An introduction to timed systems

Patricia Bouyer-Decitre

LSV, CNRS & ENS Cachan, France
Outline

1. Introduction

2. The timed automaton model

3. Timed automata, decidability issues

4. How far can we extend the model and preserve decidability?
   - Hybrid systems
   - Smaller extensions of timed automata
   - An alternative way of proving decidability

5. Timed automata in practice

6. Conclusion
Time!

**Context:** verification of critical systems

**Time**
- naturally appears in real systems (for ex. protocols, embedded systems)
- appears in properties (for ex. bounded response time)
  
  “Will the airbag open within 5ms after the car crashes?”

→ Need of models and specification languages integrating timing aspects
Adding timing informations

- **Untimed case:** sequence of observable events
  
a: send message  
b: receive message

\[ a \ b \ a \ b \ a \ b \ a \ b \ a \ b \ \cdots = (a \ b)^\omega \]
Adding timing informations

- **Untimed case:** sequence of observable events
  
  \[ a \ b \ a \ b \ a \ b \ a \ b \ a \ b \ \cdots = (a \ b)^\omega \]

- **Timed case:** sequence of dated observable events

  \[(a, d_1) \ (b, d_2) \ (a, d_3) \ (b, d_4) \ (a, d_5) \ (b, d_6) \ \cdots \]

  \[d_1: \text{date at which the first } a \text{ occurs}\]
  
  \[d_2: \text{date at which the first } b \text{ occurs}, \ldots\]
Adding timing informations

- **Untimed case:** sequence of observable events
  - \(a\): send message
  - \(b\): receive message

\[
a \ b \ a \ b \ a \ b \ a \ b \ a \ b \cdots = (a \ b)^\omega
\]

- **Timed case:** sequence of **dated** observable events

\[
(a, d_1) \ (b, d_2) \ (a, d_3) \ (b, d_4) \ (a, d_5) \ (b, d_6) \cdots
\]

  - \(d_1\): date at which the first \(a\) occurs
  - \(d_2\): date at which the first \(b\) occurs, \ldots

- **Discrete-time semantics:** dates are e.g. taken in \(\mathbb{N}\)

  - **Ex:** \((a, 1)(b, 3)(c, 4)(a, 6)\)
Adding timing informations

- **Untimed case:** sequence of observable events
  
  \[ a: \text{send message} \quad b: \text{receive message} \]

  \[ a \ b \ a \ b \ a \ b \ a \ b \ a \ b \ \cdots = (a \ b)^\omega \]

- **Timed case:** sequence of **dated** observable events

  \[ (a, d_1) \ (b, d_2) \ (a, d_3) \ (b, d_4) \ (a, d_5) \ (b, d_6) \ \cdots \]

  \[ d_1: \text{date at which the first } a \text{ occurs} \]

  \[ d_2: \text{date at which the first } b \text{ occurs}, \ldots \]

  - **Discrete-time semantics:** dates are e.g. taken in \( \mathbb{N} \)
    
    Ex: \((a, 1)(b, 3)(c, 4)(a, 6)\)

  - **Dense-time semantics:** dates are e.g. taken in \( \mathbb{Q}_+ \), or in \( \mathbb{R}_+ \)
    
    Ex: \((a, 1.28).(b, 3.1).(c, 3.98)(a, 6.13)\)
A case for dense-time

**Time domain:** discrete (e.g. \( \mathbb{N} \)) or dense (e.g. \( \mathbb{Q}_+ \) or \( \mathbb{R}_+ \))

- A compositionality problem with discrete time
- Dense-time is a more general model than discrete time
- But, can we not always discretize?
A digital circuit [Alu91]

Discussion in the context of reachability problems for asynchronous digital circuits [BS91]

A digital circuit [Alu91]

Discussion in the context of reachability problems for asynchronous digital circuits [BS91]

Start with $x=0$ and $y=[101]$ (stable configuration)

A digital circuit [Alu91]

Discussion in the context of reachability problems for asynchronous digital circuits [BS91]

Start with $x=0$ and $y=[101]$ (stable configuration)

The input $x$ changes to 1. The corresponding stable state is $y=[011]$
A digital circuit [Alu91]

Discussion in the context of reachability problems for asynchronous digital circuits [BS91]

Start with $x=0$ and $y=[101]$ (stable configuration)

The input $x$ changes to 1. The corresponding stable state is $y=[011]$

However, many possible behaviours, e.g.

$$
[101] \xrightarrow{1.2} [111] \xrightarrow{2.5} [110] \xrightarrow{2.8} [010] \xrightarrow{4.5} [011]
$$


A digital circuit [Alu91]

Discussion in the context of reachability problems for asynchronous digital circuits [BS91]

Start with $x=0$ and $y=[101]$ (stable configuration)

The input $x$ changes to 1. The corresponding stable state is $y=[011]$

However, many possible behaviours, e.g.

$$ [101] \xrightarrow{y_2}{1.2} [111] \xrightarrow{y_3}{2.5} [110] \xrightarrow{y_1}{2.8} [010] \xrightarrow{y_3}{4.5} [011] $$

**Reachable configurations:** $\{[101], [111], [110], [010], [011], [001]\}$


Is discretizing sufficient? An example [Alu91]

- This digital circuit is not 1-discretizable.
Is discretizing sufficient? An example [Alu91]

- This digital circuit is not 1-discretizable.
- Why that? (initially $x = 0$ and $y = [11100000]$, $x$ is set to 1)
Is discretizing sufficient? An example [Alu91]

- This digital circuit is not 1-discretizable.
- Why that? (initially $x = 0$ and $y = [11100000]$, $x$ is set to 1)

$$
\begin{align*}
[11100000] & \xrightarrow{y_1} [01100000] & \xrightarrow{y_2} [00100000] & \xrightarrow{y_3, y_5} [00001000] & \xrightarrow{y_5, y_7} [00000010] & \xrightarrow{y_7, y_8} [00000001] \\
1 & & 1.5 & & 2 & & 3 & & 4 & & 1
\end{align*}
$$
Is discretizing sufficient? An example [Alu91]

- This digital circuit is not 1-discretizable.
- Why that?
  (initially $x = 0$ and $y = [11100000]$, $x$ is set to 1)

\[
\begin{array}{c}
[11100000] \xrightarrow{y_1} [01100000] \xrightarrow{y_2} [00100000] \xrightarrow{y_3, y_5} [00001000] \xrightarrow{y_5, y_7} [00000010] \xrightarrow{y_7, y_8} [00000001] \\
[11100000] \xrightarrow{y_1, y_2, y_3} [00000000] \\
\end{array}
\]
Is discretizing sufficient? An example [Alu91]

- This digital circuit is not 1-discretizable.
- Why that? (initially $x = 0$ and $y = [11100000]$, $x$ is set to 1)

\[
\begin{align*}
[11100000] & \xrightarrow{1} [01100000] \xrightarrow{1.5} [00100000] \xrightarrow{2} [00001000] \xrightarrow{3} [00000010] \xrightarrow{4} [00000001] \\
[11100000] & \xrightarrow{1} [00000000] \\
[11100000] & \xrightarrow{1} [01111000] \xrightarrow{2} [00000000]
\end{align*}
\]
Is discretizing sufficient? An example [Alu91]

- This digital circuit **is not** 1-discretizable.
- Why that? (initially $x = 0$ and $y = [11100000]$, $x$ is set to 1)

$$
\begin{align*}
[11100000] & \xrightarrow{y_1} [01100000] & \xrightarrow{y_2} [00100000] & \xrightarrow{y_3, y_5} [00001000] & \xrightarrow{y_5, y_7} [00000100] & \xrightarrow{y_7, y_8} [00000001] \\
[11100000] & \xrightarrow{y_1, y_2, y_3} [00000000] \\
[11100000] & \xrightarrow{y_1} [01111000] & \xrightarrow{y_2, y_3, y_4, y_5} [00000000] \\
[11100000] & \xrightarrow{y_1, y_2} [00100000] & \xrightarrow{y_3, y_5, y_6} [00001100] & \xrightarrow{y_5, y_6} [00000000]
\end{align*}
$$
Is discretizing sufficient? An example [Alu91]

- This digital circuit is not 1-discretizable.
- Why that? (initially $x = 0$ and $y = [11100000]$, $x$ is set to 1)

\[
\begin{align*}
[11100000] & \xrightarrow{\frac{y_1}{1}} [01100000] & \xrightarrow{\frac{y_2}{1.5}} [00100000] & \xrightarrow{\frac{y_3, y_5}{2}} [00001000] & \xrightarrow{\frac{y_5, y_7}{3}} [00000010] & \xrightarrow{\frac{y_7, y_8}{4}} [00000001] \\
[11100000] & \xrightarrow{\frac{y_1, y_2, y_3}{1}} [00000000] \\
[11100000] & \xrightarrow{\frac{y_1}{1}} [01110000] & \xrightarrow{\frac{y_2, y_3, y_4, y_5}{2}} [00000000] \\
[11100000] & \xrightarrow{\frac{y_1, y_2}{1}} [00100000] & \xrightarrow{\frac{y_3, y_5, y_6}{2}} [00001100] & \xrightarrow{\frac{y_5, y_6}{3}} [00000000]
\end{align*}
\]
Is discretizing sufficient?

**Theorem [BS91]**

For every $k \geq 1$, there exists a digital circuit such that the reachability set of states in dense-time is strictly larger than the one in discrete time (with granularity $\frac{1}{k}$).

Is discretizing sufficient?

**Theorem [BS91]**

For every $k \geq 1$, there exists a digital circuit such that the reachability set of states in dense-time is strictly larger than the one in discrete time (with granularity $\frac{1}{k}$).

**Claim**

Finding a correct granularity is as difficult as computing the set of reachable states in dense-time.

Is discretizing sufficient?

**Theorem [BS91]**

For every $k \geq 1$, there exists a digital circuit such that the reachability set of states in dense-time is strictly larger than the one in discrete time (with granularity $\frac{1}{k}$).

**Claim**

Finding a correct granularity is as difficult as computing the set of reachable states in dense-time.

**Further counter-example**

There exist systems for which no granularity exists. (see later)

Is discretizing sufficient?

**Theorem [BS91]**

For every $k \geq 1$, there exists a digital circuit such that the reachability set of states in dense-time is strictly larger than the one in discrete time (with granularity $\frac{1}{k}$).

**Claim**

Finding a correct granularity is as difficult as computing the set of reachable states in dense-time.

**Further counter-example**

There exist systems for which no granularity exists. (see later)

Hence, we better consider a dense-time domain!

Outline

1. Introduction

2. The timed automaton model

3. Timed automata, decidability issues

4. How far can we extend the model and preserve decidability?
   - Hybrid systems
   - Smaller extensions of timed automata
   - An alternative way of proving decidability

5. Timed automata in practice

6. Conclusion
A plethora of models...

... for real-time systems:
- timed circuits,
- time(d) Petri nets,
- timed process algebra,
- timed automata,
- ...

... and for real-time properties:
- timed observers,
- real-time logics: MTL, TPTL, TCTL, QTL, MITL...
A plethora of models...

... for real-time systems:
- timed circuits,
- time(d) Petri nets,
- timed process algebra,
- timed automata,
... 

