Robust Model-Checking of Timed Automata

Ocan Sankur ¹ ²

March - September 2010

(Joint work with Patricia Bouyer-Decitre ² and Nicolas Markey ²)

¹ École Normale Supérieure, Paris

² LSV, CNRS & École Normale Supérieure de Cachan
Timed Automata (TA)

Timed automata = Finite automata + Clocks. [Alur and Dill 1994]
Timed Automata (TA)

**Timed automata** = Finite automata + Clocks. [Alur and Dill 1994] Clocks grow continuously, all at the same rate. They are used to (de)activate the transitions of the automaton and can be reset when taking a transition.
Timed Automata (TA)

Timed automata $=$ Finite automata + Clocks. [Alur and Dill 1994] Clocks grow continuously, all at the same rate. They are used to (de)activate the transitions of the automaton and can be reset when taking a transition.

- **a**: $x \leq 2$ / $x := 0$
- **b**: $y \geq 2$ / $y := 0$
- **c**: $x = 0$ & $y \geq 2$

![Diagram](image-url)
Timed Automata (TA)

**Timed automata** = Finite automata + Clocks. [Alur and Dill 1994]

Clocks grow continuously, all at the same rate. They are used to (de)activate the transitions of the automaton and can be reset when taking a transition.

\[
a: x \leq 2 / x := 0
\]

\[
b: y \geq 2 / y := 0
\]

\[
c: x = 0 & y \geq 2
\]

---

**Exact semantics of TA**

Given a TA $\mathcal{A}$, the **exact semantics** of $\mathcal{A}$ is denoted by $[\mathcal{A}]$. 
Timed Automata (TA)

**Timed automata** = Finite automata + Clocks. [Alur and Dill 1994]

Clocks grow continuously, all at the same rate. They are used to (de)activate the transitions of the automaton and can be reset when taking a transition.

\[
\begin{align*}
\text{a: } & \ x \leq 2 \ / \ x := 0 \\
\text{b: } & \ y \geq 2 \ / \ y := 0 \\
\text{c: } & \ x = 0 \& y \geq 2
\end{align*}
\]

Exact semantics of TA

Given a TA $\mathcal{A}$, the **the exact semantics** of $\mathcal{A}$ is denoted by $\llbracket \mathcal{A} \rrbracket$.

A run of $\llbracket \mathcal{A} \rrbracket$ is as follows.

\[(q_0, (x = 0, y = 0))\]
Timed Automata (TA)

**Timed automata** = Finite automata + Clocks. [Alur and Dill 1994]

Clocks grow continuously, all at the same rate. They are used to (de)activate the transitions of the automaton and can be reset when taking a transition.

\[
\begin{align*}
\text{a: } & x \leq 2 / x := 0 \\
\text{b: } & y \geq 2 / y := 0 \\
\text{c: } & x = 0 \& y \geq 2
\end{align*}
\]

![Diagram of Timed Automaton](image)

**Exact semantics of TA**

Given a TA \( \mathcal{A} \), the **exact semantics** of \( \mathcal{A} \) is denoted by \( \llbracket \mathcal{A} \rrbracket \).

A run of \( \llbracket \mathcal{A} \rrbracket \) is as follows.

\[
(q_0, (x = 0, y = 0)) \xrightarrow{1.7} (q_0, (x = 1.7, y = 1.7))
\]
Timed Automata (TA)

Timed automata = Finite automata + Clocks. [Alur and Dill 1994]

Clocks grow continuously, all at the same rate. They are used to (de)activate the transitions of the automaton and can be reset when taking a transition.

```
a: x ≤ 2 / x := 0
b: y ≥ 2 / y := 0
c: x = 0 & y ≥ 2
```

Exact semantics of TA

Given a TA $\mathcal{A}$, the **exact semantics** of $\mathcal{A}$ is denoted by $[[\mathcal{A}]]$. A run of $[[\mathcal{A}]]$ is as follows.

$$(q_0, (x = 0, y = 0)) \xrightarrow{1.7} (q_0, (x = 1.7, y = 1.7)) \xrightarrow{a} (q_1, (x = 0, y = 1.7))$$
Timed Automata (TA)

