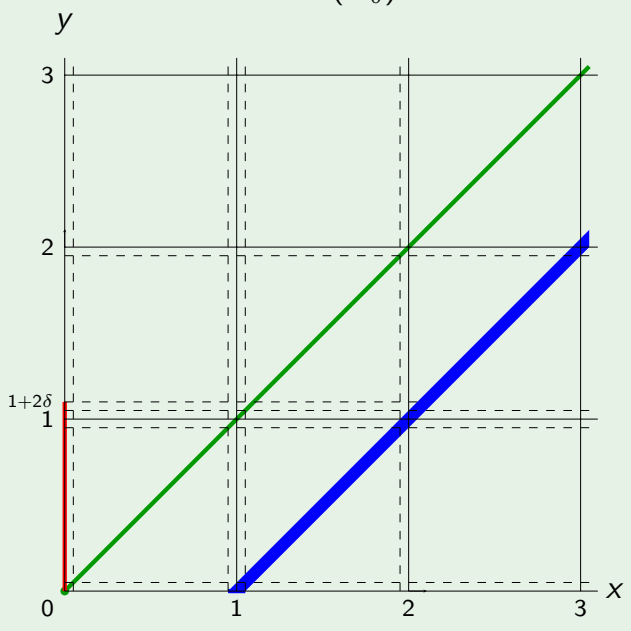
New behaviours on an example

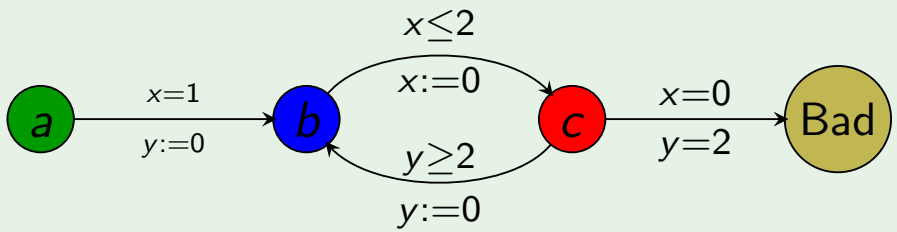
Example ([Puri00])

Reach($\mathcal{A}$)



Reach($\mathcal{A}_\delta$)

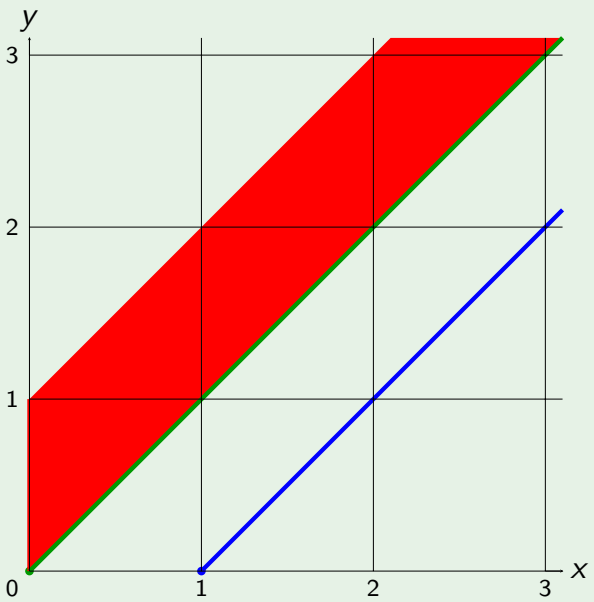




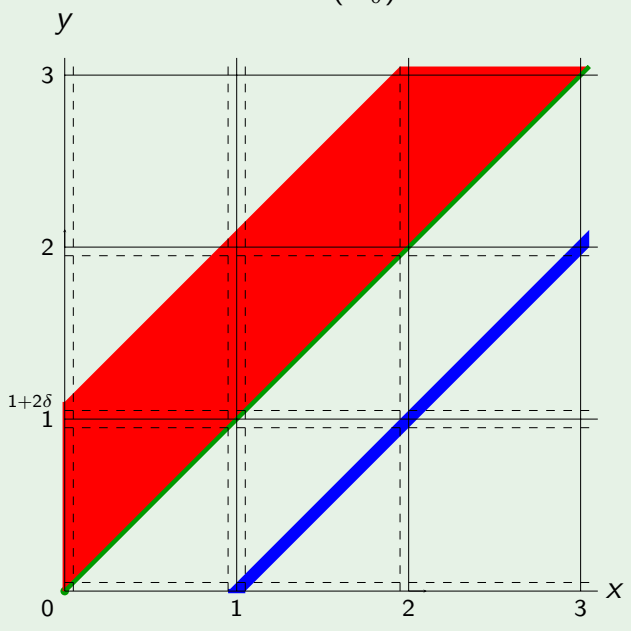
New behaviours on an example

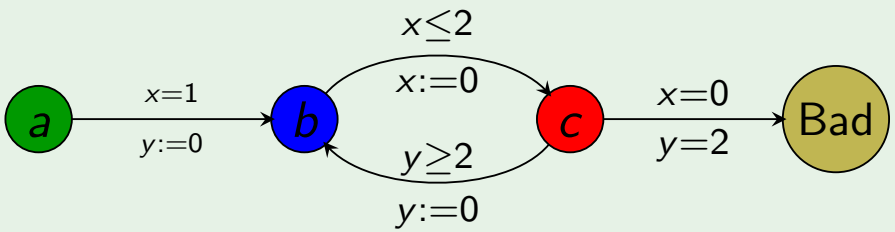
Example ([Puri00])

Reach($\mathcal{A}$)



Reach($\mathcal{A}_\delta$)

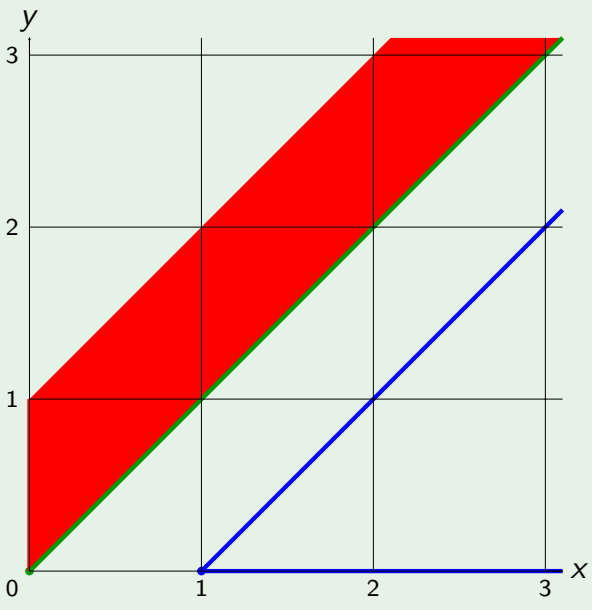




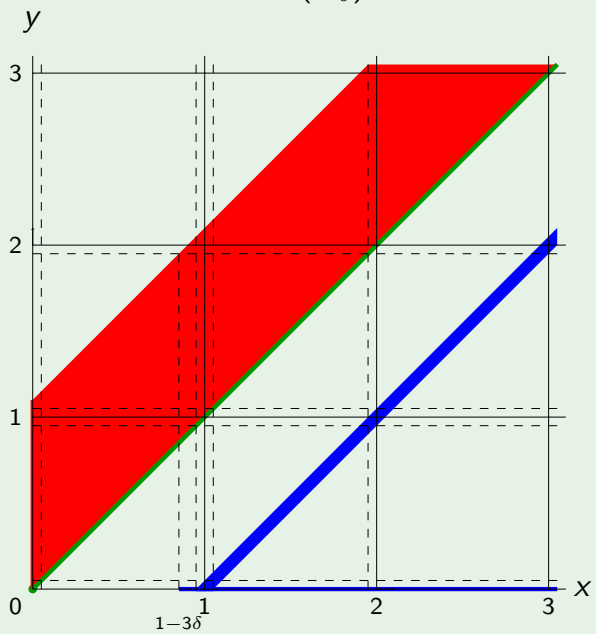
New behaviours on an example

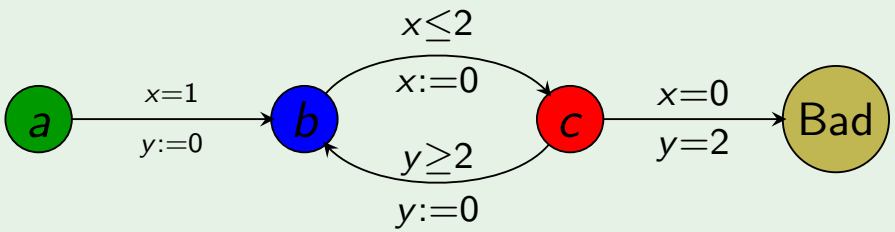
Example ([Puri00])

Reach($\mathcal{A}$)



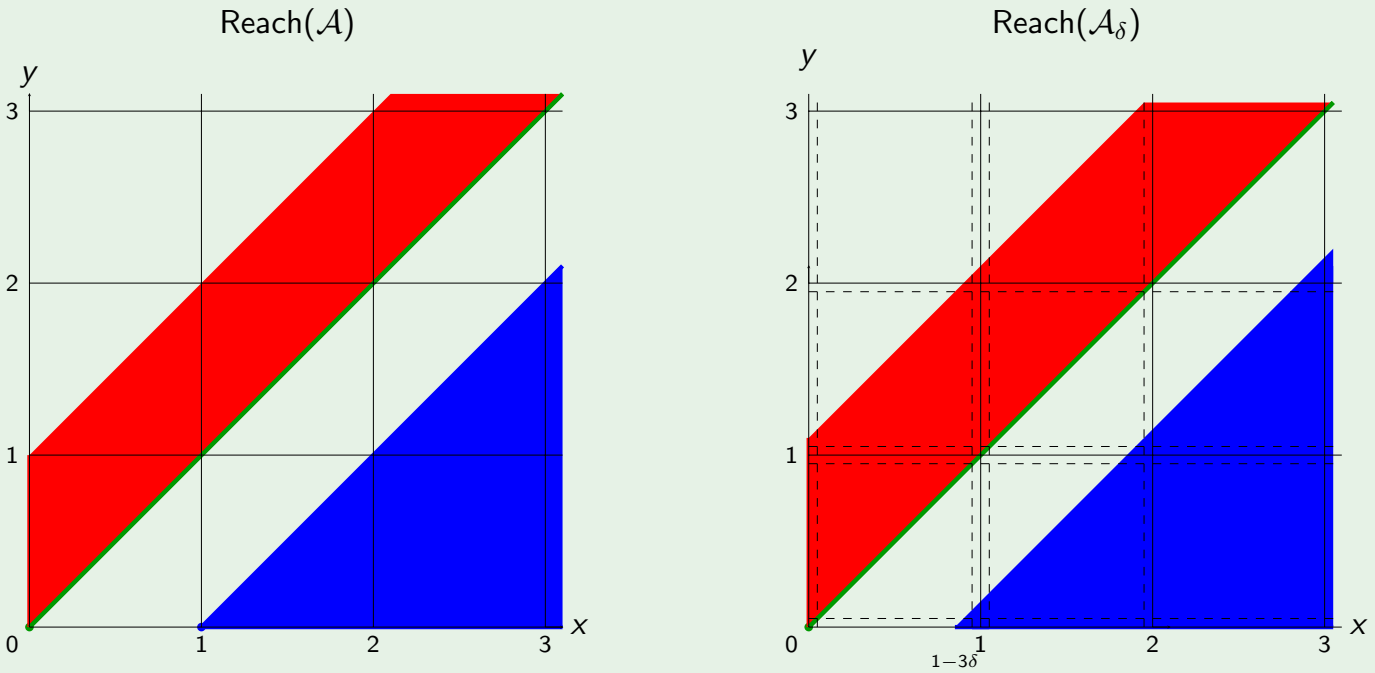
Reach($\mathcal{A}_\delta$)

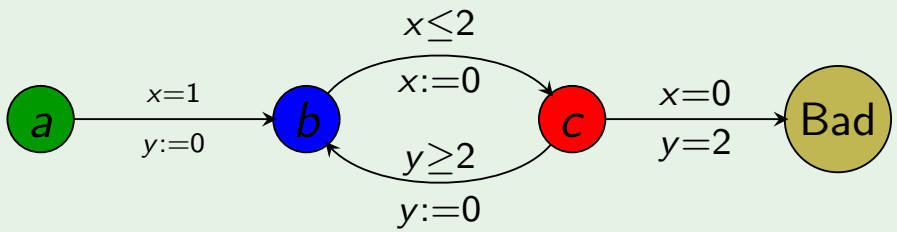




New behaviours on an example

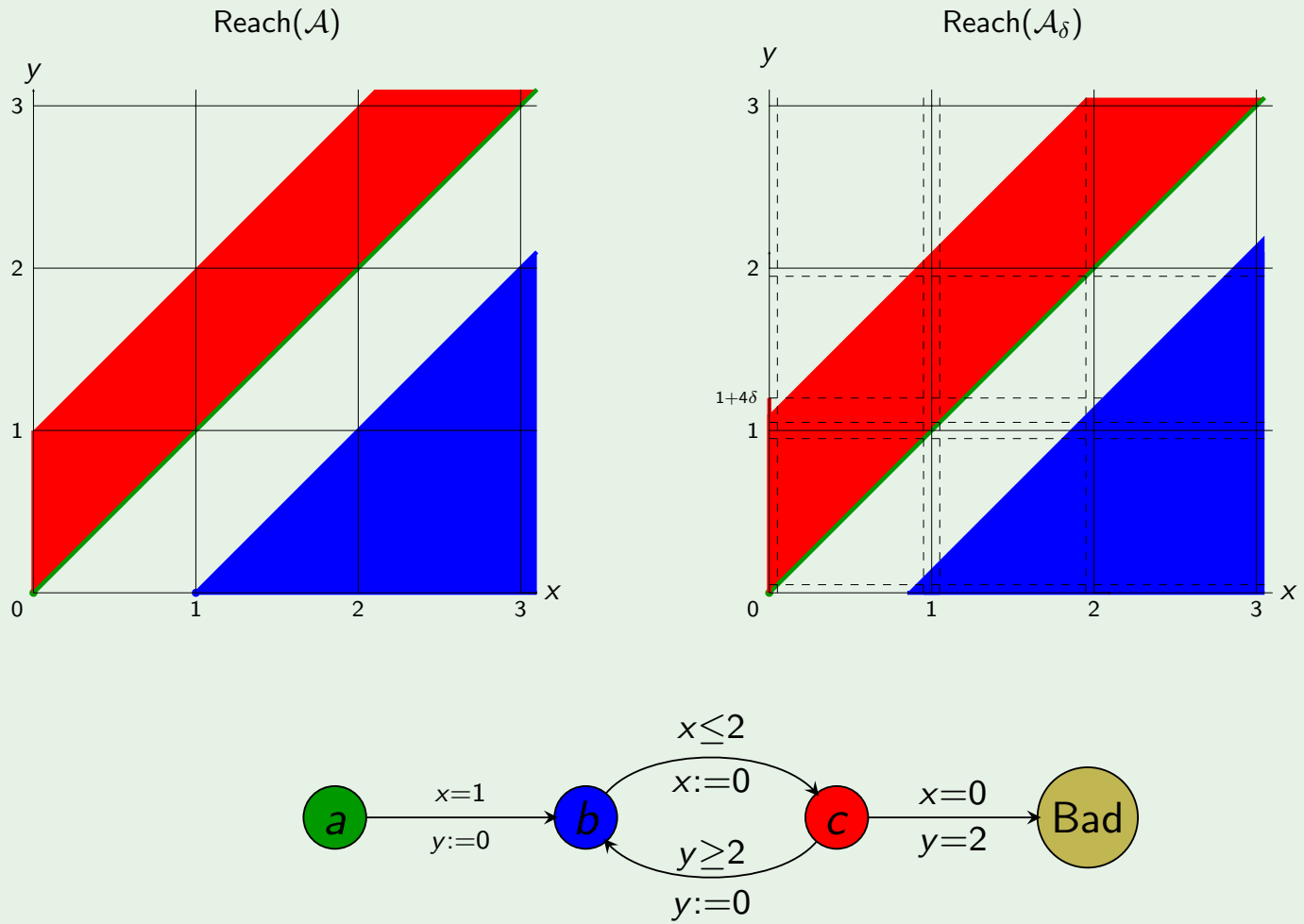
Example ([Puri00])





New behaviours on an example

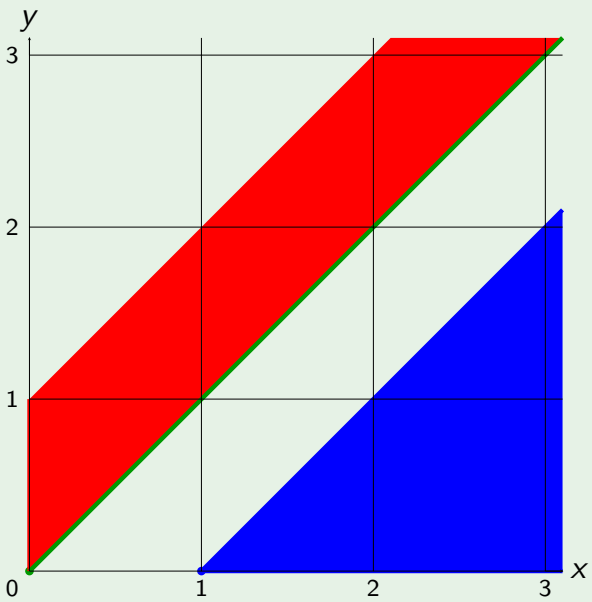
Example ([Puri00])



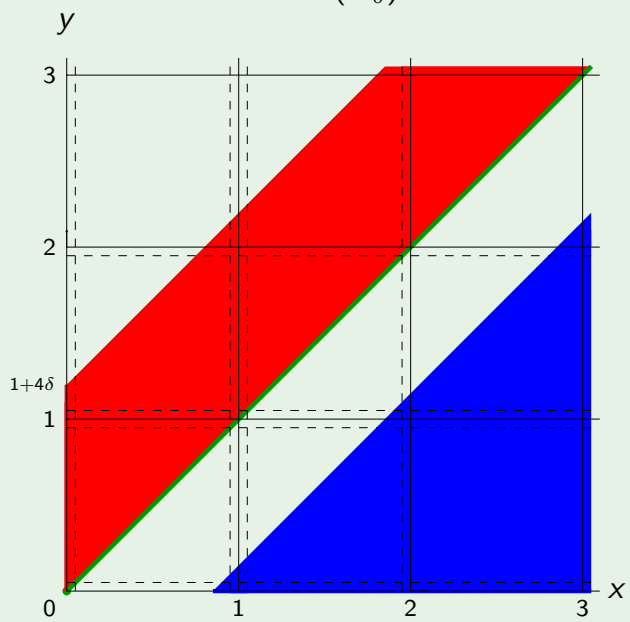
New behaviours on an example

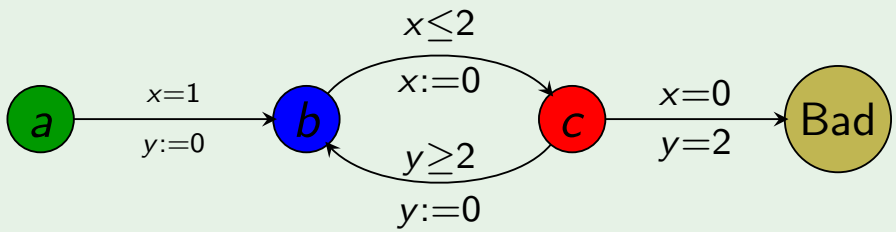
Example ([Puri00])

