

Robustness in timed automata

Emily Clement³ Thierry Jéron¹ Nicolas Markey¹ David Mentré²

¹IRISA, Inria & CNRS & Univ. Rennes, France

²Mitsubishi Electric R&D Centre Europe – Rennes, France: MERCE

³ISIR, Sorbonne université, Paris, France

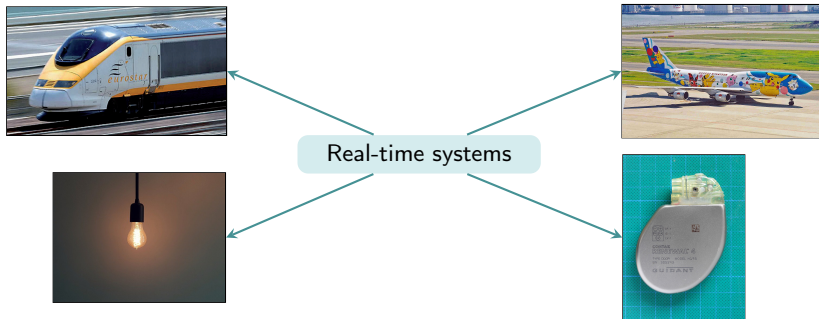
⁴LRDE, EPITA, Paris, France

June 23 2022

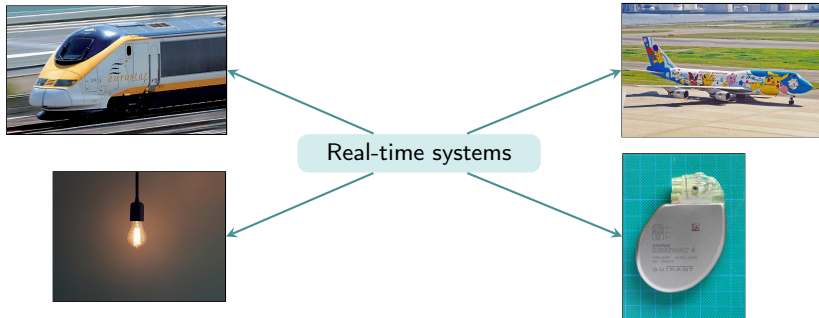
Introduction

Formal methods and model-checking

Why verifying real-time systems?

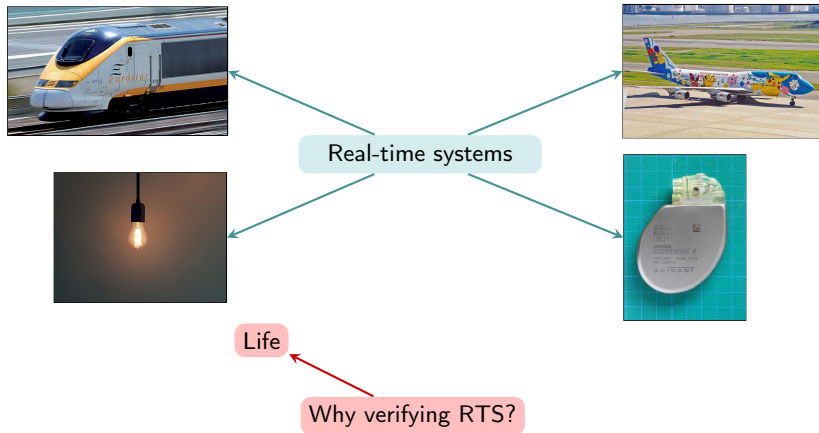


Why verifying real-time systems?

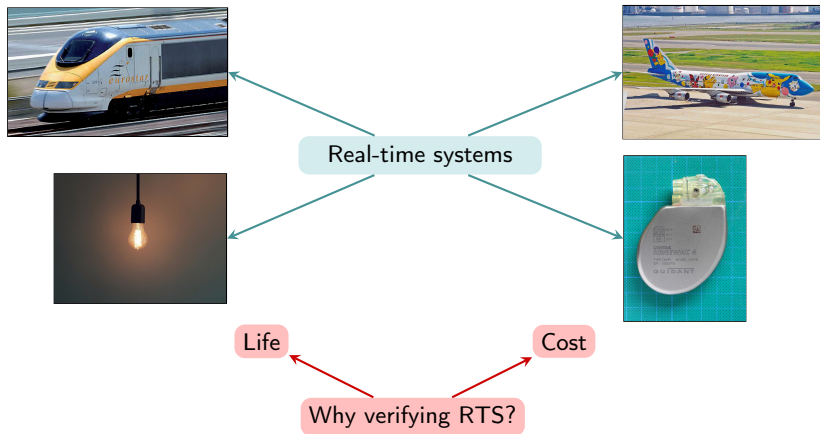


Why verifying RTS?

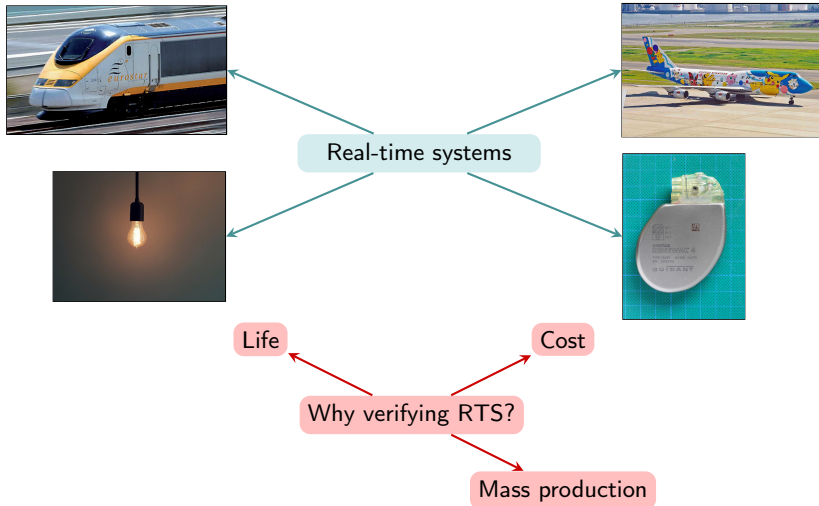
Why verifying real-time systems?



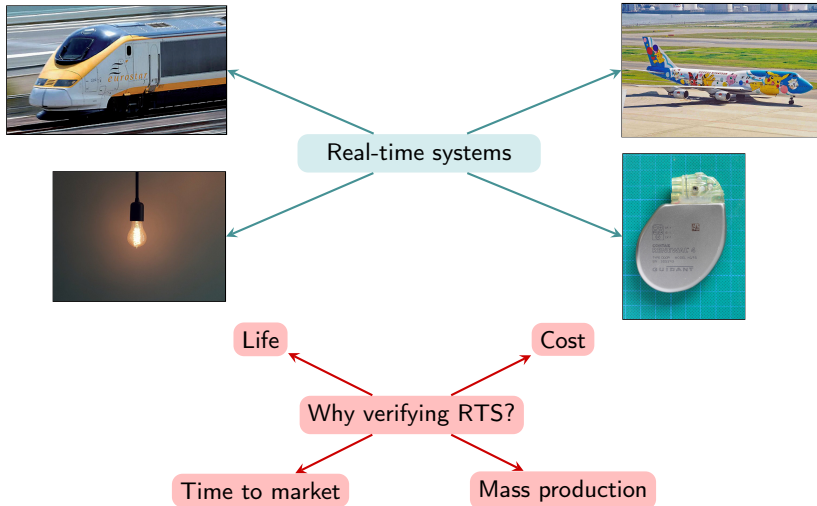
Why verifying real-time systems?

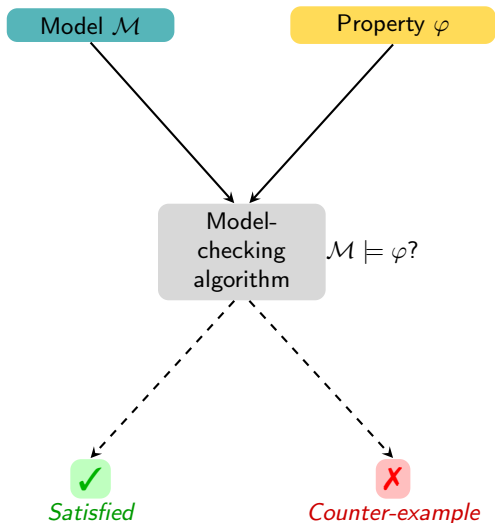


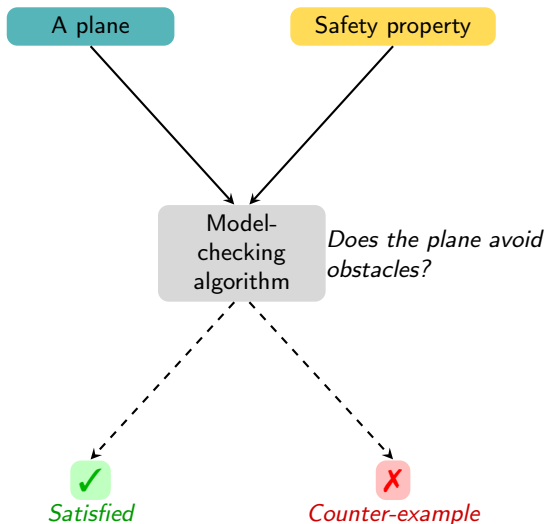
Why verifying real-time systems?



Why verifying real-time systems?







- The model

An abstract
model

Timed automata, timed games, weighted timed automata, timed Petri nets...

- The model

An abstract model

Timed automata, Timed games, weighted timed automata, timed petri nets...

- The property

Reachability

A state **can be** visited
to light on, arrival of a train/plane...

Büchi

A state **can be** visited **infinitely often**

Liveness

A (good) event **will eventually** happen

Safety

A bad event **will never** happen
a car crash, a collision...

- The model

An abstract model

Timed automata, Timed games, weighted timed automata, timed petri nets...

- The property

Reachability

A state **can be** visited
to light on, arrival of a train/plane...

Büchi

A state **can be** visited **infinitely often**

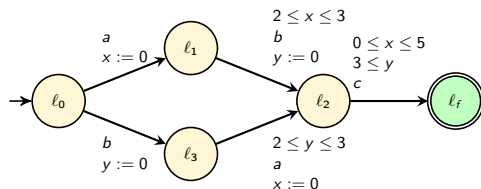
Liveness

A (good) event **will eventually** happen

Safety

A bad event **will never** happen
a car crash, a collision...

- Example: Scheduling system



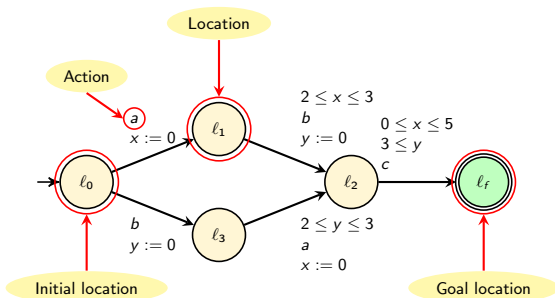
Acyclic timed automata:



Linear timed automata:



- Example: Scheduling system



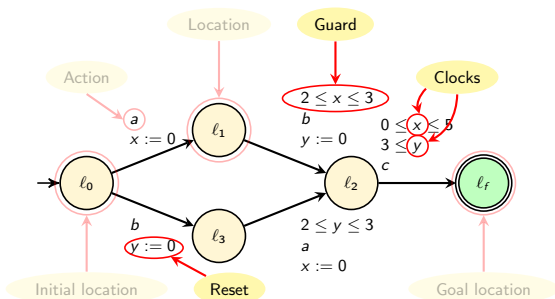
Acyclic timed automata:



Linear timed automata:



- Example: Scheduling system



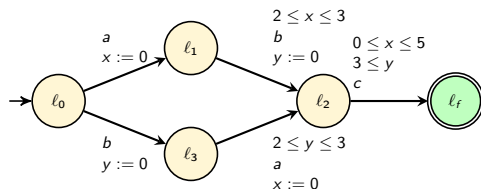
Acyclic timed automata:



Linear timed automata:



- Example: Scheduling system



$$l_0, (0, 0) \xrightarrow{\delta_0=1}$$

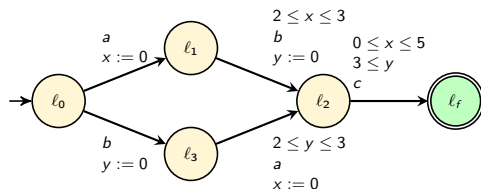
Acyclic timed automata:



Linear timed automata:



- Example: Scheduling system



$$l_0, (0, 0) \xrightarrow{\delta_0=1} l_0, (1, 1)$$

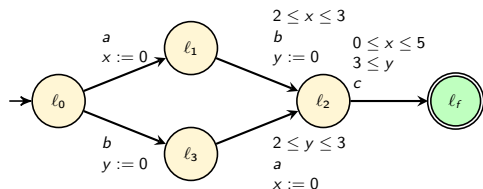
Acyclic timed automata:



Linear timed automata:



- Example: Scheduling system



$$l_0, (0, 0) \xrightarrow{\delta_0=1} l_0, (1, 1) \xrightarrow{a, x:=0}$$

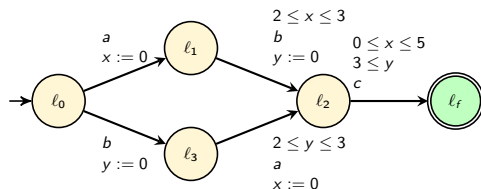
Acyclic timed automata:



Linear timed automata:



- Example: Scheduling system



Acyclic timed automata:

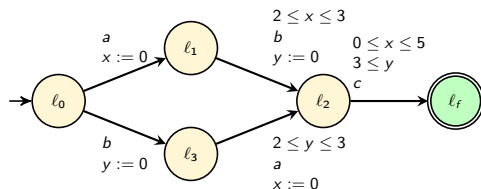


Linear timed automata:



$$l_0, (0, 0) \xrightarrow{\delta_0=1} l_0, (1, 1) \xrightarrow{a, x:=0} l_1, (0, 1)$$

