Model-checking

Does the system satisfy the property?
Model-checking

Does the system satisfy the property?

Modelling

Model-checking Algorithm
**Context:** verification of embedded critical systems

**Time**
- naturally appears in real systems
- appears in properties (for ex. bounded response time)

→ Need of models and specification languages integrating timing aspects
Outline

1. About time semantics
2. Timed automata, decidability issues
3. Some extensions of the model
4. Implementation of timed automata
5. Conclusion
About time semantics

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
  
  $a$: 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
About time semantics

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

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

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

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

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

  \(d_1\): 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 \(N\)
    
    Ex: \((a, 1)(b, 3)(c, 4)(a, 6)\)

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

**Time domain:** discrete (e.g. $N$) or dense (e.g. $Q^+$)
- Dense-time is a more general model than discrete time
- A compositionality problem with discrete time
- But, can we not always discretize?
A digital circuit

Discussion in the context of reachability problems for asynchronous digital circuits

[Alur 91]

[Brzozowski, Seger 1991]
A digital circuit

Discussion in the context of reachability problems for asynchronous digital circuits

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

Discussion in the context of reachability problems for asynchronous digital circuits

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

The input \( x \) changes to 1. The corresponding stable state is \( y=[011] \)
About time semantics

A digital circuit

Discussion in the context of reachability problems for asynchronous digital circuits

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.

$\begin{align*}
[101] & \xrightarrow{y_2} 1.2 \quad [111] & \xrightarrow{y_3} 2.5 \quad [110] & \xrightarrow{y_1} 2.8 \quad [010] & \xrightarrow{y_3} 4.5 \quad [011]
\end{align*}$
A digital circuit

Discussion in the context of reachability problems for asynchronous digital circuits

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.

\[
\begin{align*}
[101] & \xrightarrow{y_2=1.2} [111] & \xrightarrow{y_3=2.5} [110] & \xrightarrow{y_1=2.8} [010] & \xrightarrow{y_3=4.5} [011]
\end{align*}
\]

Reachable configurations: $\{[101], [111], [110], [010], [011], [001]\}$
Is discretizing sufficient? An example [Alur 91]

- This digital circuit is not 1-discretizable.
Is discretizing sufficient? An example [Alur 91]

- 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

[Alur 91]

This digital circuit **is not** 1-discretizable.

Why that? (initially \( x = 0 \) and \( y = [11100000] \), \( x \) is set to 1)

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

This digital circuit is not 1-discretizable.

Why that?

(initially $x = 0$ and $y = [11100000]$, $x$ is set to 1)

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

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

\[
\begin{align*}
[11100000] & \xrightarrow{y_1} [01100000] \xrightarrow{y_2} [00100000] \xrightarrow{y_3,y_5} [00001000] \xrightarrow{y_5,y_7} [00000010] \xrightarrow{y_7,y_8} [00000001] \\
[11100000] & \xrightarrow{y_1,y_2,y_3} [00000000] \\
[11100000] & \xrightarrow{y_1} [01111000] \xrightarrow{y_2,y_3,y_4,y_5} [00000000]
\end{align*}
\]
Is discretizing sufficient? An example

[Alur 91]

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

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

\[
\begin{align*}
[11100000] & \xrightarrow{y_1 = 1} [01100000] & \xrightarrow{y_2 = 1.5} [00100000] & \xrightarrow{y_3, y_5 = 2} [00001000] & \xrightarrow{y_5, y_7 = 3} [00000010] & \xrightarrow{y_7, y_8 = 4} [00000001] \\
[11100000] & \xrightarrow{y_1, y_2, y_3 = 1} [00000000] \\
[11100000] & \xrightarrow{y_1 = 1} [01110000] & \xrightarrow{y_2, y_3, y_4, y_5 = 2} [00000000] \\
[11100000] & \xrightarrow{y_1, y_2 = 1} [00100000] & \xrightarrow{y_3, y_5, y_6 = 2} [00001100] & \xrightarrow{y_5, y_6 = 3} [00000000]
\end{align*}
\]
**Theorem [Brzozowski Seger 1991]**

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** [Brzozowski Seger 1991]

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** [Brzozowski Seger 1991]

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.

