Robust Model-Checking via Channel Automata

Nicolas Markey
Lab. Specification et Verification
ENS Cachan & CNRS, France

Joint work with
Patricia Bouyer and Pierre-Alain Reynier

(also starring Martin De Wulf, Laurent Doyen, Jean-François Raskin, Joël Ouaknine, James Worrell)

EPFL (Lausanne) – February 7, 2008
Controller Synthesis and Implementation

system:

⇒

property:

\[ G(\text{request} \Rightarrow F \text{grant}) \]

controller synthesis

yes/no
Controller Synthesis and Implementation

system:

property:

\[ G(\text{request} \Rightarrow F \text{grant}) \]

controller synthesis

yes/no
Timed Automata [AD90]

A finite control structure + a set $X$ of variables (clocks) → A configuration is a pair $(\ell, v) \in \text{Loc} \times \mathbb{R}_0^X$

A transition is of the form: $\ell \xrightarrow{g,a,r} \ell'$

An enabling condition (or guard) is $g ::= x \sim c | g \land g$

Two kinds of steps:

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

Discrete: if $\ell \xrightarrow{g,a,r} \ell'$ and $v \models g$, then $(\ell, v) \xrightarrow{a} (\ell', v[r \leftarrow 0])$
The semantics of timed automata is a mathematical idealization:

- Infinitely punctual: Exact synchronization is required when composing several TAs;
- Infinitely precise: Different clocks are assumed to increase at the same rate in both the controller and the system;
- Infinitely fast: It may happen, for instance, that the delays elapsed in a location will be shorter and shorter (and have bounded sum).

In practice, a processor is digital and imprecise. Even if we prove that a TA will not enter a set of bad states, its implementations could still generate bad behaviors.
Implementability of Timed Controllers

- The semantics of timed automata is a mathematical idealization:

  **Infinitely punctual**: Exact synchronization is required when composing several TAs;
  **Infinitely precise**: Different clocks are assumed to increase at the same rate in both the controller and the system.
  **Infinitely fast**: It may happen, for instance, that the delays elapsed in a location will be shorter and shorter (and have bounded sum).
The semantics of timed automata is a mathematical idealization:

- Infinitely punctual: Exact synchronization is required when composing several TAs;
- Infinitely precise: Different clocks are assumed to increase at the same rate in both the controller and the system;
- Infinitely fast: It may happen, for instance, that the delays elapsed in a location will be shorter and shorter (and have bounded sum).

In practice, a processor is digital and imprecise. Even if we prove that a TA will not enter a set of bad states, its implementations could still generate bad behaviors.
Implementability of Timed Controllers

Examples

- Zeno behaviors:

\[
\ell_0 \quad \text{y} \leq 1 \quad \ell_1
\]

\[x := 0\]
Examples

- Zeno behaviors:
- “Fragile” controllers [CHR02]:

\[
\begin{align*}
\ell_0 & : x = 0, \\
\ell_1 & : y = 0, \\
\ell_2 & : z = 0
\end{align*}
\]

Graph:

- \( \ell_3 \) → \( \ell_0 \): \( x > 1 \)
- \( \ell_0 \) → \( \ell_1 \): \( x = 1 \)
- \( \ell_1 \) → \( \ell_2 \): \( y = 1 \)
- \( \ell_2 \) → \( \ell_3 \): \( z > 0 \)
Implementability of Timed Controllers

Examples

- Zeno behaviors:

- “Fragile” controllers [CHR02]:

- Strict guards: Fischer’s protocol [KLL+97]
“trajectory tubes”: this approach “discards” behaviours that have too “strict” constraints [GHJ97].
Related works

- “trajectory tubes”: this approach “discards” behaviours that have too “strict” constraints [GHJ97].

- “probabilistic semantics”: removes unlikely trajectories [BBB+07].
Related works

- "trajectory tubes": this approach "discards" behaviours that have too "strict" constraints [GHJ97].

- "probabilistic semantics": removes unlikely trajectories [BBB+07].

- "platform modeling": more expressive, not much developed [AT05].
Outline of the talk

1. Introduction

2. A semantical approach to implementability
   - From implementability to robustness
   - Robust model checking for safety properties
   - Robust model checking for LTL properties

3. Timed Robust Model-Checking
   - Alternating timed automata and channel machines
   - A new approach to robust model-checking
   - Robust Model-Checking for CoFlatMTL

4. Conclusion
Outline of the talk

1. Introduction

2. A semantical approach to implementability
   • From implementability to robustness
   • Robust model checking for safety properties
   • Robust model checking for LTL properties

3. Timed Robust Model-Checking
   • Alternating timed automata and channel machines
   • A new approach to robust model-checking
   • Robust Model-Checking for CoFlatMTL

4. Conclusion
We consider a simple model of a platform, that repeatedly executes the following actions:

- read the value of the global clock;
- compute guards;
- fire one of the enabled transitions.

We assume that

- one loop takes at most $\Delta_P$ t.u. to execute;
- the global clock is updated every $\Delta_L$ t.u.

We write $\llbracket A \rrbracket_{\Delta_P, \Delta_L}^{\text{Impl}}$ for the set of executions of a timed automaton $A$ under this semantics.
The enlarged semantics for timed automata is defined by enlarging guards on transitions by a small tolerance $\Delta$:

$$\text{If } \llbracket g \rrbracket = [a; b], \text{ then } \llbracket g \rrbracket^\text{AASAP}_\Delta = [a - \Delta, b + \Delta].$$

We write $\llbracket A \rrbracket^\text{AASAP}_\Delta$ for the set of executions of a timed automaton $A$ under this semantics.
**Definition**

Let $\mathcal{A}$ be a timed automaton and $\varphi$ be a path property.

$\mathcal{A}$ is implementable w.r.t. $\varphi$ if, for some $\Delta_P > 0$ and $\Delta_L > 0$,

$$[\mathcal{A}]^{\text{impl}}_{\Delta_P, \Delta_L} \subseteq \mathcal{L}(\varphi).$$

$\mathcal{A}$ robustly satisfies $\varphi$, written $\mathcal{A} \models \varphi$, if for some $\Delta > 0$,

$$[\mathcal{A}]^{\text{AASAP}}_{\Delta} \subseteq \mathcal{L}(\varphi).$$
From implementability to robustness [DDR04]

Definition

Let $\mathcal{A}$ be a timed automaton and $\varphi$ be a path property.

$\mathcal{A}$ is implementable w.r.t. $\varphi$ if, for some $\Delta_P > 0$ and $\Delta_L > 0$,

$$\llbracket \mathcal{A} \rrbracket^{\text{impl}}_{\Delta_P, \Delta_L} \subseteq \mathcal{L}(\varphi).$$

$\mathcal{A}$ robustly satisfies $\varphi$, written $\mathcal{A} \models \varphi$, if for some $\Delta > 0$,

$$\llbracket \mathcal{A} \rrbracket^{\text{AASAP}}_{\Delta} \subseteq \mathcal{L}(\varphi).$$

Theorem

If $\Delta > 3\Delta_L + 4\Delta_P$, then $\llbracket \mathcal{A} \rrbracket^{\text{impl}}_{\Delta_P, \Delta_L} \subseteq \llbracket \mathcal{A} \rrbracket^{\text{AASAP}}_{\Delta}$. 
From implementability to robustness [DDR04]

Definition
Let $\mathcal{A}$ be a timed automaton and $\varphi$ be a path property. $\mathcal{A}$ is implementable w.r.t. $\varphi$ if, for some $\Delta_P > 0$ and $\Delta_L > 0$,

$$\llbracket \mathcal{A} \rrbracket_{\Delta_P, \Delta_L}^{\text{impl}} \subseteq \mathcal{L}(\varphi).$$

$\mathcal{A}$ robustly satisfies $\varphi$, written $\mathcal{A} \models \varphi$, if for some $\Delta > 0$,

$$\llbracket \mathcal{A} \rrbracket_{\Delta}^{\text{AASAP}} \subseteq \mathcal{L}(\varphi).$$

Theorem
If $\Delta > 3\Delta_L + 4\Delta_P$, then $\llbracket \mathcal{A} \rrbracket_{\Delta_P, \Delta_L}^{\text{impl}} \subseteq \llbracket \mathcal{A} \rrbracket_{\Delta}^{\text{AASAP}}$.

In other terms, implementability can be checked via robustness.
Some (harmless) assumptions...

In the sequel, we assume that:

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

This is not a restriction since

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

\[ \leadsto \text{we consider only closed regions.} \]
Some (harmless) assumptions...

In the sequel, we assume that:

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

This is not a restriction since

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

$$\leadsto$$ we consider only closed regions.

- the clocks are bounded by some constant $M$.

This is not a restriction: if $x > M$, enter a second copy of the state where the value of $x$ is irrelevant.
Some (harmless) assumptions...

In the sequel, we assume that:

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

This is not a restriction since

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

\[ \implies \text{we consider only closed regions.} \]

- the clocks are bounded by some constant \( M \).

