

# Computing the maximal permissiveness of timed automata

Emily Clement<sup>312</sup>   Thierry Jéron<sup>1</sup>   Nicolas Markey<sup>1</sup>   David Mentré<sup>2</sup>

<sup>1</sup>IRISA, Inria & CNRS & Univ. Rennes, France

<sup>2</sup>Mitsubishi Electric R&D Centre Europe – Rennes, France: MERCE

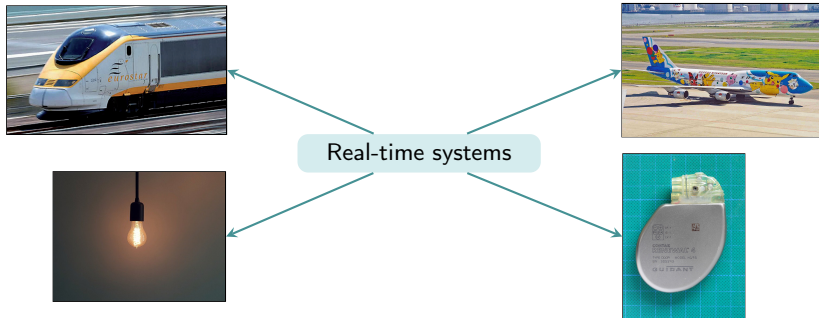
<sup>3</sup>Sorbonne Université, CNRS, Institut des Systèmes Intelligents et de Robotique, ISIR, F-75005 Paris, France

March 16 2023

## Introduction

*Formal methods and model-checking*

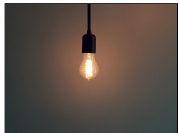
# Why verifying real-time systems?



# Why verifying real-time systems?

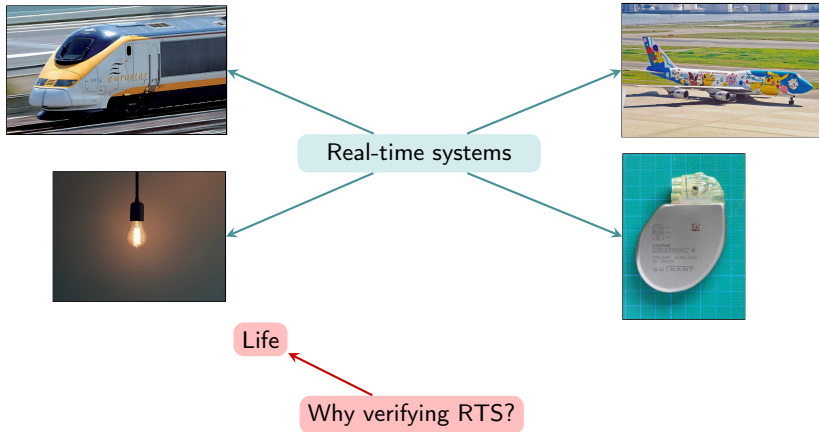


Real-time systems

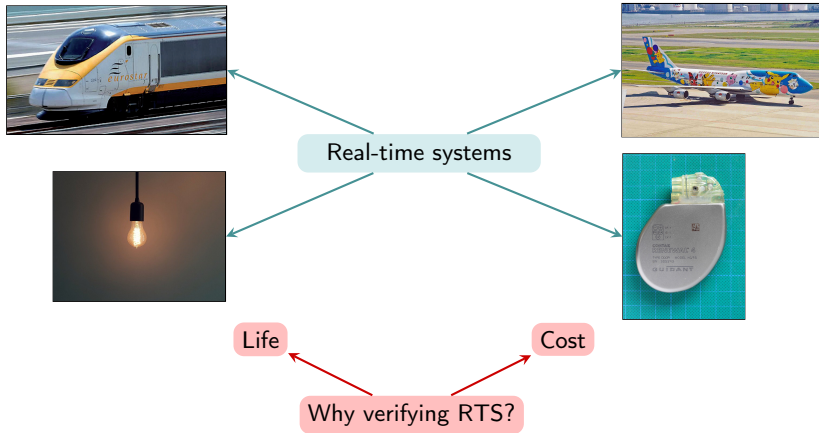


Why verifying RTS?

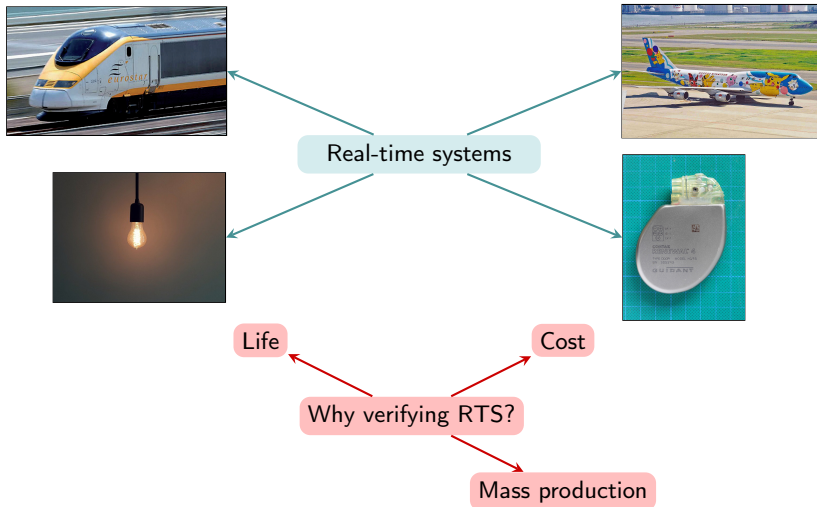
# Why verifying real-time systems?



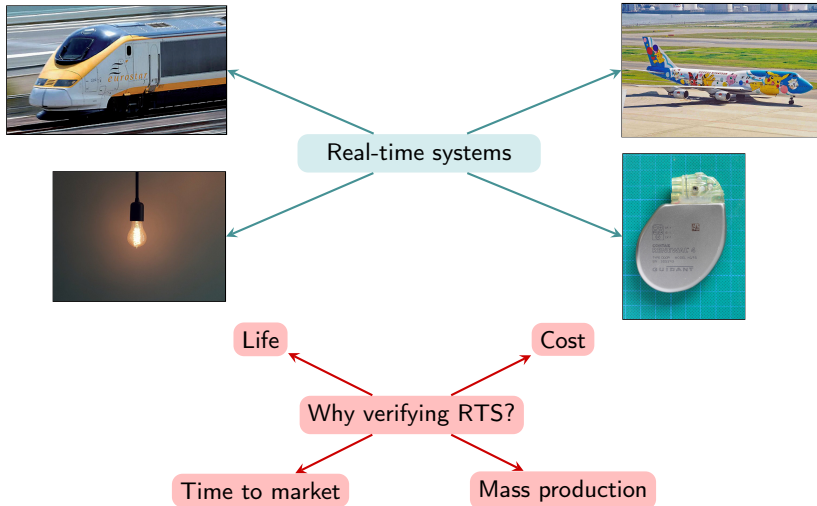
# Why verifying real-time systems?

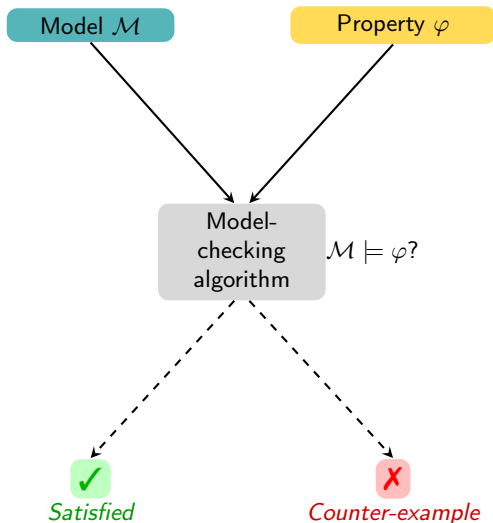


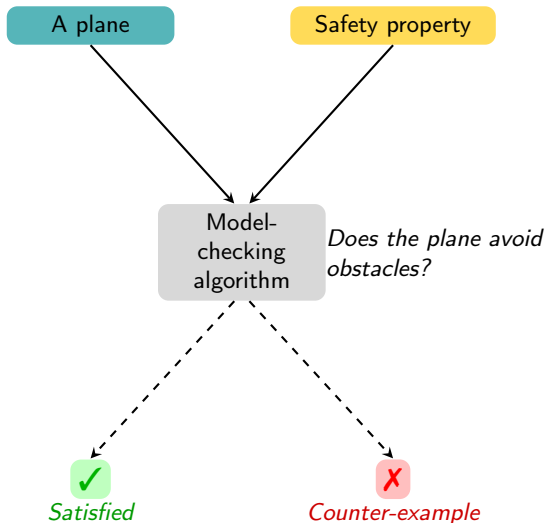
# Why verifying real-time systems?



# Why verifying real-time systems?

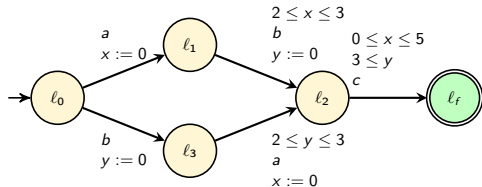






An abstract model Timed automata The property Reachability: a state *can* be visited

- Example: Scheduling system



Acyclic timed automata:

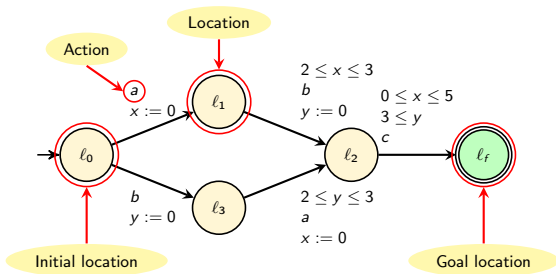


Linear timed automata:



An abstract model Timed automata    The property Reachability: a state *can* be visited

## • Example: Scheduling system



Acyclic timed automata:

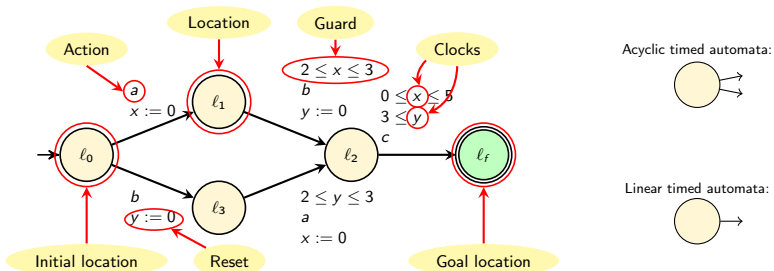


Linear timed automata:



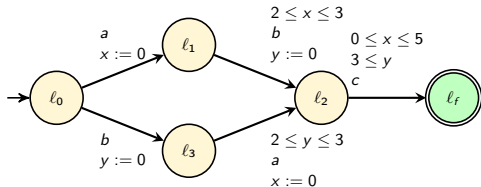
An abstract model Timed automata The property Reachability: a state *can* be visited

- Example: Scheduling system



An abstract model Timed automata The property Reachability: a state *can* be visited

- Example: Scheduling system



Acyclic timed automata:



Linear timed automata:

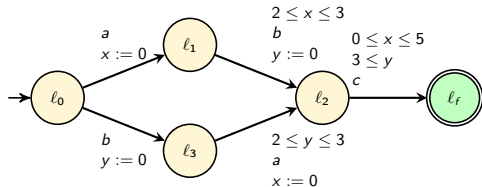


- Can we reach  $l_f$  from  $l_0, (0, 0)$  ?