... and for real-time properties:
- timed observers,
- real-time logics: MTL, TPTL, TCTL, QTL, MITL...
Timed automata [AD90]

- A finite control structure + variables (clocks)
- A transition is of the form:

\[ g, \ a, \ C := 0 \]

- An enabling condition (or guard) is:

\[ g ::= x \sim c \mid x - y \sim c \mid g \land g \]

where \( \sim \in \{<, \leq, =, \geq, >\} \)

[AD90] Alur, Dill. Automata for modeling real-time systems (ICALP’90).
Timed automata [AD90]

- A finite control structure + variables (clocks)
- A transition is of the form:

  \[ g, \ a, \ C := 0 \]

  Enabling condition

  Reset to zero

- An enabling condition (or guard) is:

  \[
  g ::= x \sim c \mid x - y \sim c \mid g \land g
  \]

  where \( \sim \in \{<, \leq, =, \geq, >\} \)

[AD90] Alur, Dill. Automata for modeling real-time systems (ICALP’90).
An example of a timed automaton

- **Safe** (Problem, $x:=0$)
- **Alarm**
  - Repair, $x \leq 15$
  - 15 $\leq x \leq 16$
- **Repairing**
  - Repair, $x \leq 15$
  - $y:=0$
- **Failsafe**
  - Done, $22 \leq y \leq 25$
  - $2 \leq y \land x \leq 56$
  - $y:=0$, $y:=0$
An example of a timed automaton

```
safe
  x  0
  y  0
```

- **safe**
  - done, $22 \leq y \leq 25$
  - problem, $x:=0$
  - repair, $x \leq 15$
  - 15$\leq x \leq 16$
  - delayed, $y:=0$

- **alarm**
  - repair, $y:=0$

- **repaing**
  - repair, $2 \leq y \land x \leq 56$
  - $y:=0$

- **failsafe**

The timed automaton model
An example of a timed automaton

The timed automaton model

1. **Safe**
   - Transition: Safe, $x:=0$
   - Next State: Problem

2. **Problem**
   - Transition: Problem, $x:=0$
   - Next State: Alarm

3. **Alarm**
   - Transition: Repair, $x \leq 15$
   - Next State: Repair
   - Transition: Delayed, $y:=0$
   - Next State: Fail safe

4. **Repair**
   - Transition: Repair, $2 \leq y \wedge x \leq 56$
   - Next State: Repair
   - Transition: Repair, $y:=0$
   - Next State: Repair

5. **Fail Safe**
   - Transition: Fail Safe

The timed automaton model

<table>
<thead>
<tr>
<th>State</th>
<th>Transition</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safe</td>
<td>$x:=0$</td>
<td>Problem</td>
</tr>
<tr>
<td>Safe</td>
<td>$y:=0$</td>
<td>Repair</td>
</tr>
<tr>
<td>Repair</td>
<td>$x:=0$</td>
<td>Repair</td>
</tr>
<tr>
<td>Repair</td>
<td>$y:=0$</td>
<td>Repair</td>
</tr>
<tr>
<td>Repair</td>
<td>$2 \leq y \wedge x \leq 56$</td>
<td>Repair</td>
</tr>
<tr>
<td>Repair</td>
<td>$y:=0$</td>
<td>Repair</td>
</tr>
<tr>
<td>Fail Safe</td>
<td></td>
<td>Fail Safe</td>
</tr>
</tbody>
</table>

The timed automaton model
An example of a timed automaton

The timed automaton model

<table>
<thead>
<tr>
<th></th>
<th>safe</th>
<th>23</th>
<th>safe</th>
<th>problem</th>
<th>alarm</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td>23</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>0</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td></td>
</tr>
</tbody>
</table>
An example of a timed automaton

The timed automaton model
An example of a timed automaton

\[ \begin{align*}
\text{safe} & \xrightarrow{23} \text{safe} & \text{problem} & \xrightarrow{} \text{alarm} & \xrightarrow{15.6} \text{alarm} & \xrightarrow{\text{delayed}} \text{failsafe} \\
X & 0 & 23 & 0 & 15.6 & 15.6 & \ldots \\
Y & 0 & 23 & 23 & 38.6 & 0 \\
\text{failsafe} & \xrightarrow{} \text{failsafe} & \ldots & 15.6 & \ldots \\
\end{align*} \]
An example of a timed automaton

The timed automaton model
An example of a timed automaton

The timed automaton model
An example of a timed automaton

The timed automaton model
An example of a timed automaton

The timed automaton model

<table>
<thead>
<tr>
<th>State</th>
<th>Transition</th>
<th>State</th>
<th>Transition</th>
<th>State</th>
<th>Transition</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>safe</td>
<td>23</td>
<td>safe</td>
<td>problem</td>
<td>alarm</td>
<td>15.6</td>
<td>alarm</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>23</td>
<td>0</td>
<td>15.6</td>
<td>15.6</td>
<td>15.6</td>
</tr>
<tr>
<td>y</td>
<td>0</td>
<td>23</td>
<td>23</td>
<td>38.6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>failsafe</td>
<td>2.3</td>
<td>failsafe</td>
<td>repair</td>
<td>repairing</td>
<td>22.1</td>
<td>repairing</td>
</tr>
<tr>
<td>...</td>
<td>15.6</td>
<td>17.9</td>
<td>17.9</td>
<td>40</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>2.3</td>
<td>0</td>
<td>22.1</td>
<td>22.1</td>
<td></td>
</tr>
</tbody>
</table>
An example of a timed automaton

The timed automaton model

This run read the timed word

(problem, 23)(delayed, 38.6)(repair, 40.9), (done, 63).
Timed automata semantics

- $\mathcal{A} = (\Sigma, L, X, \longrightarrow)$ is a TA

- **Configurations**: $(\ell, \nu) \in L \times T^X$ where $T$ is the time domain
  - $\nu$ is called the (clock) valuation

- **Timed transition system**:
  - **action transition**: $(\ell, \nu) \xrightarrow{a} (\ell', \nu')$ if $\exists \ell \xrightarrow{g, a, r} \ell' \in \mathcal{A}$ s.t.
    
    \[
    \begin{cases}
      \nu = g \\
      \nu' = \nu[r \leftarrow 0]
    \end{cases}
    \]

  - **delay transition**: $(\ell, \nu) \xrightarrow{\delta(d)} (\ell, \nu + d)$ if $d \in T$
Discrete vs dense-time semantics

$x = 1, a, x := 0$

$b, y := 0$

$x = 1, a, x := 0$

$y < 1, b, y := 0$
Discrete vs dense-time semantics

\[ L_{\text{dense}} = \{ ((ab)^\omega, \tau) \mid \forall i, \ \tau_{2i-1} = i \text{ and } \tau_{2i} - \tau_{2i-1} > \tau_{2i+2} - \tau_{2i+1} \} \]
Discrete vs dense-time semantics

- **Dense-time:**
  \[ L_{dense} = \{ ((ab)^\omega, \tau) \mid \forall i, \, \tau_{2i-1} = i \text{ and } \tau_{2i} - \tau_{2i-1} > \tau_{2i+2} - \tau_{2i+1} \} \]

- **Discrete-time:** \( L_{discrete} = \emptyset \)
Discrete vs dense-time semantics

- Dense-time:
  \[ L_{\text{dense}} = \{ ((ab)^\omega, \tau) \mid \forall i, \; \tau_{2i-1} = i \text{ and } \tau_{2i} - \tau_{2i-1} > \tau_{2i+2} - \tau_{2i+1} \} \]

- Discrete-time: \( L_{\text{discrete}} = \emptyset \)

However, it does result from the following parallel composition:
Classical verification problems

- reachability of a control state
- \( S \sim S' \): bisimulation, etc...
- \( L(S) \subseteq L(S') \): language inclusion
- \( S \models \varphi \) for some formula \( \varphi \): model-checking
- \( S \parallel A_T + \) reachability: testing automata
- ...
Classical temporal logics

Path formulas:

\[
\begin{align*}
G \varphi & \quad \text{“Always”} \\
F \varphi & \quad \text{“Eventually”} \\
\varphi U \varphi' & \quad \text{“Until”} \\
X \varphi & \quad \text{“Next”}
\end{align*}
\]

State formulas:

\[
\begin{align*}
A \psi & \\
E \psi
\end{align*}
\]

\[\leadsto\] LTL: Linear Temporal Logic \cite{Pnu77},
CTL: Computation Tree Logic \cite{EC82}

\cite{Pnu77} Pnueli. The temporal logic of programs (FoCS’77).
\cite{EC82} Emerson, Clarke. Using branching time temporal logic to synthesize synchronization skeletons (Science of Computer Programming 1982).
Adding time to temporal logics

Classical temporal logics allow us to express that
“any problem is followed by an alarm”

Adding time to temporal logics

Classical temporal logics allow us to express that
“any problem is followed by an alarm”

With CTL:

$$\forall G (\text{problem} \Rightarrow \exists F \text{ alarm})$$

Adding time to temporal logics

Classical temporal logics allow us to express that

“any problem is followed by an alarm”

With CTL:

\[ \text{AG}(\text{problem} \Rightarrow \text{AF alarm}) \]

How can we express:

“any problem is followed by an alarm within 20 time units”? 

Adding time to temporal logics

Classical temporal logics allow us to express that
“any problem is followed by an alarm”

With CTL:
\[ \text{AG (problem } \Rightarrow \text{ AF alarm)} \]

How can we express:
“any problem is followed by an alarm \textit{within 20 time units}”?

- Temporal logics with \textbf{subscripts}.  

\[ \text{ex: CTL + } \begin{array}{c}
\text{E} \varphi \; \text{U}_{\sim k} \; \psi \\
\text{A} \varphi \; \text{U}_{\sim k} \; \psi \\
\end{array} \]


Adding time to temporal logics

Classical temporal logics allow us to express that
“any problem is followed by an alarm”

With CTL:

\[ AG(\text{problem} \Rightarrow AF \text{ alarm}) \]

How can we express:
“any problem is followed by an alarm within 20 time units”?

- Temporal logics with **subscripts**.

\[ AG(\text{problem} \Rightarrow AF_{\leq 20} \text{ alarm}) \]

---

[ACD93] Alur, Courcoubetis, Dill. Model-checking in dense real-time (*Information and Computation*).
[HNSY94] Henzinger, Nicollin, Sifakis, Yovine. Symbolic model-checking for real-time systems (*ACM Transactions on Computational Logic*).
Adding time to temporal logics

Classical temporal logics allow us to express that
“any problem is followed by an alarm”

With CTL:

\[ \text{AG}(\text{problem } \Rightarrow \text{AF alarm}) \]

How can we express:
“any problem is followed by an alarm within 20 time units”?

- Temporal logics with **subscripts**.

\[ \text{AG}(\text{problem } \Rightarrow \text{AF}^{\leq 20} \text{ alarm}) \]

- Temporal logics with **clocks**.

\[ \text{AG}(\text{problem } \Rightarrow (x \text{ in AF}(x \leq 20 \land \text{alarm}))) \]

[ACD93] Alur, Courcoubetis, Dill. Model-checking in dense real-time (*Information and Computation*).
[HNSY94] Henzinger, Nicollin, Sifakis, Yovine. Symbolic model-checking for real-time systems (*ACM Transactions on Computational Logic*).
Adding time to temporal logics

Classical temporal logics allow us to express that
"any problem is followed by an alarm"

With CTL:

\[ \text{AG} (\text{problem} \Rightarrow \text{AF} \, \text{alarm}) \]

How can we express:
"any problem is followed by an alarm within 20 time units"?

- Temporal logics with subscripts.

\[ \text{AG} (\text{problem} \Rightarrow \text{AF}_{\leq 20} \, \text{alarm}) \]

- Temporal logics with clocks.

\[ \text{AG} (\text{problem} \Rightarrow (x \in \text{AF} (x \leq 20 \land \text{alarm}))) \]

\[ \leadsto \text{TCTL: Timed CTL} \quad [\text{ACD90, ACD93, HNSY94}] \]

The train crossing example (1)

Train$_i$ with $i = 1, 2, ...$

- **Far**
  - $10 < x_i < 20$, Exit!
  - $20 < x_i < 30$, $a, x_i := 0$

- **Before, $x_i < 30$**
  - App!, $x_i := 0$

- **On, $x_i < 20$**
The train crossing example

The gate:

\[ \text{Open} \xrightarrow{\text{GoDown?}, H_g := 0} \text{Lowering}, H_g < 10 \]

\[ \text{Raising}, H_g < 10 \xrightarrow{H_g < 10,a} \text{Close} \]

\[ \text{GoUp?}, H_g := 0 \xrightarrow{H_g < 10,a} \text{Open} \]
The train crossing example

The controller:

$c_1, H_c \leq 20$

Exit?, $H_c := 0$

$c_0$

$H_c = 20$, GoUp!

App?, $H_c := 0$

$c_2, H_c \leq 10$

Exit?

$H_c \leq 10$, GoDown!

Exit?

