Implementability of Timed Controllers

Based on joint works with Karine Altisen, Patricia Bouyer, Martin De Wulf, Laurent Doyen, Jean-François Raskin, Pierre-Alain Reynier, and Stavros Tripakis

Nicolas Markey

Lab. Spécification et Vérification – ENS Cachan & CNRS

September 5, 2007
Controller Synthesis and Implementation

system:

property:

\[ G(\text{request} \Rightarrow F\text{grant}) \]
Controller Synthesis and Implementation

system:

property:

G(request ⇒ Fgrant)
Controller Synthesis and Implementation

system:

⇒

property:

G(request ⇒ grant)

controller synthesis

G(request ⇒ Fgrant)
Controller Synthesis and Implementation

system:

⇒

property:

G(request ⇒ Fgrant)

controller synthesis

yes/no
Controller Synthesis and Implementation

system:

⇒

property:

G(request \Rightarrow F_{grant})

controller synthesis

yes/no
Implementatibility 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;

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 lead to 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 lead to 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 $n$;
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 lead to bad behaviors.
The red state can be avoided;
But this would require to prevent time to elapse.
Implementability of Timed Controllers

Examples (Cassez et al., 2002)

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

<table>
<thead>
<tr>
<th>loc.</th>
<th>\ell_0</th>
<th>\ell_1</th>
<th>\ell_2</th>
<th>\ell_0</th>
<th>\ell_1</th>
<th>\ell_2</th>
<th>\ell_0</th>
<th>\ell_1</th>
<th>\ell_2</th>
<th>\ell_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Implementability of Timed Controllers

Examples (Cassez et al., 2002)

\[
\begin{align*}
\ell_0 & \xrightarrow{x=1} \ell_1 \\
& \xleftarrow{x:=0} \\
\ell_1 & \xrightarrow{z>0, y:=0, y=1} \ell_2 \\
& \xleftarrow{z:=0} \\
\ell_3 & \xrightarrow{x>1} \ell_0 \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>loc.</th>
<th>$\ell_0$</th>
<th>$\ell_1$</th>
<th>$\ell_2$</th>
<th>$\ell_0$</th>
<th>$\ell_1$</th>
<th>$\ell_0$</th>
<th>$\ell_1$</th>
<th>$\ell_2$</th>
<th>$\ell_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y$</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$z$</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Implementability of Timed Controllers

Examples (Cassez et al., 2002)

\[
\begin{align*}
x & = 1 \\
x & := 0 \\
x & > 1 \\
z & := 0 \\
z & > 0 \\
y & := 0 \\
y & = 1
\end{align*}
\]

<table>
<thead>
<tr>
<th>loc.</th>
<th>$\ell_0$</th>
<th>$\ell_1$</th>
<th>$\ell_2$</th>
<th>$\ell_0$</th>
<th>$\ell_1$</th>
<th>$\ell_2$</th>
<th>$\ell_0$</th>
<th>$\ell_1$</th>
<th>$\ell_2$</th>
<th>$\ell_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y$</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$z$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Implementability of Timed Controllers

Examples (Cassez et al., 2002)

\[
\begin{align*}
\ell_0 & \xrightarrow{x=1} \ell_1 \\
\ell_0 & \xleftarrow{x:=0} \ell_0 \\
\ell_0 & \xrightarrow{x>1} \ell_3 \\
\ell_3 & \xrightarrow{z:=0} \ell_2 \\
\ell_2 & \xrightarrow{y:=0} \ell_1 \\
\ell_1 & \xrightarrow{y=1} \ell_0 \\
\ell_2 & \xrightarrow{z>0} \ell_2 \\
\ell_0 & \xrightarrow{\text{loc.}} \ell_0
\end{align*}
\]

<table>
<thead>
<tr>
<th>loc.</th>
<th>(\ell_0)</th>
<th>(\ell_1)</th>
<th>(\ell_2)</th>
<th>(\ell_0)</th>
<th>(\ell_1)</th>
<th>(\ell_2)</th>
<th>(\ell_0)</th>
<th>(\ell_1)</th>
<th>(\ell_2)</th>
<th>(\ell_0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(y)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(z)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Implementability of Timed Controllers

Examples (Cassez et al., 2002)

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

\[
\begin{array}{c|cccccccccc}
\text{loc.} & \ell_0 & \ell_1 & \ell_2 & \ell_0 & \ell_1 & \ell_2 & \ell_0 & \ell_1 & \ell_2 & \ell_0 \\
\hline
x & 0 & 0 & 0 & \epsilon_1 & & & & & & \\
y & 0 & 1 & 1 & 0 & & & & & & \\
z & 0 & 1 & 0 & \epsilon_1 & & & & & & \\
\end{array}
\]
Implementability of Timed Controllers