This is not a restriction: if \( x > M \), enter a second copy of the state where the value of \( x \) is irrelevant.

- along any cycle of the region graph, all the clocks are reset.

This is a real restriction, but it is weaker than the “strongly non-Zeno” restriction.
Robust safety [DDMR04]

Reach_Δ(𝐀) is the set of reachable states in [𝐀]_Δ.

Δ₁ ≤ Δ₂ ⇒ Reach_Δ₁(𝐀) ⊆ Reach_Δ₂(𝐀)
Reach\(_{\Delta}(A)\) is the set of reachable states in \([A]_\Delta\).

\(\Delta_1 \leq \Delta_2 \Rightarrow \text{Reach}_{\Delta_1}(A) \subseteq \text{Reach}_{\Delta_2}(A)\)

Reach\(_{>0}(A) = \bigcap_{\Delta>0} \text{Reach}_{\Delta}(A)\) is the set of reachable states under the AASAP semantics for any \(\Delta > 0\).
Robust safety \cite{DDMR04}

\[ \text{Reach}_\Delta(\mathcal{A}) \text{ is the set of reachable states in } [[\mathcal{A}]]_\Delta. \]

\[ \Delta_1 \leq \Delta_2 \implies \text{Reach}_{\Delta_1}(\mathcal{A}) \subseteq \text{Reach}_{\Delta_2}(\mathcal{A}) \]

\[ \text{Reach}_{>0}(\mathcal{A}) = \bigcap_{\Delta > 0} \text{Reach}_\Delta(\mathcal{A}) \text{ is the set of reachable states under the AASAP semantics for any } \Delta > 0. \]

**Lemma**

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

\[ \text{Reach}_{>0}(\mathcal{A}) \cap B = \emptyset \quad \text{iff} \quad \exists \Delta > 0. \text{Reach}_\Delta(\mathcal{A}) \cap B = \emptyset. \]
Example

\[ x = 1 \quad y = 0 \]

\[ x \leq 2 \]

\[ x = 0 \quad y = 0 \]

\[ y \geq 2 \]

\[ x = 0 \quad y = 2 \]

\[ \text{Bad} \]
Example

\[ \begin{align*}
  x & = 1 \\
  y & = 0 \\
  x & \leq 2 \\
  y & \geq 2 \\
  x & = 0 \\
  y & = 2
\end{align*} \]

Graph with points and edges:
- A point at \((1, 0)\) labeled as \(a\) with \(x = 1\) and \(y = 0\).
- An edge from \(a\) to \(b\) with \(x = 0\) and \(y = 0\).
- An edge from \(b\) to \(c\) with \(x = 0\) and \(y = 2\).
- An edge from \(c\) to Bad with \(x = 0\) and \(y = 2\).
Example

\[ x = 1 \]
\[ y = 0 \]

\[ x \leq 2 \]
\[ y \geq 2 \]

\[ x = 0 \]
\[ y = 2 \]
Example

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

Graph:

- Node a: \( x = 1 \), \( y = 0 \)
- Node b: \( x = 0 \), \( y = 0 \)
- Node c: \( x = 0 \), \( y = 2 \)
- Node Bad

Lines:

- Green line: \( x \leq 2 \)
- Blue line: \( y \geq 2 \)
Example

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

Graph:

- Node a: \( x = 1 \), \( y = 0 \)
- Node b: \( x = 0 \), \( y = 0 \)\( y \geq 2 \)
- Node c: \( x = 0 \), \( y = 2 \)
- Node Bad

Lines:
- Green line: \( y = 0 \)
- Blue line: \( x = 0 \)
- Red line: \( x = 2 \)

Points:
- a at (1,0)
- b at (0,0)
- c at (0,2)
- Bad at (2,0)
Example

\[x = 1\]
\[y = 0\]

\[x \leq 2\]
\[y \geq 2\]

\[x = 0\]
\[y = 2\]
Example

\[ x = 1 \]
\[ y = 0 \]

\[ x \leq 2 \]
\[ x = 0 \]
\[ y \geq 2 \]
\[ y = 0 \]

\[ x = 0 \]
\[ y = 2 \]

Bad
Example

\[x = 1\]
\[y = 0\]

\[x \leq 2\]
\[y \geq 2\]

\[x = 0\]
\[y = 2\]

\[x = 0\]
\[y = 0\]
Example

\[ \begin{align*}
  x &\in [1-\Delta;1+\Delta] \\
y &:= 0
\end{align*} \]

\[ \begin{align*}
  x &\leq 2+\Delta \\
y &\geq 2-\Delta
\end{align*} \]

\[ \begin{align*}
  x &:= 0 \\
y &\leq 2+\Delta \\
y &:= 0
\end{align*} \]
Example

\[ x \in [1-\Delta; 1+\Delta] \]
\[ y := 0 \]
\[ x \leq 2 + \Delta \]
\[ y \geq 2 - \Delta \]
\[ y := 0 \]
\[ x \leq \Delta \]

Diagram:

- Node A with condition \( x \in [1-\Delta; 1+\Delta] \) and \( y := 0 \)
- Node B with conditions \( x := 0 \) and \( y \geq 2 - \Delta \)
- Node C with condition \( y \in [2-\Delta, 2+\Delta] \)
- Transition to Bad node with condition \( x \leq \Delta \)
Example

\[
x \in [1-\Delta;1+\Delta] \\
y := 0
\]

\[
x \leq 2 + \Delta
\]

\[
x := 0
\]

\[
y \geq 2 - \Delta
\]

\[
y := 0
\]

\[
x \leq \Delta
\]

\[
y \in [2-\Delta,2+\Delta]
\]

Bad

Diagram:

- Node a with x ∈ [1−Δ; 1+Δ], y := 0
- Node b with x ≤ 2 + Δ, y := 0
- Node c with x ≤ Δ, y ∈ [2−Δ, 2+Δ]
- Transition arrows connecting nodes:
  - a to b: x ∈ [1−Δ; 1+Δ], y := 0
  - b to c: x ≤ 2 + Δ
  - c to Bad: x ≤ Δ, y ∈ [2−Δ, 2+Δ]
Example

\[ x \in [1-\Delta; 1+\Delta] \]

\[ y : = 0 \]

\[ x \leq 2 + \Delta \]

\[ y \geq 2 - \Delta \]

\[ y : = 0 \]
Example

\[ x \in [1-\Delta;1+\Delta] \]
\[ y := 0 \]
\[ x \leq 2+\Delta \]
\[ x := 0 \]
\[ y \geq 2-\Delta \]
\[ y := 0 \]
\[ y \in [2-\Delta,2+\Delta] \]
\[ x \leq \Delta \]

Graph:
- Node a: \[ x \in [1-\Delta;1+\Delta] \]
- Node b: \[ x := 0 \]
- Node c: \[ y \geq 2-\Delta \]
- Node Bad: \[ y \in [2-\Delta,2+\Delta] \]

Diagram:
- Node a to b: \[ x \in [1-\Delta;1+\Delta] \]
- b to c: \[ x := 0 \]
- c to Bad: \[ y \geq 2-\Delta \]
- a to c: \[ x \leq \Delta \]

Graph edges:
- a to b: \[ x \in [1-\Delta;1+\Delta] \]
- b to c: \[ x := 0 \]
- c to bad: \[ y \geq 2-\Delta \]
- a to c: \[ x \leq \Delta \]
Example

$$x \in [1-\Delta; 1+\Delta]$$

$$y := 0$$

$$x \leq 2 + \Delta$$

$$x := 0$$

$$y \geq 2 - \Delta$$

$$y := 0$$

$$x \leq \Delta$$

$$y \in [2-\Delta, 2+\Delta]$$

$$\text{Bad}$$

$$b$$

$$c$$

$$a$$
Example

\[ x \in [1 - \Delta; 1 + \Delta] \]

\[ y = 0 \]

\[ x \leq 2 + \Delta \]

\[ y \geq 2 - \Delta \]

\[ y = 0 \]

\[ x \leq \Delta \]

\[ y \in [2 - \Delta, 2 + \Delta] \]

\[ \text{Bad} \]
Example

\[ x \in [1-\Delta;1+\Delta] \]
\[ y := 0 \]
\[ x \leq 2+\Delta \]
\[ y \geq 2-\Delta \]
\[ y := 0 \]
\[ x \leq \Delta \]
\[ y \in [2-\Delta,2+\Delta] \]
Example

\[ x \in [1-\Delta;1+\Delta] \]
\[ y := 0 \]
\[ x \leq 2+\Delta \]
\[ x := 0 \]
\[ y \geq 2-\Delta \]
\[ y := 0 \]
\[ x \leq \Delta \]
\[ y \in [2-\Delta,2+\Delta] \]

BAD
Example

$x \in [1-\Delta; 1+\Delta]$
$y := 0$
$x \leq 2 + \Delta$
$y \geq 2 - \Delta$
$y := 0$