App?
The train crossing example

We use the synchronization function $f$:

<table>
<thead>
<tr>
<th>Train$_1$</th>
<th>Train$_2$</th>
<th>Gate</th>
<th>Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>App!</td>
<td>.</td>
<td>.</td>
<td>App?</td>
</tr>
<tr>
<td></td>
<td>App!</td>
<td>.</td>
<td>App?</td>
</tr>
<tr>
<td>Exit!</td>
<td>.</td>
<td>.</td>
<td>Exit?</td>
</tr>
<tr>
<td></td>
<td>Exit!</td>
<td>.</td>
<td>Exit?</td>
</tr>
<tr>
<td>a</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>.</td>
<td>a</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>.</td>
<td>GoUp?</td>
<td>GoUp!</td>
</tr>
<tr>
<td></td>
<td>.</td>
<td>GoDown?</td>
<td>GoDown!</td>
</tr>
</tbody>
</table>

to define the parallel composition $(\text{Train}_1 \parallel \text{Train}_2 \parallel \text{Gate} \parallel \text{Controller})$

**NB:** the parallel composition does not add expressive power!
The train crossing example (5)

Some properties one could check:

- Is the gate closed when a train crosses the road?
The train crossing example

Some properties one could check:

Is the gate closed when a train crosses the road?

$$\mathbf{AG} (\text{train.} \, \text{On} \Rightarrow \text{gate.} \, \text{Close})$$
The train crossing example (5)

Some properties one could check:

- Is the gate closed when a train crosses the road?

\[ AG (\text{train.On} \Rightarrow \text{gate.Close}) \]

- Is the gate always closed for less than 5 minutes?
The train crossing example

Some properties one could check:

- Is the gate closed when a train crosses the road?
  \[
  \text{AG} (\text{train.On} \Rightarrow \text{gate.Close})
  \]

- Is the gate always closed for less than 5 minutes?
  \[
  \neg \text{EF} (\neg \text{gate.Close} \land \text{E}(\neg \text{gate.Close} \cup_{>5\text{ min}} \neg \text{gate.Close}))
  \]
Another example: A Fischer protocol

A mutual exclusion protocol with a shared variable $id$ [AL94].

Another example: A Fischer protocol

A mutual exclusion protocol with a shared variable $id$ [AL94].

Process $i$:

- $a : \text{await } (id = 0)$;
- $b : \text{set } id \text{ to } i$;
- $c : \text{await } (id = i)$;
- $d : \text{enter critical section}$.

$\leadsto$ a max. delay $k_1$ between $a$ and $b$

a min. delay $k_2$ between $b$ and $c$

Another example: A Fischer protocol

A mutual exclusion protocol with a shared variable $id$ [AL94].

Process $i$:
\begin{align*}
a & : \text{await ($id = 0$);} \\
b & : \text{set $id$ to $i$;} \\
c & : \text{await ($id = i$);} \\
d & : \text{enter critical section.}
\end{align*}

$\leadsto$ a max. delay $k_1$ between $a$ and $b$

$\leadsto$ a min. delay $k_2$ between $b$ and $c$

$\leadsto$ See the demo with the tool Uppaal
(can be downloaded freely on http://www.uppaal.com/)

Outline

1. Introduction

2. The timed automaton model

3. Timed automata, decidability issues

4. How far can we extend the model and preserve decidability?
   Hybrid systems
   Smaller extensions of timed automata
   An alternative way of proving decidability

5. Timed automata in practice

6. Conclusion
Verification

Emptiness problem: is the language accepted by a timed automaton empty?
- basic reachability/safety properties
- basic liveness properties
  (final states)
  (ω-regular conditions)

Verification

Emptiness problem: is the language accepted by a timed automaton empty?

- **Problem:** the set of configurations is infinite
  \[\leadsto\] classical methods for finite-state systems cannot be applied

Verification

**Emptiness problem:** is the language accepted by a timed automaton empty?

- **Problem:** the set of configurations is infinite
  \[ \leadsto \] classical methods for finite-state systems cannot be applied

- **Positive key point:** variables (clocks) increase at the same speed

Verification

**Emptiness problem**: is the language accepted by a timed automaton empty?

- **Problem**: the set of configurations is infinite
  - classical methods for finite-state systems cannot be applied

- **Positive key point**: variables (clocks) increase at the same speed

**Theorem** [AD90,AD94]

The emptiness problem for timed automata is decidable and PSPACE-complete.

[AD90] Alur, Dill. Automata for modeling real-time systems (ICALP’90).
Verification

Emptiness problem: is the language accepted by a timed automaton empty?

- Problem: the set of configurations is infinite
  \[ \leadsto \] classical methods for finite-state systems cannot be applied

- Positive key point: variables (clocks) increase at the same speed

Theorem [AD90,AD94]
The emptiness problem for timed automata is decidable and PSPACE-complete.

Method: construct a finite abstraction

[AD90] Alur, Dill. Automata for modeling real-time systems (ICALP’90).
The region abstraction
The region abstraction

- “compatibility” between regions and constraints
The region abstraction

- “compatibility” between regions and constraints
- “compatibility” between regions and time elapsing
The region abstraction

- “compatibility” between regions and constraints
- “compatibility” between regions and time elapsing
The region abstraction

- “compatibility” between regions and constraints
- “compatibility” between regions and time elapsing

\[ \sim \] an equivalence of finite index
a time-abstract bisimulation
Time-abstract bisimulation

\[ \forall \quad a \quad \rightarrow \]

Timed automata, decidability issues
Time-abstract bisimulation

\[ \forall \quad \exists \]

\[ a \]

\[ a \]
Time-abstract bisimulation
Time-abstract bisimulation

\[ \forall a \quad \exists a \]

\[ \forall d > 0 \quad \delta(d) \]

\[ \exists d' > 0 \quad \delta(d') \]
Time-abstract bisimulation

$(l_0, v_0) \xrightarrow{a_1, t_1} (l_1, v_1) \xrightarrow{a_2, t_2} (l_2, v_2) \xrightarrow{a_3, t_3} \ldots$
Time-abstract bisimulation

\[\forall a \quad \exists a \quad a \rightarrow \quad \delta(d) \quad \delta(d') \]

\[(\ell_0, v_0) \xrightarrow{a_1, t_1} (\ell_1, v_1) \xrightarrow{a_2, t_2} (\ell_2, v_2) \xrightarrow{a_3, t_3} \ldots \]

\[(\ell_0, R_0) \xrightarrow{a_1} (\ell_1, R_1) \xrightarrow{a_2} (\ell_2, R_2) \xrightarrow{a_3} \ldots \]

with \(v_i \in R_i\) for all \(i\).
Time-abstract bisimulation

\[
\begin{align*}
\forall \quad & (\ell_0, v_0) \xrightarrow{a_1, t_1} (\ell_1, v_1) \xrightarrow{a_2, t_2} (\ell_2, v_2) \xrightarrow{a_3, t_3} \ldots \\
\exists \quad & (\ell_0, R_0) \xrightarrow{a_1} (\ell_1, R_1) \xrightarrow{a_2} (\ell_2, R_2) \xrightarrow{a_3} \ldots \\
\end{align*}
\]

with \( v_i \in R_i \) for all \( i \).
The region abstraction
The region abstraction

\[
\begin{align*}
2 < x &< 3 \\
1 < y &< 2 \\
\{x\} &< \{y\}
\end{align*}
\]
The region abstraction

time successors
The region abstraction

reset of clock $y$
The region abstraction

reset of clock $x$
The region graph

A finite graph representing time elapsing and reset of clocks:

\[\text{time elapsing} \rightarrow \]

\[\text{reset to 0} \rightarrow \]
Region automaton \equiv finite bisimulation quotient

\textbf{timed automaton} \otimes \textbf{region graph}
Region automaton $\equiv$ finite bisimulation quotient

**timed automaton $\otimes$ region graph**

$\ell \xrightarrow{\text{g, } a, \text{C:=0}} \ell'$ is transformed into:

$(\ell, \mathcal{R}) \xrightarrow{a} (\ell', \mathcal{R'})$ if there exists $\mathcal{R''} \in \text{Succ}_t^*(\mathcal{R})$ s.t.

- $\mathcal{R''} \subseteq g$
- $[C \leftarrow 0]\mathcal{R''} \subseteq \mathcal{R'}$
Region automaton $\equiv$ finite bisimulation quotient

**timed automaton $\otimes$ region graph**

$\ell \xrightarrow{g,a,C:=0} \ell'$ is transformed into:

$(\ell, R) \xrightarrow{a} (\ell', R')$ if there exists $R'' \in \text{Succ}_t^*(R)$ s.t.

- $R'' \subseteq g$
- $[C \leftarrow 0]R'' \subseteq R'$

$\mathcal{L}(\text{reg. aut.}) = \text{UNTIME}(\mathcal{L}(\text{timed aut.}))$

where $\text{UNTIME}((a_1, t_1)(a_2, t_2)\ldots) = a_1a_2\ldots$
Timed automata, decidability issues

\[ \mathcal{L}(\text{reg. aut.}) = \text{UNTIME}(\mathcal{L}(\text{timed aut.})) \]
An example [AD94]
PSPACE membership

The size of the region graph is in $O(|X|!.2^{|X|})$

- **One configuration**: a discrete location + a region
PSPACE membership

The size of the region graph is in $O(|X|!.2^{|X|})$

- **One configuration:** a discrete location + a region
  - a discrete location: log-space
PSPACE membership

The size of the region graph is in $O(|X|! \cdot 2^{|X|})$

- **One configuration**: a discrete location + a region
  - a discrete location: log-space
  - a region:
    - an interval for each clock
    - an interval for each pair of clocks
PSPACE membership

The size of the region graph is in $O(|X|! \cdot 2^{|X|})$

- **One configuration:** a discrete location + a region
  - a discrete location: log-space
  - a region:
    - an interval for each clock
    - an interval for each pair of clocks
  \[\leadsto\] requires polynomial space
PSPACE membership

The size of the region graph is in \( O(|X|!.2^{|X|}) \)

- **One configuration:** a discrete location + a region
  - a discrete location: log-space
  - a region:
    - an interval for each clock
    - an interval for each pair of clocks

\[ \leadsto \text{ requires polynomial space} \]

- By guessing a path of length at most exponential: needs only to store two consecutive configurations
PSPACE membership

The size of the region graph is in $O(|X|! \cdot 2^{|X|})$

- **One configuration:** a discrete location + a region
  - a discrete location: log-space
  - a region:
    - an interval for each clock
    - an interval for each pair of clocks

  $\leadsto$ requires polynomial space

- By guessing a path of length at most exponential: needs only to store two consecutive configurations

  $\leadsto$ in NPSPACE, thus in PSPACE
PSPACE-hardness

\[ \mathcal{M} \text{ LBTM } \{a, b\}^* \sim A_{\mathcal{M}, w_0} \text{ s.t. } \mathcal{M} \text{ accepts } w_0 \text{ iff the final state of } A_{\mathcal{M}, w_0} \text{ is reachable} \]

\[ w_0 \]

\( C_j \)

\{x_j, y_j\}

\( C_j \) contains an “a” if \( x_j = y_j \)

\( C_j \) contains a “b” if \( x_j < y_j \)

(these conditions are invariant by time elapsing)

LBTM: linearly bounded Turing machine (a witness for PSPACE-complete problems)
PSPACE-hardness (cont.)

If \( q \xrightarrow{\alpha,\alpha',\delta} q' \) is a transition of \( M \), then for each position \( i \) of the tape, we have a transition

\[
(q, i) \xrightarrow{g, r := 0} (q', i')
\]

where:

- \( g \) is \( x_i = y_i \) (resp. \( x_i < y_i \)) if \( \alpha = a \) (resp. \( \alpha = b \))
- \( r = \{x_i, y_i\} \) (resp. \( r = \{x_i\} \)) if \( \alpha' = a \) (resp. \( \alpha' = b \))
- \( i' = i + 1 \) (resp. \( i' = i - 1 \)) if \( \delta \) is right and \( i < n \) (resp. left)

**Enforcing time elapsing:** on each transition, add the condition \( t = 1 \) and clock \( t \) is reset.

**Initialization:** \( \text{init} \xrightarrow{t=1, r_0 := 0} (q_0, 1) \) where \( r_0 = \{x_i \mid w_0[i] = b\} \cup \{t\} \)

**Termination:** \( (q_f, i) \longrightarrow \text{end} \)
The case of single-clock timed automata

Exercise [LMS04]

Think of the special case of single-clock timed automata. Can we do better than PSPACE?

[LMS04] Laroussinie, Markey, Schnoebelen. Model checking timed automata with one or two clocks (CONCUR’04).
Consequence of region automata construction

Region automata:

correct finite (and exponential) abstraction for checking reachability/Büchi-like properties.
Consequence of region automata construction

Region automata:
correct finite (and exponential) abstraction for checking reachability/Büchi-like properties.

However...
everything can not be reduced to finite automata...
A model not far from undecidability

Some bad news...

- Language universality is undecidable  [AD90]
- Language inclusion is undecidable  [AD90]
- Complementability is undecidable  [Tri03, Fin06]
- ...

