Robustness of timed automata:
computing the maximally-permissive strategies

Emily Clement
Supervisors:
Thierry Jéron
Nicolas Markey
David Mentré

1 IRISA, Inria & CNRS & Univ. Rennes, France
2 Mitsubishi Electric R&D Centre Europe – Rennes, France: MERCE

March 11 2022
Why verifying real-time systems?
Why verifying real-time systems?

Real-time systems

Why verifying RTS?
Why verifying real-time systems?

Real-time systems

Life

Why verifying RTS?

2 / 36

Emily Clement

Robustness of timed automata
Why verifying real-time systems?

Life

Cost

Why verifying RTS?
Why verifying real-time systems?

Real-time systems

Life

Cost

Why verifying RTS?

Mass production
Why verifying real-time systems?

Real-time systems

Life

Cost

Why verifying RTS?

Time to market

Mass production

Life

Cost

Why verifying RTS?
Verifying real-time systems: model-checking

Model $\mathcal{M}$

Property $\varphi$

Model-checking algorithm $\mathcal{M} \models \varphi$?

Satisfied

Counter-example
An abstract model

**Timed automata**, timed games, weighted timed automata, timed Petri nets...
Real-time systems modelling: timed automata

An abstract model

Timed automata, Timed games, weighted timed automata, timed petri nets...
An abstract model

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

- Example: Scheduling system

```
ℓ0 → ℓ1
  ↓      ↓
   x := 0 2 ≤ x ≤ 3

ℓ1 → ℓ2
  ↓      ↓
   b     0 ≤ x ≤ 5

ℓ2 → ℓ3
  ↓      ↓
   y := 0 2 ≤ y ≤ 3

ℓ3 → ℓ0
  ↓      ↓
   b     3 ≤ y

ℓf  
```

Initial location
Goal location
Action
Clocks
Guard
Reset

Acyclic timed automata:

Linear timed automata:

Can we execute...
Real-time systems modelling: timed automata

An abstract model

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

- Example: Scheduling system

![Diagram of Timed Automaton]

- Initial location: $\ell_0$
- Goal location: $\ell_f$
- Locations: $\ell_0, \ell_1, \ell_2, \ell_3, \ell_f$
- Actions: $a$, $b$, $c$
- Guards:
  - $2 \leq x \leq 3$
  - $0 \leq x \leq 5$
  - $3 \leq y$
- Resets:
  - $x := 0$
  - $y := 0$

Can we execute...
Real-time systems modelling: timed automata

An abstract model

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

Example: Scheduling system

Initial location

Location

Guard

Clocks

Action

Reset

Goal location

Acyclic timed automata:

Linear timed automata:
Real-time systems modelling: timed automata

An abstract model

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

- Example: Scheduling system

![Diagram of a timed automaton with transitions and clock assignments]

- Can we execute...
Real-time systems modelling: timed automata

An abstract model

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

• Example: Scheduling system

\[
\begin{align*}
\ell_0 & \xrightarrow{a} \ell_1 \quad \text{subject to} & 2 \leq x \leq 3 \\
\ell_0 & \xrightarrow{b} \ell_3 \quad \text{subject to} & 0 \leq x \leq 5 \\
\ell_1 & \xrightarrow{a, x := 0} \ell_1, (0, \delta_0) \quad \text{subject to} & 2 \leq y \leq 3 \\
\ell_1 & \xrightarrow{b, y := 0} \ell_2, (\delta_1, \delta_0 + \delta_1) \quad \text{subject to} & 3 \leq y \\
\ell_2 & \xrightarrow{c} \ell_f, (\delta_1 + \delta_2, \delta_2)
\end{align*}
\]

Can we execute...abc?

\[
\begin{align*}
\ell_0, (0, 0) & \xrightarrow{\delta_0} \ell_0, (\delta_0, \delta_0) \xrightarrow{a, x := 0} \ell_1, (0, \delta_0) \xrightarrow{\delta_1} \ell_1, (\delta_1, \delta_0 + \delta_1) \xrightarrow{b, y := 0} \\
\ell_2, (\delta_1, 0) & \xrightarrow{\delta_2} \ell_2, (\delta_1 + \delta_2, \delta_2) \xrightarrow{c} \ell_f, (\delta_1 + \delta_2, \delta_2)
\end{align*}
\]

Acyclic timed automata:

Linear timed automata:
**Real-time systems modelling: timed automata**

**An abstract model**

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

- **Example:** Scheduling system

\[\begin{align*}
\ell_0 & \xrightarrow{a} \ell_1 & 2 \leq x \leq 3 \\
& \xrightarrow{b} \ell_3 & 0 \leq x \leq 5 \\
& \xrightarrow{c} \ell_f & 3 \leq y
\end{align*}\]

Can we execute... \(abc\)?

\[\begin{align*}
\ell_0, (0, 0) & \xrightarrow{\delta_0=1} \ell_0, (1, 1) \\
& \xrightarrow{a,x:=0} \ell_1, (0, 1) \\
& \xrightarrow{b,y:=0} \ell_2, (2, 0) \\
& \xrightarrow{c} \ell_f, (5, 3)
\end{align*}\]
Real-time systems modelling: timed automata

**An abstract model**

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

- Example: Scheduling system

![Diagram of a timed automaton]

- Acyclic timed automata:
- Linear timed automata:

Can we execute... *bac*?

\[
\begin{align*}
\ell_0, (0, 0) & \xrightarrow{\delta_0} \ell_0, (\delta_0, 0) \\
& \xrightarrow{b, y:=0} \ell_3, (\delta_0, 0) \\
& \xrightarrow{\delta_1} \ell_3, (\delta_1 + \delta_0, \delta_1) \\
& \xrightarrow{a, x:=0} \\
\ell_2, (0, \delta_1) & \xrightarrow{\delta_2} \ell_2, (\delta_2, \delta_1 + \delta_2) \\
& \xrightarrow{c} \ell_f, (\delta_2, \delta_1 + \delta_2)
\end{align*}
\]
Real-time systems modelling: timed automata