$x \leq \Delta$
$y \in [2-\Delta, 2+\Delta]$

Bad
Example

\[ \begin{align*}
    x, y &\in [1-\Delta; 1+\Delta] \\
    x &\leq 2+\Delta \\
    y &\geq 2-\Delta \\
    x &\leq \Delta \\
    y &\in [2-\Delta, 2+\Delta] \\
\end{align*} \]
Example

\[ x \in [1-\Delta; 1+\Delta] \]
\[ x = 0 \]
\[ y \geq 2 - \Delta \]
\[ y = 0 \]
\[ x \leq 2 + \Delta \]
\[ y \in [2-\Delta, 2+\Delta] \]
Example

\[ x \in [1-\Delta; 1+\Delta] \]
\[ y = 0 \]
\[ x = 0 \]
\[ y \geq 2-\Delta \]
\[ y = 0 \]
\[ x \leq 2+\Delta \]
\[ x \leq \Delta \]
\[ y \in [2-\Delta, 2+\Delta] \]
Example

\[ x \in [1-\Delta; 1+\Delta] \]
\[ y := 0 \]
\[ x \leq 2 + \Delta \]
\[ x := 0 \]
\[ y \geq 2 - \Delta \]
\[ y := 0 \]
\[ y \in [2-\Delta, 2+\Delta] \]
Example

\[ x \in [1-\Delta; 1+\Delta] \]
\[ y := 0 \]
\[ x \leq 2 + \Delta \]
\[ x := 0 \]
\[ x \leq \Delta \]

bad

\[ y \geq 2 - \Delta \]
\[ y := 0 \]

\[ y \in [2-\Delta, 2+\Delta] \]
An algorithm for computing $\text{Reach}_{>0}(\mathcal{A})$ [DDMR04]

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

**Output:** The set $\text{Reach}_{> 0}(\mathcal{A})$
An algorithm for computing $\text{Reach}_{>0}(\mathcal{A})$ [DDMR04]

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

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

1. build the region graph $G$ of $\mathcal{A}$;
An algorithm for computing $\text{Reach}_{>0}(\mathcal{A})$ [DDMR04]

**Input:** A Timed Automaton $\mathcal{A}$
**Output:** The set $\text{Reach}_{>0}(\mathcal{A})$

1. build the region graph $G$ of $\mathcal{A}$;
2. compute $\text{SCC}(G) =$ the set of strongly connected components of $G$;
An algorithm for computing $\text{Reach}_{>0}(\mathcal{A})$ [DDMR04]

Input: A Timed Automaton $\mathcal{A}$
Output: The set $\text{Reach}_{>0}(\mathcal{A})$

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

6. return($J$);
An algorithm for computing \( \text{Reach}_0(\mathcal{A}) \) [DDMR04]

**Input:** A Timed Automaton \( \mathcal{A} \)

**Output:** The set \( \text{Reach}_0(\mathcal{A}) \)

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

6. return \( (J) \);
An algorithm for computing $\text{Reach}_{>0}(\mathcal{A})$ [DDMR04]

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

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

1. build the region graph $G$ of $\mathcal{A}$;
2. compute $\text{SCC}(G) =$ the set of strongly connected components of $G$;
3. $J := [(q_0)]$;
4. $J := \text{Reach}(G, J)$;
5. while $\exists \; S \in \text{SCC}(G). \; S \not\subseteq J$ and $S \cap J \neq \emptyset$,
   
   $J := J \cup S$;
   
   $J := \text{Reach}(G, J)$;
6. return($J$);
An algorithm for computing $\text{Reach}_{>0}(\mathcal{A})$ [DDMR04]

**Input:** A Timed Automaton $\mathcal{A}$  
**Output:** The set $\text{Reach}_{>0}(\mathcal{A})$

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

**Theorem**

This algorithm is correct.
An algorithm for computing $\text{Reach}_{>0}(\mathcal{A})$ [DDMR04]

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

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

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

**Theorem**

*Robustness w.r.t. safety properties can be checked in PSPACE.*
The algorithm is correct ($J \subseteq \text{Reach}_{>0}(\mathcal{A})$):

**Lemma**

If $[\mathcal{A}]$ then $[\mathcal{A}]_{\Delta}$

*Proof.* We build the new trajectory by slightly modifying the delay transitions in $\pi$. This crucially depends on the fact that all clocks are reset along the cycle.

□
Some intuition about the proof [DDMR04]

- The algorithm is correct ($J \subseteq \text{Reach}_{>0}(\mathcal{A})$):

**Lemma**

If $\llbracket A \rrbracket$ then $\llbracket A \rrbracket_\Delta$

**Lemma**

If $R(\mathcal{A})$ then $\llbracket A \rrbracket_\Delta$
The algorithm is complete ($J \supseteq \text{Reach}_{>0}(A)$):

**Lemma**

For any $k$ and $\alpha > 0$, there is a $\Delta_0$ s.t. for all $\Delta \leq \Delta_0$, for all path $\pi$ in $\llbracket A \rrbracket_\Delta$, there is a path $\pi'$ in $\llbracket A \rrbracket$ s.t. $d(\pi, \pi') \leq \alpha$. 
Some intuition about the proof [DDMR04]

The algorithm is complete ($J \supseteq \text{Reach}_{>0}(\mathcal{A})$):

**Lemma**

*For any $k$ and $\alpha > 0$, there is a $\Delta_0$ s.t. for all $\Delta \leq \Delta_0$, for all path $\pi$ in $[\mathcal{A}]_\Delta$, there is a path $\pi'$ in $[\mathcal{A}]$ s.t. $d(\pi, \pi') \leq \alpha$.***
The algorithm is complete ($J \supseteq \text{Reach}_{>0}(\mathcal{A})$):

**Lemma**

*For any $k$ and $\alpha > 0$, there is a $\Delta_0$ s.t. for all $\Delta \leq \Delta_0$, for all path $\pi$ in $[\mathcal{A}]_\Delta$, there is a path $\pi'$ in $[\mathcal{A}]$ s.t. $d(\pi, \pi') \leq \alpha$.***
Some intuition about the proof [DDMR04]

- The algorithm is complete ($J \supseteq \text{Reach}_{>0}(\mathcal{A})$):

**Lemma**

For any $k$ and $\alpha > 0$, there is a $\Delta_0$ s.t. for all $\Delta \leq \Delta_0$, for all path $\pi$ in $[\mathcal{A}]_\Delta$, there is a path $\pi'$ in $[\mathcal{A}]$ s.t. $d(\pi, \pi') \leq \alpha$.

*Proof.* Parametric DBMs.

□
Some intuition about the proof [DDMR04]

- The algorithm is complete \((J \supseteq \text{Reach}_{>0}(\mathcal{A}))\):

**Lemma**

For any \(k\) and \(\alpha > 0\), there is a \(\Delta_0\) s.t. for all \(\Delta \leq \Delta_0\), for all path \(\pi\) in \([\mathcal{A}]\_\Delta\), there is a path \(\pi'\) in \([\mathcal{A}]\) s.t. \(d(\pi, \pi') \leq \alpha\).

\[
\begin{array}{c}
\alpha \quad [\mathcal{A}] \\
p_1 \\
\end{array}
\quad
\begin{array}{c}
[\mathcal{A}]_\Delta \\
p_k \\
\end{array}
\]

Proof. Parametric DBMs.

**Lemma**

For all \(\alpha > 0\), there is a \(\Delta_0\) s.t. for all \(\Delta \leq \Delta_0\), for all path \(x \rightarrow y\) in \([\mathcal{A}]\_\Delta\),

\[
\text{if } x \in J \text{ then } d(y, J) \leq \alpha.
\]
Our algorithm suggests to extend the region automaton with extra transitions:

For any location $\ell$ and any two regions $r$ and $r'$, if

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

then we add a transition $\ell \rightarrow (\ell, r).$

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

Our algorithm suggests to extend the region automaton with extra transitions:

For any location $\ell$ and any two regions $r$ and $r'$, if
1. $r \cap r' \neq \emptyset$ and
2. $(\ell, r')$ belongs to an SCC of $R(\mathcal{A})$,
then we add a transition $(\ell, r) \xrightarrow{\gamma} (\ell, r')$.