[Tri03] Tripakis. Folk theorems on the determinization and minimization of timed automata (FORMATS’03).
[Fin06] Finkel. Undecidable problems about timed automata (FORMATS’06).
A model not far from undecidability

Some bad news...

- Language universality is undecidable
- Language inclusion is undecidable
- Complementability is undecidable
- ...

An example of non-determinizable/non-complementable timed aut.:

---

[Tri03] Tripakis. Folk theorems on the determination and minimization of timed automata (FORMATS’03).
[Fin06] Finkel. Undecidable problems about timed automata (FORMATS’06).
[AM04] Alur, Madhusudan. Decision problems for timed automata: A survey (SFM-04:RT)).
A model not far from undecidability

Some bad news...

- Language universality is undecidable [AD90]
- Language inclusion is undecidable [AD90]
- Complementability is undecidable [Tri03, Fin06]
- ...

An example of non-determinizable/non-complementable timed aut.: [AM04]

---

[Tri03] Tripakis. Folk theorems on the determination and minimization of timed automata (FORMATS’03).
[Fin06] Finkel. Undecidable problems about timed automata (FORMATS’06).
A model not far from undecidability

Some bad news...

- Language universality is undecidable  
  [AD90]
- Language inclusion is undecidable  
  [AD90]
- Complementability is undecidable  
  [Tri03,Fin06]
- ...

An example of non-determinizable/non-complementable timed aut.:  

[AM04]

\[
\begin{align*}
&\text{UNTIME } (L \cap \{(a^* b^*, \tau) \mid \text{all } a's \text{ happen before } 1 \text{ and no two } a's \text{ simultaneously}\}) \text{ is not regular} \\
&a, b \quad x \neq 1, \quad a, b \\
&\quad a, \quad x := 0 \\
&\quad a, b
\end{align*}
\]

[Tri03] Tripakis. Folk theorems on the determination and minimization of timed automata (FORMATS’03).
[Fin06] Finkel. Undecidable problems about timed automata (FORMATS’06).
The two-counter machine

Definition

A two-counter machine is a finite set of instructions over two counters \((c\) and \(d)\):

- Incrementation:
  
  \((p)\): \(c := c + 1; \ \text{goto} \ (q)\)

- Decrementation:
  
  \((p)\): \(\text{if} \ c > 0 \ \text{then} \ c := c - 1; \ \text{goto} \ (q) \ \text{else} \ \text{goto} \ (r)\)

Theorem [Minsky 67]

The halting problem for two counter machines is undecidable.
Undecidability of universality

Theorem [AD90]

Universality of timed automata is undecidable.

- one configuration is encoded in one time unit
- number of $c$’s: value of counter $c$
- number of $d$’s: value of counter $d$
- one time unit between two corresponding $c$’s (resp. $d$’s)

\[\leadsto\] We encode “non-behaviours” of a two-counter machine
Example

Module to check that if instruction $i$ does not decrease counter $c$, then all actions $c$ appearing less than 1 t.u. after $b_i$ has to be followed by an other $c$ 1 t.u. later.
Example

Module to check that if instruction $i$ does not decrease counter $c$, then all actions $c$ appearing less than 1 t.u. after $b_i$ has to be followed by an other $c$ 1 t.u. later.

$x = 1, \neg c$

$x \neq 1$

The union of all small modules is not universal iff

The two-counter machine has a recurring computation
Partial conclusion

- This idea of a finite bisimulation quotient has been applied to many “timed” or “hybrid” systems:
  - various extensions of timed automata
    - [Bérard, Diekert, Gastin, Petit 1998]
    - [Choffrut, Goldwurm 2000]
    - [Bouyer, Dufourd, Fleury, Petit 2004] · · ·
Partial conclusion

- This idea of a finite bisimulation quotient has been applied to many “timed” or “hybrid” systems:
  - various extensions of timed automata
    [Bérard, Diekert, Gastin, Petit 1998] [Choffrut, Goldwurm 2000]
    [Bouyer, Dufourd, Fleury, Petit 2004] · · ·
  - model-checking of branching-time properties (TCTL, timed μ-calculus)
    [Alur, Courcoubetis, Dill 1993] [Laroussinie, Larsen, Weise 1995]
Partial conclusion

- This idea of a finite bisimulation quotient has been applied to many “timed” or “hybrid” systems:
  - various extensions of timed automata
    [Bérard, Diekert, Gastin, Petit 1998] [Choffrut, Goldwurm 2000]
    [Bouyer, Dufourd, Fleury, Petit 2004]
  - model-checking of branching-time properties (TCTL, timed μ-calculus)
    [Alur, Courcoubetis, Dill 1993] [Laroussinie, Larsen, Weise 1995]
  - weighted/-priced timed automata (e.g. WCTL model-checking, optimal games)
    [Bouyer, Larsen, Markey, Rasmussen 2006] [Bouyer, Larsen, Markey 2007]
Partial conclusion

- This idea of a finite bisimulation quotient has been applied to many “timed” or “hybrid” systems:
  - various extensions of timed automata
    - [Bérard,Diekert,Gastin,Petit 1998] [Choffrut,Goldwurm 2000]
    - [Bouyer,Dufourd,Fleury,Petit 2004] · · ·
  - model-checking of branching-time properties (TCTL, timed $\mu$-calculus)
    - [Alur,Courcoubetis,Dill 1993] [Laroussinie,Larsen,Weise 1995]
  - weighted/priced timed automata (e.g. WCTL model-checking, optimal games)
    - [Bouyer,Larsen,Markey,Rasmussen 2006] [Bouyer,Larsen,Markey 2007]
  - o-minimal hybrid systems
    - [Lafferriere,Pappas,Sastry 2000] [Brihaye 2005]
Partial conclusion

- This idea of a finite bisimulation quotient has been applied to many “timed” or “hybrid” systems:
  - various extensions of timed automata
    [Bérard, Diekert, Gastin, Petit 1998] [Choffrut, Goldwurm 2000]
    [Bouyer, Dufourd, Fleury, Petit 2004] ···
  - model-checking of branching-time properties (TCTL, timed $\mu$-calculus)
    [Alur, Courcoubetis, Dill 1993] [Laroussinie, Larsen, Weise 1995]
  - weighted/priced timed automata (e.g. WCTL model-checking, optimal games)
    [Bouyer, Larsen, Markey, Rasmussen 2006] [Bouyer, Larsen, Markey 2007]
  - o-minimal hybrid systems
    [Lafferriere, Pappas, Sastry 2000] [Brihaye 2005]
  - ···
Partial conclusion

This idea of a finite bisimulation quotient has been applied to many “timed” or “hybrid” systems:

- various extensions of timed automata
  [Bérard,Diekert,Gastin,Petit 1998] [Choffrut,Goldwurm 2000]
  [Bouyer,Dufourd,Fleury,Petit 2004] · · ·

- model-checking of branching-time properties (TCTL, timed μ-calculus)
  [Alur,Courcoubetis,Dill 1993] [Laroussinie,Larsen,Weise 1995]

- weighted/priced timed automata (e.g. WCTL model-checking, optimal games)
  [Bouyer,Larsen,Markey,Rasmussen 2006] [Bouyer,Larsen,Markey 2007]

- o-minimal hybrid systems
  [Lafferriere,Pappas,Sastry 2000] [Brihaye 2005]

· · ·

Note however that it might be hard to prove there is a finite bisimulation quotient!
Outline

1. Introduction

2. The timed automaton model

3. Timed automata, decidability issues

4. How far can we extend the model and preserve decidability?
   Hybrid systems
   Smaller extensions of timed automata
   An alternative way of proving decidability

5. Timed automata in practice

6. Conclusion
Outline

1. Introduction

2. The timed automaton model

3. Timed automata, decidability issues

4. How far can we extend the model and preserve decidability?
   Hybrid systems
   Smaller extensions of timed automata
   An alternative way of proving decidability

5. Timed automata in practice

6. Conclusion
A general model: hybrid systems

What is a hybrid system?

- a discrete control (the mode of the system)
- a continuous evolution within a mode (given by variables)

[Henzinger. The theory of hybrid automata (LICS'96).]
A general model: hybrid systems

What is a hybrid system?

- a discrete control (the mode of the system)
- a continuous evolution within a mode (given by variables)

Example (The thermostat)

A simple thermostat, where $T$ (the temperature) depends on the time:

- Off: $\dot{T} = -0.5T$ ($T \geq 18$)
- On: $\dot{T} = 2.25 - 0.5T$ ($T \leq 22$)

The thermostat example

**Off**
\[ \dot{T} = -0.5T \quad (T \geq 18) \]

**On**
\[ \dot{T} = 2.25 - 0.5T \quad (T \leq 22) \]
The thermostat example

\[ T \leq 19 \]

Off
\[ \dot{T} = -0.5T \]
\( T \geq 18 \)

On
\[ \dot{T} = 2.25 - 0.5T \]
\( T \leq 22 \)

\[ T \geq 21 \]
Ok...

How far can we extend the model and preserve decidability?
Ok...

Easy...
How far can we extend the model and preserve decidability?

Ok...

Easy...
Ok...

How far can we extend the model and preserve decidability?
Ok... but?

How far can we extend the model and preserve decidability?
Ok... but?

How far can we extend the model and preserve decidability?
What about decidability?

\[ \leadsto \text{almost everything is undecidable} \]

Negative results [HKPV95]

- The class of hybrid systems with clocks and only one variable having possibly two slopes \( k_1 \neq k_2 \) is undecidable.
- The class of \textit{stopwatch} automata is undecidable.

Outline

1. Introduction

2. The timed automaton model

3. Timed automata, decidability issues

4. How far can we extend the model and preserve decidability?
   - Hybrid systems
     - Smaller extensions of timed automata
     - An alternative way of proving decidability

5. Timed automata in practice

6. Conclusion
Role of diagonal constraints

\[ x - y \sim c \quad \text{and} \quad x \sim c \]
Role of diagonal constraints

\[ x - y \sim c \quad \text{and} \quad x \sim c \]

- Decidability: yes, using the region abstraction
Role of diagonal constraints

\[ x - y \sim c \text{ and } x \sim c \]

- **Decidability**: yes, using the region abstraction

- **Expressiveness**: no additional expressive power
Role of diagonal constraints (cont.)

$c$ is positive

\[ x := 0 \quad \quad \quad \quad y := 0 \quad \quad \quad \quad x - y \leq c \]

copy where \( x - y \leq c \)

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

\[ y := 0 \]

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

\[ x := 0 \]

\[ y := 0 \]

copy where \( x - y > c \)

\[ \leadsto \text{proof in [BDGP98]} \]

[BDGP98] Bérard, Diekert, Gastin, Petit. Characterization of the expressive power of silent transitions in timed automata (Fundamenta Informaticae).
Exercise [BC05]

Consider, for every positive integer $n$, the timed language:

$$\mathcal{L}_n = \{(a, t_1) \ldots (a, t_{2^n}) \mid 0 < t_1 < \cdots < t_{2^n} < 1\}$$

- Construct a timed automaton with diagonal constraints which recognizes $\mathcal{L}_n$. What is the size of this automaton?
- Idem without diagonal constraints. Can you do better?
- Conclude.

Adding silent actions

\[ g, \varepsilon, C := 0 \]

[BDGP98]
Adding silent actions

\[ g, \varepsilon, C := 0 \]

[BDGP98]

- **Decidability:** yes
  (actions have no influence on region automaton construction)
Adding silent actions

- **Decidability:** yes
  (actions have no influence on region automaton construction)

- **Expressiveness:** strictly more expressive!

\[
\begin{align*}
g, \varepsilon, C &:= 0 \\
x = 1, a, x &:= 0 \\
x = 1, \varepsilon, x &:= 0
\end{align*}
\]
Adding additive constraints

\[ x + y \sim c \quad \text{and} \quad x \sim c \quad [BD00] \]

[BD00] Bérard, Dufourd. Timed automata and additive clock constraints (Information Processing Letters).
Adding additive constraints

\[ x + y \sim c \quad \text{and} \quad x \sim c \]  

**Decidability:** for two clocks, *decidable* using the abstraction

[Berard, Dufourd. Timed automata and additive clock constraints (Information Processing Letters).]
Adding additive constraints

**Decidability:** - for two clocks, **decidable** using the abstraction

- for four clocks (or more), **undecidable**!