Reach($\mathcal{A}$)



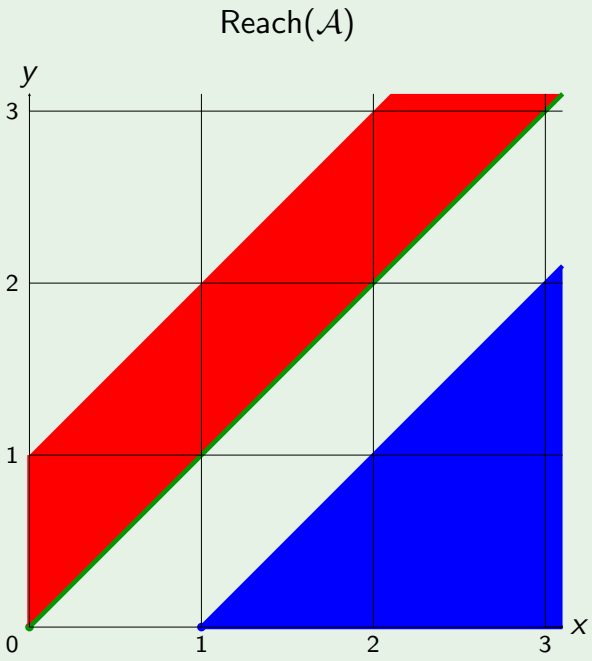
Reach($\mathcal{A}_\delta$)

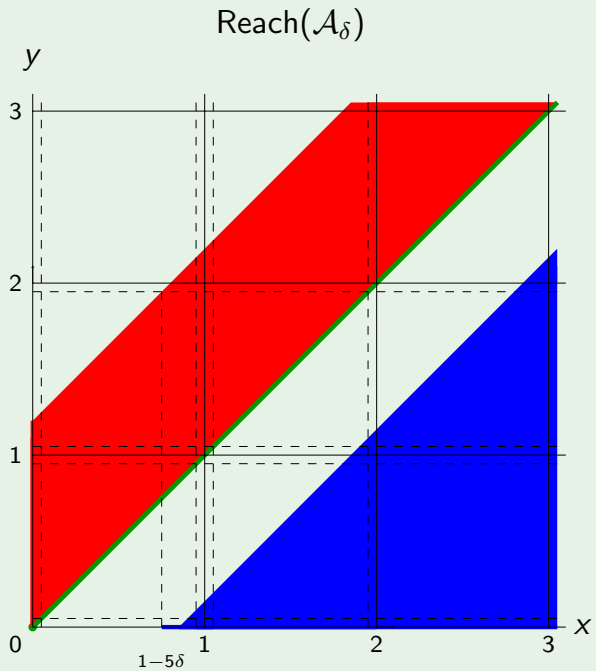


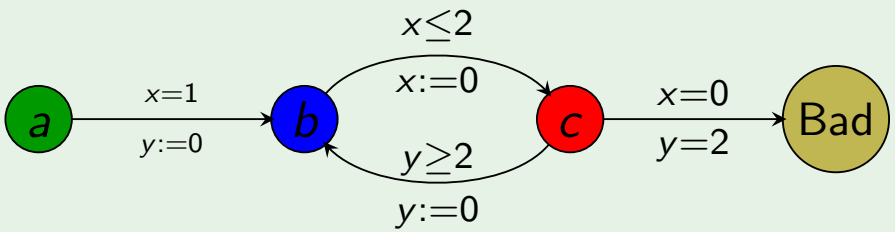


New behaviours on an example

Example ([Puri00])



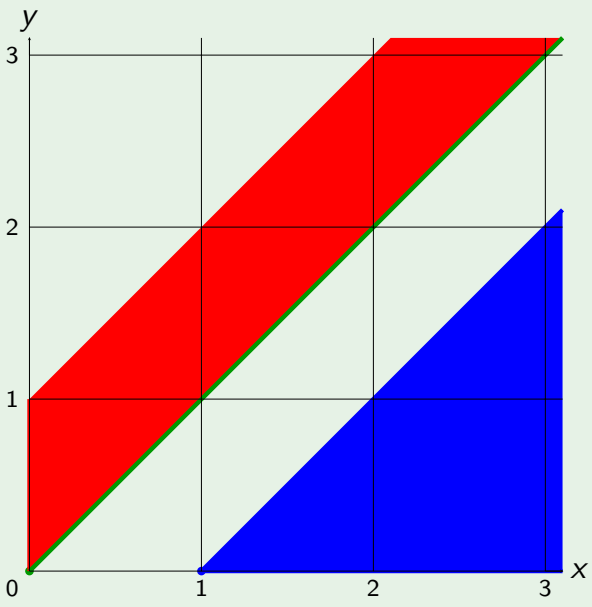




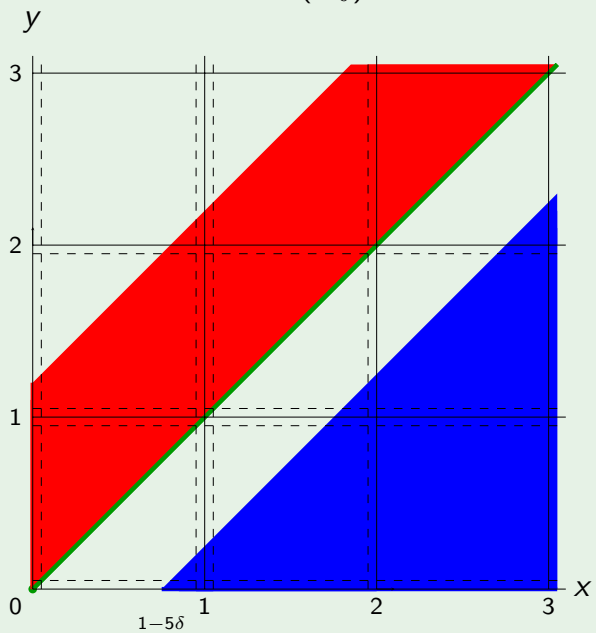
New behaviours on an example

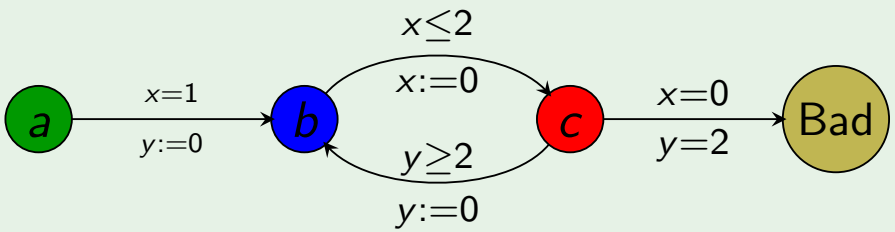
Example ([Puri00])

Reach($\mathcal{A}$)



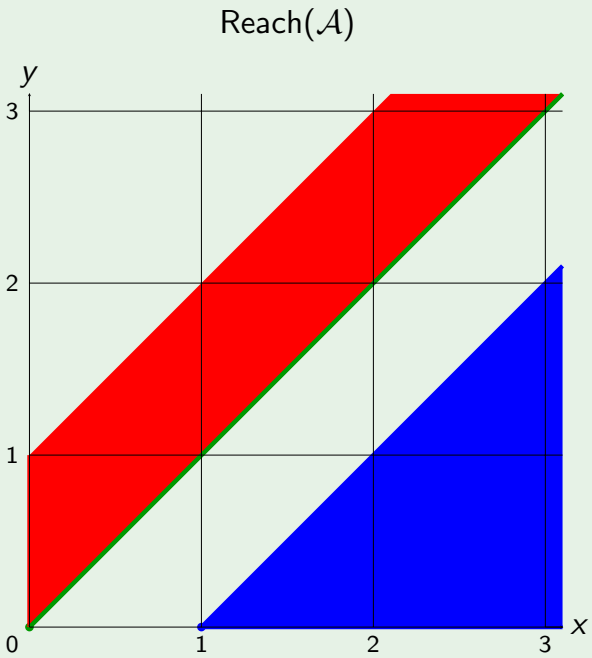
Reach($\mathcal{A}_\delta$)

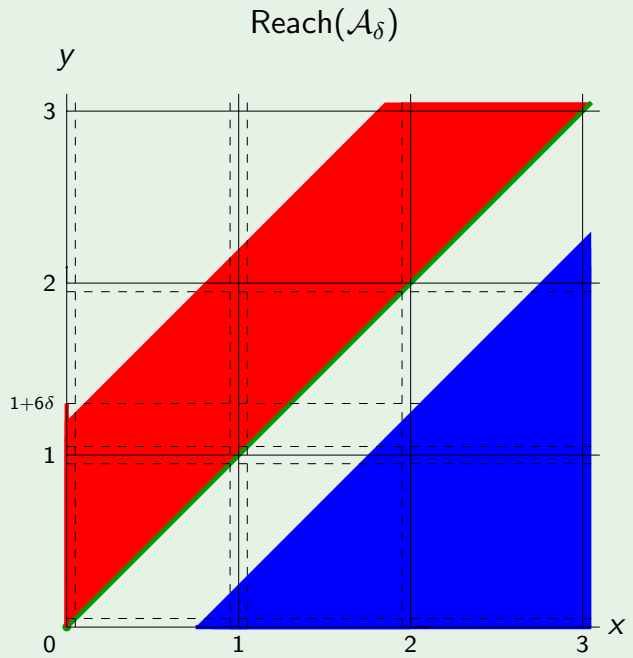


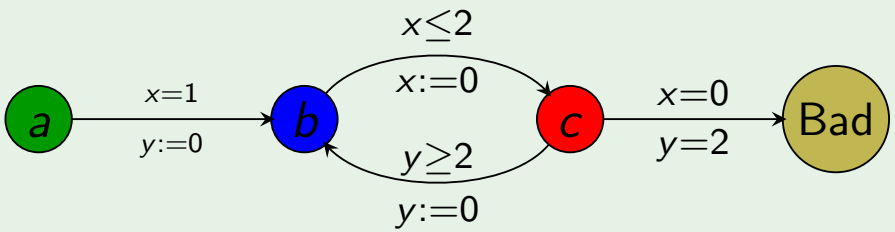


New behaviours on an example

Example ([Puri00])



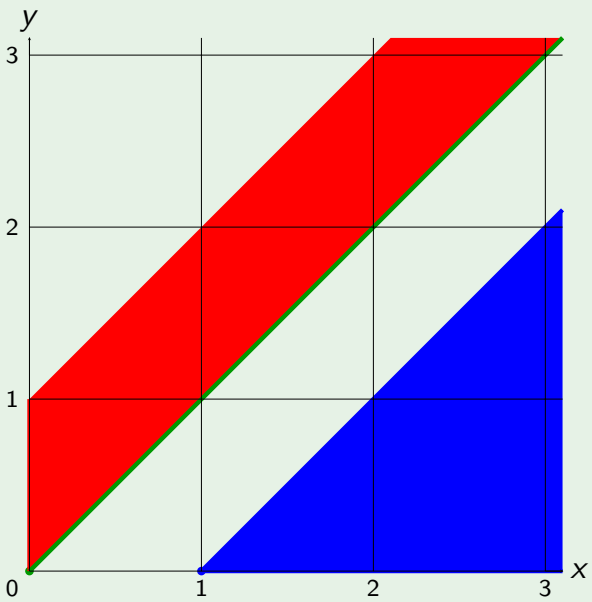




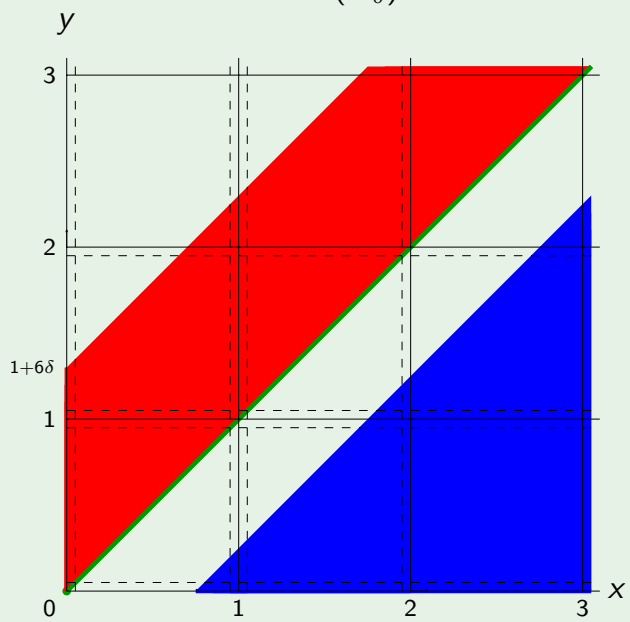
New behaviours on an example

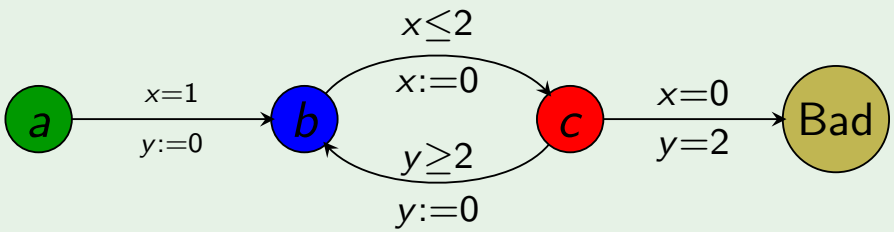
Example ([Puri00])

Reach($\mathcal{A}$)



Reach($\mathcal{A}_\delta$)

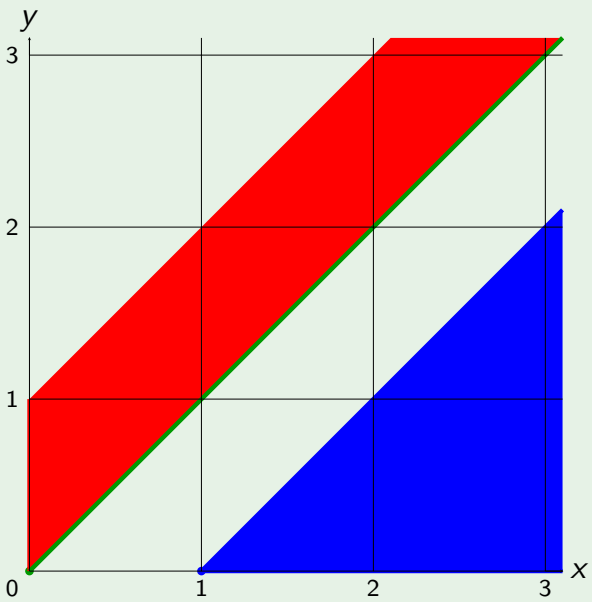




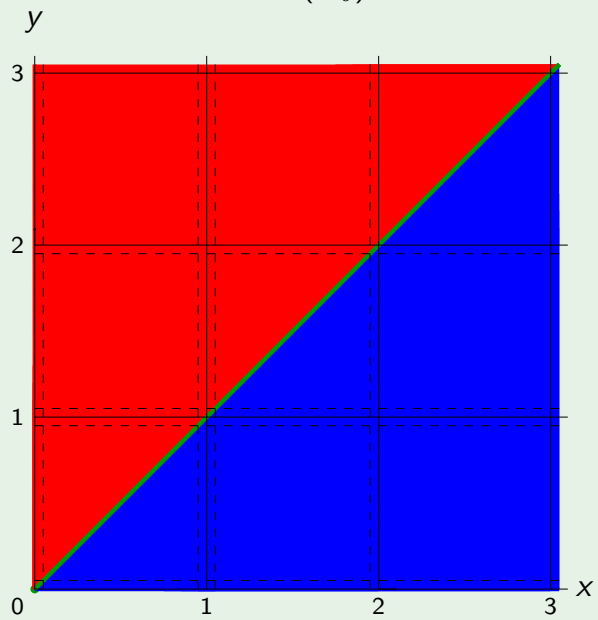
New behaviours on an example

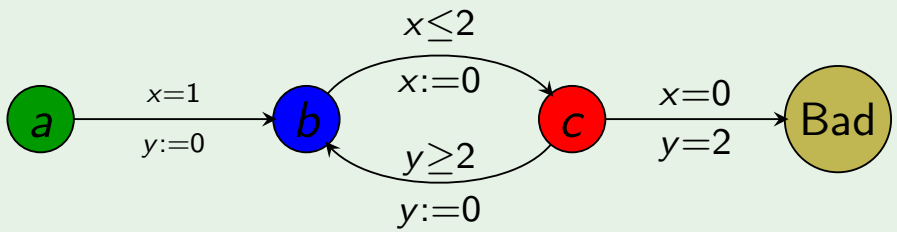
Example ([Puri00])

Reach($\mathcal{A}$)



Reach($\mathcal{A}_\delta$)