An abstract model

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

- Example: Scheduling system

▶ Can we execute... *bac*? ✓

▶ \( \ell_0, (0, 0) \xrightarrow{a=10} \ell_0, (10, 10) \xrightarrow{b, y:=0} \ell_3, (10, 0) \xrightarrow{c=3} \ell_3, (13, 3) \xrightarrow{a, x:=0} \)

\( \ell_2, (0, 3) \xrightarrow{\delta_2=4} \ell_2, (4, 7) \xrightarrow{c} \ell_f, (4, 7) \)
Real-time systems modelling: timed automata

An abstract model

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

- Example: Scheduling system

\[ \begin{align*}
  x &:= 0 & 2 \leq x \leq 3 \\
  y &:= 0 & 0 \leq x \leq 5 \\
  3 \leq y &
\end{align*} \]

\[ \begin{align*}
  a &\quad \rightarrow \quad \ell_1 \\
  b &\quad \rightarrow \quad \ell_2 \\
  c &\quad \rightarrow \quad \ell_f
\end{align*} \]

Can we execute... \( bac? \)

\[ \begin{align*}
  \ell_0, (0, 0) &\overset{\delta_0=10}{\rightarrow} \ell_0, (10, 10) & b, y:=0 &\quad \ell_3, (10, 0) &\overset{\delta_1=2}{\rightarrow} \ell_3, (12, 2) & a, x:=0 \\
  \ell_2, (0, 2) &\overset{\delta_2=2}{\rightarrow} \ell_2, (2, 2) & c &\quad \ell_f, (2, 2)
\end{align*} \]
Assumptions made on timed automata

- Synchronised clocks

Timed automata
Assumptions made on timed automata

- Synchronised clocks

- Exact guards

Timed automata
Assumptions made on timed automata

- Synchronised clocks
- Exact guards
- Exact delays
Example of perturbed semantics: clock drifting

- **Timed automaton \( \mathcal{A} \)**

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

---

\[ \ell_0 \xrightarrow{a, x := 0} \ell_1 \xrightarrow{2 \leq x \leq 3} \ell_2 \xrightarrow{0 \leq x \leq 5} \ell_f \]
Example of perturbed semantics: clock drifting

- Timed automaton $\mathcal{A}$

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

---

Example of perturbed semantics: clock drifting

- **Timed automaton** $\mathcal{A}$

\[
\begin{align*}
\ell_0 & \xrightarrow{a, x := 0} \ell_1 & 2 \leq x \leq 3 & \xrightarrow{b, y := 0} \ell_2 & 0 \leq x \leq 5 \\
& & & & 3 \leq y \\
& & & & c \\
\ell_f & 
\end{align*}
\]

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

---

Example of perturbed semantics: clock drifting

- Timed automaton $\mathcal{A}$

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

---

Example of perturbed semantics: clock drifting\(^1\)

- **Timed automaton** \(\mathcal{A}\)

  \[
  \ell_0 \xrightarrow{a, x := 0} \ell_1 \quad \ell_1 \xrightarrow{2 \leq x \leq 3} \ell_2 \quad \ell_2 \xrightarrow{0 \leq x \leq 5} \ell_f
  \]
  \[
  \ell_0 \xrightarrow{b, y := 0} \ell_1 \quad \ell_1 \xrightarrow{3 \leq y} \ell_f
  \]

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

Example of perturbed semantics: guard enlargement

- **Timed automaton** $A$:

![Timed automaton diagram]

$\ell_0$: $0 \leq x \leq 1$ \hspace{1cm} $0 \leq y \leq 1$

$\ell_1$: $1 \leq x \leq 2$

$\ell_f$: $0 \leq y \leq 1$

$\ell_0 \xrightarrow{a_1, y := 0} \ell_1 \xrightarrow{a_2} \ell_f$

---

Example of perturbed semantics: guard enlargement

- Timed automaton $A$:

![Diagram of timed automaton]

Example of perturbed semantics: guard enlargement

- Timed automaton $A$:

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

---


Emily Clement

Robustness of timed automata
Example of perturbed semantics: guard enlargement

- Timed automaton $A$:

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

---

Example of perturbed semantics: delay perturbation

- Timed automaton $A$:

- Run with delay perturbations of at most $\delta = 0.2$
Example of perturbed semantics: delay perturbation

- Timed automaton $\mathcal{A}$:

- Run with delay perturbations of at most $\delta = 0.2$
Example of perturbed semantics: delay perturbation

- **Timed automaton** $\mathcal{A}$:

  ![Timed automaton diagram](image)

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

---


Emily Clement Robustness of timed automata
Example of perturbed semantics: delay perturbation

- Timed automaton $A$:

$$0 \leq x \leq 1 \quad 0 \leq y \leq 1 \quad a_1, y := 0$$

$$1 \leq x \leq 2 \quad 0 \leq y \leq 1 \quad a_2$$

- Run with delay perturbations of at most $\delta = 0.2$
Example of perturbed semantics: delay perturbation

- Timed automaton $A$:

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

---

Example of perturbed semantics: delay perturbation

- **Timed automaton \( \mathcal{A} \):**

  \[
  \ell_0 \xrightarrow{a_1, y := 0} \ell_1 \xrightarrow{a_2} \ell_f
  \]

  - \( 0 \leq x \leq 1 \)
  - \( 0 \leq y \leq 1 \)
  - \( 1 \leq x \leq 2 \)
  - \( 0 \leq y \leq 1 \)

- **Run with delay perturbations of at most \( \delta = 0.2 \)**
Example of perturbed semantics: delay perturbation

- Timed automaton $A$:

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

---


Emily Clement

Robustness of timed automata
Principle of robustness

1. Perturbed semantics
2. Model $\mathcal{M}$

Property $\varphi$

Robustness analysis

$\mathcal{M} \models \varphi$?
Principle of robustness

1. Perturbed semantics
2. Model $\mathcal{M}$

Property $\varphi$

Robustness analysis

$\exists \delta > 0 \text{ s.t. } \mathcal{M}^\delta \models \varphi$?