**Expressiveness:** more expressive! (even using two clocks)

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

\[ \{(a^n, t_1 \ldots t_n) \mid n \geq 1 \text{ and } t_i = 1 - \frac{1}{2^i} \} \]

[BD00] Bérard, Dufourd. Timed automata and additive clock constraints (Information Processing Letters).
Undecidability proof

\[ c \text{ is unchanged} \quad \text{c is incremented} \]

\[ 20 \quad 21 \quad 22 \quad 23 \quad 24 \quad 25 \quad 26 \quad \text{time} \]

\[ \leadsto \text{simulation of} \quad \text{ decrementation of a counter} \]

\[ \quad \text{ incrementation of a counter} \]

We will use 4 clocks:

\[ \bullet u, \text{“tic” clock (each time unit)} \]

\[ \bullet x_0, x_1, x_2: \text{ reference clocks for the two counters} \]

\[ “x_i \text{ reference for } c” \equiv “\text{the last time } x_i \text{ has been reset is the last time action } c \text{ has been performed”} \]
Undecidability proof (cont.)

- **Incrementation of counter c:**
  \[ x_0 \leq 2, \ u + x_2 = 1, \ c, \ x_2 := 0 \]
  
  \[ x_2 := 0 \]
  
  \[ u = 1, \ * , \ u := 0 \]

  ref for c is \( x_0 \)

  \[ x_0 > 2, \ c, \ x_2 := 0 \]
  
  \[ u + x_2 = 1 \]

- **Decrementation of counter c:**
  \[ x_0 < 2, \ u + x_2 = 1, \ c, \ x_2 := 0 \]
  
  \[ x_2 := 0 \]
  
  \[ u = 1, \ * , \ u := 0 \]

  \[ x_0 = 2, \ c, \ x_2 := 0 \]
  
  \[ u + x_2 = 1 \]

  \[ u = 1, \ x_0 = 2, \ * , \ u := 0, \ x_2 := 0 \]
Adding constraints of the form $x + y \sim c$

- **Two clocks**: decidable using the abstraction

- **Four clocks (or more)**: undecidable!
Adding constraints of the form $x + y \sim c$

- **Two clocks**: decidable using the abstraction

```
  y

  2

  1

  0 1 2 3
```

- **Three clocks**: open question

- **Four clocks (or more)**: undecidable!
Adding new operations on clocks

Several types of updates: \( x := y + c \), \( x <: c \), \( x :> c \), etc...
Adding new operations on clocks

Several types of updates: \( x := y + c, x <: c, x :> c \), etc...

- The general model is **undecidable**.
  (simulation of a two-counter machine)
Adding new operations on clocks

Several types of updates: \( x := y + c, \ x < c, \ x > c \), etc...

- The general model is **undecidable**.
  (simulation of a two-counter machine)

- Only decrementation also leads to undecidability

  - **Incrementation of counter \( x \)**
  
  \[
  \begin{align*}
  z &= 0 \\
  z &= 1, \ z := 0 \\
  z &= 0, \ y := y - 1
  \end{align*}
  \]

  - **Decrementation of counter \( x \)**
  
  \[
  \begin{align*}
  z &= 0 \\
  x &\geq 1 \\
  z &= 0, \ x := x - 1
  \end{align*}
  \]
Decidability

$y := 0 \quad \rightarrow \quad y := 1 \quad \rightarrow \quad x - y < 1$

The classical region automaton construction is not correct.

$\leadsto$ the bisimulation property is not met
Decidability (cont.)

\[ A \leadsto \text{Diophantine linear inequations system} \]
\[ \leadsto \text{is there a solution?} \]
\[ \leadsto \text{if yes, belongs to a decidable class} \]

Examples:

- constraint \( x \sim c \)
  \[ c \leq \max_x \]
- constraint \( x - y \sim c \)
  \[ c \leq \max_{x,y} \]
- update \( x :\sim y + c \)
  \[ \max_x \leq \max_y + c \]
  and for each clock \( z \), \( \max_{x,z} \geq \max_{y,z} + c \), \( \max_{z,x} \geq \max_{z,y} - c \)
- update \( x :< c \)
  \[ c \leq \max_x \]
  and for each clock \( z \), \( \max_z \geq c + \max_{z,x} \)

The constants \((\max_x)\) and \((\max_{x,y})\) define a set of regions.
Decidability (cont.)

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

\[
\begin{align*}
  \max_y &\geq 0 \\
  \max_x &\geq 0 + \max_{x,y} \\
  \max_y &\geq 1 \\
  \max_x &\geq 1 + \max_{x,y} \\
  \max_{x,y} &\geq 1
\end{align*}
\]

implies

\[
\begin{align*}
  \max_x & = 2 \\
  \max_y & = 1 \\
  \max_{x,y} & = 1 \\
  \max_y, x & = -1
\end{align*}
\]

The bisimulation property is met.
What’s wrong when undecidable?

Decrementation \( x := x - 1 \)

\[ \max_x \leq \max_x - 1 \]
What’s wrong when undecidable?

**Decrementation** $x := x - 1$

$$\max_x \leq \max_x - 1$$
What’s wrong when undecidable?

**Decrementation** \( x := x - 1 \)

\[ \max_x \leq \max_x - 1 \]
What’s wrong when undecidable?

**Decrementation** \( x := x - 1 \)

\[ \max_x \leq \max_x - 1 \]
What’s wrong when undecidable?

**Decrementation**  $x := x - 1$

$$\max_x \leq \max_x - 1$$
What’s wrong when undecidable?

**Decrementation** $x := x - 1$

$max_x \leq max_x - 1$
What’s wrong when undecidable?

**Decrementation** $x := x - 1$

$max_x \leq max_x - 1$

![Graph](image)
Decidability (cont.)

<table>
<thead>
<tr>
<th></th>
<th>Diagonal-free constraints</th>
<th>General constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x := c$, $x := y$</td>
<td>PSPACE-complete</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>$x := x + 1$</td>
<td></td>
<td>Undecidable</td>
</tr>
<tr>
<td>$x := y + c$</td>
<td>Undecidable</td>
<td></td>
</tr>
<tr>
<td>$x := x - 1$</td>
<td></td>
<td>Undecidable</td>
</tr>
<tr>
<td>$x &lt; c$</td>
<td></td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>$x &gt; c$</td>
<td></td>
<td>Undecidable</td>
</tr>
<tr>
<td>$x \sim y + c$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y + c &lt; x &lt; y + d$</td>
<td></td>
<td>Undecidable</td>
</tr>
<tr>
<td>$y + c &lt; x &lt; z + d$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Outline

1. Introduction

2. The timed automaton model

3. Timed automata, decidability issues

4. How far can we extend the model and preserve decidability?
   Hybrid systems
   Smaller extensions of timed automata
   An alternative way of proving decidability

5. Timed automata in practice

6. Conclusion
The example of alternating timed automata

Alternating timed automata \(\equiv\) ATA

\[\text{[LW05,OW05]}\]

---

[LW05] Lasota, Walukiewicz. Alternating timed automata (FoSSaCS’05).
[OW05] Ouaknine, Worrell. On the decidability of Metric Temporal Logic (LICS’05).
The example of alternating timed automata

**Alternating timed automata** $\equiv$ ATA

**Example**

“No two $a$’s are separated by 1 unit of time”

\[
\begin{align*}
\ell_0, a, \text{true} & \quad \mapsto \quad \ell_0 \land (x := 0, \ell_1) \\
\ell_1, a, x \neq 1 & \quad \mapsto \quad \ell_1 \\
\ell_1, a, x = 1 & \quad \mapsto \quad \ell_2 \\
\ell_2, a, \text{true} & \quad \mapsto \quad \ell_2
\end{align*}
\]

\[
\begin{align*}
\ell_0 & \quad \text{initial state} \\
\ell_0, \ell_1 & \quad \text{final states} \\
\ell_2 & \quad \text{losing state}
\end{align*}
\]

[**LW05**] Lasota, Walukiewicz. Alternating timed automata (FoSSaCS’05).

[**OW05**] Ouaknine, Worrell. On the decidability of Metric Temporal Logic (LICS’05).
The example of alternating timed automata

Alternating timed automata $\equiv$ ATA

Example

“No two $a$’s are separated by 1 unit of time”

\[
\begin{align*}
\ell_0, a, \text{true} & \quad \Rightarrow \quad \ell_0 \land (x := 0, \ell_1) \\
\ell_1, a, x \neq 1 & \quad \Rightarrow \quad \ell_1 \\
\ell_1, a, x = 1 & \quad \Rightarrow \quad \ell_2 \\
\ell_2, a, \text{true} & \quad \Rightarrow \quad \ell_2
\end{align*}
\]

\[
\begin{align*}
\ell_0 & \text{ initial state} \\
\ell_0, \ell_1 & \text{ final states} \\
\ell_2 & \text{ losing state}
\end{align*}
\]
nice closure properties

[Sch02] Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity (Information Processing Letters).
nice closure properties

\[ \leadsto \text{universality is as difficult as reachability} \]

[Sch02] Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity (Information Processing Letters).
- nice closure properties
  \[\leadsto\] universality is as difficult as reachability
- more expressive than timed automata

[Sch02] Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity (*Information Processing Letters*).
- nice closure properties
  \[ \leadsto \text{universality is as difficult as reachability} \]
- more expressive than timed automata

**Theorem**
- Emptiness of ATA is undecidable.
- Emptiness of one-clock ATA is decidable, but non-primitive recursive.
- Emptiness for Büchi properties of one-clock ATA is undecidable.
- Emptiness of one-clock ATA with $\varepsilon$-transitions is undecidable.

[Sch02] Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity (*Information Processing Letters*).
- nice closure properties
  \[ \leadsto \text{universality is as difficult as reachability} \]
- more expressive than timed automata

**Theorem**

- Emptiness of ATA is undecidable.
- Emptiness of one-clock ATA is decidable, but non-primitive recursive.
- Emptiness for Büchi properties of one-clock ATA is undecidable.
- Emptiness of one-clock ATA with ε-transitions is undecidable.

**Lower bound:** simulation of a lossy channel system...

[Sch02] Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity (Information Processing Letters).
Example

\[ l_0 \xrightarrow{a} l_0 \]
\[ l_0 \xrightarrow{x := 0} l_1 \]
\[ l_1 \xrightarrow{x = 1, a} l_2 \]
\[ l_1 \xrightarrow{x \neq 1, a} l_1 \]
\[ l_2 \xrightarrow{a} l_2 \]

How far can we extend the model and preserve decidability?
Example

Execution over timed word \((a, .3)(a, .8)(a, 1.4)(a, 1.8)(a, 2)\)
Example

Execution over timed word $(a, .3)(a, .8)(a, 1.4)(a, 1.8)(a, 2)$

\{ $(l_0, 0)$ \}
Example

Execution over timed word \((a, .3)(a, .8)(a, 1.4)(a, 1.8)(a, 2)\)

\[
\{ (\ell_0, 0) \} \\
\downarrow \\
\{ (\ell_0, .3), (\ell_1, 0) \}
\]
How far can we extend the model and preserve decidability?