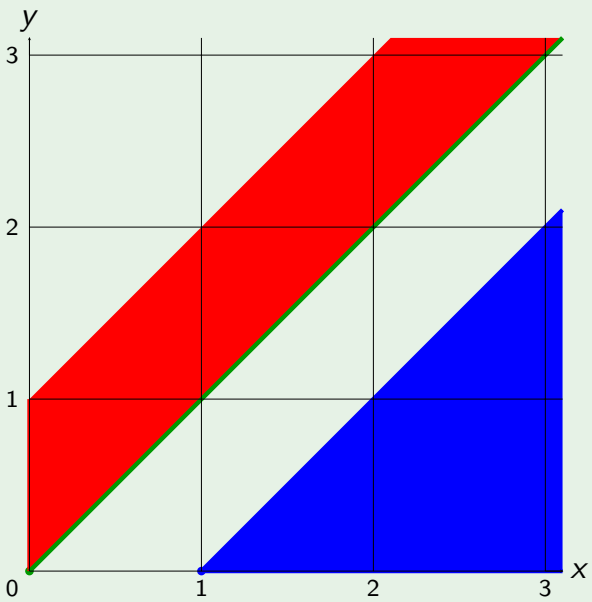




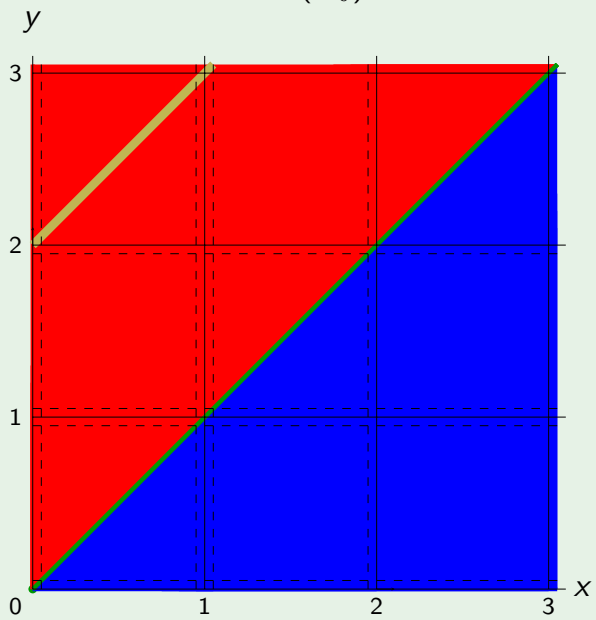
New behaviours on an example

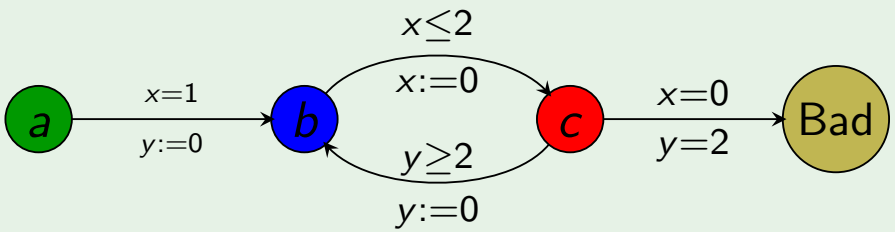
Example ([Puri00])

Reach($\mathcal{A}$)



Reach($\mathcal{A}_\delta$)



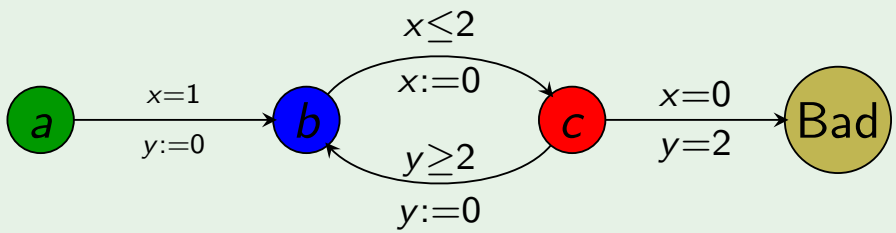


New behaviours on an example

Example ([Puri00])

There may exist configurations (ℓ, ν) such that:

- $(\ell, \nu) \notin \text{Reach}(\mathcal{A})$
- $(\ell, \nu) \in \text{Reach}(\mathcal{A}_\delta)$ for any $\delta > 0$
- the length of the path to (ℓ, ν) increases when δ tends to zero (perturbations are accumulated along loops)



Robust safety [DDMR04]

$\rightsquigarrow$ Reach($\mathcal{A}_\delta$) is the set of reachable states in $\mathcal{A}_\delta$.

Robust Safety Problem:

- Input: a TA $\mathcal{A}$, a set of bad states B.
- Does there exist $\delta > 0$ such that $\text{Reach}(\mathcal{A}_\delta) \cap B = \emptyset$?

Intuitively, can we find a bound δ such that, for every perturbations less than δ, nothing bad happens?

Robust safety [DDMR04]

$\rightsquigarrow$ $\mathsf{Reach}(\mathcal{A}_\delta)$ is the set of reachable states in $\mathcal{A}_\delta$.

Robust Safety Problem:

- Input: a TA $\mathcal{A}$, a set of bad states B.
- Does there exist $\delta > 0$ such that $\mathsf{Reach}(\mathcal{A}_\delta) \cap B = \emptyset$?

$$\delta_1 \leq \delta_2 \Rightarrow \mathsf{Reach}(\mathcal{A}_{\delta_1}) \subseteq \mathsf{Reach}(\mathcal{A}_{\delta_2})$$

$\rightsquigarrow$ $\mathsf{Reach}_{>0}(\mathcal{A}) = \bigcap_{\delta>0} \mathsf{Reach}(\mathcal{A}_\delta)$ is the set of reachable states under the enlarged semantics for any $\delta > 0$.

Robust safety [DDMR04]

$\rightsquigarrow$ $\mathsf{Reach}(\mathcal{A}_\delta)$ is the set of reachable states in $\mathcal{A}_\delta$.

Robust Safety Problem:

- Input: a TA $\mathcal{A}$, a set of bad states B.
- Does there exist $\delta > 0$ such that $\mathsf{Reach}(\mathcal{A}_\delta) \cap B = \emptyset$?

$$\delta_1 \leq \delta_2 \Rightarrow \mathsf{Reach}(\mathcal{A}_{\delta_1}) \subseteq \mathsf{Reach}(\mathcal{A}_{\delta_2})$$

$\rightsquigarrow$ $\mathsf{Reach}_{>0}(\mathcal{A}) = \bigcap_{\delta>0} \mathsf{Reach}(\mathcal{A}_\delta)$ is the set of reachable states under the enlarged semantics for any $\delta > 0$.

Lemma

For any timed automata $\mathcal{A}$ and for any set of zones B,

$$\mathit{Reach}_{>0}(\mathcal{A}) \cap B = \varnothing \quad \mathit{iff} \quad \exists \delta > 0.\ \mathit{Reach}(\mathcal{A}_\delta) \cap B = \varnothing.$$

Region Automaton [AD90]

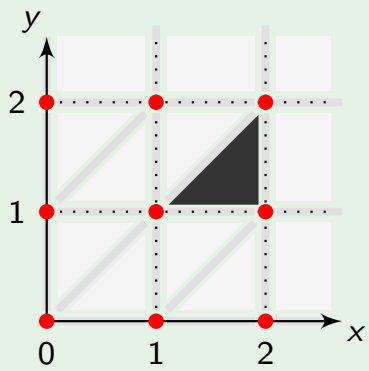
Region Automaton $\mathcal{R}(\mathcal{A})$

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.

Closed guards $\Rightarrow$ topologically closed regions

An algorithm for computing $\mathsf{Reach}_{>0}(\mathcal{A})$ [DDMR04]

`Input: A Timed Automaton` $\mathcal{A}$

`Output: The set` $\mathsf{Reach}\_{>}0(\mathcal{A})$

An algorithm for computing $\mathsf{Reach}_{>0}(\mathcal{A})$ [DDMR04]

Input: A Timed Automaton $\mathcal{A}$

Output: The set $\mathsf{Reach}\_{>}0(\mathcal{A})$

1. build the region graph G of $\mathcal{A}$;

An algorithm for computing $\mathrm{Reach}_{>0}(\mathcal{A})$ [DDMR04]

Input: A Timed Automaton $\mathcal{A}$

Output: The set $\mathrm{Reach}\_{>}0(\mathcal{A})$

1. build the region graph G of $\mathcal{A}$;
2. compute $\mathrm{SCC}(G)$ = the set of strongly connected components of G;

An algorithm for computing $\mathrm{Reach}_{>0}(\mathcal{A})$ [DDMR04]

Input: A Timed Automaton $\mathcal{A}$

Output: The set $\mathrm{Reach}_{>0}(\mathcal{A})$

1. build the region graph G of $\mathcal{A}$;
2. compute $\mathrm{SCC}(G)$ = the set of strongly connected components of G;
3. $J := [(q_0)]$;

6. return(J);

An algorithm for computing $\mathrm{Reach}_{>0}(\mathcal{A})$ [DDMR04]

Input: A Timed Automaton $\mathcal{A}$

Output: The set $\mathrm{Reach}\_{>}0(\mathcal{A})$

1. build the region graph G of $\mathcal{A}$;
2. compute $\mathrm{SCC}(G)$ = the set of strongly connected components of G;
3. $J := [(q_0)]$;
4. $J := \mathrm{Reach}(G, J)$;

6. return(J);

An algorithm for computing $\mathrm{Reach}_{>0}(\mathcal{A})$ [DDMR04]

Input: A Timed Automaton $\mathcal{A}$

Output: The set $\mathrm{Reach}\_{>}0(\mathcal{A})$

1. build the region graph G of $\mathcal{A}$;
2. compute $\mathrm{SCC}(G)$ = the set of strongly connected components of G;
3. $J := [(q_0)]$;
4. $J := \mathrm{Reach}(G, J)$;
5. while $\exists\ S \in \mathrm{SCC}(G).\ S \not\subseteq J$ and $\overline{S} \cap \overline{J} \neq \varnothing$,
   $J := J \cup S$;
   $J := \mathrm{Reach}(G, J)$;
6. return(J);

An algorithm for computing $\mathrm{Reach}_{>0}(\mathcal{A})$ [DDMR04]

Input: A Timed Automaton $\mathcal{A}$

Output: The set $\mathrm{Reach}_{>0}(\mathcal{A})$

1. build the region graph G of $\mathcal{A}$;
2. compute $\mathrm{SCC}(G)$ = the set of strongly connected components of G;
3. $J := [(q_0)]$;
4. $J := \mathrm{Reach}(G, J)$;
5. while $\exists\ S \in \mathrm{SCC}(G).\ S \not\subseteq J$ and $\overline{S} \cap \overline{J} \neq \varnothing$,
   $J := J \cup S$;
   $J := \mathrm{Reach}(G, J)$;
6. return(J);

Theorem

This algorithm is correct.

An algorithm for computing $\mathrm{Reach}_{>0}(\mathcal{A})$ [DDMR04]

Input: A Timed Automaton $\mathcal{A}$

Output: The set $\mathrm{Reach}_{>0}(\mathcal{A})$

1. build the region graph G of $\mathcal{A}$;
2. compute $\mathrm{SCC}(G)$ = the set of strongly connected components of G;
3. $J := [(q_0)]$;
4. $J := \mathrm{Reach}(G, J)$;
5. while $\exists\ S \in \mathrm{SCC}(G).\ S \not\subseteq J$ and $\overline{S} \cap \overline{J} \neq \varnothing$,
   $J := J \cup S$;
   $J := \mathrm{Reach}(G, J)$;
6. return(J);

Theorem

Robustness and implementability w.r.t. safety properties can be checked in PSPACE.

Symbolic algorithm computing $\mathrm{Reach}_{>0}(\mathcal{A})$ [DK06]

Hypothesis: flat timed automata

(ℓ, Z): symbolic state, Z is a zone

(1) $\mathit{Wait} = \{(\ell_0, Z_0)\}$;
(2) $\mathit{Passed} = \emptyset$;
(3) while ($\mathit{Wait} \neq \emptyset$)
(4) pop (ℓ, Z) from Wait ;
(5) if $(\forall (\ell, Z') \in \mathit{Passed},\ Z \not\subseteq Z')$

(6) $\mathit{Wait} = \mathit{Wait} \cup \mathit{Succ}(\ell, Z)$; // classical forward analysis
(7) $\mathit{Passed} = \mathit{Passed} \cup (\ell, Z)$;

Symbolic algorithm computing $\mathrm{Reach}_{>0}(\mathcal{A})$ [DK06]

Hypothesis: flat timed automata

(ℓ, Z): symbolic state, Z is a zone

(1) Computation of $W_\rho = \nu Y.Pre^\rho(Y) \cap \nu Y.Post^\rho(Y)$ for all cycle ρ in $\mathcal{A}$;
(2) $Wait = \{(\ell_0, Z_0)\}$;
(3) $Passed = \emptyset$;
(4) while $(Wait \neq \emptyset)$
(5) pop (ℓ, Z) from $Wait$;
(6) if $(\forall (\ell, Z') \in Passed,\ Z \not\subseteq Z')$

(7) $Wait = Wait \cup Succ(\ell, Z)$; // classical forward analysis
(8) $Passed = Passed \cup (\ell, Z)$;

Symbolic algorithm computing $\mathrm{Reach}_{>0}(\mathcal{A})$ [DK06]

Hypothesis: flat timed automata

(ℓ, Z): symbolic state, Z is a zone

(1) Computation of $W_\rho = \nu Y.Pre^\rho(Y) \cap \nu Y.Post^\rho(Y)$ for all cycle ρ in $\mathcal{A}$;
(2) $Wait = \{(\ell_0, Z_0)\}$;
(3) $Passed = \emptyset$;
(4) while $(Wait \neq \emptyset)$
(5) pop (ℓ, Z) from $Wait$;
(6) if $(\forall (\ell, Z') \in Passed,\ Z \not\subseteq Z')$
(7) if $(\exists \rho : \ell \in \rho)$ // acceleration of the cycle
(8) if $(Z \cap W_\rho \neq \emptyset)$
(9) $Wait = Wait \cup Succ(\ell, W_\rho)$;
(10) $Passed = Passed \cup (\ell, W_\rho)$;
(11) $Wait = Wait \cup Succ(\ell, Z)$; // classical forward analysis
(12) $Passed = Passed \cup (\ell, Z)$;

Symbolic algorithm computing $\mathrm{Reach}_{>0}(\mathcal{A})$ [DK06]

Hypothesis: flat timed automata

(ℓ, Z): symbolic state, Z is a zone

(1) Computation of $W_\rho = \nu Y.Pre^\rho(Y) \cap \nu Y.Post^\rho(Y)$ for all cycle ρ in $\mathcal{A}$;
(2) $Wait = \{(\ell_0, Z_0)\}$;
(3) $Passed = \emptyset$;
(4) while $(Wait \neq \emptyset)$
(5) pop (ℓ, Z) from $Wait$;
(6) if $(\forall (\ell, Z') \in Passed,\ Z \not\subseteq Z')$
(7) if $(\exists \rho : \ell \in \rho)$ // acceleration of the cycle
(8) if $(Z \cap W_\rho \neq \emptyset)$
(9) $Wait = Wait \cup Succ(\ell, W_\rho)$;
(10) $Passed = Passed \cup (\ell, W_\rho)$;
(11) $Wait = Wait \cup Succ(\ell, Z)$; // classical forward analysis
(12) $Passed = Passed \cup (\ell, Z)$;

Theorem

This algorithm is correct and can be implemented using zones.

Maximal perturbation [JR11]

These algorithms allow to decide the existence of a positive perturbation.

➜ What about the maximal admissible one?

Maximal perturbation [JR11]

These algorithms allow to decide the existence of a positive perturbation.

➜ What about the maximal admissible one?

Theorem

For flat timed automata, the (parametric) set Reach($\mathcal{A}_\delta$) can be computed.

Proof relies on the previous algorithm:

- slight modification of the algorithm
- use Parametric DBMs

Maximal perturbation [JR11]

These algorithms allow to decide the existence of a positive perturbation.

➜ What about the maximal admissible one?

Theorem

For flat timed automata, the (parametric) set Reach($\mathcal{A}_\delta$) can be computed.

Proof relies on the previous algorithm:

- slight modification of the algorithm
- use Parametric DBMs

Corollary

Given a flat timed automata $\mathcal{A}$ and some safety property B, we can compute the largest δ such that Reach($\mathcal{A}_\delta$) $\cap B = \emptyset$.

Extended region automaton

The algorithm of [DDMR04] suggests to extend the region automaton with extra transitions:

For any location ℓ and any two regions r and r', if

- $r \cap r' \neq \varnothing$ and
- (ℓ, r') belongs to an SCC of $\mathcal{R}(\mathcal{A})$,

then we add a transition $(\ell, r) \xrightarrow{\gamma} (\ell, r')$.

We write $\mathcal{R}^*(\mathcal{A})$ for the resulting automaton.

Extended region automaton

The algorithm of [DDMR04] suggests to extend the region automaton with extra transitions:

For any location ℓ and any two regions r and r', if

- $r \cap r' \neq \varnothing$ and
- (ℓ, r') belongs to an SCC of $\mathcal{R}(\mathcal{A})$,

then we add a transition $(\ell, r) \xrightarrow{\gamma} (\ell, r')$.

We write $\mathcal{R}^*(\mathcal{A})$ for the resulting automaton.

Theorem

The set $Reach_{>0}(\mathcal{A})$ is the set of reachable regions in $\mathcal{R}^*(\mathcal{A})$.

