

Robustness of timed automata:
computing the maximally-permissive strategies

Emily Clement^{1,2}

Supervisors:

Thierry Jéron¹

Nicolas Markey¹

David Menré²

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

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

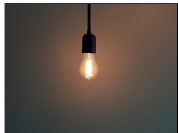
March 11 2022



Inria



Why verifying real-time systems?



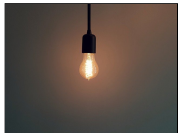
Real-time systems



Why verifying real-time systems?

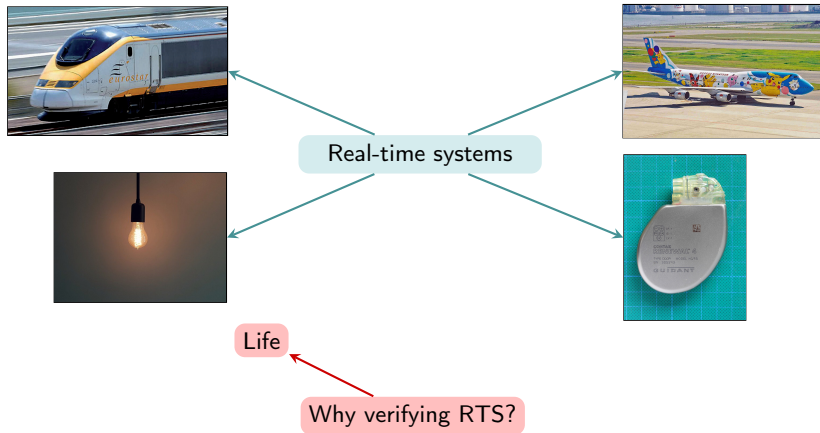


Real-time systems

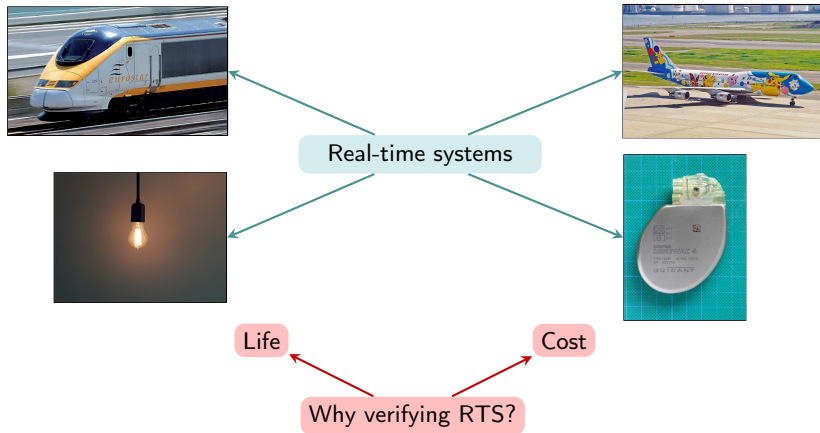


Why verifying RTS?

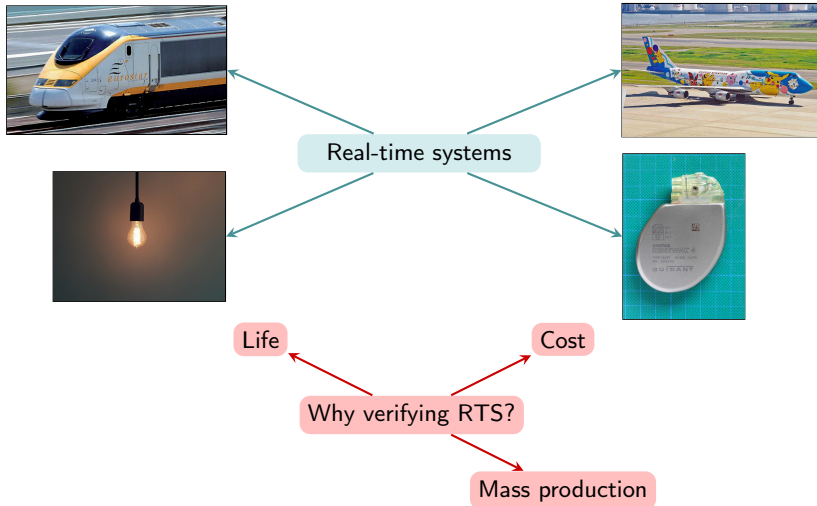
Why verifying real-time systems?



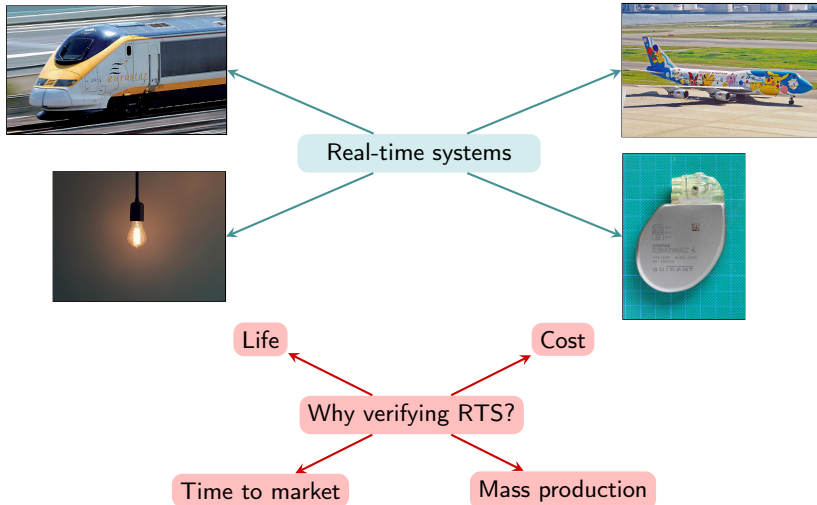
Why verifying real-time systems?

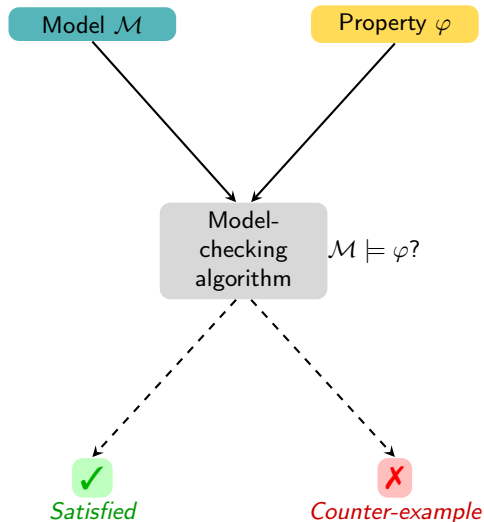


Why verifying real-time systems?



Why verifying real-time systems?





An abstract
model

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

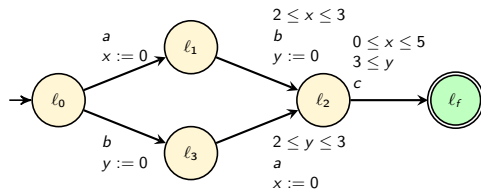
An abstract
model

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

An abstract model

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

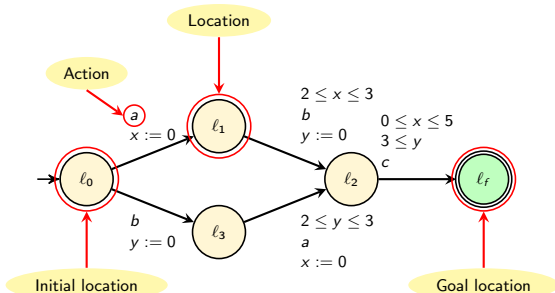
- Example: Scheduling system



An abstract model

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

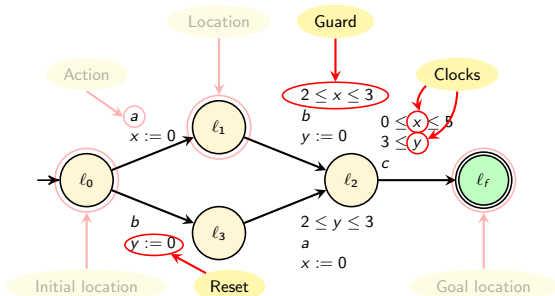
- Example: Scheduling system



An abstract model

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

- Example: Scheduling system



Acyclic timed automata:



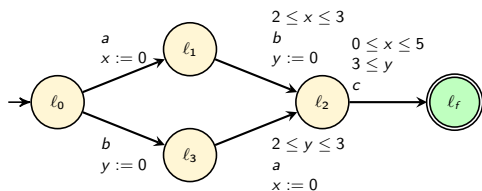
Linear timed automata:



An abstract model

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

- Example: Scheduling system



Acyclic timed automata:



Linear timed automata:



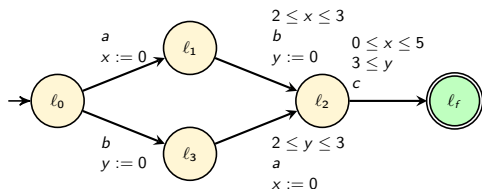
▷ Can we execute...

▷

An abstract model

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

- Example: Scheduling system



Acyclic timed automata:



Linear timed automata:



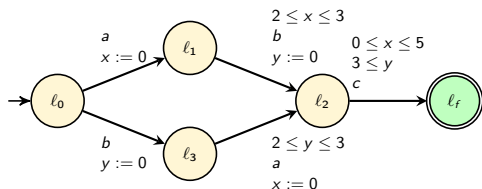
▶ Can we execute...*abc*?

▶ $l_0, (0, 0) \xrightarrow{\delta_0} l_0, (\delta_0, \delta_0) \xrightarrow{a, x:=0} l_1, (0, \delta_0) \xrightarrow{\delta_1} l_1, (\delta_1, \delta_0 + \delta_1) \xrightarrow{b, y:=0} l_2, (\delta_1, 0) \xrightarrow{\delta_2} l_2, (\delta_1 + \delta_2, \delta_2) \xrightarrow{c} l_f, (\delta_1 + \delta_2, \delta_2)$

An abstract model

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

- Example: Scheduling system



Acyclic timed automata:



Linear timed automata:



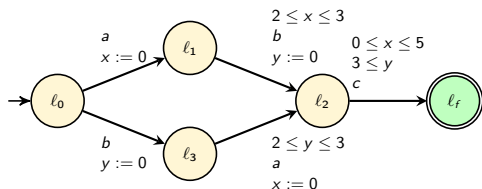
▷ Can we execute... abc ? ✓

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

An abstract model

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

- Example: Scheduling system



Acyclic timed automata:



Linear timed automata:



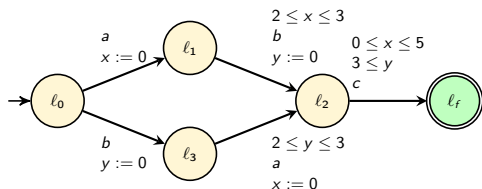
▷ Can we execute...*bac*?