**Going further…** There exist systems for which no granularity exists. (see later)
Outline

1. About time semantics

2. Timed automata, decidability issues

3. Some extensions of the model

4. Implementation of timed automata

5. Conclusion
Timed automata

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

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

- An enabling condition (or guard) is:

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

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

\[ x, y : \text{clocks} \]

\[ x \leq 5, \ a, \ y := 0 \]

\[ y > 1, \ b, \ x := 0 \]
Timed automata (example)

$x, y$ : clocks

$x \leq 5, \ a, \ y := 0$

$y > 1, \ b, \ x := 0$
Timed automata (example)

$x, y : \text{clocks}$

$x \leq 5, \ a, \ y := 0$

$y > 1, \ b, \ x := 0$

(clock) valuation
Timed automata (example)

$x, y : \text{clocks}$

$x \leq 5, \ a, \ y := 0 \quad y > 1, \ b, \ x := 0$

$\\delta(4.1) \quad \delta(1.4)$

$\begin{array}{c|c|c|c}
\ell_0 & \ell_0 & a & \ell_1 \\
 x & 0 & 4.1 & 4.1 \\
 y & 0 & 4.1 & 0 \\
\end{array}$

$\begin{array}{c|c|c|c}
\ell_1 & b & \ell_2 \\
 5.5 & 0 \\
 1.4 & 1.4 \\
\end{array}$

(clock) valuation

→ timed word $(a, 4.1)(b, 5.5)$
Timed automata semantics

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

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

- **Timed Transition System:**
  - **action transition:** $(\ell, v) \xrightarrow{a} (\ell', v')$ if $\exists \ell \xrightarrow{g, a, r} \ell' \in \mathcal{A}$ s.t.
    \[
    \begin{cases}
    v \models g \\
    v' = v[r \leftarrow 0]
    \end{cases}
    \]
  - **delay transition:** $(\ell, v) \xrightarrow{\delta(d)} (\ell, v + d)$ if $d \in T$
Discrete vs dense-time semantics
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 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:** \[ L_{\text{discrete}} = \emptyset \]
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:

- $G\phi$ « Always »
- $F\phi$ « Eventually »
- $\phi U \phi'$ « Until »
- $X\phi$ « Next »

State formulas:

- $A\psi$
- $E\psi$

→ LTL: Linear Temporal Logic [Pnueli 1977],
CTL: Computation Tree Logic [Emerson, Clarke 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:

\[ AG(\text{problem} \Rightarrow AF \text{ alarm}) \]
Adding time to temporal logics

Classical temporal logics allow us to express that

“any problem is followed by an alarm”

With CTL:

$$AG(\text{problem } \Rightarrow \text{AF alarm})$$

How can we express:

“any problem is followed by an alarm \textit{in at most 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:

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

How can we express:

“any problem is followed by an alarm in at most 20 time units”

- Temporal logics with **subscripts**.

  ex: \( CTL + E\varphi U_{\sim k}\psi \)

  \( A\varphi U_{\sim k}\psi \)
Adding time to temporal logics

Classical temporal logics allow us to express that

“any problem is followed by an alarm”

With CTL:

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

How can we express:

“any problem is followed by an alarm in at most 20 time units”

- Temporal logics with **subscripts**.

\[ AG(\text{problem } \Rightarrow \text{AF}_{\leq 20} \text{ alarm}) \]
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 in at most 20 time units”

- Temporal logics with **subscripts**.

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

- Temporal logics with **clocks**.

\[ \text{AG}(\text{problem } \Rightarrow (x \text{ in } \text{AF}(x \leq 20 \land \text{alarm}))) \]
Adding time to temporal logics

Classical temporal logics allow us to express that

“any problem is followed by an alarm”

With CTL:

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

How can we express:

“any problem is followed by an alarm in at most 20 time units”

- Temporal logics with **subscripts**.

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

- Temporal logics with **clocks**.

