Robust model-checking

Nicolas Markey

Lab. Specification et Verification
ENS Cachan & CNRS, France

Joint works with
Martin De Wulf, Laurent Doyen, Jean-François Raskin,
Patricia Bouyer and Pierre-Alain Reynier

ULB, Brussels – October 26, 2007
Controller Synthesis and Implementation

system: request \Rightarrow F grant

property:

G(request \Rightarrow F grant)

controller synthesis

yes/no
Controller Synthesis and Implementation

System:

Property:

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

Controller synthesis

Yes/no
Implementability of Timed Controllers

- The semantics of timed automata is a mathematical idealization:
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 a TA will have to perform actions at time $n$ and $n + 1/n$, for all $n$;

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.

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 a TA will have to perform actions at time $n$ and $n + 1/n$, for all $n$;
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 a TA will have to perform actions at time $n$ and $n + 1/n$, for all $n$;

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:

```
x := 0
```

```
ℓ₀
y ≤ 1
```

```
ℓ₁
```

"Fragile" controllers

[CHR02]

[KLL 97]
Examples

- Zeno behaviors:

- “Fragile” controllers [CHR02]:

\[
\begin{align*}
\ell_0 & \quad x = 1 \quad x := 0 \\
\ell_1 & \quad y = 1 \quad z := 0 \\
\ell_2 & \quad z > 0 \\
\ell_3 & \quad x > 1
\end{align*}
\]
Implementability of Timed Controllers

Examples

- Zeno behaviors:

- “Fragile” controllers [CHR02]:

- Strict guards: Fischer’s protocol [KLL+97]
Related works

- **“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.

$\leadsto$ We write $\llbracket \mathcal{A} \rrbracket_{\Delta_P, \Delta_L}^{\text{Impl}}$ for the set of executions of a timed automaton $\mathcal{A}$ under this semantics.
Enlarged semantics [DDR04]

The enlarged semantics for timed automata is defined by enlarging guards on transitions by a small tolerance $\Delta$:

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

$\leadsto$ We write $\llbracket \mathcal{A} \rrbracket_{\Delta}^{AASAP}$ for the set of executions of a timed automaton $\mathcal{A}$ 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 $\Delta_L > 0$,

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

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

$$\llbracket \mathcal{A} \rrbracket_{\Delta}^{\text{AASAP}} \subseteq \mathcal{L}(\varphi).$$
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 $\Delta_L > 0$,

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

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

$$\left[ \mathcal{A} \right]^{\text{AASAP}}_{\Delta} \subseteq L(\varphi).$$

Theorem
If $\Delta > 3\Delta_L + 4\Delta_P$, then $\left[ \mathcal{A} \right]^{\text{impl}}_{\Delta_P, \Delta_L} \subseteq \left[ \mathcal{A} \right]^{\text{AASAP}}_{\Delta}$. 
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 \( \Delta_L > 0 \),

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

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

\[
\mathcal{A}^{\text{AASAP}}_{\Delta} \subseteq \mathcal{L}(\varphi).
\]

**Theorem**

If \( \Delta > 3\Delta_L + 4\Delta_P \), then \( \mathcal{A}^{\text{impl}}_{\Delta_P, \Delta_L} \subseteq \mathcal{A}^{\text{AASAP}}_{\Delta} \).

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

In the sequel, we assume that:

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

This is not a restriction since

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

\[ \sim \] 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

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

\( \leadsto \) 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.

- 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.
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
Robust safety \cite{DDMR04}

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

$\Delta_1 \leq \Delta_2 \Rightarrow \text{Reach}_{\Delta_1}(\mathcal{A}) \subseteq \text{Reach}_{\Delta_2}(\mathcal{A})$
Robust safety \cite{DDMR04}

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

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

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

\( \sim \) Reach_\( \Delta \) (\( A \)) is the set of reachable states in \( \llbracket A \rrbracket_\Delta \).

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

\( \sim \) Reach_\( >0 \) (\( A \)) = \( \bigcap_{\Delta >0} \) Reach_\( \Delta \) (\( A \)) is the set of reachable states under the AASAP semantics for any \( \Delta > 0 \).