- Example: Scheduling system



Acyclic timed automata:

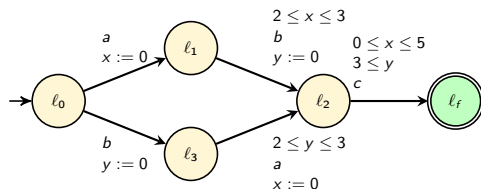


Linear timed automata:



$$l_0, (0, 0) \xrightarrow{\delta_0=1} l_0, (1, 1) \xrightarrow{a, x:=0} l_1, (0, 1) \xrightarrow{\delta_1=2}$$

- Example: Scheduling system



Acyclic timed automata:

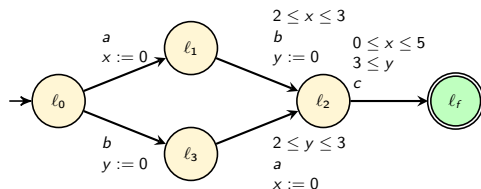


Linear timed automata:



$$l_0, (0, 0) \xrightarrow{\delta_0=1} l_0, (1, 1) \xrightarrow{a, x:=0} l_1, (0, 1) \xrightarrow{\delta_1=2} l_1, (2, 3)$$

- Example: Scheduling system



Acyclic timed automata:

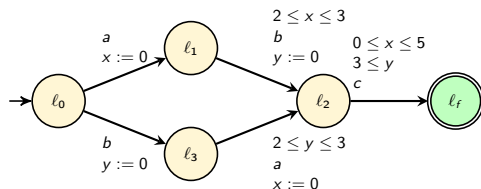


Linear timed automata:



$$l_0, (0, 0) \xrightarrow{\delta_0=1} l_0, (1, 1) \xrightarrow{a, x:=0} l_1, (0, 1) \xrightarrow{\delta_1=2} l_1, (2, 3) \xrightarrow{b, y:=0}$$

- Example: Scheduling system



Acyclic timed automata:

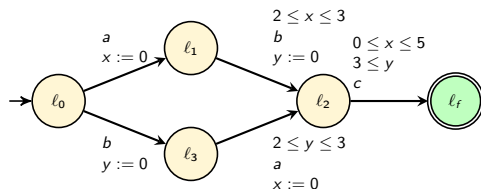


Linear timed automata:



$$l_0, (0, 0) \xrightarrow{\delta_0=1} l_0, (1, 1) \xrightarrow{a, x:=0} l_1, (0, 1) \xrightarrow{\delta_1=2} l_1, (2, 3) \xrightarrow{b, y:=0} l_2, (2, 0)$$

- Example: Scheduling system



Acyclic timed automata:

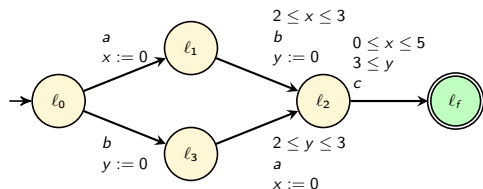


Linear timed automata:



$$l_0, (0, 0) \xrightarrow{\delta_0=1} l_0, (1, 1) \xrightarrow{a, x:=0} l_1, (0, 1) \xrightarrow{\delta_1=2} l_1, (2, 3) \xrightarrow{b, y:=0} l_2, (2, 0) \xrightarrow{\delta_2=3}$$

- Example: Scheduling system



Acyclic timed automata:

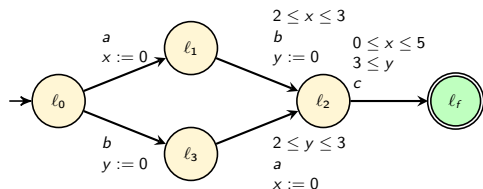


Linear timed automata:



$$l_0, (0, 0) \xrightarrow{\delta_0=1} l_0, (1, 1) \xrightarrow{a, x:=0} l_1, (0, 1) \xrightarrow{\delta_1=2} l_1, (2, 3) \xrightarrow{b, y:=0} l_2, (2, 0) \xrightarrow{\delta_2=3} l_2, (5, 3)$$

- Example: Scheduling system



Acyclic timed automata:

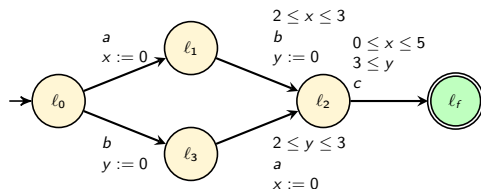


Linear timed automata:



$$l_0, (0, 0) \xrightarrow{\delta_0=1} l_0, (1, 1) \xrightarrow{a, x:=0} l_1, (0, 1) \xrightarrow{\delta_1=2} l_1, (2, 3) \xrightarrow{b, y:=0} l_2, (2, 0) \xrightarrow{\delta_2=3} l_2, (5, 3) \xrightarrow{c}$$

- Example: Scheduling system



Acyclic timed automata:



Linear timed automata:



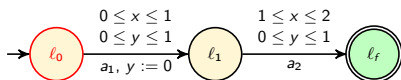
$$l_0, (0, 0) \xrightarrow{\delta_0=1} l_0, (1, 1) \xrightarrow{a, x:=0} l_1, (0, 1) \xrightarrow{\delta_1=2} l_1, (2, 3) \xrightarrow{b, y:=0} l_2, (2, 0) \xrightarrow{\delta_2=3} l_2, (5, 3) \xrightarrow{c} l_f, (5, 3).$$

Robustness in model checking

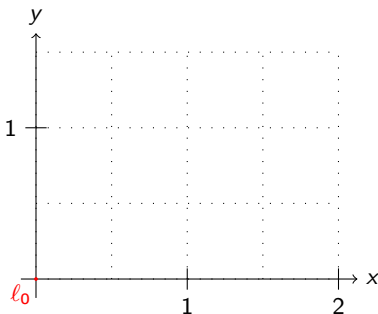
Delay perturbation

Example of perturbed semantics: delay perturbation¹

- Timed automaton \mathcal{A} :



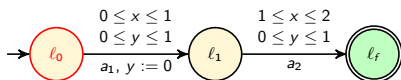
- Run with delay perturbations of at most $\delta = 0.2$



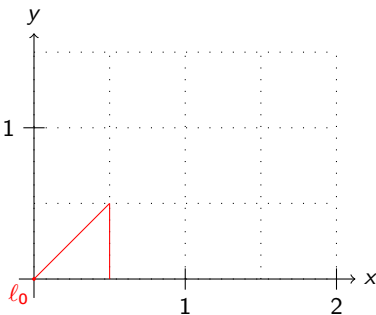
¹BFM15.

Example of perturbed semantics: delay perturbation¹

- Timed automaton \mathcal{A} :



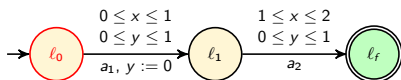
- Run with delay perturbations of at most $\delta = 0.2$



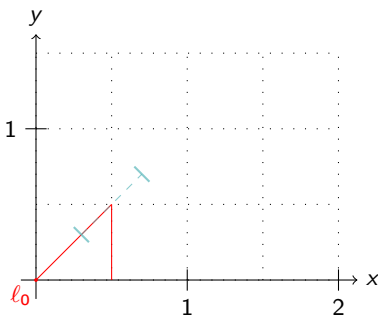
¹BFM15.

Example of perturbed semantics: delay perturbation¹

- Timed automaton \mathcal{A} :



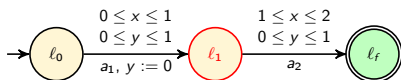
- Run with delay perturbations of at most $\delta = 0.2$



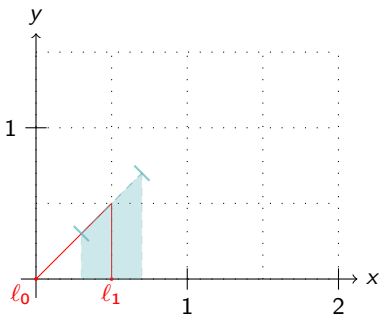
¹BFM15.

Example of perturbed semantics: delay perturbation¹

- Timed automaton \mathcal{A} :



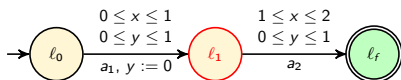
- Run with delay perturbations of at most $\delta = 0.2$



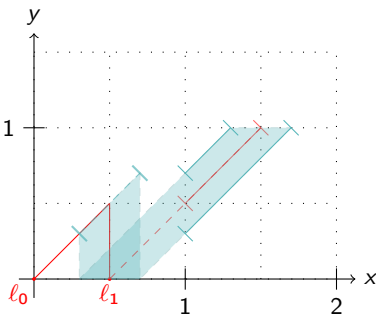
¹BFM15.

Example of perturbed semantics: delay perturbation¹

- Timed automaton \mathcal{A} :



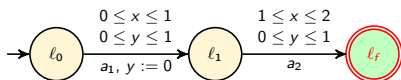
- Run with delay perturbations of at most $\delta = 0.2$



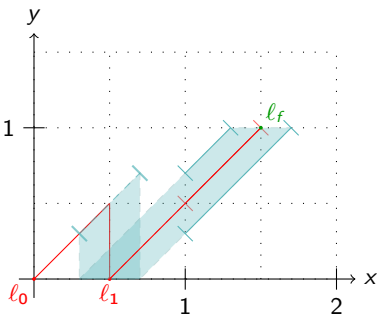
¹BFM15.

Example of perturbed semantics: delay perturbation¹

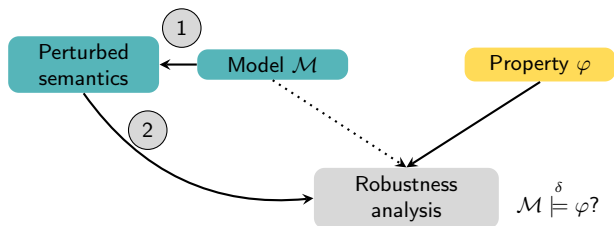
- Timed automaton \mathcal{A} :

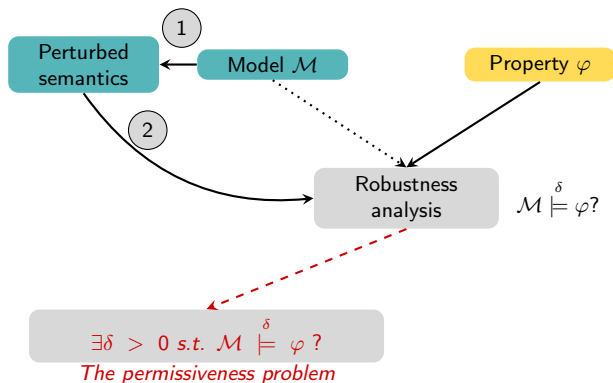


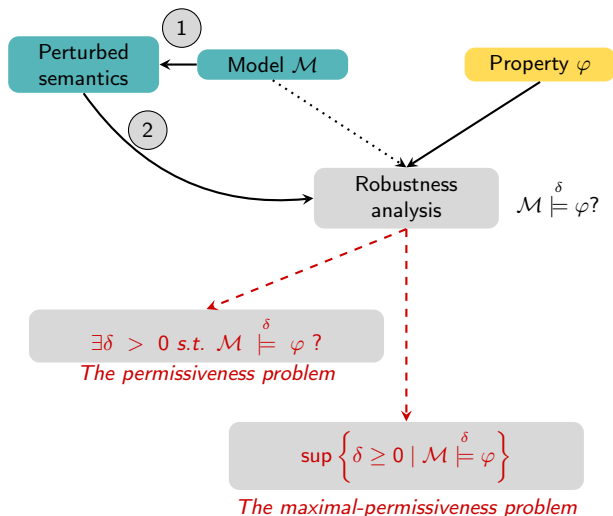
- Run with delay perturbations of at most $\delta = 0.2$

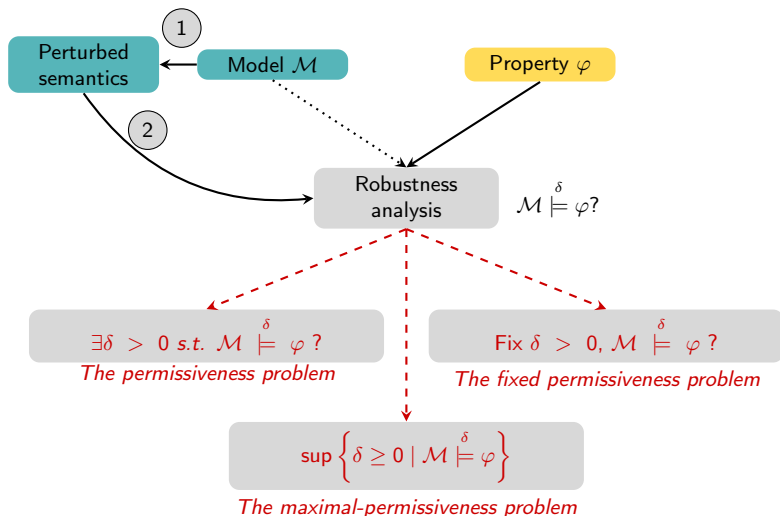


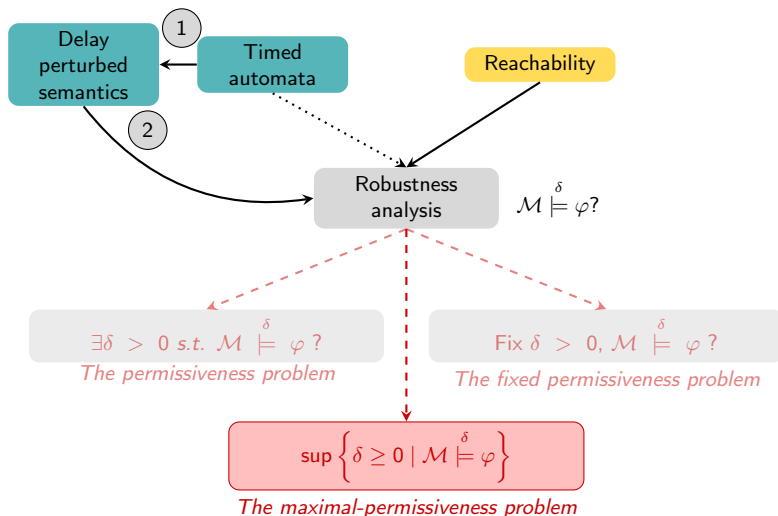
¹BFM15.