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

\[ \rightarrow \text{TCTL: Timed CTL} \quad [\text{ACD90, ACD93, HNSY94}] \]
The train crossing example

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

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

- **Before, $x_i < 30$**
  - $20 < x_i < 30, a, x_i := 0$

- **On, $x_i < 20$**
  - $x_i := 0$

The train crossing example

The gate:

- **Open** → **Lowering, \( H_g < 10 \)**: GoDown?, \( H_g := 0 \)
- **Raising, \( H_g < 10 \)** → **Close**: GoUp?, \( H_g := 0 \)
- **Lowering, \( H_g < 10 \)** → **Raising, \( H_g < 10 \)**: \( H_g < 10, a \)
- **Open** → **Raising, \( H_g < 10 \)**: \( H_g < 10, a \)
- **Close** → **Lowering, \( H_g < 10 \)**: \( H_g < 10, a \)
The train crossing example (3)

The controller:

\[ c_1, x_c \leq 20 \]

\[ H_c = 20, \text{GoUp!} \]

\[ c_0 \]

\[ H_c = 0, \text{App?} \]

\[ c_2, x_c \leq 10 \]

\[ H_c \leq 10, \text{GoDown!} \]
We use the synchronization function $f$:

<table>
<thead>
<tr>
<th>Train₁</th>
<th>Train₂</th>
<th>Gate</th>
<th>Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>App!</td>
<td>.</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 \((Train₁ \parallel Train₂ \parallel Gate \parallel Controller)\)

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

Some properties one could check:

- Is the gate closed when a train crosses the road?
Some properties one could check:

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

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

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?

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

- Is the gate always closed for less than 5 minutes?

  \[ AG \ AF_{<5\text{min}}(\neg gate.\text{Close}) \]


Verification

**Emptiness problem**: is the language accepted by a timed automaton empty?
- reachability properties (final states)
- basic liveness properties (Büchi (or other) conditions)
Verification

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

- **Problem:** the set of configurations is infinite
  - \( \Rightarrow \) classical methods can not be applied
Verification

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

- **Problem:** the set of configurations is infinite
  ➔ classical methods cannot be applied

- **Positive key point:** variables (clocks) have the same speed
Verification

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

- Problem: the set of configurations is infinite
  ➔ classical methods can not be applied

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

Theorem [Alur & Dill 1990’s]

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

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

- **Problem:** the set of configurations is infinite
  - classical methods can not be applied

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

**Theorem [Alur & Dill 1990’s]**

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

**Note:** This is also the case for the discrete semantics.
Verification

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

- **Problem:** the set of configurations is infinite
  => classical methods can not be applied

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

---

**Theorem** [Alur & Dill 1990’s]

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

---

**Method:** construct a finite abstraction
The region abstraction

Equivalence of finite index
The region abstraction

Equivalence of finite index

“compatibility” between regions and constraints
The region abstraction

“compatibility” between regions and constraints

“compatibility” between regions and time elapsing
The region abstraction

Equivalence of finite index

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

Equivalence of finite index

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

⇒ a bisimulation property
The region abstraction

Equivalence of finite index

region defined by
\( l_x = ]1; 2[ \), \( l_y = ]0; 1[ \)
\( \{x\} < \{y\} \)

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

⇒ a bisimulation property
The region abstraction

Equivalence of finite index

- region defined by
  \[ l_x = ]1; 2[, \ l_y = ]0; 1[\]
  \[ \{x\} < \{y\}\]
- successor regions

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

\(\Rightarrow\) a bisimulation property
The region automaton

timed automaton ⊗ region abstraction

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

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

- \( R'' \subseteq g \)
- \([C \leftarrow 0]R'' \subseteq R'\)

→ time-abstract bisimulation

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

where \( \text{UNTIME}(((a_1, t_1)(a_2, t_2)\ldots) = a_1a_2\ldots \)
An example [AD 90's]
Time-abstract bisimulation
Time-abstract bisimulation
Time-abstract bisimulation

∀

∃

∀> 0

δ(d)

a

∀

a

∃

a

∀

a

∀

δ(d)

∀

∃
Time-abstract bisimulation

∀a → b
∃a → b

∀d > 0 δ(d) → b
∃d' > 0 δ(d') → b
Time-abstract bisimulation

\[ \forall a \exists d > 0, \delta(d) \]

\[ \exists a \forall d' > 0, \delta(d') \]

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

∀ \quad \begin{array}{c}
\bullet \\
\end{array} \xrightarrow{a} \begin{array}{c}
\bullet
\end{array}

\exists \quad \begin{array}{c}
\bullet \\
\end{array} \xrightarrow{a} \begin{array}{c}
\bullet
\end{array}

∀ d > 0 \quad \begin{array}{c}
\bullet \\
\end{array} \xrightarrow{\delta(d)} \begin{array}{c}
\bullet
\end{array}