$l_0, (0, 0)$

An abstract model Timed automata The property Reachability: a state *can* be visited

- Example: Scheduling system



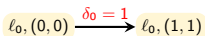
Acyclic timed automata:



Linear timed automata:

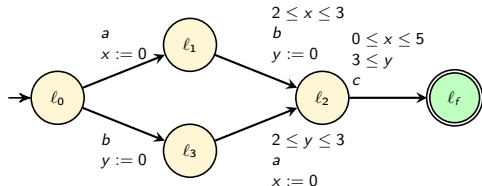


- Can we reach  $l_f$  from  $l_0, (0, 0)$  ?



An abstract model Timed automata The property Reachability: a state *can* be visited

- Example: Scheduling system



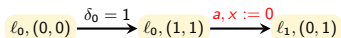
Acyclic timed automata:



Linear timed automata:

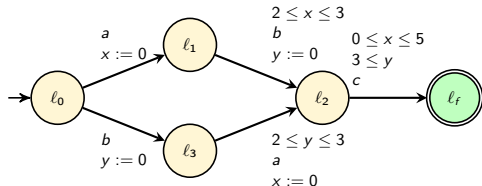


- Can we reach  $l_f$  from  $l_0, (0, 0)$  ?



An abstract model Timed automata The property Reachability: a state *can* be visited

- Example: Scheduling system



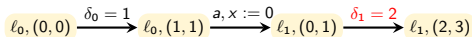
Acyclic timed automata:



Linear timed automata:

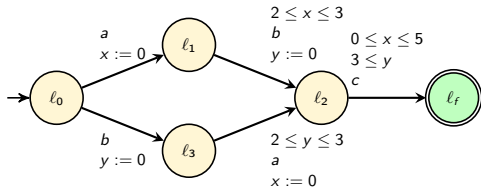


- Can we reach  $l_f$  from  $l_0, (0, 0)$  ?



An abstract model Timed automata The property Reachability: a state *can* be visited

- Example: Scheduling system



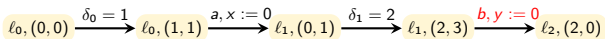
Acyclic timed automata:



Linear timed automata:

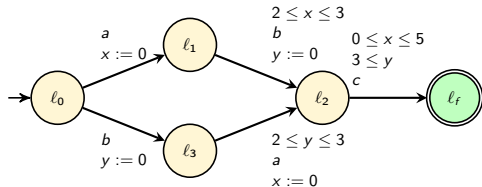


- Can we reach  $l_f$  from  $l_0, (0, 0)$  ?



An abstract model Timed automata The property Reachability: a state *can* be visited

- Example: Scheduling system



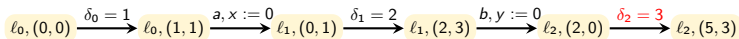
Acyclic timed automata:



Linear timed automata:

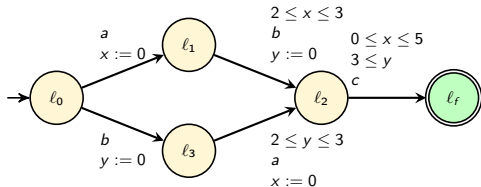


- Can we reach  $l_f$  from  $l_0, (0, 0)$  ?



An abstract model Timed automata The property Reachability: a state *can* be visited

- Example: Scheduling system



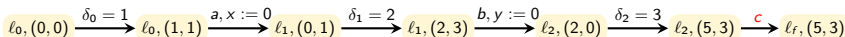
Acyclic timed automata:



Linear timed automata:

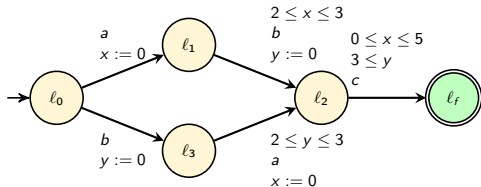


- Can we reach  $l_f$  from  $l_0, (0, 0)$  ?



An abstract model Timed automata The property Reachability: a state *can* be visited

- Example: Scheduling system



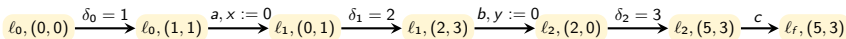
Acyclic timed automata:



Linear timed automata:



- Can we reach  $l_f$  from  $l_0, (0, 0)$  ?

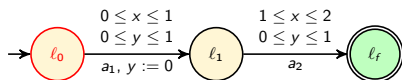


## Robustness of Timed Automata

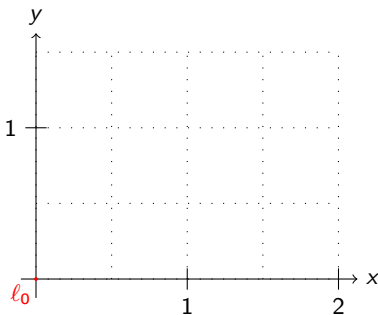
*Delay perturbation*

# Example of perturbed semantics: delay perturbation<sup>1</sup>

- Timed automaton  $\mathcal{A}$ :



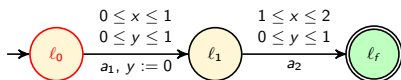
- Run with delay perturbations of at most  $\delta = 0.2$



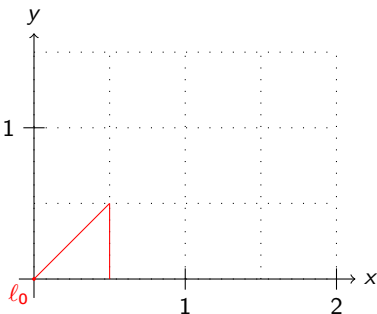
<sup>1</sup>BFM15.

# Example of perturbed semantics: delay perturbation<sup>1</sup>

- Timed automaton  $\mathcal{A}$ :



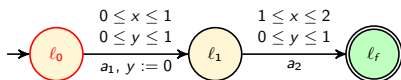
- Run with delay perturbations of at most  $\delta = 0.2$



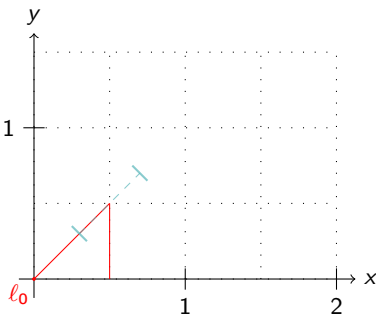
<sup>1</sup>BFM15.

# Example of perturbed semantics: delay perturbation<sup>1</sup>

- Timed automaton  $\mathcal{A}$ :



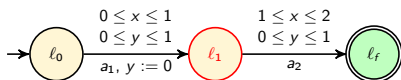
- Run with delay perturbations of at most  $\delta = 0.2$



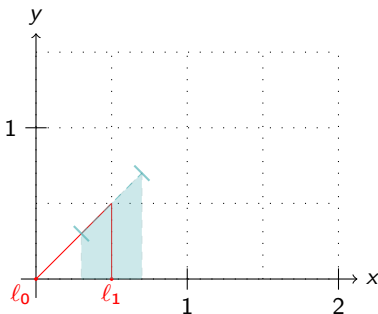
<sup>1</sup>BFM15.

# Example of perturbed semantics: delay perturbation<sup>1</sup>

- Timed automaton  $\mathcal{A}$ :



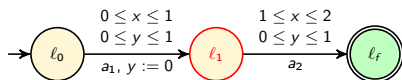
- Run with delay perturbations of at most  $\delta = 0.2$



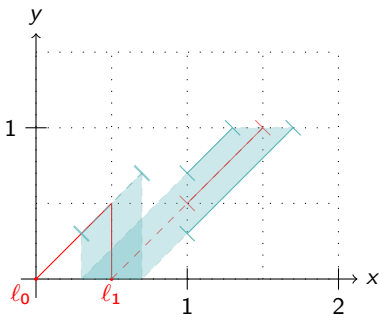
<sup>1</sup>BFM15.

# Example of perturbed semantics: delay perturbation<sup>1</sup>

- Timed automaton  $\mathcal{A}$ :



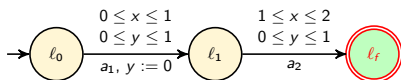
- Run with delay perturbations of at most  $\delta = 0.2$



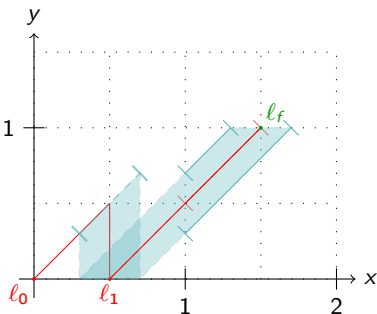
<sup>1</sup>BFM15.

# Example of perturbed semantics: delay perturbation<sup>1</sup>

- Timed automaton  $\mathcal{A}$ :

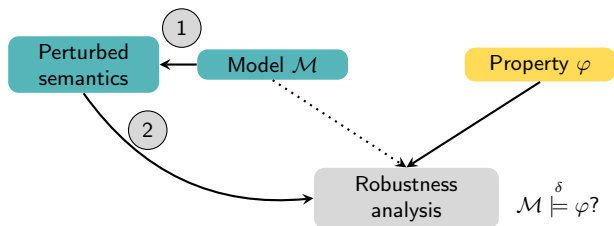


- Run with delay perturbations of at most  $\delta = 0.2$

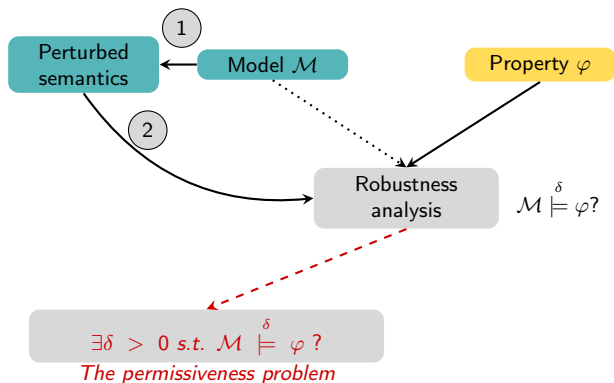


<sup>1</sup>BFM15.