LTL Robust Model-Checking [BMR06]

Definition

$$\text{LTL} \ni \psi, \varphi ::= p \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \neg\varphi \mid \mathbf{X}\,\varphi \mid \varphi\ \mathbf{U}\ \psi$$

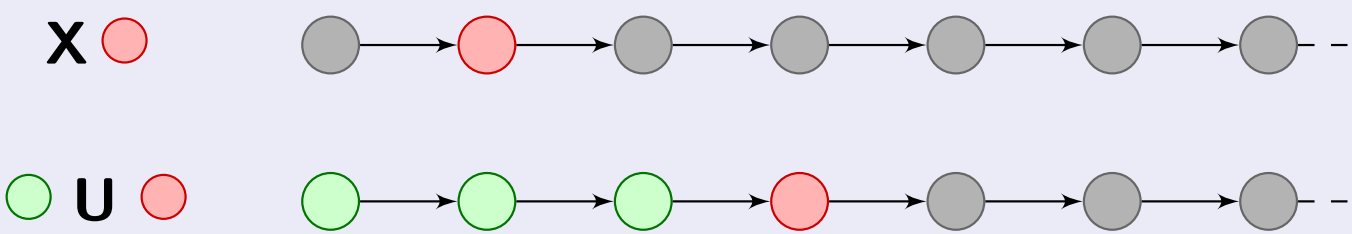
LTL Robust Model-Checking [BMR06]

Definition

$$\text{LTL} \ni \psi, \varphi ::= p \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \neg\varphi \mid \mathbf{X}\,\varphi \mid \varphi\ \mathbf{U}\ \psi$$

LTL formulas are evaluated along paths:

Definition



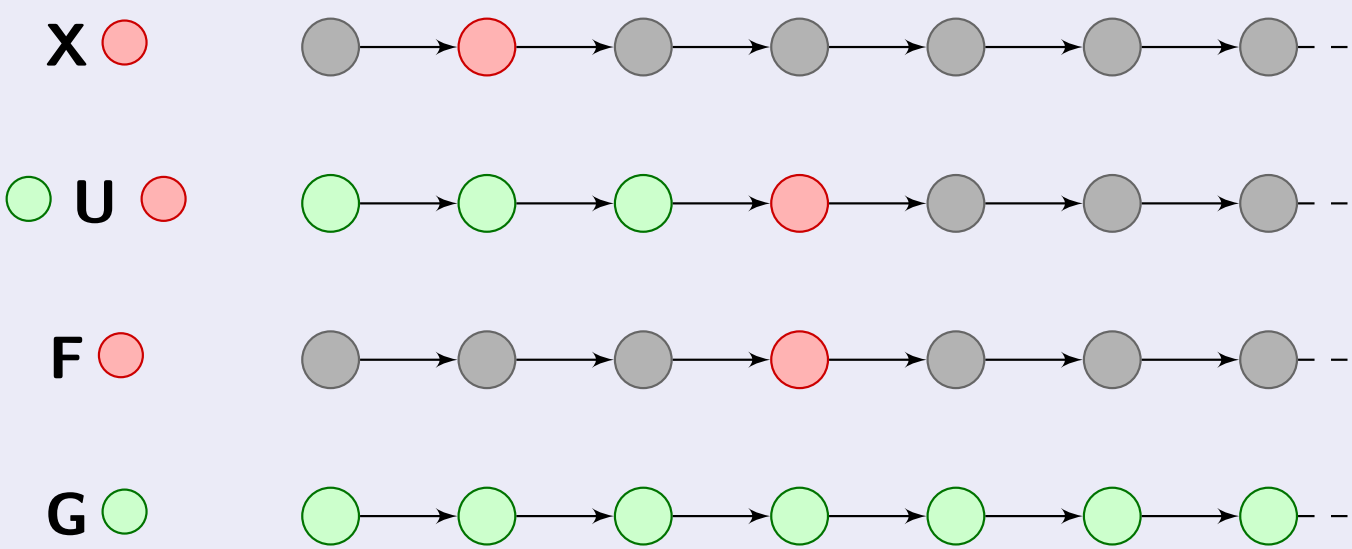
LTL Robust Model-Checking [BMR06]

Definition

$$\text{LTL} \ni \psi, \varphi ::= p \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \neg\varphi \mid \mathbf{X}\,\varphi \mid \varphi\ \mathbf{U}\ \psi$$

LTL formulas are evaluated along paths:

Definition



LTL Robust Model-Checking [BMR06]

Theorem

Any LTL formula ϕ can be turned into an equivalent Büchi automaton $\mathcal{B}_\phi$.

LTL Robust Model-Checking [BMR06]

Theorem

Any LTL formula ϕ can be turned into an equivalent Büchi automaton $\mathcal{B}_\phi$.

Theorem

Let $\mathcal{B}_{\neg\phi}$ be a Büchi automaton corresponding to LTL formula $\neg\phi$, and with accepting set $Q_{\neg\phi}$. Then

$$\mathcal{A} \models \phi \quad \Leftrightarrow \quad \mathcal{A} \times \mathcal{B}_{\neg\phi} \models \text{co-Büchi}(L \times Q_{\neg\phi}).$$

LTL Robust Model-Checking [BMR06]

Theorem

Any LTL formula ϕ can be turned into an equivalent Büchi automaton $\mathcal{B}_\phi$.

Theorem

Let $\mathcal{B}_{\neg\phi}$ be a Büchi automaton corresponding to LTL formula $\neg\phi$, and with accepting set $Q_{\neg\phi}$. Then

$$\mathcal{A} \models \phi \quad \Leftrightarrow \quad \mathcal{A} \times \mathcal{B}_{\neg\phi} \models \text{co-Büchi}(L \times Q_{\neg\phi}).$$

Theorem

Let $\mathcal{A}$ be a timed automaton and Q a set of locations of $\mathcal{A}$. Then

$$\mathcal{A} \models \text{co-Büchi}(Q) \Leftrightarrow \mathcal{R}^*(\mathcal{A}) \models \text{co-Büchi}(Q).$$

LTL Robust Model-Checking [BMR06]

Theorem

Any LTL formula ϕ can be turned into an equivalent Büchi automaton $\mathcal{B}_\phi$.

Theorem

Let $\mathcal{B}_{\neg\phi}$ be a Büchi automaton corresponding to LTL formula $\neg\phi$, and with accepting set $Q_{\neg\phi}$. Then

$$\mathcal{A} \models \phi \quad \Leftrightarrow \quad \mathcal{A} \times \mathcal{B}_{\neg\phi} \models \text{co-Büchi}(L \times Q_{\neg\phi}).$$

Theorem

Let $\mathcal{A}$ be a timed automaton and Q a set of locations of $\mathcal{A}$. Then

$$\mathcal{A} \models \text{co-Büchi}(Q) \;\Leftrightarrow\; \mathcal{R}^*(\mathcal{A}) \models \text{co-Büchi}(Q).$$

Corollary

LTL robust model-checking is PSPACE-complete.

Metric Temporal Logic [Koy87,AH92]

Definition

$$\text{MTL} \ni \psi, \varphi ::= p \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \neg\varphi \mid \varphi \, \mathbf{U}_I \, \psi$$

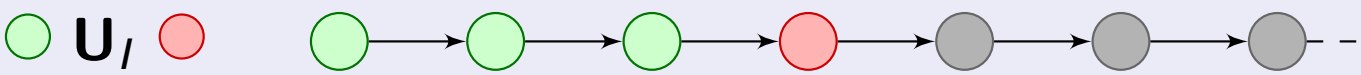
Metric Temporal Logic [Koy87,AH92]

Definition

$$\text{MTL} \ni \psi, \varphi ::= p \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \neg\varphi \mid \varphi \, \mathbf{U}_I \, \psi$$

MTL formulas are evaluated along timed words:

Definition



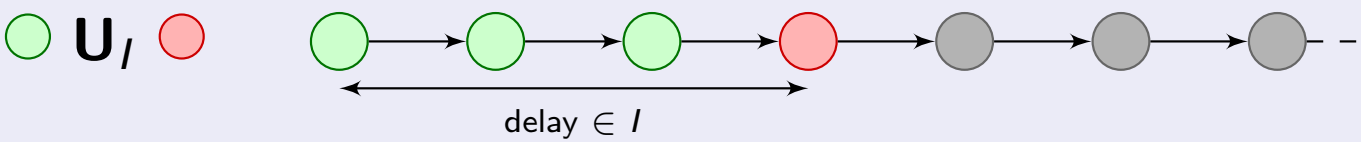
Metric Temporal Logic [Koy87,AH92]

Definition

$$\text{MTL} \ni \psi, \varphi ::= p \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \neg\varphi \mid \varphi \, \mathbf{U}_I \, \psi$$

MTL formulas are evaluated along timed words:

Definition



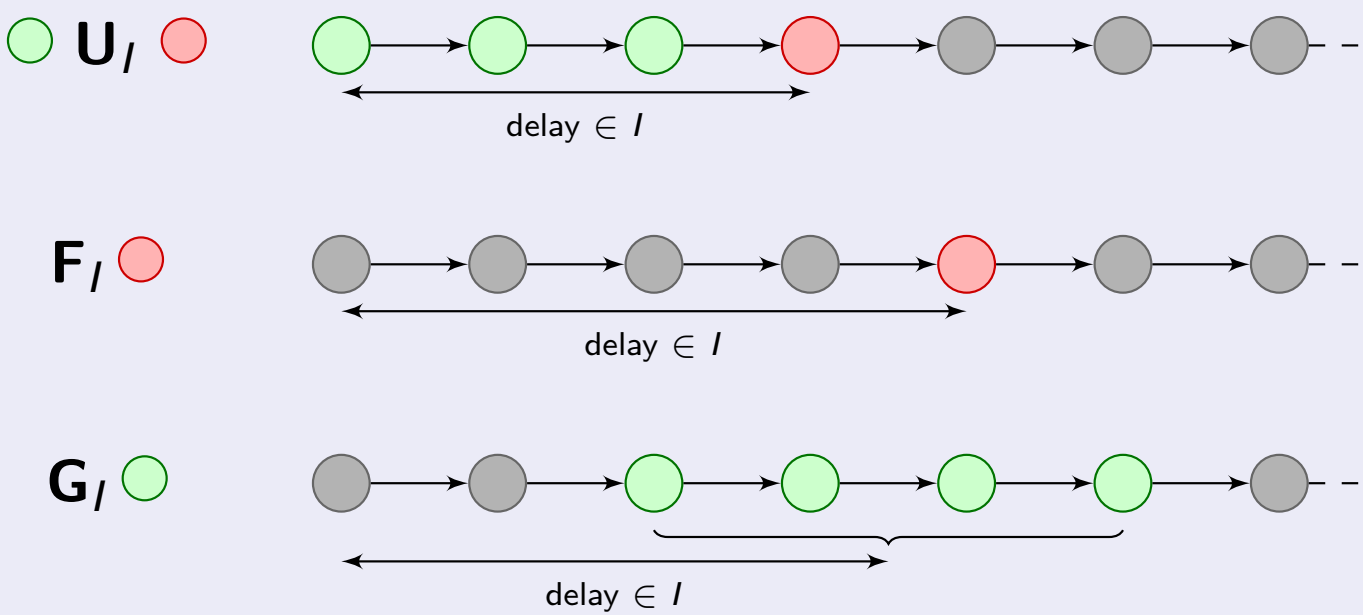
Metric Temporal Logic [Koy87,AH92]

Definition

$$\text{MTL} \ni \psi, \varphi ::= p \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \neg\varphi \mid \varphi \, \mathbf{U}_I \, \psi$$

MTL formulas are evaluated along timed words:

Definition



MTL and Alternating timed automata

LTL formulas can (also) be turned into linear alternating Büchi automata:

Example

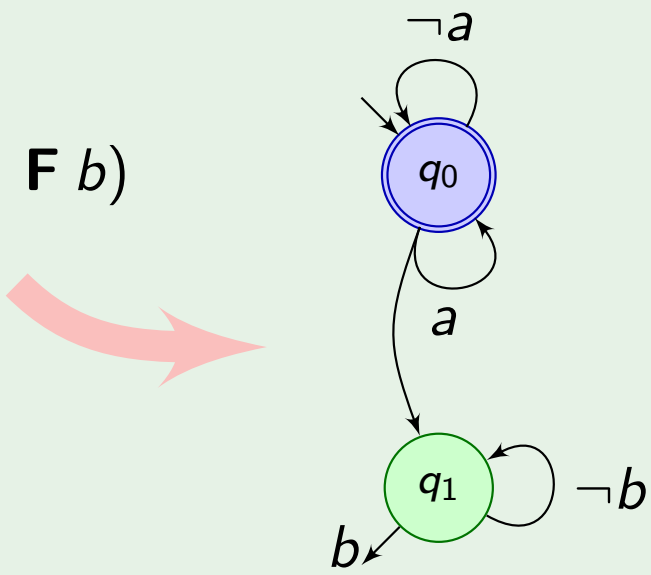
$$\mathbf{G}(a \Rightarrow \mathbf{F}\, b)$$

MTL and Alternating timed automata

LTL formulas can (also) be turned into linear alternating Büchi automata:

Example

$\mathbf{G}(a \Rightarrow \mathbf{F}\, b)$

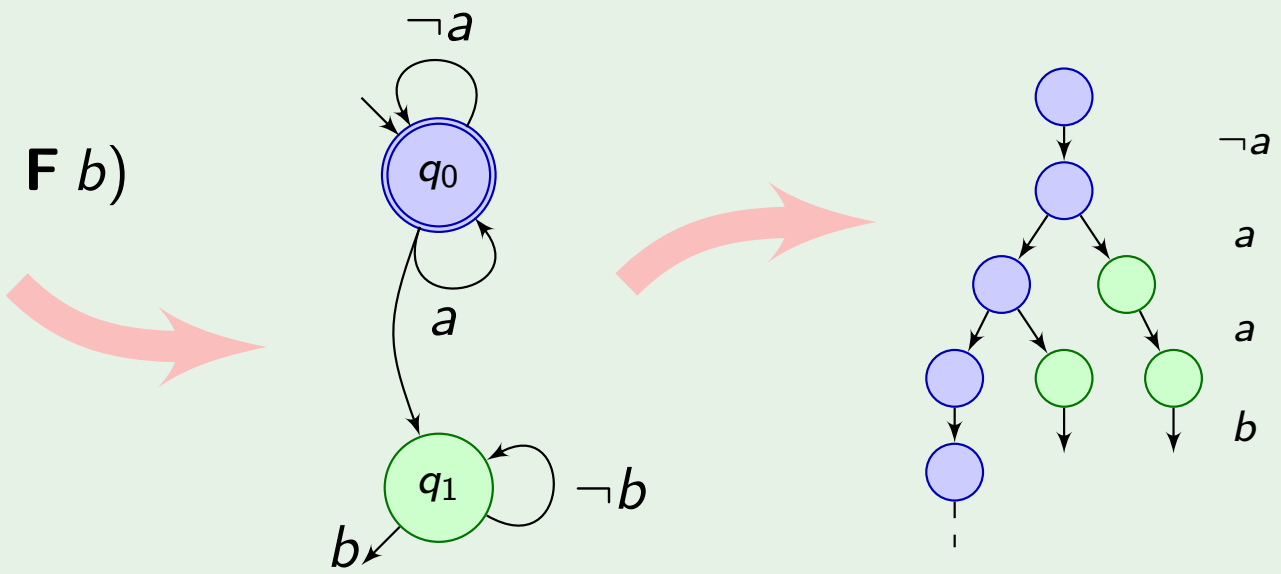


MTL and Alternating timed automata

LTL formulas can (also) be turned into linear alternating Büchi automata:

Example

$\mathbf{G}(a \Rightarrow \mathbf{F}\, b)$

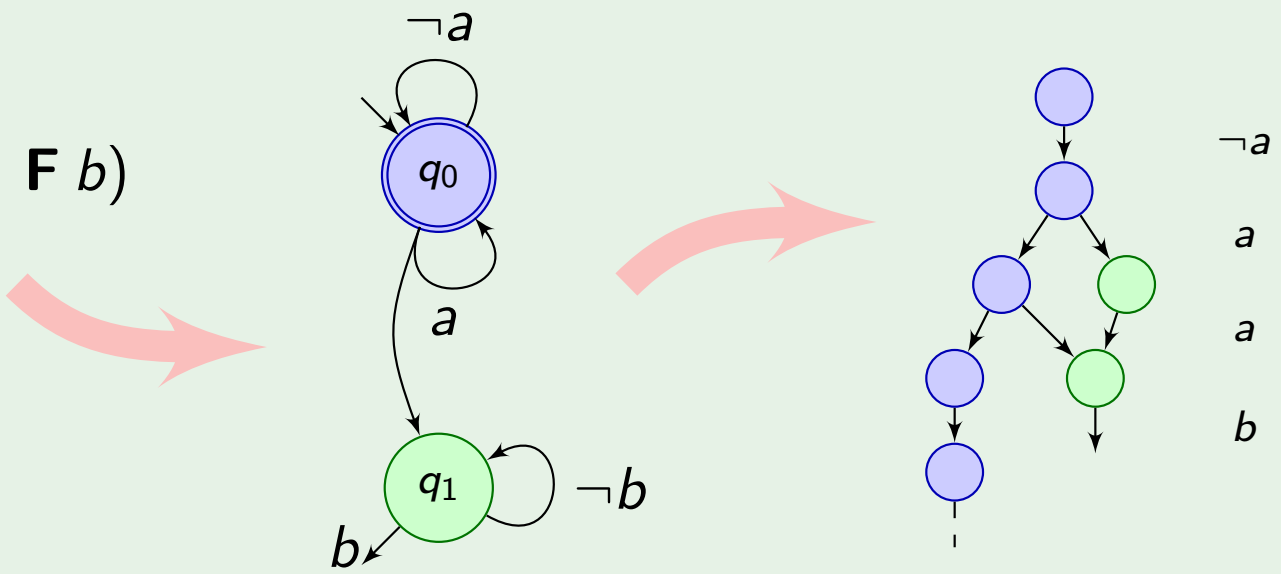


MTL and Alternating timed automata

LTL formulas can (also) be turned into linear alternating Büchi automata:

Example

$\mathbf{G}(a \Rightarrow \mathbf{F}\, b)$



MTL and Alternating timed automata

Similarly, MTL formulas can be turned into 1-clock alternating Büchi automata [OW05]:

Example

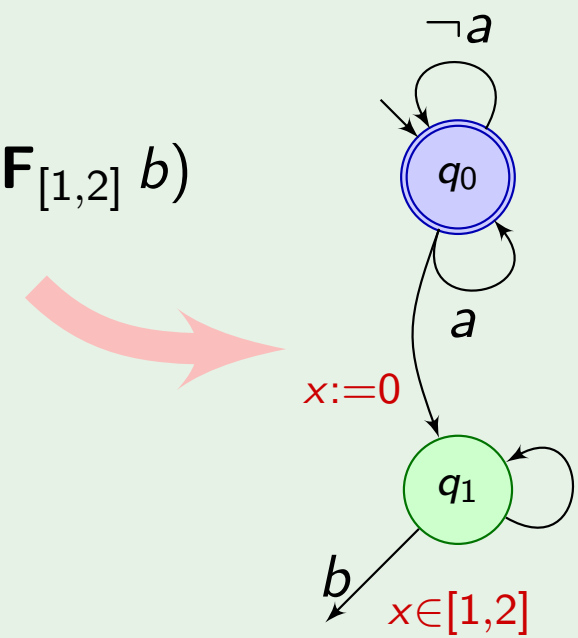
$$\mathbf{G}(a \Rightarrow \mathbf{F}_{[1,2]}\, b)$$

MTL and Alternating timed automata

Similarly, MTL formulas can be turned into 1-clock alternating Büchi automata [OW05]:

Example

$\mathbf{G}(a \Rightarrow \mathbf{F}_{[1,2]}\, b)$

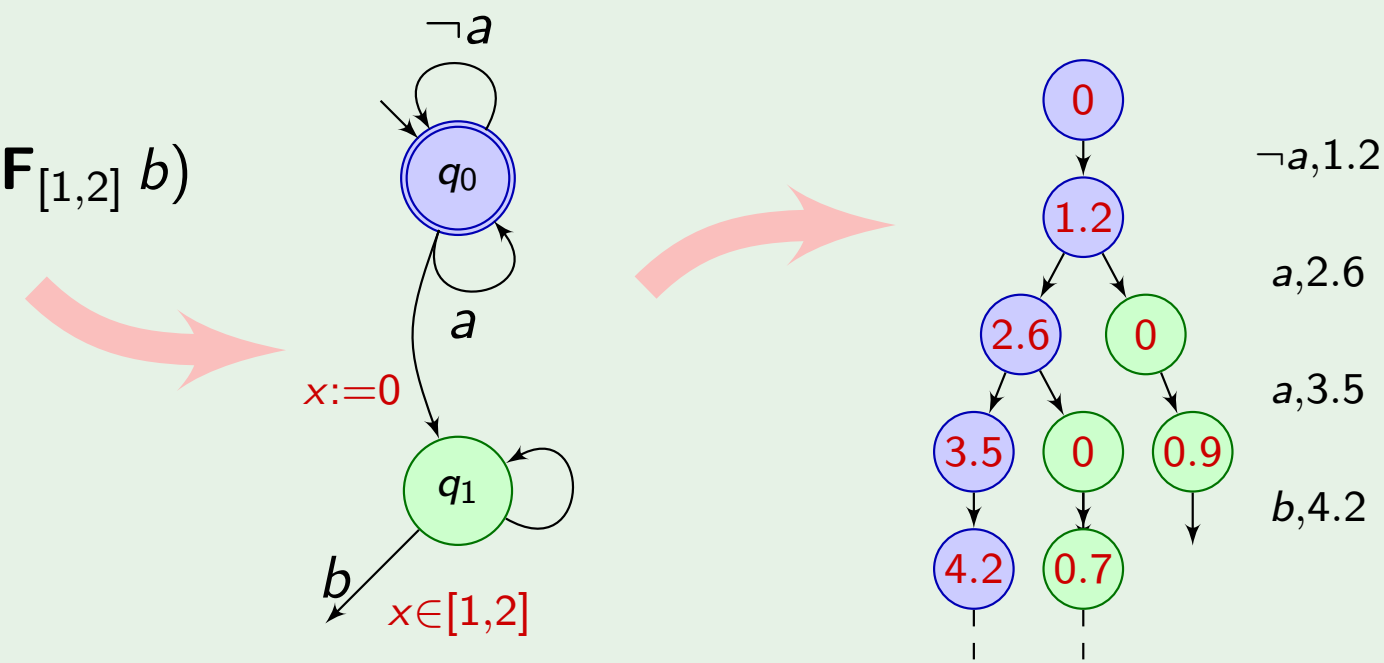


MTL and Alternating timed automata

Similarly, MTL formulas can be turned into 1-clock alternating Büchi automata [OW05]:

Example

$\mathbf{G}(a \Rightarrow \mathbf{F}_{[1,2]}\, b)$



The logic CoFlatMTL

Definition

CoFlatMTL $\ni \psi, \varphi ::= p \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \, \mathbf{U}_{(a,b)} \, \psi \mid \varphi \, \mathbf{U}_{(a,+\infty)} \, \alpha$

where α is an LTL formula.