\exists d' > 0 \quad \begin{array}{c}
\bullet \\
\end{array} \xrightarrow{\delta(d')} \begin{array}{c}
\bullet
\end{array}

(\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{array}{c} \bullet \\ \rightarrow \end{array}^{a} \begin{array}{c} \bullet \\ \rightarrow \end{array} \\
\exists \begin{array}{c} \bullet \\ \rightarrow \end{array}^{a} \begin{array}{c} \bullet \\ \rightarrow \end{array}

∀d > 0 \begin{array}{c} \bullet \\ \rightarrow \end{array}^{\delta(d)} \begin{array}{c} \bullet \\ \rightarrow \end{array} \\
\exists d' > 0 \begin{array}{c} \bullet \\ \rightarrow \end{array}^{\delta(d')} \begin{array}{c} \bullet \\ \rightarrow \end{array}

(\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

\text{with } v_i \in R_i \text{ for all } i.
Time-abstract bisimulation

\[
\begin{align*}
\forall a & \\
\exists a & \\
\forall d > 0 & \\
\exists d' > 0 & \\
\end{align*}
\]

Remark: Real-time properties can not be checked with a time-abstract bisimulation. For TCTL, a clock associated with the formula needs to be added.
PSPACE-easiness

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

- One configuration: a discrete location + a region
The size of the region graph is in $\mathcal{O}(|X|! \cdot 2^{|X|})$.

**One configuration:** a discrete location + a region
- a discrete location: log-space
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
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
  
  → needs polynomial space
PSPACE-easiness

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
  
  $\Rightarrow$ needs polynomial space

- By guessing a path: needs only to store two configurations
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
  → needs polynomial space

- By guessing a path: needs only to store two configurations
  → in NPSPACE, thus in PSPACE
PSPACE-hardness

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

\[ \{x_j, y_j\} \quad C_j \]

- \( C_j \) 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)

\[ \rightarrow \quad \text{proof taken in [Aceto & Laroussinie 2002]} \]
PSPACE-hardness (cont.)

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

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

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: $(q_f, i) \rightarrow \text{end}$
Consequence of region automata construction

Region automata: correct finite abstraction for checking reachability/Büchi-like properties
Consequence of region automata construction

**Region automata**: correct finite abstraction for checking reachability/Büchi-like properties

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

- Universality is **undecidable**
- Inclusion is **undecidable**
- Determinizability is **undecidable**
- Complementability is **undecidable**
- ...

[Alur & Dill 90’s]
[Alur & Dill 90’s]
[Tripakis 2003]
[Tripakis 2003]
A model not far from undecidability

- Universality is undecidable
- Inclusion is undecidable
- Determinizability is undecidable
- Complementability is undecidable
- ...

An example of non-determinizable/non-complementable timed aut.:
A model not far from undecidability

- Universality is undecidable
- Inclusion is undecidable
- Determinizability is undecidable
- Complementability is undecidable
- ...

An example of non-determinizable/non-complementable timed aut.: [Alur, Madhusudan 2004]

\[
\text{UNTIME}\left(\bar{L} \cap \{(a^* b^*, \tau) \mid \text{all } a's \text{ happen before } 1 \text{ and no two } a's \text{ simultaneously} \}\right) \text{ is not regular} \quad (\text{exercise!})
\]
Partial conclusion

→ a timed model interesting for verification purposes

Numerous works have been (and are) devoted to:

- the “theoretical” comprehension of timed automata (cf [Asarin 2004])
- extensions of the model (to ease modelling)
  - expressiveness
  - analyzability
- algorithmic problems and implementation
Outline

1. About time semantics
2. Timed automata, decidability issues
3. Some extensions of the model
4. Implementation of timed automata
5. Conclusion
Some extensions of the model

Role of diagonal constraints

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

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

```
0 1 2
x
```

- **Expressiveness:** no additional expressive power
Some extensions of the model

Role of diagonal constraints (cont.)

c is positive

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

proof in [Bérard, Diekert, Gastin, Petit 1998]

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

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

copy where \( x - y > c \)
Some extensions of the model

Role of diagonal constraints (cont.)