Examples (Cassez et al., 2002)

\[
\begin{align*}
\ell_0 & \xrightarrow{x=1} \ell_1 \\
\ell_0 & \xleftarrow{x:=0} \ell_1 \\
\ell_0 & \xrightarrow{y=1} \ell_2 \\
\ell_0 & \xrightarrow{z:=0} \ell_2 \\
\ell_0 & \xrightarrow{x>1} \ell_3 \\
\ell_0 & \xrightarrow{z>0} \ell_3 \\
\ell_0 & \xrightarrow{y:=0} \ell_3 \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>loc.</th>
<th>$\ell_0$</th>
<th>$\ell_1$</th>
<th>$\ell_2$</th>
<th>$\ell_0$</th>
<th>$\ell_1$</th>
<th>$\ell_2$</th>
<th>$\ell_0$</th>
<th>$\ell_1$</th>
<th>$\ell_2$</th>
<th>$\ell_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$\epsilon_1$</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$1 - \epsilon_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$z$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$\epsilon_1$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Examples (Cassez et al., 2002)

Implementability of Timed Controllers

- \( x = 1 \)
- \( x := 0 \)
- \( z > 0 \)
- \( y = 1 \)
- \( z := 0 \)
- \( y := 0 \)

<table>
<thead>
<tr>
<th>loc.</th>
<th>( l_0 )</th>
<th>( l_1 )</th>
<th>( l_2 )</th>
<th>( l_0 )</th>
<th>( l_1 )</th>
<th>( l_2 )</th>
<th>( l_0 )</th>
<th>( l_1 )</th>
<th>( l_2 )</th>
<th>( l_0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( \epsilon_1 )</td>
<td>0</td>
<td>( \epsilon_1 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( y )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1 - ( \epsilon_1 )</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( z )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>( \epsilon_1 )</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Implementability of Timed Controllers

Examples (Cassez et al., 2002)

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

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

\[ z = 0 \quad z > 0 \]

\[ \ell_0 \quad \ell_1 \quad \ell_2 \quad \ell_3 \]

<table>
<thead>
<tr>
<th>loc.</th>
<th>\ell_0</th>
<th>\ell_1</th>
<th>\ell_2</th>
<th>\ell_3</th>
<th>\ell_0</th>
<th>\ell_1</th>
<th>\ell_2</th>
<th>\ell_0</th>
<th>\ell_1</th>
<th>\ell_2</th>
<th>\ell_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\epsilon_1</td>
<td>0</td>
<td>\epsilon_1</td>
<td>\epsilon_1 + \epsilon_2</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1 - \epsilon_1</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>\epsilon_1</td>
<td>1</td>
<td>0</td>
<td>\epsilon_2</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
It can be proved that this protocol enforces mutual exclusion in the critical (red) state.

Any imprecise implementation will fail to fulfil that property.
Outline of the talk

1. Introduction

2. Modeling the execution platform [Altisen & Tripakis, 2005]

3. A semantical approach [De Wulf et al., 2004]

4. Conclusions
Outline of the talk

1. Introduction

2. Modeling the execution platform [Altisen & Tripakis, 2005]

3. A semantical approach [De Wulf et al., 2004]

4. Conclusions
Modeling the execution platform [Altisen & Tripakis, 2005]

The automaton $A$ is now a discrete automaton, using input variables given by the platform; the automaton $P$ is a timed automaton that triggers $A$ (modeling a digital CPU), and sends input variables to $A$ depending on the values of the variables in $Env$.
Modeling the execution platform [Altisen & Tripakis, 2005]

The automaton $\mathcal{A}$ is now a discrete automaton, using input variables given by the platform;

The automaton $\mathcal{P}$ is a timed automaton that triggers $\mathcal{A}$ (modeling a digital CPU), and sends input variables to $\mathcal{A}$ depending on the values of the variables in Env;
Modeling the execution platform [Altisen & Tripakis, 2005]

(“digitized”) model \( \text{Prog}(A) \)

platform \( P \)

variables, environment

clocks of the model

internal clock

\[ \text{trig!} \]
1. Transforming $\mathcal{A}$ into $\text{Prog}(\mathcal{A})$.