Lemma

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

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

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

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

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

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

Diagram:

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

Graph:

- Edge from a to b
- Edge from b to c
- Edge from c to Bad

Graphical representation:

- Axis x with values 0, 1, 2
- Axis y with values 0, 1, 2

Graph:

- Node a
- Node b
- Node c
- Node Bad
Example

\[ \begin{align*}
\text{Bad} & \quad x = 1 \\
& \quad y = 0 \\
\text{a} & \quad x = 0 \\
& \quad y = 0 \\
\text{b} & \quad x \leq 2 \\
& \quad y \geq 2 \\
\text{c} & \quad x = 0 \\
& \quad y = 2 \\
\text{Bad} & \end{align*} \]
Example

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

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

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

Diagram:

- **a** to **b** with conditions:
  - \( x = 1 \)
  - \( y = 0 \)
- **b** to **c** with conditions:
  - \( x \leq 2 \)
  - \( x = 0 \)
  - \( y = 0 \)
  - \( y \geq 2 \)
- **c** to **Bad** with conditions:
  - \( x = 0 \)
  - \( y = 2 \)
Example

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

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

\[ y := 0 \]

\[ x \leq 2+\Delta \]

\[ x := 0 \]

\[ y \geq 2-\Delta \]

\[ y := 0 \]

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

Consider the following constraints:

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

The diagram illustrates the possible transitions between states $a$, $b$, and $c$, leading to a bad state labeled 'Bad'.
Example

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

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

\[ x \leq \Delta \]

Bad
Example

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

\[ y := 0 \]
\[ x := 0 \]

\[ x \leq 2 + \Delta \]
\[ x := 0 \]
\[ y \geq 2 - \Delta \]
\[ y := 0 \]
Example

\[ \begin{align*}
\text{Bad } x & \in [1-\Delta; 1+\Delta] \\
y & := 0 \\
x & \leq 2 + \Delta \\
x & \leq \Delta \\
y & \geq 2 - \Delta \\
y & := 0 \\
y & \in [2-\Delta, 2+\Delta] \\
\text{Bad}
\end{align*} \]
Example

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

\[ y = 0 \]

\[ x \leq 2 + \Delta \]

\[ x = 0 \]

\[ y \geq 2 - \Delta \]

\[ y = 0 \]

\[ x \leq \Delta \]

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

\[ \text{Bad} \]
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

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

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

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

\[\begin{align*}
\text{Bad} & \quad x \in [1 - \Delta; 1 + \Delta] \\
& \quad y := 0 \\
& \quad y \geq 2 - \Delta \\
& \quad y := 0 \\
& \quad y \in [2 - \Delta; 2 + \Delta] \\
\end{align*}\]
Example

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

\[ b \]

\[ c \]

\[ \text{Bad} \]
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 $S \cap J = \emptyset$, $J := J \cup S$;
6. $J := \text{Reach}(G, J)$;
7. 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 $\mathcal{A}$;
An algorithm for computing \( \text{Reach}_{>0}(A) \) [DDMR04]

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

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

skip proof
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 and implementability w.r.t. safety properties can be checked in PSPACE.
Some intuition about the proof \[\text{[DDMR04]}\]

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

**Lemma**

\[\text{If } \llbracket \mathcal{A} \rrbracket \text{ then } \llbracket \mathcal{A} \rrbracket_{\Delta}\]

*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.

\(\square\)
Some intuition about the proof \[\text{DDMR04}\]

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

\[\text{Lemma}\]

- If \(\lbrack A \rbrack\) then \(\lbrack A \rbrack_{\Delta}\)

\[\text{Lemma}\]

- If \(\mathcal{R}(\mathcal{A})\) then \(\lbrack \mathcal{A} \rbrack_{\Delta}\)
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 $\llbracket \mathcal{A} \rrbracket_\Delta$, there is a path $\pi'$ in $\llbracket \mathcal{A} \rrbracket$ 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 $\llbracket \mathcal{A} \rrbracket_\Delta$, there is a path $\pi'$ in $\llbracket \mathcal{A} \rrbracket$ s.t. $d(\pi, \pi') \leq \alpha$.

**Proof.** Parametric DBMs.
Some intuition about the proof \cite{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 $[A]_{\Delta}$, there is a path $\pi'$ in $[A]$ s.t. $d(\pi, \pi') \leq \alpha$.

