Real-Time and Hybrid Systems

Patricia Bouyer

LSV – CNRS & ENS de Cachan

Winter School MOVEP’04
Model-checking

Does the system satisfy the property?

Modelling

ϕ
Model-checking

Does the system satisfy the property?

Modelling

Model-checking Algorithm

MOVEP’04

Real-Time and Hybrid Systems 2 / 79
**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 & bibliography
Adding timing informations

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

\[
\begin{aligned}
  a \ b \ a \ b \ a \ b \ a \ b \ a \ b \ \cdots &= (a \ b)^\omega
\end{aligned}
\]
About time semantics

Adding timing informations

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

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

- **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
  - \( a \): send message
  - \( b \): receive message
  
  \[
  a \; b \; a \; b \; a \; b \; a \; b \; \cdots = (a \; b)^\omega
  \]

- **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**: \( (a, 1)(b, 3)(c, 4)(a, 6) \)
About time semantics

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

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

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

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

  - **Discrete-time semantics**: dates are e.g. taken in \( N \)
    
    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. $\mathbb{N}$) or dense (e.g. $\mathbb{Q}^+$)
- 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

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

[Brzozowski, Seger 1991]

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 \( y=[011] \)
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.

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

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

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

This digital circuit is not 1-discretizable.
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)
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)

$$
[11100000] \xrightarrow{y_1} [01100000] \xrightarrow{y_2} [00100000] \xrightarrow{y_3, y_5} [00001000] \xrightarrow{y_5, y_7} [00000010] \xrightarrow{y_7, y_8} [00000001]
$$
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)

$$
\begin{align*}
[11100000] & \xrightarrow{y_1} [01100000] \xrightarrow{y_2} [00110000] \xrightarrow{y_3, y_5} [00001000] \xrightarrow{y_5, y_7} [00000010] \xrightarrow{y_7, y_8} [00000001] \\
[11100000] & \xrightarrow{y_1, y_2, y_3} [00000000]
\end{align*}
$$
Is discretizing sufficient? An example [Alur 91]

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

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

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

This digital circuit is not 1-discretizable.

Why that?

[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] $\xrightarrow{y_1} [01110000]$ $\xrightarrow{y_2,y_3,y_4,y_5} [00000000]$

[11100000] $\xrightarrow{y_1,y_2} [00100000]$ $\xrightarrow{y_3,y_5,y_6} [00001100]$ $\xrightarrow{y_5,y_6} [00000000]$
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)

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

[Brzozowski Seger 1991]

**Theorem:** 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?

[Brzozowski Seger 1991]

**Theorem:** 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?

[Brzozowski Seger 1991]

**Theorem:** 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)
Outline

1. About time semantics
2. **Timed automata, decidability issues**
3. Some extensions of the model
4. Implementation of timed automata
5. Conclusion & bibliography
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 : $ 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)

$x, y$ : clocks

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

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

$\ell_0 \xrightarrow{\delta(4.1)} \ell_0 \xrightarrow{a} \ell_1 \xrightarrow{\delta(1.4)} \ell_1 \xrightarrow{b} \ell_2$

$(\text{clock}) \text{ valuation}$
Timed automata (example)

$x, y : \text{clocks}$

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

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

$\ell_0 \overset{\delta(4.1)}{\rightarrow} \ell_0 \overset{a}{\rightarrow} \ell_1 \overset{\delta(1.4)}{\rightarrow} \ell_1 \overset{b}{\rightarrow} \ell_2$

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

$\begin{array}{c|c|c}
\ell_1 & \ell_1 & \ell_2 \\
\hline
x & 5.5 & 0 \\
y & 1.4 & 1.4 \\
\end{array}$

(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

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

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

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

\[ y < 1, \quad b, \quad y := 0 \]
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 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:** \[ L_{\text{discrete}} = \emptyset \]
Discrete vs dense-time semantics

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

- **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:

$$AG(problem \Rightarrow AF \, alarm)$$
Adding time to temporal logics

Classical temporal logics allow us to express that

“any problem is followed by an alarm”

With CTL:

\[ AG(problem \Rightarrow AF\ alarm) \]

How can we express:

“any problem is followed by an alarm 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:

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

How can we express:

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

- Temporal logics with **subscripts**.

  ex: \( CTL + \begin{array}{c}
  E \varphi \mathbin{U}_{\sim k} \psi \\
  A \varphi \mathbin{U}_{\sim k} \psi
  \end{array} \)
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 in at most 20 time units”

- Temporal logics with **subscripts**.

\[ AG(\text{problem} \Rightarrow AF_{\leq 20} \text{ alarm}) \]
Adding time to temporal logics

Classical temporal logics allow us to express that

“any problem is followed by an alarm”

With CTL:

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

How can we express:

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

- Temporal logics with **subscripts**.

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

- Temporal logics with **clocks**.

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

Classical temporal logics allow us to express that

“any problem is followed by an alarm”

With CTL:

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

How can we express:

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

- Temporal logics with *subscripts*.

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

- Temporal logics with *clocks*.

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

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

\( \text{Train}_i \) with \( i = 1, 2, \ldots \)
The train crossing example (2)

The gate:

- **Open**
  - **GoDown?**, $H_g := 0$
  - $H_g < 10$, $a$

- **Lowering**, $H_g < 10$
  - $H_g < 10$, $a$

- **Raising**, $H_g < 10$
  - **GoUp?**, $H_g := 0$

- **Close**
  - $H_g < 10$, $a$
The train crossing example (3)

The controller:

\[
\begin{align*}
&c_1, x_c \leq 20 \\
&c_0 \\
&c_2, x_c \leq 10
\end{align*}
\]

- **Exit?**
  - \( c_1, x_c \leq 20 \)
  - \( H_c := 0 \)
  - \( H_c = 20, \text{ GoUp!} \)

- **App?**
  - \( c_0 \)
  - \( H_c := 0 \)
  - \( H_c \leq 10, \text{ GoDown!} \)

- **Exit?**
  - \( c_2, x_c \leq 10 \)
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>$Exit!$</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>$GoUp?$</td>
<td>$GoUp!$</td>
<td>$GoUp$</td>
</tr>
<tr>
<td>.</td>
<td>$GoDown?$</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

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?

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

Some properties one could check:
- Is the gate closed when a train crosses the road?
  \[ AG(train.On \Rightarrow gate.Close) \]
- Is the gate always closed for less than 5 minutes?
Some properties one could check:

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

- Is the gate always closed for less than 5 minutes?
  \[ \neg EF(gate.Close \land (gate.Close \mathbin{U}_{>5\text{ min}} \neg gate.Close)) \]
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
  ➔ 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 can not 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:** The emptiness problem for timed automata is decidable.
It is PSPACE-complete.  
[Alur & Dill 1990’s]
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:** The emptiness problem for timed automata is decidable.
It is PSPACE-complete.  
[Alur & Dill 1990’s]

**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:** The emptiness problem for timed automata is decidable.
It is PSPACE-complete.  
[Alur & Dill 1990’s]

**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

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
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
\( I_x = ]1; 2[ \), \( I_y = ]0; 1[ \)
\( \{x\} < \{y\} \)

- “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
Time-abstract bisimulation
Time-abstract bisimulation
Time-abstract bisimulation

∀a

∃a

∀d > 0

δ(d)
Time-abstract bisimulation

∀ \ a \rightarrow \exists \ a

∀d > 0 \Rightarrow \delta(d)

∃d' > 0 \Rightarrow \delta(d')
Time-abstract bisimulation

∀ \quad \exists

∀d > 0 \quad \exists d' > 0

(\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**

∀   ![Diagram](image)

∃   ![Diagram](image)

∀   ![Diagram](image)

∃   ![Diagram](image)

(ℓ₀, v₀) \xrightarrow{a₁,t₁} (ℓ₁, v₁) \xrightarrow{a₂,t₂} (ℓ₂, v₂) \xrightarrow{a₃,t₃} \ldots

(ℓ₀, R₀) \xrightarrow{a₁} (ℓ₁, R₁) \xrightarrow{a₂} (ℓ₂, R₂) \xrightarrow{a₃} \ldots

with vᵢ ∈ Rᵢ for all i.
Time-abstract bisimulation

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

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

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

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

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

\[
(\ell_0, v_0) \xrightarrow{a_1,t_1} (\ell_1, v_1) \xrightarrow{a_2,t_2} (\ell_2, v_2) \xrightarrow{a_3,t_3} \ldots \\
(\ell_0, R_0) \xrightarrow{a_1} (\ell_1, R_1) \xrightarrow{a_2} (\ell_2, R_2) \xrightarrow{a_3} \ldots
\]

with \( v_i \in R_i \) for all \( i \).