- $\text{trig!}$ is an input event allowing $\mathcal{A}$ to perform one step;

- the value of a clock is the difference between the current value of the internal clock ($\text{now}$) and the date at which the clock was last reset:

  "$x > 2$" becomes "$\text{now} - x > 2$"

  "$x := 0$" becomes "$x := \text{now}$"
1. Transforming $A$ into $\text{Prog}(A)$.

**Example (Fischer's Mutual Exclusion Protocol)**

- $S_1$:
  - trig?
  - id:=0

- $C_1$:
  - trig?

- $W_1$:
  - trig?, id=1, now−$x_1$ > 2

- $r_1$:
  - trig?, $x_1$:=now, id:=0
  - trig?
  - now−$x_1$ ≤ 2
  - $x_1$:=now
  - id:=1

- $W_1$ (recursive transition):
  - trig?

- $S_1$ (recursive transition):
  - trig?
1. Transforming $\mathcal{A}$ into $\text{Prog}(\mathcal{A})$.

2. Modeling the digital CPU.

Examples

$x = \Delta$, $x := 0$

$\text{trig!}$
1. Transforming $\mathcal{A}$ into $\text{Prog}(\mathcal{A})$.

2. Modeling the digital CPU.

Examples

$\begin{align*}
x &= \Delta, \ x := 0 \\
\text{trig!}
\end{align*}$

$\begin{align*}
x &\in [\Delta_1, \Delta_2], \ x := 0 \\
\text{trig!}
\end{align*}$
1. Transforming $\mathcal{A}$ into $\text{Prog}(\mathcal{A})$.

2. Modeling the digital CPU.

3. Modeling the global clock.

Examples

\[
\begin{align*}
x &= \Delta, \quad x := 0 \\
now &:= now + \Delta \quad \text{( Wednesday, April 26, 2017, 17:05:26.2565475 )}
\end{align*}
\]
1. Transforming $A$ into $\text{Prog}(A)$.

2. Modeling the digital CPU.

3. Modeling the global clock.

Examples

- $x = \Delta$, $x := 0$
  - now := now $+$ $\Delta$
- $x \in [\Delta - \varepsilon, \Delta + \varepsilon]$, $x := 0$
  - now := now $+$ $\Delta$
Modeling the execution platform [Altisen & Tripakis, 2005]

1. Transforming $A$ into $\text{Prog}(A)$.

2. Modeling the digital CPU.

3. Modeling the global clock.

4. Modeling the input/output variables.

- delays for reading variables...
- lock mechanism for writing variables...
Modeling the execution platform [Altisen & Tripakis, 2005]

1. Transforming $\mathcal{A}$ into $\text{Prog}(\mathcal{A})$.

2. Modeling the digital CPU.

3. Modeling the global clock.

4. Modeling the input/output variables.

5. Classical verification techniques on the product of those automata.
Pros and cons of this approach

- **Pros:**
  - **Very expressive:** the platform can be described with many details;
  - **Relies on classical techniques:** the verification step is applied on standard timed automata. Existing tools can be used.
Pros and cons of this approach

**Pros:**
- **Very expressive:** the platform can be described with many details;
- **Relies on classical techniques:** the verification step is applied on standard timed automata. Existing tools can be used.

**Cons:**
- **Formal meaning?:** if the model satisfies some property, what does it *really* mean?
- **Faster is better?:** we expect that a program proved to be implementable on a given platform remains implementable on a faster platform. This property fails to hold with this modeling.
Outline of the talk

1. Introduction

2. Modeling the execution platform [Altisen & Tripakis, 2005]

3. A semantical approach [De Wulf et al., 2004]

4. Conclusions
A semantical approach [De Wulf et al., 2004]

1. “Implementation” Semantics

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

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

We assume that

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

We write $\mathbb{J}_{\Delta_P, \Delta_L}^{\text{Impl}}$ for the set of executions of a timed automaton $\mathcal{A}$ under this semantics.
A semantical approach [De Wulf et al., 2004]

1. “Implementation” Semantics

2. Enlarged Semantics
We define the enlarged semantics for timed automata, by enlarging guards on transitions by a small tolerance $\Delta$:

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

We write $\mathcal{[A]}_{\Delta}^{\text{AASAP}}$ for the set of executions of a timed automaton $\mathcal{A}$ under this semantics.
A semantical approach [De Wulf et al., 2004]

1. “Implementation” Semantics

2. Enlarged Semantics
We define the enlarged semantics for timed automata, by enlarging guards on transitions by a small tolerance $\Delta$:

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

$\Rightarrow$ We write $[[A]]^{\text{AASAP}}_{\Delta}$ for the set of executions of a timed automaton $A$ under this semantics.