**Proof.** Parametric DBMs. \hfill \square

**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 $[A]_{\Delta}$,

\[
\text{if } x \in J \text{ then } d(y, J) \leq \alpha.
\]
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 $\mathcal{R}(\mathcal{A})$,

then we add a transition $(\ell, r) \xrightarrow{\gamma} (\ell, r')$.

We write $\mathcal{R}^*(\mathcal{A})$ for the resulting automaton.
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 $\mathcal{R}(\mathcal{A})$,
then we add a transition $(\ell, r) \xrightarrow{\gamma} (\ell, r')$.

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

**Theorem**

The set $\text{Reach}_{>0}(\mathcal{A})$ is the set of reachable regions in $\mathcal{R}^*(\mathcal{A})$. 
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
LTL Robust Model-Checking [BMR06]

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\exists \psi, \varphi ::= p \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \neg \varphi \mid X \varphi \mid \varphi U \psi$</td>
</tr>
</tbody>
</table>
Definition

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

LTL formulas are evaluated along paths:
## LTL Robust Model-Checking [BMR06]

### Definition

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

### LTL formulas are evaluated along paths:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Path 1</th>
<th>Path 2</th>
<th>Path 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
<td><img src="path_diagram1.png" alt="Path Diagram" /></td>
<td><img src="path_diagram2.png" alt="Path Diagram" /></td>
<td><img src="path_diagram3.png" alt="Path Diagram" /></td>
</tr>
<tr>
<td>$U$</td>
<td><img src="path_diagram4.png" alt="Path Diagram" /></td>
<td><img src="path_diagram5.png" alt="Path Diagram" /></td>
<td><img src="path_diagram6.png" alt="Path Diagram" /></td>
</tr>
<tr>
<td>$F$</td>
<td><img src="path_diagram7.png" alt="Path Diagram" /></td>
<td><img src="path_diagram8.png" alt="Path Diagram" /></td>
<td><img src="path_diagram9.png" alt="Path Diagram" /></td>
</tr>
<tr>
<td>$G$</td>
<td><img src="path_diagram10.png" alt="Path Diagram" /></td>
<td><img src="path_diagram11.png" alt="Path Diagram" /></td>
<td><img src="path_diagram12.png" alt="Path Diagram" /></td>
</tr>
</tbody>
</table>
Any LTL formula $\varphi$ can be turned into an equivalent Büchi automaton $\mathcal{B}_\varphi$. 
Theorem

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

Theorem

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

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

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

$$A \models \text{co-Büchi}(Q) \iff R^*(A) \models \text{co-Büchi}(Q).$$
Theorem
Let $\mathcal{A}$ be a timed automaton and $Q$ a set of locations of $\mathcal{A}$. Then

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

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

**Theorem**

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

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

**Proof.**

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

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

Hence

$A \models \varphi \iff \exists \Delta > 0. \forall \pi \in \llbracket A \times B_{\neg \varphi} \rrbracket_\Delta. \pi \models \text{co-Büchi}(L \times Q_{\neg \varphi})$
Some intuition about the proof [BMR06]

**Theorem**

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

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

**Proof.**

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

**Theorem**

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

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

**Proof.**

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

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

  $$\forall \Delta > 0. \exists \pi \in \llbracket \mathcal{A} \rrbracket_\Delta. \pi \models \text{Büchi}({q}).$$

  $\Rightarrow$ there is a path in $R^*(\mathcal{A})$ 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

\[
\text{MTL} \ni \psi, \varphi ::= p \mid \varphi \land \psi \mid \varphi \lor \psi \mid \neg \varphi \mid \varphi \text{ U, } \psi
\]
Metric Temporal Logic \([\text{Koy87, AH92}]\)

**Definition**

\[ \text{MTL} \ni \psi, \phi ::= p \mid \phi \land \psi \mid \phi \lor \psi \mid \neg \phi \mid \phi \mathbf{U} I \psi \]

MTL formulas are evaluated along timed words:

**Definition**

\begin{tikzpicture}
  \node[circle, fill=green!30] (a) at (0,0) {};
  \node[circle, fill=red!30] (b) at (1,0) {};
  \node[circle, fill=gray!30] (c) at (2,0) {};
  \node[circle, fill=green!30] (d) at (3,0) {};
  \node[circle, fill=red!30] (e) at (4,0) {};
  \node[circle, fill=gray!30] (f) at (5,0) {};

  \draw[->] (a) -- (b);
  \draw[->] (b) -- (c);
  \draw[->] (c) -- (d);
  \draw[->] (d) -- (e);
  \draw[->] (e) -- (f);
\end{tikzpicture}
**Metric Temporal Logic**  \([\text{Koy87, AH92}]\)

**Definition**

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

MTL formulas are evaluated along timed words:

**Definition**

\[ \mathbf{U}_I \text{ delay} \in I \]
Metric Temporal Logic \cite{Koy87,AH92}

**Definition**

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

MTL formulas are evaluated along timed words:

**Definition**

- **U** \(_I\): Delay in interval \(I\)
- **F** \(_I\): Event in interval \(I\)
- **G** \(_I\): Event infinitely often in interval \(I\)

\[\text{delay} \in I\]
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:

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

\[ G(a \Rightarrow F b) \]
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]:

$$G(a \Rightarrow F_{[1,2]} b)$$

Example
Similarly, MTL formulas can be turned into 1-clock alternating Büchi automata [OW05]:

Example

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

Example

\[ C : \]

\[ a!,b! \rightarrow a\rightarrow\{a,b\} \]

\[ \text{zero}(a) ? \]

\[ \triangledown ! \rightarrow \triangledown ? \]

\[ \triangledown ! \rightarrow \triangledown ? \]
**Definition**

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

**Example**

\[
\begin{align*}
C &: \quad a!, b! \\
& \xrightarrow{a\rightarrow\{a,b\}} a?, b? \\
& \xrightarrow{\text{zero}(a)\text{?}} \\
\end{align*}
\]
Definition

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

Example

C : 

\[
\begin{align*}
&\xrightarrow{a!,b!} \\
&s \xrightarrow{a \rightarrow \{a, b\}} t \\
&s \xleftarrow{zero(a)??} t \\
&t \xrightarrow{a?,b?} \\
\end{align*}
\]
Definition

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

Example

\[ C : \]

- \( a!, b! \)
- \( a \rightarrow \{a, b\} \)
- \( zero(a)? \)
- \( a?, b? \)
- \( \triangleleft \)
- \( \triangleleft \)
- \( \triangleleft \)
- \( \triangleleft \)
- \( \triangleleft \)
Definition

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

Example

\[ C : \]

- States: \( s \) and \( t \)
- Transitions:
  - \( s \) to \( t \) on \( a \rightarrow \{ a, b \} \)
  - \( s \) to \( s \) on \( \text{zero}(a) \)?
  - \( t \) to \( t \) on \( \text{zero}(a) \)!
- Channels: \( a !, b ! \) and \( a ?, b ? \)
Definition

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

Example

\[ C : \]

- **States:**
  - \( S \): the start state.
  - \( T \): another state.

- **Transitions:**
  - \( a! \), \( b! \):
    - From \( S \) to \( T \):
      - \( a \rightarrow \{ a, b \} \)
    - From \( T \) to \( S \):
      - \( \text{zero}(a) \)?
  - \( a? \), \( b? \):
    - From \( T \) to \( S \):
    - From \( S \) to \( T \):

- **Labels:**
  - \( a \), \( b \):
  - \( \triangleleft \) (input channel):
  - \( \triangleright \) (output channel):

- **Initial State:** \( S \)
- **Final State:** \( T \)
Definition

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

Example

\[ C : \]

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

\[ \begin{array}{c}
\text{s} \\
\text{t} \\
\end{array} \]

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

\[ \begin{array}{c}
\text{s} \\
\text{t} \\
\end{array} \]
Definition

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

Example

\[ C : \]

\[ a!, b! \quad a?; b? \]

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

\[ \text{zero}(a)? \]

\[ s \quad t \]

\[ \blacktriangleright \]

\[ \text{zero}(a)? \]

\[ s \quad t \]

\[ b \quad a \quad b \quad \blacktriangleright \]

\[ \text{zero}(a)? \]
Definition

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

Example

 Alternating timed aut. and channel machines [BMOW07]
Alternating timed aut. and channel machines [BMOW07]