$c$ is positive

$x - y \leq c$

$x := 0$

$y := 0$

$x := 0$

$y := 0$

$x \leq c$

$x > c$

$y := 0$

$y := 0$

$\rightarrow$ proof in [Bérard, Diekert, Gastin, Petit 1998]

$\rightarrow$ exponential blowup unavoidable in general

[Bouyer, Chevalier 2005]
Some extensions of the model

Adding silent actions

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

[Bérard, Diekert, Gastin, Petit 1998]

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

- **Expressiveness**: strictly more expressive!

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

\[ x = 1, \varepsilon, x := 0 \]
Some extensions of the model

Adding constraints of the form \( x + y \sim c \)

\[ x + y \sim c \quad \text{and} \quad x \sim c \]  
[\text{Bérard, Dufourd 2000}]

- **Decidability**: for two clocks, \textit{decidable} using the abstraction

\[ 0 \leq x \leq 2 \]

- for four clocks (or more), \textit{undecidable}!

- **Expressiveness**: \textit{more expressive}! (even using two clocks)

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

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

Definition

A two-counter machine is a finite set of instructions over two counters ($x$ and $y$):

- **Incrementation**:
  
  $$(p): \quad x := x + 1; \text{ goto } (q)$$

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

Theorem [Minsky 67]

The halting problem for two counter machines is undecidable.
Some extensions of the model

Undecidability proof

We will use 4 clocks:
- \( u \), “tic” clock (each time unit)
- \( x_0, x_1, x_2 \): reference clocks for the two counters

\[
\text{“} x_i \text{ reference for } c \text{” } \equiv \text{ “the last time } x_i \text{ has been reset is the last time action } c \text{ has been performed”}
\]

[Bérard, Dufourd 2000]
Some extensions of the model

Undecidability proof (cont.)

- Incrementation of counter $c$:

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

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

  ref for $c$ is $x_0$

- Decrementation of counter $c$:

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

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

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

  ref for $c$ is $x_2$
Some extensions of the model

Adding constraints of the form $x + y \sim c$

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

  ![Diagram](chart.png)

- **Four clocks (or more)**: undecidable!
Some extensions of the model

Adding constraints of the form $x + y \sim c$

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

- **Three clocks**: open question
  
  We only know that the coarsest time-abstract bisimulation respecting these constraints is infinite.  
  
  [Robin 2004]

- **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)
Some extensions of the model

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** \( x \)

  \[
  z = 0 \quad z = 1, \ z := 0 \quad z = 0, \ y := y - 1
  \]

- **Decrementation of counter** \( x \)

  \[
  z = 0 \quad x \geq 1 \quad z = 0, \ x := x - 1
  \]
Some extensions of the model

Decidability

The classical region automaton construction is not correct.

→ the bisimulation property is not met
Some extensions of the model

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.)

\[ y := 0 \quad y := 1 \quad x - y < 1 \]

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

implies

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

The bisimulation property is met.
Some extensions of the model

What’s wrong when undecidable?

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

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

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

\[ \max_x \leq \max_x - 1 \]
Some extensions of the model

What’s wrong when undecidable?

Decrementation $x := x - 1$

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

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

\[
\max_x \leq \max_x - 1
\]
Some extensions of the model

What’s wrong when undecidable?

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

$max_x \leq max_x - 1$
Decrementation $x := x - 1$

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

Decrementation \( x := x - 1 \)

\[ \max_x \leq \max_x - 1 \]

etc...
Decidability (cont.)

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

[Bouyer, Dufourd, Fleury, Petit 2000]
Some extensions of the model

### Other extensions which have been considered

- **New operations on clocks**
  
  \[ x := y + c, \ x :< c, \ x :> c, \text{ etc...} \]

  [Bouyer, Dufourd, Fleury, Petit 2004]

- **Alternation**
  
  [Lasota, Walukiewicz 2005]  
  [Ouaknine, Worrell 2005]

  - One-clock alternating timed automata are decidable.
  - *n*-clock alternating timed automata are undecidable \((n \geq 2)\).