Theorem ([DDR04])

If $\Delta > 3\Delta_L + 4\Delta_P$, then $[[A]]^{\text{Impl}}_{\Delta_P, \Delta_L} \subseteq [[A]]^{\text{AASAP}}_{\Delta}$. 
A semantical approach [De Wulf et al., 2004]

We focus on safety properties for the implementation semantics: we want to ensure that an implementation will avoid bad states.

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

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

\[ \text{R}(\mathcal{A}) = \bigcap_{\Delta > 0} \text{Reach}_{\Delta}(\mathcal{A}) \text{ is the set of reachable states under the AASAP semantics for any } \Delta > 0. \]
A semantical approach [De Wulf et al., 2004]

We focus on safety properties for the implementation semantics: we want to ensure that an implementation will avoid bad states.

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

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

\[ R(\mathcal{A}) = \bigcap_{\Delta > 0} \text{Reach}_{\Delta}(\mathcal{A}) \] 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 \( B \),

\[ R(\mathcal{A}) \cap B = \emptyset \quad \text{iff} \quad \exists \Delta > 0. \text{Reach}_{\Delta}(\mathcal{A}) \cap B = \emptyset. \]
An example: Standard semantics

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

\[ x \leq 2 \]
\[ y \geq 2 \]
\[ x = 0 \]
\[ y = 2 \]
An example: Standard semantics

\[ x = 1 \]
\[ y = 0 \]
\[ x \leq 2 \]
\[ x = 0 \]
\[ y \geq 2 \]
\[ y = 0 \]
\[ y = 2 \]
An example: Standard semantics

![Diagram with a grid and labeled points, connected by arrows with conditions on x and y values.]

- **Standard semantics**
- **Graph**
- **Points**
  - $x=1$, $y=0$ (Point A)
  - $x:=0$, $y:=0$ (Point B)
  - $x:=0$, $y=2$ (Point C)
  - $x\leq 2$, $y\geq 2$ (Transition between B and C)

**Conditions**
- $x=1$
- $y:=0$
- $x:=0$
- $y:=0$
- $x=0$
- $y=2$

**States**
- **A**
- **B**
- **C**
- **Bad**

**Arrows**
- From A to B
- From B to C
- From C to Bad

**Conditions on Arrows**
- From A to B: $x=1$, $y:=0$
- From B to C: $x:=0$, $y:=0$
- From C to Bad: $x=0$, $y=2$
An example: Standard semantics

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

\[ x \leq 2 \]

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

\[ y \geq 2 \]

\[ x = 0 \quad y = 2 \]
An example: Standard semantics

\[
\begin{align*}
\text{Bad} & \quad x = 1 \\
y & = 0
\end{align*}
\]

\[
\begin{align*}
\text{Bad} & \quad x = 0 \\
y & = 2
\end{align*}
\]
An example: Standard semantics
An example: Standard semantics
An example: Standard semantics

\[ x=1 \quad y=0 \]
\[ x=0 \quad y\geq 2 \quad y=0 \]
\[ x=0 \quad y=2 \]
An example: Standard semantics
An example with $\Delta > 0$
An example with $\Delta > 0$
An example with $\Delta > 0$
An example with $\Delta > 0$
An example with $\Delta > 0$
An example with $\Delta > 0$

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

$y := 0$

$x \leq 2+\Delta$

$y \geq 2-\Delta$

$y := 0$

$x \leq \Delta$

$y \in [2-\Delta, 2+\Delta]$
An example with $\Delta > 0$

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

$$y = 0$$

$$x \leq 2 + \Delta$$

$$x = 0$$

$$y \geq 2 - \Delta$$

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

$$y = 0$$

$$x \leq \Delta$$

$$\text{Bad}$$
An example with $\Delta > 0$

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

$$x \leq 2+\Delta$$

$$x:=0$$

$$y \geq 2-\Delta$$

$$y:=0$$

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

$$x \leq \Delta$$
An example with $\Delta > 0$
An example with $\Delta > 0$
An example with $\Delta > 0$
An example with $\Delta > 0$
An example with $\Delta > 0$
An example with $\Delta > 0$
An example with $\Delta > 0$
An example with \( \Delta \) very small

\[
x \in [1-\Delta; 1+\Delta] \\
y = 0 \leq x \leq 2+\Delta \\
y = 0 \geq y \geq 2-\Delta \\
x = \Delta \\
y \in [2-\Delta, 2+\Delta]
\]
An example with $\Delta$ very small