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

Theorem

The set $\text{Reach}_{>0}(\mathcal{A})$ is the set of reachable regions in $R^*(\mathcal{A})$. 
Outline of the talk

1. Introduction

2. A semantical approach to implementability
   - From implementability to robustness
   - Robust model checking for safety properties
   - Robust model checking for LTL properties

3. Timed Robust Model-Checking
   - Alternating timed automata and channel machines
   - A new approach to robust model-checking
   - Robust Model-Checking for CoFlatMTL

4. Conclusion
Definition

\[ \text{LTL } \exists \psi, \varphi ::= p \mid \varphi \land \psi \mid \varphi \lor \psi \mid \neg \varphi \mid X \varphi \mid \varphi U \psi \]
LTL Robust Model-Checking [BMR06]

Definition

\[
\text{LTL } \exists \psi, \varphi ::= p \mid \varphi \land \psi \mid \varphi \lor \psi \mid \neg \varphi \mid X \varphi \mid \varphi U \psi
\]

LTL formulas are evaluated along paths:

Definition

\[
X
\]

\[
U
\]
**LTL Robust Model-Checking** [BMR06]

### Definition

LTL \( \exists \psi, \varphi ::= p \mid \varphi \land \psi \mid \varphi \lor \psi \mid \neg \varphi \mid X \varphi \mid \varphi U \psi \)

LTL formulas are evaluated along paths:

- **X**
  - Next
  - \( \circ \rightarrow \circ \rightarrow \circ \rightarrow \circ \rightarrow \circ \rightarrow \circ \rightarrow \circ \rightarrow \cdots \)

- **U**
  - Until
  - \( \circ \rightarrow \circ \rightarrow \circ \rightarrow \circ \rightarrow \circ \rightarrow \circ \rightarrow \circ \rightarrow \cdots \)

- **F**
  - Eventually
  - \( \circ \rightarrow \circ \rightarrow \circ \rightarrow \circ \rightarrow \circ \rightarrow \circ \rightarrow \circ \rightarrow \cdots \)

- **G**
  - Always
  - \( \circ \rightarrow \circ \rightarrow \circ \rightarrow \circ \rightarrow \circ \rightarrow \circ \rightarrow \circ \rightarrow \cdots \)
Theorem

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

Any LTL formula $\phi$ can be turned into an equivalent Büchi automaton $B_\phi$.

Theorem

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

$$A \models \phi \iff A \times B_{\neg \phi} \models \text{co-Büchi}(L \times Q_{\neg \phi}).$$
### Theorem

Any **LTL** formula \( \varphi \) can be turned into an equivalent **Büchi automaton** \( \mathcal{B}_\varphi \).

### Theorem

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

\[
\mathcal{A} \models \varphi \iff \mathcal{A} \times \mathcal{B}_{\neg \varphi} \models \text{co-Büchi}(L \times Q_{\neg \varphi}).
\]

### Theorem

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

\[
\mathcal{A} \models \text{co-Büchi}(Q) \iff \mathcal{R}^*(\mathcal{A}) \models \text{co-Büchi}(Q).
\]
Theorem

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

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

Corollary

$LTL$ robust model-checking is $\text{PSPACE}$-complete.
Some intuition about the proof [BMR06]

**Theorem**

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

$$\mathcal{A} \not\models \varphi \iff \mathcal{A} \times \mathcal{B}_{\neg \varphi} \models \text{co-Büchi}(L \times Q_{\neg \varphi}).$$

**Proof.**

$$\mathcal{A} \not\models \varphi \iff \forall \Delta > 0. \exists \pi \in \llbracket \mathcal{A} \rrbracket_\Delta. \pi \not\models \varphi$$

$$\iff \forall \Delta > 0. \exists \pi \in \llbracket \mathcal{A} \times \mathcal{B}_{\neg \varphi} \rrbracket_\Delta. \pi \models \text{Büchi}(L \times Q_{\neg \varphi}).$$

Hence

$$\mathcal{A} \models \varphi \iff \exists \Delta > 0. \forall \pi \in \llbracket \mathcal{A} \times \mathcal{B}_{\neg \varphi} \rrbracket_\Delta. \pi \models \text{co-Büchi}(L \times Q_{\neg \varphi}).$$
Theorem

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

$$\mathcal{A} \models \text{co-B"uchi}(Q) \iff \mathcal{R}^*(\mathcal{A}) \models \text{co-B"uchi}(Q).$$

Proof.

- From a path $\pi$ in $\mathcal{R}^*(\mathcal{A})$, we can build a path in $\llbracket \mathcal{A} \rrbracket_\Delta$ visiting (at least) the locations visited by $\pi$:
Some intuition about the proof [BMR06]

**Theorem**

Let $A$ be a timed automaton and $Q$ a set of locations of $A$. Then

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

**Proof.**

- From a path $\pi$ in $R^*(A)$, we can build a path in $[A]_{\Delta}$ visiting (at least) the locations visited by $\pi$:

- if $A \not\models \text{co-Büchi}(Q)$, since $q$ is finite, there is a $q \in Q$ s.t.

$$\forall \Delta > 0. \exists \pi \in [A]_{\Delta}. \pi \models \text{Büchi}\{q\}.$$

$\leadsto$ there is a path in $R^*(A)$ visiting $q$ infinitely many times.
Outline of the talk

1. Introduction

2. A semantical approach to implementability
   - From implementability to robustness
   - Robust model checking for safety properties
   - Robust model checking for LTL properties

3. Timed Robust Model-Checking
   - Alternating timed automata and channel machines
   - A new approach to robust model-checking
   - Robust Model-Checking for CoFlatMTL

4. Conclusion
Metric Temporal Logic [Koy87, AH92]

Definition

\[
\begin{align*}
\text{MTL } & \ni \psi, \varphi ::= p \mid \phi \land \psi \mid \phi \lor \psi \mid \neg \phi \mid \phi \text{ U}_I \psi
\end{align*}
\]
Metric Temporal Logic [Koy87,AH92]

### Definition

\[
\text{MTL} \ni \psi, \varphi ::= p \mid \varphi \land \psi \mid \varphi \lor \psi \mid \neg \varphi \mid \varphi \mid \varphi U_i \psi
\]

**MTL** formulas are evaluated along timed words:

![Diagram of MTL formulas evaluation along timed words]
Metric Temporal Logic \cite{Koy87,AH92}

**Definition**

\[
\text{MTL} \ni \psi, \varphi ::= p \mid \varphi \land \psi \mid \varphi \lor \psi \mid \neg \varphi \mid \varphi \ U_I \psi
\]

MTL formulas are evaluated along timed words:

\[
\text{Definition}
\]

\[
\text{delay} \in I
\]
Metric Temporal Logic [Koy87, AH92]

Definition

\[ \text{MTL } \owns \psi, \phi ::= p \mid \phi \land \psi \mid \phi \lor \psi \mid \neg \phi \mid \phi \U_I \psi \]

MTL formulas are evaluated along timed words:

- \( U_I \)
- \( F_I \)
- \( G_I \)
LTL formulas can (also) be turned into linear alternating Büchi automata:

Example

\[ G(a \Rightarrow F b) \]
LTL formulas can (also) be turned into linear alternating Büchi automata:

\[ G(a \Rightarrow F b) \]
LTL formulas can (also) be turned into linear alternating Büchi automata:

Example

\[ G(a \Rightarrow F b) \]
LTL formulas can (also) be turned into linear alternating Büchi automata:

Example

$$G(a \Rightarrow F b)$$
Similarly, MTL formulas can be turned into 1-clock alternating Büchi automata [OW05]:

Example

\[ G(a \Rightarrow F_{[1,2]} b) \]
Similarly, MTL formulas can be turned into 1-clock alternating Büchi automata [OW05]:

Example

\[ G(a \Rightarrow F_{[1,2]} b) \]
Similarly, MTL formulas can be turned into 1-clock alternating Büchi automata [OW05]:

Example

\[ G(a \Rightarrow F_{[1,2]} b) \]
Channel Automata with Renaming and Occurrence Testing (CAROT) are (roughly) Büchi automata equipped with a FIFO channel.

Example

\[
C : \\
\begin{array}{c}
s \\
\leftarrow a?:b? \\
a \rightarrow \{a,b\} \\
\rightarrow zero(a)? \\
t \\
\end{array}
\]
Definition

Channel Automata with Renaming and Occurrence Testing (CAROT) are (roughly) Büchi automata equipped with a FIFO channel.

Example

\[ C : \]

\[ s \]  \[ \Rightarrow a!,b! \]

\[ t \]  \[ \Rightarrow a?,b? \]

\[ s \]  \[ \Rightarrow zero(a)? \]

\[ t \]  \[ \Rightarrow a\rightarrow\{a,b\} \]

\[ s \]  \[ \Rightarrow t \]

\[ t \]  \[ \Rightarrow s \]
Definition

Channel Automata with Renaming and Occurrence Testing (CAROT) are (roughly) Büchi automata equipped with a FIFO channel.

Example

\[ C : \]

\[ a!,b! \]

\[ a\to\{a,b\} \]

\[ zero(a)? \]

\[ a?,b? \]

\[ \triangleright! \]

\[ \triangleright? \]

\[ s_{\triangleright} \]

\[ \triangleright \]

\[ \triangleright \]

\[ \triangleright \]

\[ \triangleright \]

\[ \triangleright \]

\[ t_{\triangleright} \]
Definition

Channel Automata with Renaming and Occurrence Testing (CAROT) are (roughly) Büchi automata equipped with a FIFO channel.