Timed automata = Finite automata + Clocks. [Alur and Dill 1994]
Clocks grow continuously, all at the same rate. They are used to (de)activate the transitions of the automaton and can be reset when taking a transition.

\[
\begin{align*}
q_0 &\xrightarrow{a} q_1 \\
q_0 &\xleftarrow{b} q_0 \\
q_1 &\xrightarrow{c} \text{error}
\end{align*}
\]

\begin{align*}
a: &x \leq 2 / x := 0 \\
b: &y \geq 2 / y := 0 \\
c: &x = 0 \& y \geq 2
\end{align*}

Exact semantics of TA

Given a TA $\mathcal{A}$, the **the exact semantics** of $\mathcal{A}$ is denoted by $\llbracket \mathcal{A} \rrbracket$.

A run of $\llbracket \mathcal{A} \rrbracket$ is as follows.

\[
\begin{align*}
(q_0, (x = 0, y = 0)) &\xrightarrow{1.7} (q_0, (x = 1.7, y = 1.7)) \xrightarrow{a} (q_1, (x = 0, y = 1.7)) \\
&\xrightarrow{0.5} (q_1, (x = 0.5, y = 2.2))
\end{align*}
\]
Timed Automata (TA)

Timed automata = Finite automata + Clocks. [Alur and Dill 1994] Clocks grow continuously, all at the same rate. They are used to (de)activate the transitions of the automaton and can be reset when taking a transition.

\[
\begin{align*}
q_0 & \xrightarrow{a} q_0 & a: x \leq 2 & / & x := 0 \\
q_0 & \xrightarrow{b} q_1 & b: y \geq 2 & / & y := 0 \\
q_1 & \xrightarrow{c} q_1 & c: x = 0 & \& & y \geq 2 \\
q_1 & \xrightarrow{0.5} q_1 & & b & (x = 0.5, y = 2.2) \\
q_1 & \xrightarrow{1.7} q_0 & & a & (x = 0, y = 1.7) \\
q_0 & \xrightarrow{0.5} q_0 & & b & (x = 0.5, y = 0) \\
\end{align*}
\]

Exact semantics of TA

Given a TA \( \mathcal{A} \), the the exact semantics of \( \mathcal{A} \) is denoted by \([\mathcal{A}]\). A run of \([\mathcal{A}]\) is as follows.

\[
(q_0, (x = 0, y = 0)) \xrightarrow{1.7} (q_0, (x = 1.7, y = 1.7)) \xrightarrow{a} (q_1, (x = 0, y = 1.7)) \xrightarrow{0.5} (q_1, (x = 0.5, y = 2.2)) \xrightarrow{b} (q_0, (x = 0.5, y = 0)) \ldots
\]
Model-Checking Timed Automata

**Model-checking**: Given a TA $\mathcal{A}$, decide whether all runs of $\llbracket \mathcal{A} \rrbracket$ verify some property $P$, written $\mathcal{A} \models P$. 
**Model-checking**: Given a TA $A$, decide whether all runs of $\llbracket A \rrbracket$ verify some property $P$, written $A \models P$.

where $P$ is a LTL formula (such as a safety or liveness property).

**Theorem (Alur and Dill 1994)**

*Model-checking timed-automata against LTL formulae is PSPACE-complete.*

Industrial applications: audio/video, communication protocols, ...

Existing model-checking tools: Uppaal, Kronos, ...
Implementability of Timed Automata

**Problem:** The exact semantics of timed automata makes unrealistic assumptions:

- Systems have instant reaction time, $a \rightarrow 0.00001 \rightarrow b$.
- Clocks are infinitely precise. $\forall x \leq k$.

Ocan Sankur (ENS & ENS Cachan)
Robust Model-Checking of Timed Automata
September 7, 2010 4 / 15
Implementability of Timed Automata

**Problem:** The exact semantics of timed automata makes unrealistic assumptions:

- Systems have instant reaction time,
- clocks are infinitely precise.

\[
\begin{align*}
&\rightarrow 0.00001 \rightarrow b.
\end{align*}
\]

\[x \leq k\].