The permissiveness problem

$\mathcal{M} \models \varphi$?
Principle of robustness

1. Perturbed semantics
2. Model $M$

$\exists \delta > 0 \text{ s.t. } M^\delta \models \varphi$

The permissiveness problem

$\sup \left\{ \delta \geq 0 \mid M^\delta \models \varphi \right\}$
Principle of robustness

1. Perturbed semantics
2. Model $\mathcal{M}$
   - Property $\varphi$

Robustness analysis

$\exists \delta > 0 \text{ s.t. } \mathcal{M}^\delta \models \varphi$?

The permissiveness problem

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

The fixed permissiveness problem

$\exists \delta > 0 \text{ s.t. } \mathcal{M}^\delta \models \varphi$?

Fix $\delta > 0$, $\mathcal{M}^\delta \models \varphi$?
Principle of robustness

1. Perturbed semantics

Model $\mathcal{M}$

Property $\varphi$

Robustness analysis

$\mathcal{M}^\delta \models \varphi$?

$\exists \delta > 0 \text{ s.t. } \mathcal{M}^\delta \models \varphi$?

The permissiveness problem

$\sup \{ \delta \geq 0 | \mathcal{M}^\delta \models \varphi \}$

The maximal-permissiveness problem

Fix $\delta > 0$, $\mathcal{M}^\delta \models \varphi$?

The fixed permissiveness problem
Principle of robustness

1. Delay perturbed semantics
2. Timed automata

Reachability

Robustness analysis

\[ M^\delta | = \varphi ? \]

\[ \exists \delta > 0 \text{ s.t. } M^\delta | = \varphi ? \]

The permissiveness problem

Fix \( \delta > 0 \), \( M^\delta | = \varphi ? \)

The fixed permissiveness problem

\[ \sup \left\{ \delta \geq 0 \mid M^\delta | = \varphi \right\} \]

The maximal-permissiveness problem

\[ M^\delta | = \varphi ? \]
Our permissive semantics: a turn-based game

Player maximises the permissiveness

Opponent minimises the permissiveness

\[(0, 0) \quad 0 \leq x, y \leq 2 \quad a, y := 0 \quad 2 \leq x \quad 0 \leq y \leq 2 \quad b\]
Our permissive semantics: a turn-based game

Player

Opponent

Permissiveness = 2
δ₀ = 0

I₀ = [0, 2]
Permissiveness = 2
δ₀ = 0
Our permissive semantics: a turn-based game

Player

Opponent

Let $I_0 = [0, 2]$ and $I_1 = [2, 2]$. The permissiveness of the runs is $\delta_0 = 0$ and $\delta_1 = 2$. The player maximizes the permissiveness, while the opponent minimizes it.
Our permissive semantics: a turn-based game

Player: **maximises** the permissiveness

Opponent: **minimises** the permissiveness

![Diagram showing a turn-based game with states and transitions]

- \( (0,0) \) with \( 0 \leq x, y \leq 2 \)
- \( a, y := 0 \)
- \( l_0 = [0, 2] \)
- Permissiveness = 2
- \( \delta_0 = 0 \)

- \( (0,0) \) with \( 2 \leq x \)
- \( 0 \leq y \leq 2 \)
- \( l_1 = [2, 2] \)
- Permissiveness = 0
- \( \delta_1 = 2 \)

**Permissiveness of the run**: \( \min (|l_0|, |l_1|) = \min (2, 0) = 0 \)
Our permissive semantics: a turn-based game

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

Player's turn:
- Initial state: $(0,0)$
- Constraints: $0 \leq x, y \leq 2$
- Transition: $a, y := 0$
- New state: $(1,0)$

Opponent's turn:
- Constraints: $2 \leq x$
- New state: $(2,2)$

Permissiveness of the run:
- $l_0 = [1, 2]$
- Permissiveness $= 1$
- $\delta_0 = 1$
- $l_1 = [1, 2]$
- Permissiveness $= 1$
- $\delta_1 = 1$

$\min(|l_0|, |l_1|) = \min(1, 1) = 1$
Our permissive semantics: a turn-based game

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

\[
\begin{align*}
\delta_0 &= 1 \\
\text{Permissiveness} &= 1 \\
I_0 &= [1, 2]
\end{align*}
\]

\[
\begin{align*}
\delta_1 &= 1 \\
\text{Permissiveness} &= 1 \\
I_1 &= [1, 2]
\end{align*}
\]

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

- **Opponent**: worst-case environment
- **Our goal**: compute the player best strategy, whatever the opponent decides
Permissiveness: an infinite number of choices

\[ \ell_0 \xrightarrow{a,x := 0} \ell_1 \xrightarrow{b,y := 0} \ell_2 \xrightarrow{c} \ell_f \]

Rest of the automaton...

\[ (\ell_1, v) \]
Permissiveness: an infinite number of choices

\[ \ell_0 \xrightarrow{a,x \colonequals 0} \ell_1 \xrightarrow{2 \leq x \leq 3} \ell_2 \xrightarrow{x \leq 5, y \geq 3} \ell_f \]

Choice of interval \( I \), permissiveness = \(|I|\)

Current action/permissiveness

Player

Choice of delay \( \delta \in I \)

All future permissiveness

Opponent

Rest of the automaton...