Example

Execution over timed word \((a, .3)(a, .8)(a, 1.4)(a, 1.8)(a, 2)\)
Example

Execution over timed word \((a, .3)(a, .8)(a, 1.4)(a, 1.8)(a, 2)\)

\[
\begin{align*}
\{ (l_0, 0) \} \\
\downarrow \\
\{ (l_0, .3), (l_1, 0) \} \\
\downarrow \\
\{ (l_0, .8), (l_1, 0), (l_1, .5) \} \\
\downarrow \\
\{ (l_0, 1.4), (l_1, 0), (l_1, .6), (l_1, 1.1) \}
\end{align*}
\]
Example

\[ l_0 \quad \xrightarrow{x := 0} \quad l_1 \quad \xrightarrow{x \neq 1, a} \quad l_2 \]

Execution over timed word \((a, .3)(a, .8)(a, 1.4)(a, 1.8)(a, 2)\)

\[
\{ (l_0, 0) \} \\
\downarrow
\{ (l_0, .3), (l_1, 0) \} \\
\downarrow
\{ (l_0, .8), (l_1, 0), (l_1, .5) \} \\
\downarrow
\{ (l_0, 1.4), (l_1, 0), (l_1, .6), (l_1, 1.1) \} \\
\downarrow
\{ (l_0, 1.8), (l_1, 0), (l_1, .4), (l_2, 1), (l_1, 1.5) \}
\]
Example

\[ x := 0 \]
\[ x = 1, a \]
\[ x \neq 1, a \]

Execution over timed word \((a, .3)(a, .8)(a, 1.4)(a, 1.8)(a, 2)\)

\[
\begin{array}{c}
\{ (l_0, 0) \} \\
\downarrow \\
\{ (l_0, .3), (l_1, 0) \} \\
\downarrow \\
\{ (l_0, .8), (l_1, 0), (l_1, .5) \} \\
\downarrow \\
\{ (l_0, 1.4), (l_1, 0), (l_1, .6), (l_1, 1.1) \} \\
\downarrow \\
\{ (l_0, 1.8), (l_1, 0), (l_1, .4), (l_2, 1), (l_1, 1.5) \} \\
\downarrow \\
\{ (l_0, 2), (l_1, 0), (l_1, .2), (l_1, .6), (l_2, 1.2), (l_1, 1.7) \}
\end{array}
\]
An abstraction

A configuration = a finite set of pairs \((\ell, x)\)

\[(\ell, 0) \quad (\ell, 0.3) \quad (\ell, 1.2) \quad (\ell, 2.3) \quad (\ell', 0.4) \quad (\ell', 1) \quad (\ell', 0.8)\]
An abstraction

A configuration = a finite set of pairs \((\ell, x)\)

\[
(\ell, 0) \quad (\ell, 0.3) \quad (\ell, 1.2) \quad (\ell, 2.3) \quad (\ell', 0.4) \quad (\ell', 1) \quad (\ell', 0.8)
\]

\[
\{(\ell, 0), (\ell', 1)\}
\]

0.0
An abstraction

A configuration = a finite set of pairs \((\ell, x)\)

\[(\ell, 0) \quad (\ell, 0.3) \quad (\ell, 1.2) \quad (\ell, 2.3) \quad (\ell', 0.4) \quad (\ell', 1) \quad (\ell', 0.8)\]

\[\{(\ell, 0), (\ell', 1)\} \quad \{(\ell, 1)\}\]

0.0 \quad 0.2
An abstraction

A configuration = a finite set of pairs \((\ell, x)\)

\[
\begin{align*}
(\ell, 0) & \quad (\ell, 0.3) & \quad (\ell, 1.2) & \quad (\ell, 2.3) & \quad (\ell', 0.4) & \quad (\ell', 1) & \quad (\ell', 0.8) \\
\{(\ell, 0), (\ell', 1)\} & \quad \{(\ell, 1)\} & \quad \{(\ell, 0), (\ell, 2)\} & \quad 0.0 & \quad 0.2 & \quad 0.3
\end{align*}
\]
An abstraction

A configuration = a finite set of pairs \((\ell, x)\)

\[
\begin{align*}
(\ell, 0) & \quad (\ell, 0.3) & \quad (\ell, 1.2) & \quad (\ell, 2.3) & \quad (\ell', 0.4) & \quad (\ell', 1) & \quad (\ell', 0.8) \\
\{(\ell, 0), (\ell', 1)\} & \quad \{(\ell, 1)\} & \quad \{(\ell, 0), (\ell, 2)\} & \quad \{(\ell', 0)\} \\
0.0 & \quad 0.2 & \quad 0.3 & \quad 0.4
\end{align*}
\]
An abstraction

A configuration = a finite set of pairs \((\ell, x)\)

\[
\begin{align*}
(\ell, 0) &\quad (\ell, 0.3) &\quad (\ell, 1.2) &\quad (\ell, 2.3) &\quad (\ell', 0.4) &\quad (\ell', 1) &\quad (\ell', 0.8) \\
\{ (\ell, 0), (\ell', 1) \} &\quad \{ (\ell, 1) \} &\quad \{ (\ell, 0), (\ell, 2) \} &\quad \{ (\ell', 0) \} &\quad \{ (\ell', 0) \} \\
0.0 &\quad 0.2 &\quad 0.3 &\quad 0.4 &\quad 0.8
\end{align*}
\]
An abstraction

A configuration = a finite set of pairs \((\ell, x)\)

\[(\ell, 0) \quad (\ell, 0.3) \quad (\ell, 1.2) \quad (\ell, 2.3) \quad (\ell', 0.4) \quad (\ell', 1) \quad (\ell', 0.8)\]

\[
\begin{align*}
\{(\ell, 0), (\ell', 1)\} & \quad \{(\ell, 1)\} & \quad \{(\ell, 0), (\ell, 2)\} & \quad \{(\ell', 0)\} & \quad \{(\ell', 0)\} \\
0.0 & \quad 0.2 & \quad 0.3 & \quad 0.4 & \quad 0.8
\end{align*}
\]
An abstraction

A configuration = a finite set of pairs \((\ell, x)\)

\[
\begin{align*}
(\ell, 0) & \quad (\ell, 0.3) & (\ell, 1.2) & (\ell, 2.3) & (\ell', 0.4) & (\ell', 1) & (\ell', 0.8) \\
{(\ell, 0), (\ell', 1)} & \quad {(\ell, 1)} & {(\ell, 0), (\ell, 2)} & {(\ell', 0)} & \quad {(\ell', 0)} \\
0.0 & \quad 0.2 & \quad 0.3 & \quad 0.4 & \quad 0.8 \\
\end{align*}
\]

Abstracted into:

\[
\{(\ell, 0), (\ell', 1)\} \cdot \{(\ell, 1)\} \cdot \{(\ell, 0), (\ell, 2)\} \cdot \{(\ell', 0)\} \cdot \{(\ell', 0)\}
\]
Abstract transition system

\{ (\ell, 0), (\ell', 1) \} \cdot \{ (\ell, 1) \} \cdot \{ (\ell, 0), (\ell, 2) \} \cdot \{ (\ell', 0) \} \cdot \{ (\ell', 0) \}
Abstract transition system

\{(\ell, 0), (\ell', 1)\} \cdot \{(\ell, 1)\} \cdot \{(\ell, 0), (\ell, 2)\} \cdot \{(\ell', 0)\} \cdot \{(\ell', 0)\}

Time successors:
Abstract transition system

{((\ell, 0), (\ell', 1))} \cdot \{(\ell, 1)\} \cdot \{(\ell, 0), (\ell, 2)\} \cdot \{(\ell', 0)\} \cdot \{(\ell', 0)\}

Time successors:

{((\ell', 1))} \cdot \{(\ell, 0), (\ell', 1)\} \cdot \{(\ell, 1)\} \cdot \{(\ell, 0), (\ell, 2)\} \cdot \{(\ell', 0)\}
Abstract transition system

({(ℓ, 0), (ℓ', 1)} • {(ℓ, 1)} • {(ℓ, 0), (ℓ, 2)} • {(ℓ', 0)} • {(ℓ', 0)})

Time successors:

{((ℓ', 1)} • {(ℓ, 0), (ℓ', 1)} • {(ℓ, 1)} • {(ℓ, 0), (ℓ, 2)} • {(ℓ', 0)}

{((ℓ', 1)} • {((ℓ, 1)} • {((ℓ, 0), (ℓ', 1)} • {((ℓ, 1)} • {((ℓ, 0), (ℓ, 2)}}
Abstract transition system

Time successors:
Abstract transition system

Time successors:
Abstract transition system

Time successors:
Abstract transition system

Time successors:

Transition $\ell \xrightarrow{x>2,x:=0} \ell''$: 
Abstract transition system

Time successors:

Transition $\ell \xrightarrow{x>2,x:=0} \ell''$:
What can we do with that abstract transition system?

Correctness?
What can we do with that abstract transition system?

Correctness?

The previous abstraction is (almost) a time-abstract bisimulation.
What can we do with that abstract transition system?

Correctness?
The previous abstraction is (almost) a time-abstract bisimulation.

Termination?
What can we do with that abstract transition system?

Correctness?

The previous abstraction is (almost) a time-abstract bisimulation.

Termination?

😢 possibly infinitely many abstract configurations
What can we do with that abstract transition system?

Correctness?
The previous abstraction is (almost) a time-abstract bisimulation.

Termination?

😊 possibly infinitely many abstract configurations  
😊 there is a well-quasi ordering on the set of abstract configurations!  
(subword relation \(\sqsubseteq\))
What can we do with that abstract transition system?

Correctness?
The previous abstraction is (almost) a time-abstract bisimulation.

Termination?

😊 possibly infinitely many abstract configurations
😊 there is a well-quasi ordering on the set of abstract configurations!
    (subword relation \( \sqsubseteq \))
    + downward compatibility:
      \[
      (\gamma_1 \sqsubseteq \gamma_1' \text{ and } \gamma_1' \leadsto \gamma_2') \Rightarrow (\gamma_1 \leadsto^* \gamma_2 \text{ and } \gamma_2 \sqsubseteq \gamma_2')
      \]
What can we do with that abstract transition system?

Correctness?
The previous abstraction is (almost) a time-abstract bisimulation.

Termination?

😊 possibly infinitely many abstract configurations
😊 there is a well-quasi ordering on the set of abstract configurations! (subword relation \( \sqsubseteq \))
   + downward compatibility:
     \[
     (\gamma_1 \sqsubseteq \gamma'_1 \text{ and } \gamma'_1 \sim \gamma'_2) \Rightarrow (\gamma_1 \sim^* \gamma_2 \text{ and } \gamma_2 \sqsubseteq \gamma'_2)
     \]
   + downward-closed objective (all states are accepting)
What can we do with that abstract transition system?

Correctness?
The previous abstraction is (almost) a time-abstract bisimulation.

Termination?

😊 possibly infinitely many abstract configurations
😊 there is a well-quasi ordering on the set of abstract configurations!
   (subword relation $\sqsubseteq$)
   + downward compatibility:
     \[(\gamma_1 \sqsubseteq \gamma_1' \text{ and } \gamma_1' \sim \gamma_2') \Rightarrow (\gamma_1 \sim^* \gamma_2 \text{ and } \gamma_2 \sqsubseteq \gamma_2')\]
   + downward-closed objective (all states are accepting)

A recipe:
What can we do with that abstract transition system?

Correctness?
The previous abstraction is (almost) a **time-abstract bisimulation**.

Termination?

😊 possibly infinitely many abstract configurations
😊 there is a well-quasi ordering on the set of abstract configurations!
  (subword relation $\sqsubseteq$)
  + downward compatibility:
    
    $$(\gamma_1 \sqsubseteq \gamma'_1 \text{ and } \gamma'_1 \sim \gamma'_2 \implies (\gamma_1 \sim^* \gamma_2 \text{ and } \gamma_2 \sqsubseteq \gamma'_2))$$
  + downward-closed objective (all states are accepting)

A recipe:

$$(\text{Higman's lemma } + \text{ Koenig's lemma}) \implies \text{termination}$$
What can we do with that abstract transition system?

Correctness?
The previous abstraction is (almost) a time-abstract bisimulation.

Termination?

😊 possibly infinitely many abstract configurations
😊 there is a well-quasi ordering on the set of abstract configurations!
  (subword relation $\sqsubseteq$)
  + downward compatibility:
    $$(\gamma_1 \sqsubseteq \gamma'_1 \text{ and } \gamma'_1 \sim \gamma'_2) \Rightarrow (\gamma_1 \sim^* \gamma_2 \text{ and } \gamma_2 \sqsubseteq \gamma'_2)$$
  + downward-closed objective (all states are accepting)

A recipe:

$$(\text{Higman’s lemma } + \text{ Koenig’s lemma}) \Rightarrow \text{termination}$$

Alternative

The abstract transition system can be simulated by a kind of FIFO channel machine.
A digression on timed automata
A digression on timed automata

\[ x, y \in r_0, \{y\} < \{x\} \]

\[ (y, r_0) \cdot (x, r_0) \]
A digression on timed automata

\[ x \in r_1, \ y \in r_0, \ \{x\} < \{y\} \]

\[(x, r_1) \cdot (y, r_0)\]
A digression on timed automata

How far can we extend the model and preserve decidability?

\[ x, y \in r_1, \{y\} < \{x\} \]
A digression on timed automata

The classical region automaton can be simulated by a channel machine (with a single bounded channel).
Partial conclusion

Similar technics apply to:

- networks of single-clock timed automata

[Abdulla, Jonsson 1998]
Partial conclusion

Similar technics apply to:

- networks of single-clock timed automata
  [Abdulla,Jonsson 1998]
- timed Petri nets
  [Abdulla,Nylén 2001]
Partial conclusion

Similar technics apply to:

- networks of single-clock timed automata  
  [Abdulla, Jonsson 1998]
- timed Petri nets  
  [Abdulla, Nylén 2001]
- MTL model checking  
  [Ouaknine, Worrell 2005, 2007]
Partial conclusion

Similar technics apply to:

- networks of single-clock timed automata [Abdulla, Jonsson 1998]
- timed Petri nets [Abdulla, Nylén 2001]
- MTL model checking [Ouaknine, Worrell 2005, 2007]
- coFlatMTL model checking [Bouyer, Markey, Ouaknine, Worrell 2007]
  (using channel machines with a bounded number of cycles)
Partial conclusion

Similar technics apply to:
- networks of single-clock timed automata [Abdulla, Jonsson 1998]
- timed Petri nets [Abdulla, Nylén 2001]
- MTL model checking [Ouaknine, Worrell 2005, 2007]
- coFlatMTL model checking [Bouyer, Markey, Ouaknine, Worrell 2007]
  (using channel machines with a bounded number of cycles)
- single-clock automata inclusion checking [Ouaknine, Worrell 2004]
Partial conclusion

Similar technics apply to:

- networks of single-clock timed automata [Abdulla, Jonsson 1998]
- timed Petri nets [Abdulla, Nylén 2001]
- MTL model checking [Ouaknine, Worrell 2005, 2007]
- coFlatMTL model checking [Bouyer, Markey, Ouaknine, Worrell 2007] (using channel machines with a bounded number of cycles)
- single-clock automata inclusion checking [Ouaknine, Worrell 2004]
- ...
Outline

1. Introduction

2. The timed automaton model

3. Timed automata, decidability issues

4. How far can we extend the model and preserve decidability?
   Hybrid systems
   Smaller extensions of timed automata
   An alternative way of proving decidability

5. Timed automata in practice

6. Conclusion
What about the practice?

- the region automaton is never computed
- instead, symbolic computations are performed

What do we need?