[De Wulf, Doyen and Raskin 2004] introduced the enlarged semantics of \(\mathcal{A}\), parameterized by \(\delta > 0\), taking into account these problems. \([\mathcal{A}]_\delta\) is obtained by relaxing all constraints by \(\delta\), i.e. each constraint of the form

\[x \leq k \quad x \geq k.\]
Implementability of Timed Automata

Problem: The exact semantics of timed automata makes unrealistic assumptions:

- Systems have instant reaction time, \( \rightarrow \)
- Clocks are infinitely precise.

[De Wulf, Doyen and Raskin 2004] introduced the enlarged semantics of \( A \), parameterized by \( \delta > 0 \), taking into account these problems.

\( [A]_\delta \) is obtained by relaxing all constraints by \( \delta \), i.e. each constraint of the form becomes

\[
x \leq k + \delta \quad x \geq k - \delta.
\]
Implementability of Timed Automata

**Problem:** The exact semantics of timed automata makes unrealistic assumptions:

- Systems have instant reaction time, \( a \xrightarrow{0.00001} b \).
- Clocks are infinitely precise. \( "x \leq k".\)

[De Wulf, Doyen and Raskin 2004] introduced the enlarged semantics of \( \mathcal{A} \), parameterized by \( \delta > 0 \), taking into account these problems. \( [\mathcal{A}]_\delta \) is obtained by relaxing all constraints by \( \delta \), i.e. each constraint of the form becomes

\[
 x \leq k + \delta \quad x \geq k - \delta.
\]

- This corresponds to the (over-approximation of the) implementation of \( \mathcal{A} \) in a simple micro-processor model, with finite precision and a nonzero reaction time.
- Fast micro-processor \( \Leftrightarrow \) small \( \delta \).
For $\delta = 0.1$, $[[A]]_{\delta}$ is defined by,

- **a:** $x \leq 2.1$ / $x := 0$
- **b:** $y \geq 1.9$ / $y := 0$
- **c:** $x \leq 0.1 \& y \geq 1.9$

**Diagram:**
- $q_0$ (start) to $q_1$
- $q_1$ to error
- $q_0$ to $q_1$
- $q_1$ to error
For $\delta = 0.1$, $[A]_\delta$ is defined by,

\begin{align*}
\text{a: } & x \leq 21 \land x := 0 \\
\text{b: } & y \geq 19 \land y := 0 \\
\text{c: } & x \leq 1 \land y \geq 19
\end{align*}

There is an equivalent timed automaton obtained by changing the scale of time (multiplying all constants by 10).
- For fixed $\delta$, $[A]_\delta$ is the exact semantics of a timed automaton.
Robust model-checking

Given $\mathcal{A}$ and a property $P$, does $[\mathcal{A}]_\delta$ verify $P$ for some $\delta > 0$? If it does, we write $\mathcal{A} \models P$. 
Robust model-checking

Given $A$ and a property $P$, does $[A]_\delta$ verify $P$ for some $\delta > 0$? If it does, we write $A \models P$.

Question Does $A \models P$ imply $A \models P$?
Robustness in Timed Automata - 3

Robust model-checking

Given $\mathcal{A}$ and a property $P$, does $[\mathcal{A}]_\delta$ verify $P$ for some $\delta > 0$?
If it does, we write $\mathcal{A} \models P$.

Question Does $\mathcal{A} \models P$ imply $\mathcal{A} \equiv P$?

No! There exists automata $\mathcal{A}$ such that $\text{Reach}([\mathcal{A}]) \subsetneq \text{Reach}([\mathcal{A}]_\delta)$ for any $\delta > 0$. 
Robustness in Timed Automata - 3

Robust model-checking

Given \( \mathcal{A} \) and a property \( P \), does \([\mathcal{A}]_\delta\) verify \( P \) for some \( \delta > 0 \)? If it does, we write \( \mathcal{A} \models P \).

**Question** Does \( \mathcal{A} \models P \) imply \( \mathcal{A} \equiv P \)?

**No!** There exists automata \( \mathcal{A} \) such that \( \text{Reach}(\lceil \mathcal{A} \rceil) \subset \text{Reach}(\lceil \mathcal{A} \rceil_\delta) \) for any \( \delta > 0 \).

An error state that is not reachable in \( \lceil \mathcal{A} \rceil \) may be reachable in the implementation.
Robust model-checking

Given $A$ and a property $P$, does $[A]_\delta$ verify $P$ for some $\delta > 0$? If it does, we write $A \models P$.