Example

\[ C : \]

```
S → a!, b!
s → a→{a,b}
S₀ → zero(a)?
S → a!, b!
S → S₀
S → t
S → a!, b?
t → a→{a,b}
t → t
```

```
|$\quad\quad\quad\quad\quad$ |
|---|---|---|---|---|
| $\triangleright$ | | | | |

```
Definition

Channel Automata with Renaming and Occurrence Testing (CAROT) are (roughly) Büchi automata equipped with a FIFO channel.

Example

C :  

\[
\begin{array}{c}
C: a!, b! \\
S: a, b! \\
zero(a)? \\
S \\
t \\
a, b? \\
\end{array}
\]
Definition

Channel Automata with Renaming and Occurrence Testing (CAROT) are (roughly) Büchi automata equipped with a FIFO channel.

Example

\[
C: \quad \begin{array}{c}
\text{zero}(a)? \\
\triangleright ! \\
\triangleright ? \\
{\text{S}}_{\triangleright} \\
\text{a} \rightarrow \{a, b\} \\
\triangleright ! \\
\triangleright ? \\
{\text{t}}_{\triangleright} \\
\end{array}
\]

\[
\begin{array}{c}
\text{a} \\
\triangleright \\
\text{a} \\
\triangleright \\
\end{array}
\]
Definition

Channel Automata with Renaming and Occurrence Testing (CAROT) are (roughly) Büchi automata equipped with a FIFO channel.

Example

\[ C : \begin{array}{ll}
    s & \quad a!,b! \\
    t & \quad a?,b? \\
    s & \quad \text{zero}(a)? \\
    t & \quad \text{zero}(a)? \\
    b & \quad \text{a} \quad \text{a} \\
\end{array} \]
Definition

Channel Automata with Renaming and Occurrence Testing (CAROT) are (roughly) Büchi automata equipped with a FIFO channel.

Example

\[ C : \]

\[
\begin{array}{c}
\text{a!,b!} \\
\text{a?},\text{b?}
\end{array}
\]

\[
\begin{array}{c}
s \quad \text{zero}(a)? \\
t
\end{array}
\]

\[
\text{b} \quad \text{a} \quad \text{b} \quad \text{!} \quad \text{?}
\]

\[
\text{a} \rightarrow \{a,b\}
\]
Definition
Channel Automata with Renaming and Occurrence Testing (CAROT) are (roughly) Büchi automata equipped with a FIFO channel.

Example
\[ C : \]

\[ a!,b! \quad a?\rightarrow\{a,b\} \quad zero(a)? \quad a?,b? \quad \]

\[ \begin{array}{c}
\text{zero}(a) ? \\
\text{a} \rightarrow \{a,b\} \\
a!,b! \\
\text{a}?,b? \\
\end{array} \]

\[ b \quad a \quad b \quad  \]
Definition

Channel Automata with Renaming and Occurrence Testing (CAROT) are (roughly) Büchi automata equipped with a FIFO channel.

Example

\[
C : \\
C : \\
C : \\
C : \\
C : \\
C : \\
\]

\[
\begin{array}{c}
\text{\(s\)} \\
\text{\(s\)} \\
\text{\(s\)} \\
\text{\(s\)} \\
\text{\(s\)} \\
\text{\(s\)} \\
\text{\(s\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(t\)} \\
\text{\(t\)} \\
\text{\(t\)} \\
\text{\(t\)} \\
\text{\(t\)} \\
\text{\(t\)} \\
\text{\(t\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(a!,b!\)} \\
\text{\(a!,b!\)} \\
\text{\(a!,b!\)} \\
\text{\(a!,b!\)} \\
\text{\(a!,b!\)} \\
\text{\(a!,b!\)} \\
\text{\(a!,b!\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(a\rightarrow\{a,b\}\)} \\
\text{\(a\rightarrow\{a,b\}\)} \\
\text{\(a\rightarrow\{a,b\}\)} \\
\text{\(a\rightarrow\{a,b\}\)} \\
\text{\(a\rightarrow\{a,b\}\)} \\
\text{\(a\rightarrow\{a,b\}\)} \\
\text{\(a\rightarrow\{a,b\}\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(a?,b?\)} \\
\text{\(a?,b?\)} \\
\text{\(a?,b?\)} \\
\text{\(a?,b?\)} \\
\text{\(a?,b?\)} \\
\text{\(a?,b?\)} \\
\text{\(a?,b?\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(s\)} \\
\text{\(s\)} \\
\text{\(s\)} \\
\text{\(s\)} \\
\text{\(s\)} \\
\text{\(s\)} \\
\text{\(s\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(t\)} \\
\text{\(t\)} \\
\text{\(t\)} \\
\text{\(t\)} \\
\text{\(t\)} \\
\text{\(t\)} \\
\text{\(t\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(b\)} \\
\text{\(a\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(b\)} \\
\text{\(a\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(b\)} \\
\text{\(a\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(b\)} \\
\text{\(a\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(b\)} \\
\text{\(a\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(b\)} \\
\text{\(a\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(b\)} \\
\text{\(a\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(b\)} \\
\text{\(a\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(b\)} \\
\text{\(a\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(b\)} \\
\text{\(a\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(b\)} \\
\text{\(a\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(b\)} \\
\text{\(a\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(b\)} \\
\text{\(a\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(b\)} \\
\text{\(a\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(b\)} \\
\text{\(a\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(b\)} \\
\text{\(a\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(b\)} \\
\text{\(a\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(b\)} \\
\text{\(a\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(b\)} \\
\text{\(a\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(b\)} \\
\text{\(a\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\text{\(b\)} \\
\end{array}
\]
Definition

Channel Automata with Renaming and Occurrence Testing (CAROT) are (roughly) Büchi automata equipped with a FIFO channel.

Example

\[ C : \]

\[ \begin{array}{c}
\text{a!,b!} \\
\text{a?},b? \\
\text{zero(a)?} \\
\text{a\rightarrow\{a,b\}} \\
\text{S\rightarrow} \\
\hline
\text{b} & \text{a} & \text{b} & \text{zero(a)} \\
\hline
\end{array} \]
Channel Automata with Renaming and Occurrence Testing (CAROT) are (roughly) Büchi automata equipped with a FIFO channel.

Example

\[ C : \]

\[
\begin{array}{c}
\text{zero}(a)\
\text{a→}\{a,b\}\
a?,b?\
a!,b!\
\end{array}
\]

\[
\begin{array}{c}
\text{!}
\text{?}
\text{!}
\text{?}
\text{!}
\text{?}
\end{array}
\]
Channel Automata with Renaming and Occurrence Testing (CAROT) are (roughly) Büchi automata equipped with a FIFO channel.

Example

\[ C : \]

- Transition: \( a \rightarrow \{a, b\} \)
- Zero test: \( \text{zero}(a) ? \)
- Channel events: \( a!, b! \), \( a?, b? \)

Diagram:

- States: \( S \), \( T \)
- Edges:
  - \( S \rightarrow T \)
  - \( T \rightarrow S \)
  - \( S \rightarrow S \)
  - \( T \rightarrow T \)
  - \( S \rightarrow T \) with \( a\)!
  - \( T \rightarrow S \) with \( a? \)

Termination:

- Channel buffer:
  - \( b \)
  - \( \text{zero}(a) ? \)

References:

- [BMOW07]
Definition

Channel Automata with Renaming and Occurrence Testing (CAROT) are (roughly) Büchi automata equipped with a FIFO channel.

Example

\[ C : \]

\[
\begin{array}{c}
S & a!,b! & a\rightarrow\{a,b\} & a?,b? \\
\text{zero}(a)\?
\end{array}
\]

\[
\begin{array}{c}
S & a!,b! & a\rightarrow\{a,b\} & a?,b? \\
\text{zero}(a)\?
\end{array}
\]
Definition
Channel Automata with Renaming and Occurrence Testing (CAROT) are (roughly) Büchi automata equipped with a FIFO channel.

Example

$$C : a!, b!$$

$$a \rightarrow \{a, b\}$$

$$\text{zero}(a)?$$

$$a?, b?$$

$$\triangleright b$$
**Definition**
Channel Automata with Renaming and Occurrence Testing (CAROT) are (roughly) Büchi automata equipped with a FIFO channel.