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

$$y := 0$$

$$x := 0$$

$$y \geq 2-\Delta$$

$$y := 0$$

$$x \leq 2+\Delta$$

$$x \leq \Delta$$
An example with $\Delta$ very small

\[
\begin{align*}
0 & \leq x \leq 2 + \Delta \\
y & \leq 2 - \Delta \\
y & \geq 2 - \Delta \\
x & \in [1 - \Delta; 1 + \Delta] \\
y & := 0
\end{align*}
\]
An example with $\Delta$ very small

$$\begin{align*}
&\text{Bad} \\
&x \in [1-\Delta; 1+\Delta] \\
&y := 0 \\
&x \leq 2 + \Delta \\
&y \geq 2 - \Delta \\
&y := 0 \\
&y \in [2-\Delta, 2+\Delta] \\
&x \leq \Delta
\end{align*}$$
An example with $\Delta$ very small

$0 \leq x \leq 2 + \Delta, \quad y \geq 2 - \Delta$

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

$x \leq 2 + \Delta, \quad y := 0$

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

Bad
An example with $\Delta$ very small

\[
\begin{align*}
0 & \leq x \leq 2 + \Delta \\
y & \in [2 - \Delta, 2 + \Delta]
\end{align*}
\]
An example with $\Delta$ very small

\[
\begin{align*}
0 & \leq x \leq 2+\Delta \\
0 & \leq y \leq 2-\Delta
\end{align*}
\]
An example with $\Delta$ very small

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

$$y:=0$$

$$x \leq 2+\Delta$$

$$x:=0$$

$$y \geq 2-\Delta$$

$$y:=0$$

$$x \leq \Delta$$

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

Bad
An example with $\Delta$ very small

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

$$y := 0$$

$$x \leq 2+\Delta$$

$$x \leq \Delta$$

$$x := 0$$

$$y \geq 2-\Delta$$

$$y := 0$$

$$y \in [2-\Delta,2+\Delta]$$
An example with $\Delta$ very small

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

$\text{Bad}$

Diagram:
- Node $a$ with $x \in [1-\Delta; 1+\Delta]$, $y:=0$.
- Node $b$ with $x:=0$, $y \geq 2-\Delta$.
- Node $c$ with $x \leq \Delta$, $y \in [2-\Delta, 2+\Delta]$.
- Arrows connecting $a$ to $b$, $b$ to $c$, and $c$ to $\text{Bad}$. 
An example with $\Delta$ very small

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

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

\[ y \geq 2-\Delta, \quad y \in [2-\Delta; 2+\Delta], \quad y = 0 \]

\[ b \rightarrow c \rightarrow \text{Bad} \]
An example with $\Delta$ very small

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

\[ y \geq 2 - \Delta \quad y \in [2-\Delta, 2+\Delta] \]
An example with $\Delta$ very small

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

$$y := 0$$

$$x \leq 2+\Delta$$

$$y \geq 2-\Delta$$

$$y := 0$$

$$x \leq \Delta$$
An example with $\Delta$ very small
An example with $\Delta$ very small

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

$y = 0$

$x \leq 2 + \Delta$

$y \geq 2 - \Delta$

$y = 0$

$x \leq \Delta$

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

$\text{Bad}$
Difference between $\mathcal{A}$ and $R(\mathcal{A})$

Reach($\mathcal{A}$)

- $x \leq 2$
- $y \geq 2$
- $x = 1$
- $y = 0$

$R(\mathcal{A})$

- $x \leq \Delta$
- $y \in [2-\Delta, 2+\Delta]$
- $x = 0$
- $y = 0$
- $y = 0$

Bad
An algorithm for computing $R(A)$

Input: A Timed Automaton $A$
Output: The set $R(A)$
An algorithm for computing $R(\mathcal{A})$

Input: A Timed Automaton $\mathcal{A}$
Output: The set $R(\mathcal{A})$

1. build the region graph $G$ of $\mathcal{A}$;
An algorithm for computing $R(\mathcal{A})$

Input: A Timed Automaton $\mathcal{A}$
Output: The set $R(\mathcal{A})$

1. build the region graph $G$ of $\mathcal{A}$;
2. compute $\text{SCC}(G) = \text{the set of strongly connected components of } G$;
An algorithm for computing $R(A)$

Input: A Timed Automaton $A$
Output: The set $R(A)$

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

6. return($J$);
An algorithm for computing $R(A)$

Input: A Timed Automaton $A$
Output: The set $R(A)$

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

6. return($J$);
An algorithm for computing \( R(\mathcal{A}) \)