Question Does $A \models P$ imply $A \models P$? **No!** There exists automata $A$ such that $\text{Reach}([A]) \subsetneq \text{Reach}([A]_{\delta})$ for any $\delta > 0$.

An error state that is not reachable in $[A]$ may be reachable in the implementation.

Modeling $\rightarrow$ Verification $\rightarrow$ Implementation
Robust model-checking

Given $A$ and a property $P$, does $[A]_\delta$ verify $P$ for some $\delta > 0$? If it does, we write $A \models P$.

Question: Does $A \models P$ imply $A \models P$?

No! There exists automata $A$ such that $\text{Reach}([A]) \subsetneq \text{Reach}([A]_\delta)$ for any $\delta > 0$.

An error state that is not reachable in $[A]$ may be reachable in the implementation.

Modeling $\rightarrow$ Verification $\rightarrow$ Implementation

Not ok $\rightarrow$ ok $\rightarrow$ not ok
Background

- Robust model-checking of reachability properties is \textbf{PSPACE}-complete. 
  
  [Puri 1998] and [De Wulf, Doyen, Markey, Raskin 2004].
Background

Robust model-checking of reachability properties is PSPACE-complete.
[Puri 1998] and [De Wulf, Doyen, Markey, Raskin 2004].
Background

- Robust model-checking of reachability properties is \( \text{PSPACE}\)-complete. [Puri 1998] and [De Wulf, Doyen, Markey, Raskin 2004].
- Robust model-checking of co-Büchi (LTL) properties is \( \text{PSPACE}\)-complete [Bouyer, Markey, Reynier 2006].
- and a fragment of MTL is \( \text{EXPSPACE}\)-complete [Bouyer, Markey, Reynier 2008].
Background

- Robust model-checking of reachability properties is \text{PSPACE}\text{-complete}. [Puri 1998] and [De Wulf, Doyen, Markey, Raskin 2004].
- Robust model-checking of co-Büchi (LTL) properties is \text{PSPACE}\text{-complete} [Bouyer, Markey, Reynier 2006].
- and a fragment of MTL is \text{EXPSPACE}\text{-complete} [Bouyer, Markey, Reynier 2008].

All previous works are only valid for a \textbf{subclass of timed automata} (verifying the progress cycles hypothesis)
Background

- Robust model-checking of reachability properties is PSPACE-complete. [Puri 1998] and [De Wulf, Doyen, Markey, Raskin 2004].
- Robust model-checking of co-Büchi (LTL) properties is PSPACE-complete [Bouyer, Markey, Reynier 2006].
  and a fragment of MTL is EXPSPACE-complete [Bouyer, Markey, Reynier 2008].

All previous works are only valid for a subclass of timed automata (verifying the progress cycles hypothesis)

Progress cycles

A timed automaton verifies the **progress cycles hypothesis** if all cycles of its region automaton resets all clocks at least once.

⇒ “one cannot measure time spent in a loop”.
A program that waits for a special signal (ignoring other signals) violates this hypothesis.
Our results

All our results are valid for general timed automata.
Our results

All our results are valid for **general timed automata**.

- Reduction of **robust model-checking** against co-Büchi properties (LTL) to **model-checking in exact semantics** in optimal complexity (PSPACE).
Our results

All our results are valid for **general timed automata**.

- Reduction of **robust model-checking** against co-Büchi properties (LTL) to **model-checking in exact semantics** in optimal complexity (PSPACE).
- A new algorithm for robust model-checking of co-Büchi properties based on region automaton (generalizes [BMR06]).
Our results

All our results are valid for general timed automata.

- Reduction of robust model-checking against co-Büchi properties (LTL) to model-checking in exact semantics in optimal complexity (PSPACE).
- A new algorithm for robust model-checking of co-Büchi properties based on region automaton (generalizes [BMR06]).

Our proof techniques are original and are based on an encoding by channel machines, introduced in [Bouyer, Markey, Ouaknine, Worrell 2007].
Our results

All our results are valid for **general timed automata**.

- **Reduction** of robust model-checking against co-Büchi properties (LTL) to model-checking in exact semantics in optimal complexity (PSPACE).

- A new algorithm for robust model-checking of co-Büchi properties based on region automaton (generalizes [BMR06]).