**Theorem**
*Cycle-bounded reachability is decidable and is in PSPACE (if the number of cycles is given in unary).*
From timed automata to CAROTs [BMOW07]

**location:**
- location of $\mathcal{A}$
- integ. part clocks of $\mathcal{A}$
- clocks (of $\mathcal{A}$ and $\mathcal{B}_{-\varphi}$) with integ. value

**channel:**
- config. (loc. + integ. parts) of $\mathcal{B}_{-\varphi}$
- order of frac. parts of clocks of both $\mathcal{A}$ and $\mathcal{B}_{-\varphi}$
Example of encoding

We consider the following configuration:

$$\mathcal{A} : \begin{cases} x = 1.4 \\ \ell, \quad y = 2 \\ z = 1.7 \end{cases}$$

$$\mathcal{B}_{-\varphi} : (\ell_1', u = 1), (\ell_2', u = 3.4), (\ell_1', u = 0.1)$$

This is encoded by the following configuration of the channel machine:
Example of encoding

We consider the following configuration:

\[ \mathcal{A} : \begin{cases} x = 1.4 \\ \ell, & y = 2 \\ z = 1.7 \end{cases} ; \quad \mathcal{B}_{\varphi} : (\ell_1', u = 1), (\ell_2', u = 3.4), (\ell_1', u = 0.1) \]

This is encoded by the following configuration of the channel machine:

\[ \begin{pmatrix} \lfloor x \rfloor = 1 \\ \ell, & \lfloor y \rfloor = 2 \\ \lfloor z \rfloor = 1 \end{pmatrix} \]

\[ \begin{array}{c}
  0 \quad 1 \\
  y, (\ell_1', 1) \quad (\ell_1', 0) \quad x, (\ell_2', 3) \quad z \\
\end{array} \]
Example of encoding

We consider the following configuration:

\[ \mathcal{A}: \begin{pmatrix} \ell, & y = 2 \\ x = 1.4 & \\ z = 1.7 \end{pmatrix}; \quad \mathcal{B}_{\varphi}: (\ell', u = 1), (\ell', u = 3.4), (\ell', u = 0.1) \]

This is encoded by the following configuration of the channel machine:

\[ \begin{pmatrix} \ell, & \lfloor x \rfloor = 1 \\ \lfloor y \rfloor = 2 \\ \lfloor z \rfloor = 1 \end{pmatrix} \]
Example of encoding

We consider the following configuration:

$$\mathcal{A} : \begin{cases} x = 1.4 \\
\ell, \quad y = 2 \\
z = 1.7 \end{cases} \quad ; \quad \mathcal{B}_{-\varphi} : (\ell'_1, u = 1), (\ell'_2, u = 3.4), (\ell'_1, u = 0.1)$$

This is encoded by the following configuration of the channel machine:

$$\begin{pmatrix} x \left\lfloor x \right\rfloor = 1 \\
\ell, \quad \left\lfloor y \right\rfloor = 2 \\
\left\lfloor z \right\rfloor = 2 \end{pmatrix}$$
Example of encoding

We consider the following configuration:

\[ \mathcal{A} : \left( \ell, \begin{array}{c} x = 1.4 \\ y = 2 \\ z = 1.7 \end{array} \right) \]; \quad \mathcal{B}_{\varphi} : (\ell'_1, u = 1), (\ell'_2, u = 3.4), (\ell'_1, u = 0.1) \]

This is encoded by the following configuration of the channel machine:

\[ \left( \begin{array}{c} [x] = 1 \\ \ell, \begin{array}{c} [y] = 2 \\ [z] = 2 \end{array} \end{array} \right) \]

\[ \xrightarrow{\sim} 1 \text{ cycle corresponds to } 1 \text{ t.u. elapsing} \]
The logic CoFlatMTL [BMOW07]

Definition

CoFlatMTL ∋ ψ, φ ::= p | φ ∧ ψ | φ ∨ ψ |
                   φ U_I ψ | φ U_J α | φ R_I ψ | β R_J ψ

where α and β are LTL formulas, I are bounded intervals, and J are unbounded intervals.

BoundedMTL is the fragment where all intervals are bounded.
The logic CoFlatMTL [BMOW07]

**Definition**

CoFlatMTL \( \ni \psi, \phi \ ::= \ p \ | \ \phi \land \psi \ | \ \phi \lor \psi \ | \)

\( \phi \ U_I \psi \ | \ \phi \ U_J \alpha \ | \ \phi \ R_I \psi \ | \ \beta \ R_J \psi \)

where \( \alpha \) and \( \beta \) are LTL formulas, \( I \) are bounded intervals, and \( J \) are unbounded intervals.

BoundedMTL is the fragment where all intervals are bounded.

**Theorem**

*Model-checking is EXPSPACE-complete for BoundedMTL and CoFlatMTL.*
The logic CoFlatMTL \[\text{[BMOW07]}\]

**Definition**

CoFlatMTL \( \ni \psi, \phi \) ::= \( p \mid \phi \land \psi \mid \phi \lor \psi \mid \phi \mathcal{U} I \psi \mid \phi \mathcal{U} J \alpha \mid \phi \mathcal{R} I \psi \mid \beta \mathcal{R} J \psi \)

where \( \alpha \) and \( \beta \) are LTL formulas, \( I \) are bounded intervals, and \( J \) are unbounded intervals.

BoundedMTL is the fragment where all intervals are bounded.

**Theorem**

*Model-checking is EXPSPACE-complete for BoundedMTL and CoFlatMTL.*

BoundedMTL is in PSPACE with unary-encoded constants.
Theorem

Cycle-bounded reachability is decidable and is in PSPACE (if the number of cycles is given in unary).

Proof.

\[
\begin{align*}
  s & b! & s & b! & s & R & t & \triangleright? & t_	riangleright & \triangleright! & t \\
  b? & t & c? & s & a! & s & b! & s & R & t & \triangleright? & t_	riangleright & \triangleright! & t \\
  a? & t & c? & s & b! & s & R & t & \triangleright? & t_	riangleright & \triangleright! & t \\
  c? & s & R & t & \triangleright? & t_	riangleright & \triangleright! & t
\end{align*}
\]
Some intuition about the proof [BMOW07]

**Theorem**

*Cycle-bounded reachability is decidable and is in PSPACE (if the number of cycles is given in unary).*

**Proof.**

s   b! s   b! s   R   t   ≥?   t≥   ≥!   t

b?   t   c?   s   a!   s   b! s   R   t   ≥?   t≥   ≥!   t

a?   t   c?   s   b! s   R   t   ≥?   t≥   ≥!   t

c?   s   R   t   ≥?   t≥   ≥!   t
Some intuition about the proof [BMOW07]

Theorem

*Cycle-bounded reachability is decidable and is in PSPACE (if the number of cycles is given in unary).*

Proof.

\[
\begin{array}{cccccccc}
  s & b! & s & b! & s & R & t & \triangleright ? & t \triangleright \\
  b? & t & c? & s & a! & s & b! & s & R & t & \triangleright ? & t \triangleright & \triangleright ! & t \\
  a? & t & c? & s & b! & s & R & t & \triangleright ? & t \triangleright & \triangleright ! & t \\
  c? & s & R & t & \triangleright ? & t \triangleright & \triangleright ! & t
\end{array}
\]
Some intuition about the proof \[BMOW07\]

**Theorem**

*Cycle-bounded reachability* is decidable and is in *PSPACE* (if the number of cycles is given in unary).

**Proof.**

\[
\begin{align*}
s \quad b! \quad s \quad b! \quad s \quad R \quad t \quad \triangleright? \quad t_\triangleright \quad - \quad t_\triangleright \quad \triangleright! \quad t \quad - \quad t \quad - \quad t \quad - \quad t \\
t \quad b? \quad t \quad c? \quad s \quad a! \quad s \quad b! \quad s \quad R \quad t \quad \triangleright? \quad t_\triangleright \quad \triangleright! \quad t \quad - \quad t \quad - \quad t \\
t \quad - \quad t \quad - \quad t \quad a? \quad t \quad c? \quad s \quad b! \quad s \quad R \quad t \quad \triangleright? \quad t_\triangleright \quad \triangleright! \quad t \quad - \quad t \\
t \quad - \quad t \quad - \quad t \quad - \quad t \quad - \quad t \quad c? \quad s \quad R \quad t \quad - \quad t \quad \triangleright? \quad t_\triangleright \quad \triangleright! \quad t \quad - \quad t
\end{align*}
\]
Some intuition about the proof [BMOW07]

**Theorem**

*Cycle-bounded reachability is decidable and is in PSPACE (if the number of cycles is given in unary).*

**Proof.**

```
s  b!  s  b!  s  R  t  ▷?  temaker  ▷!  t  -  t  -  t  -  t  -  t
  t  b?  t  c?  s  a!  s  b!  s  R  t  ▷?  temaker  ▷!  t  -  t  -  t
  t  -  t  -  t  a?  t  c?  s  b!  s  R  t  ▷?  temaker  ▷!  t  -  t
  t  -  t  -  t  -  t  -  t  c?  s  R  t  -  t  ▷?  temaker  ▷!  t
```
Some intuition about the proof [BMOW07]

Theorem

**Cycle-bounded reachability is decidable and is in** PSPACE (if the number of cycles is given in unary).

Proof.

```
s b! s b! s R t ▷? t ▷ t ▷ t ▷ t ▷ t ▷ t
```

```
t b? t c? s a! s b! s R t ▷? t ▷ t ▷ t ▷ t ▷ t ▷ t
```

```
t t t t a? t c? s b! s R t ▷? t ▷ t ▷ t ▷ t ▷ t ▷ t
```

```
t t t t t t t c? s R t t ▷? t ▷ t ▷ t ▷ t ▷ t ▷ t
```
Theorem

Model-checking is EXPSPACE-complete for BoundedMTL and CoFlatMTL.