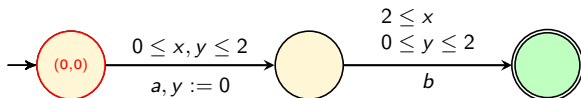


Our model

The permissive semantics

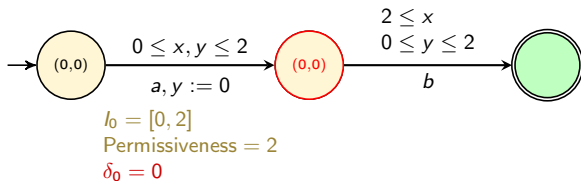
Our permissive semantics: a turn-based game

- ▷ Player
- ▷ Opponent



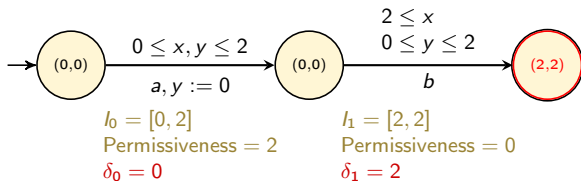
Our permissive semantics: a turn-based game

- ▷ Player
- ▷ Opponent



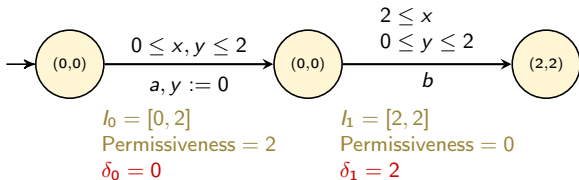
Our permissive semantics: a turn-based game

- ▷ Player
- ▷ Opponent



Our permissive semantics: a turn-based game

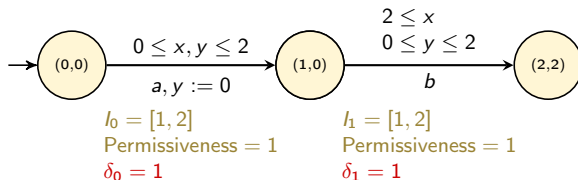
- ▶ **Player** : maximises the permissiveness
- ▶ **Opponent** : minimises the permissiveness



Permissiveness of the run : $\min(|l_0|, |l_1|) = \min(2, 0) = 0$

Our permissive semantics: a turn-based game

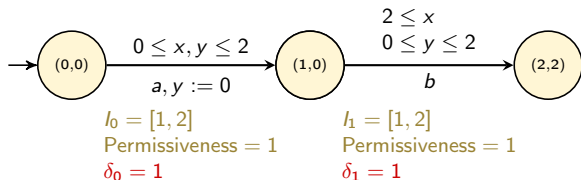
- ▶ **Player** : maximises the permissiveness
- ▶ **Opponent** : minimises the permissiveness



Permissiveness of the run : $\min(|l_0|, |l_1|) = \min(1, 1) = 1$

Our permissive semantics: a turn-based game

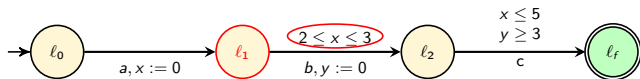
- ▶ **Player** : maximises the permissiveness
- ▶ **Opponent** : minimises the permissiveness



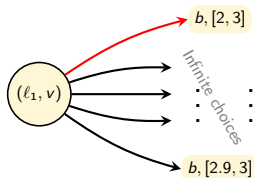
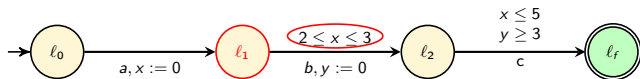
Permissiveness of the run : $\min(|l_0|, |l_1|) = \min(1, 1) = 1$

- ▶ **Opponent** : worst-case environment
- ▶ Our goal: compute the **player** best strategy, whatever the **opponent** decides

Permissiveness: an infinite number of choices



Permissiveness: an infinite number of choices

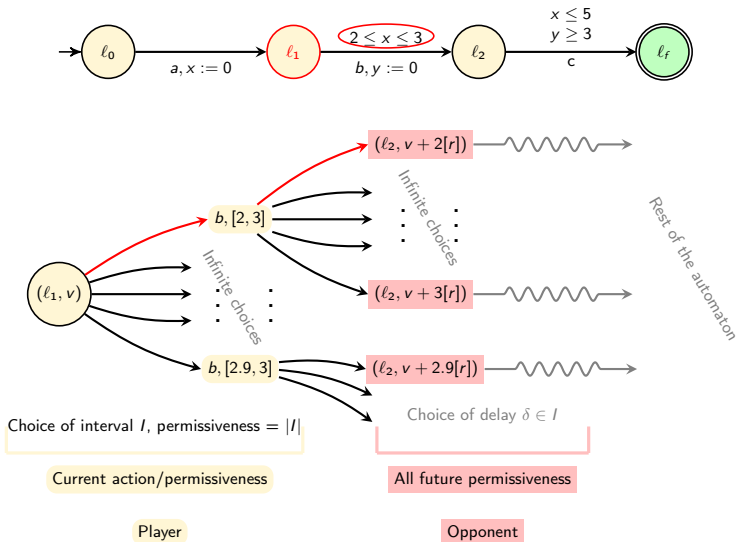


Choice of interval I , permissiveness = $|I|$

Current action/permissiveness

Player

Permissiveness: an infinite number of choices



Our model

Goal and contribution of this thesis

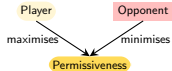
Goal of this thesis: compute the maximal permissiveness

• Permissive semantics: a turn-based game

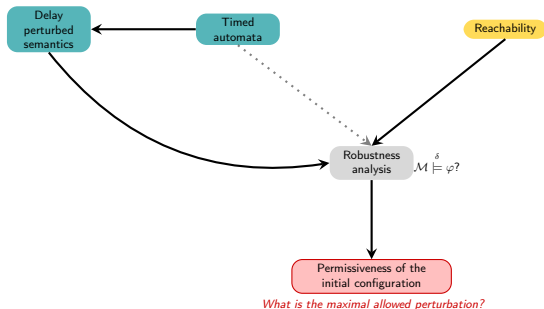
Player : Interval & action: (I, a)

Opponent : delay $\delta \in I$

• Permissiveness



$\min(|I_0|, |I_1|, |I_2|, \dots)$



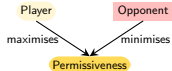
Goal of this thesis: compute the maximal permissiveness

• Permissive semantics: a turn-based game

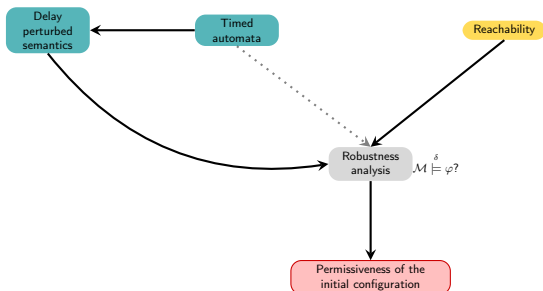
Player : Interval & action: (I, a)

Opponent : delay $\delta \in I$

• Permissiveness



$\min(|I_0|, |I_1|, |I_2|, \dots)$



What is the maximal allowed perturbation?

1st contribution

Symbolic computation of the permissiveness for linear TA, with controlled approximate results

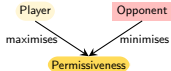
Goal of this thesis: compute the maximal permissiveness

• Permissive semantics: a turn-based game

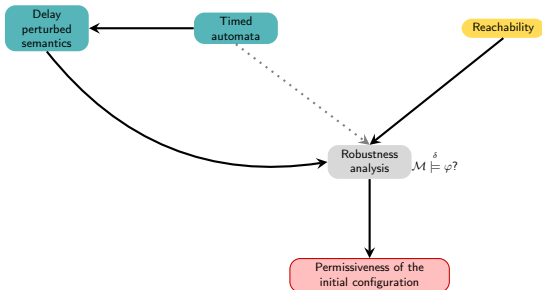
Player : Interval & action: (I, a)

Opponent : delay $\delta \in I$

• Permissiveness



$\min(|I_0|, |I_1|, |I_2|, \dots)$



What is the maximal allowed perturbation?

1st contribution

Symbolic computation of the permissiveness for linear TA, with controlled approximate results

2nd contribution

 Implemented

Symbolic computation of the permissiveness & the strategy of the player for acyclic TA and TG

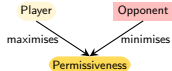
Goal of this thesis: compute the maximal permissiveness

• Permissive semantics: a turn-based game

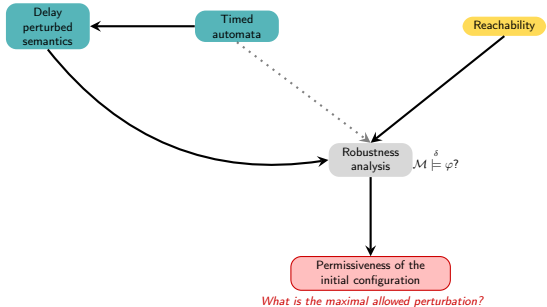
Player : Interval & action: (I, a)

Opponent : delay $\delta \in I$

• Permissiveness



$\min(|I_0|, |I_1|, |I_2|, \dots)$



1st contribution

Symbolic computation of the permissiveness for linear TA, with **controlled approximate results**

2nd contribution

 Implemented

Symbolic computation of the permissiveness & the strategy of the player for acyclic TA and TG

3rd contribution

 Implemented

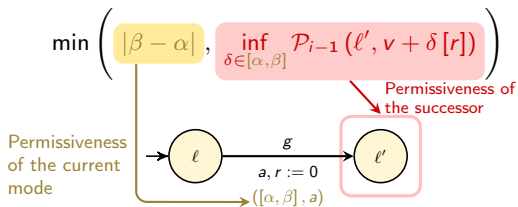
Numerical computation of the permissiveness for acyclic TA, with **approximate results**

The permissiveness function

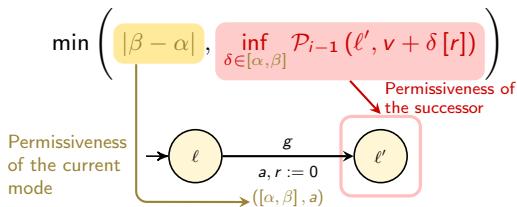
A sequence of suboptimal functions

A suboptimal sequence of functions

- Strategy of the player: maximises



- Strategy of the player: maximises



- Opponent strategy lemma (linear case):

player : $([\alpha, \beta], a) \xrightarrow{\text{Opponent's best strategy}} \alpha \text{ or } \beta$

$$\inf_{\delta \in [\alpha, \beta]} \mathcal{P}_{i-1}(l', v + \delta[r]) = \min \left(\mathcal{P}_{i-1}(l', v + \alpha[r]), \mathcal{P}_{i-1}(l', v + \beta[r]) \right)$$

- A recursive function

$$\mathcal{P}_i(\ell, \nu) = \sup_{([\alpha, \beta], a) \in \text{p-moves}(\ell, \nu)} \left(\min \left(\beta - \alpha, \inf_{\delta \in [\alpha, \beta]} \mathcal{P}_{i-1}(\ell', \nu + \delta[r]) \right) \right)$$

- A recursive function

$$\mathcal{P}_i(\ell, \nu) = \sup_{([\alpha, \beta], a) \in \mathbf{p}\text{-moves}(\ell, \nu)} \left(\min \left(\beta - \alpha, \inf_{\delta \in [\alpha, \beta]} \mathcal{P}_{i-1}(\ell', \nu + \delta[r]) \right) \right)$$

For linear timed automata:

$$\mathcal{P}_i(\ell, \nu) = \sup_{([\alpha, \beta], a) \in \mathbf{p}\text{-moves}(\ell, \nu)} \left(\min \left(\beta - \alpha, \mathcal{P}_{i-1}(\ell', \nu + \alpha[r]), \mathcal{P}_{i-1}(\ell', \nu + \beta[r]) \right) \right)$$

$$\lim_{i \rightarrow +\infty} \mathcal{P}_i(\ell, \nu) = \text{the permissiveness on } (\ell, \nu)$$

- A recursive function

$$\mathcal{P}_i(\ell, \nu) = \sup_{([\alpha, \beta], a) \in \text{p-moves}(\ell, \nu)} \left(\min \left(\beta - \alpha, \inf_{\delta \in [\alpha, \beta]} \mathcal{P}_{i-1}(\ell', \nu + \delta[r]) \right) \right)$$

For linear timed automata:

$$\mathcal{P}_i(\ell, \nu) = \sup_{([\alpha, \beta], a) \in \text{p-moves}(\ell, \nu)} \left(\min \left(\beta - \alpha, \mathcal{P}_{i-1}(\ell', \nu + \alpha[r]), \mathcal{P}_{i-1}(\ell', \nu + \beta[r]) \right) \right)$$

$\lim_{i \rightarrow +\infty} \mathcal{P}_i(\ell, \nu) =$ the permissiveness on (ℓ, ν)

limit reached in d_ℓ steps

- A recursive function

$$\mathcal{P}_i(\ell, v) = \sup_{([\alpha, \beta], a) \in \text{p-moves}(\ell, v)} \left(\min \left(\beta - \alpha, \inf_{\delta \in [\alpha, \beta]} \mathcal{P}_{i-1}(\ell', v + \delta[r]) \right) \right)$$

For linear timed automata:

$$\mathcal{P}_i(\ell, v) = \sup_{([\alpha, \beta], a) \in \text{p-moves}(\ell, v)} \left(\min \left(\beta - \alpha, \mathcal{P}_{i-1}(\ell', v + \alpha[r]), \mathcal{P}_{i-1}(\ell', v + \beta[r]) \right) \right)$$

$\lim_{i \rightarrow +\infty} \mathcal{P}_i(\ell, v) =$ the permissiveness on (ℓ, v)

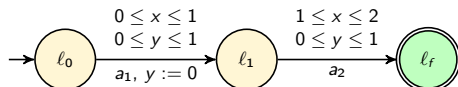
limit reached in d_ℓ steps

- Goal

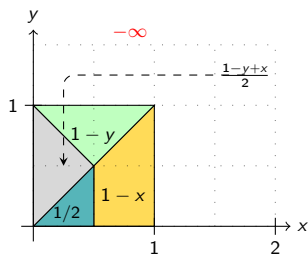
Compute $v \mapsto \mathcal{P}_{d_\ell}(\ell, v)$ knowing $v \mapsto \mathcal{P}_{d_\ell-1}(\ell', v)$

Example of permissiveness function

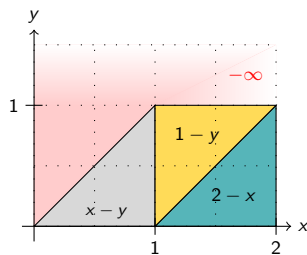
- A linear timed automaton



- Its permissiveness on l_0 and l_1



(a) Permissiveness on l_0



(b) Permissiveness on l_1

How to compute the permissiveness function ?