▷ $l_0, (0, 0) \xrightarrow{\delta_0} l_0, (\delta_0, \delta_0) \xrightarrow{b, y:=0} l_3, (\delta_0, 0) \xrightarrow{\delta_1} l_3, (\delta_1 + \delta_0, \delta_1) \xrightarrow{a, x:=0} l_2, (0, \delta_1) \xrightarrow{\delta_2} l_2, (\delta_2, \delta_1 + \delta_2) \xrightarrow{c} l_f, (\delta_2, \delta_1 + \delta_2)$

An abstract model

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

- Example: Scheduling system



Acyclic timed automata:



Linear timed automata:



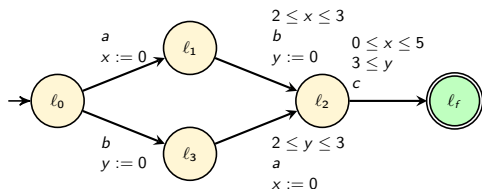
▷ Can we execute...*bac*? ✓

▷ $l_0, (0, 0) \xrightarrow{\delta_0=10} l_0, (10, 10) \xrightarrow{b, y:=0} l_3, (10, 0) \xrightarrow{\delta_1=3} l_3, (13, 3) \xrightarrow{a, x:=0} l_2, (0, 3) \xrightarrow{\delta_2=4} l_2, (4, 7) \xrightarrow{c} l_f, (4, 7)$

An abstract model

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

- Example: Scheduling system



Acyclic timed automata:



Linear timed automata:



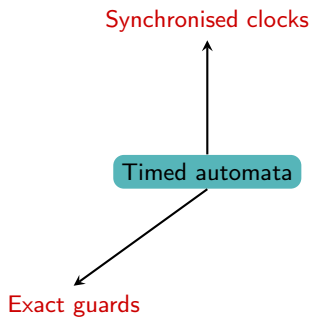
▷ Can we execute...*bac*? ✓

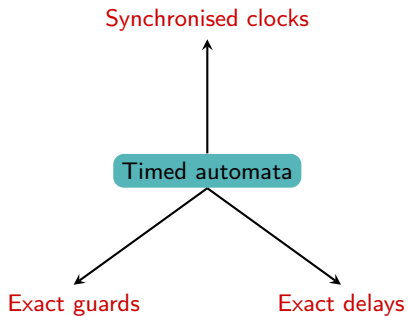
▷ $l_0, (0, 0) \xrightarrow{\delta_0=10} l_0, (10, 10) \xrightarrow{b, y:=0} l_3, (10, 0) \xrightarrow{\delta_1=2} l_3, (12, 2) \xrightarrow{a, x:=0} l_2, (0, 2) \xrightarrow{\delta_2=2} l_2, (2, 2) \xrightarrow{c} l_f, (2, 2)$

Synchronised clocks



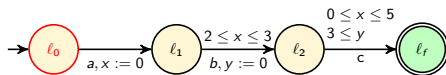
Timed automata



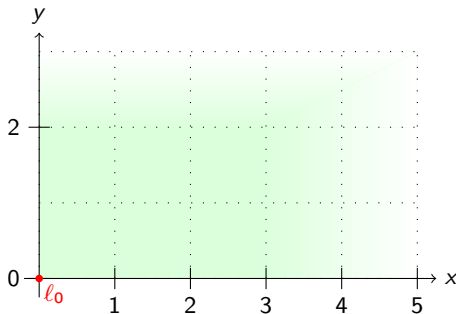


Example of perturbed semantics: clock drifting¹

- Timed automaton \mathcal{A}



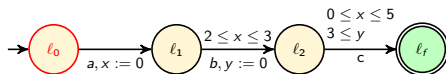
- Run with desynchronised clocks: $0.9 \cdot \dot{y} \leq \dot{x} \leq 1.1 \cdot \dot{y}$



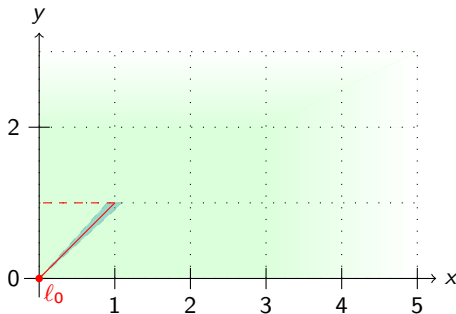
¹Puri, 'Dynamical Properties of Timed Automata', 2000.

Example of perturbed semantics: clock drifting¹

- Timed automaton \mathcal{A}



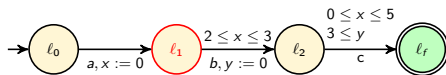
- Run with desynchronised clocks: $0.9 \cdot \dot{y} \leq \dot{x} \leq 1.1 \cdot \dot{y}$



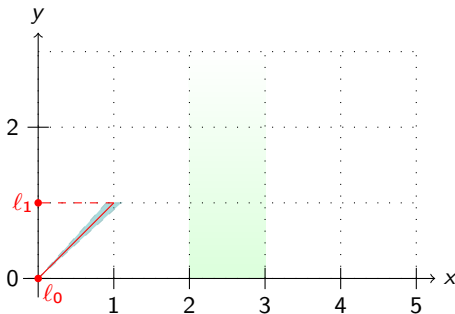
¹Puri, 'Dynamical Properties of Timed Automata', 2000.

Example of perturbed semantics: clock drifting¹

- Timed automaton \mathcal{A}



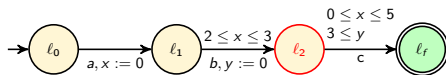
- Run with desynchronised clocks: $0.9 \cdot \dot{y} \leq \dot{x} \leq 1.1 \cdot \dot{y}$



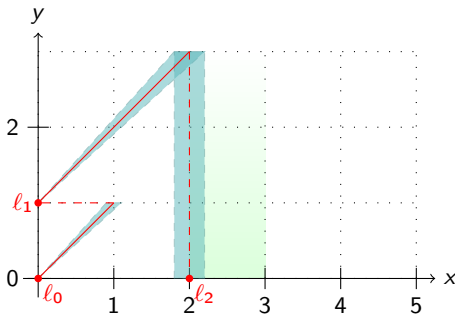
¹Puri, 'Dynamical Properties of Timed Automata', 2000.

Example of perturbed semantics: clock drifting¹

- Timed automaton \mathcal{A}



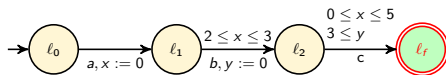
- Run with desynchronised clocks: $0.9 \cdot \dot{y} \leq \dot{x} \leq 1.1 \cdot \dot{y}$



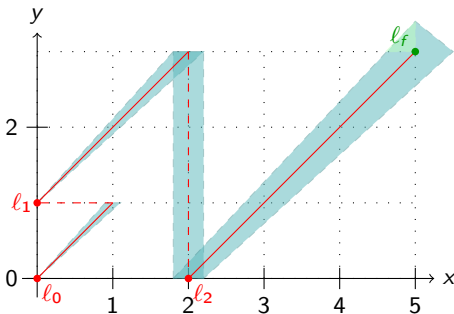
¹Puri, 'Dynamical Properties of Timed Automata', 2000.

Example of perturbed semantics: clock drifting¹

- Timed automaton \mathcal{A}

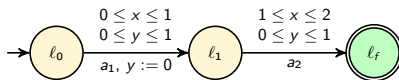


- Run with desynchronised clocks: $0.9 \cdot \dot{y} \leq \dot{x} \leq 1.1 \cdot \dot{y}$



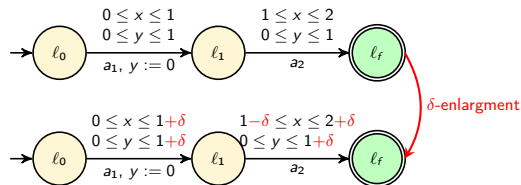
¹Puri, 'Dynamical Properties of Timed Automata', 2000.

- Timed automaton \mathcal{A} :



²Puri, 'Dynamical Properties of Timed Automata', 2000.

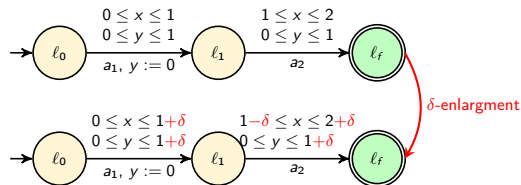
- Timed automaton \mathcal{A} :



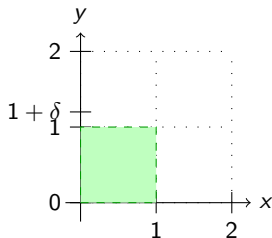
²Puri, 'Dynamical Properties of Timed Automata', 2000.

Example of perturbed semantics: guard enlargement²

- Timed automaton \mathcal{A} :



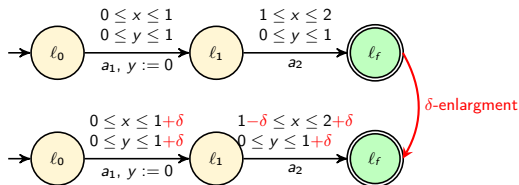
- Representation of an enlarged guard $0 \leq x \leq 1 \wedge 0 \leq y \leq 1$



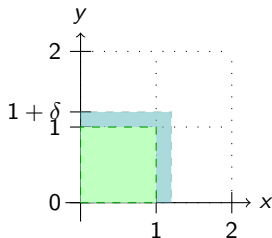
²Puri, 'Dynamical Properties of Timed Automata', 2000.

Example of perturbed semantics: guard enlargement²

- Timed automaton \mathcal{A} :



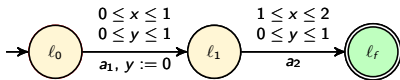
- Representation of an enlarged guard $0 \leq x \leq 1 \wedge 0 \leq y \leq 1$



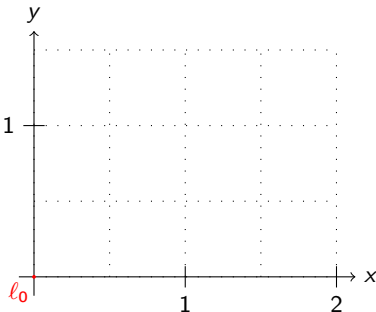
²Puri, 'Dynamical Properties of Timed Automata', 2000.