The logic CoFlatMTL

Definition

CoFlatMTL $\ni \psi, \varphi ::= p \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \, \mathbf{U}_{(a,b)} \, \psi \mid \varphi \, \mathbf{U}_{(a,+\infty)} \, \alpha$

where α is an LTL formula.

Using channel automata with special instructions, one can prove:

Theorem ([BMOW07])

CoFlatMTL model-checking is EXPSPACE-complete.

The logic CoFlatMTL

Definition

CoFlatMTL $\ni \psi, \varphi ::= p \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \, \mathbf{U}_{(a,b)} \, \psi \mid \varphi \, \mathbf{U}_{(a,+\infty)} \, \alpha$

where α is an LTL formula.

Using channel automata with special instructions, one can prove:

Theorem ([BMOW07])

CoFlatMTL model-checking is EXPSPACE-complete.

This approach can actually be adapted to robustness:

Theorem ([BMR08])

CoFlatMTL robust model-checking is EXPSPACE-complete.

Robust Verification: Summary

Only unbounded behaviours matter

Robust Model-Checking

- Safety, LTL, ω-regular : PSPACE-c — No hypothesis
- CoFlatMTL : EXPSPACE-c — progress cycles

➜ same theoretical complexities as w/o robustness

Flat timed automata, against safety properties:

- symbolic approach based on zones
- the largest admissible perturbation can be computed

Overview

Playing in Timed Automata with Büchi condition

Two-player turn-based game played in $\mathcal{A}$. In a configuration (ℓ, ν)

1. Controller suggests a delay d and an action a:

$$(\ell, \nu) \xrightarrow{d} (\ell, \nu + d) \xrightarrow{a}$$

2. Perturbator resolves non-determinism of action a

Playing in Timed Automata with Büchi condition

Two-player turn-based game played in $\mathcal{A}$. In a configuration (ℓ, ν)

1. Controller suggests a delay d and an action a:

$$(\ell, \nu) \xrightarrow{d} (\ell, \nu + d) \xrightarrow{a}$$

2. Perturbator resolves non-determinism of action a

➜ Notions of strategies for Controller and Perturbator, denoted σ_C and σ_P
Depend on the history of the play, may be infinite.

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

σ_C is winning iff for any σ_P, the outcome of σ_C and σ_P is winning.

Playing in Timed Automata with Büchi condition

Two-player turn-based game played in $\mathcal{A}$. In a configuration (ℓ, ν)

1. Controller suggests a delay d and an action a:

$$(\ell, \nu) \xrightarrow{d} (\ell, \nu + d) \xrightarrow{a}$$

2. Perturbator resolves non-determinism of action a

➜ Notions of strategies for Controller and Perturbator, denoted σ_C and σ_P
Depend on the history of the play, may be infinite.

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

σ_C is winning iff for any σ_P, the outcome of σ_C and σ_P is winning.

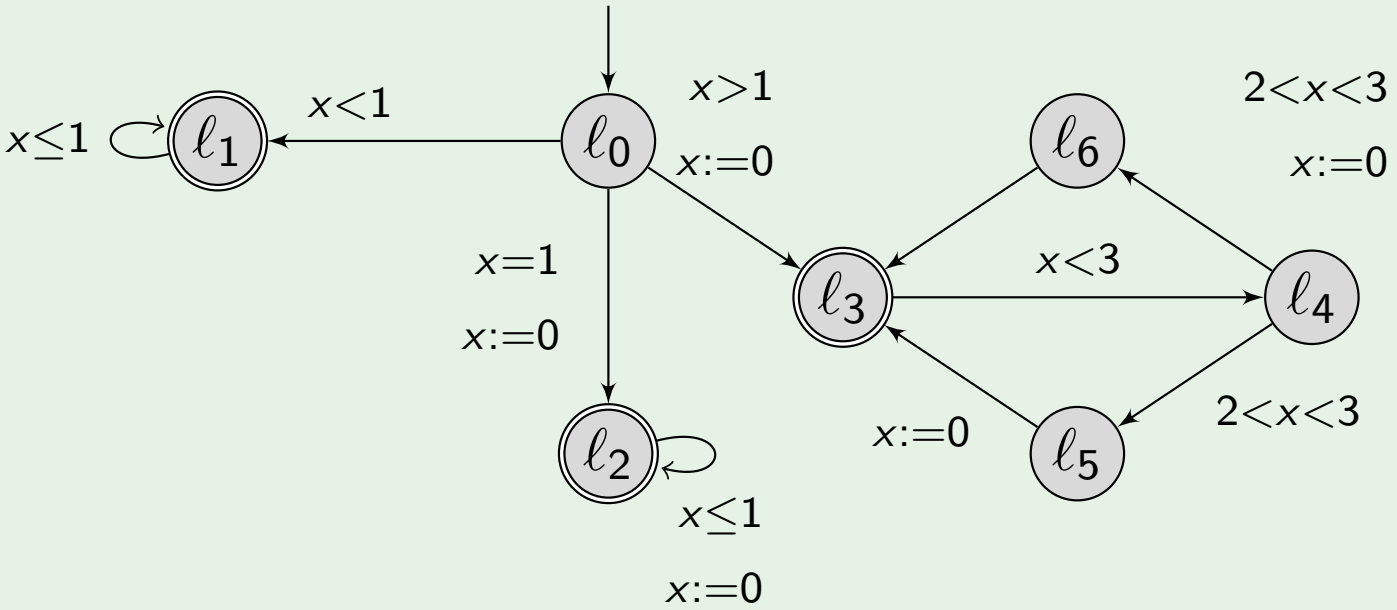
Control Problem

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

A (simple) example

All edges are labelled by action a.

Example

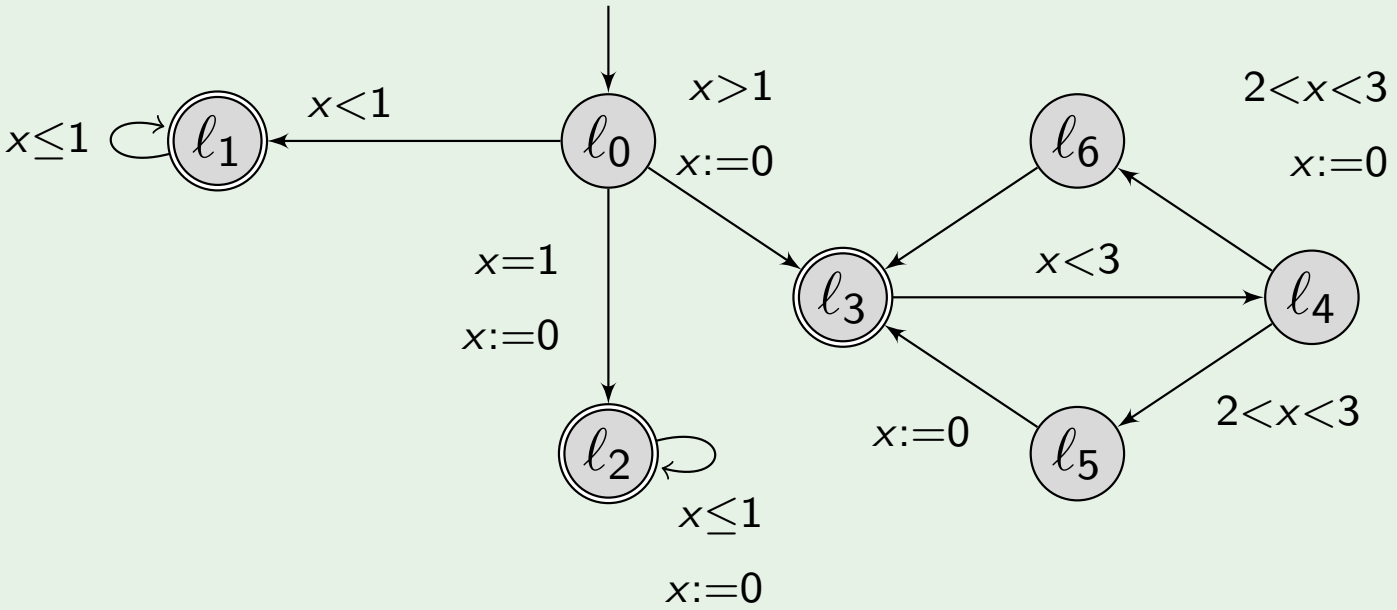


➜ Does Controller has a winning strategy?

A (simple) example

All edges are labelled by action a.

Example



➜ Does Controller has a winning strategy?

Yes! ℓ_1, ℓ_2 and ℓ_3 are valid choices.

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 $\rightsquigarrow \sigma_C$
- Perturbator resolves non-determinism $\rightsquigarrow \sigma_P$

Given σ_C, σ_P, there is a single outcome $\rho[\sigma_C, \sigma_P]$.

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 $\rightsquigarrow \sigma_C$
- Perturbator resolves non-determinism $\rightsquigarrow \sigma_P$

Given σ_C, σ_P, there is a single outcome $\rho[\sigma_C, \sigma_P]$.

Winning Condition

An infinite play is winning for Controller iff it satisfies the Büchi condition.

A Controller's strategy σ_C is winning iff for any σ_P, $\rho[\sigma_C, \sigma_P]$ is winning.

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 $\rightsquigarrow \sigma_C$
- Perturbator resolves non-determinism $\rightsquigarrow \sigma_P$

Given σ_C, σ_P, there is a single outcome $\rho[\sigma_C, \sigma_P]$.

Winning Condition

An infinite play is winning for Controller iff it satisfies the Büchi condition.

A Controller's strategy σ_C is winning iff for any σ_P, $\rho[\sigma_C, \sigma_P]$ is winning.

Existing results on Büchi games imply:

- determinacy: one of the two player has a winning strategy
- decidability in EXPTIME
- finite memory strategies are sufficient

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 $\leadsto \sigma_C$
- Perturbator resolves non-determinism $\leadsto \sigma_P$

Given σ_C, σ_P, there is a single outcome $\rho[\sigma_C, \sigma_P]$.

Winning Condition

An infinite play is winning for Controller iff it satisfies the Büchi condition.

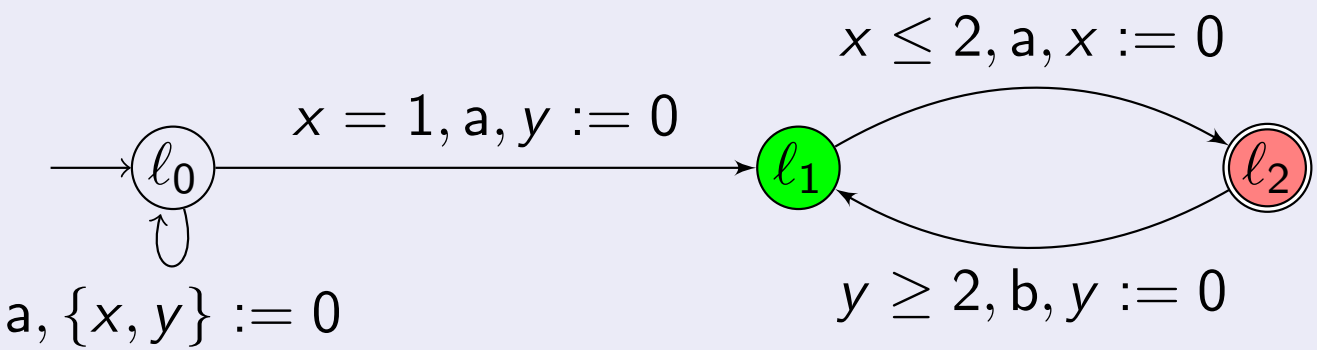
A Controller's strategy σ_C is winning iff for any σ_P, $\rho[\sigma_C, \sigma_P]$ is winning.

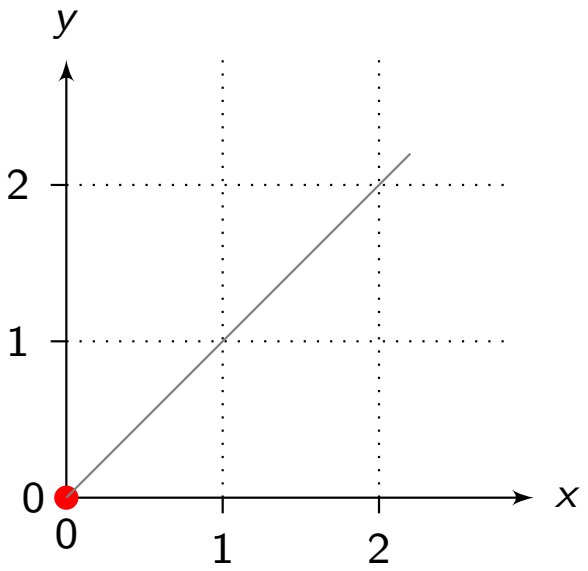
Theorem

The Control Problem for Büchi timed automata is EXPTIME-complete.

Zone based algorithms do exist.

An example with convergence



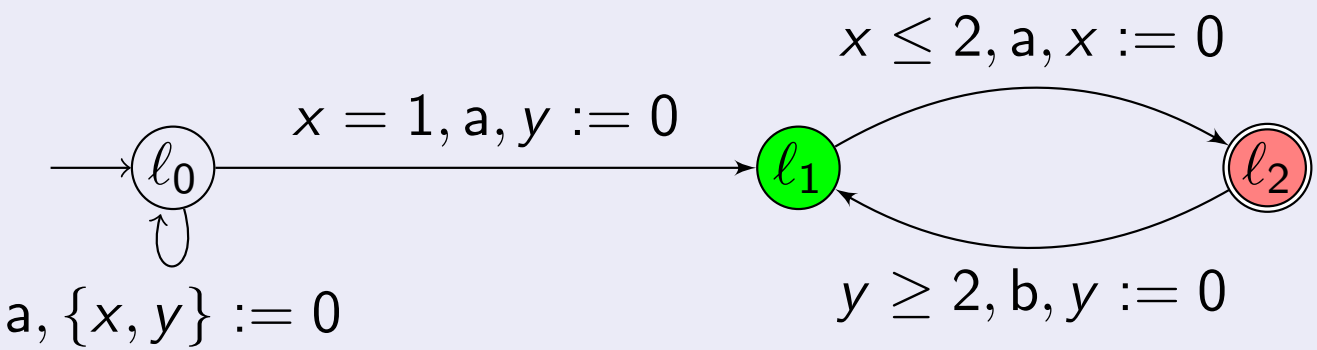


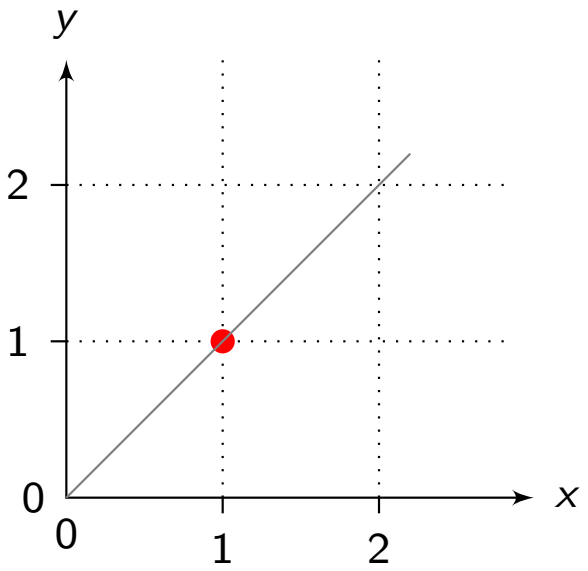
State $(\ell_0, 0, 0)$: fire action a after delay 1