Our proof techniques are original and are based on an encoding by **channel machines**, introduced in [Bouyer, Markey, Ouaknine, Worrell 2007].
Encoding of $\llbracket A \rrbracket_\delta$ explained (by example)

**Goal:** For any $A$ and $\delta \in [0, 1]$, define a **finite-state** machine $C_A(N)$ with a FIFO **channel**, parameterized by $N \in \mathbb{N}$, that captures the behaviour of $\llbracket A \rrbracket_\delta$. 
Encoding of $[\mathcal{A}]_\delta$ explained (by example)

**Goal:** For any $\mathcal{A}$ and $\delta \in [0, 1]$, define a **finite-state** machine $C_{\mathcal{A}}(N)$ with a FIFO **channel**, parameterized by $N \in \mathbb{N}$, that captures the behaviour of $[\mathcal{A}]_\delta$.

Let be a state of $\mathcal{A}$ (where $\lfloor x \rfloor = 1$, $\lfloor y \rfloor = 2$, $\lfloor z \rfloor = 0$).
Encoding of $[\mathcal{A}]_\delta$ explained (by example)

**Goal:** For any $\mathcal{A}$ and $\delta \in [0, 1]$, define a **finite-state** machine $C_\mathcal{A}(N)$ with a FIFO **channel**, parameterized by $N \in \mathbb{N}$, that captures the behaviour of $[\mathcal{A}]_\delta$.

<table>
<thead>
<tr>
<th>$\Delta_0$</th>
<th>$\Delta_1$</th>
<th>$\Delta_2$</th>
<th>$\Delta_3$</th>
<th>$\Delta_4$</th>
<th>$\Delta_5$</th>
<th>$\Delta_6$</th>
<th>$\Delta_7$</th>
<th>$\Delta_8$</th>
<th>$\Delta_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Add $N$ new clocks that are regularly distributed in $[0, 1]$ and that have values mod 1.

$C_\mathcal{A}(N)$ encodes the **regions** of the states of $\mathcal{A} + \{\Delta_0, \ldots, \Delta_{N-1}\}$ using a **discrete state** and a **channel**.
Encoding of $[A]_\delta$ explained (by example)

<table>
<thead>
<tr>
<th>$\Delta_0$</th>
<th>$\Delta_1$</th>
<th>$\Delta_2$</th>
<th>$\Delta_3$</th>
<th>$\Delta_4$</th>
<th>$\Delta_5$</th>
<th>$\Delta_6$</th>
<th>$\Delta_7$</th>
<th>$\Delta_8$</th>
<th>$\Delta_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\times$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

head $\rightarrow$ $\Delta x \Delta \Delta \Delta \Delta \Delta y z \Delta \Delta \Delta$ $\leftarrow$ tail $([x] = 1, [y] = 2, [z] = 0)$. 

channel discrete state
Encoding of $\llbracket A \rrbracket_\delta$ explained (by example)

\[\begin{array}{cccccccccc}
\Delta_0 & \Delta_1 & \Delta_2 & \Delta_3 & \Delta_4 & \Delta_5 & \Delta_6 & \Delta_7 & \Delta_8 & \Delta_9 \\
0 & x & & & & & y & z & & \\
\end{array}\]

Delay of 0.04 time units

$C_A(N) = \Delta x \Delta \Delta \Delta \Delta \Delta \Delta y z \Delta \Delta \Delta$ ( $|x| = 1$, $|y| = 2$, $|z| = 0$).

Ocan Sankur (ENS & ENS Cachan)
Robust Model-Checking of Timed Automata
September 7, 2010 9 / 15
Encoding of $\mathcal{A}_\delta$ explained (by example)

Delay of 0.02 time units

$C_A(N): \Delta \Delta x \Delta \Delta \Delta \Delta \Delta \Delta y z \Delta \Delta \ (\lfloor x \rfloor = 1, \lfloor y \rfloor = 2, \lfloor z \rfloor = 0)$.