Example of perturbed semantics: delay perturbation³

- Timed automaton \mathcal{A} :



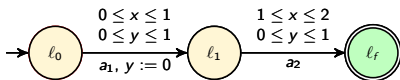
- Run with delay perturbations of at most $\delta = 0.2$



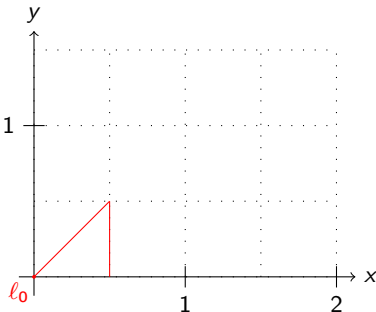
³Bouyer, Fang and Markey, 'Permissive strategies in timed automata and games', 2015.

Example of perturbed semantics: delay perturbation³

- Timed automaton \mathcal{A} :



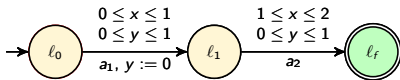
- Run with delay perturbations of at most $\delta = 0.2$



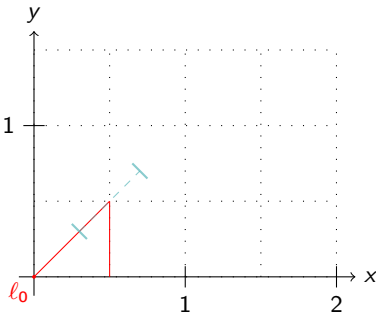
³Bouyer, Fang and Markey, 'Permissive strategies in timed automata and games', 2015.

Example of perturbed semantics: delay perturbation³

- Timed automaton \mathcal{A} :



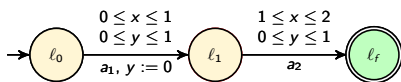
- Run with delay perturbations of at most $\delta = 0.2$



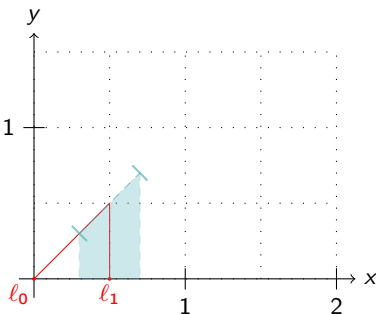
³Bouyer, Fang and Markey, 'Permissive strategies in timed automata and games', 2015.
Emily Clement Robustness of timed automata

Example of perturbed semantics: delay perturbation³

- Timed automaton \mathcal{A} :



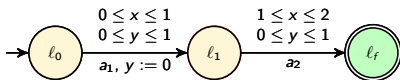
- Run with delay perturbations of at most $\delta = 0.2$



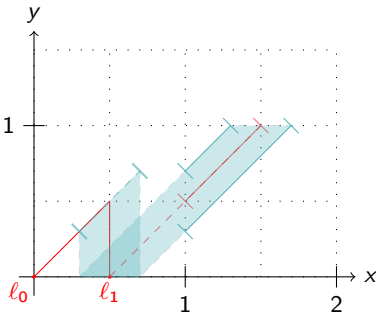
³Bouyer, Fang and Markey, 'Permissive strategies in timed automata and games', 2015.

Example of perturbed semantics: delay perturbation³

- Timed automaton \mathcal{A} :



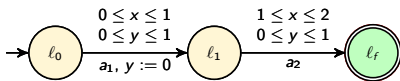
- Run with delay perturbations of at most $\delta = 0.2$



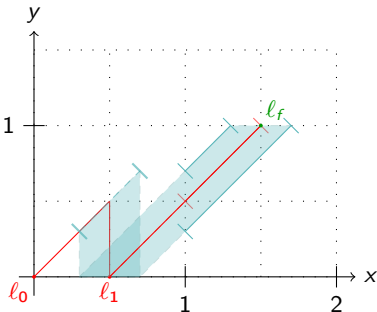
³Bouyer, Fang and Markey, 'Permissive strategies in timed automata and games', 2015.
Emily Clement

Example of perturbed semantics: delay perturbation³

- Timed automaton \mathcal{A} :

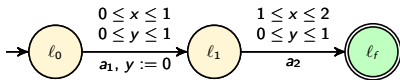


- Run with delay perturbations of at most $\delta = 0.2$

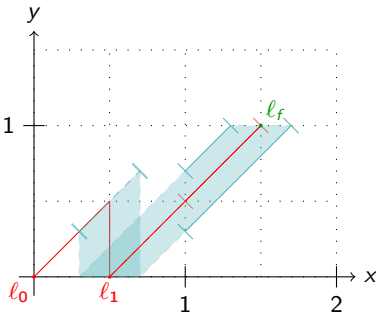


³Bouyer, Fang and Markey, 'Permissive strategies in timed automata and games', 2015.
Emily Clement

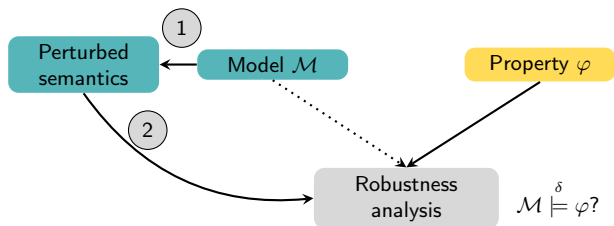
- Timed automaton \mathcal{A} :

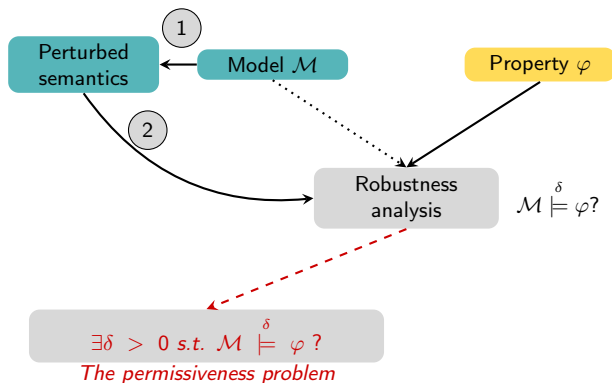


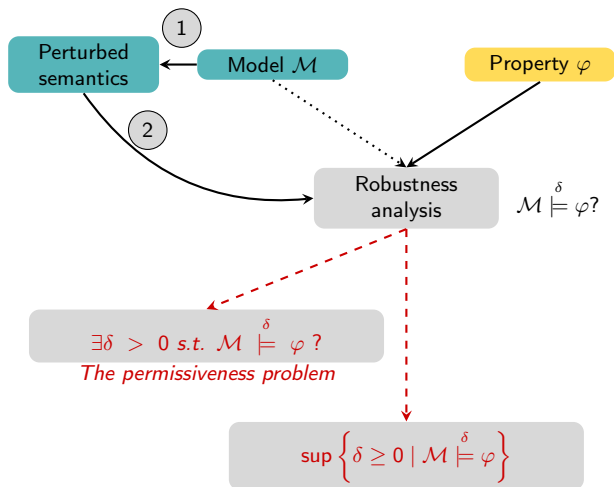
- Run with delay perturbations of at most $\delta = 0.2$

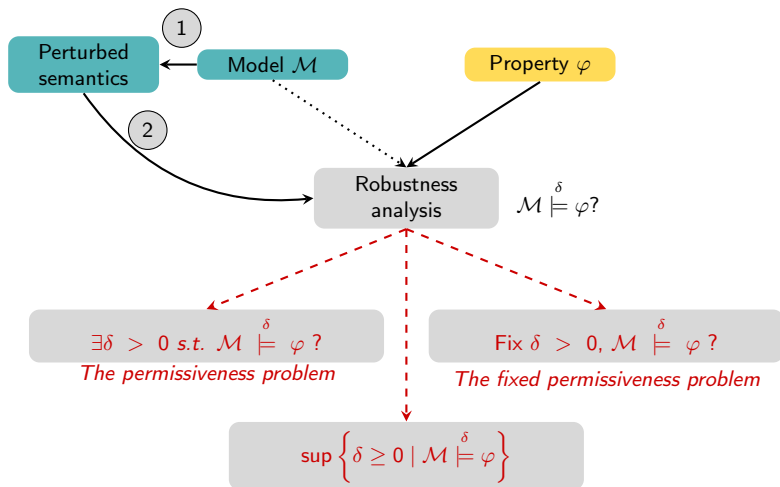


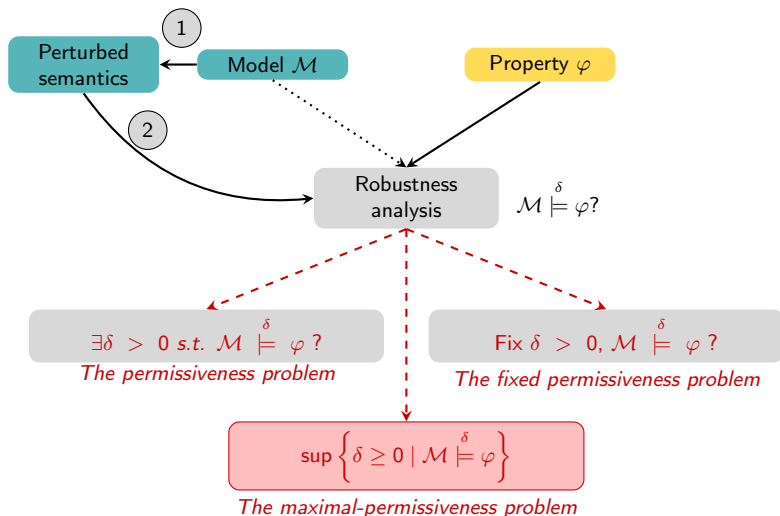
³Bouyer, Fang and Markey, 'Permissive strategies in timed automata and games', 2015.
 Emily Clement Robustness of timed automata