- Goal: find the α and β that maximises:

$$\min \left(|\beta - \alpha|, \mathcal{P}_{i-1}(\ell', v + \alpha[r]), \mathcal{P}_{i-1}(\ell', v + \beta[r]) \right)$$

How to compute the permissiveness function ?

- Goal: find the α and β that maximises:

$$\min \left(|\beta - \alpha|, \mathcal{P}_{i-1}(\ell', v + \alpha[r]), \mathcal{P}_{i-1}(\ell', v + \beta[r]) \right)$$

How to compute the permissiveness function ?

- Goal: find the α and β that maximises:

$$\min \left(|\beta - \alpha|, \mathcal{P}_{i-1}(\ell', v + \alpha[r]), \mathcal{P}_{i-1}(\ell', v + \beta[r]) \right)$$

- $v \mapsto \mathcal{P}_i(\ell, v)$: a 2-Lipschitz piecewise-affine function

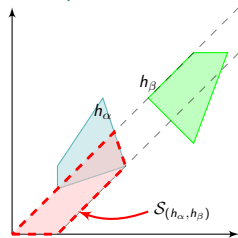
How to compute the permissiveness function ?

- Goal: find the α and β that maximises:

$$\min \left(|\beta - \alpha|, \mathcal{P}_{i-1}(\ell', v + \alpha[r]), \mathcal{P}_{i-1}(\ell', v + \beta[r]) \right)$$

- $v \mapsto \mathcal{P}_i(\ell, v)$: a 2-Lipschitz piecewise-affine function

- Steps: for each couple of cells (h_α, h_β)



(a) Step 1: compute $S(h_\alpha, h_\beta)$

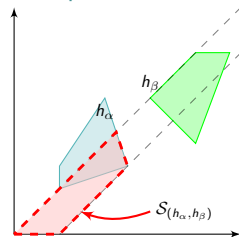
How to compute the permissiveness function ?

- Goal: find the α and β that maximises:

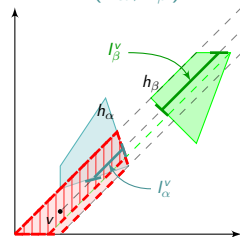
$$\min \left(|\beta - \alpha|, \mathcal{P}_{i-1}(\ell', v + \alpha[r]), \mathcal{P}_{i-1}(\ell', v + \beta[r]) \right)$$

- $v \mapsto \mathcal{P}_i(\ell, v)$: a 2-Lipschitz piecewise-affine function

- Steps: for each couple of cells (h_α, h_β)



(a) Step 1: compute $S(h_\alpha, h_\beta)$



(b) Step 2: compute the possible α and β

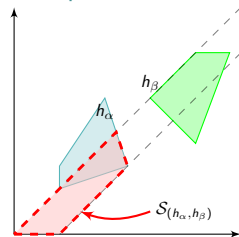
How to compute the permissiveness function ?

- Goal: find the α and β that maximises:

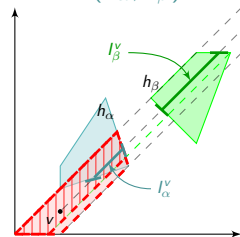
$$\min \left(|\beta - \alpha|, \mathcal{P}_{i-1}(\ell', v + \alpha[r]), \mathcal{P}_{i-1}(\ell', v + \beta[r]) \right)$$

- $v \mapsto \mathcal{P}_i(\ell, v)$: a 2-Lipschitz piecewise-affine function

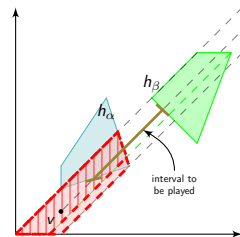
- Steps: for each couple of cells (h_α, h_β)



(a) Step 1: compute $S(h_\alpha, h_\beta)$



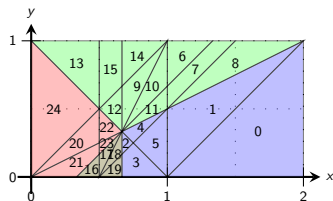
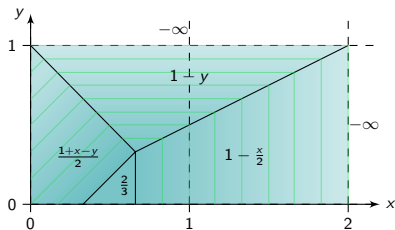
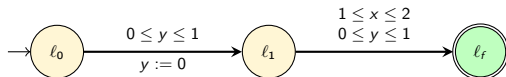
(b) Step 2: compute the possible α and β



(c) Step 3: compute the optimal α and β

Why compute an approximation of the permissiveness function?

- Exact symbolic algorithm
 - A non-elementary algorithm
 - A symbolic computation
 - An overtilling computation:



Symbolic computation with controlled approximated results

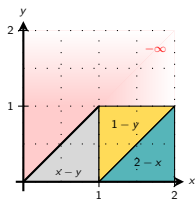
Computing binary and levelled permissiveness

- ▷ A symbolic computation
- ▷ A controlled approximate value
- ▷ Reduce the number of cells

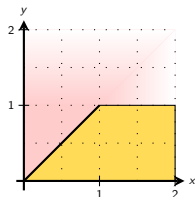
- Binary permissiveness principle

Fix $p \geq 0$ and ℓ , compute $\mathcal{S}(\ell, p) = \{v \mid \text{permissiveness of } (\ell, v) \text{ is greater than } p\}$

- Permissiveness function



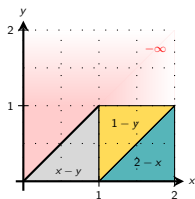
- Binary permissiveness, $p = 0$



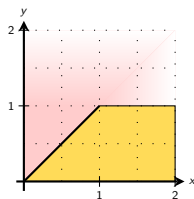
- Binary permissiveness principle

Fix $p \geq 0$ and ℓ , compute $\mathcal{S}(\ell, p) = \{v \mid \mathcal{P}_{d_\ell}(\ell, v) \geq p\}$

- Permissiveness function



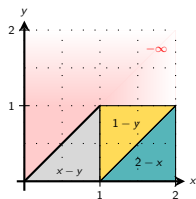
- Binary permissiveness, $p = 0$



- Binary permissiveness principle

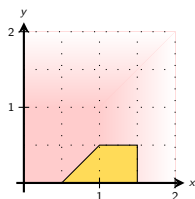
Fix $p \geq 0$ and ℓ , compute $\mathcal{S}(\ell, p) = \{v \mid \mathcal{P}_{d_\ell}(\ell, v) \geq p\}$

- Permissiveness function



- Some reductions

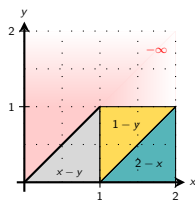
- Binary permissiveness, $p = 1/2$



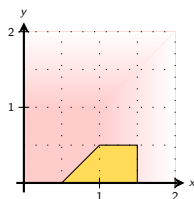
- Binary permissiveness principle

Fix $p \geq 0$ and ℓ , compute $\mathcal{S}(\ell, p) = \{v \mid \mathcal{P}_{d_\ell}(\ell, v) \geq p\}$

- Permissiveness function



- Binary permissiveness, $p = 1/2$



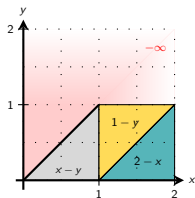
- Some reductions

▷ **Linear Lemma**: for linear TA, $\mathcal{S}(\ell, p)$ is a polyhedron.

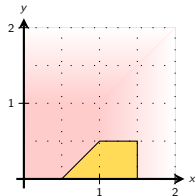
- Binary permissiveness principle

Fix $p \geq 0$ and ℓ , compute $\mathcal{S}(\ell, p) = \{v \mid \mathcal{P}_{d_\ell}(\ell, v) \geq p\}$

- Permissiveness function



- Binary permissiveness, $p = 1/2$



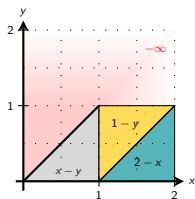
- Some reductions

▷ **Linear Lemma**: for linear TA, $\mathcal{S}(\ell, p)$ is a polyhedron.

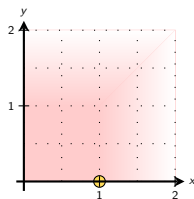
- Binary permissiveness principle

Fix $p \geq 0$ and ℓ , compute $\mathcal{S}(\ell, p) = \{v \mid \mathcal{P}_{d_\ell}(\ell, v) \geq p\}$

- Permissiveness function



- Binary permissiveness, $p = 1$



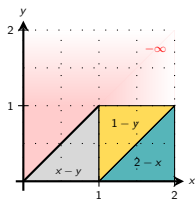
- Some reductions

▷ **Linear Lemma**: for linear TA, $\mathcal{S}(\ell, p)$ is a polyhedron.

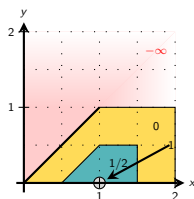
- Binary permissiveness principle

Fix $p \geq 0$ and ℓ , compute $\mathcal{S}(\ell, p) = \{v \mid \mathcal{P}_{d_\ell}(\ell, v) \geq p\}$

- Permissiveness function



- Levelled permissiveness for $\{0, 1/2, 1\}$

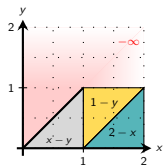


- Some reductions

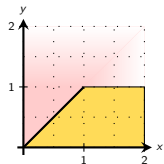
▷ **Linear Lemma**: for linear TA, $\mathcal{S}(\ell, p)$ is a polyhedron.

▷ **Levelled permissiveness** $\xrightarrow{\text{reduces}}$ **binary permissiveness**

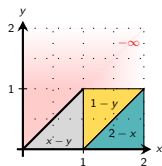
- Permissiveness



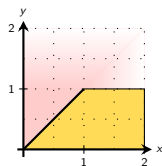
- Binary, $p = 0$



- Permissiveness



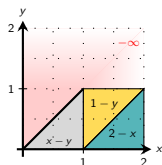
- Binary, $p = 0$



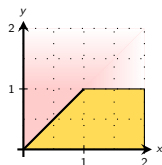
- Sequence of suboptimal permissiveness functions

$$\mathcal{P}_i(\ell, v) = \sup_{([\alpha, \beta], a) \in \mathbf{p}\text{-moves}(\ell, v)} \min(\beta - \alpha, \mathcal{P}_{i-1}(\ell', v + \alpha[r]), \mathcal{P}_{i-1}(\ell', v + \beta[r]))$$

- Permissiveness



- Binary, $p = 0$



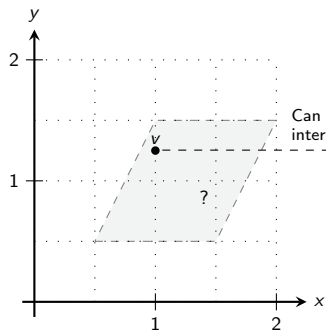
- Sequence of suboptimal permissiveness functions

$$\mathcal{P}_i(\ell, v) = \sup_{([\alpha, \beta], a) \in \mathbf{p}\text{-moves}(\ell, v)} \min(\beta - \alpha, \mathcal{P}_{i-1}(\ell', v + \alpha[r]), \mathcal{P}_{i-1}(\ell', v + \beta[r]))$$

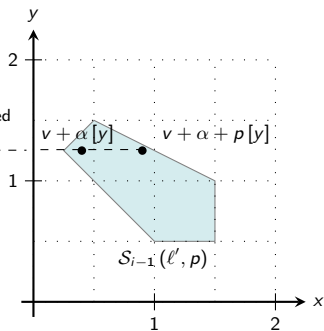
- Sequence of suboptimal binary function

$$\mathcal{B}_i(\ell, v) = \sup_{([\alpha, \alpha+p], a) \in \mathbf{p}\text{-moves}(\ell, v)} \inf_{\delta \in [\alpha, \alpha+p]} (\mathbb{1}_{v+\delta[r] \in \mathcal{S}_{i-1}(\ell'_a, p)}(\ell, v, \delta, p))$$

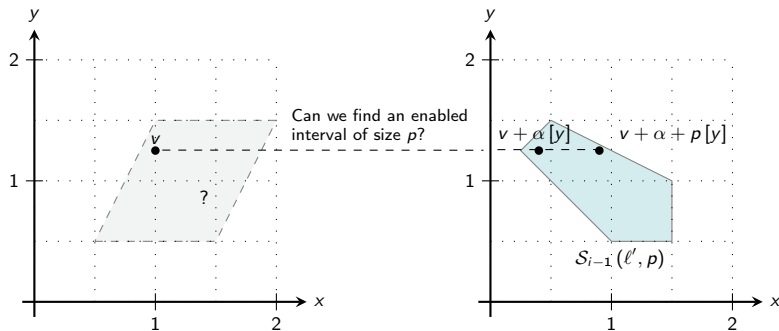
Algorithm intuition: compute $\mathcal{S}_i(\ell, \rho)$ knowing $\mathcal{S}_{i-1}(\ell', \rho)$



Can we find an enabled interval of size ρ ?