Input: A Timed Automaton \( \mathcal{A} \)
Output: The set \( R(\mathcal{A}) \)

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 \( \exists \ S \in \text{SCC}(G). \ S \not\subset J \) and \( S \cap J \neq \emptyset \),
   \( J := J \cup S \);
   \( J := \text{Reach}(G, J) \);
6. return(\( J \));
Lemma

Let \( A \) be a TA with \( n \) clocks, \( \Delta \in \mathbb{Q}^{>0} \), and \( \delta = \Delta/n \). Let \( u \) be a valuation s.t. there exists a trajectory \( \pi[0, T] \) in \([A]\) with \( \pi(0) = \pi(T) = u \). Let \( v \in [u] \cap B(u, \delta) \). Then there exists a trajectory from \( u \) to \( v \) in \([A]^{\Delta}\).

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. \( \square \)
Lemma

Let $A$ be a TA with $n$ clocks, $\Delta \in \mathbb{Q}^{>0}$, and $\delta = \Delta / n$. Let $u$ be a valuation s.t. there exists a trajectory $\pi[0, T]$ in $\mathbb{[A]}$ with $\pi(0) = \pi(T) = u$. Let $v \in [u] \cap B(u, \delta)$. Then there exists a trajectory from $u$ to $v$ in $\mathbb{[A]}^\Delta$.

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. □

Corollary

Let $A$ be a TA and $p = p_0p_1 \ldots p_k$ be a cycle in the region graph (i.e. $p_k = p_0$). For any $\Delta > 0$ and any $x, y \in p_0$, there exists a trajectory from $x$ to $y$. 
Lemma

Let $\mathcal{A}$ be a TA, $\delta \in \mathbb{R}^>0$ and $k \in \mathbb{N}$. There exists $D \in \mathbb{Q}^>0$ s.t. for all $\Delta \leq D$, any $k$-step trajectory $\pi' = (q'_0, t'_0)(q'_1, t'_1) \ldots (q'_k, t'_k)$ in $[\mathcal{A}]^\Delta$ can be approximated be a $k$-step trajectory $\pi = (q_0, t_0)(q_1, t_1) \ldots (q_k, t_k)$ in $[\mathcal{A}]$ with $\|q_i - q'_i\| \leq \delta$ for all $i$.

The proof involves parametric DBMs.
\[ J \supseteq R_\Delta(A) \]

**Lemma**

Let \( A \) be a TA, \( \delta \in \mathbb{R}^{>0} \) and \( k \in \mathbb{N} \). There exists \( D \in \mathbb{Q}^{>0} \) s.t. for all \( \Delta \leq D \), any \( k \)-step trajectory \( \pi' = (q'_0, t'_0)(q'_1, t'_1) \ldots (q'_k, t'_k) \) in \([A]_\Delta\) can be approximated by a \( k \)-step trajectory \( \pi = (q_0, t_0)(q_1, t_1) \ldots (q_k, t_k) \) in \([A]\) with \( \|q_i - q'_i\| \leq \delta \) for all \( i \).

The proof involves parametric DBMs.

**Corollary**

Let \( A \) be a TA with \( n \) clocks and \( W \) regions, \( \alpha < 1/(2n) \), and \( \Delta < \frac{\alpha}{2^2W \cdot (4n+2)} \). Let \( x \in J \) and \( y \) s.t. there exists a trajectory from \( x \) to \( y \) in \([A]_\Delta\). Then \( d(J, y) < \alpha \).
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the “progress-cycle” constraint. For instance:

![Diagram of a network with constraints](image)

```
x \leq \Delta, \ x:=0
y, z \geq 1 - \Delta
x \leq \Delta
x, y \leq \Delta, z \geq 1 - \Delta
x = y = 0, \ z = 1
```

```
x = 0, \ x:=0
y = z = 1
```

```
x = 0
```

```
x \leq \Delta, \ x:=0
```

```
x, y \leq \Delta, z \geq 1 - \Delta
```
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the “progress-cycle” constraint. For instance:
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the “progress-cycle” constraint. For instance:

\[
\begin{align*}
 x &\leq \Delta, \quad x := 0 \\
 y, z &\geq 1 - \Delta, \quad x \leq \Delta \\
 x, y &\leq \Delta, z \geq 1 - \Delta
\end{align*}
\]