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 $2 - y$

An example with convergence



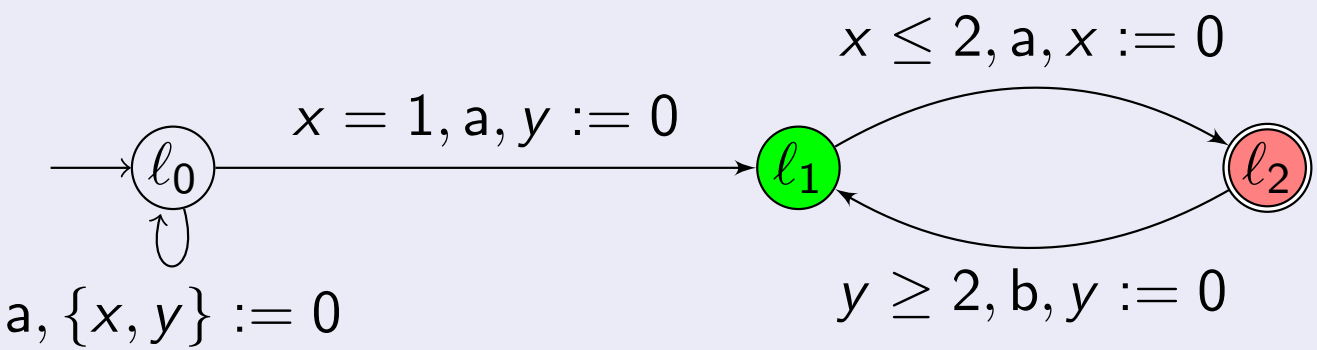


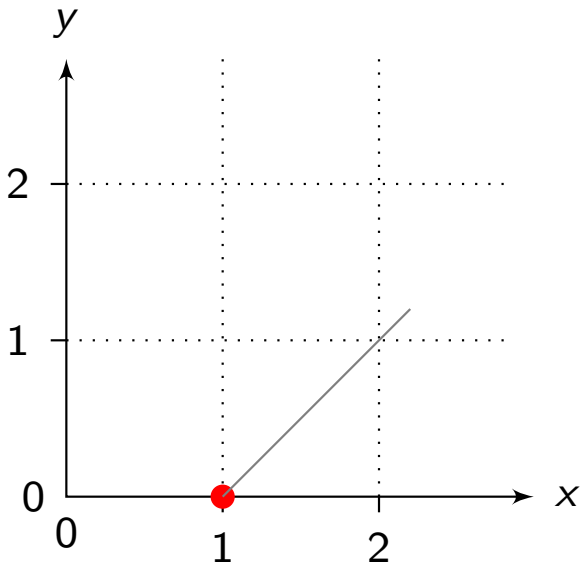
State $(\ell_0, 0, 0)$: fire action a after delay 1

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 $2 - y$

An example with convergence



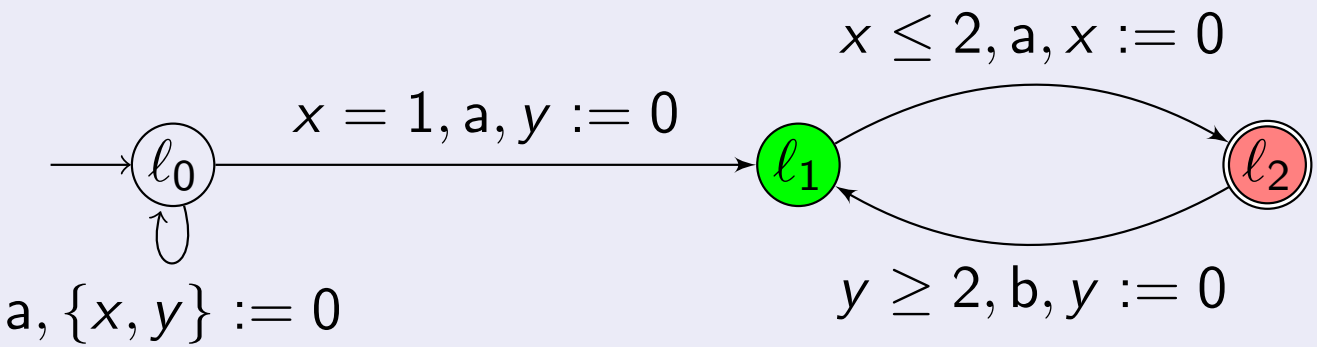


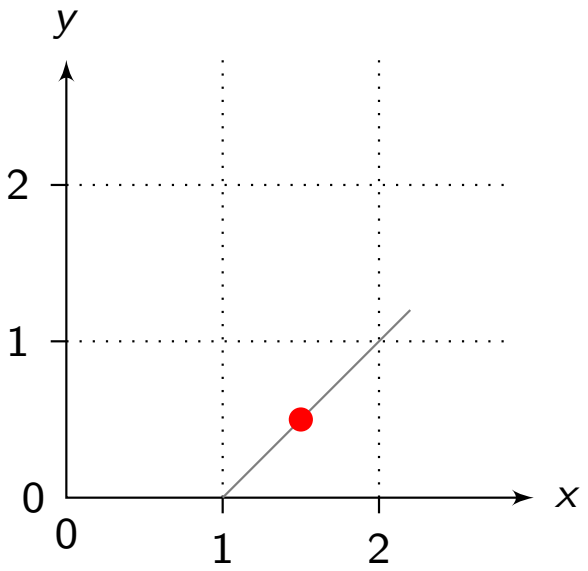
State $(\ell_0, 0, 0)$: fire action a after delay 1

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 $2 - y$

An example with convergence



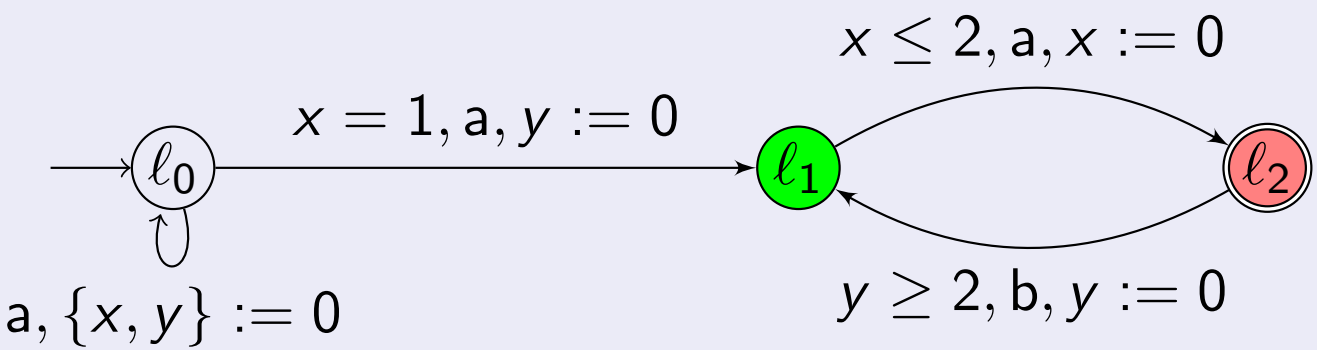


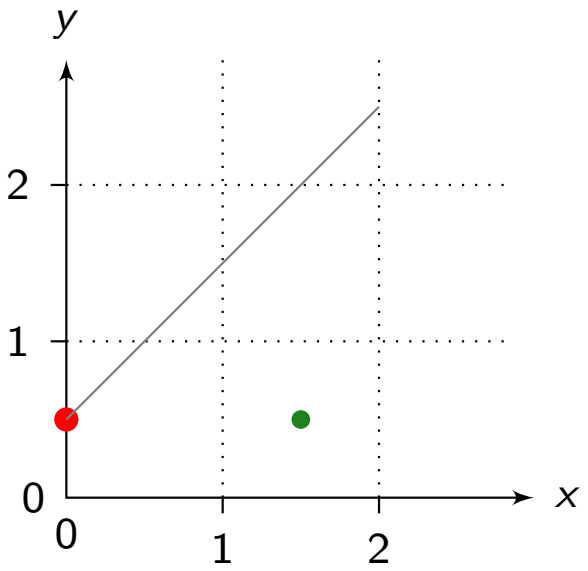
State $(\ell_0, 0, 0)$: fire action a after delay 1

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 $2 - y$

An example with convergence



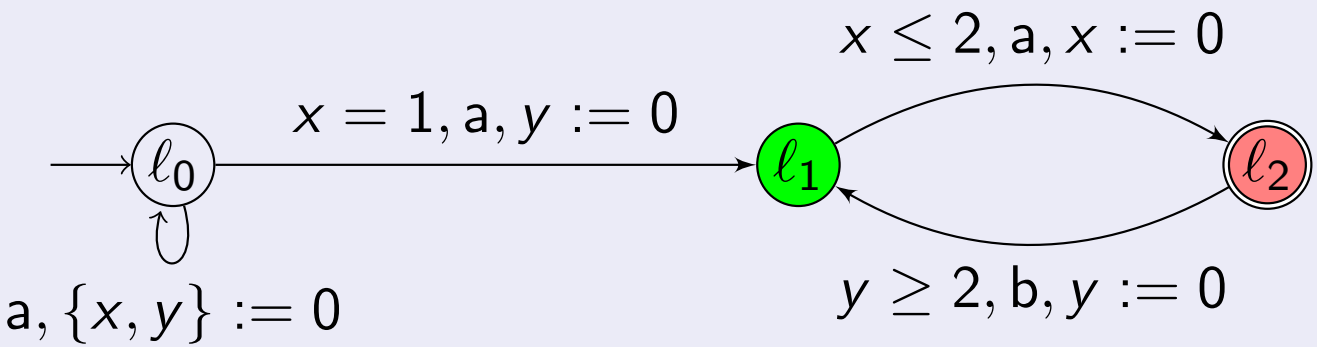


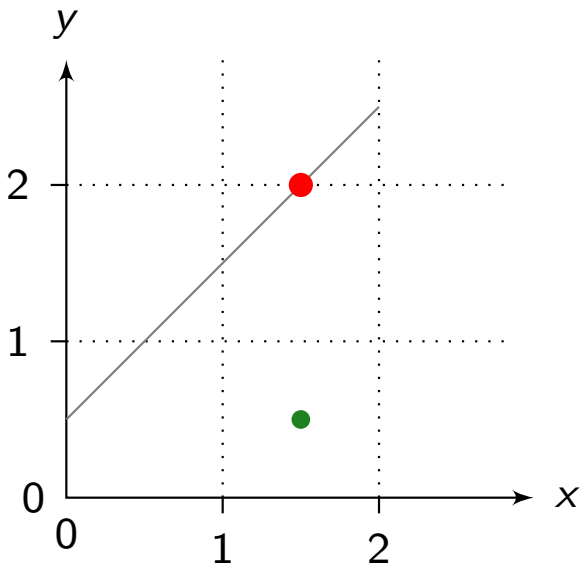
State $(\ell_0, 0, 0)$: fire action a after delay 1

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 $2 - y$

An example with convergence



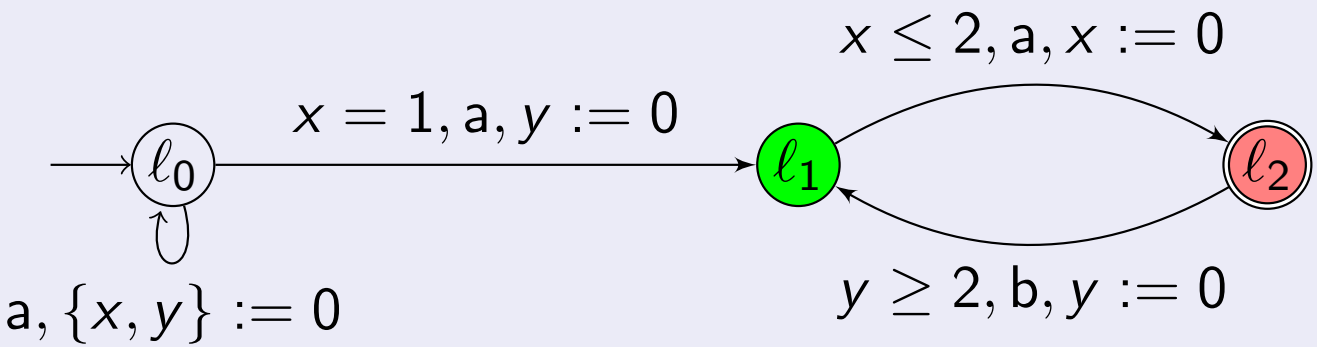


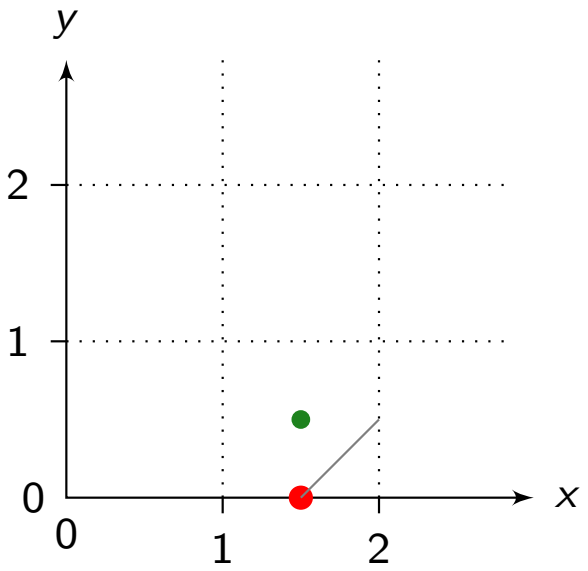
State $(\ell_0, 0, 0)$: fire action a after delay 1

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 $2 - y$

An example with convergence



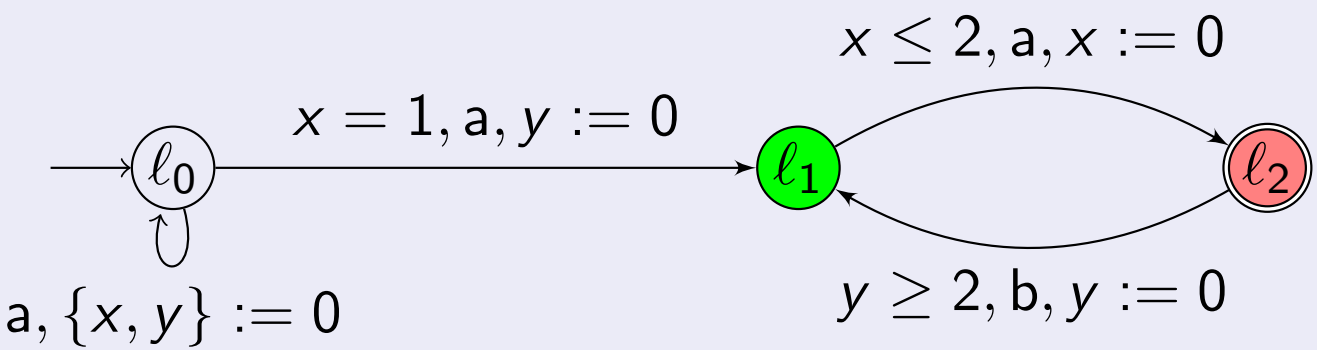


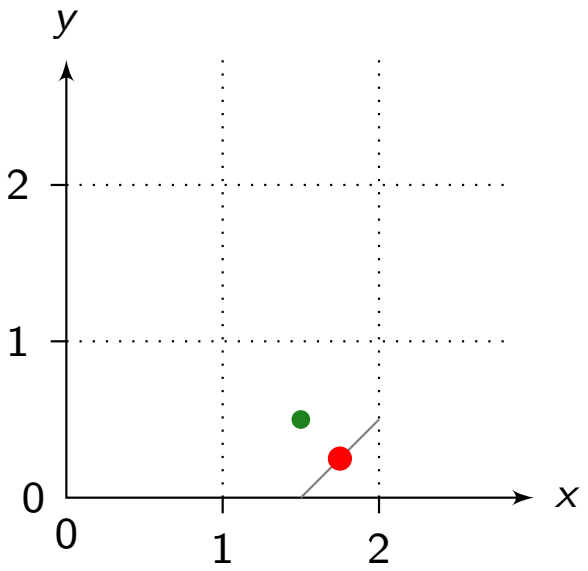
State $(\ell_0, 0, 0)$: fire action a after delay 1

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 $2 - y$

An example with convergence



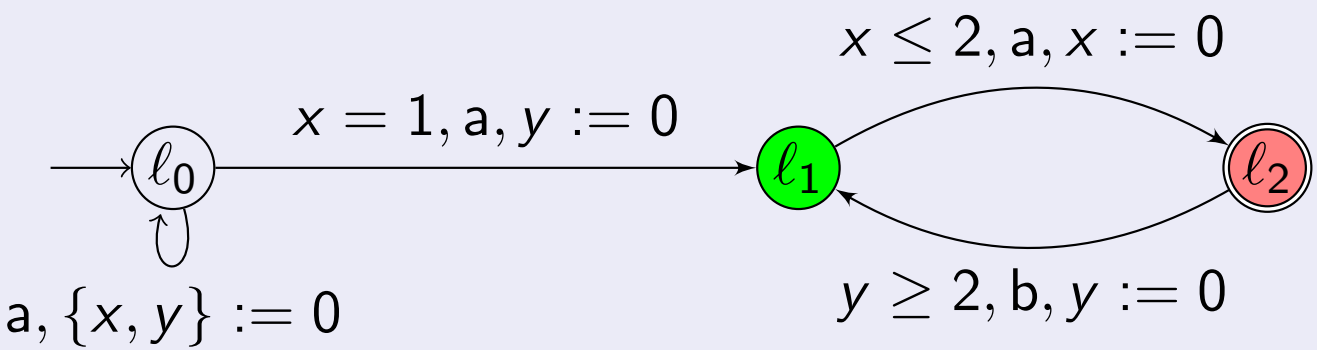


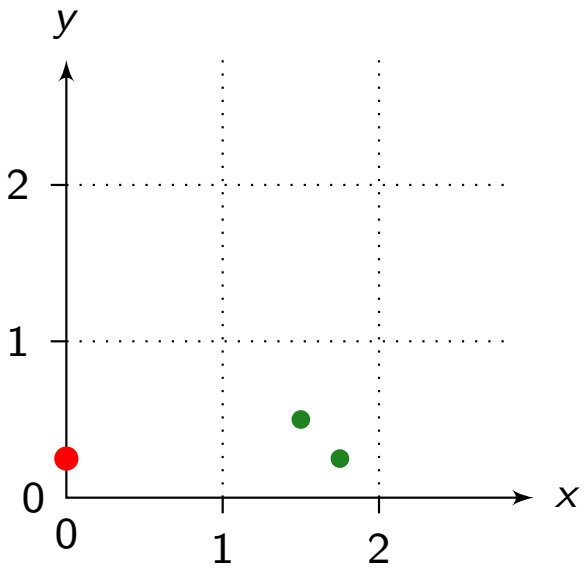
State $(\ell_0, 0, 0)$: fire action a after delay 1