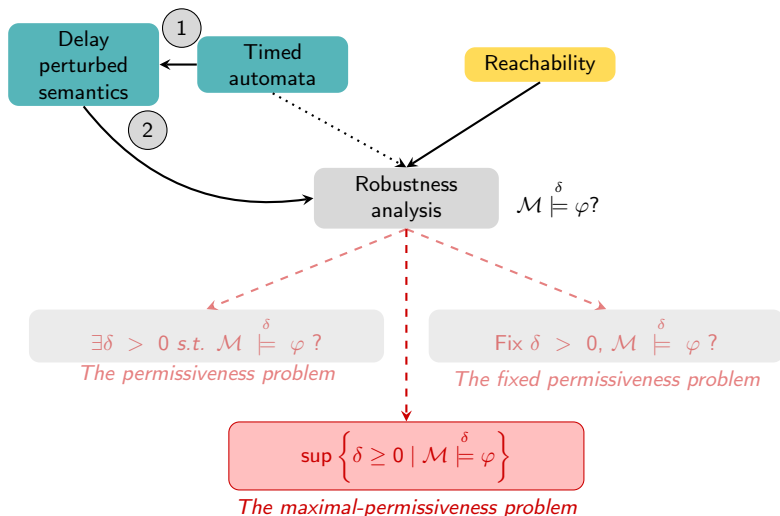






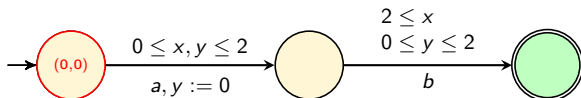






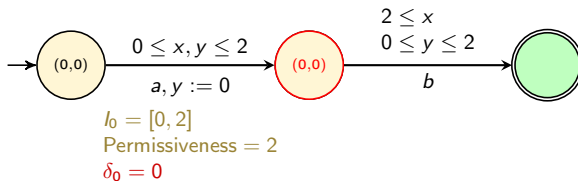
Our permissive semantics: a turn-based game

- ▷ Player
- ▷ Opponent



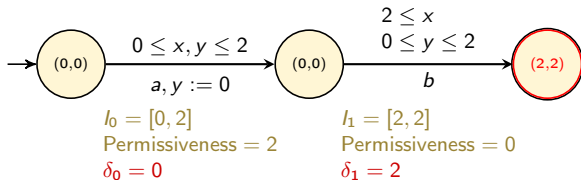
Our permissive semantics: a turn-based game

- ▷ Player
- ▷ Opponent



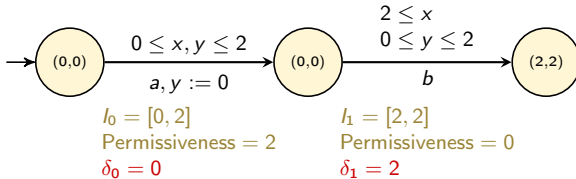
Our permissive semantics: a turn-based game

- ▷ Player
- ▷ Opponent



Our permissive semantics: a turn-based game

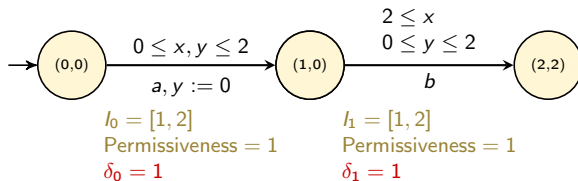
- ▶ **Player** : maximises the permissiveness
- ▶ **Opponent** : minimises the permissiveness



Permissiveness of the run : $\min(|I_0|, |I_1|) = \min(2, 0) = 0$

Our permissive semantics: a turn-based game

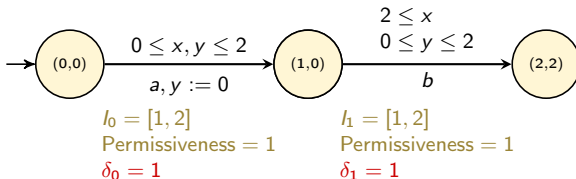
- ▶ **Player** : maximises the permissiveness
- ▶ **Opponent** : minimises the permissiveness



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

Our permissive semantics: a turn-based game

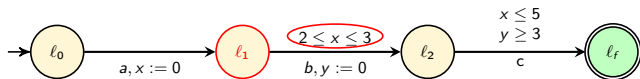
- ▶ **Player** : maximises the permissiveness
- ▶ **Opponent** : minimises the permissiveness



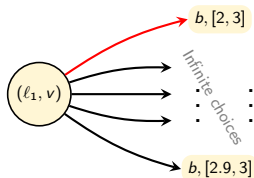
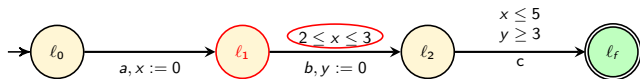
Permissiveness of the run : $\min(|l_0|, |l_1|) = \min(1, 1) = 1$

- ▶ **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

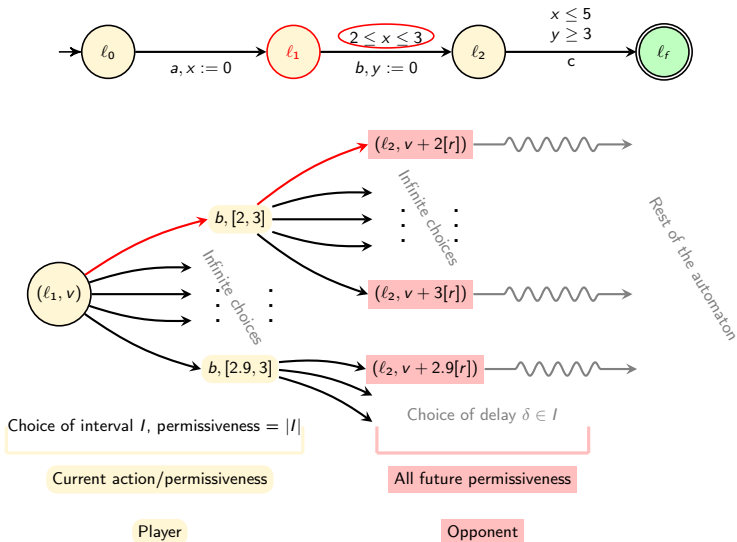


Choice of interval I , permissiveness = $|I|$

Current action/permissiveness

Player

Permissiveness: an infinite number of choices



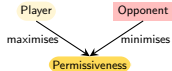
Goal of this thesis: compute the maximal permissiveness

• Permissive semantics: a turn-based game

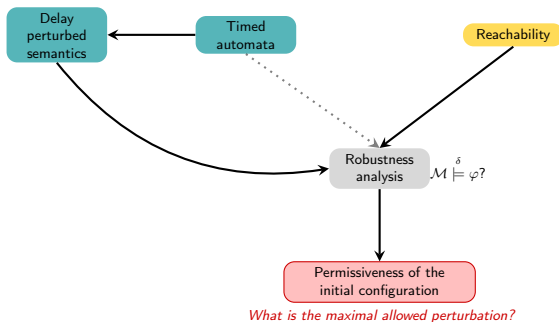
Player : Interval & action: (I, a)

Opponent : delay $\delta \in I$

• Permissiveness



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



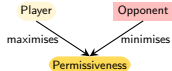
Goal of this thesis: compute the maximal permissiveness

• Permissive semantics: a turn-based game

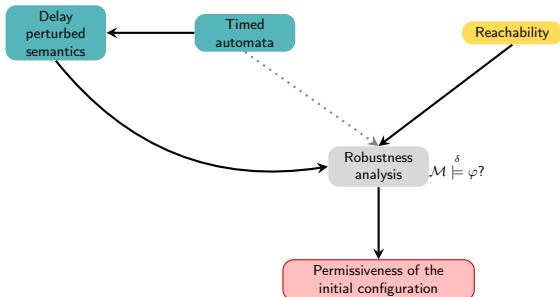
Player : Interval & action: (I, a)

Opponent : delay $\delta \in I$

• Permissiveness



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



What is the maximal allowed perturbation?

1st contribution

 Implemented

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

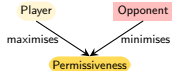
Goal of this thesis: compute the maximal permissiveness

• Permissive semantics: a turn-based game

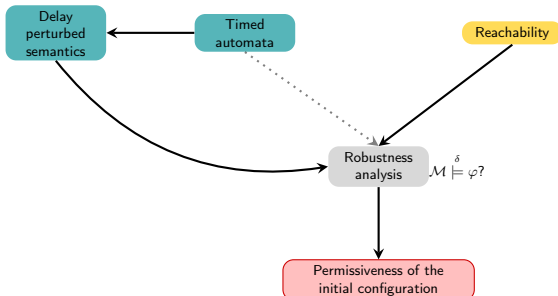
Player : Interval & action: (I, a)

Opponent : delay $\delta \in I$

• Permissiveness



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



What is the maximal allowed perturbation?

1st contribution

 Implemented

2nd contribution

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

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

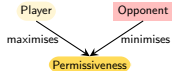
Goal of this thesis: compute the maximal permissiveness

• Permissive semantics: a turn-based game

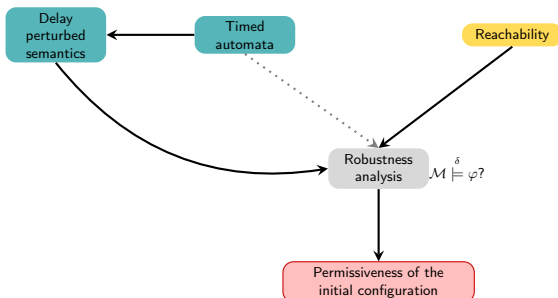
Player : Interval & action: (I, a)

Opponent : delay $\delta \in I$

• Permissiveness



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



What is the maximal allowed perturbation?

1st contribution

 Implemented

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

2nd contribution

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

3rd contribution

 Implemented

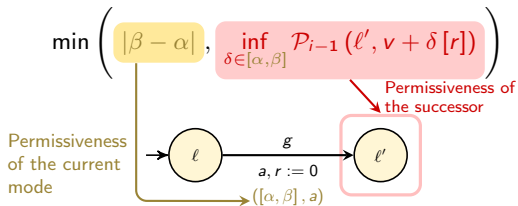
Numerical computation of the permissiveness for acyclic TA, with **approximate results**

First contribution: an exact symbolic computation⁴

⁴Clement, Jérôme, Markey and Mentré, 'Computing Maximally-Permissive Strategies in Acyclic Timed Automata', 2020.