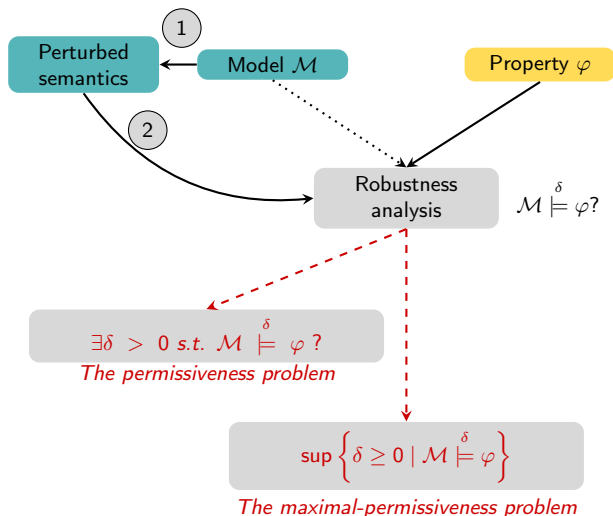
## Robustness: What do we compute?



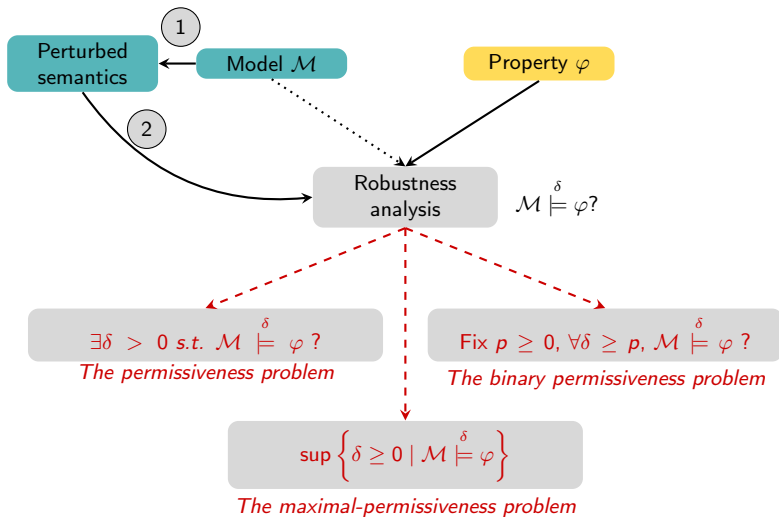
# Robustness: What do we compute?



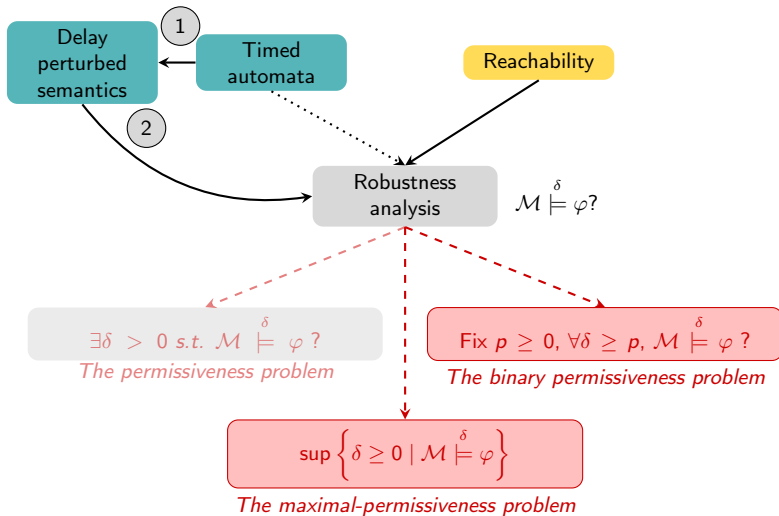
# Robustness: What do we compute?



# Robustness: What do we compute?



# Robustness: What do we compute?



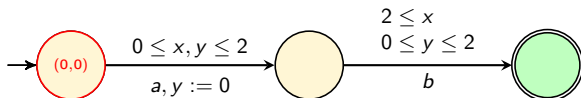
## Our model

*The permissive semantics*

# Our permissive semantics: a turn-based game

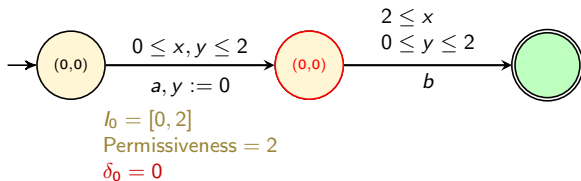
▷ Player

▷ Opponent



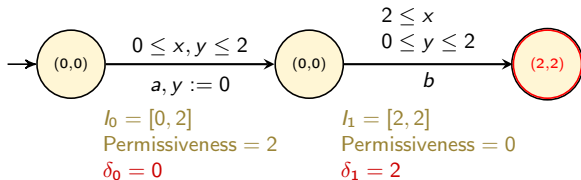
# Our permissive semantics: a turn-based game

- ▷ Player
- ▷ Opponent



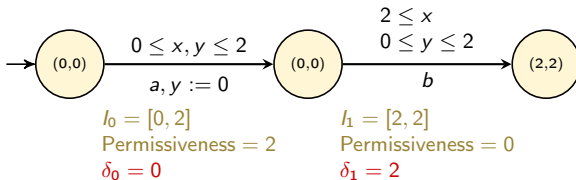
# Our permissive semantics: a turn-based game

- ▶ Player
- ▶ Opponent



# Our permissive semantics: a turn-based game

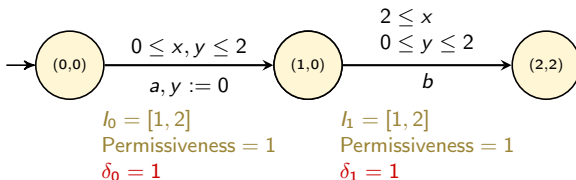
- ▶ **Player** : maximises the permissiveness
- ▶ **Opponent** : minimises the permissiveness



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

# Our permissive semantics: a turn-based game

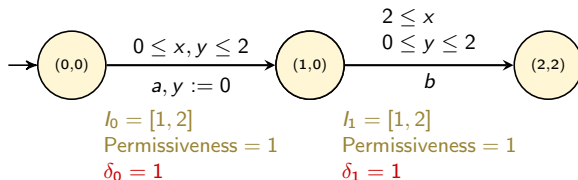
- ▶ **Player** : maximises the permissiveness
- ▶ **Opponent** : minimises the permissiveness



**Permissiveness of the run** :  $\min(|I_0|, |I_1|) = \min(1, 1) = 1$

# Our permissive semantics: a turn-based game

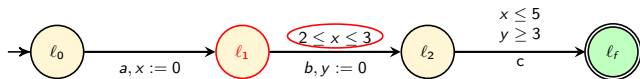
- ▶ **Player** : maximises the permissiveness
- ▶ **Opponent** : minimises the permissiveness



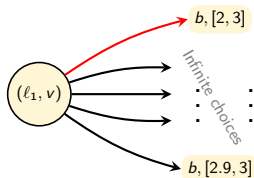
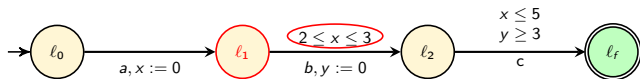
**Permissiveness of the run** :  $\min(|l_0|, |l_1|) = \min(1, 1) = 1$

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

# Permissiveness: an infinite number of choices



# Permissiveness: an infinite number of choices

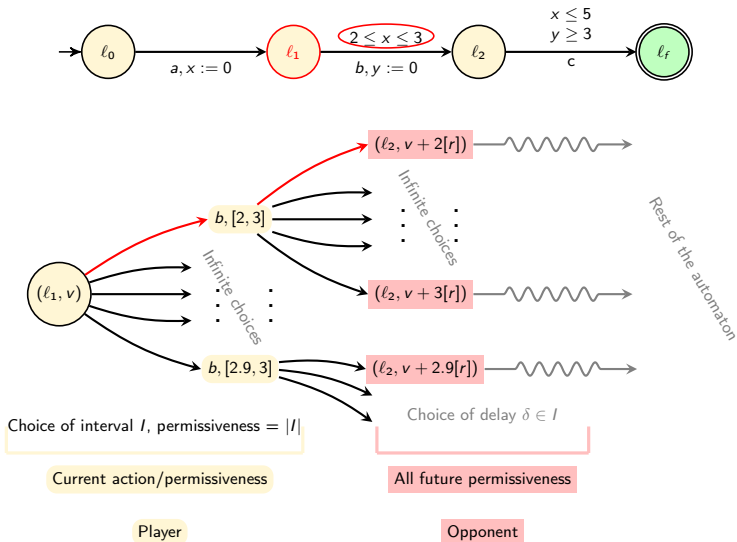


Choice of interval  $I$ , permissiveness =  $|I|$

Current action/permissiveness

Player

# Permissiveness: an infinite number of choices



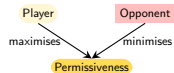
## Our model

*Goals and contributions*

## • Turn-based game

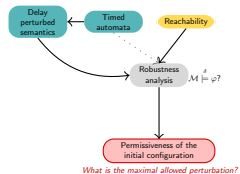
Player : Interval & action:  $(I, a)$

Opponent : delay  $\delta \in I$



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

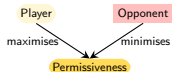
## • Goal



- Turn-based game

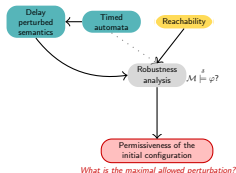
Player : Interval & action:  $(I, a)$

Opponent : delay  $\delta \in I$



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

- Goal



- The maximal-permissiveness problem:

$$\sup \left\{ \delta \geq 0 \mid \mathcal{M} \models^\delta \varphi \right\}$$

Computation



Implemented

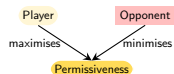
*The permissiveness as function:  $v \mapsto \text{Perm}(\ell, v)$ : linear and acyclic TA*

*The optimal strategy for player & opponent*

- Turn-based game

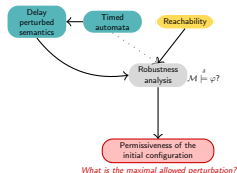
Player : Interval & action:  $(I, a)$

Opponent : delay  $\delta \in I$



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

- Goal



- The maximal-permissiveness problem:

$$\sup \left\{ \delta \geq 0 \mid \mathcal{M} \stackrel{\delta}{\models} \varphi \right\}$$

Computation

*The permissiveness as function:  $v \mapsto \text{Perm}(\ell, v)$ :  
linear and acyclic TA*