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 $2 - y$

An example with convergence



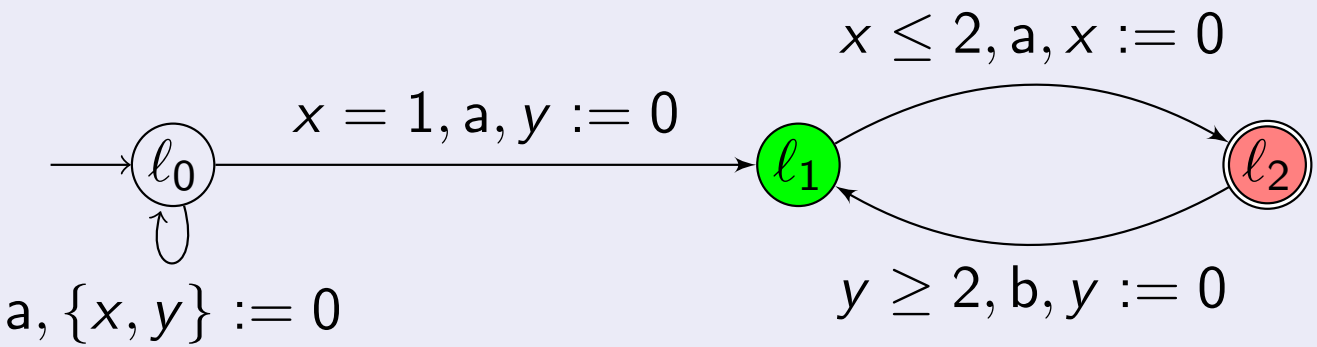


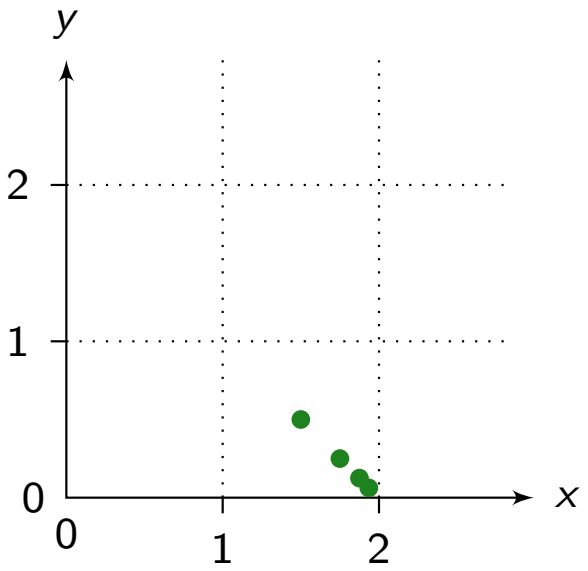
State $(\ell_0, 0, 0)$: fire action a after delay 1

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 $2 - y$

An example with convergence





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 $2 - y$

Robustness Issues

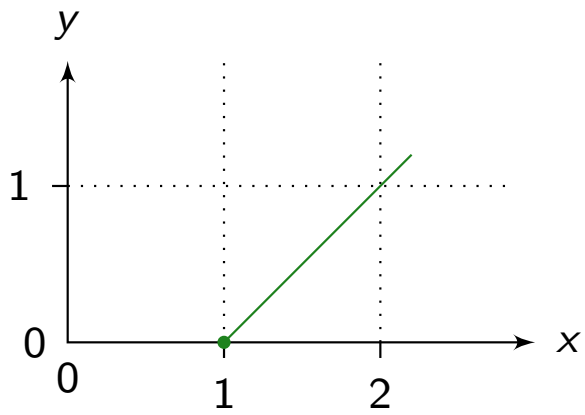
1. Exact timings cannot be ensured in real-time systems.
So the strategy $\left(x \mapsto \text{delay } \frac{2-x}{2}\right)$ cannot be implemented exactly.

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.

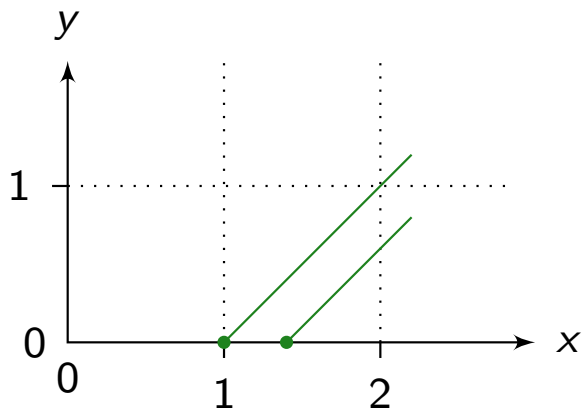
In our example, one can show that x is increasing during consecutive visits to ℓ_1, and the guard is $x \leq 2$.



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.

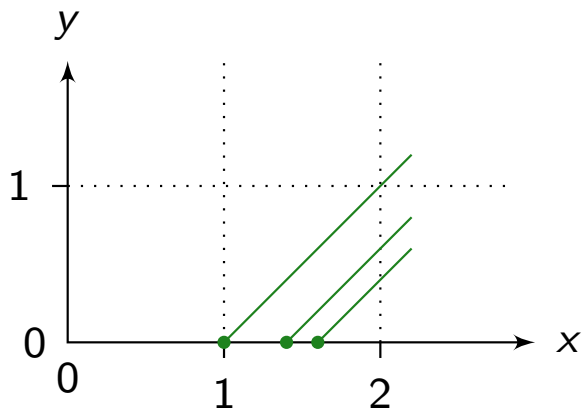
In our example, one can show that x is increasing during consecutive visits to ℓ_1, and the guard is $x \leq 2$.



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.

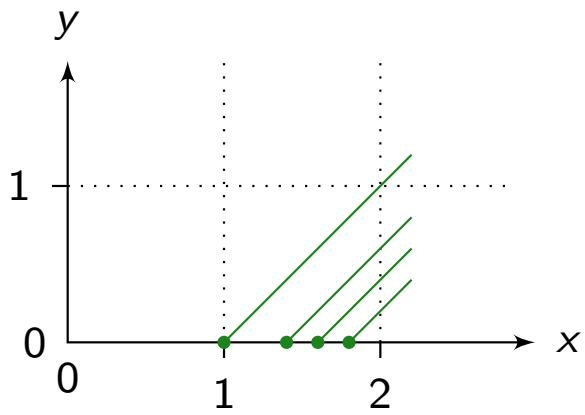
In our example, one can show that x is increasing during consecutive visits to ℓ_1, and the guard is $x \leq 2$.



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 Control Problem 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),

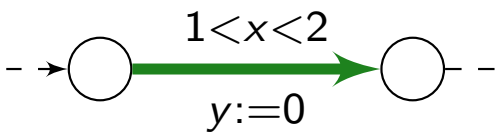
1. **Controller** chooses a delay $d \geq \delta$ and an action a s.t. for every edge $\ell \xrightarrow{g,R} \ell'$, if $\nu + d \models g$ then $\nu + d + \epsilon \models g$ for all $\epsilon \in [-\delta, \delta]$.
2. **Perturbator** resolves non-determinism and chooses $\epsilon \in [-\delta, +\delta]$,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

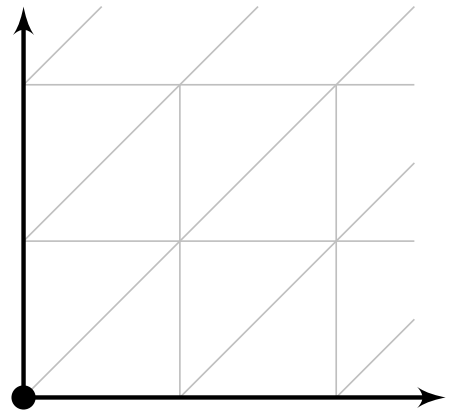
Perturbation Game Semantics

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 $(\ell', (\nu + d + \epsilon)[R \leftarrow 0])$.



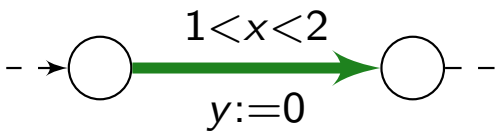


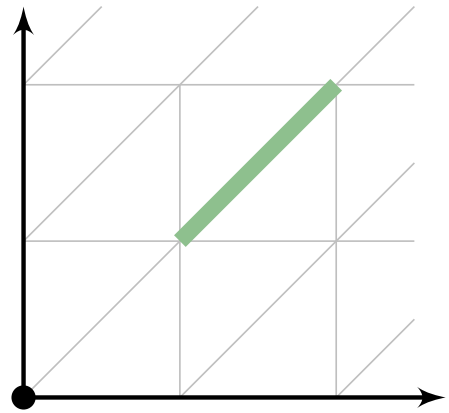
Perturbation Game Semantics

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 $(\ell', (\nu + d + \epsilon)[R \leftarrow 0])$.



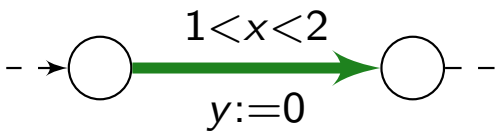


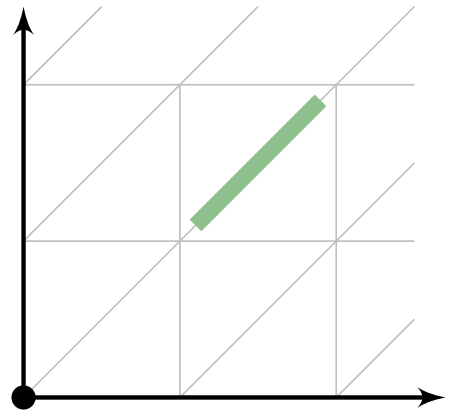
Perturbation Game Semantics

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 $(\ell', (\nu + d + \epsilon)[R \leftarrow 0])$.



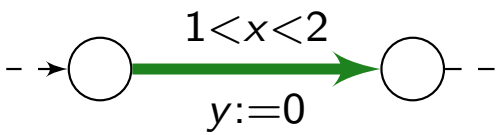


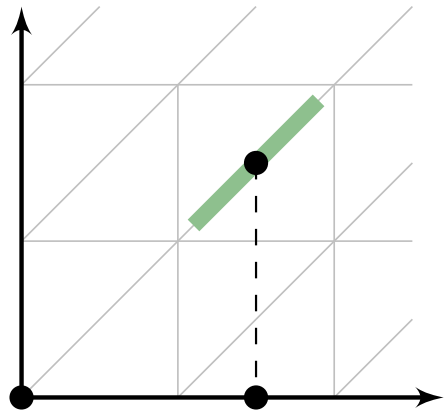
Perturbation Game Semantics

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 $(\ell', (\nu + d + \epsilon)[R \leftarrow 0])$.



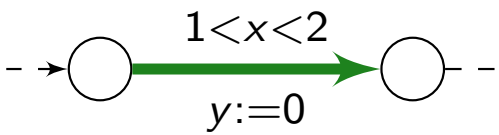


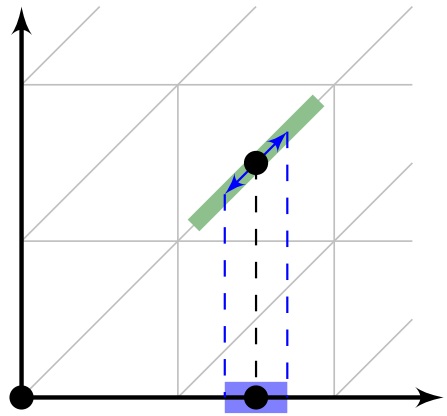
Perturbation Game Semantics

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 Control Problem

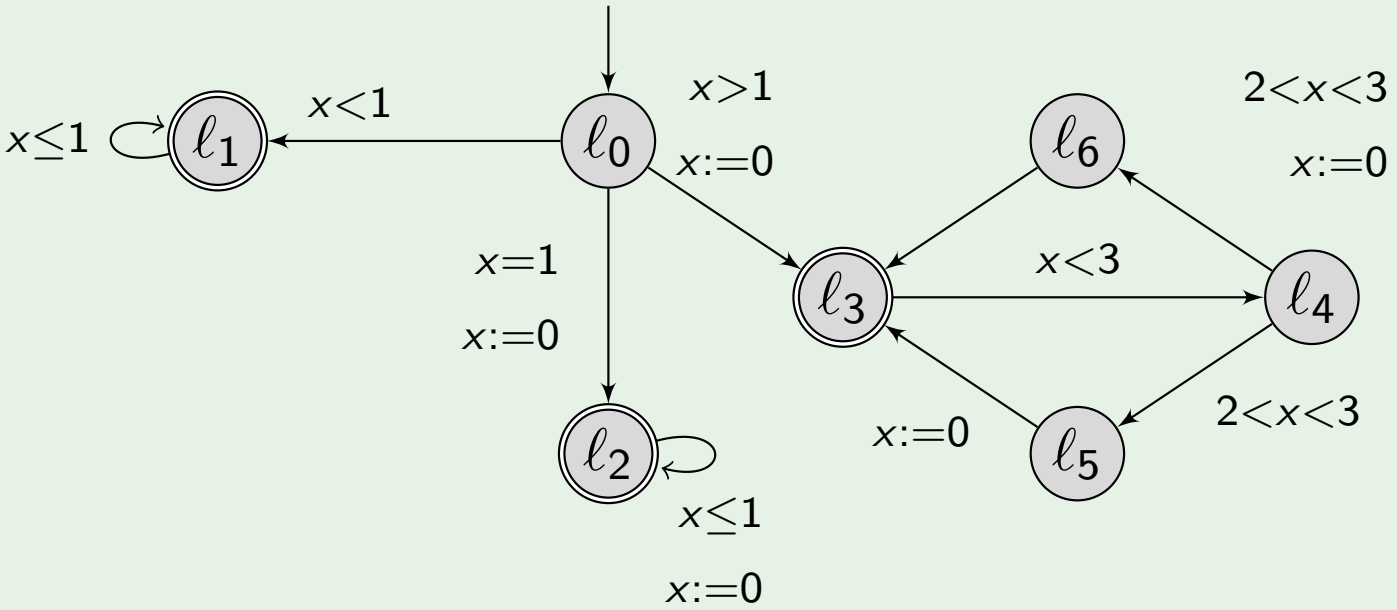
Given a timed automaton A, and a Büchi condition ϕ, decide whether there exists $\delta > 0$ s.t. Controller wins in $\mathcal{G}_\delta(A)$ for ϕ.

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

A (simple?) example

All edges are labelled by action a.

Example

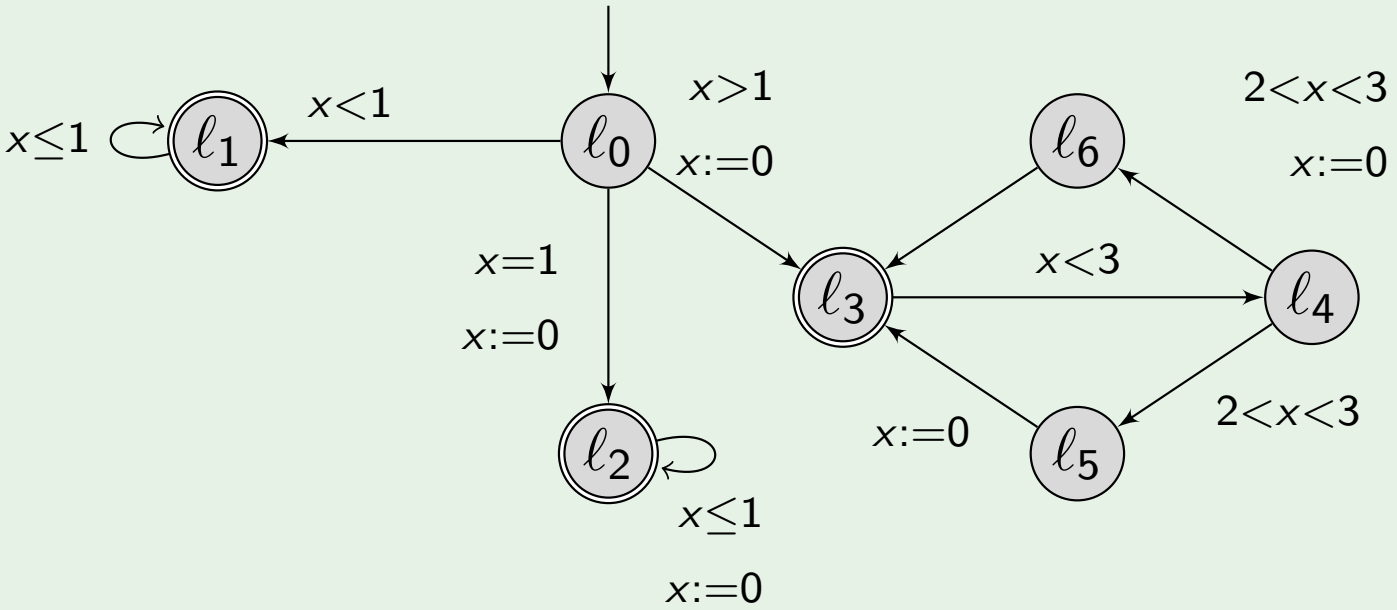


➜ Does Controller has a robust winning strategy?

A (simple?) example

All edges are labelled by action a.

Example



➜ Does Controller has a robust winning strategy?

Yes! Go to ℓ_3. For both choices of Perturbator, x is reset.