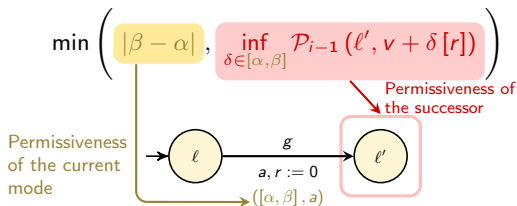
(1st contribution) A symbolic backward algorithm

- Strategy of the player: maximises



(1st contribution) A symbolic backward algorithm

- 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

(1st contribution) The permissiveness function

- 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

- Goal of our algorithm

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

- 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

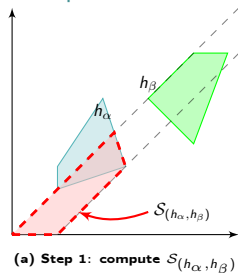
(1st contribution) Steps of the algorithm (linear timed automata)

- 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)



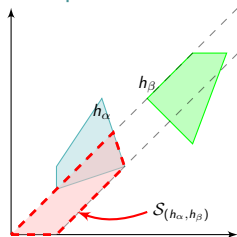
(1st contribution) Steps of the algorithm (linear timed automata)

- Goal: find the α and β that maximises:

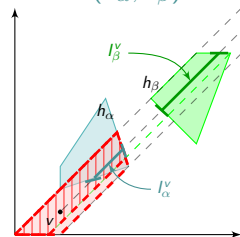
$$\min \left(|\beta - \alpha|, \mathcal{P}_{i-1}(\ell', v + \alpha[r]), \mathcal{P}_{i-1}(\ell', v + \beta[r]) \right)$$

- $v \mapsto \mathcal{P}_i(\ell, v)$: 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

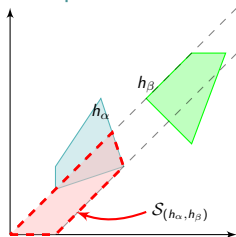
(1st contribution) Steps of the algorithm (linear timed automata)

- Goal: find the α and β that maximises:

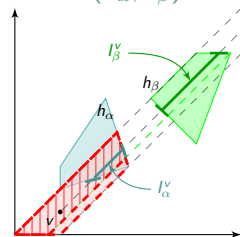
$$\min \left(|\beta - \alpha|, \mathcal{P}_{i-1}(\ell', v + \alpha[r]), \mathcal{P}_{i-1}(\ell', v + \beta[r]) \right)$$

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

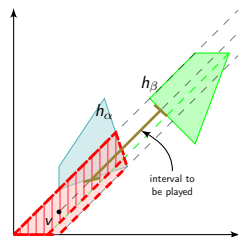
- Steps: for each couple of cells (h_α, h_β)



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

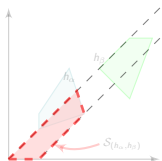


(b) Step 2: compute the possible α and β

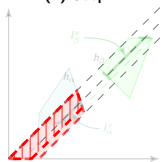


(c) Step 3: compute the optimal α and β

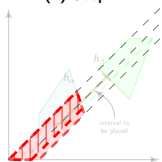
(1st contribution) Example



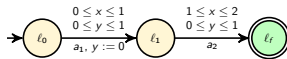
(a) Step 1



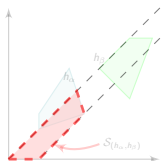
(b) Step 2



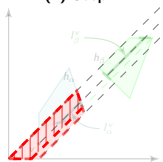
(c) Step 3



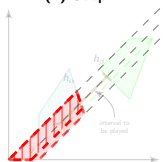
(1st contribution) Example



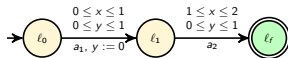
(a) Step 1



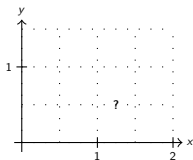
(b) Step 2



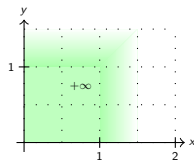
(c) Step 3



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



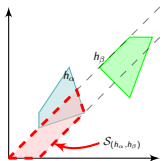
(a) Permissiveness on l_1



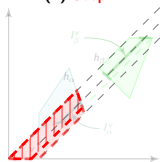
(b) Permissiveness on l_f

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

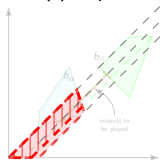
(1st contribution) Example



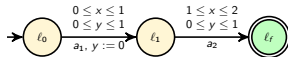
(a) Step 1



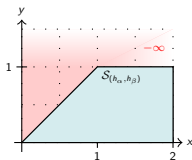
(b) Step 2



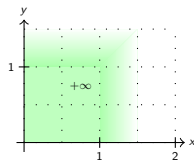
(c) Step 3



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

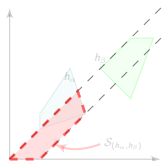


(a) Permissiveness on ℓ_1

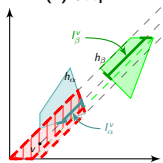


(b) Permissiveness on ℓ_f

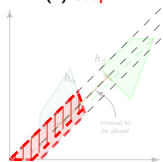
Step 1: computing $\mathcal{S}(h_\alpha, h_\beta)$ (Fourier-Motzkin algorithm)



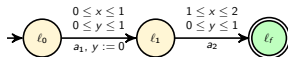
(a) Step 1



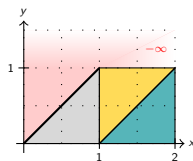
(b) Step 2



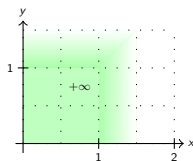
(c) Step 3



- 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

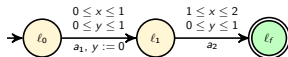
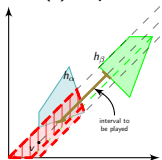
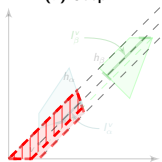
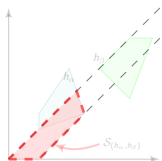


(b) Permissiveness on l_f

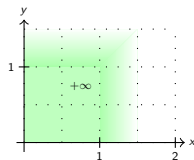
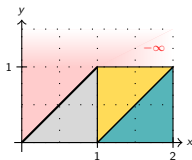
Step 2: computing the intervals of α and β (Fourier-Motzkin algorithm)

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

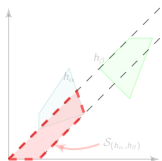
(1st contribution) Example



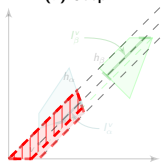
- Permissiveness on l_1 : maximise $\min(\beta - \alpha, \mathcal{P}_0(l_f, v + \alpha), \mathcal{P}_0(l_f, v + \beta))$



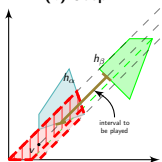
Step 3: computing the optimal α and β , s.t $\alpha \leq \beta$, that maximises...



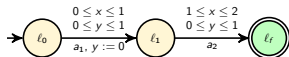
(a) Step 1



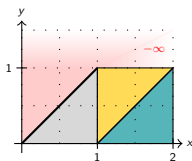
(b) Step 2



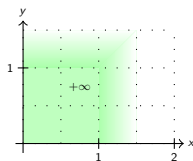
(c) Step 3



- 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

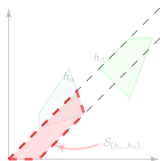


(b) Permissiveness on ℓ_f

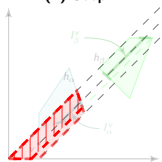
Step 3: computing the optimal α and β , s.t $\alpha \leq \beta$, that maximises...

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

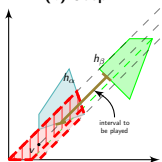
(1st contribution) Example



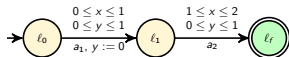
(a) Step 1



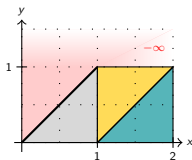
(b) Step 2



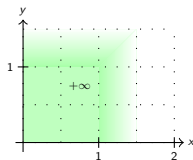
(c) Step 3



- 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

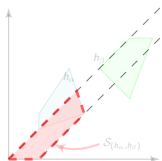


(b) Permissiveness on l_f

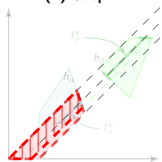
Step 3: computing the optimal α and β , s.t $\alpha \leq \beta$, that maximises...

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

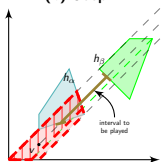
(1st contribution) Example



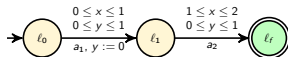
(a) Step 1



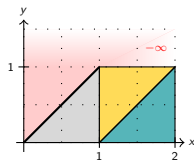
(b) Step 2



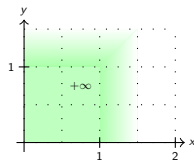
(c) Step 3



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



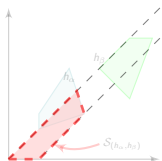
(a) Permissiveness on l_1



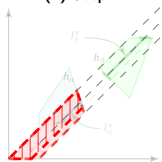
(b) Permissiveness on l_f

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

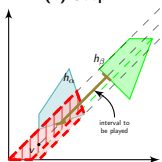
$$\min(\beta - \alpha, +\infty, +\infty) \quad \text{(Technical lemma)}$$



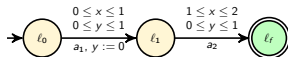
(a) Step 1



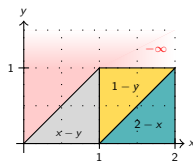
(b) Step 2



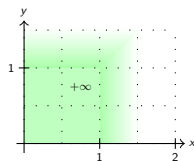
(c) Step 3



- 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



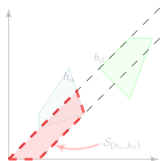
(b) Permissiveness on l_f

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

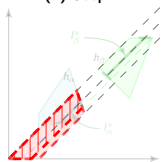
$$\min(\beta - \alpha, +\infty, +\infty) \quad \text{(Technical lemma)}$$

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

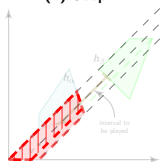
(1st contribution) Example



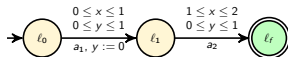
(a) Step 1



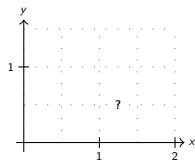
(b) Step 2



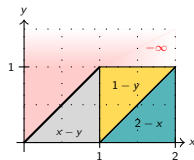
(c) Step 3



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



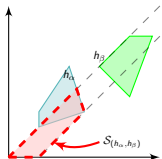
(a) Permissiveness on l_0



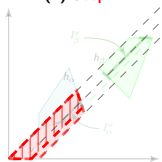
(b) Permissiveness on l_1

Let us fix $h_\alpha = h_\beta = \begin{matrix} \triangle \\ x - y \end{matrix}$

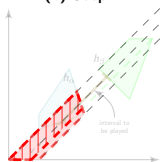
(1st contribution) Example



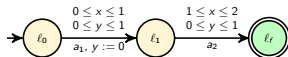
(a) Step 1



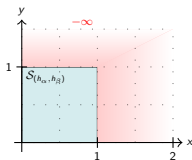
(b) Step 2



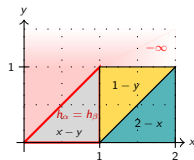
(c) Step 3



- 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

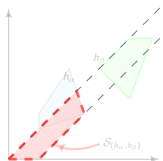


(b) Permissiveness on l_1

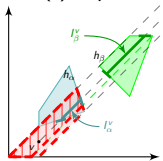
Step 1: Computing the corresponding entry set $\mathcal{S}(h_\alpha, h_\beta)$