Infinite choices...
Permissiveness: an infinite number of choices

\[ \ell_0 \xrightarrow{a, x := 0} \ell_1 \xrightarrow{2 \leq x \leq 3} \ell_2 \xrightarrow{x \leq 5, y \geq 3} \ell_f \]

\[ a, x := 0 \quad b, y := 0 \]

Choice of interval \( I \), permissiveness = \(|I|\)

Current action/permissiveness

Choice of delay \( \delta \in I \)

Player

Opponent

Choice of delay \( \delta \in I \)

All future permissiveness

Rest of the automaton

Infinite choices

\((\ell_2, v + 2[r])\)

\((\ell_2, v + 2.9[r])\)

\((\ell_2, v + 3[r])\)

\((\ell_2, v + 3[r])\)
Goal of this thesis: compute the maximal permissiveness

- Permissive semantics: a turn-based game
  - **Player**: Interval & action: \((I, a)\)
  - **Opponent**: delay \(\delta \in I\)

- Permissiveness
  - **Player** maximises
  - **Opponent** minimises
  - \(\min (|I_0|, |I_1|, |I_2|, \cdots)\)

### Diagram

- **Delay perturbed semantics**
- **Timed automata**
- **Robustness analysis**
  - \(M \models \varphi?\)
  - **Permissiveness of the initial configuration**
  - **Reachability**

**What is the maximal allowed perturbation?**
Goal of this thesis: compute the maximal permissiveness

- Permissive semantics: a turn-based game
  - Player: Interval & action: \((I, a)\)
  - Opponent: delay \(\delta \in I\)

- Permissiveness
  - Player: minimises
  - Opponent: maximises
  - Minimises: \(\min (|I_0|, |I_1|, |I_2|, \cdots)\)

1. **1st contribution**
   - Symbolic computation of the permissiveness & the strategy of the player for acyclic TA and TG
   - Implemented

2. **2nd contribution**
   - Symbolic computation of the permissiveness for acyclic TA, with controlled approximate results

3. **3rd contribution**
   - Numerical computation of the permissiveness for acyclic TA, with approximate results

**Robustness analysis**

- **Reachability**

- **Permissiveness analysis**
  - What is the maximal allowed perturbation?

**Timed automata**

- **Reachability**

**Permissiveness of the initial configuration**

- **Permissiveness**
Goal of this thesis: compute the maximal permissiveness

- **Permissive semantics**: a turn-based game
  - Player: Interval & action: $(I, a)$
  - Opponent: delay $\delta \in I$

- **Permissiveness**
  - Player: maximises
  - Opponent: minimises
  - $\min(|I_0|, |I_1|, |I_2|, \cdots)$

---

**1st contribution**
- Symbolic computation of the permissiveness & the strategy of the player for acyclic TA and TG
- Implemented

**2nd contribution**
- Symbolic computation of the permissiveness for acyclic TA, with controlled approximate results
Goal of this thesis: compute the maximal permissiveness

- Permissive semantics: a turn-based game
  
  **Player**: Interval & action: \((I, a)\)
  
  **Opponent**: delay \(\delta \in I\)

- Permissiveness

\[
\min (|I_0|, |I_1|, |I_2|, \cdots)
\]

**1st contribution**

*Implemented*

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

**2nd contribution**

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

**3rd contribution**

*Implemented*

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

What is the maximal allowed perturbation?

Robustness analysis

Reachability

Delay perturbed semantics

Timed automata

Permissiveness of the initial configuration

\[M \models \varphi?\]
First contribution: an exact symbolic computation\textsuperscript{4}

Strategy of the player: maximises

\[
\min \left( |\beta - \alpha|, \inf_{\delta \in [\alpha, \beta]} P_{i-1}(\ell', v + \delta [r]) \right)
\]

Permissiveness of the current mode

Permissiveness of the successor

Opponent's best strategy

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

\[
\inf_{\delta \in [\alpha, \beta]} P_{i-1}(\ell', v + \delta [r]) = \min \left( P_{i-1}(\ell', v + \alpha [r]), P_{i-1}(\ell', v + \beta [r]) \right)
\]
**Strategy of the player:** maximises

\[
\min \left( |\beta - \alpha|, \inf_{\delta \in [\alpha, \beta]} P_{i-1}(\ell', v + \delta [r]) \right)
\]

- **Opponent strategy lemma** (linear case):
  
  player : \(([\alpha, \beta], a)\) \[\text{Opponent's best strategy}\] \(\alpha\) or \(\beta\)

\[
\inf_{\delta \in [\alpha, \beta]} P_{i-1}(\ell', v + \delta [r]) = \min \left( P_{i-1}(\ell', v + \alpha [r]), P_{i-1}(\ell', v + \beta [r]) \right)
\]
(1st contribution) The permissiveness function

- A recursive function

\[ P_i(\ell, v) = \sup_{([\alpha, \beta], a) \in \text{p-moves}(\ell, v)} \left( \min \left( \beta - \alpha, \inf_{\delta \in [\alpha, \beta]} P_{i-1}(\ell', v + \delta[r]) \right) \right) \]
The permissiveness function

- A recursive function

\[
P_i(\ell, v) = \sup_{([\alpha, \beta], a) \in p\text{-moves}(\ell, v)} \left( \min \left( \beta - \alpha, \inf_{\delta \in [\alpha, \beta]} P_{i-1}(\ell', v + \alpha \cdot [r]) \right) \right)
\]

For linear timed automata:

\[
P_i(\ell, v) = \sup_{([\alpha, \beta], a) \in p\text{-moves}(\ell, v)} \left( \min \left( \beta - \alpha, P_{i-1}(\ell', v + \alpha \cdot [r]), P_{i-1}(\ell', v + \beta \cdot [r]) \right) \right)
\]

\[
\lim_{i \to +\infty} P_i(\ell, v) = \text{the permissiveness on } (\ell, v)
\]
A recursive function

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

For linear timed automata:

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

\[ \lim_{i \to +\infty} P_i(\ell, v) = \text{the permissiveness on } (\ell, v) \]

limit reached in \( d_\ell \) steps
A recursive function

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

For linear timed automata:

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

\[
\lim_{i \to +\infty} P_i(\ell, v) = \text{the permissiveness on } (\ell, v)
\]