**Definition**

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

**Example**

\[ C : \]

\[ \begin{align*}
\text{zero}(a) &
\end{align*} \]

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

\[ t \rightarrow ! \]

\[ s \rightarrow ? \]

\[ t \rightarrow ! \]

\[ s \rightarrow ? \]

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

Example

\[ C : \]

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

\[ \text{Diagram:} \]

\[ \text{Input:} b a b \]

[BMOW07]
**Definition**

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

**Example**

The figure illustrates a CAROT with states $s$ and $t$. The transitions are as follows:
- $s$ to $t$ with $a \to \{a, b\}$
- $t$ to $s$ with $\text{zero}(a)$?
- $s$ with $\triangleright!$
- $t$ with $\triangleright?$
- $t$ with $\triangleright!$
- $s$ with $\triangleright?$

The FIFO channel is represented by the sequence $\triangleright b a$. The automaton transitions are shown in the diagram.
**Definition**

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

**Example**

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

\[
\begin{array}{c}
\text{▷} \\
\text{▷?} \\
\text{▷} \\
\text{▷?} \\
\end{array}
\]

\[
\begin{array}{c}
\text{▷} \\
\text{▷} \\
\text{b} \\
\end{array}
\]

\[
\begin{array}{c}
\text{▷} \\
\text{▷} \\
\text{▷} \\
\text{▷} \\
\text{▷} \\
\end{array}
\]

\[
\begin{array}{c}
\text{▷} \\
\text{▷} \\
\text{▷} \\
\text{▷} \\
\text{▷} \\
\end{array}
\]

\[
\begin{array}{c}
\text{▷} \\
\text{▷} \\
\text{▷} \\
\text{▷} \\
\text{▷} \\
\end{array}
\]

\[
\begin{array}{c}
\text{▷} \\
\text{▷} \\
\text{▷} \\
\text{▷} \\
\text{▷} \\
\end{array}
\]

\[
\begin{array}{c}
\text{▷} \\
\text{▷} \\
\text{▷} \\
\text{▷} \\
\text{▷} \\
\end{array}
\]

\[
\begin{array}{c}
\text{▷} \\
\text{▷} \\
\text{▷} \\
\text{▷} \\
\text{▷} \\
\end{array}
\]

\[
\begin{array}{c}
\text{▷} \\
\text{▷} \\
\text{▷} \\
\text{▷} \\
\text{▷} \\
\end{array}
\]

\[
\begin{array}{c}
\text{▷} \\
\text{▷} \\
\text{▷} \\
\text{▷} \\
\text{▷} \\
\end{array}
\]

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

Example
C : $a!,b! \rightarrow a\rightarrow\{a,b\}$

zero$(a)$?
Definition

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

Example

\[ C : \]

- \( s \) to \( t \): \( a \rightarrow \{a, b\} \) and \( \text{zero}(a) \)
- \( s \) to \( s \): \( a!, b! \)
- \( t \) to \( t \): \( a?, b? \)
- \( s \) to \( t \): \( \triangleright! \) and \( \triangleright？ \)
- \( t \) to \( s \): \( \triangleright！ \) and \( \triangleright？ \)
- \( s \) to \( t \) and \( t \) to \( s \): FIFO channel with \( \triangleright b \)
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).
Channel Automata with Renaming and Occurrence Testing (CAROT) are (roughly) Büchi automata equipped with a FIFO channel.

The channel is used to store the configuration of the alternating timed automaton.
Definition

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

Example

The channel is used to store the configuration of the alternating timed automaton.
From timed automata to CAROTs [BMOW07]

bounded information

location:
- location of $A$
- integ. part clocks of $A$
- clocks (of $A$ and $B_\varphi$) with integ. value

channel:
- config. (loc. + integ. parts) of $B_\varphi$
- order of frac. parts of clocks of both $A$ and $B_\varphi$
The logic CoFlatMTL \cite{BMOW07}

**Definition**

CoFlatMTL \( \ni \psi, \varphi ::= p \mid \varphi \land \psi \mid \varphi \lor \psi \mid \varphi \U_I \psi \mid \varphi \U_J \alpha \mid \varphi \R_I \psi \mid \beta \R_J \psi \)