\[
\begin{align*}
 x &= 0, \quad x := 0 \\
 y &= z = 1, \quad x = 0 \\
 x &= y = 0, z = 1
\end{align*}
\]
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the “progress-cycle” constraint. For instance:

\[
\begin{align*}
0 & \leq x \leq \Delta, \quad x := 0 \\
y, z & \geq 1 - \Delta \\
x & \leq \Delta \\
x, y & \leq \Delta, z \geq 1 - \Delta \\
\end{align*}
\]
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the “progress-cycle” constraint. For instance:

\[
\begin{align*}
x \leq \Delta, & \quad x := 0 \\
y, z \geq 1 - \Delta & \quad x \leq \Delta \\
x, y \leq \Delta, z \geq 1 - \Delta
\end{align*}
\]
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the “progress-cycle” constraint. For instance:

\[ x \leq \Delta, \ x := 0 \]
\[ y, z \geq 1 - \Delta \]
\[ x \leq \Delta \]
\[ x, y \leq \Delta, z \geq 1 - \Delta \]

\[ y = z = 1 \]
\[ x = 0 \]
\[ x = y = 0, z = 1 \]
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the “progress-cycle” constraint. For instance:

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

\[
y, z \geq 1 - \Delta \\
x \leq \Delta
\]

\[
x, y \leq \Delta, z \geq 1 - \Delta
\]

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

\[
x = y = 0, z = 1
\]
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the “progress-cycle” constraint. For instance:

\[
\begin{align*}
0 & \leq x & \leq \Delta, \quad x := 0 \\
y, z & \geq 1 - \Delta \\
x, y & \leq \Delta, z \geq 1 - \Delta
\end{align*}
\]

\[
\begin{align*}
x & = 0, \quad x := 0 \\
y, z & = 1 \\
x & = y = 0, z = 1
\end{align*}
\]
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the “progress-cycle” constraint. For instance:

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

\[ y, z \geq 1 - \Delta \]

\[ x \leq \Delta \]

\[ y = z = 1 \]

\[ x = y = 0, z = 1 \]
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the “progress-cycle” constraint. For instance:

\[
\begin{align*}
\text{Can we relax the assumption on cycles?} \\
\text{Our algorithm does not work if we relax the “progress-cycle” constraint. For instance:} \\
\end{align*}
\]
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the “progress-cycle” constraint. For instance:

$$x, y, z \leq \Delta, x:=0$$

$$y, z \geq 1 - \Delta$$

$$x \leq \Delta$$

$$x, y \leq \Delta, z \geq 1 - \Delta$$

$$x=0, x:=0$$

$$y = z = 1$$

$$x = y = 0, z = 1$$
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the “progress-cycle” constraint. For instance:
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the “progress-cycle” constraint. For instance:

\[
\begin{align*}
  x &\leq \Delta, \quad x := 0 \\
  y, z &\geq 1 - \Delta \\
  x &\leq \Delta \\
  x, y &\leq \Delta, z \geq 1 - \Delta
\end{align*}
\]
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the “progress-cycle” constraint. For instance:

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

\[ y, z \geq 1 - \Delta \]

\[ x \leq \Delta, \ y \leq 1 - \Delta, \ z \geq 1 - \Delta \]

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

\[ y = z = 1 \]

\[ x = y = 0, z = 1 \]
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the “progress-cycle” constraint. For instance:

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

\[ y, z \geq 1 - \Delta \]

\[ x, y \leq \Delta, z \geq 1 - \Delta \]

\[ x = y = 0, z = 1 \]
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the “progress-cycle” constraint. For instance:
Can we relax the assumption on cycles?

Our algorithm does not work if we relax the “progress-cycle” constraint. For instance:

\[
\begin{align*}
0 
\end{align*}
\]

\[
\begin{align*}
x & \leq \Delta, \ x := 0 \\
y, z & \geq 1 - \Delta \\
x & \leq \Delta \\
x, y & \leq \Delta, z \geq 1 - \Delta \\
\end{align*}
\]

\[
\begin{align*}
x & = 0, \ x := 0 \\
y & = z = 1 \\
x & = 0 \\
x & = y = 0, z = 1 \\
\end{align*}
\]
Extension with clock drifts

when $d$ time unit elapse, each clock is incremented by some value between $d \times (1 - \epsilon)$ and $d \times (1 + \epsilon)$. 
when \( d \) time unit elapse, each clock is incremented by some value between \( d \times (1 - \epsilon) \) and \( d \times (1 + \epsilon) \).
when $d$ time unit elapse, each clock is incremented by some value between $d \times (1 - \epsilon)$ and $d \times (1 + \epsilon)$.