Goal of our algorithm

Compute \(v \mapsto P_{d_\ell}(\ell, v)\) knowing \(v \mapsto P_{d_\ell-1}(\ell', v)\)
• 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:

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

$\nu \mapsto \mathcal{P}_i(l, \nu)$: a 2-Lipschitz piecewise-affine function
(1st contribution) 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 $\beta$ interval to be played

  (c) **Step 3**: compute the optimal $\alpha$ 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 $\beta$
• **Goal:** find the $\alpha$ and $\beta$ that maximises:

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

• $v \mapsto P_i(\ell, v)$: 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 $\beta$
\[
S(h_\alpha, h_\beta)
\]

(a) Step 1

\[
\ell_0 \rightarrow \ell_1 \rightarrow \ell_f
\]

\(0 \leq x \leq 1\)
\(0 \leq y \leq 1\)
\(a_1, y := 0\)
\(1 \leq x \leq 2\)
\(0 \leq y \leq 1\)
\(a_2\)

(b) Step 2

(c) Step 3

interval to be played

\(a_1, y := 0\)

Emily Clement

Robustness of timed automata
(1st contribution) Example

(a) Step 1

(b) Step 2

(c) Step 3

- Permissiveness on $\ell_1$: maximise

$$\min (\beta - \alpha, P_0(\ell_f, v + \alpha), P_0(\ell_f, v + \beta))$$

 Fixing the cells of arrival of the successors $h_\alpha$ and $h_\beta$: $\mathbb{R}^2$
Permissiveness on $\ell_1$: maximise
\[\min (\beta - \alpha, \mathcal{P}_0(\ell_f, \nu + \alpha), \mathcal{P}_0(\ell_f, \nu + \beta))\]

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

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

\[
I_\alpha^y = I_\beta^y = \left[ \max \left( 0, 1 - x \right), \min \left( 2 - x, 1 - y \right) \right]
\]
Step 3: computing the optimal $\alpha$ and $\beta$, s.t $\alpha \leq \beta$, that maximises...

- Permissiveness on $\ell_1$: maximise
$\min (\beta - \alpha, P_0(\ell_f, v + \alpha), P_0(\ell_f, v + \beta))$
Step 1: 

\[ S(h_\alpha, h_\beta) \]

(a) Step 1

Step 2:

\[ h_\beta \quad h_\alpha \]

(b) Step 2

Step 3:

\[ \ell_0 \quad 0 \leq x \leq 1 \]
\[ 0 \leq y \leq 1 \]
\[ a_1, y := 0 \]
\[ \ell_1 \quad 1 \leq x \leq 2 \]
\[ 0 \leq y \leq 1 \]
\[ a_2 \]
\[ \ell_f \]

- **Permissiveness on** \( \ell_1 \): maximise 
  \[ \min (\beta - \alpha, P_0 (\ell_f, v + \alpha), P_0 (\ell_f, v + \beta)) \]

(b) Permissiveness on \( \ell_f \)

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

\[ \min (\beta - \alpha, P_{i-1} (\ell', v + \alpha [r]), P_{i-1} (\ell', v + \beta [r])) \]

(a) Permissiveness on \( \ell_1 \)
Step 1:

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

Step 2:

- Interval to be played

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

\[
\min (\beta - \alpha, +\infty, +\infty)
\]
(1st contribution) Example

Step 1

Step 2

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

\[ \min (\beta - \alpha, +\infty, +\infty) \]  

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

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

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

\[
\alpha^* = \max (0, 1 - x) \quad \beta^* = \min (2 - x, 1 - y)
\]
\( S(h_\alpha, h_\beta) \)

(a) Step 1

(b) Step 2

(c) Step 3

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

Let us fix \( h_\alpha = h_\beta = x - y \)
Step 1: Computing the corresponding entry set $S_{(h_\alpha, h_\beta)}$

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

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

\[ S(h_\alpha, h_\beta) \]

- Permissiveness on \( \ell_0 \): maximise
  \[ \min (\beta - \alpha, P_1(\ell_1, \nu + \alpha), P_1(\ell_1, \nu + \beta)) \]

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

\[ \min (\beta - \alpha, 1 \cdot \alpha + x, 1 \cdot \beta + x) \] (Technical lemma)
Example

(a) Step 1

(b) Step 2

(c) Step 3

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

\[
\min (\beta - \alpha, 1 \cdot \alpha + x, 1 \cdot \beta + x)
\]

(technical lemma)
(1st contribution) Example

Step 1: Given $h_\alpha = h_\beta$, the step involves setting $a_1, y := 0$.

Step 2: The interval to be played is determined.

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

$$\min (\beta - \alpha, 1 \cdot \alpha + x, 1 \cdot \beta + x)$$
(1st contribution) Fourier-Motzkin algorithm

- The principle of the algorithm

\[
\begin{align*}
x + \alpha - 1 & \geq 0 \\
x + y + \alpha - 3 & \geq 0
\end{align*}
\]

*System of linear equations (S)*
The principle of the algorithm

\[ x + \alpha - 1 \geq 0 \]
\[ x + y + \alpha - 3 \geq 0 \]

System of linear equations \((S)\)
The principle of the algorithm

\[ x + \alpha - 1 \geq 0 \]
\[ x + y + \alpha - 3 \geq 0 \]

System of linear equations \((S)\)

Fourier-Motzkin
Eliminate \(\alpha\)

\[ -x + 1 \leq x + y - 3 \]

System of linear equations of
\[ \{(x, y) | \exists \alpha \text{ s.t. } (x, y, \alpha) \text{ verify } (S)\} \]
The principle of the algorithm

\[
\begin{align*}
  x + \alpha - 1 & \geq 0 \\
  x + y + \alpha - 3 & \geq 0
\end{align*}
\]

Eliminate \( \alpha \)

\[
-x + 1 \leq x + y - 3
\]

System of linear equations (\( S \))

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

Computing the entry set \( S_{(h_\alpha, h_\beta)} \): the set of \( (x, y) \in \mathbb{R}_+^2 \text{ s.t. } \exists \alpha, \beta \text{ s.t.} \)
The principle of the algorithm