(Fourier-Motzkin algorithm)

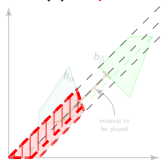
(1st contribution) Example



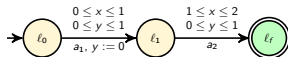
(a) Step 1



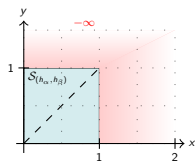
(b) Step 2



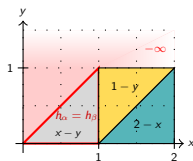
(c) Step 3



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



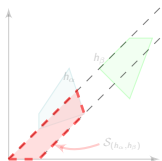
(a) Permissiveness on l_0



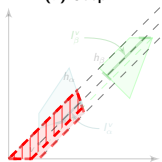
(b) Permissiveness on l_1

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

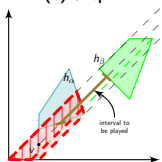
$$I_\alpha^v = I_\beta^v = [0, \min(1 - x, 1 - y)]$$



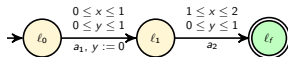
(a) Step 1



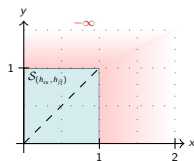
(b) Step 2



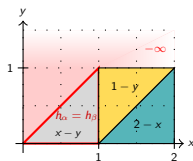
(c) Step 3



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



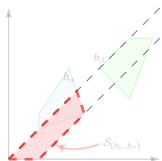
(a) Permissiveness on l_0



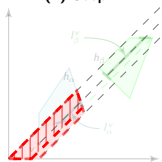
(b) Permissiveness on l_1

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

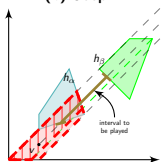
$$\min(\beta - \alpha, 1 \cdot \alpha + x, 1 \cdot \beta + x) \quad \text{(Technical lemma)}$$



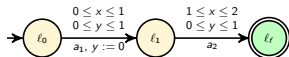
(a) Step 1



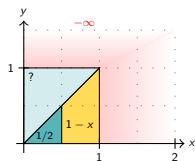
(b) Step 2



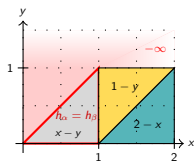
(c) Step 3



- 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

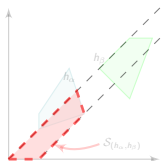


(b) Permissiveness on l_1

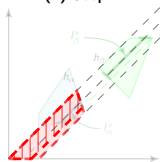
- Step 3: For $y \leq x$: Computing the optimal α and β in $[0, 1 - x]^2$, s.t. $\alpha \leq \beta$, that maximise:

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

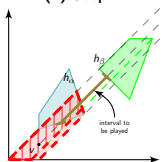
(1st contribution) Example



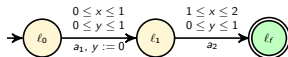
(a) Step 1



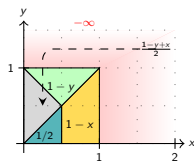
(b) Step 2



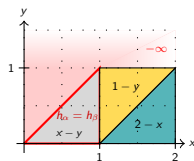
(c) Step 3



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

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

- The principle of the algorithm

$$x + \alpha - 1 \geq 0$$

$$x + y + \alpha - 3 \geq 0$$

System of linear equations (S)

- The principle of the algorithm

$$x + \alpha - 1 \geq 0$$

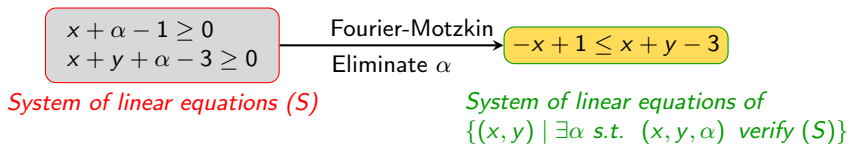
$$x + y + \alpha - 3 \geq 0$$

Fourier-Motzkin

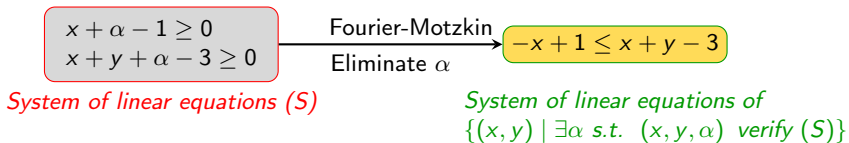
Eliminate α

System of linear equations (S)

- The principle of the algorithm

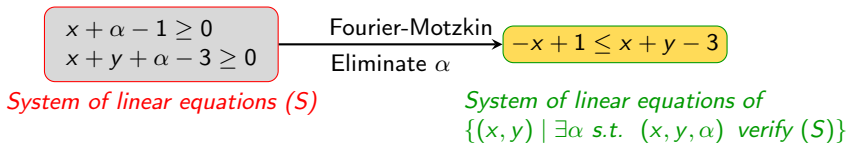


- The principle of the algorithm



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

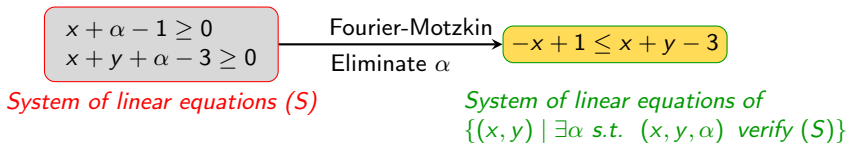
- The principle of the algorithm



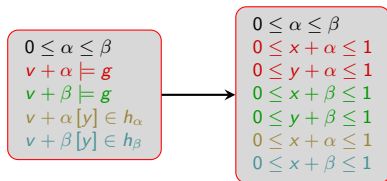
- Computing the entry set $\mathcal{S}_{(h_\alpha, h_\beta)}$: the set of $(x, y) \in \mathbb{R}_+^2$ s.t. $\exists \alpha, \beta$ s.t.

$$\begin{aligned} 0 &\leq \alpha \leq \beta \\ v + \alpha &\models g \\ v + \beta &\models g \\ v + \alpha[y] &\in h_\alpha \\ v + \beta[y] &\in h_\beta \end{aligned}$$

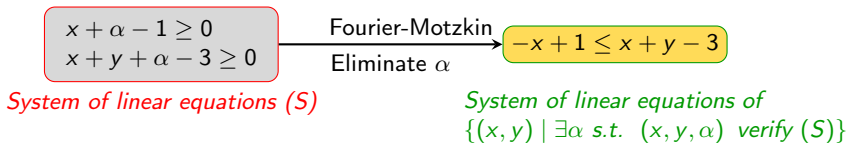
- The principle of the algorithm



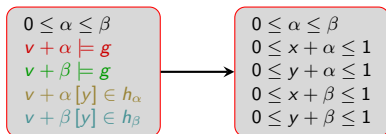
- Computing the entry set $\mathcal{S}_{(h_\alpha, h_\beta)}$: the set of $(x, y) \in \mathbb{R}_+^2$ s.t. $\exists \alpha, \beta$ s.t.



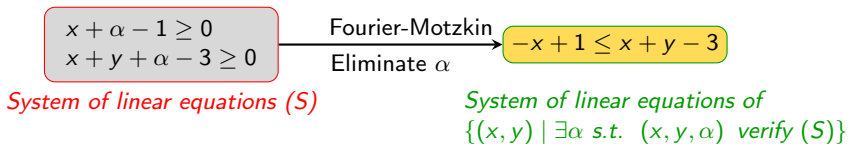
- The principle of the algorithm



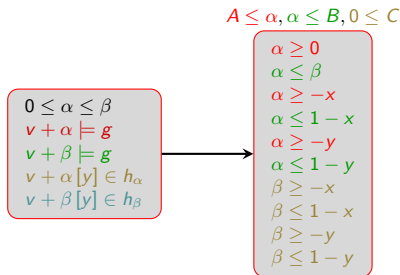
- Computing the entry set $\mathcal{S}_{(h_\alpha, h_\beta)}$: the set of $(x, y) \in \mathbb{R}_+^2$ s.t. $\exists \alpha, \beta$ s.t.



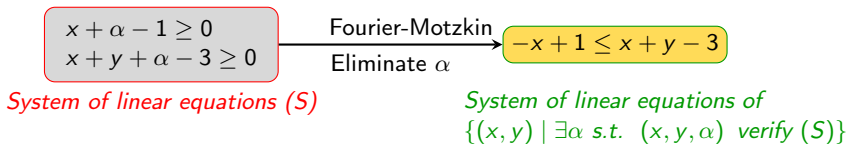
- The principle of the algorithm



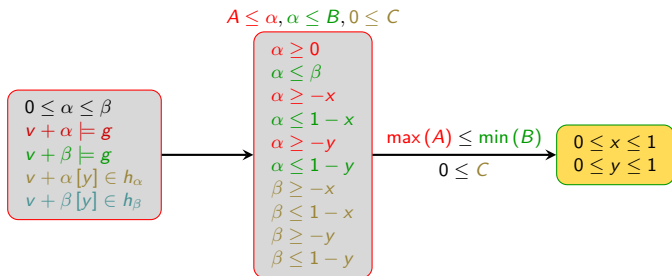
- Computing the entry set $\mathcal{S}_{(h_\alpha, h_\beta)}$: the set of $(x, y) \in \mathbb{R}_+^2$ s.t. $\exists \alpha, \beta$ s.t.