- Need of a symbolic representation:
  Finite representation of infinite sets of configurations
What about the practice?

- the region automaton is never computed
- instead, symbolic computations are performed

What do we need?

- Need of a symbolic representation:
  - Finite representation of infinite sets of configurations

- in the plane, a line represented by two points.
What about the practice?

- the region automaton is never computed
- instead, symbolic computations are performed

What do we need?

- Need of a symbolic representation:
  
  Finite representation of infinite sets of configurations

- in the plane, a line represented by two points.
- set of words \( aa, aaaa, aaaaaa... \) represented by a rational expression \( aa(aa)^* \)
What about the practice?

- the region automaton is never computed
- instead, symbolic computations are performed

What do we need?

- Need of a symbolic representation:

  Finite representation of infinite sets of configurations

- in the plane, a line represented by two points.

- set of words *aa, aaaa, aaaaaa...* represented by a rational expression *aa(aa)*

- set of integers, represented using **semi-linear** sets
What about the practice?

- the region automaton is never computed
- instead, symbolic computations are performed

What do we need?

- Need of a symbolic representation:
  - Finite representation of infinite sets of configurations
  - in the plane, a line represented by two points.
  - set of words $aa$, $aaaa$, $aaaaaa...$ represented by a rational expression $aa(aa)^*$
  - set of integers, represented using semi-linear sets
  - sets of constraints, polyhedra, zones, regions
What about the practice?

- the region automaton is never computed
- instead, symbolic computations are performed

What do we need?

- Need of a symbolic representation:
  - Finite representation of infinite sets of configurations

  - in the plane, a line represented by two points.
  - set of words $aa$, $aaaa$, $aaaaaa$... represented by a rational expression $aa(aa)^*$
  - set of integers, represented using semi-linear sets
  - sets of constraints, polyhedra, zones, regions
  - BDDs, DBMs (see later), CDDs, etc...
What about the practice?

- the region automaton is never computed
- instead, symbolic computations are performed

What do we need?

- Need of a symbolic representation:
  
  Finite representation of infinite sets of configurations

  - in the plane, a line represented by two points.
  - set of words $aa$, $aaaa$, $aaaaaa$...
    represented by a rational expression $aa(aa)^*$
  - set of integers, represented using semi-linear sets
  - sets of constraints, polyhedra, zones, regions
  - BDDs, DBMs (see later), CDDs, etc...

- Need of abstractions, heuristics, etc...
An example of computation with HyTech

command: /usr/local/bin/hytech gas_burner

HyTech: symbolic model checker for embedded systems
Version 1.04f (last modified 1/24/02) from v1.04a of 12/6/96
For more info:
  email: hytech@eecs.berkeley.edu
  http://www.eecs.berkeley.edu/~tah/HyTech
Warning: Input has changed from version 1.00(a). Use -i for more info

Backward computation
Number of iterations required for reachability: 6
System satisfies non-leaking duration property

Location: not_leaking
x >= 0 & t >= 3 & y <= 20t & y >= 0
  | x + 20t >= y + 11 & y <= 20t + 19 & t >= 2 & x >= 0 & y >= 0
  | y >= 0 & t >= 1 & x + 20t >= y + 22 & y <= 20t + 8 & x >= 0
  | y >= 0 & x + 20t >= y + 33 & 20t >= y + 3 & x >= 0

Location: leaking
19x + y <= 20t + 19 & y >= x + 59 & x <= 1 & x >= 0
  | t >= x + 2 & x <= 1 & y >= 0 & 19x + y <= 20t + 19 & x >= 0
  | t >= x + 1 & x <= 1 & y >= 0 & 19x + y <= 20t + 8 & x >= 0
  | 20t >= 19x + y + 3 & y >= 0 & x <= 1 & x >= 0

Max memory used = 0 pages = 0 bytes = 0.00 MB
Time spent = 0.02u + 0.00s = 0.02 sec total
Zones: A symbolic representation for timed systems

Example of a zone and its DBM representation

\[ Z = (x_1 \geq 3) \land (x_2 \leq 5) \land (x_1 - x_2 \leq 4) \]

DBM: Difference Bound Matrice [BM83,Dill89]

[BM83] Berthomieu, Menasche. An enumerative approach for analyzing time Petri nets World Comuter Congress.
Zones: A symbolic representation for timed systems

Example of a zone and its DBM representation

\[ Z = (x_1 \geq 3) \land (x_2 \leq 5) \land (x_1 - x_2 \leq 4) \]

```
x_2
5
2

3 4 9
x_1
```

```
x_0 \ x_1 \ x_2
0  -3  0
9   0  4
5   2  0
```

“normal form”

DBM: Difference Bound Matrice [BM83,Dill89]

[BM83] Berthomieu, Menasche. An enumerative approach for analyzing time Petri nets World Comuter Congress.
Backward computation
Backward computation
Backward computation
Backward computation
Backward computation
Note on the backward analysis of TA

\[ [C \leftarrow 0]^{-1}(Z \cap (C = 0)) \cap g \]

\[ g, \ a, \ C := 0 \]

\[ Z \]
Note on the backward analysis of TA

\[ g, \ a, \ C := 0 \]

\[ [C \leftarrow 0]^{-1}(Z \cap (C = 0)) \cap g \]

\[ Z \]
Note on the backward analysis of TA

\[ [C \leftarrow 0]^{-1}(Z \cap (C = 0)) \cap g \]

\[ Z \]

\[ [C \leftarrow 0]^{-1}(Z \cap (C = 0)) \]
Note on the backward analysis of TA

\[ \ell \xrightarrow{g, a, C := 0} \ell' \]

\[ \text{[} C \leftarrow 0 \text{]}^{-1}(Z \cap (C = 0)) \cap g \quad \text{Z} \]

\[ \text{Z} \quad \text{[} C \leftarrow 0 \text{]}^{-1}(Z \cap (C = 0)) \]
Note on the backward analysis of TA

\[ \ell \xrightarrow{g, a, C := 0} \ell' \]

\[ [C \leftarrow 0]^{-1}(Z \cap (C = 0)) \cap g \]

\[ Z \]

\[ [C \leftarrow 0]^{-1}(Z \cap (C = 0)) \]

\[ [C \leftarrow 0]^{-1}(Z \cap (C = 0)) \cap g \]
Note on the backward analysis of TA

$\ell \xrightarrow{g, \ a, \ C := 0} \ell'$

$[C \leftarrow 0]^{-1}(Z \cap (C = 0)) \cap g$

$Z$

Z  $[C \leftarrow 0]^{-1}(Z \cap (C = 0))$

$\leftarrow [C \leftarrow 0]^{-1}(Z \cap (C = 0)) \cap g$

😊 the backward computation always terminates!

😊 😊 ... and it is correct!!!
Note on the backward analysis (cont.)

If $\mathcal{A}$ is a timed automaton, we construct its corresponding set of regions.

Because of the bisimulation property, we get that:

“Every set of valuations which is computed along the backward computation is a finite union of regions”
Note on the backward analysis (cont.)

If $\mathcal{A}$ is a timed automaton, we construct its corresponding set of regions.

Because of the bisimulation property, we get that:

“Every set of valuations which is computed along the backward computation is a finite union of regions”

Let $R$ be a region. Assume:

- $v \in \bar{R}$ (for ex. $v + t \in R$)
- $v' \equiv_{\text{reg.}} v$

There exists $t'$ s.t. $v' + t' \equiv_{\text{reg.}} v + t$, which implies that $v' + t' \in R$ and thus $v' \in \bar{R}$. 
Note on the backward analysis (cont.)

If $\mathcal{A}$ is a timed automaton, we construct its corresponding set of regions.

Because of the bisimulation property, we get that:

“Every set of valuations which is computed along the backward computation is a finite union of regions”

But, the backward computation is not so nice, when also dealing with integer variables...

\[ i := j \cdot k + \ell \cdot m \]
Forward computation
Forward computation
Forward computation
Forward computation
Forward computation
Forward analysis of timed automata

\[ g, \ a, \ C := 0 \]

zones \[ Z \]

\[ [C \leftarrow 0](\overrightarrow{Z} \cap g) \]
Forward analysis of timed automata

\[ g, \ a, \ C := 0 \]

zones \( Z \) \[ [C \leftarrow 0](\overrightarrow{Z} \cap g) \]
Forward analysis of timed automata

\[ g, \ a, \ C := 0 \]

\[ \ell \rightarrow \ell' \]

zones \( Z \)

\([C \leftarrow 0](\vec{Z} \cap g)\)

\( \vec{Z} \)
Forward analysis of timed automata

$g, \ a, \ C := 0$

zones $Z$

$\left[ C \leftarrow 0 \right](\vec{\tilde{Z}} \cap g)$

$Z$

$\vec{Z}$

$\vec{Z} \cap g$
Forward analysis of timed automata

\[ g, a, C := 0 \]

zones \[ Z \]

\[ [C \leftarrow 0](\overrightarrow{Z} \cap g) \]

\[ Z \]

\[ \overrightarrow{Z} \]

\[ \overrightarrow{Z} \cap g \]

\[ [y \leftarrow 0](\overrightarrow{Z} \cap g) \]
Forward analysis of timed automata

g, a, C := 0

zones

\[ Z \]

\[ \overrightarrow{Z} \cap g \]

\[ [y \leftarrow 0](\overrightarrow{Z} \cap g) \]

😢 the forward computation may not terminate...
Non termination of the forward analysis

\[ y := 0, \]
\[ x := 0 \]
\[ x \geq 1 \land y = 1, \]
\[ y := 0 \]
Non termination of the forward analysis

\( y := 0, \)
\( x := 0 \)
\( x \geq 1 \land y = 1, \)
\( y := 0 \)
Non termination of the forward analysis

\[ y := 0, \]
\[ x := 0 \]
\[ x \geq 1 \land y = 1, \]
\[ y := 0 \]
Non termination of the forward analysis

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

\[ x \geq 1 \land y = 1, \]
\[ y := 0 \]
Non termination of the forward analysis

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

\[x \geq 1 \land y = 1,
\]

\[y := 0\]

\[\leadsto \text{an infinite number of steps...}\]
“Solutions” to this problem

(f.ex. in [Larsen, Pettersson, Yi 1997] or in [Daws, Tripakis 1998])

- inclusion checking: if $Z \subseteq Z'$ and $Z'$ already considered, then we don’t need to consider $Z$

$\leadsto$ correct w.r.t. reachability
“Solutions” to this problem

(f.ex. in [Larsen, Pettersson, Yi 1997] or in [Daws, Tripakis 1998])

- **inclusion checking**: if $Z \subseteq Z'$ and $Z'$ already considered, then we don’t need to consider $Z$
  \[\leadsto\text{correct w.r.t. reachability}\]

- **activity**: eliminate redundant clocks
  \[\text{[Daws, Yovine 1996]}\]
  \[\leadsto\text{correct w.r.t. reachability}\]

\[q \xrightarrow{g,a,C:=0} q' \implies \text{Act}(q) = \text{clocks}(g) \cup (\text{Act}(q') \setminus C)\]

\[\ldots\]
“Solutions” to this problem (cont.)