\[
\begin{align*}
\text{a} & \quad \text{b} \\
\text{c} & \quad \text{Bad}
\end{align*}
\]
when $d$ time unit elapse, each clock is incremented by some value between $d \times (1 - \epsilon)$ and $d \times (1 + \epsilon)$.
Extension with clock drifts

when \( d \) time unit elapse, each clock is incremented by some value between \( d \times (1 - \epsilon) \) and \( d \times (1 + \epsilon) \).
Extension with clock drifts

when $d$ time unit elapse, each clock is incremented by some value between $d \times (1 - \epsilon)$ and $d \times (1 + \epsilon)$.
Extension with clock drifts

when \( d \) time unit elapse, each clock is incremented by some value between \( d \times (1 - \epsilon) \) and \( d \times (1 + \epsilon) \).
when $d$ time unit elapse, each clock is incremented by some value between $d \times (1 - \epsilon)$ and $d \times (1 + \epsilon)$. 

\[
\begin{align*}
\text{Bad} & \quad x=1 \quad y:=0 \\
\text{a} & \quad x:=0, y:=0 \\
\text{b} & \quad x\leq2 \\
\text{c} & \quad y\leq2 \\
\end{align*}
\]
when $d$ time unit elapse, each clock is incremented by some value between $d \times (1 - \epsilon)$ and $d \times (1 + \epsilon)$. 

$$\begin{align*}
&x = 1 \\
y := 0
\end{align*}$$
when $d$ time unit elapse, each clock is incremented by some value between $d \times (1 - \epsilon)$ and $d \times (1 + \epsilon)$.
when \( d \) time unit elapse, each clock is incremented by some value between \( d \times (1 - \epsilon) \) and \( d \times (1 + \epsilon) \).
Extension with clock drifts

when \( d \) time unit elapse, each clock is incremented by some value between \( d \times (1 - \epsilon) \) and \( d \times (1 + \epsilon) \).

Since our algorithm is the same as [Pur98]'s, we get the following:

**Theorem**

\[
R_\Delta(\mathcal{A}) = R_\epsilon(\mathcal{A}) = R_{\Delta,\epsilon}(\mathcal{A}).
\]
Pros and cons of this approach

- **Cons:**
  - **Not very expressive:** the platform is very simple, thus not very realistic. Also, we over-approximate the set of executions.
  - **New techniques**, and much work still needed in order to be applicable;
Pros and cons of this approach

- **Cons:**
  - Not very expressive: the platform is very simple, thus not very realistic. Also, we over-approximate the set of executions.
  - New techniques, and much work still needed in order to be applicable;

- **Pros:**
  - Formal approach: we know what we are doing...
  - Reasonnable complexity: “only” PSPACE;
  - Faster is better: the enlarged semantics obviously satisfies this property.
Recent related work

This approach has received much attention in the last 3 years:

- extension to LTL properties [BMR06]:
  - Büchi automata techniques;
  - Repeated reachability.
Recent related work

This approach has received much attention in the last 3 years:

- extension to **LTL properties** [BMR06]:
  - Büchi automata techniques;
  - Repeated reachability.

- Extension to **timed properties**:
  - Different techniques;
  - No restrictions on cycles.
Recent related work

This approach has received much attention in the last 3 years:

- extension to LTL properties [BMR06]:
  - Büchi automata techniques;
  - Repeated reachability.

- Extension to timed properties:
  - Different techniques;
  - No restrictions on cycles.

- adaptations towards symbolic (zone-based) algorithms [DK06, SF07].
Outline of the talk

1. Introduction

2. Modeling the execution platform [Altisen & Tripakis, 2005]

3. A semantical approach [De Wulf et al., 2004]

4. Conclusions
Conclusions & Future Work

- **Implementability** is an important problem: the semantics of timed automata is too **mathematical**;
- **Two different approaches:**
  - **modeling the platform** is a very expressive approach that involves only classical techniques;
  - **enlarging the semantics** is a coarser solution, but has nice theoretical properties.
Conclusions & Future Work

- **Implementability** is an important problem: the semantics of timed automata is too *mathematical*;

- **Two different approaches:**
  - *modeling the platform* is a very expressive approach that involves only classical techniques;
  - *enlarging the semantics* is a coarser solution, but has nice theoretical properties.

- **Future work:**
  - Development and implementation of *symbolic (zone-based) algorithms*;
  - Direct synthesis of *robust controllers*. 