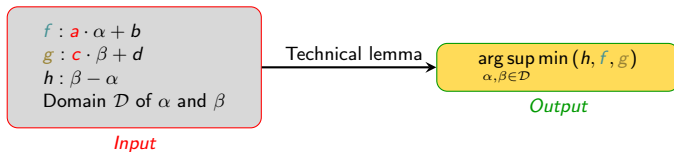
- The principle of the algorithm



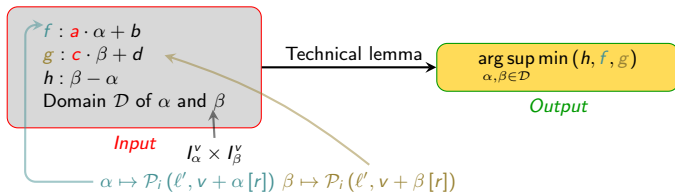
- Computing the entry set $\mathcal{S}_{(h_\alpha, h_\beta)}$: the set of $(x, y) \in \mathbb{R}_+^2$ s.t. $\exists \alpha, \beta$ s.t.



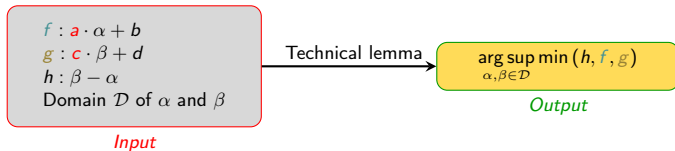
(1st contribution) Technical Lemma



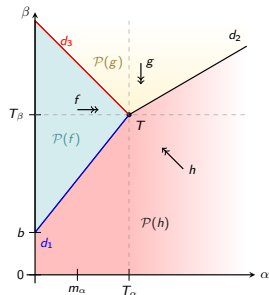
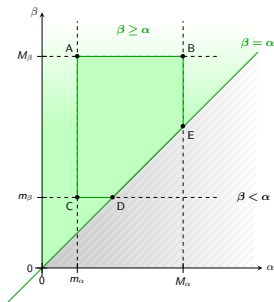
(1st contribution) Technical Lemma



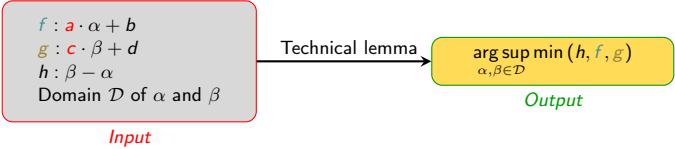
(1st contribution) Technical Lemma



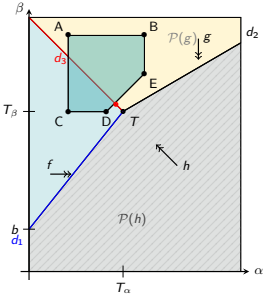
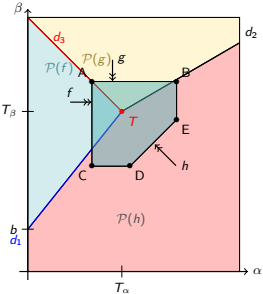
- When $a > 0$ and $c < 0$:



(1st contribution) Technical Lemma



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



- The algorithm
 - ▷ Limited to acyclic timed automata

- The algorithm

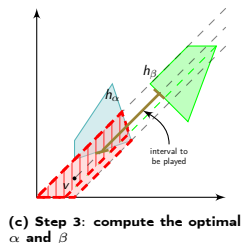
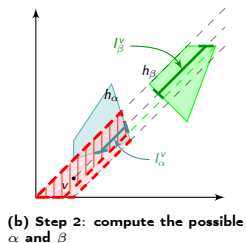
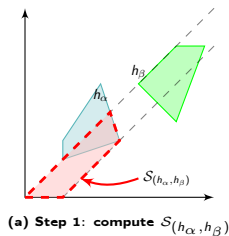
- ▷ Limited to acyclic timed automata
- ▷ Upper bound time complexity: **non-elementary**
- ▷ Complexity: grows with the number of cells, of clocks and d_ℓ

- 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)



- ▶ Based on *pplpy*

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

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

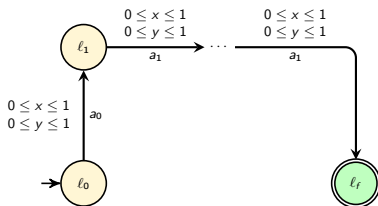
Clocks	Nb. of transitions	Runtime for ℓ_0	Runtime for ℓ_1	Runtime for ℓ_2	Runtime for ℓ_3
2	2	0.82 (2 cells)	0.059 (2 cells)	-	-
2	2	0.071 (6 cells)	0.062 (3 cells)	-	-
2	2	0.73 (24 cells)	0.034 (3 cells)	-	-
2	3	38.16 (582 cells)	1.01 (25 cells)	0.09 (3 cells)	-
3	4	19.95 (234 cells)	0.41 (12 cells)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

(1st contribution) Implementation

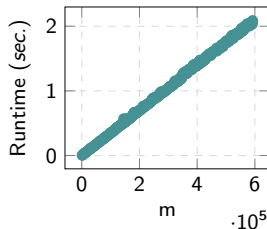
- ▶ Based on *pplpy*
- ▶ Covers the case of **linear timed automata with polyhedral guards**
- ▶ Runtime results on our examples:

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:



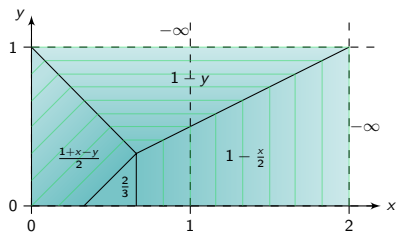
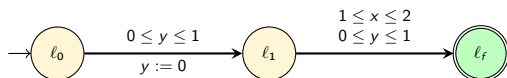
(a) A m transitions timed automaton



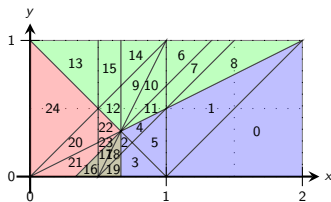
(b) Runtimes depending on the number of transitions (m)

(1st contribution) The overtiling

- Example of overtiling



(a) Permissiveness on l_0



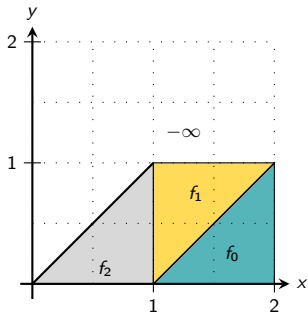
(b) Permissiveness computed by our tool

- ▶ Redundancy in the technical lemma

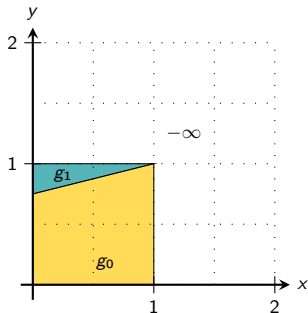
- ▷ Redundancy in the technical lemma
- ▷ Maximisation/Minimisation (when comparing candidate permissiveness functions)

(1st contribution) The causes of the overtaking

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



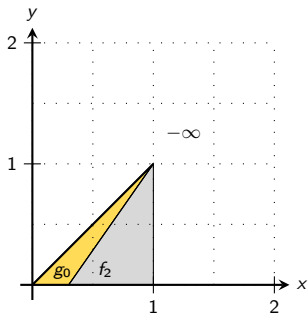
(a) $f: f_0(x, y) = 2 - x$, $f_1(x, y) = 1 - y$,
 $f_2(x, y) = x - y$



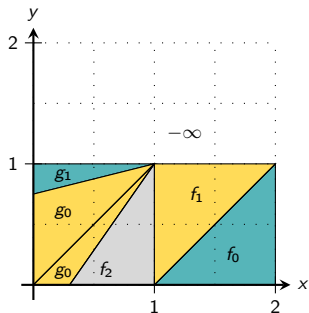
(b) $g: g_0(x, y) = (1 - x) / 2$, $g_1(x, y) = 1 - y$

(1st contribution) The causes of the overtiling

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



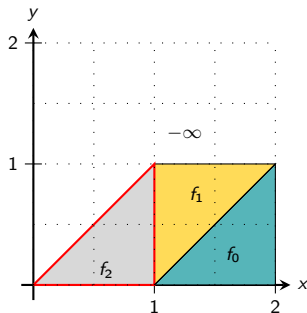
(a) Wrong result



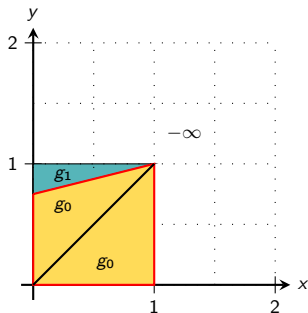
(b) Correct overtiled result $\max(f, g)$

(1st contribution) The causes of the overtiling

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

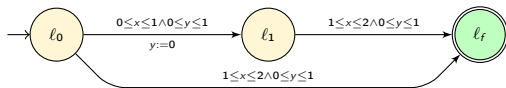


(a) f

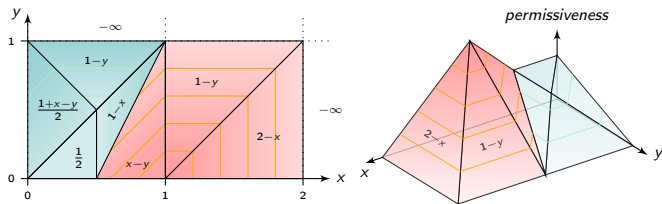


(b) Overtiled representation of g

- Example of an acyclic timed automata



- (1st contribution) Its permissiveness on l_0

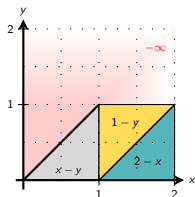


Second contribution: binary and levelled permissiveness.