- **Slopes of variables:** "Linear hybrid automata"

  [Henzinger 1996]  
  [Henzinger, Kopke, Puri, Varaiya 98]

  - Almost everything is undecidable.
  - The class of LHA with clocks and only one variable having possibly two slopes \(k_1 \neq k_2\) is undecidable.
  - The class of *stopwatch* automata is undecidable.
  - One of the “largest” classes of LHA which are decidable is the class of initialized rectangular automata
Outline

1. About time semantics
2. Timed automata, decidability issues
3. Some extensions of the model
4. Implementation of timed automata
5. Conclusion
The region automaton is not used for implementation:

- suffers from a combinatorics explosion
  (the number of regions is exponential in the number of clocks)
- no really adapted data structure
Notice

The region automaton is not used for implementation:

- suffers from a combinatorics explosion
  (the number of regions is exponential in the number of clocks)
- no really adapted data structure

Algorithms for “minimizing” the region automaton have been proposed...

[Alur & Co 1992] [Tripakis, Yovine 2001]
Notice

The region automaton is not used for implementation:

- suffers from a combinatorics explosion
  (the number of regions is exponential in the number of clocks)
- no really adapted data structure

Algorithms for “minimizing” the region automaton have been proposed...

[Alur & Co 1992] [Tripakis, Yovine 2001]

...but on-the-fly technics are prefered.
Reachability analysis

- **forward analysis algorithm:**
  compute the successors of initial configurations
Forward analysis algorithm:
compute the successors of initial configurations
Reachability analysis

- **forward analysis algorithm:**
  compute the successors of initial configurations

- **backward analysis algorithm:**
  compute the predecessors of final configurations
Reachability analysis

- **forward analysis algorithm:** compute the successors of initial configurations

- **backward analysis algorithm:** compute the predecessors of final configurations
Note on the backward analysis of TA

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

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

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

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

\[ Z \]

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

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

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

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

\[ Z \]

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

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

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

\[ Z \]

\[ Z \]

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

\[ [C \leftarrow 0]^{-1}(Z \cap (C = 0)) \cap g \]
The exact backward computation terminates and 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”
If $A$ 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:

- $v \in \hat{R}$ (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 $v' \in \hat{R}$. 
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 analysis of timed automata

zones

\[ Z \]

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

A **zone** is a set of valuations defined by a clock constraint

\[ \varphi ::= x \sim c \mid x - y \sim c \mid \varphi \land \varphi \]
Implementation of timed automata

Forward analysis of timed automata

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

zones \( Z \)

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

\( \ell \quad \rightarrow \quad \ell' \)
Forward analysis of timed automata

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

zones \[ Z \]

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

$\ell, a, C := 0$

zones $Z$

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

$Z$

$\bar{Z}$

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

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

zones \( Z \)

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

\( Z \) \hspace{2cm} \( \overrightarrow{Z} \) \hspace{2cm} \( \overrightarrow{Z} \cap g \) \hspace{2cm} \( [y \leftarrow 0](\overrightarrow{Z} \cap g) \)
Forward analysis of timed automata

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

zones

\[ Z \]

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

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

\[ \rightarrow \text{a termination problem} \]
Non termination of the forward analysis

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

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

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

\[\begin{align*}
y &:= 0, \\
x &:= 0 \\
x &\geq 1 \land y = 1, \\
y &:= 0
\end{align*}\]

\(\Rightarrow\) an infinite number of steps...
The DBM data structure

DBM (Difference Bound Matrice) data structure

[Berthomieu, Menasche 1983] [Dill 1989]

\((x_1 \geq 3) \land (x_2 \leq 5) \land (x_1 - x_2 \leq 4)\)

\[
\begin{pmatrix}
  x_0 & x_1 & x_2 \\
  +\infty & -3 & +\infty \\
  +\infty & +\infty & 4 \\
  5 & +\infty & +\infty 
\end{pmatrix}
\]
The DBM data structure

DBM (Difference Bound Matrice) data structure

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

\[
\begin{pmatrix}
  x_0 & x_1 & x_2 \\
  +\infty & -3 & +\infty \\
  +\infty & +\infty & 4 \\
  5 & +\infty & +\infty \\
\end{pmatrix}
\]

- Existence of a normal form

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

**DBM (Difference Bound Matrice) data structure**

[Berthomieu, Menasche 1983] [Dill 1989]

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

- Existence of a normal form

- All previous operations on zones can be computed using DBMs
The extrapolation operator

Fix an integer $k$ ($^*$ represents an integer between $-k$ and $+k$)

\[
\begin{pmatrix}
^* & > k & ^* \\
^* & ^* & ^* \\
< -k & ^* & ^*
\end{pmatrix}
\sim
\begin{pmatrix}
^* & +\infty & ^* \\
^* & ^* & ^* \\
- k & ^* & ^*
\end{pmatrix}
\]