Rule: When a $\Delta$ is read from the channel, write it back into the channel.
Encoding of $\lceil A \rceil_\delta$ explained (by example)

<table>
<thead>
<tr>
<th>$\Delta_7$</th>
<th>$\Delta_8$</th>
<th>$\Delta_9$</th>
<th>$\Delta_0$</th>
<th>$\Delta_1$</th>
<th>$\Delta_2$</th>
<th>$\Delta_3$</th>
<th>$\Delta_4$</th>
<th>$\Delta_5$</th>
<th>$\Delta_6$</th>
</tr>
</thead>
</table>
| y          | z           | x           | yz          | $\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\D
Guard $y \leq k$ is satisfied if

$$|y| \leq k - 1$$

or

$$|y| = k \text{ and } \Delta y \leq \Delta^1$$
Guard $y \leq k$ is satisfied if

\[
\lfloor y \rfloor \leq k - 1
\]

or

\[
\lfloor y \rfloor = k \text{ and } \Delta y \leq \Delta^1
\]

From the encoding, we know that $|y - \lfloor y \rfloor| \leq \frac{2}{N}$.

- Small $\delta \Leftrightarrow$ large $N$. 
Relation between $C_A(N)$ and $[A]_\delta$

Lemma (Simulation lemma - Adapted from BMR08)

$$[A]_{\frac{1}{N}} \subseteq C_A(N) \subseteq [A]_{\frac{2}{N}}$$

*Valid in our case, with no “progress cycles hypothesis”.*

$\{C_A(N)\}_{N>0}$ can be used to study robust linear properties of $\{[A]_{\frac{1}{N}}\}_{N>0}$.
Relation between $C_A(N)$ and $[A]_\delta$

**Lemma (Simulation lemma - Adapted from BMR08)**

$[A]_{1/N} \subseteq C_A(N) \subseteq [A]_{2/N}$

Valid in our case, with no “progress cycles hypothesis”.

$\{C_A(N)\}_{N>0}$ can be used to study robust linear properties of $\{[A]_{1/N}\}_{N>0}$.

**Result of BMR’08**: A robust MC algorithm against coFlat-MTL (a timed logic that subsumes LTL) in EXPSPACE. They make a limited use of this encoding (only for bounded executions). Proofs mix $C_A(N)$ and $[A]_\delta$. 
Relation between $C_A(N)$ and $[A]_\delta$

Lemma (Simulation lemma - Adapted from BMR08)

$$[A]_N^1 \subseteq C_A(N) \subseteq [A]_N^2$$

Valid in our case, with no “progress cycles hypothesis”.

$\{C_A(N)\}_{N>0}$ can be used to study robust linear properties of $\{[A]_N^1\}_{N>0}$.

Result of BMR’08: A robust MC algorithm against coFlat-MTL (a timed logic that subsumes LTL) in EXPSPACE. They make a limited use of this encoding (only for bounded executions). Proofs mix $C_A(N)$ and $[A]_\delta$.

Our work: We develop proof techniques based entirely on this encoding.

- A finer analysis of the enlarged semantics w.r.t untimed properties
- and a study of non-progress cycles,

yields a reduction to classical model-checking against (untimed) co-Büchi properties for general timed automata (PSPACE).
Reduction to classical model-checking

**Theorem**

There exists $N_0 > 0$ (of order $2^{|A|}$), such that

$$\exists N > 0, C_A(N) \models P \iff C_A(N_0) \models P.$$ 

“$C_A(N_0)$ captures the behaviours of all $C_A(N)$”.
Reduction to classical model-checking

Theorem

There exists $N_0 > 0$ (of order $2^{|A|}$), such that

$$\exists N > 0, C_{A}(N) \models P \iff C_{A}(N_0) \models P.$$  

“$C_{A}(N_0)$ captures the behaviours of all $C_{A}(N)$”.

- By simulation lemma, $\llbracket A \rrbracket_{1/2N_0}$ captures the behaviours of all $\llbracket A \rrbracket_\delta$. And $\llbracket A \rrbracket_{1/2N_0}$ is a timed automaton of size $O(|A|)$. 
Reduction to classical model-checking

**Theorem**

There exists $N_0 > 0$ (of order $2^{|A|}$), such that

$$\exists N > 0, C_A(N) \models P \iff C_A(N_0) \models P.$$

“$C_A(N_0)$ captures the behaviours of all $C_A(N)$”.

By simulation lemma, $\llbracket A \rrbracket_{\frac{1}{2N_0}}$ captures the behaviours of all $\llbracket A \rrbracket_\delta$. And $\llbracket A \rrbracket_{\frac{1}{2N_0}}$ is a timed automaton of size $O(|A|)$.