Algorithm intuition: compute $\mathcal{S}_i(\ell, \rho)$ knowing $\mathcal{S}_{i-1}(\ell', \rho)$



• Steps:

- ▶ Compute $\mathcal{S}_{i-1}(\ell', \rho)$
- ▶ Fourier-Motzkin: compute the set of v s.t. there exists $\alpha \geq 0$ s.t.:

$$\begin{aligned} v + \alpha, v + \alpha + \rho &\models g \\ v + \alpha[y], v + \alpha + \rho[y] &\in \mathcal{S}_{i-1}(\ell', \rho) \end{aligned}$$

- The principle of the algorithm

$$\begin{aligned}x + \alpha - 1 &\geq 0 \\x + y + \alpha - 3 &\geq 0\end{aligned}$$

System of linear inequalities (S)

- The principle of the algorithm

$$\begin{array}{l} x + \alpha - 1 \geq 0 \\ x + y + \alpha - 3 \geq 0 \end{array}$$

Fourier-Motzkin
Eliminate α

System of linear inequalities (S)

- The principle of the algorithm

$$\begin{array}{l} x + \alpha - 1 \geq 0 \\ x + y + \alpha - 3 \geq 0 \end{array}$$

System of linear inequalities (S)

Fourier-Motzkin
Eliminate α

$$-x + 1 \leq x + y - 3$$

*System of linear equations of
 $\{(x, y) \mid \exists \alpha \text{ s.t. } (x, y, \alpha) \text{ verify } (S)\}$*

- The principle of the algorithm

$$\begin{array}{c} x + \alpha - 1 \geq 0 \\ x + y + \alpha - 3 \geq 0 \end{array} \xrightarrow[\text{Eliminate } \alpha]{\text{Fourier-Motzkin}} -x + 1 \leq x + y - 3$$

System of linear inequalities (S)

*System of linear equations of
 $\{(x, y) \mid \exists \alpha \text{ s.t. } (x, y, \alpha) \text{ verify } (S)\}$*

- Computing the set of $(x, y) \in \mathbb{R}_+^2$ s.t. $\exists \alpha$ s.t.

- The principle of the algorithm

$$\begin{array}{ccc} \boxed{\begin{array}{l} x + \alpha - 1 \geq 0 \\ x + y + \alpha - 3 \geq 0 \end{array}} & \xrightarrow[\text{Eliminate } \alpha]{\text{Fourier-Motzkin}} & \boxed{-x + 1 \leq x + y - 3} \end{array}$$

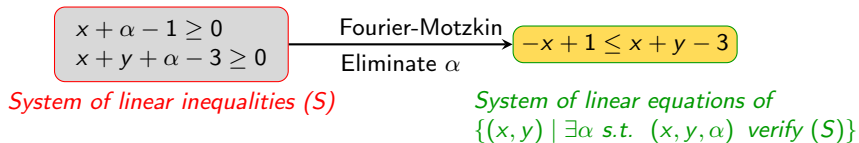
System of linear inequalities (S)

*System of linear equations of
 $\{(x, y) \mid \exists \alpha \text{ s.t. } (x, y, \alpha) \text{ verify (S)}\}$*

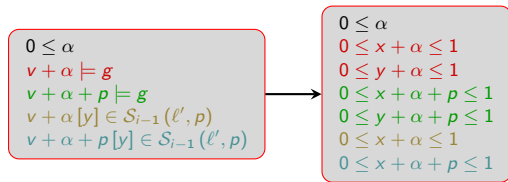
- Computing the set of $(x, y) \in \mathbb{R}_+^2$ s.t. $\exists \alpha$ s.t.

$$\begin{array}{l} 0 \leq \alpha \\ v + \alpha \models g \\ v + \alpha + p \models g \\ v + \alpha[y] \in S_{i-1}(\ell', p) \\ v + \alpha + p[y] \in S_{i-1}(\ell', p) \end{array}$$

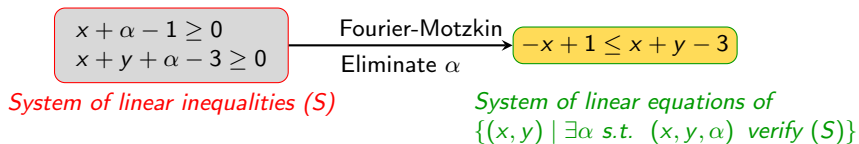
- The principle of the algorithm



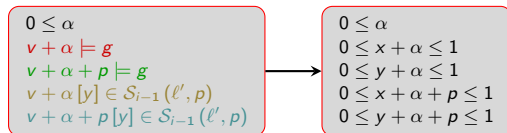
- Computing the set of $(x, y) \in \mathbb{R}_+^2$ s.t. $\exists \alpha$ s.t.



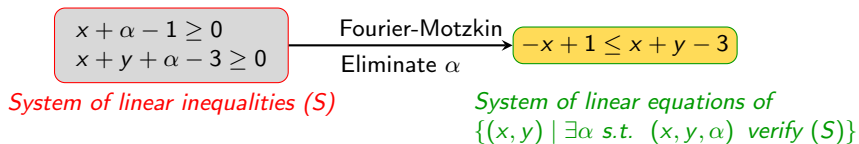
- The principle of the algorithm



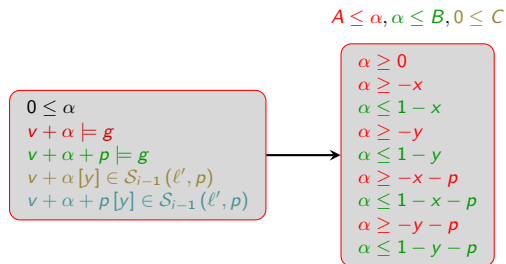
- Computing the set of $(x, y) \in \mathbb{R}_+^2$ s.t. $\exists \alpha$ s.t.



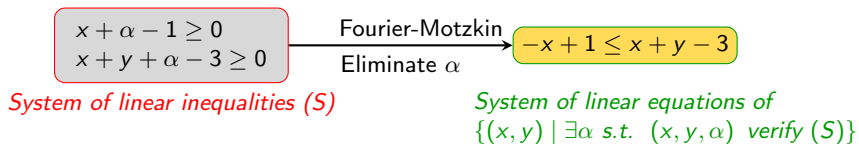
- The principle of the algorithm



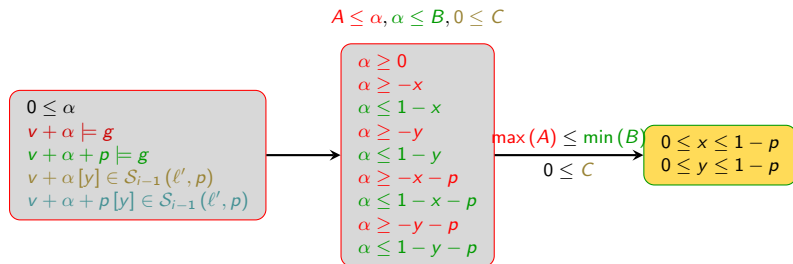
- Computing the set of $(x, y) \in \mathbb{R}_+^2$ s.t. $\exists \alpha$ s.t.



- The principle of the algorithm



- Computing the set of $(x, y) \in \mathbb{R}_+^2$ s.t. $\exists \alpha$ s.t.



- Upper bound complexity for..

Binary permissiveness algorithm:

$$\mathcal{O}\left((4c_g)^{2 \cdot d_\ell}\right)$$

longest path between
 ℓ and a goal location

maximal number of constraints of any guard

- Upper bound complexity for..

Binary permissiveness algorithm:

$$\mathcal{O}\left((4c_g)^{2 \cdot d_\ell}\right)$$

longest path between
 ℓ and a goal location

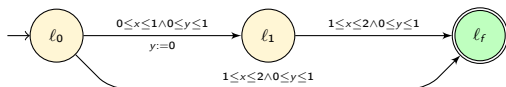
maximal number of constraints of any guard

Levelled permissiveness algorithm $\{p_0, \dots, p_m\}$:

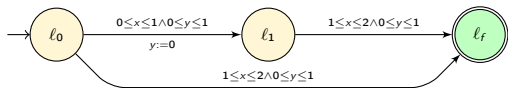
$$\mathcal{O}\left((m+1)(4c_g)^{2 \cdot d_\ell}\right)$$

number of levels

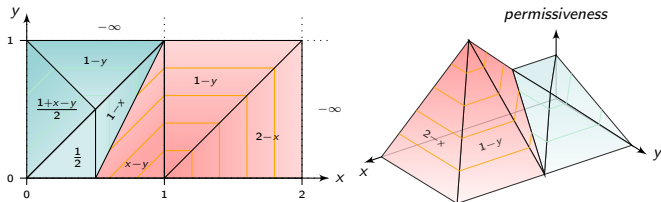
Case of acyclic timed automata



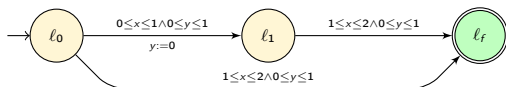
Case of acyclic timed automata



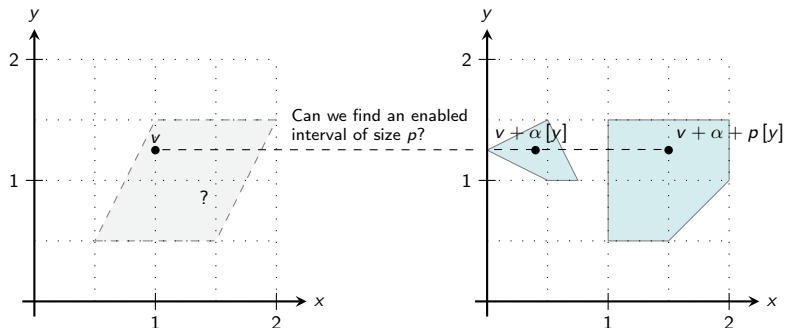
- Its permissiveness function on l_0



Case of acyclic timed automata



- Issues with binary permissiveness

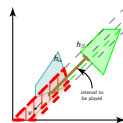
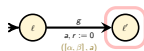


Contribution during this thesis

Conclusion & Future work

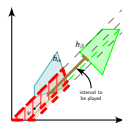
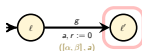
- Symbolic algorithm

- ▷ A non-elementary algorithm
- ▷ Restricted to acyclic timed automata
- ▷ Exact result



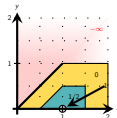
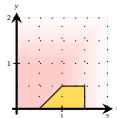
- Symbolic algorithm

- ▷ A non-elementary algorithm
- ▷ Restricted to acyclic timed automata
- ▷ Exact result



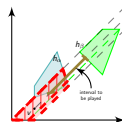
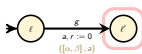
- Binary and levelled permissiveness

- ▷ A doubly-exponential algorithm
- ▷ Linear w.r.t the number of levels
- ▷ Restricted to linear timed automata
- ▷ Controlled approximation of the permissiveness



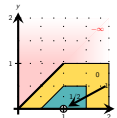
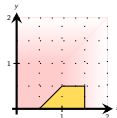
- Symbolic algorithm

- ▷ A non-elementary algorithm
- ▷ Restricted to acyclic timed automata
- ▷ Exact result



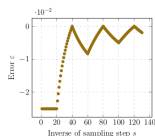
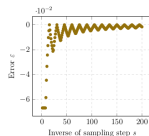
- Binary and levelled permissiveness

- ▷ A doubly-exponential algorithm
- ▷ Linear w.r.t the number of levels
- ▷ Restricted to linear timed automata
- ▷ Controlled approximation of the permissiveness



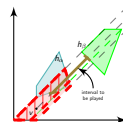
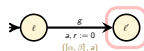
- Numeric algorithm

- ▷ A doubly-exponential algorithm
- ▷ Restricted to acyclic timed automata
- ▷ Stability not proven



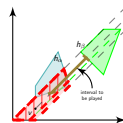
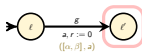
- Symbolic algorithm

- ▶ Cyclic TA
- ▶ Implementation of acyclic TA
- ▶ Minimise the overtiling



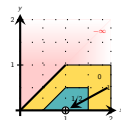
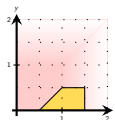
- Symbolic algorithm

- ▷ Cyclic TA
- ▷ Implementation of acyclic TA
- ▷ Minimise the overtiling



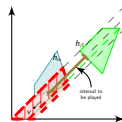
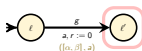
- Binary and levelled permissiveness

- ▷ Acyclic (and cyclic) TA
- ▷ Implementation with *pplpy*



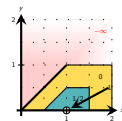
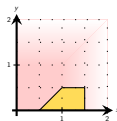
- Symbolic algorithm

- ▷ Cyclic TA
- ▷ Implementation of acyclic TA
- ▷ Minimise the overtiling



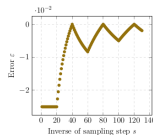
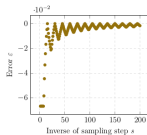
- Binary and levelled permissiveness