System of linear equations (S)

\[ \begin{align*}
    x + \alpha - 1 & \geq 0 \\
    x + y + \alpha - 3 & \geq 0 
\end{align*} \]

Fourier-Motzkin

Eliminate \( \alpha \)

\[ -x + 1 \leq x + y - 3 \]

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

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

\[ \begin{align*}
    0 & \leq \alpha \leq \beta \\
    v + \alpha & \models g \\
    v + \beta & \models g \\
    v + \alpha [y] & \in h_\alpha \\
    v + \beta [y] & \in h_\beta
\end{align*} \]
The principle of the algorithm

\[
\begin{align*}
x + \alpha - 1 & \geq 0 \\
x + y + \alpha - 3 & \geq 0
\end{align*}
\]

System of linear equations \((S)\)

\[-x + 1 \leq x + y - 3\]

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

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

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

\[
\begin{align*}
0 & \leq \alpha \leq \beta \\
0 & \leq x + \alpha \leq 1 \\
0 & \leq y + \alpha \leq 1 \\
0 & \leq x + \beta \leq 1 \\
0 & \leq y + \beta \leq 1 \\
0 & \leq x + \alpha \leq 1 \\
0 & \leq x + \beta \leq 1
\end{align*}
\]
(1st contribution) Fourier-Motzkin algorithm

- The principle of the algorithm

\[ \begin{align*}
  x + \alpha - 1 & \geq 0 \\
  x + y + \alpha - 3 & \geq 0
\end{align*} \]

System of linear equations \((S)\)

\[ -x + 1 \leq x + y - 3 \]

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

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

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

\[ \begin{align*}
  0 \leq \alpha \leq \beta \\
  0 \leq x + \alpha & \leq 1 \\
  0 \leq y + \alpha & \leq 1 \\
  0 \leq x + \beta & \leq 1 \\
  0 \leq y + \beta & \leq 1
\end{align*} \]
(1st contribution) Fourier-Motzkin algorithm

- The principle of the algorithm

\[
\begin{align*}
    x + \alpha - 1 & \geq 0 \\
    x + y + \alpha - 3 & \geq 0
\end{align*}
\]

\[
\text{Fourier-Motzkin} \quad \rightarrow \quad \neg x + 1 \leq x + y - 3
\]

*System of linear equations (S)*

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

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

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

\[
A \leq \alpha, \alpha \leq B, 0 \leq C
\]

\[
\begin{align*}
    \alpha & \geq 0 \\
    \alpha & \leq \beta \\
    \alpha & \geq -x \\
    \alpha & \leq 1 - x \\
    \alpha & \geq -y \\
    \alpha & \leq 1 - y \\
    \beta & \geq -x \\
    \beta & \leq 1 - x \\
    \beta & \geq -y \\
    \beta & \leq 1 - y
\end{align*}
\]
\( \text{Fourier-Motzkin algorithm} \)

- **The principle of the algorithm**

\[
\begin{align*}
x + \alpha - 1 & \geq 0 \\
x + y + \alpha - 3 & \geq 0
\end{align*}
\]

System of linear equations \((S)\)

\[
-x + 1 \leq x + y - 3
\]

Eliminate \(\alpha\)

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

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

\[
\begin{align*}
A & \leq \alpha \leq B, 0 \leq C \\
0 & \leq \alpha \leq \beta \\
v + \alpha & \models g \\
v + \beta & \models g \\
v + \alpha [y] & \in h_\alpha \\
v + \beta [y] & \in h_\beta
\end{align*}
\]

\[
\begin{align*}
\alpha & \geq 0 \\
\alpha & \leq \beta \\
\alpha & \geq -x \\
\alpha & \leq 1 - x \\
\alpha & \geq -y \\
\alpha & \leq 1 - y \\
\beta & \geq -x \\
\beta & \leq 1 - x \\
\beta & \geq -y \\
\beta & \leq 1 - y
\end{align*}
\]

max (\(A\)) \leq min (\(B\))

\[
\begin{align*}
0 & \leq x \leq 1 \\
0 & \leq y \leq 1
\end{align*}
\]

\[
0 \leq C
\]
**Technical Lemma**

\[ f : a \cdot \alpha + b \]
\[ g : c \cdot \beta + d \]
\[ h : \beta - \alpha \]

Domain \( D \) of \( \alpha \) and \( \beta \)

\[ \text{Input} \]

Technical lemma

\[ \text{arg sup } \min (h, f, g) \]

\( \alpha, \beta \in D \)

\[ \text{Output} \]
Technical Lemma

\[ f : a \cdot \alpha + b \]
\[ g : c \cdot \beta + d \]
\[ h : \beta - \alpha \]

Domain \( D \) of \( \alpha \) and \( \beta \)

Input

\[ l'_\alpha \times l'_\beta \]

\[ \alpha \mapsto P_i (l', v + \alpha [r]) \]
\[ \beta \mapsto P_i (l', v + \beta [r]) \]

Output

\[ \text{arg sup } \min_{\alpha, \beta \in D} (h, f, g) \]
(1st contribution) Technical Lemma

\[ f : a \cdot \alpha + b \]
\[ g : c \cdot \beta + d \]
\[ h : \beta - \alpha \]

Domain \( \mathcal{D} \) of \( \alpha \) and \( \beta \)

Input

Output

- When \( a > 0 \) and \( c < 0 \):

(a) Domain \( \mathcal{D} \)

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

Technical lemma

\[ \text{arg sup } \min_{\alpha, \beta \in \mathcal{D}} (h, f, g) \]
Technical Lemma

\[ f : a \cdot \alpha + b \]
\[ g : c \cdot \beta + d \]
\[ h : \beta - \alpha \]

Domain \( \mathcal{D} \) of \( \alpha \) and \( \beta \)