Proof.

- **BoundedMTL**: only the first $h$ time units are relevant;
- **CoFlatMTL**: all accepting path can be split as follows:

![Diagram showing active and inactive parts]

Active parts: bounded duration.  
Inactive parts: “untimed” constraints.
Outline of the talk

1. Introduction

2. A semantical approach to implementability
   - From implementability to robustness
   - Robust model checking for safety properties
   - Robust model checking for LTL properties

3. Timed Robust Model-Checking
   - Alternating timed automata and channel machines
   - A new approach to robust model-checking
   - Robust Model-Checking for CoFlatMTL

4. Conclusion
Intuition

[Diagram of a state machine with states a, b, c, and Bad, with transitions labeled by conditions on x and y.]
Intuition

\[ a \xrightarrow{x=1, \ y:=0} b \xrightarrow{x=2, \ x:=0} c \xrightarrow{x=0, \ y=2} \text{Bad} \]

\[ \lfloor x \rfloor = a, \ \lfloor y \rfloor = b \]

\[ x \geq 2, \ x:=0 \]

\[ y \geq 2, \ y:=0 \]

\[ x, y \]

\[ [x] = 1, \ [y] = 0 \]
Intuition

\[ \lfloor x \rfloor = 2, \lfloor y \rfloor = 1 \]

\[ x = 1, y = 0 \]

\[ x \leq 2, x := 0 \]

\[ y \geq 2, y := 0 \]

\[ x = 0, y = 2 \]

\[ \text{Bad} \]
Intuition

\[ x=1, \quad y:=0 \]

\[ x \leq 2, \quad x:=0 \]

\[ y \geq 2, \quad y:=0 \]

\[ x=0, y=2 \]

\[ \text{Bad} \]

\[ [x]=2, \quad [y]=1 \]

\[ 2 < v(x) \leq 2 + \Delta \]
Intuition

\[\lfloor x \rfloor = 0\]

\[a \xrightarrow{x=1, y:=0} b \xrightarrow{\text{\(x\leq2, x:=0\)}} c \xrightarrow{\text{\(x=0, y=2\)}} \text{Bad}\]

\[y \geq 2, y:=0\]

\[\begin{array}{c}
\text{0} \\
\text{\(\Delta\)} \\
\text{1} - \Delta \\
\text{1}
\end{array}\]

\[\begin{array}{c}
x \\
y \\
\text{\(1\)}
\end{array}\]

\([x]=0, [y]=1\]
Intuition

A single channel machine whatever the number of clocks!

Additionnal clocks can be handled by a channel machine!

\[
\begin{align*}
x \leq 2, & \quad x := 0 \\
y \geq 2, & \quad y := 0
\end{align*}
\]
Intuition

[Diagram with states and transitions labeled with equations and conditions]

\[ [x] = 0, [y] = 1 \]

\[ 2 - \Delta \leq v(y) < 2 \]
Intuition

\[\text{Intuition}\]

\[\begin{align*}
x\geq 2, & \quad x := 0 \\
y & \geq 2, \quad y := 0
\end{align*}\]

Additionnal clocks can be handled by a channel machine!
Intuition

\[ x = 1, \ y = 0 \]

\[ x = 2, \ x = 0, \ y = 2 \]

\[ y \geq 2, \ y = 0 \]

Additionnal clocks can be handled by a channel machine!
Intuition

A single channel machine whatever the number of clocks!
Intuition

A single channel machine whatever the number of clocks!

\[ \lfloor x \rfloor = 1, \lfloor y \rfloor = 0 \]

\[ x = 1, y = 0 \]

\[ x = 2, x := 0 \]

\[ y = 2, y := 0 \]

\[ x = 0, y = 2 \]

\[ \Delta := 0 \]
Intuition

\[ \begin{align*}
\lfloor x \rfloor &= 0, \\lfloor y \rfloor &= 0 \\
&\quad \Rightarrow x_0, x, y, x_1, x_2, x_3, x_4, x_5
\end{align*} \]
Intuition

\[ x = 1, \ y = 0 \]

\[ x \leq 2, \ x = 0 \]

\[ y \geq 2, \ y = 0 \]

\[ x = 0, \ y = 2 \]

\[ [x] = 1, \ [y] = 0 \]
Intuition

\[ x = 1, \ y = 0 \]

(a) \[ x \leq 2, \ x = 0 \]

(b) \[ x = 0, \ y = 2 \]

(c) \[ y \geq 2, \ y = 0 \]

Bad

\[ \left\lceil x \right\rceil = 2, \ \left\lceil y \right\rceil = 1 \]

\[ x_0, x, y \quad \begin{array}{cccccc}
  x_1 & x_2 & x_3 & x_4 & x_5
\end{array} \]

\[ 0 \quad 1 \]
Intuition

\[ x = 1, \quad y = 0 \]

\[ x \leq 2, \quad x := 0 \]

\[ x = 0, y = 2 \]

\[ y \geq 2, \quad y := 0 \]

\[ [x] = 2, \quad [y] = 1 \]

\[ 2 < v(x) \land \forall i, \{v(x)\} \leq v(x_i) \]
Intuition

A single channel machine whatever the number of clocks!

Additionnal clocks can be handled by a channel machine!

$\lfloor x, y \rfloor = x_0$,

$x, y \geq 0$,

$x_2 = 0$,

$y, x = 3$,

$1$.
Intuition

A single channel machine whatever the number of clocks!

Additionnal clocks can be handled by a channel machine!

\[ \lfloor, \lfloor, y \rfloor = x_1 \]

\[ x_1 < y \geq x_2 \]

\( \forall x_0, y \)

\[ \mathcal{v}(y) < 2 \wedge \forall i, \{ \mathcal{v}(y) \} \geq \mathcal{v}(x_i) \]
Intuition

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

\[ y \geq 2, \ y := 0 \]

\[ x = 0, \ y = 2 \quad \rightarrow \quad \text{Bad} \]

\[ [x] = 0, \ [y] = 0 \]

\[ \begin{array}{ccccccc}
0 & x_1, y & x_2 & x_3 & x_4 & x_5 & x & x_0 \\
1 & & & & & & &
\end{array} \]
Intuition

$x < 3 \land (x > 2 \Rightarrow \bigvee_i \{x \leq x_{i+1} < x_{i-1}\})$

$\begin{align*}
&x \leq 2, \; x := 0 \\
y \geq 2, \; y := 0
\end{align*}$

Additionnal clocks can be handled by a channel machine!
Intuition

\[ x < 3 \land (x > 2 \Rightarrow \bigvee_i \{x\} \leq x_{i+1} < x_{i-1}) \]

\[ x \leq 2, \quad x := 0 \]

\[ y \geq 2, \quad y := 0 \]

\[ [x] = 0, \quad [y] = 0 \]

\[ \Delta, y \quad \Delta \quad \Delta \quad \Delta \quad \Delta \quad x \quad \Delta \]

A single channel machine whatever the number of clocks!
Let $\mathcal{A}$ be a timed automaton, $n$ be an integer.

- For each $0 \leq i < n$, we add a $\Delta$-automaton $\mathcal{B}_i$ involving a fresh clock $x_i$. We write $X^n = \{x_i \mid 0 \leq i < n\}$.

- We rewrite the guards in $\mathcal{A}$ as follows:

$$x \leq k \quad \sim \quad (x < k + 1) \land (x > k \implies \bigvee_{0 \leq i < n} \{x\} \leq x_{i+1} < x_{i-1})$$

(similarly for $x \geq k$)

**Definition**

We write $\mathcal{N}^n(\nu)$ for the product of those “timed automata” with initial valuation $\nu$ for clocks in $X^n$. 
Clocks in $X^n$ are intended to simulate a “tick” every $1/n$ time unit:

- $v^n$ is the valuation s.t. $v(x_i) = i/n$ for all $0 \leq i < n$;
- $N^n$ is the timed system $N^n(v^n)$. 
Networks of timed systems for enlarged semantics

Clocks in $X^n$ are intended to simulate a “tick” every $1/n$ time unit:

- $v^n$ is the valuation s.t. $v(x_i) = i/n$ for all $0 \leq i < n$;
- $N^n$ is the timed system $N^n(v^n)$.

Theorem

For any $n \geq 3$,

$$L([A]_{1/n}) \subseteq L([N^n]) \subseteq L([A]_{2/n})$$
Networks of timed systems for enlarged semantics

Clocks in $X^n$ are intended to simulate a “tick” every $1/n$ time unit:

- $v^n$ is the valuation s.t. $v(x_i) = i/n$ for all $0 \leq i < n$;
- $\mathcal{N}^n$ is the timed system $\mathcal{N}^n(v^n)$.