- ▷ Acyclic (and cyclic) TA
- ▷ Implementation with *pplpy*



- Numeric algorithm

- ▷ Cyclic TA
- ▷ Prove the stability
- ▷ Implementation of polyhedral guards



Formal method and robotics

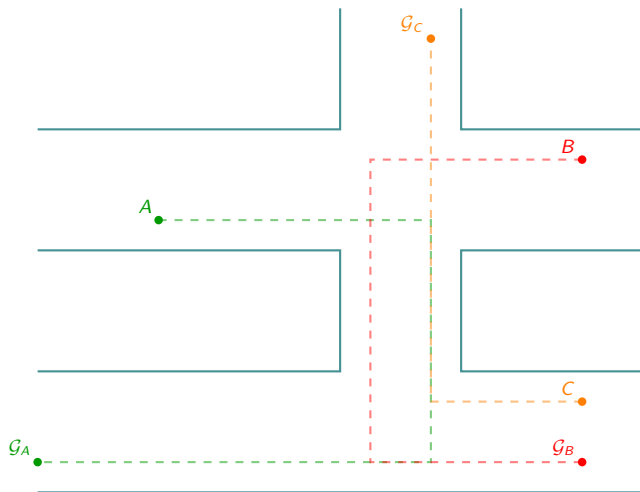
Model controlled cars with fixed trajectories

Joint work with Nicolas Perrin-Gilbert² & Philipp Schlehuber³

²ISIR, Sorbonne université, Paris, France

³LRDE, EPITA, Paris, France

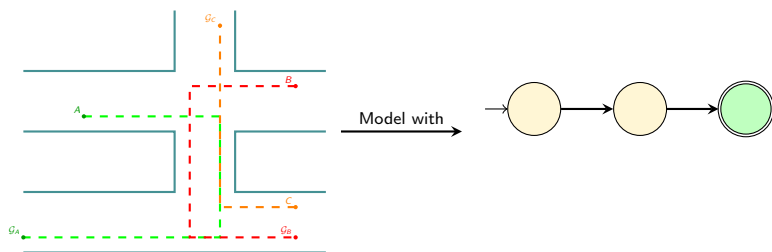
Example of our problem



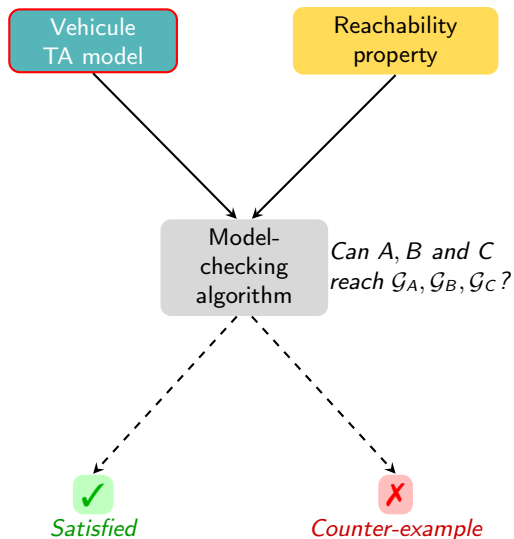
- Goal
 - ▷ Avoid collision
 - ▷ Reach all goals

- Goal
 - ▷ Avoid collision
 - ▷ Reach all goals
- Our model
 - ▷ We control each car, with (Stop, Go)
 - ▷ All cars have the same speed
 - ▷ Represent the non-collision conditions in the timed automaton

- Goal
 - ▷ Avoid collision
 - ▷ Reach all goals
- Our model
 - ▷ We control each cars, with (Stop, Go)
 - ▷ All cars have the same speed
 - ▷ Represent the non-collision conditions in the timed automaton
- Formal methods:



Conclusion: Goal of the post-doc



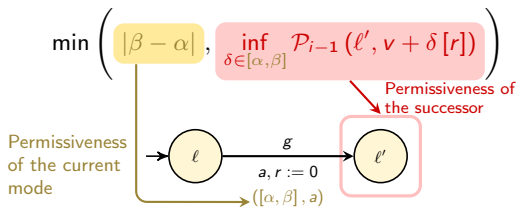
Appendix

Appendix

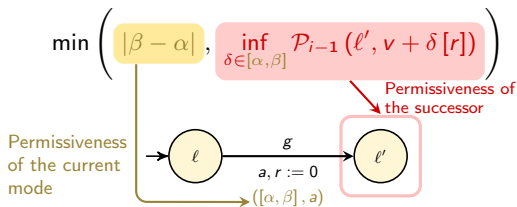
*Other contribution: an exact symbolic computation*⁴

⁴CJMM20.

- Strategy of the player: maximises



- Strategy of the player: maximises



- Opponent strategy lemma (linear case):

player : $([\alpha, \beta], a) \xrightarrow{\text{Opponent's best strategy}} \alpha \text{ or } \beta$

$$\inf_{\delta \in [\alpha, \beta]} \mathcal{P}_{i-1}(l', v + \delta[r]) = \min \left(\mathcal{P}_{i-1}(l', v + \alpha[r]), \mathcal{P}_{i-1}(l', v + \beta[r]) \right)$$

- A recursive function

$$\mathcal{P}_i(\ell, \nu) = \sup_{([\alpha, \beta], a) \in \text{p-moves}(\ell, \nu)} \left(\min \left(\beta - \alpha, \inf_{\delta \in [\alpha, \beta]} \mathcal{P}_{i-1}(\ell', \nu + \delta[r]) \right) \right)$$

- A recursive function

$$\mathcal{P}_i(\ell, \nu) = \sup_{([\alpha, \beta], a) \in \mathbf{p}\text{-moves}(\ell, \nu)} \left(\min \left(\beta - \alpha, \inf_{\delta \in [\alpha, \beta]} \mathcal{P}_{i-1}(\ell', \nu + \delta[r]) \right) \right)$$

For linear timed automata:

$$\mathcal{P}_i(\ell, \nu) = \sup_{([\alpha, \beta], a) \in \mathbf{p}\text{-moves}(\ell, \nu)} \left(\min \left(\beta - \alpha, \mathcal{P}_{i-1}(\ell', \nu + \alpha[r]), \mathcal{P}_{i-1}(\ell', \nu + \beta[r]) \right) \right)$$

$$\lim_{i \rightarrow +\infty} \mathcal{P}_i(\ell, \nu) = \text{the permissiveness on } (\ell, \nu)$$

- A recursive function

$$\mathcal{P}_i(\ell, \nu) = \sup_{([\alpha, \beta], a) \in \text{p-moves}(\ell, \nu)} \left(\min \left(\beta - \alpha, \inf_{\delta \in [\alpha, \beta]} \mathcal{P}_{i-1}(\ell', \nu + \delta[r]) \right) \right)$$

For linear timed automata:

$$\mathcal{P}_i(\ell, \nu) = \sup_{([\alpha, \beta], a) \in \text{p-moves}(\ell, \nu)} \left(\min \left(\beta - \alpha, \mathcal{P}_{i-1}(\ell', \nu + \alpha[r]), \mathcal{P}_{i-1}(\ell', \nu + \beta[r]) \right) \right)$$

$\lim_{i \rightarrow +\infty} \mathcal{P}_i(\ell, \nu) =$ the permissiveness on (ℓ, ν)

↑
limit reached in d_ℓ steps

The permissiveness function

- A recursive function

$$\mathcal{P}_i(\ell, v) = \sup_{([\alpha, \beta], a) \in \text{p-moves}(\ell, v)} \left(\min \left(\beta - \alpha, \inf_{\delta \in [\alpha, \beta]} \mathcal{P}_{i-1}(\ell', v + \delta[r]) \right) \right)$$

For linear timed automata:

$$\mathcal{P}_i(\ell, v) = \sup_{([\alpha, \beta], a) \in \text{p-moves}(\ell, v)} \left(\min \left(\beta - \alpha, \mathcal{P}_{i-1}(\ell', v + \alpha[r]), \mathcal{P}_{i-1}(\ell', v + \beta[r]) \right) \right)$$

$\lim_{i \rightarrow +\infty} \mathcal{P}_i(\ell, v) =$ the permissiveness on (ℓ, v)

limit reached in d_ℓ steps

- Goal of our algorithm

Compute $v \mapsto \mathcal{P}_{d_\ell}(\ell, v)$ knowing $v \mapsto \mathcal{P}_{d_\ell-1}(\ell', v)$

- Goal: find the α and β that maximises:

$$\min \left(|\beta - \alpha|, \mathcal{P}_{i-1}(\ell', v + \alpha[r]), \mathcal{P}_{i-1}(\ell', v + \beta[r]) \right)$$

- Goal: find the α and β that maximises:

$$\min \left(|\beta - \alpha|, \mathcal{P}_{i-1}(\ell', v + \alpha[r]), \mathcal{P}_{i-1}(\ell', v + \beta[r]) \right)$$

- Goal: find the α and β that maximises:

$$\min \left(|\beta - \alpha|, \mathcal{P}_{i-1}(\ell', v + \alpha[r]), \mathcal{P}_{i-1}(\ell', v + \beta[r]) \right)$$

- $v \mapsto \mathcal{P}_i(\ell, v)$: a 2-Lipschitz piecewise-affine function

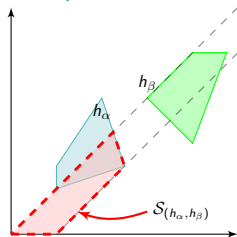
Steps of the algorithm (linear timed automata)

- Goal: find the α and β that maximises:

$$\min \left(|\beta - \alpha|, \mathcal{P}_{i-1}(\ell', v + \alpha[r]), \mathcal{P}_{i-1}(\ell', v + \beta[r]) \right)$$

- $v \mapsto \mathcal{P}_i(\ell, v)$: a 2-Lipschitz piecewise-affine function

- Steps: for each couple of cells (h_α, h_β)



(a) Step 1: compute $S(h_\alpha, h_\beta)$

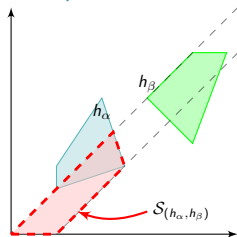
Steps of the algorithm (linear timed automata)

- Goal: find the α and β that maximises:

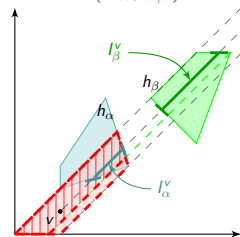
$$\min \left(|\beta - \alpha|, \mathcal{P}_{i-1}(\ell', v + \alpha[r]), \mathcal{P}_{i-1}(\ell', v + \beta[r]) \right)$$

- $v \mapsto \mathcal{P}_i(\ell, v)$: a 2-Lipschitz piecewise-affine function

- Steps: for each couple of cells (h_α, h_β)



(a) Step 1: compute $S(h_\alpha, h_\beta)$



(b) Step 2: compute the possible α and β

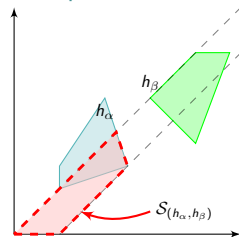
Steps of the algorithm (linear timed automata)

- Goal: find the α and β that maximises:

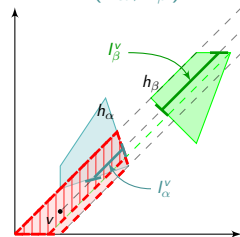
$$\min \left(|\beta - \alpha|, \mathcal{P}_{i-1}(\ell', v + \alpha[r]), \mathcal{P}_{i-1}(\ell', v + \beta[r]) \right)$$

- $v \mapsto \mathcal{P}_i(\ell, v)$: a 2-Lipschitz piecewise-affine function

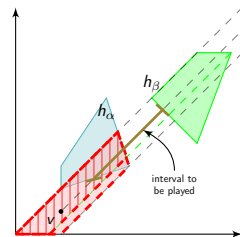
- Steps: for each couple of cells (h_α, h_β)



(a) Step 1: compute $S(h_\alpha, h_\beta)$

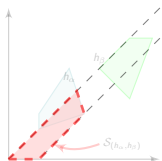


(b) Step 2: compute the possible α and β

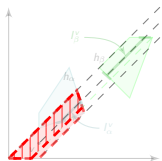


(c) Step 3: compute the optimal α and β

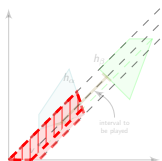
Example



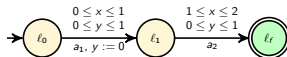
(a) Step 1

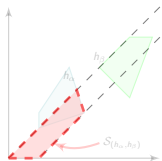


(b) Step 2

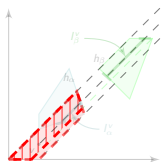


(c) Step 3

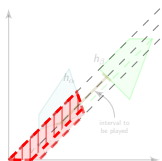




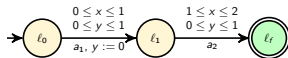
(a) Step 1



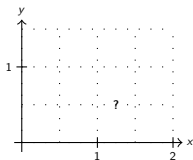
(b) Step 2



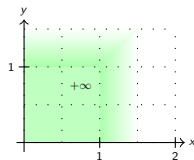
(c) Step 3



- Permissiveness on l_1 : maximise $\min(\beta - \alpha, \mathcal{P}_0(l_f, v + \alpha), \mathcal{P}_0(l_f, v + \beta))$

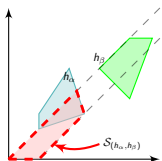


(a) Permissiveness on l_1

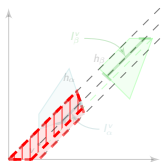


(b) Permissiveness on l_f

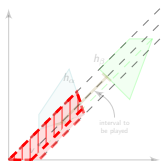
Fixing the cells of arrival of the successors h_α and h_β : \mathbb{R}^2



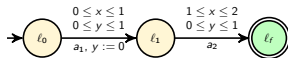
(a) Step 1



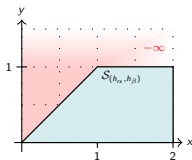
(b) Step 2



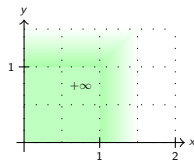
(c) Step 3



- Permissiveness on ℓ_1 : maximise $\min(\beta - \alpha, \mathcal{P}_0(\ell_f, v + \alpha), \mathcal{P}_0(\ell_f, v + \beta))$

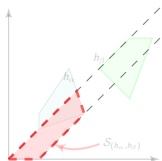


(a) Permissiveness on ℓ_1

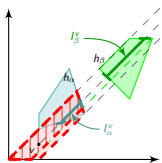


(b) Permissiveness on ℓ_f

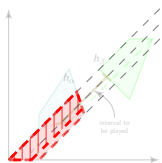
Step 1: computing $\mathcal{S}(h_\alpha, h_\beta)$ (Fourier-Motzkin algorithm)