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

**Theorem**

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

BoundedMTL is in PSPACE with unary-encoded constants.

skip proof
The logic CoFlatMTL \cite{BMOW07}

**Definition**

\[
\text{CoFlatMTL} \ni \psi, \varphi ::= p \mid \varphi \land \psi \mid \varphi \lor \psi \mid \\
\varphi \ U_I \psi \mid \varphi \ U_J \alpha \mid \varphi \ R_I \psi \mid \beta \ R_J \psi
\]

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 [BMOW07]

**Definition**

CoFlatMTL \[\exists \psi, \varphi ::= p \mid \varphi \land \psi \mid \varphi \lor \psi \mid \varphi \mathbf{U}_I \psi \mid \varphi \mathbf{U}_J \alpha \mid \varphi \mathbf{R}_I \psi \mid \beta \mathbf{R}_J \psi\]

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.
Some intuition about the proof [BMOW07]

Theorem

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

Proof.

\[
\begin{align*}
    s & \ b! & \ s & \ b! & \ s & \ R & \ t & \ \triangleright? & \ t_{\triangleright} & \ \triangleright! & \ t \\
    b? & \ t & \ c? & \ s & \ a! & \ s & \ b! & \ s & \ R & \ t & \ \triangleright? & \ t_{\triangleright} & \ \triangleright! & \ t \\
    a? & \ t & \ c? & \ s & \ b! & \ s & \ R & \ t & \ \triangleright? & \ t_{\triangleright} & \ \triangleright! & \ t \\
    c? & \ s & \ R & \ t & \ \triangleright? & \ t_{\triangleright} & \ \triangleright! & \ t
\end{align*}
\]
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.**

\[
\begin{array}{ccccccccc}
  s & b! & s & b! & s & R & t & ? & t_> & ! & t \\
  b? & t & c? & s & a! & s & b! & s & R & t & ? & t_> & ! & t \\
  a? & t & c? & s & b! & s & R & t & ? & t_> & ! & t \\
  c? & s & R & t & ? & t_> & ! & t
\end{array}
\]
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.**

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

Theorem

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

Proof.

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

**Theorem**

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

**Proof.**

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

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:

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

- 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
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:

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

(similarly for $x \geq k$)

**Definition**

We write $\mathcal{N}^n(v)$ for the product of those “timed automata” with initial valuation $v$ 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 $n \geq 3$,

$$L([\mathcal{A}]_{1/n}) \subseteq L([\mathcal{N}^n]) \subseteq L([\mathcal{A}]_{2/n})$$
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 \( 0 \leq i < n \);
- \( N^n \) is the timed system \( N^n(v^n) \).

**Theorem**

For any \( n \geq 3 \),

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

**Theorem**

For any \( \varphi \in MTL \),

\[
\mathcal{A} \models \varphi \iff \exists n \geq 3. [N^n] \models \varphi.
\]
From networks of timed systems to channel automata

Example

\[ x = 1, y = 0 \]

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

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

\[ x = 0, y = 2 \]

\[ \text{Bad} \]

This can be encoded on a channel automaton!

The same channel automaton can be used to encode $N_n$ for any $n \geq 3$. 
From networks of timed systems to channel automata

Example

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

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

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

This can be encoded on a channel automaton!

The same channel automaton can be used to encode \( N_n \) for any \( n \geq 3 \).
From networks of timed systems to channel automata

Example

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

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

\[ x = 0, y = 2 \]

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

This can be encoded on a channel automaton!
From networks of timed systems to channel automata

Example

This can be encoded on a channel automaton!

The same channel automaton can be used to encode $N_n$ for any $n \geq 3$. 
From networks of timed systems to channel automata

Example

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

This can be encoded on a channel automaton!

The same channel automaton can be used to encode \( N_{n} \) for any \( n \geq 3 \).
From networks of timed systems to channel automata

Example

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

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

\[ x = 0, y = 2 \]

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

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

\[ x_5, x_0, x, y, x_1, x_2, x_3, x_4 \]
From networks of timed systems to channel automata

Example

This can be encoded on a channel automaton!

The same channel automaton can be used to encode $N_n$ for any $n \geq 3$. 