**Robust Model-checking Algorithm**

Let $A'$ obtained from $A$ by changing time scale ($\times 2N_0$). Apply classical model-checking algorithm to $A'$. 
Proof idea of the reduction

Proof of
\[ \exists N > 0, C_A(N) \models P \iff C_A(N_0) \models P. \]

One direction is obvious (from right to left). The other direction is equivalent to
\[ C_A(N_0) \not\models P \Rightarrow \forall N > 0, C_A(N) \not\models P. \]  \hspace{1cm} (1)
Proof idea of the reduction

Proof of

\[ \exists N > 0, \mathcal{C}_A(N) \models P \iff \mathcal{C}_A(N_0) \models P. \]

One direction is obvious (from right to left). The other direction is equivalent to

\[ \mathcal{C}_A(N_0) \not\models P \Rightarrow \forall N > 0, \mathcal{C}_A(N) \not\models P. \]

(1)

Proof of (1) in two steps.

- For all \( 0 < K < N_0 \), \( \mathcal{C}_A(N_0) \subseteq \mathcal{C}_A(N_0 - K) \) (easy),
Proof idea of the reduction

Proof of

\[ \exists N > 0, \mathcal{C}_A(N) \models P \iff \mathcal{C}_A(N_0) \models P. \]

One direction is obvious (from right to left). The other direction is equivalent to

\[ \mathcal{C}_A(N_0) \not\models P \Rightarrow \forall N > 0, \mathcal{C}_A(N) \not\models P. \]  \hfill (1)

\textbf{Proof of (1) in two steps.}

- For all \( 0 < K < N_0 \), \( \mathcal{C}_A(N_0) \subseteq \mathcal{C}_A(N_0 - K) \) (easy), proves (1) for \( 1, \ldots, N_0 - 1 \)
Proof idea of the reduction

Proof of

\[ \exists N > 0, \mathcal{C}_A(N) \models P \Leftrightarrow \mathcal{C}_A(N_0) \models P. \]

One direction is obvious (from right to left). The other direction is equivalent to

\[ \mathcal{C}_A(N_0) \not\models P \Rightarrow \forall N > 0, \mathcal{C}_A(N) \not\models P. \quad (1) \]

Proof of (1) in two steps.

- For all \( 0 < K < N_0 \), \( \mathcal{C}_A(N_0) \sqsubseteq \mathcal{C}_A(N_0 - K) \) (easy), proves (1) for \( 1, \ldots, N_0 - 1 \)

- For any run \( \pi \) of \( \mathcal{C}_A(N) \), there exists a run \( \pi' \) of \( \mathcal{C}_A(N + 1) \) that verifies the same co-Büchi properties (difficult).
Proof idea of the reduction

Proof of

$$\exists N > 0, \mathcal{A}(N) \models P \iff \mathcal{A}(N_0) \models P.$$  

One direction is obvious (from right to left). The other direction is equivalent to

$$\mathcal{A}(N_0) \not\models P \Rightarrow \forall N > 0, \mathcal{A}(N) \not\models P. \quad (1)$$

Proof of (1) in two steps.

- For all $0 < K < N_0$, $\mathcal{A}(N_0) \subseteq \mathcal{A}(N_0 - K)$ (easy), proves (1) for $1, \ldots, N_0 - 1$

- For any run $\pi$ of $\mathcal{A}(N)$, there exists a run $\pi'$ of $\mathcal{A}(N + 1)$ that verifies the same co-Büchi properties (difficult). proves (1) for $N_0 + 1, \ldots, \infty$

→ Pumping lemma (Main lemma, see report).
Pumping lemma: simple case by example

- Delay transitions.

\[ C_A(N) \triangleq \Delta \Delta x \Delta \Delta \Delta y \Delta \Delta \]
\[ C_A(N + 1) \triangleq \Delta \Delta \Delta x \Delta \Delta \Delta y \Delta \Delta \Delta \]
Pumping lemma: simple case by example

- Delay transitions.