Implemented

*The optimal strategy for player & opponent*

- The Binary permissiveness problem:

$$\text{Given } p \geq 0, \forall \delta \geq p, \mathcal{M} \stackrel{\delta}{\models} \varphi ?$$

Computation

*binary permissiveness as function:  $v \mapsto \text{Bin}(\ell, v)$ :  
linear TA*



Implemented

## The permissiveness function

*A sequence of suboptimal functions*

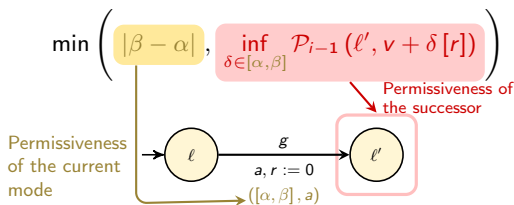
- A recursive function

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

- A recursive function

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

- Strategy of the player: maximises



- 1- Opponent strategy lemma (linear case):

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

- 1- Opponent strategy lemma (linear case):

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

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

- 1- Opponent strategy lemma (linear case):

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

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

Consequence:

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

- 1- Opponent strategy lemma (linear case):

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

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

Consequence:

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

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

- 1- Opponent strategy lemma (linear case):

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

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

Consequence:

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

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

 limit reached in  $d_\ell$  steps

- 1- Opponent strategy lemma (linear case):

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

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

Consequence:

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

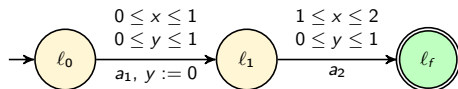
- 2-  $\lim_{i \rightarrow +\infty} \mathcal{P}_i(\ell, v) =$  the permissiveness on  $(\ell, v)$

 limit reached in  $d_\ell$  steps

- Goal:

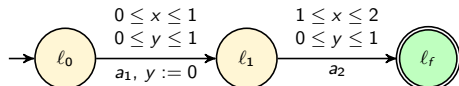
Compute  $v \mapsto \mathcal{P}_{d_\ell}(\ell, v)$  knowing  $v \mapsto \mathcal{P}_{d_\ell-1}(\ell', v)$

- A linear timed automaton

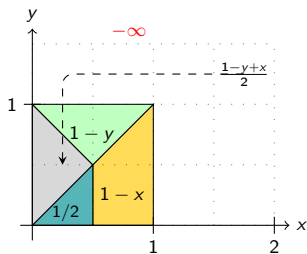


# Example of permissiveness function

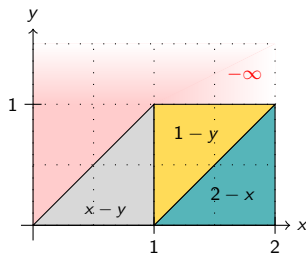
- A linear timed automaton



- Its permissiveness on  $l_0$  and  $l_1$



(a) Permissiveness on  $l_0$



(b) Permissiveness on  $l_1$

## How to compute the permissiveness function ?

- Goal: find the  $\alpha$  and  $\beta$  that maximises:

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

## How to compute the permissiveness function ?

- Goal: find the  $\alpha$  and  $\beta$  that maximises:

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

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

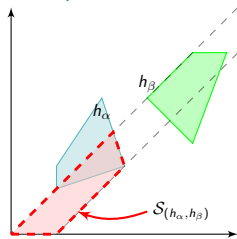
# How to compute the permissiveness function ?

- Goal: find the  $\alpha$  and  $\beta$  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  $(h_\alpha, h_\beta)$



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

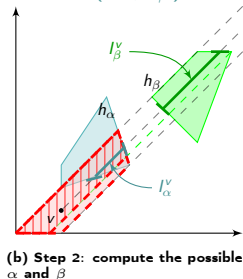
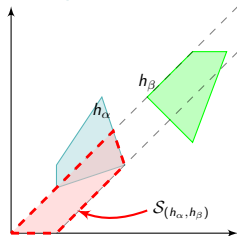
# How to compute the permissiveness function ?

- Goal: find the  $\alpha$  and  $\beta$  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  $(h_\alpha, h_\beta)$



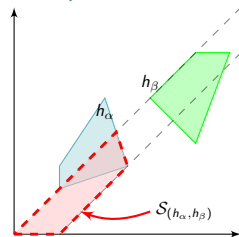
# How to compute the permissiveness function ?

- Goal: find the  $\alpha$  and  $\beta$  that maximises:

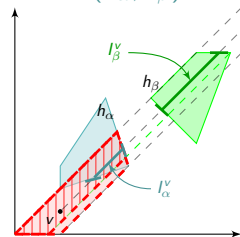
$$\min \left( |\beta - \alpha|, \mathcal{P}_{i-1}(\ell', v + \alpha[r]), \mathcal{P}_{i-1}(\ell', v + \beta[r]) \right)$$

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

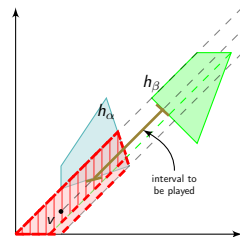
- Steps: for each couple of cells  $(h_\alpha, h_\beta)$



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

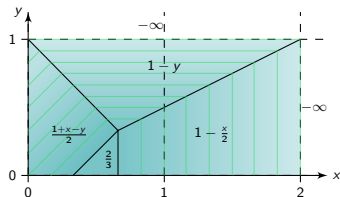
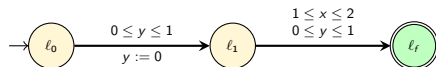


(b) Step 2: compute the possible  $\alpha$  and  $\beta$

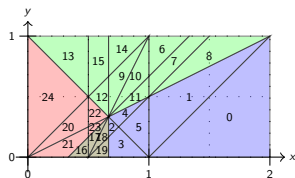


(c) Step 3: compute the optimal  $\alpha$  and  $\beta$

- Overtilling on computation:



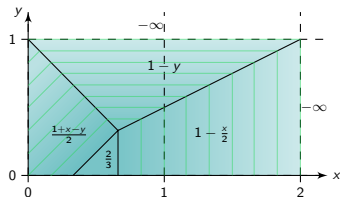
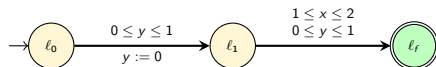
(a) Permissiveness on  $\ell_0$



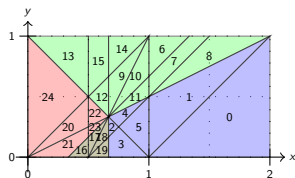
(b) Permissiveness computed by our tool

- Optimisation lemma to tackle overtilling (linear case)

- Overtilling on computation:



(a) Permissiveness on  $\ell_0$

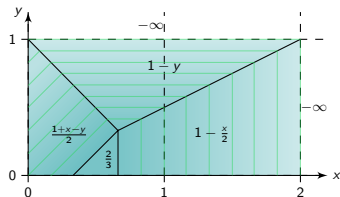
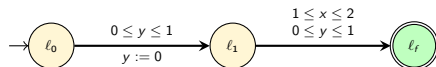


(b) Permissiveness computed by our tool

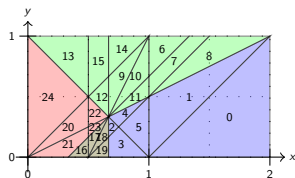
- Optimisation lemma to tackle overtilling (linear case)

▷ **Union of polyhedra**: open problem for more than two polyhedra.

- Overtilling on computation:



(a) Permissiveness on  $\ell_0$



(b) Permissiveness computed by our tool

- Optimisation lemma to tackle overtilling (linear case)

- ▷ **Union of polyhedra**: open problem for more than two polyhedra.
- ▷ The union of polyhedra associated with the same affine function is **equal** to their **convex hull**.

- ▶ Based on *pplpy*

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

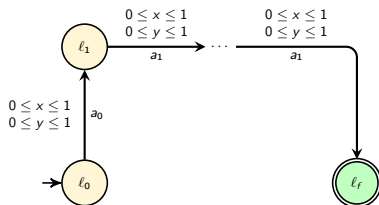
- ▷ Based on *ppipy*
- ▷ Covers the case of **linear timed automata** with **polyhedral guards**
- ▷ Runtime results on our examples:

Clocks	Nb. of transitions	Runtime for $\ell_0$	Runtime for $\ell_1$	Runtime for $\ell_2$	Runtime for $\ell_3$
2	2	0.82 (2 cells)	0.059 (2 cells)	-	-
2	2	0.071 (6 cells)	0.062 (3 cells)	-	-
2	2	0.73 (24 cells)	0.034 (3 cells)	-	-
2	3	38.16 (582 cells)	1.01 (25 cells)	0.09 (3 cells)	-
3	4	19.95 (234 cells)	0.41 (12 cells)	0.56 (6 cells)	0.14 (6 cells)
3	4	143.48 (1825 cells)	0.39 (12 cells)	0.59 (6 cells)	0.14 (6 cells)

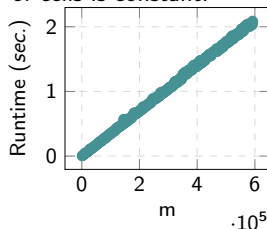
- ▶ Based on *pplpy*
- ▶ Covers the case of **linear timed automata with polyhedral guards**
- ▶ Runtime results on our examples:

Clocks	Nb. of transitions	Runtime for $\ell_0$	Runtime for $\ell_1$	Runtime for $\ell_2$	Runtime for $\ell_3$
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



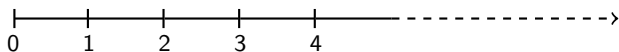
(b) Runtimes depending on the number of transitions ( $m$ )

## Symbolic computation of binary and levelled permissiveness

*Computing an approximated, but with controlled error, value of the permissiveness*

- Goal of computing approximated (controlled) results
  - ▷ A symbolic computation
  - ▷ A bound of the approximation of the value of the permissiveness
  - ▷ Reduce the number of cells

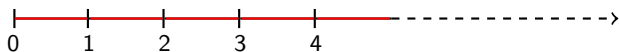
- Goal of computing approximated (controlled) results
  - ▷ A symbolic computation
  - ▷ A bound of the approximation of the value of the permissiveness
  - ▷ Reduce the number of cells
- Example for levelled permissiveness for  $[0, 1[, \dots [4, +\infty[$ :



Levelled:

Compute:  $v \mapsto ([p, p + 1[$  s.t.  $\text{Perm}(l, v) \in [p, p + 1[$

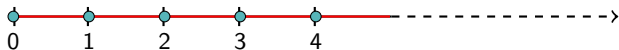
- Goal of computing approximated (controlled) results
  - ▷ A symbolic computation
  - ▷ A bound of the approximation of the value of the permissiveness
  - ▷ Reduce the number of cells
- Example for levelled permissiveness for  $[0, 1[ , \dots [4, +\infty[$ :



Levelled:

Compute:  $v \mapsto ([p, p + 1[$  s.t.  $\text{Perm}(l, v) \in [p, p + 1[$

- Goal of computing approximated (controlled) results
  - ▷ A symbolic computation
  - ▷ A bound of the approximation of the value of the permissiveness
  - ▷ Reduce the number of cells
- Example for levelled permissiveness for  $[0, 1[, \dots [4, +\infty[$ :



Levelled:

Compute:  $v \mapsto ([p, p + 1[$  s.t.  $\text{Perm}(\ell, v) \in [p, p + 1[$

↓ reduces

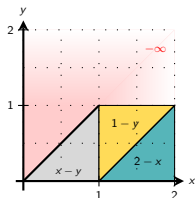
Binary, for threshold  $p \geq 0$ :

Compute:  $v \mapsto \text{Perm}(\ell, v) \geq p$

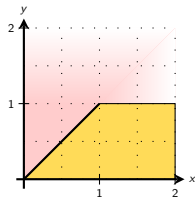
- Binary permissiveness principle

Fix  $p \geq 0$  and  $\ell$ , compute  $S(\ell, p) = \{v \in \mathbb{R}_+^n \mid \mathcal{P}_{d_\ell}(\ell, v) \geq p\}$

- Permissiveness function



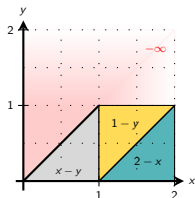
- Binary permissiveness,  $p = 0$



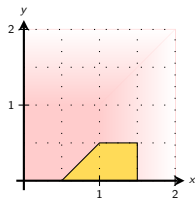
- Binary permissiveness principle

Fix  $p \geq 0$  and  $\ell$ , compute  $\mathcal{S}(\ell, p) = \{v \in \mathbb{R}_+^n \mid \mathcal{P}_{d_\ell}(\ell, v) \geq p\}$

- Permissiveness function



- Binary permissiveness,  $p = 1/2$



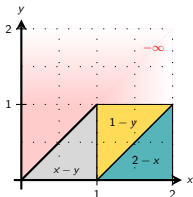
- First optimizations

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

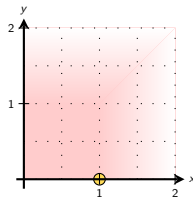
- Binary permissiveness principle

Fix  $p \geq 0$  and  $\ell$ , compute  $\mathcal{S}(\ell, p) = \{v \in \mathbb{R}_+^n \mid \mathcal{P}_{d_\ell}(\ell, v) \geq p\}$

- Permissiveness function



- Binary permissiveness,  $p = 1$



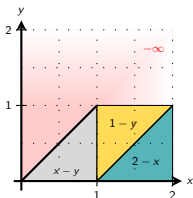
- First optimizations

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

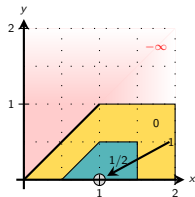
- Binary permissiveness principle

Fix  $p \geq 0$  and  $\ell$ , compute  $\mathcal{S}(\ell, p) = \{v \in \mathbb{R}_+^n \mid \mathcal{P}_{d_\ell}(\ell, v) \geq p\}$

- Permissiveness function



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



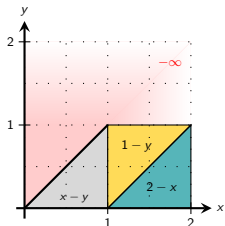
- First optimizations

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

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

# Differences between Permissiveness and binary permissiveness functions

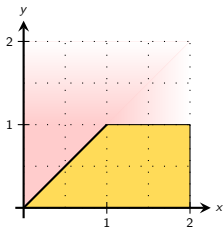
- Permissiveness



*Sequence of suboptimal permissiveness functions*

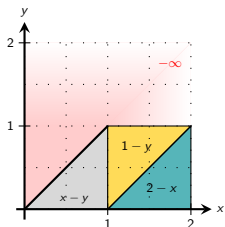
$$\mathcal{P}_i(\ell, v) = \sup_{\{(\alpha, \beta), a\} \in \text{p-moves}(\ell, v)} \min(\beta - \alpha, \mathcal{P}_{i-1}(\ell', v + \alpha[r]), \mathcal{P}_{i-1}(\ell', v + \beta[r]))$$

- Binary permissiveness



# Differences between Permissiveness and binary permissiveness functions

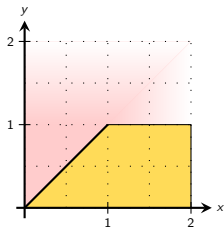
## • Permissiveness



*Sequence of suboptimal permissiveness functions*

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

## • Binary permissiveness



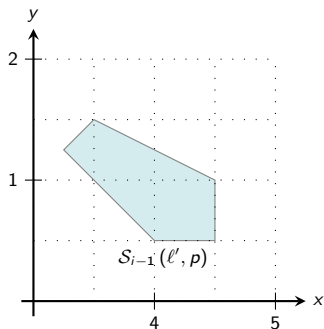
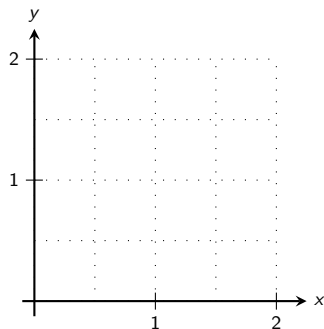
*Sequence of suboptimal binary functions*

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

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

- Algorithm:

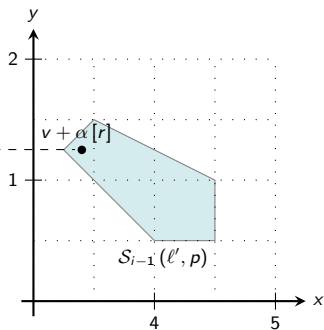
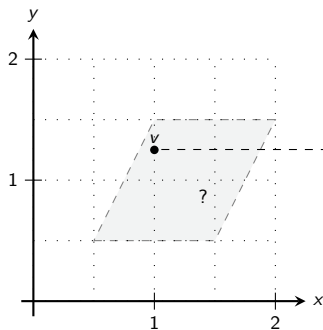
- ▷ Compute  $\mathcal{S}_{i-1}(\ell', \rho)$



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

- Algorithm:

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

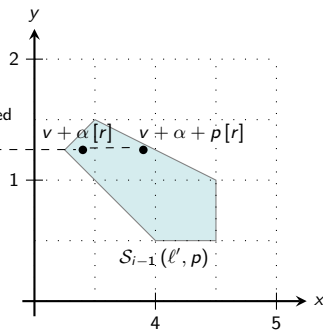
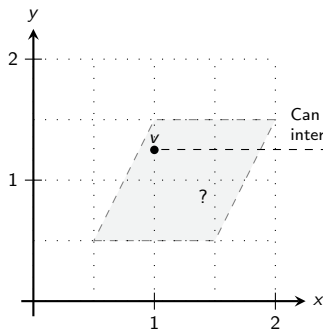


# Algorithm intuition: compute $\mathcal{S}_i(\ell, \rho)$ knowing $\mathcal{S}_{i-1}(\ell', \rho)$

- Algorithm:

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

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



Binary permissiveness algorithm:

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

longest path between  
 $\ell$  and a goal location

maximal number of constraints of any guard

Binary permissiveness algorithm:

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

longest path between  
 $\ell$  and a goal location

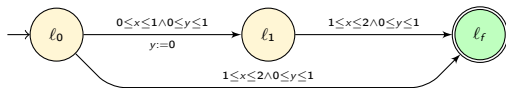
maximal number of constraints of any guard

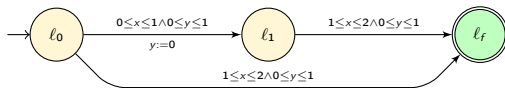
Levelled permissiveness algorithm  $\{p_0, \dots, p_m\}$ :

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

## What happens for **acyclic** Timed Automata?

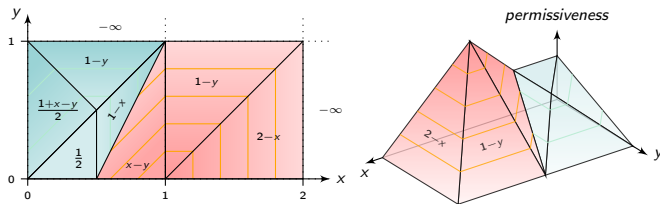
*Permissiveness & Binary permissiveness*

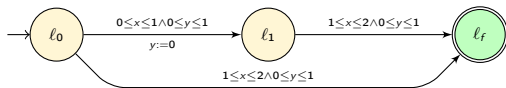




- Its permissiveness function on  $l_0$

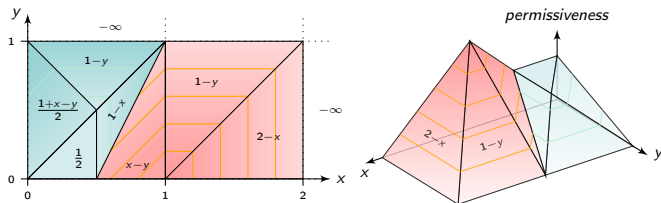
Permissiveness is no longer concave in the general case:





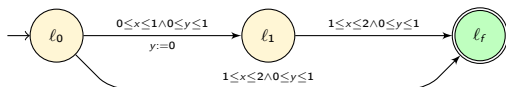
- Its permissiveness function on  $l_0$

Permissiveness is no longer concave in the general case:



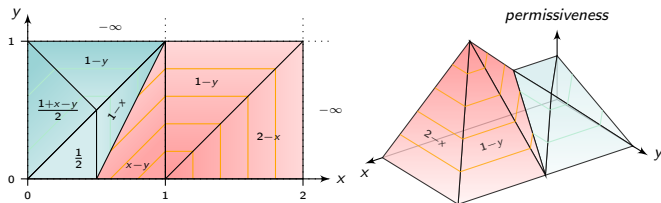
- Adapted algorithm

- ▶ A more complicated strategy for the opponent.



- Its permissiveness function on  $l_0$

Permissiveness is no longer concave in the general case:



- Adapted algorithm

- ▷ A more complicated strategy for the opponent.
- ▷ Algorithmic upper bound complexity becomes non-elementary.