Input

Output

- When \( a > 0 \) and \( c < 0 \):

\[ \text{arg sup} \min_{\alpha, \beta \in \mathcal{D}} (h, f, g) \]
The algorithm

- Limited to acyclic timed automata
Complexity and issues

- The algorithm
  - Limited to acyclic timed automata
  - Upper bound time complexity: non-elementary
  - Complexity: grows with the number of cells, of clocks and $d_\ell$
(1st contribution) Complexity and issues

- The algorithm
  - Limited to acyclic timed automata
  - Upper bound time complexity: non-elementary
  - Complexity: grows with the number of cells, of clocks and $d_\ell$

- Causes of the high complexity
  - Overtiling
  - Exploration of all 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 $\beta$
Based on *pplpy*

<table>
<thead>
<tr>
<th>Clocks</th>
<th>Nb. of transitions</th>
<th>Runtime for $\ell_0$</th>
<th>Runtime for $\ell_1$</th>
<th>Runtime for $\ell_2$</th>
<th>Runtime for $\ell_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>0.82</td>
<td>0.059 (2 cells)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.071 (6 cells)</td>
<td>0.062 (3 cells)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0.73</td>
<td>0.034 (3 cells)</td>
<td>-</td>
<td>0.14 (6 cells)</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3.16 (582 cells)</td>
<td>1.01 (25 cells)</td>
<td>0.09 (3 cells)</td>
<td>0.41 (12 cells) 0.56 (6 cells) 0.14 (6 cells)</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1.95 (234 cells)</td>
<td>0.39 (12 cells)</td>
<td>0.59 (6 cells) 0.14 (6 cells)</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1.48 (1825 cells)</td>
<td>0.39 (12 cells)</td>
<td>0.59 (6 cells) 0.14 (6 cells)</td>
<td>-</td>
</tr>
</tbody>
</table>

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 ($m$)
Based on \textit{pplpy}

Covers the case of \textbf{linear timed automata} with \textbf{polyhedral} guards

<table>
<thead>
<tr>
<th>Clocks</th>
<th>Nb. of transitions</th>
<th>Runtime for $\ell_0$</th>
<th>Runtime for $\ell_1$</th>
<th>Runtime for $\ell_2$</th>
<th>Runtime for $\ell_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>0.82 (2 cells)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0.059 (2 cells)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.071 (6 cells)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0.062 (3 cells)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0.034 (3 cells)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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

$\ell_0 \leq x \leq 10 \leq y \leq 1$

(a) A $m$ transitions timed automaton

Runtime ($\text{sec.}$)

(b) Run times depending on the number of transitions ($m$)
Based on *pplpy*

Covers the case of **linear timed automata** with **polyhedral** guards

Runtime results on our examples:

<table>
<thead>
<tr>
<th>Clocks</th>
<th>Nb. of transitions</th>
<th>Runtime for $\ell_0$</th>
<th>Runtime for $\ell_1$</th>
<th>Runtime for $\ell_2$</th>
<th>Runtime for $\ell_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>0.82 (2 cells)</td>
<td>0.059 (2 cells)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.071 (6 cells)</td>
<td>0.062 (3 cells)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.73 (24 cells)</td>
<td>0.034 (3 cells)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>38.16 (582 cells)</td>
<td>1.01 (25 cells)</td>
<td>0.09 (3 cells)</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>19.95 (234 cells)</td>
<td>0.41 (12 cells)</td>
<td>0.56 (6 cells)</td>
<td>0.14 (6 cells)</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>143.48 (1825 cells)</td>
<td>0.39 (12 cells)</td>
<td>0.59 (6 cells)</td>
<td>0.14 (6 cells)</td>
</tr>
</tbody>
</table>
Based on *pplpy*

Covers the case of **linear timed automata with polyhedral guards**

Runtime results on our examples:

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

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

(a) A $m$ transitions timed automaton

(b) Runtimes depending on the number of transitions ($m$)
(1st contribution) The overtiling

- Example of overtiling

\[ 0 \leq y \leq 1 \]
\[ y := 0 \]
\[ 1 \leq x \leq 2 \]
\[ 0 \leq y \leq 1 \]

(a) Permissiveness on \( \ell_0 \)

(b) Permissiveness computed by our tool
(1st contribution) The causes of the overtiling

- 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) $g$: $g_0(x, y) = (1 - x) / 2$, $g_1(x, y) = 1 - y$
(1st contribution) The causes of the overtiling

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

• Example of maximisation: computing $\max(f, g)$

![Diagram](image)

(a) Wrong result

(b) Correct overtilled result $\max(f, g)$
The causes of the overtilling

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

- Example of maximisation: computing $\max(f, g)$
(1st contribution) Acyclic timed automata

- Example of an acyclic timed automata

\[ 0 \leq x \leq 1 \land 0 \leq y \leq 1 \]

\[ y := 0 \]

1 \leq x \leq 2 \land 0 \leq y \leq 1

- (1st contribution) Its permissiveness on \( \ell_0 \)

\[ \begin{array}{c}
0 \leq x \leq 1 \\
0 \leq y \leq 1 \\
y := 0 \\
1 \leq x \leq 2 \\
0 \leq y \leq 1 \\
1 \leq x \leq 2 \\
0 \leq y \leq 1
\end{array} \]
Second contribution: binary and levelled permissiveness.
Binary permissiveness principle

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

Permissiveness function

Binary permissiveness, $p = 0$
• Binary permissiveness principle

\[ \text{Fix } p \geq 0 \text{ and } \ell, \text{ compute } S(p, \ell) = \{ v \mid P_{d\ell}(\ell, v) \geq p \} \]

• Permissiveness function

• Binary permissiveness, \( p = 1/2 \)

• Some reductions
(2nd contribution) Binary and Levelled permissiveness

- **Binary permissiveness principle**

  \[ S(p, \ell) = \{ v \mid P_{d\ell}(\ell, v) \geq p \} \]