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

$x = 1, y := 0$

$y \geq 2, y := 0$

$x = 0, y = 2$

$[x] = 1, [y] = 1$
From networks of timed systems to channel automata

Example

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

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

\[ x = 0, y = 2 \]

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

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

The same channel automaton can be used to encode \( N^n \) for any \( n \geq 3 \).
From networks of timed systems to channel automata

Example

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

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

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

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

\[ \text{Bad} \]

\[ [\text{x}] = 2, \ [\text{y}] = 1 \]

\[ \begin{array}{ccccccc}
  x_0, x, y & x_1 & x_2 & x_3 & x_4 & x_5 & \text{Bad} \\
\end{array} \]

\[ \text{0} \quad \text{1} \]
From networks of timed systems to channel automata

Example

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

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

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

This can be encoded on a channel automaton!

The same channel automaton can be used to encode \( N_n \) for any \( n \geq 3 \).
From networks of timed systems to channel automata

Example

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

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

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

This can be encoded on a channel automaton!

The same channel automaton can be used to encode \( n \) for any \( n \geq 3 \).
From networks of timed systems to channel automata

Example

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

This can be encoded on a channel automaton!
From networks of timed systems to channel automata

Example

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

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

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

This can be encoded on a channel automaton!
Example

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

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

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

\[ x = 0, y = 2 \]

\[ \text{Bad} \]

This can be encoded on a channel automaton!

The same channel automaton can be used to encode \( N^n \) for any \( n \geq 3 \).
From networks of timed systems to CAROTs

\[
\mathcal{A}_\Delta \times \mathcal{A} \times \mathcal{B}_\varphi
\]

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

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

bounded information

unbounded information
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
BoundedMTL robust model-checking

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

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 $\langle \Delta \rangle^n$. 

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 $\langle \Delta \rangle^{N_0}$.

It is PSPACE-complete if the constants are in unary.
### 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.

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

**Theorem**

*BoundedMTL* robust model checking is EXPSPACE-complete.

It is PSPACE-complete if the constants are in unary.
CoFlatMTL robust model-checking

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

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

$\leq h$

$\geq 2^{|\varphi|} \cdot (2W + 1)$

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

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

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

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

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

Theorem

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

skip proof
CoFlatMTL robust model-checking

When $\varphi \in \text{CoFlatMTL}$, an execution of $[[A]]_{1/n} \times [[B \neg \varphi]]$ 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 $R^*(A \times F)$.

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

$\leq h$

$\geq 2^{|\varphi|} \cdot (2W+1)$

**Theorem**

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

**Theorem**

For any $n \geq 3$,

$$L(\langle A \rangle_{1/n}) \subseteq L(\langle N^n \rangle) \subseteq L(\langle A \rangle_{2/n})$$

**Proof.** The relation $(\ell, v) \prec (\ell, v\mid_X)$ is a simulation relation proving that $\langle N^n \rangle \sqsubseteq \langle A \rangle_{2/n}$.

Similarly, the relation $(\ell, v\mid_X) \prec (\ell, v)$ for $v$ satisfying $v(x_{i+1}) - v(x_i) = 1/n$ is a simulation relation. □
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**: active parts: cycle-bounded reachability in $C_{A}$.
  Inactive parts: the behaviour of $B\neg \psi$ is the behaviour of a non-deterministic $\B"uchi$ automaton $F\neg \psi$.
  Final part: $\B"uchi$ condition of $F\neg \psi$ in $R^{*}(A \times F\neg \psi)$. 

\[ h \geq 2^{\|\psi\|}(2W+1) \]
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:**
  
  ![Diagram](image)

  - **Active parts:** cycle-bounded reachability in $C_A$.
  - **Inactive parts:** the behaviour of $B_{\neg \varphi}$ is the behaviour of a non-deterministic Büchi automaton $F_{\neg \varphi}$. Reachability in $R^*(A \times F_{\neg \varphi})$.
  - **Final part:** Büchi condition of $F_{\neg \varphi}$ in $R^*(A \times F_{\neg \varphi})$. 

\[
\begin{align*}
\omega_1 & \leq h \\
\omega_4 & \geq 2 |\varphi| \cdot (2W + 1)
\end{align*}
\]
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.
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.