- "intuitively", erase non-relevant constraints

⇒ ensures termination
The extrapolation operator

Fix an integer \( k \)

\[
\begin{pmatrix}
* & > k & * \\
* & * & * \\
< -k & * & *
\end{pmatrix}
\sim
\begin{pmatrix}
* & +\infty & * \\
* & * & * \\
- k & * & *
\end{pmatrix}
\]

“intuitively”, erase non-relevant constraints

\[\Rightarrow \text{ensures termination}\]
The extrapolation operator

Fix an integer $k$

\[
\begin{pmatrix}
* & \textcolor{red}{> k} & * \\
* & * & * \\
\textcolor{green}{< -k} & * & *
\end{pmatrix}
\sim
\begin{pmatrix}
* & \textcolor{red}{\infty} & * \\
* & * & * \\
-\textcolor{green}{k} & * & *
\end{pmatrix}
\]

“intuitively”, erase non-relevant constraints

→ ensures termination
Classical algorithm, focus on correctness

Take $k$ the maximal constant appearing in the constraints of the automaton.
Implementation of timed automata

Classical algorithm, focus on correctness

Take $k$ the maximal constant appearing in the constraints of the automaton.

**Theorem**

This algorithm is correct for diagonal-free timed automata.
Implementation of timed automata

Classical algorithm, focus on correctness

Take $k$ the maximal constant appearing in the constraints of the automaton.

**Theorem**

This algorithm is correct for diagonal-free timed automata.

**However**, this theorem does not extend to timed automata using diagonal clock constraints...

- Implemented in numerous tools:
  - Kronos, http://www-verimag.imag.fr/TEMPORISE/kronos/
  - ...

- Successfully used on many real-life examples since ten years.
Outline

1. About time semantics
2. Timed automata, decidability issues
3. Some extensions of the model
4. Implementation of timed automata

5. Conclusion
Decidability is quite well understood.

There is still some progress which is done for the verification of timed automata. (see Gerd’s talk)

Some other current challenges:
- controller synthesis
- implementability issues (program synthesis) (remember Jean-François’ talk)
- optimal computations
- ...
A problematic automaton

\[ x_3 \leq 3, \quad x_1, x_3 := 0 \]

\[ x_2 = 3, \quad x_2 := 0 \]

\[ x_1 = 2, \quad x_1 := 0 \]

\[ x_2 = 2, \quad x_2 := 0 \]

\[ x_1 = 3, \quad x_2 = 2 \]

\[ x_1 := 0 \]

\[ x_2 := 0 \]

Error

\[ x_2 - x_1 > 2 \]

\[ x_4 - x_3 < 2 \]

The loop
A problematic automaton

\[
\begin{align*}
 x_1, x_3 &:= 0 \\
 x_2 &:= 0
\end{align*}
\]

\[
\begin{align*}
 x_1 &= 2, \quad x_1 := 0 \\
 x_2 &= 2, \quad x_2 := 0
\end{align*}
\]

\[
\begin{align*}
 x_1 &= 3 \\
 x_2 &= 2
\end{align*}
\]

\[
\begin{align*}
 x_3 &\leq 3 \\
 x_2 &= 3
\end{align*}
\]

\[
\begin{align*}
 x_1 &= 2 \\
 x_1 &:= 0
\end{align*}
\]

\[
\begin{align*}
 x_2 &= 2, \quad x_2 := 0
\end{align*}
\]

\[
\begin{align*}
 x_1 &= 0 \\
 x_2 &= 0
\end{align*}
\]

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

Error

The loop
A problematic automaton

\[ \begin{align*}
  x_3 & \leq 3, \\
  x_1, x_3 & := 0, \\
  x_2 & = 3, \\
  x_2 & := 0, \\
  x_1 & = 2, \\
  x_1 & := 0, \\
  x_2 & = 2, \\
  x_2 & := 0, \\
  x_1 & = 2, \\
  x_1 & := 0, \\
  x_2 & = 2, \\
  x_2 & := 0.
\end{align*} \]

\[ \begin{align*}
  x_2 - x_1 & > 2, \\
  x_4 - x_3 & < 2.
\end{align*} \]

Error

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

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

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

implies $x_1 - x_2 = x_3 - x_4$.

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