\[ C_A(N) \]
\[ \begin{align*}
\Delta\Delta\times\Delta\Delta\Delta y\Delta\Delta \\
\Delta\Delta\Delta\Delta\times\Delta\Delta\Delta y
\end{align*} \]

\[ C_A(N + 1) \]
\[ \begin{align*}
\Delta\Delta\times\Delta\Delta\Delta y\Delta\Delta\Delta \\
\Delta\Delta\Delta\Delta\times\Delta\Delta\Delta\Delta y
\end{align*} \]
Pumping lemma: simple case by example

- Delay transitions.

\[ C_A(N) \]
\[ \Delta \Delta x \Delta \Delta \Delta y \Delta \Delta \]
\[ \Delta \Delta \Delta \Delta x \Delta \Delta \Delta y \]
\[ C_A(N + 1) \]
\[ \Delta \Delta x \Delta \Delta \Delta y \Delta \Delta \Delta \]
\[ \Delta \Delta \Delta \Delta x \Delta \Delta \Delta y \]

- Discrete transitions.

**Special case:** If all \( \Delta \)-blocks are of size \( \geq 2 \), then all guards satisfied in \( C_A(N) \) are also satisfied in \( C_A(N + 1) \).
Pumping lemma: simple case by example

- Delay transitions.

\[
\begin{align*}
C_A(N) & \quad C_A(N + 1) \\
\Delta\Delta\times\Delta\Delta\Delta y\Delta\Delta & \quad \Delta\Delta\times\Delta\Delta\Delta y\Delta\Delta\Delta \\
\Delta\Delta\Delta\Delta\times\Delta\Delta\Delta y & \quad \Delta\Delta\Delta\Delta\Delta\times\Delta\Delta\Delta y
\end{align*}
\]

- Discrete transitions.

**Special case:** If all \(\Delta\)-blocks are of size \(\geq 2\), then all guards satisfied in \(C_A(N)\) are also satisfied in \(C_A(N + 1)\).

\[
\begin{align*}
\Delta\Delta\Delta\Delta\Delta\Delta y & \quad \Delta\Delta\Delta\Delta\Delta\Delta y
\end{align*}
\]
Pumping lemma: simple case by example

- Delay transitions.

\[ \mathcal{C}_A(N) \]
\[ \Delta \Delta \times \Delta \Delta \Delta \Delta y \Delta \Delta \]
\[ \Delta \Delta \Delta \Delta \times \Delta \Delta \Delta y \]

\[ \mathcal{C}_A(N + 1) \]
\[ \Delta \Delta \times \Delta \Delta \Delta \Delta y \Delta \Delta \Delta \]
\[ \Delta \Delta \Delta \Delta \times \Delta \Delta \Delta \Delta y \]

- Discrete transitions.

**Special case:** If all \( \Delta \)-blocks are of size \( \geq 2 \), then all guards satisfied in \( \mathcal{C}_A(N) \) are also satisfied in \( \mathcal{C}_A(N + 1) \).

\[ \Delta \Delta \Delta \Delta \times \Delta \Delta \Delta \Delta y \]
\[ x \Delta \Delta \Delta \Delta \Delta \Delta \Delta y \]

\[ \Delta \Delta \Delta \Delta \times \Delta \Delta \Delta \Delta \Delta y \]
\[ x \Delta \Delta \Delta \Delta \Delta \Delta \Delta \Delta y \]

*Our proofs are based on watching the evolution of the sizes of \( \Delta \)-blocks.*
Conclusion

- **Reduction to classical model-checking.**
  - Well-known model-checking theory in exact semantics.
  - No progress cycles hypothesis: no restriction to modeling.

- New proof techniques based on **encoding by channel machines.**

- (Not presented here) New algorithm for robust model-checking: extended region-automaton (generalizes BMR06) to general TA.
Future Work

- Partial enlargement: only the guards of a given subset of clocks are enlarged.
  Preliminary results:
  - Enlarging all clocks but one = enlarging all clocks.
  - Enlarging all clocks but two \( \neq \) enlarging all clocks \( \neq \) exact semantics.

- Making automata robust: instead of analyzing automata can we modify a given automaton so that it becomes robust? (Preliminary results, also joint with Claus Thrane. See DOTS’10.)

- Robust controller synthesis using our techniques (based on encoding by channel machines).

PhD thesis at ENS Cachan on Robust Analysis and Synthesis of Timed Automata.