**Theorem**

For any $n \geq 3$,

$$L(\llbracket \mathcal{A} \rrbracket_{1/n}) \subseteq L(\llbracket \mathcal{N}^n \rrbracket) \subseteq L(\llbracket \mathcal{A} \rrbracket_{2/n})$$

**Theorem**

For any $\varphi \in \text{MTL}$,

$$\mathcal{A} \models \varphi \iff \exists n \geq 3. \llbracket \mathcal{N}^n \rrbracket \models \varphi.$$
Outline of the talk

1. Introduction

2. A semantical approach to implementability
   - From implementability to robustness
   - Robust model checking for safety properties
   - Robust model checking for LTL properties

3. Timed Robust Model-Checking
   - Alternating timed automata and channel machines
   - A new approach to robust model-checking
   - Robust Model-Checking for CoFlatMTL

4. Conclusion
From networks of timed systems to CAROTs

**location:**
- location of $\mathcal{A}$
- integ. part clocks of $\mathcal{A}$
- clocks (of $\mathcal{A}$, $\mathcal{A}_\Delta$ and $\mathcal{B}_{\neg \varphi}$) with integ. value
  + clocks of $\mathcal{A}$ at **beginning** and **end** of channel

**channel:**
- config. (loc. + integ. parts) of $\mathcal{B}_{\neg \varphi}$
- order of frac. parts of clocks of $\mathcal{A}$, $\mathcal{A}_\Delta$ and $\mathcal{B}_{\neg \varphi}$
Proposition

There exists a CAROT $C_{A, \lnot \varphi}$ that, starting with initial channel $\langle \Delta \rangle^n$, encodes the joint behaviour of the network $N^n$ and the alternating timed automaton $B_{\lnot \varphi}$.

Theorem

If $\varphi \in \text{BoundedMTL}$, there exist two integers $h$ and $N_0$ s.t. $A \not\equiv \varphi$ $\iff$ $C_{A, \lnot \varphi}$ has an $h$-cycle-bounded accepting computation on $\langle \Delta \rangle^{N_0}$. It is PSPACE-complete if the constants are in unary.
**Proposition**

There exists a CAROT $C_{\mathcal{A}, \neg \varphi}$ that, starting with initial channel $\langle \Delta \rangle^n$, encodes the joint behaviour of the network $\mathcal{N}^n$ and the alternating timed automaton $\mathcal{B}_{\neg \varphi}$.

---

**Proposition**

For any $\varphi \in \text{MTL}$,

$\mathcal{A} \not\models \varphi \iff \forall n \geq 3. \ C_{\mathcal{A}, \neg \varphi}$ has an accepting computation on $\langle \Delta \rangle^n$. 
### Proposition

There exists a CAROT $C_{A, \neg \varphi}$ that, starting with initial channel $\langle \Delta \rangle^n$, encodes the joint behaviour of the network $N^n$ and the alternating timed automaton $B_{\neg \varphi}$.

### Proposition

For any $\varphi \in MTL$,

$A \not\equiv \varphi \iff \forall n \geq 3. C_{A, \neg \varphi}$ has an accepting computation on $\langle \Delta \rangle^n$.

### Theorem

If $\varphi \in BoundedMTL$, there exist two integers $h$ and $N_0$ s.t.

$A \not\equiv \varphi \iff C_{A, \neg \varphi}$ has an $h$-cycle-bounded accepting computation on $\langle \Delta \rangle^{N_0}$. 
**Theorem**

If $\varphi \in \text{BoundedMTL}$, there exist two integers $h$ and $N_0$ s.t.

$\mathcal{A} \not\equiv \varphi \iff C_{\mathcal{A}, \neg \varphi}$ has an $h$-cycle-bounded accepting computation on $\langle \Delta \rangle^{N_0}$.

**Theorem**

*BoundedMTL* robust model checking is $\text{EXPSPACE}$-complete.

It is $\text{PSPACE}$-complete if the constants are in unary.
CoFlatMTL robust model-checking

When $\varphi \in \text{CoFlatMTL}$, an execution of $[\mathcal{A}]_{1/n} \times [\mathcal{B} - \varphi]$ can be decomposed as follows:

$\omega_1 \omega_2 \omega_3 \omega_4 \omega_5 \omega_6 \omega_7 \omega_8$

- $\omega_1$ to $\omega_2$ have cycle-bounded reachability in $\mathcal{A}$.
- $\omega_3$ to $\omega_4$ are inactive parts: reachability in $R^* (\mathcal{A} \times F)$.
- $\omega_5$ to $\omega_8$ have the B"uchi condition of $F$ in $R^* (\mathcal{A} \times F)$.

Theorem: CoFlatMTL robust model-checking is EXPSPACE-complete.

skip proof
When $\varphi \in \text{CoFlatMTL}$, an execution of $[\mathcal{A}]_{1/n} \times [\mathcal{B} \neg \varphi]$ can be decomposed as follows:

- **Active parts:** cycle-bounded reachability in $C_A$.
- **Inactive parts:** reachability in $R^*(A \times F)$.
- **Final part:** Büchi condition of $F$ in $R^*(A \times F)$.

Theorem: CoFlatMTL robust model-checking is EXPSPACE-complete. Skip proof.
CoFlatMTL robust model-checking

When $\varphi \in \text{CoFlatMTL}$, an execution of $\llbracket A \rrbracket_{1/n} \times \llbracket B \neg \varphi \rrbracket$ can be decomposed as follows:

Active parts: cycle-bounded reachability in $C_A$.

Inactive parts: reachability in $R^* (A \times F)$.

Final part: Büchi condition of $F$ in $R^* (A \times F)$.

Theorem

CoFlatMTL robust model-checking is EXPSPACE-complete.
Some intuition about the proof

**Theorem**

For any $n \geq 3$,

$$L([\mathcal{A}]_{1/n}) \subseteq L([\mathcal{N}^n]) \subseteq L([\mathcal{A}]_{2/n})$$

**Proof.** The relation $(\ell, v) \prec (\ell, v|_{X})$ is a simulation relation proving that $[\mathcal{N}^n] \sqsubseteq [\mathcal{A}]_{2/n}$.

Similarly, the relation $(\ell, v|_{X}) \prec (\ell, v)$ for $v$ satisfying $v(x_{i+1}) - v(x_i) = 1/n$ is a simulation relation. \qed
Some intuition about the proof

Theorem

*BoundedMTL* and *CoFlatMTL* robust model-checking is \(\text{EXPSPACE-complete} \).

Proof.

- **BoundedMTL**: cycle-bounded reachability in the associated CAROT.
- **CoFlatMTL**: 

\[
\begin{align*}
\varpi_1 & \leq h \\
\varpi_6 & \geq 2^{|\varphi|}(2W+1)
\end{align*}
\]
Some intuition about the proof

**Theorem**

*BoundedMTL* and *CoFlatMTL* robust model-checking is **EXPSPACE-complete**.

**Proof.**

- **BoundedMTL**: cycle-bounded reachability in the associated CAROT.
- **CoFlatMTL**: cycle-bounded reachability in $C_A$.

Active parts: cycle-bounded reachability in $C_A$.

Inactive parts: the behaviour of $\mathcal{B}_{\neg \varphi}$ is the behaviour of a non-deterministic Büchi automaton $\mathcal{F}_{\neg \varphi}$.

$\leadsto$ reachability in $R^*(A \times \mathcal{F}_{\neg \varphi})$.

Final part: Büchi condition of $\mathcal{F}_{\neg \varphi}$ in $R^*(A \times \mathcal{F}_{\neg \varphi})$.
Outline of the talk

1. Introduction

2. A semantical approach to implementability
   - From implementability to robustness
   - Robust model checking for safety properties
   - Robust model checking for LTL properties

3. Timed Robust Model-Checking
   - Alternating timed automata and channel machines
   - A new approach to robust model checking
   - Robust Model-Checking for CoFlatMTL

4. Conclusion
Conclusion

- **Implementability and robust model checking:**
  - relax the strict semantics of timed automata;
  - new techniques, but similar complexity results;
- Future work: develop a fully CAROT-based technique; MITL robust model checking; zone-based approach, efficient algorithms, applications; synthesis of robust controllers.
Conclusion

- **Implementability and robust model checking:**
  - relax the strict semantics of timed automata;
  - new techniques, but similar complexity results;

- **Recent extension to robust verification of timed properties:**
  - new techniques for handling robustness, using networks of timed systems;
  - (partly) uses the power of CAROTs;
  - also heavily relies on earlier results about robustness.
Conclusion

- **Implementability and robust model checking:**
  - relax the strict semantics of timed automata;
  - new techniques, but similar complexity results;

- **Recent extension to robust verification of timed properties:**
  - new techniques for handling robustness, using networks of timed systems;
  - (partly) uses the power of CAROTs;
  - also heavily relies on earlier results about robustness.

- **Future work**
  - develop a fully CAROT-based technique;
  - MITL robust model checking;
  - zone-based approach, efficient algorithms, applications.
  - synthesis of robust controllers.