- Changes for acyclic timed automata

- The set of winning valuation is no longer a **unique** polyhedron in general
  - Principle: for a fixed  $p$ , find if we can find an enabled interval  $[\alpha, \alpha + p]$  that crosses only polyhedra of  $S_{i-1}(\ell', p)$ :

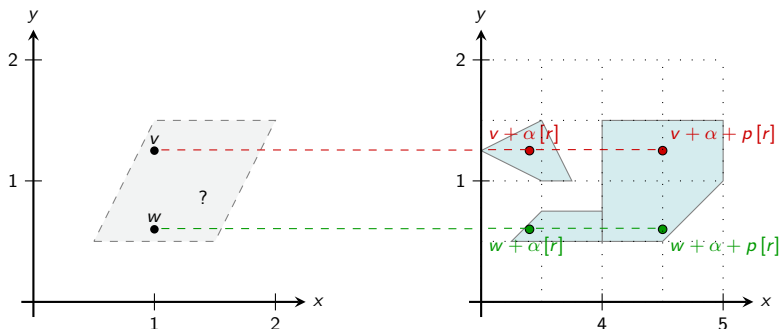
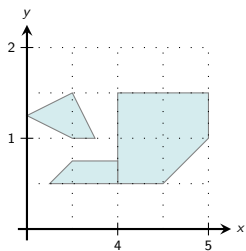


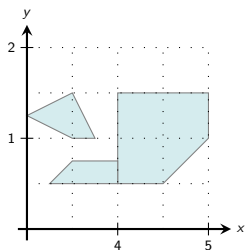
Figure:  $S_{i-1}(\ell', p)$

## Intuition to tackle this issue: good representation of polyhedra

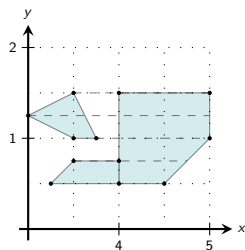


(a) Original representation

## Intuition to tackle this issue: good representation of polyhedra

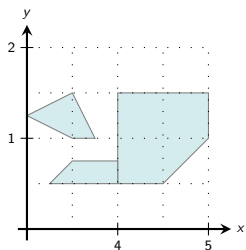


(a) Original representation

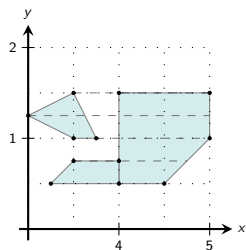


(b) Partition with respect to the vertices

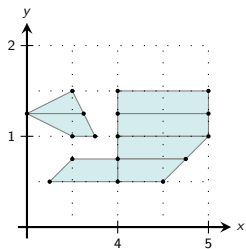
# Intuition to tackle this issue: good representation of polyhedra



(a) Original representation

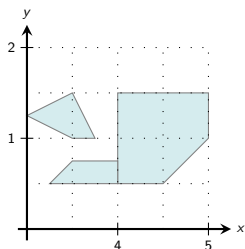


(b) Partition with respect to the vertices

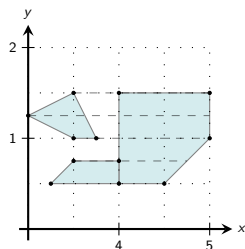


(c) 'sliced' representation of  $S_{i-1}(\ell', \rho)$

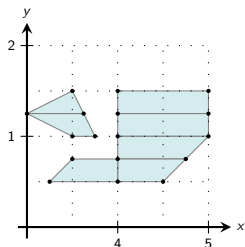
# Intuition to tackle this issue: good representation of polyhedra



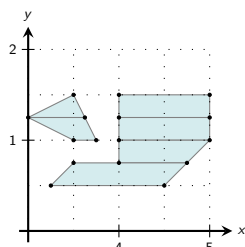
(a) Original representation



(b) Partition with respect to the vertices



(c) 'sliced' representation of  $S_{i-1}(\ell', \rho)$



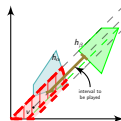
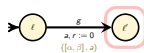
(d) Merging polyhedra in the same 'slice'

## Contribution during this thesis

*Conclusion & Future work*

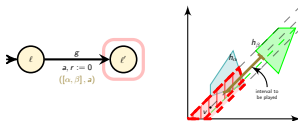
- Symbolic algorithm

- ▶ A doubly exponential algorithm in linear case, non-elementary for acyclic.
- ▶ Restricted to acyclic timed automata (for now).
- ▶ Implementation for linear TA.
- ▶ Exact and symbolic result.



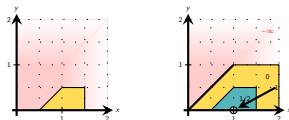
## • Symbolic algorithm

- ▷ A doubly exponential algorithm in linear case, non-elementary for acyclic.
- ▷ Restricted to acyclic timed automata (for now).
- ▷ Implementation for linear TA.
- ▷ Exact and symbolic result.



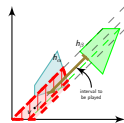
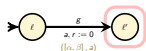
## • Binary and levelled permissiveness

- ▷ A doubly-exponential algorithm
- ▷ Linear w.r.t the number of levels
- ▷ Restricted to linear timed automata (for now)
- ▷ Implementation for linear TA
- ▷ Controlled approximation of the permissiveness



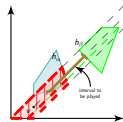
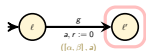
- Symbolic algorithm

- ▷ Tackling the case of cycles.
- ▷ Implementation of acyclic TA.
- ▷ Minimise the overtiling for acyclic TA (Trial-and-error merging).



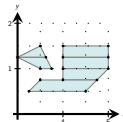
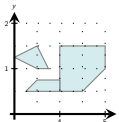
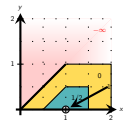
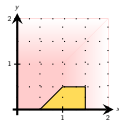
- Symbolic algorithm

- ▷ Tackling the case of cycles.
- ▷ Implementation of acyclic TA.
- ▷ Minimise the overtiling for acyclic TA (Trial-and-error merging).



- Binary and levelled permissiveness

- ▷ Compare runtime results.
- ▷ Acyclic TA: how to represent polyhedra without overtiling?



## Appendix

## Appendix

### *Fourier-Motzkin algorithm*

- The principle of the algorithm

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

*System of linear inequalities (S)*

- The principle of the algorithm

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

Fourier-Motzkin  
Eliminate  $\alpha$   $\rightarrow$

*System of linear inequalities (S)*

- The principle of the algorithm

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

*System of linear inequalities (S)*

Fourier-Motzkin  
Eliminate  $\alpha$

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

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

- The principle of the algorithm

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

*System of linear inequalities (S)*

Fourier-Motzkin  
Eliminate  $\alpha$

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

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

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

- The principle of the algorithm

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

*System of linear inequalities (S)*

Fourier-Motzkin  
Eliminate  $\alpha$

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

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

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

$$0 \leq \alpha$$

$$v + \alpha \models g$$

$$v + \alpha + p \models g$$

$$v + \alpha [y] \in \mathcal{S}_{i-1}(\ell', p)$$

$$v + \alpha + p [y] \in \mathcal{S}_{i-1}(\ell', p)$$

- The principle of the algorithm

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

Fourier-Motzkin  
Eliminate  $\alpha$

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

*System of linear inequalities (S)*

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

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

$$\begin{cases} 0 \leq \alpha \\ v + \alpha \models g \\ v + \alpha + p \models g \\ v + \alpha [y] \in \mathcal{S}_{i-1}(\ell', p) \\ v + \alpha + p [y] \in \mathcal{S}_{i-1}(\ell', p) \end{cases}$$

$$\begin{cases} 0 \leq \alpha \\ 0 \leq x + \alpha \leq 1 \\ 0 \leq y + \alpha \leq 1 \\ 0 \leq x + \alpha + p \leq 1 \\ 0 \leq y + \alpha + p \leq 1 \\ 0 \leq x + \alpha \leq 1 \\ 0 \leq x + \alpha + p \leq 1 \end{cases}$$

- The principle of the algorithm

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

Fourier-Motzkin  
Eliminate  $\alpha$

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

*System of linear inequalities (S)*

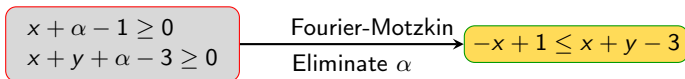
*System of linear inequalities of  
 $\{(x, y) \mid \exists \alpha \text{ s.t. } (x, y, \alpha) \text{ verify } (S)\}$*

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

$$\begin{cases} 0 \leq \alpha \\ v + \alpha \models g \\ v + \alpha + p \models g \\ v + \alpha [y] \in \mathcal{S}_{i-1}(\ell', p) \\ v + \alpha + p [y] \in \mathcal{S}_{i-1}(\ell', p) \end{cases}$$

$$\begin{cases} 0 \leq \alpha \\ 0 \leq x + \alpha \leq 1 \\ 0 \leq y + \alpha \leq 1 \\ 0 \leq x + \alpha + p \leq 1 \\ 0 \leq y + \alpha + p \leq 1 \end{cases}$$

- The principle of the algorithm

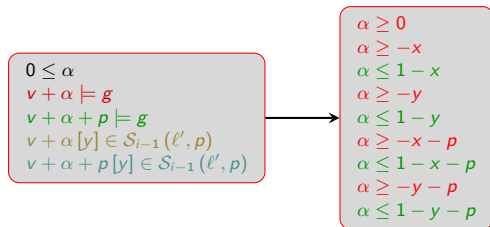


*System of linear inequalities (S)*

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

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

$$A \leq \alpha, \alpha \leq B, 0 \leq C$$



- The principle of the algorithm

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

Fourier-Motzkin  
Eliminate  $\alpha$

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

*System of linear inequalities (S)*

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

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

$$A \leq \alpha, \alpha \leq B, 0 \leq C$$

$$\begin{cases} 0 \leq \alpha \\ v + \alpha \models g \\ v + \alpha + p \models g \\ v + \alpha[y] \in \mathcal{S}_{i-1}(\ell', p) \\ v + \alpha + p[y] \in \mathcal{S}_{i-1}(\ell', p) \end{cases}$$

$$\begin{cases} \alpha \geq 0 \\ \alpha \geq -x \\ \alpha \leq 1 - x \\ \alpha \geq -y \\ \alpha \leq 1 - y \\ \alpha \geq -x - p \\ \alpha \leq 1 - x - p \\ \alpha \geq -y - p \\ \alpha \leq 1 - y - p \end{cases}$$

$$\begin{aligned} \max(A) &\leq \min(B) \\ 0 &\leq C \end{aligned}$$

$$\begin{cases} 0 \leq x \leq 1 - p \\ 0 \leq y \leq 1 - p \end{cases}$$

## Appendix

*Detail of the algorithm to compute the permissiveness<sup>2</sup>*

---

<sup>2</sup>CJMM20.