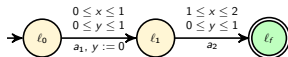
(a) Step 1



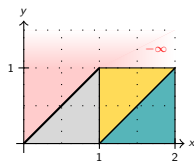
(b) Step 2



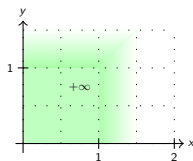
(c) Step 3



- Permissiveness on l_1 : maximise $\min(\beta - \alpha, \mathcal{P}_0(l_f, v + \alpha), \mathcal{P}_0(l_f, v + \beta))$



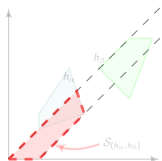
(a) Permissiveness on l_1



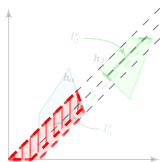
(b) Permissiveness on l_f

Step 2: computing the intervals of α and β (Fourier-Motzkin algorithm)

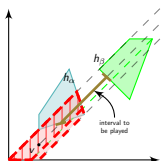
$$I_\alpha^V = I_\beta^V = [\max(0, 1 - x), \min(2 - x, 1 - y)]$$



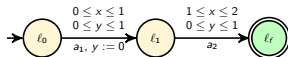
(a) Step 1



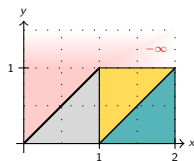
(b) Step 2



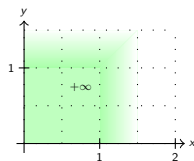
41 / 53 (c) Step 3



- Permissiveness on l_1 : maximise $\min(\beta - \alpha, \mathcal{P}_0(l_f, v + \alpha), \mathcal{P}_0(l_f, v + \beta))$

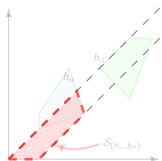


(a) Permissiveness on l_1

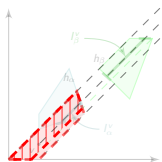


(b) Permissiveness on l_f

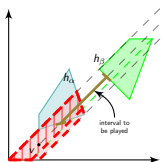
Step 3: computing the optimal α and β , s.t $\alpha \leq \beta$, that maximises...



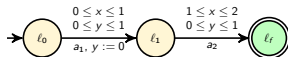
(a) Step 1



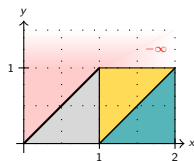
(b) Step 2



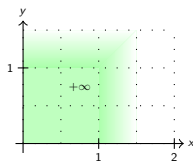
41 / 53 (c) Step 3



- Permissiveness on l_1 : maximise $\min(\beta - \alpha, \mathcal{P}_0(l_f, v + \alpha), \mathcal{P}_0(l_f, v + \beta))$



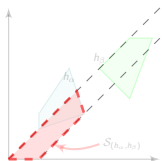
(a) Permissiveness on l_1



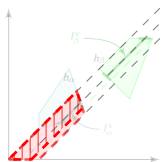
(b) Permissiveness on l_f

Step 3: computing the optimal α and β , s.t $\alpha \leq \beta$, that maximises...

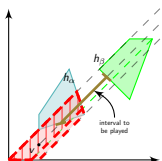
$$\min(\beta - \alpha, \mathcal{P}_{i-1}(l', v + \alpha[r]), \mathcal{P}_{i-1}(l', v + \beta[r]))$$



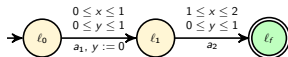
(a) Step 1



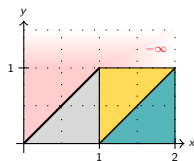
(b) Step 2



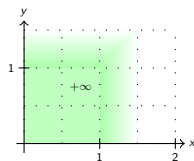
41 / 53 (c) Step 3



- Permissiveness on ℓ_1 : maximise $\min(\beta - \alpha, \mathcal{P}_0(\ell_f, v + \alpha), \mathcal{P}_0(\ell_f, v + \beta))$



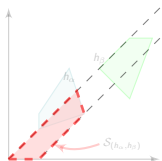
(a) Permissiveness on ℓ_1



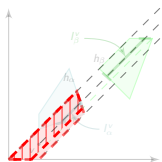
(b) Permissiveness on ℓ_f

Step 3: computing the optimal α and β , s.t $\alpha \leq \beta$, that maximises...

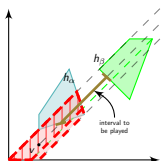
$$\min(\beta - \alpha, +\infty, +\infty)$$



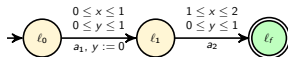
(a) Step 1



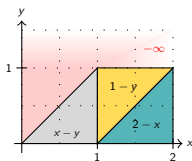
(b) Step 2



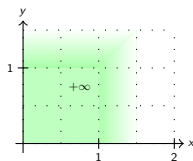
41 / 53 (c) Step 3



- Permissiveness on ℓ_1 : maximise $\min(\beta - \alpha, \mathcal{P}_0(\ell_f, v + \alpha), \mathcal{P}_0(\ell_f, v + \beta))$



(a) Permissiveness on ℓ_1

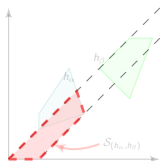


(b) Permissiveness on ℓ_f

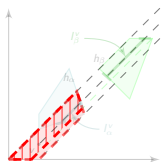
Step 3: computing the optimal α and β , s.t $\alpha \leq \beta$, that maximises...

$$\min(\beta - \alpha, +\infty, +\infty) \quad \text{(Technical lemma)}$$

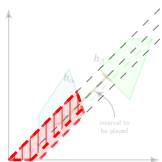
$$\alpha^* = \max(0, 1 - x), \beta^* = \min(2 - x, 1 - y)$$



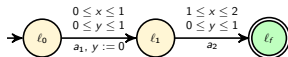
(a) Step 1



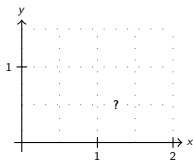
(b) Step 2



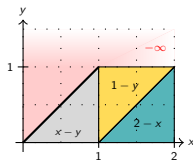
(c) Step 3




- Permissiveness on l_0 : maximise $\min(\beta - \alpha, \mathcal{P}_1(l_1, v + \alpha), \mathcal{P}_1(l_1, v + \beta))$

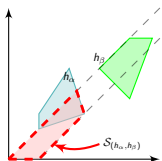


(a) Permissiveness on l_0

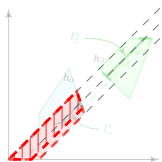


(b) Permissiveness on l_1

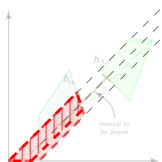
Let us fix $h_\alpha = h_\beta =$ 



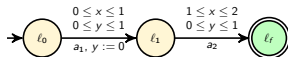
(a) Step 1



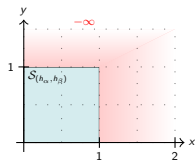
(b) Step 2



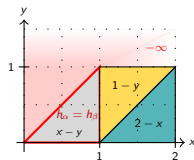
(c) Step 3



- Permissiveness on l_0 : maximise $\min(\beta - \alpha, \mathcal{P}_1(l_1, v + \alpha), \mathcal{P}_1(l_1, v + \beta))$



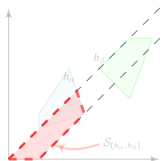
(a) Permissiveness on l_0



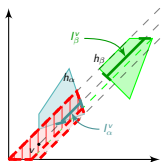
(b) Permissiveness on l_1

Step 1: Computing the corresponding entry set $\mathcal{S}(h_\alpha, h_\beta)$

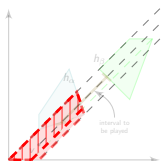
(Fourier-Motzkin algorithm)



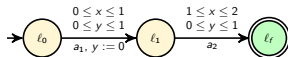
(a) Step 1



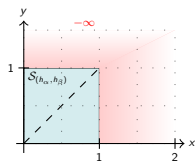
(b) Step 2



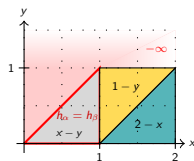
(c) Step 3



- Permissiveness on ℓ_0 : maximise $\min(\beta - \alpha, \mathcal{P}_1(\ell_1, v + \alpha), \mathcal{P}_1(\ell_1, v + \beta))$



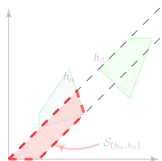
(a) Permissiveness on ℓ_0



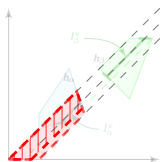
(b) Permissiveness on ℓ_1

Step 2: Computing the set of possible α and β (Fourier-Motzkin algorithm):

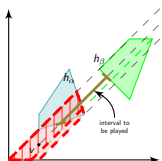
$$I_\alpha^V = I_\beta^V = [0, \min(1 - x, 1 - y)]$$



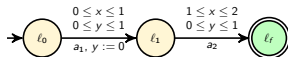
(a) Step 1



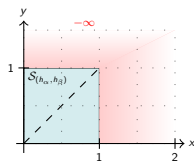
(b) Step 2



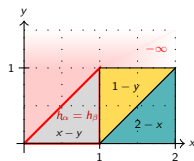
41 / 53 (c) Step 3



- Permissiveness on l_0 : maximise $\min(\beta - \alpha, \mathcal{P}_1(l_1, v + \alpha), \mathcal{P}_1(l_1, v + \beta))$



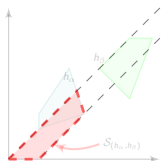
(a) Permissiveness on l_0



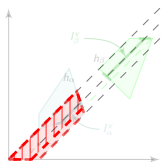
(b) Permissiveness on l_1

- Step 3: For $y \leq x$: Computing the optimal α and β in $[0, 1 - x]^2$, s.t. $\alpha \leq \beta$, that maximise:

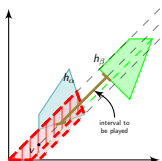
$$\min(\beta - \alpha, 1 \cdot \alpha + x, 1 \cdot \beta + x) \quad \text{(Technical lemma)}$$



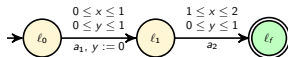
(a) Step 1



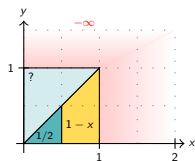
(b) Step 2



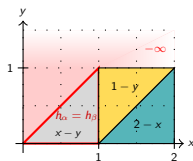
41 / 53 (c) Step 3



- Permissiveness on l_0 : maximise $\min(\beta - \alpha, \mathcal{P}_1(l_1, v + \alpha), \mathcal{P}_1(l_1, v + \beta))$



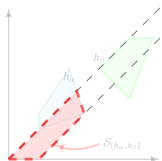
(a) Permissiveness on l_0



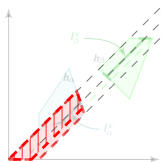
(b) Permissiveness on l_1

Step 3: For $y \leq x$: Computing the optimal α and β in $[0, 1 - x]^2$, s.t. $\alpha \leq \beta$, that maximise:

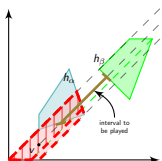
$$\min(\beta - \alpha, 1 \cdot \alpha + x, 1 \cdot \beta + x) \quad \text{(Technical lemma)}$$



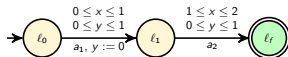
(a) Step 1



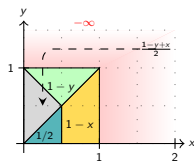
(b) Step 2



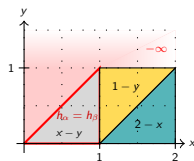
41 / 53 (c) Step 3



- Permissiveness on l_0 : maximise $\min(\beta - \alpha, \mathcal{P}_1(l_1, v + \alpha), \mathcal{P}_1(l_1, v + \beta))$



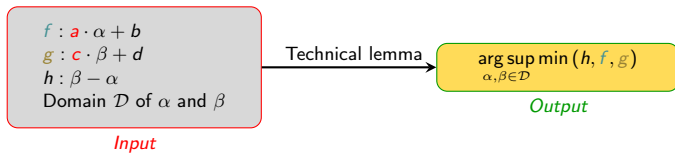
(a) Permissiveness on l_0

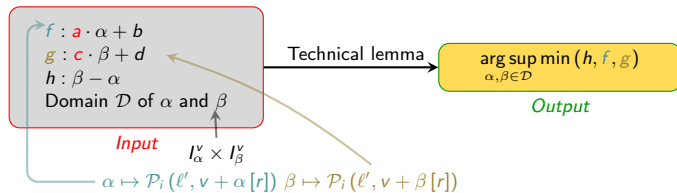


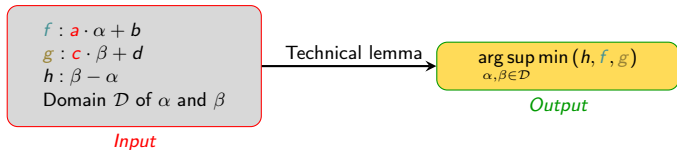
(b) Permissiveness on l_1

- Step 3:** Same for $x \leq y$: Computing the optimal α and β in $[0, 1 - y]^2$, s.t $\alpha \leq \beta$, that maximise:

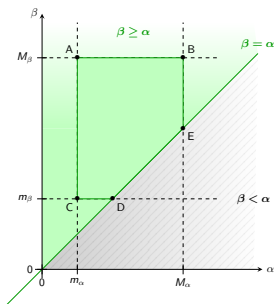
$$\min(\beta - \alpha, 1 \cdot \alpha + x, 1 \cdot \beta + x) \quad \text{(Technical lemma)}$$