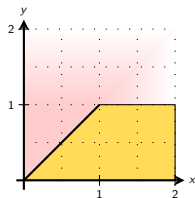
- Binary permissiveness principle

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

- Permissiveness function



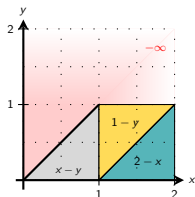
- Binary permissiveness, $p = 0$



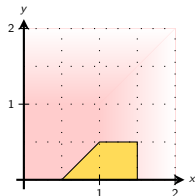
- Binary permissiveness principle

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

- Permissiveness function



- Binary permissiveness, $p = 1/2$

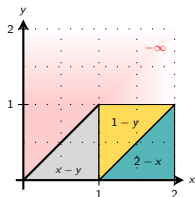


- Some reductions

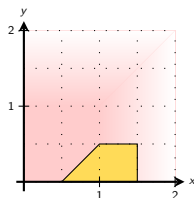
- Binary permissiveness principle

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

- Permissiveness function



- Binary permissiveness, $p = 1/2$



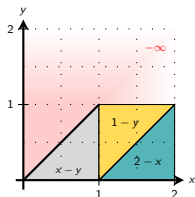
- Some reductions

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

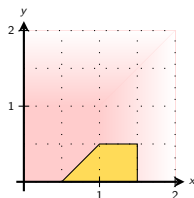
- Binary permissiveness principle

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

- Permissiveness function



- Binary permissiveness, $p = 1/2$



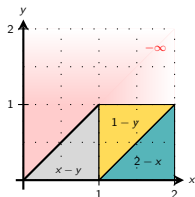
- Some reductions

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

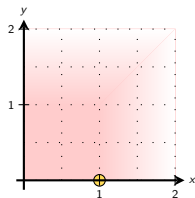
- Binary permissiveness principle

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

- Permissiveness function



- Binary permissiveness, $p = 1$



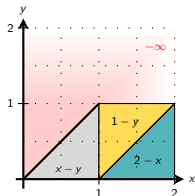
- Some reductions

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

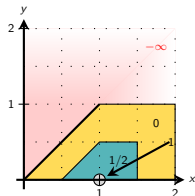
- Binary permissiveness principle

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

- Permissiveness function



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



- Some reductions

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

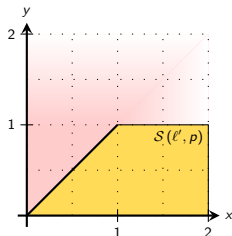
▷ **Levelled permissiveness** $\xrightarrow{\text{reduces}}$ **binary permissiveness**

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

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

- Steps of the algorithm

- ▶ **Step 1:** Compute set of future enabled valuations $\mathcal{S}(\ell', p)$

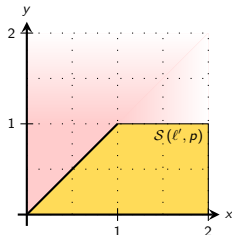


- ▶ **Step 2:** Compute the valuations ν such that there exists $\alpha \geq 0$ that verifies:

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

• Steps of the algorithm

- ▷ **Step 1:** Compute set of future enabled valuations $\mathcal{S}(\ell', p)$

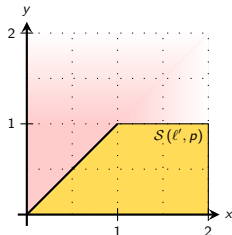


- ▷ **Step 2:** Compute the valuations v such that there exists $\alpha \geq 0$ that verifies:
- $[\alpha, \alpha + p]$ is an enabled move.
 - the successors are in $\mathcal{S}(\ell', p)$.

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

• Steps of the algorithm

- ▷ **Step 1:** Compute set of future enabled valuations $\mathcal{S}(\ell', p)$



- ▷ **Step 2:** Compute the valuations ν such that there exists $\alpha \geq 0$ that verifies:
- $[\alpha, \alpha + p]$ is an enabled move.
 - the successors are in $\mathcal{S}(\ell', p)$.

Fourier-Motzkin

- 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

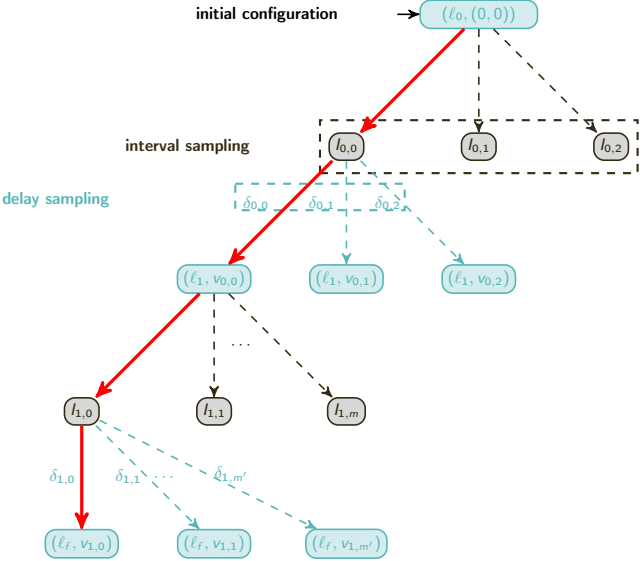
Levelled permissiveness algorithm $\{p_0, \dots, p_m\}$:

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

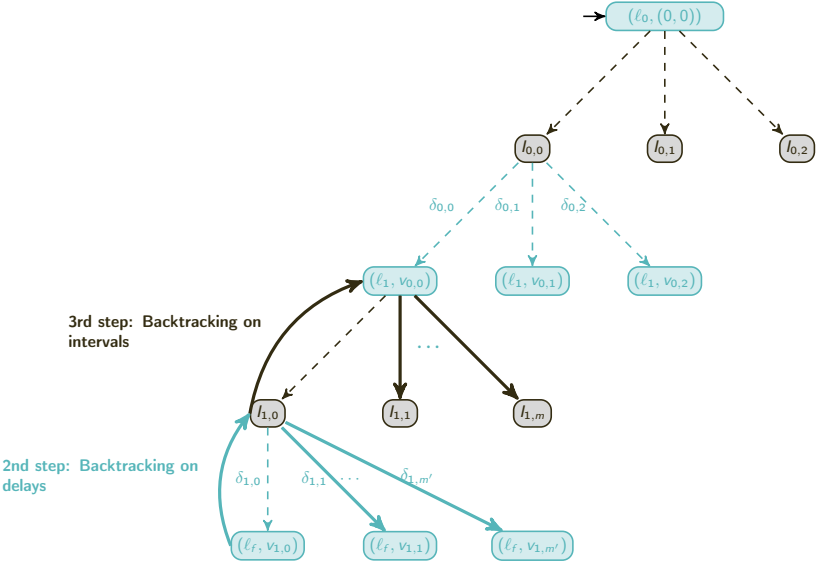
number of levels

Third contribution: an numerical approximative computation.

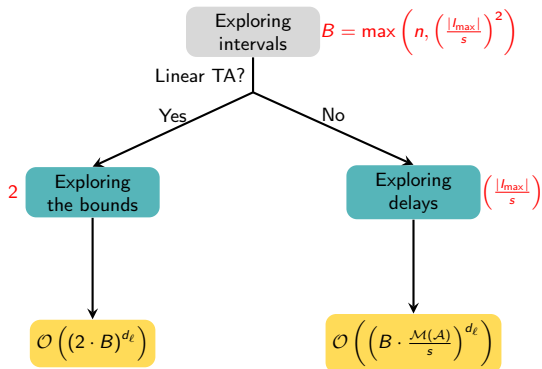
(3rd contribution) Algorithm



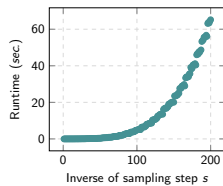
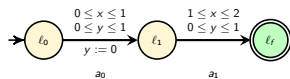
(3rd contribution) Algorithm



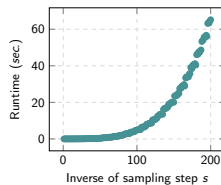
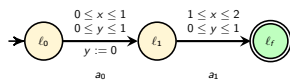
(3rd contribution) Upper bound complexity



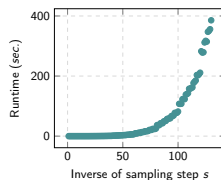
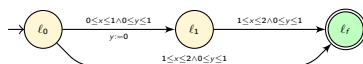
- A linear example



- A linear example



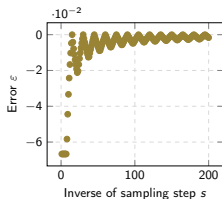
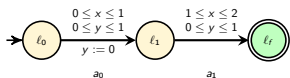
- An acyclic example



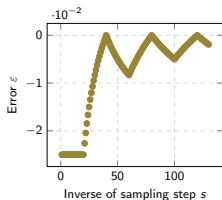
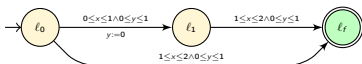
Error $\varepsilon =$ Computed permissiveness – correct permissiveness

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

- A linear example

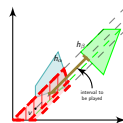
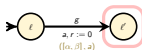


- An acyclic example



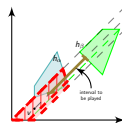
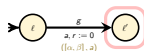
- Symbolic algorithm

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



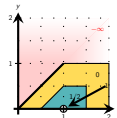
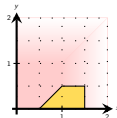
- Symbolic algorithm

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



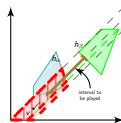
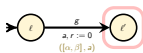
- Binary and levelled permissiveness

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



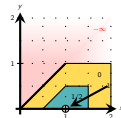
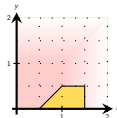
- Symbolic algorithm

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



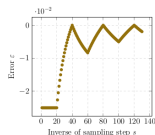
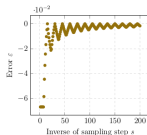
- Binary and levelled permissiveness

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



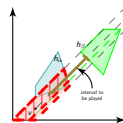
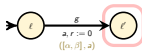
- Numeric algorithm

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



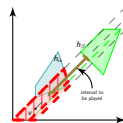
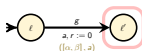
- Symbolic algorithm

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



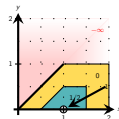
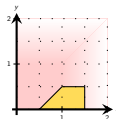
- Symbolic algorithm

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



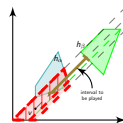
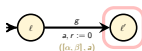
- Binary and levelled permissiveness

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



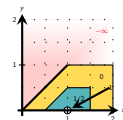
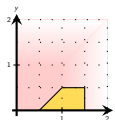
- Symbolic algorithm

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



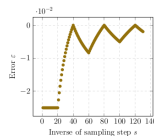
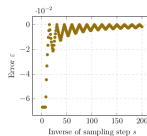
- Binary and levelled permissiveness

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



- Numeric algorithm

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



- Case of two polyhedra

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

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

⁵Bemporad, Fukuda and Torrisi, 'Convexity recognition of the union of polyhedra', 2001.

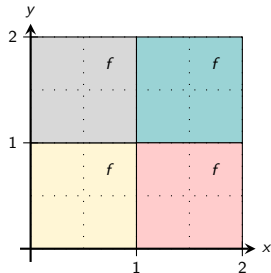
- Case of two polyhedra

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...



⁵Bemporad, Fukuda and Torrisi, 'Convexity recognition of the union of polyhedra', 2001.