- **convex-hull approximation**: if $Z$ and $Z'$ are computed then we overapproximate using "$Z \sqcup Z'$".

  $\leadsto$ “semi-correct” w.r.t. reachability
“Solutions” to this problem (cont.)

- **convex-hull approximation**: if $Z$ and $Z'$ are computed then we overapproximate using \(Z \oplus Z'\).
  \[\leadsto \text{“semi-correct” w.r.t. reachability}\]

- **extrapolation**, an abstraction operator on zones
An abstraction: the extrapolation operator

$\text{Approx}_2(Z)$: “the smallest zone containing $Z$ that is defined only with constants no more than 2”

\[
\begin{pmatrix}
0 & -3 & 0 \\
9 & 0 & 4 \\
5 & 2 & 0
\end{pmatrix}
\]

$\leadsto$ The extrapolation operator ensures termination of the computation!
An abstraction: the extrapolation operator

$\text{Approx}_2(Z)$: “the smallest zone containing $Z$ that is defined only with constants no more than 2”

\[
\begin{pmatrix}
0 & -3 & 0 \\
9 & 0 & 4 \\
5 & 2 & 0 \\
\end{pmatrix}
\xrightarrow{\text{Approx}_2}
\begin{pmatrix}
0 & -2 & 0 \\
\infty & 0 & \infty \\
\infty & 2 & 0 \\
\end{pmatrix}
\]

$\leadsto$ The extrapolation operator ensures termination of the computation!
Classical algorithm, focus on correctness

Challenge

Choose a good constant for the extrapolation so that the forward computation is correct. Classical algorithm, the choice goes to the maximal constant.

Classical algorithm, focus on correctness

Challenge

Choose a good constant for the extrapolation so that the forward computation is correct. Classical algorithm, the choice goes to the maximal constant.

- Implemented in tools like Uppaal, Kronos, RT-Spin...
- Successfully used on many real-life examples

Classical algorithm, focus on correctness

Challenge

Choose a **good** constant for the extrapolation so that the forward computation is correct. Classical algorithm, the choice goes to the maximal constant.

- Implemented in tools like Uppaal, Kronos, RT-Spin...
- Successfully used on many real-life examples

Theorem

The classical algorithm is correct for diagonal-free timed automata.

[Bou03] Bouyer. Untameable timed automata! (*STACS‘03*).
Classical algorithm, focus on correctness

Challenge

Choose a **good** constant for the extrapolation so that the forward computation is correct. Classical algorithm, the choice goes to the maximal constant.

- Implemented in tools like Uppaal, Kronos, RT-Spin...
- Successfully used on many real-life examples

Theorem

The classical algorithm is correct for diagonal-free timed automata.

This theorem does not extend to timed automata using diagonal clock constraints... [Bou03,Bou04]

A problematic automaton

x_3 \leq 3
x_1, x_3 := 0

x_2 = 3
x_2 := 0

x_1 = 2, x_1 := 0

x_2 = 2, x_2 := 0

The loop

x_1 = 3
x_1 := 0

x_2 = 2
x_2 := 0

x_1 = 3
x_1 := 0

x_2 = 2
x_2 := 0

Error

x_2 - x_1 > 2
x_4 - x_3 < 2
A problematic automaton

\[
\begin{align*}
\nu(x_1) &= 0 \\
\nu(x_2) &= d \\
\nu(x_3) &= 2\alpha + 5 \\
\nu(x_4) &= 2\alpha + 5 + d
\end{align*}
\]
A problematic automaton

\[
\begin{align*}
&\{ & v(x_1) = 0 \\
& & v(x_2) = d \\
& & v(x_3) = 2\alpha + 5 \\
& & v(x_4) = 2\alpha + 5 + d \\
\} \\
\end{align*}
\]
The problematic zone

[1; 3]  [2α + 5]  [1; 3]

implies

\[ x_1 - x_2 = x_3 - x_4. \]
The problematic zone

If $\alpha$ is sufficiently large, after extrapolation:

does not imply $x_1 - x_2 = x_3 - x_4$. 

implies $x_1 - x_2 = x_3 - x_4$. 

$[1; 3]$  
$[2\alpha + 5]$  
$x_1$  
$x_2$  
$x_3$  
$x_4$  
$[2\alpha + 2; 2\alpha + 4]$  
$[2\alpha + 6; 2\alpha + 8]$  
$> k$
The problematic zone

If \( \alpha \) is sufficiently large, after extrapolation:

Hence, any choice of constant is erroneous!
Criteria for a good abstraction operator Abs:
General abstractions

Criteria for a good abstraction operator \textbf{Abs}:

- easy computation

Abs(Z) is a zone if Z is a zone

[Effectiveness]
General abstractions

Criteria for a good abstraction operator $\text{Abs}$:

- easy computation
  $\text{Abs}(Z)$ is a zone if $Z$ is a zone
- finiteness of the abstraction
  $\{\text{Abs}(Z) \mid Z \text{ zone}\}$ is finite

[Effectiveness]

[Termination]
General abstractions

**Criteria for a good abstraction operator** \( \text{Abs} \):

- **easy computation**
  \( \text{Abs}(Z) \) is a zone if \( Z \) is a zone
- **finiteness of the abstraction**
  \( \{ \text{Abs}(Z) \mid Z \text{ zone} \} \) is finite
- **completeness of the abstraction**
  \( Z \subseteq \text{Abs}(Z) \)

[Effectiveness]

[Termination]

[Completeness]
General abstractions

Criteria for a good abstraction operator \( \text{Abs} \):

- easy computation
  \( \text{Abs}(Z) \) is a zone if \( Z \) is a zone

- finiteness of the abstraction
  \( \{ \text{Abs}(Z) \mid Z \text{ zone}\} \) is finite

- completeness of the abstraction
  \( Z \subseteq \text{Abs}(Z) \)

- soundness of the abstraction
  the computation of \((\text{Abs} \circ \text{Post})^*\) is correct w.r.t. reachability
General abstractions

Criteria for a good abstraction operator $\text{Abs}$:

- **Effectiveness**: easy computation
  
  $\text{Abs}(Z)$ is a zone if $Z$ is a zone

- **Termination**: finiteness of the abstraction
  
  $\{\text{Abs}(Z) \mid Z \text{ zone}\}$ is finite

- **Completeness**: completeness of the abstraction
  
  $Z \subseteq \text{Abs}(Z)$

- **Soundness**: soundness of the abstraction
  
  The computation of $(\text{Abs} \circ \text{Post})^*$ is correct w.r.t. reachability

For the previous automaton,

no abstraction operator can satisfy all these criteria!
Why that?

Assume there is a “nice” operator \( \text{Abs} \).

The set \( \{M \text{ DBM representing a zone } \text{Abs}(Z) \} \) is finite.

\[ \leadsto \ k \text{ the max. constant defining one of the previous DBMs} \]

We get that, for every zone \( Z \),

\[ Z \subseteq \text{Extra}_k(Z) \subseteq \text{Abs}(Z) \]
Problem!

Open questions:  - which conditions can be made weaker?
                - find a clever termination criterium?
                - use an other data structure than zones/DBMs?
Improving the classical algorithm

- the extrapolation operator can be made coarser:
  - local extrapolation constants [BBFL03];
  - distinguish between lower- and upper-bounded constraints [BBLP03,BBLP06]

[BBFL03] Behrmann, Bouyer, Fleury, Larsen. Static Guard Analysis in Timed Automata Verification (TACAS’03).
[BBLP04] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone Based Abstractions of Timed Automata (TACAS’04).
[HBL+03] Hendriks, Behrmann, Larsen, Niebert, Vaandrager. Adding symmetry reduction to Uppaal (FORMATS’03).
[DHLP06] David, Häkansson, Larsen, Pettersson. Model checking timed automata with priorities using DBM subtraction (FORMATS’06).
Improving the classical algorithm

- the extrapolation operator can be made coarser:
  - local extrapolation constants [BBFL03];
  - distinguish between lower- and upper-bounded constraints [BBLP03,BBLP06]

- heuristics can be added
  - order for exploration
  - symmetry reduction [HBL+03]

[BBFL03] Behrmann, Bouyer, Fleury, Larsen. Static Guard Analysis in Timed Automata Verification (TACAS’03).
[BBLP04] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone Based Abstractions of Timed Automata (TACAS’04).
[HBL+03] Hendriks, Behrmann, Larsen, Niebert, Vaandrager. Adding symmetry reduction to Uppaal (FORMATS’03).
[DHLP06] David, Häkansson, Larsen, Pettersson. Model checking timed automata with priorities using DBM subtraction (FORMATS’06).
Improving the classical algorithm

- the extrapolation operator can be made coarser:
  - local extrapolation constants \([BBFL03]\);
  - distinguish between lower- and upper-bounded contraints \([BBLP03,BBLP06]\)

- heuristics can be added
  - order for exploration
  - symmetry reduction \([HBL+03]\)

- the representation of zones can be improved \([DHL06]\)

---

[BBFL03] Behrmann, Bouyer, Fleury, Larsen. Static Guard Analysis in Timed Automata Verification \((TACAS'03)\).
[BBLP04] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone Based Abstractions of Timed Automata \((TACAS'04)\).
[BBLP06] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone-Based Abstractions of Timed Automata \((International Journal on Software Tools for Technology Transfer)\).
[HBL+03] Hendriks, Behrmann, Larsen, Niebert, Vaandrager. Adding symmetry reduction to Uppaal \((FORMATS'03)\).
[DHL06] David, Håkansson, Larsen, Pettersson. Model checking timed automata with priorities using DBM subtraction \((FORMATS'06)\).
Improving the classical algorithm

- the extrapolation operator can be made coarser:
  - local extrapolation constants \([\text{BBFL03}]\);
  - distinguish between lower- and upper-bounded contraints \([\text{BBLP03,BBLP06}]\)

- heuristics can be added
  - order for exploration
  - symmetry reduction \([\text{HBL+03}]\)

- the representation of zones can be improved \([\text{DHL06}]\)

\(\leadsto\) the tool Uppaal is under development since 1995...

\[\text{[BBFL03]}\] Behrmann, Bouyer, Fleury, Larsen. Static Guard Analysis in Timed Automata Verification (TACAS’03).
\[\text{[BBLP04]}\] Behrmann, Bouyer, Larsen, Pelánek. Lower and Upper Bounds in Zone Based Abstractions of Timed Automata (TACAS’04).
\[\text{[HBL+03]}\] Hendriks, Behrmann, Larsen, Niebert, Vaandrager. Adding symmetry reduction to Uppaal (FORMATS’03).
\[\text{[DHL06]}\] David, Håkansson, Larsen, Pettersson. Model checking timed automata with priorities using DBM subtraction (FORMATS’06).
Outline

1. Introduction

2. The timed automaton model

3. Timed automata, decidability issues

4. How far can we extend the model and preserve decidability?
   - Hybrid systems
   - Smaller extensions of timed automata
   - An alternative way of proving decidability

5. Timed automata in practice

6. Conclusion
Conclusion

- Justification of the dense-time paradigm
- Several technics for proving decidability of real-time systems
  - finite time-abstract bisimulation
  - well-quasi-order on the time-abstract transition system
- Timed automata are implemented in several model checking tools
  - Other timed models have been developed and have concurrent tools: for instance Romeo and Tina for time Petri nets
Conclusion

- Justification of the dense-time paradigm
- Several technics for proving decidability of real-time systems
  - finite time-abstract bisimulation
  - well-quasi-order on the time-abstract transition system
- Timed automata are implemented in several model checking tools
  - Other timed models have been developed and have concurrent tools: for instance Romeo and Tina for time Petri nets

Some current streams of research in timed systems:

- quantitative model-checking,
- real-time logics,
- robustness, implementability issues,
- timed games,
- modelling of resources,
- ...
Conclusion

- Justification of the dense-time paradigm
- Several technics for proving decidability of real-time systems
  - finite time-abstract bisimulation
  - well-quasi-order on the time-abstract transition system
- Timed automata are implemented in several model checking tools
  - Other timed models have been developed and have concurrent tools: for instance Romeo and Tina for time Petri nets

Some current streams of research in timed systems:
- quantitative model-checking,
- real-time logics,
- robustness, implementability issues,
- timed games,
- modelling of resources,
- ...