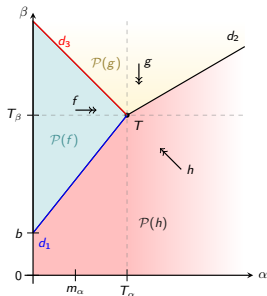




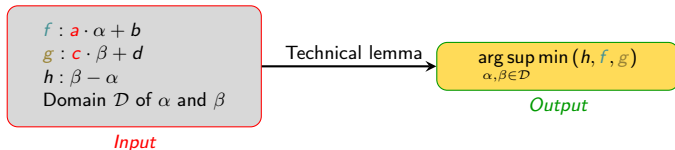
- When $a > 0$ and $c < 0$:



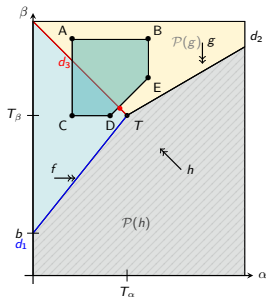
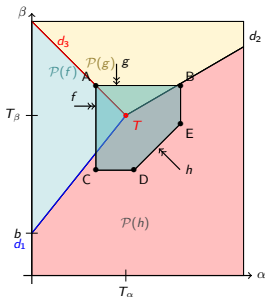
(a) Domain \mathcal{D}



(b) $\min (f, g, h)$ on \mathbb{R}_+^2



- When $a > 0$ and $c < 0$:



- The algorithm
 - ▷ Limited to acyclic timed automata

- The algorithm

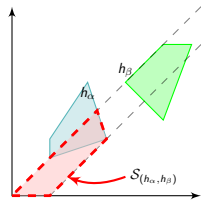
- ▷ Limited to acyclic timed automata
- ▷ Upper bound time complexity: **non-elementary**
- ▷ Complexity: grows with the number of cells, of clocks and d_ℓ

- The algorithm

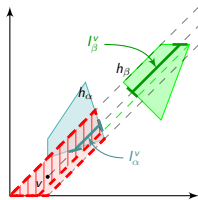
- ▷ Limited to acyclic timed automata
- ▷ Upper bound time complexity: **non-elementary**
- ▷ Complexity: grows with the number of cells, of clocks and d_ℓ

- Causes of the high complexity

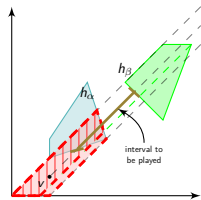
- ▷ Overtiling
- ▷ Exploration of all couple of cells (h_α, h_β)



(a) Step 1: compute $S(h_\alpha, h_\beta)$



(b) Step 2: compute the possible α and β



(c) Step 3: compute the optimal α and β

- ▶ Based on *pplpy*

- ▶ Based on *pplpy*
- ▶ Covers the case of **linear timed automata** with **polyhedral** guards

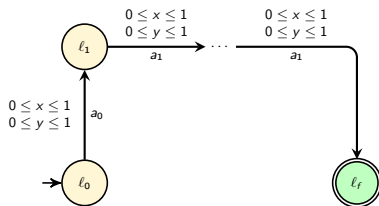
- ▷ Based on *ppipy*
- ▷ Covers the case of **linear timed automata** with **polyhedral guards**
- ▷ Runtime results on our examples:

Clocks	Nb. of transitions	Runtime for ℓ_0	Runtime for ℓ_1	Runtime for ℓ_2	Runtime for ℓ_3
2	2	0.82 (2 cells)	0.059 (2 cells)	-	-
2	2	0.071 (6 cells)	0.062 (3 cells)	-	-
2	2	0.73 (24 cells)	0.034 (3 cells)	-	-
2	3	38.16 (582 cells)	1.01 (25 cells)	0.09 (3 cells)	-
3	4	19.95 (234 cells)	0.41 (12 cells)	0.56 (6 cells)	0.14 (6 cells)
3	4	143.48 (1825 cells)	0.39 (12 cells)	0.59 (6 cells)	0.14 (6 cells)

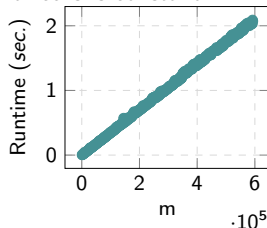
- ▶ Based on *pplpy*
- ▶ Covers the case of **linear timed automata with polyhedral guards**
- ▶ Runtime results on our examples:

Clocks	Nb. of transitions	Runtime for ℓ_0	Runtime for ℓ_1	Runtime for ℓ_2	Runtime for ℓ_3
2	2	0.82 (2 cells)	0.059 (2 cells)	-	-
2	2	0.071 (6 cells)	0.062 (3 cells)	-	-
2	2	0.73 (24 cells)	0.034 (3 cells)	-	-
2	3	38.16 (582 cells)	1.01 (25 cells)	0.09 (3 cells)	-
3	4	19.95 (234 cells)	0.41 (12 cells)	0.56 (6 cells)	0.14 (6 cells)
3	4	143.48 (1825 cells)	0.39 (12 cells)	0.59 (6 cells)	0.14 (6 cells)

- ▶ Runtime results on a case where the number of cells is constant:

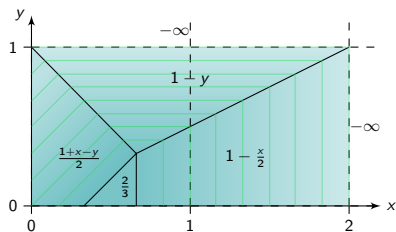
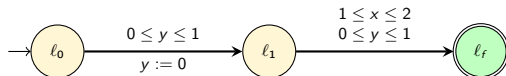


(a) A m transitions timed automaton

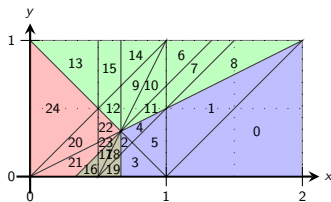


(b) Runtimes depending on the number of transitions (m)

- Example of overtiling



(a) Permissiveness on l_0



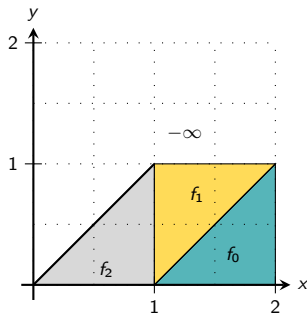
(b) Permissiveness computed by our tool

- ▶ Redundancy in the technical lemma

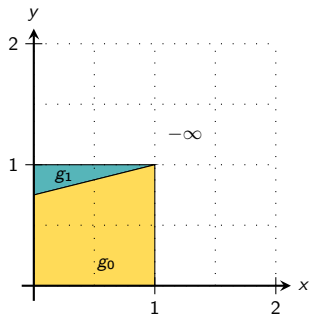
- ▷ Redundancy in the technical lemma
- ▷ Maximisation/Minimisation (when comparing candidate permissiveness functions)

The causes of the overtiling

- ▶ Redundancy in the technical lemma
- ▶ Maximisation/Minimisation (when comparing candidate permissiveness functions)
- Example of maximisation: computing $\max(f, g)$



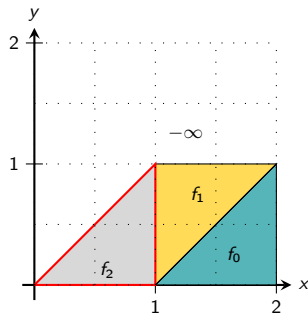
(a) $f: f_0(x, y) = 2 - x$, $f_1(x, y) = 1 - y$,
 $f_2(x, y) = x - y$



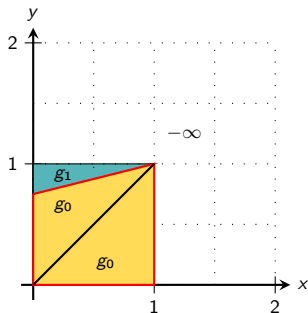
(b) $g: g_0(x, y) = (1 - x)/2$, $g_1(x, y) = 1 - y$

The causes of the overtiling

- ▷ Redundancy in the technical lemma
- ▷ Maximisation/Minimisation (when comparing candidate permissiveness functions)
- Example of maximisation: computing $\max(f, g)$

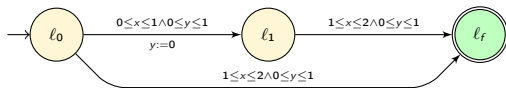


(a) f

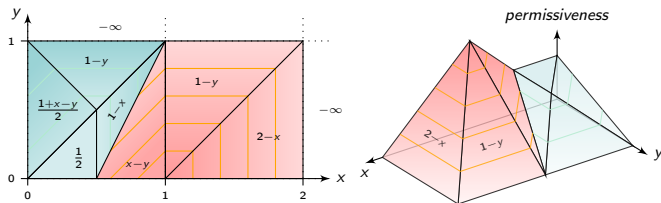


(b) Overtiled representation of g

- Example of an acyclic timed automata



- (1st contribution) Its permissiveness on l_0



- Case of two polyhedra

Solved in 2001 by Bemporad, Fukuda and D. Torrisi⁵:

- ▶ Computing convex hull by removing inequalities
- ▶ Solving a linear program

⁵BemporadFT01.

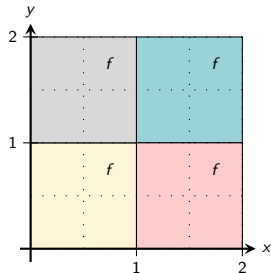
- Case of two polyhedra

Solved in 2001 by Bemporad, Fukuda and D. Torrisi⁵:

- ▶ Computing convex hull by removing inequalities
- ▶ Solving a linear program

- Case of several (≥ 3) polyhedra

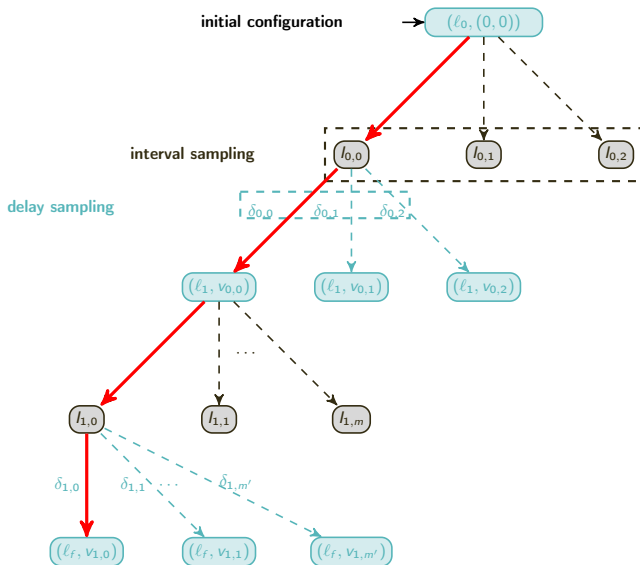
Open problem...

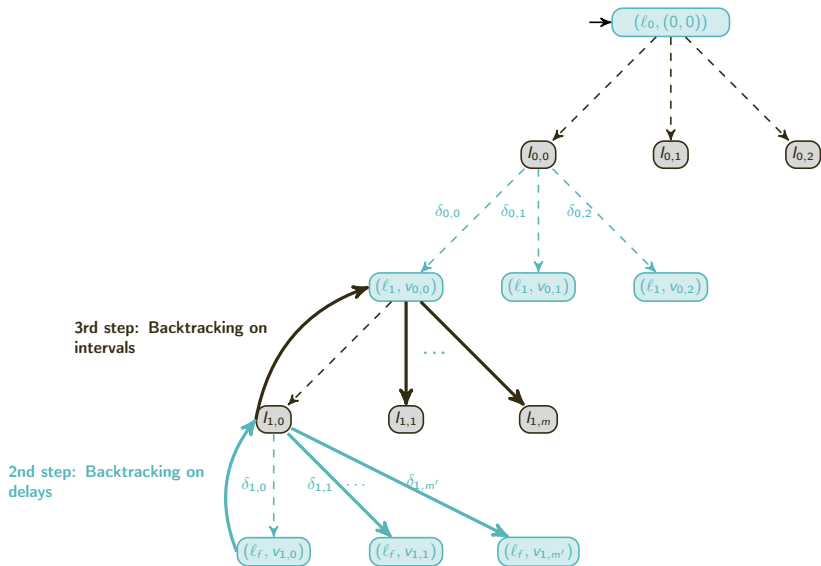


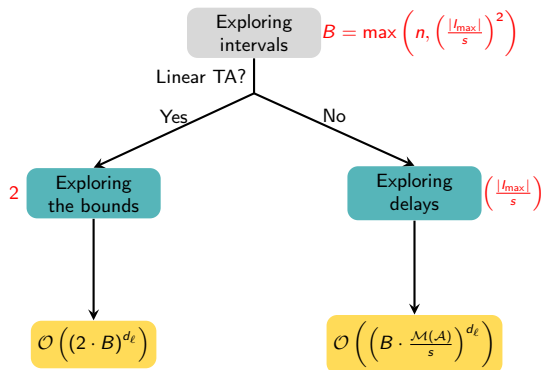
⁵BemporadFT01.

Appendix

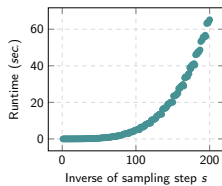
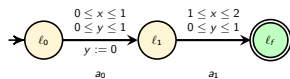
Other contribution: an numerical approximative computation



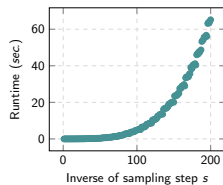
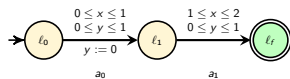




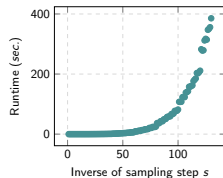
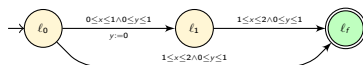
- A linear example



- A linear example



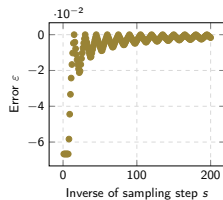
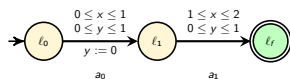
- An acyclic example



Error $\varepsilon = \text{Computed permissiveness} - \text{correct permissiveness}$

Error $\varepsilon =$ Computed permissiveness – correct permissiveness

- A linear example



- An acyclic example