- Goal: find the  $\alpha$  and  $\beta$  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  $\alpha$  and  $\beta$  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  $\alpha$  and  $\beta$  that maximises:

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

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

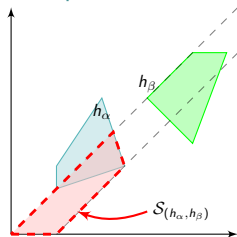
## Steps of the algorithm (linear timed automata)

- Goal: find the  $\alpha$  and  $\beta$  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  $(h_\alpha, h_\beta)$



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

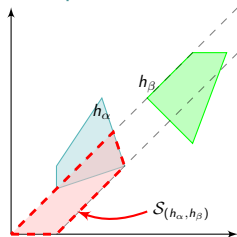
## Steps of the algorithm (linear timed automata)

- Goal: find the  $\alpha$  and  $\beta$  that maximises:

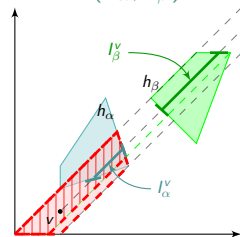
$$\min \left( |\beta - \alpha|, \mathcal{P}_{i-1}(\ell', v + \alpha[r]), \mathcal{P}_{i-1}(\ell', v + \beta[r]) \right)$$

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

- Steps: for each couple of cells  $(h_\alpha, h_\beta)$



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



(b) Step 2: compute the possible  $\alpha$  and  $\beta$

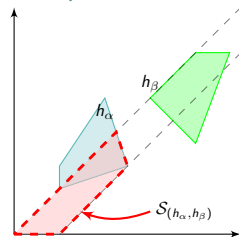
# Steps of the algorithm (linear timed automata)

- Goal: find the  $\alpha$  and  $\beta$  that maximises:

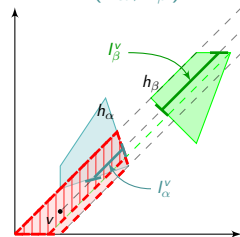
$$\min \left( |\beta - \alpha|, \mathcal{P}_{i-1}(\ell', v + \alpha[r]), \mathcal{P}_{i-1}(\ell', v + \beta[r]) \right)$$

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

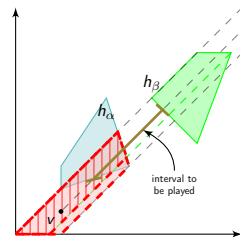
- Steps: for each couple of cells  $(h_\alpha, h_\beta)$



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

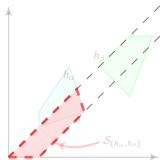


(b) Step 2: compute the possible  $\alpha$  and  $\beta$

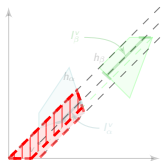


(c) Step 3: compute the optimal  $\alpha$  and  $\beta$

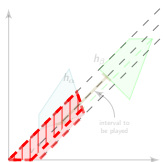
# Example



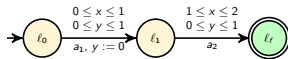
(a) Step 1

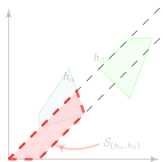


(b) Step 2

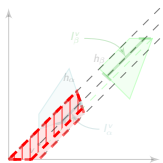


6 / 11 (c) Step 3

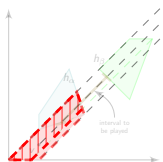




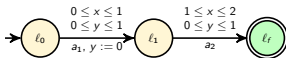
(a) Step 1



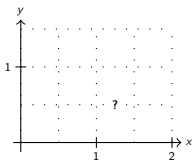
(b) Step 2



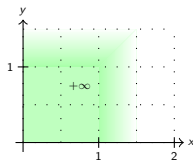
6 / 11 (c) Step 3



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

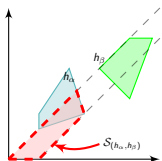


(a) Permissiveness on  $l_1$

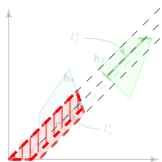


(b) Permissiveness on  $l_f$

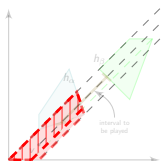
Fixing the cells of arrival of the successors  $h_{\alpha}$  and  $h_{\beta}$ :  $\mathbb{R}^2$



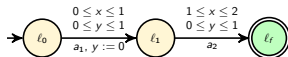
(a) Step 1



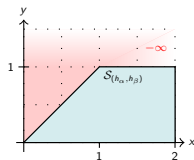
(b) Step 2



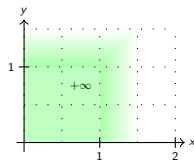
6 / 11 (c) Step 3



- Permissiveness on  $\ell_1$ : maximise  $\min(\beta - \alpha, \mathcal{P}_0(\ell_f, v + \alpha), \mathcal{P}_0(\ell_f, v + \beta))$

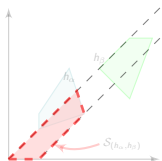


(a) Permissiveness on  $\ell_1$

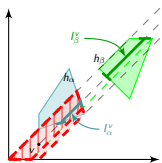


(b) Permissiveness on  $\ell_f$

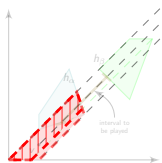
Step 1: computing  $\mathcal{S}(h_\alpha, h_\beta)$  (Fourier-Motzkin algorithm)



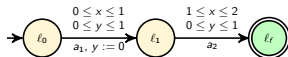
(a) Step 1



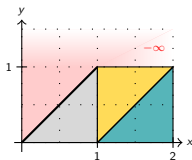
(b) Step 2



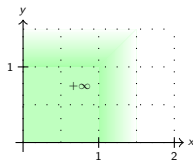
6 / 11 (c) Step 3



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



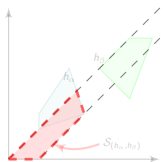
(a) Permissiveness on  $l_1$



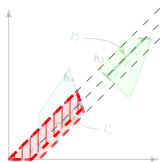
(b) Permissiveness on  $l_f$

Step 2: computing the intervals of  $\alpha$  and  $\beta$  (Fourier-Motzkin algorithm)

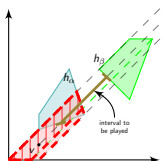
$$I_{\alpha}^V = I_{\beta}^V = [\max(0, 1 - x), \min(2 - x, 1 - y)]$$



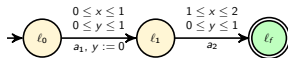
(a) Step 1



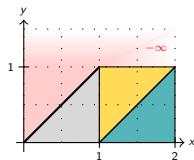
(b) Step 2



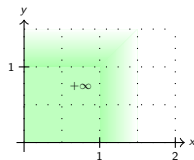
6 / 11 (c) Step 3



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

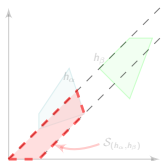


(a) Permissiveness on  $l_1$

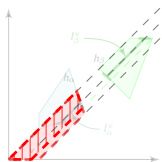


(b) Permissiveness on  $l_f$

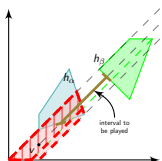
Step 3: computing the optimal  $\alpha$  and  $\beta$ , s.t  $\alpha \leq \beta$ , that maximises...



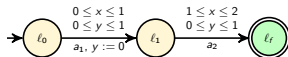
(a) Step 1



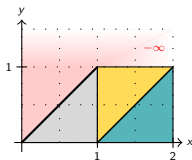
(b) Step 2



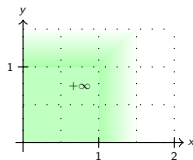
6 / 11 (c) Step 3



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



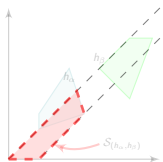
(a) Permissiveness on  $l_1$



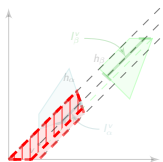
(b) Permissiveness on  $l_f$

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

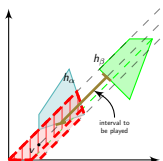
$$\min(\beta - \alpha, \mathcal{P}_{i-1}(l', v + \alpha[r]), \mathcal{P}_{i-1}(l', v + \beta[r]))$$



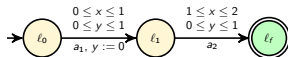
(a) Step 1



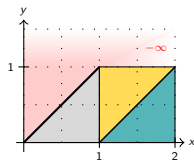
(b) Step 2



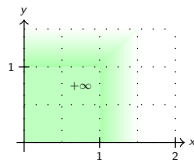
6 / 11 (c) Step 3



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



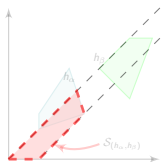
(a) Permissiveness on  $l_1$



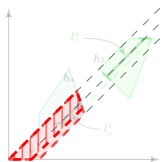
(b) Permissiveness on  $l_f$

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

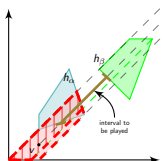
$$\min(\beta - \alpha, +\infty, +\infty)$$



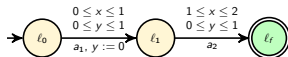
(a) Step 1



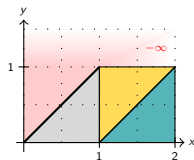
(b) Step 2



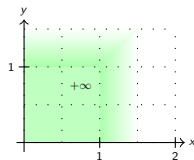
6 / 11 (c) Step 3



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



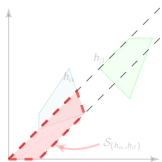
(a) Permissiveness on  $l_1$



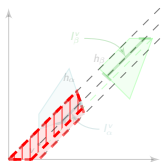
(b) Permissiveness on  $l_f$

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

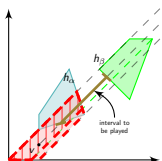
$$\min(\beta - \alpha, +\infty, +\infty) \text{ (Technical lemma)}$$



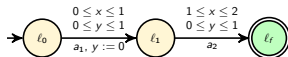
(a) Step 1



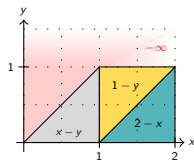
(b) Step 2



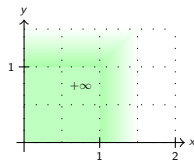
6 / 11 (c) Step 3



- 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  $l_1$

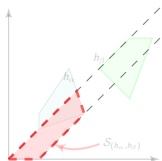


(b) Permissiveness on  $l_f$

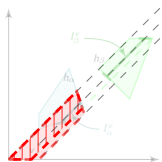
**Step 3:** computing the optimal  $\alpha$  and  $\beta$ , s.t  $\alpha \leq \beta$ , that maximises...