- **Binary permissiveness, \( p = 1/2 \)**

- **Permissiveness function**

- **Some reductions**

  - **Linear Lemma**: for linear TA, \( S(p, \ell) \) is a polyhedron.
(2nd contribution) Binary and Levelled permissiveness

- **Binary permissiveness principle**

  \[ \text{Fix } p \geq 0 \text{ and } \ell, \text{ compute } S(p, \ell) = \{ v \mid P_{d\ell}(\ell, v) \geq p \} \]

- **Permissiveness function**

  - **Binary permissiveness**, \( p = \frac{1}{2} \)

- **Some reductions**

  - **Linear Lemma**: for linear TA, \( S(p, \ell) \) is a polyhedron.
Binary and Levelled permissiveness

- **Binary permissiveness principle**

  Fix $p \geq 0$ and $\ell$, compute $S(p, \ell) = \{ v \mid P_{d_{\ell}}(\ell, v) \geq p \}$

- **Permissiveness function**

- **Binary permissiveness, $p = 1$**

- **Some reductions**

  - **Linear Lemma**: for linear TA, $S(p, \ell)$ is a polyhedron.
Binary and Levelled permissiveness

**Binary permissiveness principle**

Fix $p \geq 0$ and $\ell$, compute $S(p, \ell) = \{v \mid P_{d\ell}(\ell, v) \geq p\}$

**Permissiveness function**

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

**Some reductions**

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

- Levelled permissiveness reduces to binary permissiveness
Algorithm

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

- **Steps of the algorithm**
  - **Step 1:** Compute set of future enabled valuations \( S(\ell', p) \)

- **Step 2:** Compute the valuations \( v \) such that there exists \( \alpha \geq 0 \) that verifies:
(2nd contribution) Algorithm

\[ B_i(\ell, v) = \sup_{([\alpha, \alpha+p], a) \in \text{p-moves}(\ell, v)} \inf_{\delta \in [\alpha, \alpha+p]} \left( 1_{v + \delta[v] \in S_{i-1}(p, \ell')} \right) \]

• Steps of the algorithm
  
  ▶ **Step 1**: Compute set of future enabled valuations \( S(\ell', p) \)

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

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

- **Steps of the algorithm**
  - **Step 1**: Compute set of future enabled valuations \( S(\ell', p) \)
  - **Step 2**: Compute the valuations \( v \) such that there exists \( \alpha \geq 0 \) that verifies:
    - \([\alpha, \alpha+p]\) is an enabled move.
    - the successors are in \( S(\ell', p) \).

Fourier-Motzkin
Upper bound complexity for...

Binary permissiveness algorithm:

\[ O \left( (4c_g)^2 \cdot d_\ell \right) \]

- maximal number of constraints of any guard
- longest path between \( \ell \) and a goal location
Upper bound complexity for...

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 \( \{p_0, \cdots, p_m\} \):

\[ \mathcal{O} \left( (m + 1) (4c_g)^{2\cdot d_{\ell}} \right) \]
- number of levels
Third contribution: an numerical approximative computation.
Algorithm

(3rd contribution) Algorithm

initial configuration

interval sampling

delay sampling

2nd step: Backtracking on delays

3rd step: Backtracking on intervals

Robustness of timed automata
(3rd contribution) Algorithm

2nd step: Backtracking on delays

3rd step: Backtracking on intervals
Exploring intervals

\[ B = \max \left( n, \left( \frac{|I_{\text{max}}|}{s} \right)^2 \right) \]

Linear TA?

\[ \mathcal{O}\left((2 \cdot B)^{d\ell}\right) \]

Exploring delays

\[ \mathcal{O}\left((B \cdot \frac{M(A)}{s})^{d\ell}\right) \]
• A linear example
(3rd contribution) Implementation: runtime results

- A linear example

\[ 0 \leq x \leq 1 \quad 0 \leq y \leq 1 \quad y := 0 \]
\[ 1 \leq x \leq 2 \quad 0 \leq y \leq 1 \]

- An acyclic example
Error $\varepsilon = \text{Computed permissiveness} - \text{correct permissiveness}$
(3rd contribution) Implementation: precision results

**Error** \( \varepsilon = \text{Computed permissiveness} - \text{correct permissiveness} \)

- A linear example

- An acyclic example
Conclusion

- **Symbolic algorithm**
  - A non-elementary algorithm
  - Restricted to acyclic timed automata
  - Exact result
Conclusion

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

- **Binary and levelled permissiveness**
  - A doubly-exponential algorithm
  - Linear w.r.t the number of levels
  - Restricted to linear timed automata
  - Controlled approximation of the permissiveness
Conclusion

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

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

- **Numeric algorithm**
  - A doubly-exponential algorithm
  - Restricted to acyclic timed automata
  - Stability not proven
Current & future work

- Symbolic algorithm
  - Cyclic TA
  - Implementation of acyclic TA
  - Minimise the overtiling
Current & future work

- **Symbolic algorithm**
  - Cyclic TA
  - Implementation of acyclic TA
  - Minimise the overtiling

- **Binary and levelled permissiveness**
  - Acyclic (and cyclic) TA
  - Implementation with *pplpy*
Current & future work

- **Symbolic algorithm**
  - Cyclic TA
  - Implementation of acyclic TA
  - Minimise the overtiling

- **Binary and levelled permissiveness**
  - Acyclic (and cyclic) TA
  - Implementation with `pplpy`

- **Numeric algorithm**
  - Cyclic TA
  - Prove the stability
  - Implementation of polyhedral guards
Case of two polyhedra

Solved in 2001 by Bemporad, Fukuda and D. Torrisi:\(^5\):

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

(1st contribution) Computing the union of several polyhedra

- **Case of two polyhedra**
  Solved in 2001 by Bemporad, Fukuda and D. Torrisi\(^5\):
  - Computing convex hull by removing inequalities
  - Solving a linear program

- **Case of several \((\geq 3)\) polyhedra**
  Open problem...