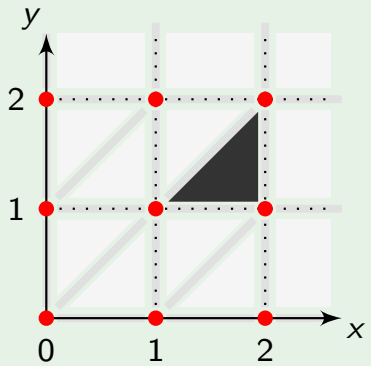
Region Automaton [AD90]

Time-abstract bisimulation:

$(\ell, r) \overset{\rho}{\rightsquigarrow} (\ell', r')$ in $\mathcal{R}(\mathcal{A})$ implies:

- $\forall \nu \in r. \exists \nu' \in r'. (\ell, \nu) \overset{\rho}{\rightsquigarrow}_{\mathcal{A}} (\ell', \nu')$
- $\forall \nu' \in r'. \exists \nu \in r. (\ell, \nu) \overset{\rho}{\rightsquigarrow}_{\mathcal{A}} (\ell', \nu')$

Example



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

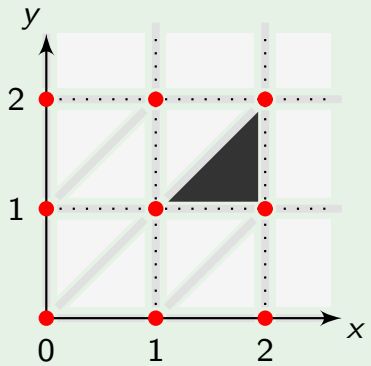
Region Automaton [AD90]

Time-abstract bisimulation:

$(\ell, r) \overset{\rho}{\rightsquigarrow} (\ell', r')$ in $\mathcal{R}(\mathcal{A})$ implies:

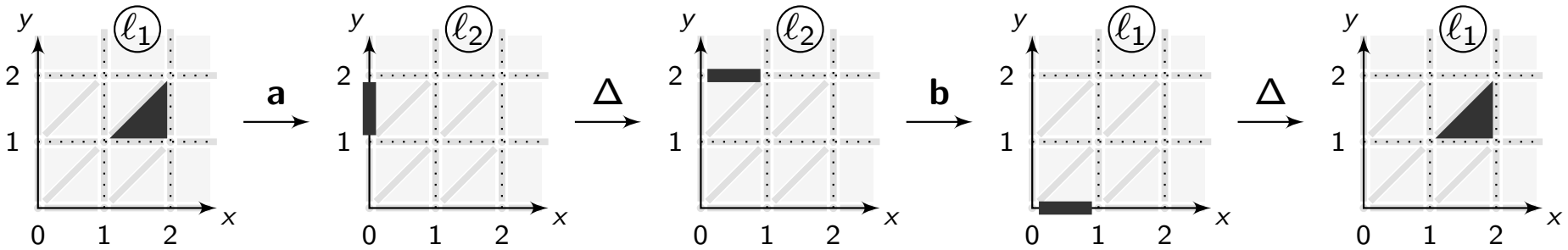
- $\forall \nu \in r. \exists \nu' \in r'. (\ell, \nu) \overset{\rho}{\rightsquigarrow}_{\mathcal{A}} (\ell', \nu')$
- $\forall \nu' \in r'. \exists \nu \in r. (\ell, \nu) \overset{\rho}{\rightsquigarrow}_{\mathcal{A}} (\ell', \nu')$

Example



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

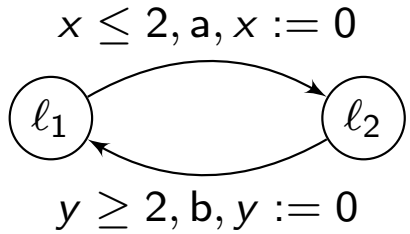
Not precise enough: convergence phenomena are not captured.

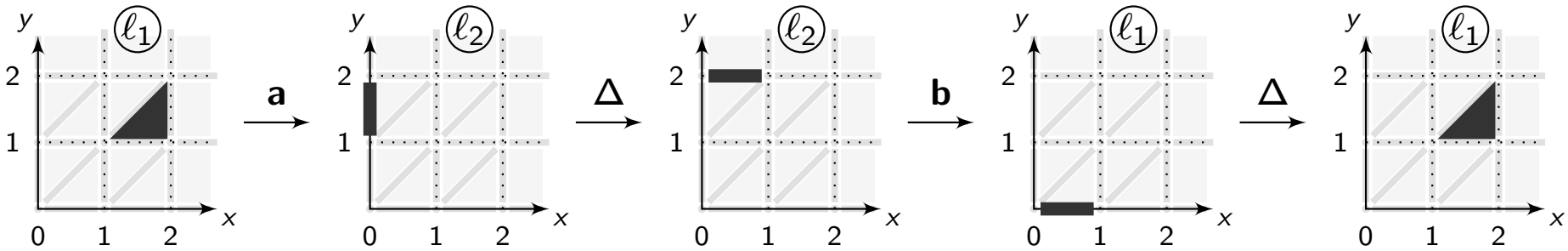


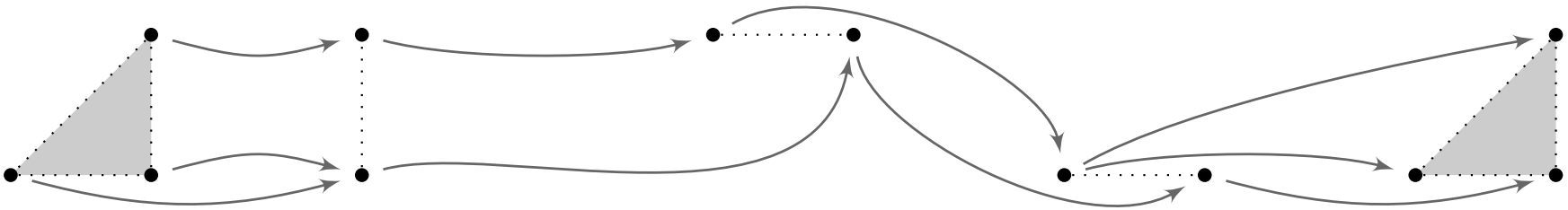
Orbit Graph [Puri00]

Orbit Graph

Stores the reachability relation between vertices of the regions



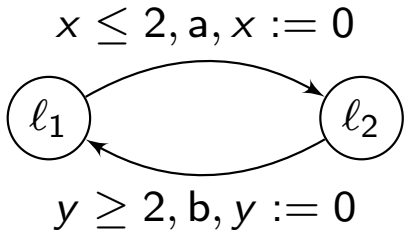


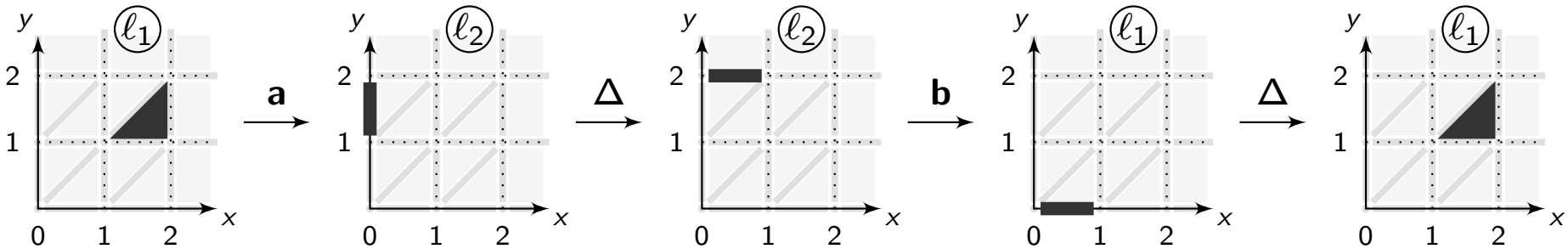


Orbit Graph [Puri00]

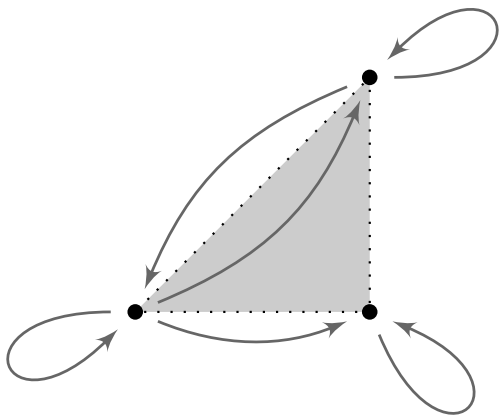
Orbit Graph

Stores the reachability relation between vertices of the regions

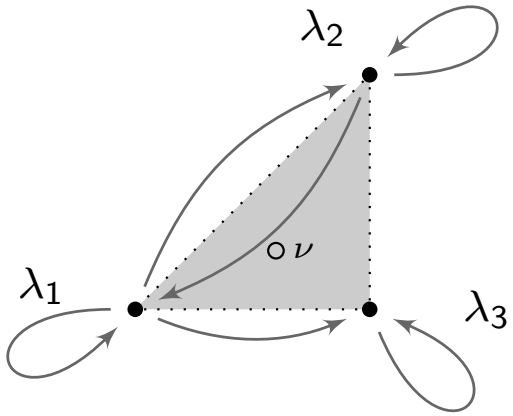




Case of cycles: Folded Orbit Graph (FOG)



Reachability with Orbit Graphs



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

Theorem [Puri00,AB11]

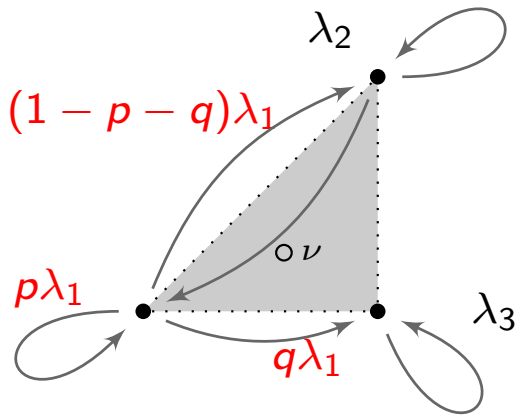
Given a path ρ, and valuation $\vec{\lambda} \cdot \vec{v}$,

$$\vec{\lambda} \cdot \vec{v} \xrightarrow{\rho} \vec{\lambda'}\vec{v}$$

$$\Leftrightarrow$$

$\vec{\lambda'}$ is computed by distributing each λ_v 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 [Puri00,AB11]

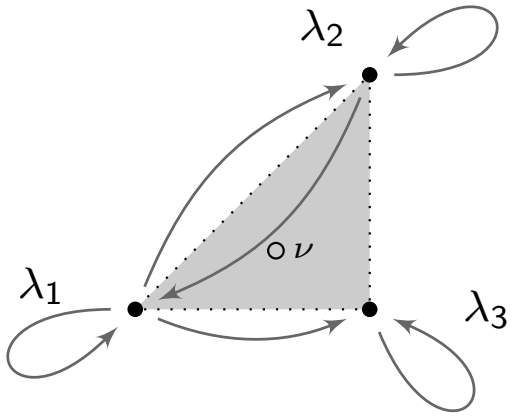
Given a path ρ, and valuation $\vec{\lambda} \cdot \vec{v}$,

$$\vec{\lambda} \cdot \vec{v} \xrightarrow{\rho} \vec{\lambda'}\vec{v}$$

$$\Leftrightarrow$$

$\vec{\lambda'}$ 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 [Puri00,AB11]

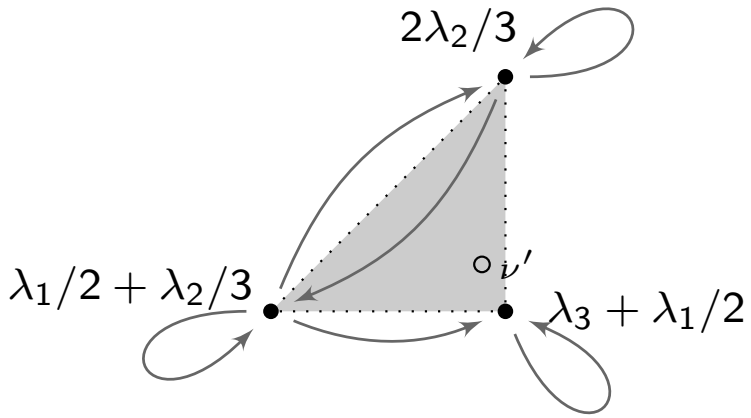
Given a path ρ, and valuation $\vec{\lambda} \cdot \vec{v}$,

$$\vec{\lambda} \cdot \vec{v} \xrightarrow{\rho} \vec{\lambda'}\vec{v}$$

$$\Leftrightarrow$$

$\vec{\lambda'}$ is computed by distributing each λ_v 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 [Puri00,AB11]

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

$$\vec{\lambda} \cdot \vec{v} \xrightarrow{\rho} \vec{\lambda'}\vec{v}$$

$$\Leftrightarrow$$

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

Characterization of complete reachability relations

Definition (Reachability relation)

Given a path $\rho : (\ell, r) \to (\ell', r')$.

$$R(\rho) = \{(\nu, \nu') \in r \times r' \mid (\ell, \nu) \xrightarrow{\rho} (\ell', \nu') \in \mathit{Runs}(\mathcal{A})\}$$

The relation is complete iff $R(\rho) = r \times r'$.

Characterization of complete reachability relations

Definition (Reachability relation)

Given a path $\rho : (\ell, r) \to (\ell', r')$.

$$R(\rho) = \{(\nu, \nu') \in r \times r' \mid (\ell, \nu) \xrightarrow{\rho} (\ell', \nu') \in \mathit{Runs}(\mathcal{A})\}$$

The relation is complete iff $R(\rho) = r \times r'$.

Consider a cycle ρ around some region r.

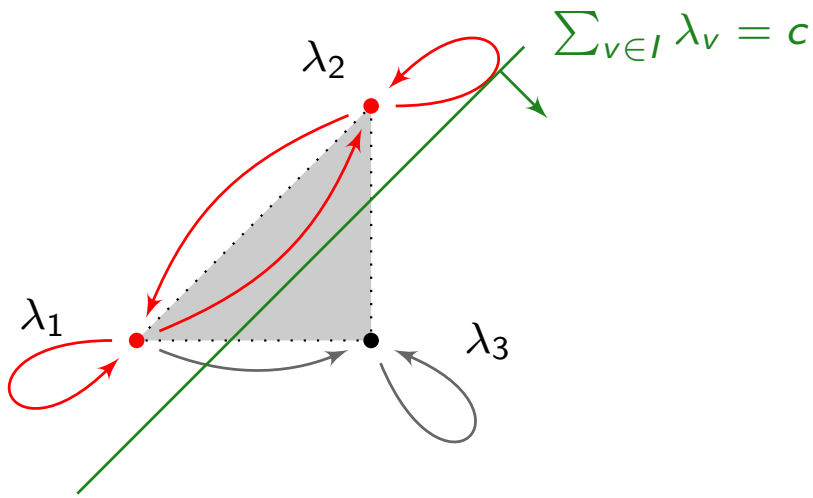
Lemma

The reachability relation of ρ is complete iff the graph $FOG(\rho)$ is complete.

Sufficient condition for convergence [AB11]

Consider a cycle ρ whose FOG is not strongly connected.

➜ then FOG(ρ) has an initial component I.



Lemma

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

The quantity $\sum_{v \in I} \lambda_v$ is **nonincreasing** along a repetition of the cycle.

Characterization of convergence

Definition (Aperiodic [AB11,Stainer12])

A cycle ρ is aperiodic iff for all $k \geq 1$, FOG(ρ^k) is strongly connected

Lemma ([SBMR13])

ρ is tolerant to perturbations iff ρ is aperiodic.

Hints:

- ρ is not aperiodic: there exists k s.t. FOG(ρ^k) is not strongly connected
  ➜ convergence
- ρ is aperiodic: there exists k s.t. FOG(ρ^k) is complete
  ➜ the reachability relation is complete: no convergence

Characterization of convergence

Definition (Aperiodic [AB11,Stainer12])

A cycle ρ is aperiodic iff for all $k \geq 1$, $\mathrm{FOG}(\rho^k)$ is strongly connected

Lemma ([SBMR13])

ρ is tolerant to perturbations iff ρ is aperiodic.

Hints:

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

➜ in order to build robust strategies, look for aperiodic cycles

A New Game on the Region Graph [ORS14]

Remember the two player turn based game played on $\mathcal{R}(\mathcal{A})$:

- Controller suggests a (time-successor) region and an action $\leadsto \sigma_C$
- Perturbator resolves non-determinism $\leadsto \sigma_P$

New Winning Condition $\mathcal{W}$

An infinite play is winning for Controller iff :

- it satisfies the Büchi condition
- it contains infinitely many factors whose FOG is complete

A Controller's strategy σ_C is winning iff for any σ_P, $\rho[\sigma_C, \sigma_P]$ is winning.

A New Game on the Region Graph [ORS14]

Remember the two player turn based game played on $\mathcal{R}(\mathcal{A})$:

- Controller suggests a (time-successor) region and an action $\rightsquigarrow \sigma_C$
- Perturbator resolves non-determinism $\rightsquigarrow \sigma_P$

New Winning Condition $\mathcal{W}$

An infinite play is winning for Controller iff :

- it satisfies the Büchi condition
- it contains infinitely many factors whose FOG is complete

A Controller's strategy σ_C is winning iff for any σ_P, $\rho[\sigma_C, \sigma_P]$ is winning.

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 New Game on the Region Graph [ORS14]

Theorem

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

Corollary

The Robust Control Problem is EXPTIME-complete.

A New Game on the Region Graph [ORS14]

Theorem

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

Corollary

The Robust Control Problem is EXPTIME-complete.

Prook sketch. As $\mathcal{W}$ is a Büchi condition, we have:

- **If** Perturbator **wins:** there exists a **finite memory** strategy σ_P such that no reachable cycle in $\mathcal{R}(\mathcal{A})[\sigma_P]$ is aperiodic and accepting.

  ➜ there exists a not strongly connected FOG which is repeated

  ➜ there is no robust strategy

A New Game on the Region Graph [ORS14]

Theorem

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

Corollary

The Robust Control Problem is EXPTIME-complete.

Prook sketch. As $\mathcal{W}$ is a Büchi condition, we have:

- **If** Perturbator **wins:** there exists a **finite memory** strategy σ_P such that no reachable cycle in $\mathcal{R}(\mathcal{A})[\sigma_P]$ is aperiodic and accepting.
- **If** Controller **wins:** there exists a **finite memory** strategy σ_C such that every reachable cycle in $\mathcal{R}(\mathcal{A})[\sigma_C]$ is aperiodic and accepting.

➜ repeat cycles enough to obtain complete FOGs
➜ for each cycle, we can build $\delta > 0$ and a concrete strategy in $\mathcal{G}_\delta(\mathcal{A})$
➜ combining these strategies yields a robust strategy

Stochastic environment

Our model of perturbation is worst-case. This is not realistic.

➜ What happens if we consider a stochastic model of perturbation?

We consider two models :

- Stochastic perturbations and adversarial resolution of non-determinism of actions
  ➜ yields stochastic timed games
- Both perturbations and non-determinism are resolved randomly
  ➜ yields (infinite state) Markov decision processes

Stochastic Perturbations

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

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. $\mathbb{P}^{\sigma}_{\mathcal{G}_\delta(\mathcal{A})}(F^{\infty} B) = 1$?

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

Theorem

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

Timed automata: nice formalism, but not always possible to transfer correctness from the model to the implemantation

A parametric framework with good properties

Robust Model Checking

- Guard enlargement
- Analyze new behaviours
- Decidability results for several properties

Robust control

- Ensure liveness
- Orbit graphs

Same theoretical complexity

I did not present...

Other approaches for robustness of timed systems:

- Topological approach
- Sampling/Discretization
- Probabilistic semantics

Other works on guard enlargement:

- Shrinkability
- Implementability using the region automaton
- Time Petri nets
- Weighted timed automata
- Another semantics for Robut Control

Perspectives

Compositional verification of robustness

Symbolic techniques, implementation
Orbit graphs for zones

Maximal admissible perturbation

Other timed games (uncontrollable edges, concurrent timed games)

Stochastic perturbations