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

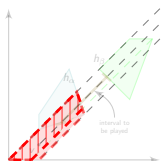
$$\alpha^* = \max(0, 1 - x), \beta^* = \min(2 - x, 1 - y)$$



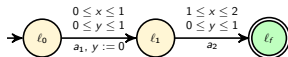
(a) Step 1



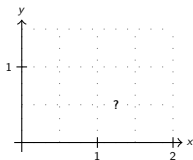
(b) Step 2



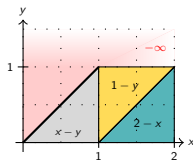
6 / 11 (c) Step 3




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

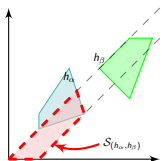


(a) Permissiveness on  $l_0$

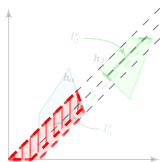


(b) Permissiveness on  $l_1$

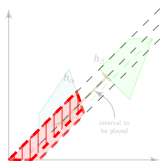
Let us fix  $h_\alpha = h_\beta =$  



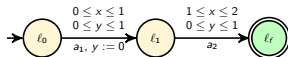
(a) Step 1



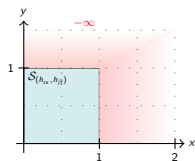
(b) Step 2



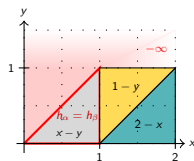
6 / 11 (c) Step 3



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



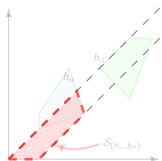
(a) Permissiveness on  $l_0$



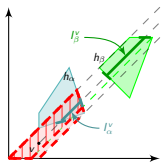
(b) Permissiveness on  $l_1$

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

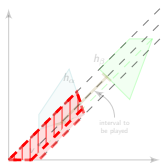
(Fourier-Motzkin algorithm)



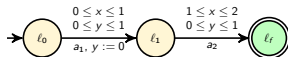
(a) Step 1



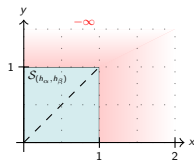
(b) Step 2



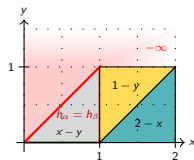
(c) Step 3



- Permissiveness on  $\ell_0$ : maximise  $\min(\beta - \alpha, \mathcal{P}_1(\ell_1, v + \alpha), \mathcal{P}_1(\ell_1, v + \beta))$



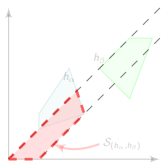
(a) Permissiveness on  $\ell_0$



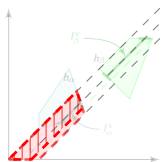
(b) Permissiveness on  $\ell_1$

Step 2: Computing the set of possible  $\alpha$  and  $\beta$  (Fourier-Motzkin algorithm):

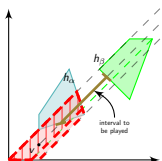
$$I_\alpha^V = I_\beta^V = [0, \min(1 - x, 1 - y)]$$



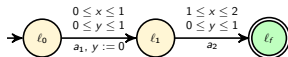
(a) Step 1



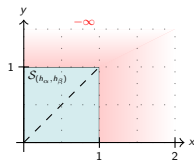
(b) Step 2



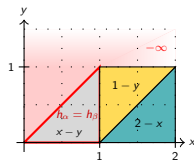
(c) Step 3



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



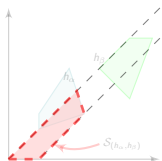
(a) Permissiveness on  $l_0$



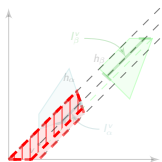
(b) Permissiveness on  $l_1$

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

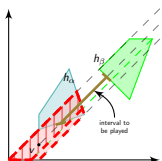
$$\min(\beta - \alpha, 1 \cdot \alpha + x, 1 \cdot \beta + x) \quad \text{(Technical lemma)}$$



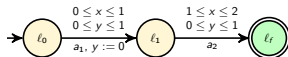
(a) Step 1



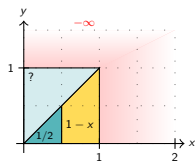
(b) Step 2



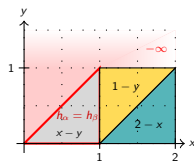
6 / 11 (c) Step 3



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



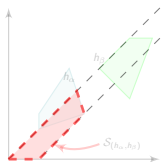
(a) Permissiveness on  $l_0$



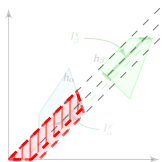
(b) Permissiveness on  $l_1$

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

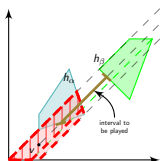
$$\min(\beta - \alpha, 1 \cdot \alpha + x, 1 \cdot \beta + x) \quad \text{(Technical lemma)}$$



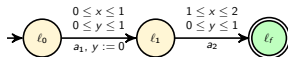
(a) Step 1



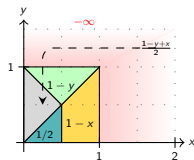
(b) Step 2



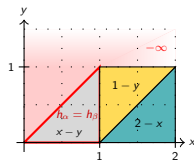
(c) Step 3



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



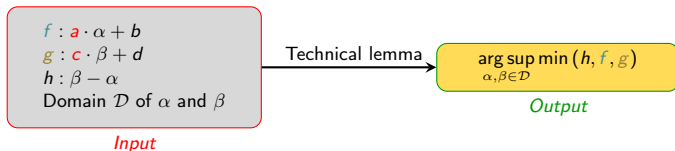
(a) Permissiveness on  $l_0$

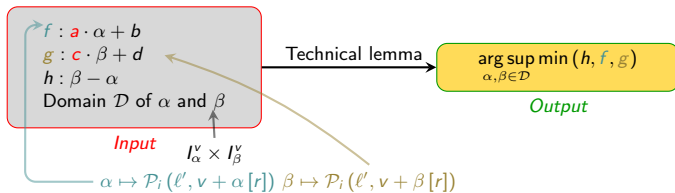


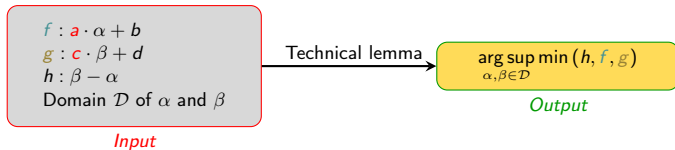
(b) Permissiveness on  $l_1$

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

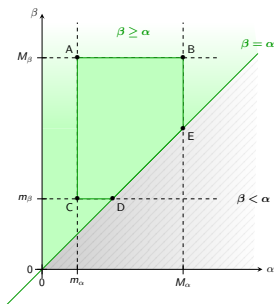
$$\min(\beta - \alpha, 1 \cdot \alpha + x, 1 \cdot \beta + x) \quad \text{(Technical lemma)}$$



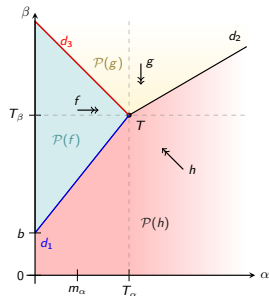




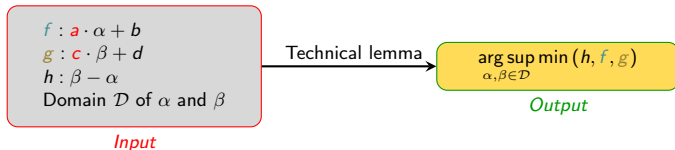
- When  $a > 0$  and  $c < 0$ :



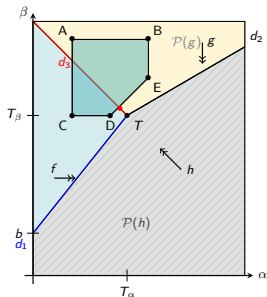
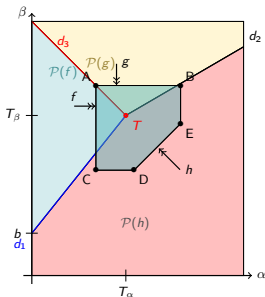
(a) Domain  $\mathcal{D}$



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



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

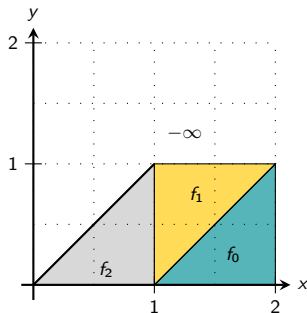


- ▶ Redundancy in the technical lemma

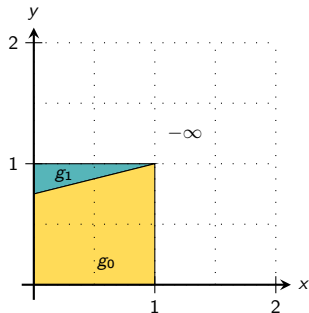
- ▷ Redundancy in the technical lemma
- ▷ Maximisation/Minimisation (when comparing candidate permissiveness functions)

# The causes of the overtiling

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



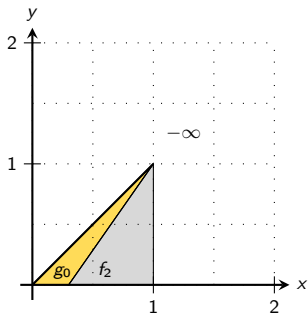
(a)  $f: f_0(x, y) = 2 - x$ ,  $f_1(x, y) = 1 - y$ ,  
 $f_2(x, y) = x - y$



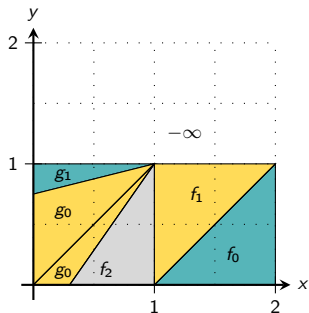
(b)  $g: g_0(x, y) = (1 - x) / 2$ ,  $g_1(x, y) = 1 - y$

# The causes of the overtiling

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



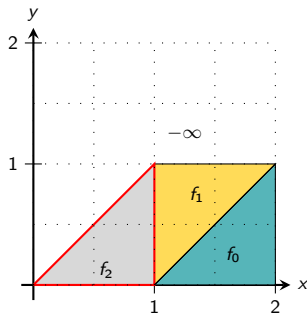
(a) Wrong result



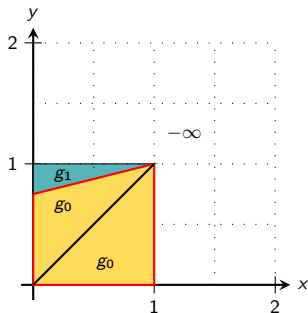
(b) Correct overtiled result  $\max(f, g)$

# 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  $g$



## Appendix

*Computing the union of several polyhedra*

- Case of two polyhedra

Solved in 2001 by Bemporad, Fukuda and D. Torrisi<sup>3</sup>:

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

---

<sup>3</sup>BemporadFT01.

# Computing the union of several polyhedra

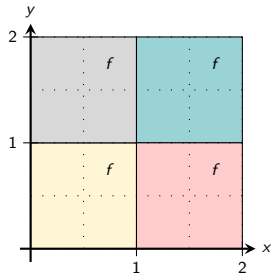
- Case of two polyhedra

Solved in 2001 by Bemporad, Fukuda and D. Torrisi<sup>3</sup>:

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

- Case of several ( $\geq 3$ ) polyhedra

Open problem...



---

<sup>3</sup>BemporadFT01.