**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.
The region automaton

timed automaton $\boxtimes$ region abstraction

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

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

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

$\Rightarrow$ time-abstract bisimulation

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

where $\text{UNTIME}((a_1, t_1)(a_2, t_2)\ldots) = a_1 a_2 \ldots$
An example [AD 90’s]
The size of the region graph is in $O(|X|! \cdot 2^{|X|})$!

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

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

- **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
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
  $\Rightarrow$ in NPSPACE, thus in PSPACE
PSPACE-hardness

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

\[ \begin{array}{c}
\text{w}_0 \\
C_j \\
\{x_j, y_j\}
\end{array} \]

\( C_j \) 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)

\[ \rightarrow \text{ proof taken in [Aceto & Laroussinie 2002]} \]
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:** 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
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** [Alur & Dill 90’s]
- Inclusion is **undecidable** [Alur & Dill 90’s]
- Determinizability is **undecidable** [Tripakis 2003]
- Complementability is **undecidable** [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.:

```
\begin{align*}
\text{a, } x &:= 0 \\
\text{a, } x &:= 1, \text{ a}
\end{align*}
```
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.:

\[ [\text{Alur, Madhusudan 2004}] \]

\[ a, b, x \] \quad \xrightarrow{a} x := 0 \quad \xrightarrow{a, b} x \neq 1, a, b

\[ \text{UNTIME}\left( \overline{L} \cap \{(a^* b^*, \tau) \mid \text{all } a's \text{ happen before 1 and no two } a's \text{ simultaneously}\} \right) \text{ is not regular (exercise!)} \]
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 & bibliography
Role of diagonal constraints

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

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

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

Role of diagonal constraints (cont.)

$c$ is positive

$x := 0 \quad y := 0$

$x - y \leq c$

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

Copy where $x - y \leq c$

$x := 0$

$y := 0$

$x \leq c$

Copy where $x - y > c$

$x := 0$

$y := 0$

$x > c$

$y := 0$
Some extensions of the model

Role of diagonal constraints (cont.)

Open question: is this construction “optimal”? In the sense that timed automata with diagonal constraints are exponentially more concise than diagonal-free timed automata.
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!

\[ x = 1 \]
\[ x := 0 \]
\[ 0 < x < 1, \ b \]
\[ x = 1, \ \varepsilon, \ x := 0 \]

\[ a \]
\[ a \]
\[ a \]

\[ a \]
\[ a \]
\[ b \]
\[ b \]
\[ a \]

0 1 2 3 4
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 $x + y \sim c$

$\begin{align*}
  x + y &\sim c \\
  x &\sim c
\end{align*}$

[Bérard, Dufourd 2000]

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

  $\begin{array}{c}
  x + y = 1, \quad a, \quad x := 0 \\
  \{ (a^n, t_1 \ldots t_n) \mid n \geq 1 \text{ and } t_i = 1 - \frac{1}{2^i} \}
  \end{array}$

- for four clocks (or more), undecidable!

- **Expressiveness:** more expressive! (even using two clocks)
The two-counter machine

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

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

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

**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

“$x_i$ reference for $c$” \equiv “the last time $x_i$ has been reset is the last time action $c$ has been performed”

[Bérard, Dufourd 2000]
Some extensions of the model

Undecidability proof (cont.)

**Incrementation of counter $c$:**

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

\[ x_2 := 0 \]

\[ u = 1, \quad \star, \quad u := 0 \]

\[ x_0 > 2, \quad c, \quad x_2 := 0 \]

\[ u + x_2 = 1 \]

ref for $c$ is $x_0$

**Decrementation of counter $c$:**

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

\[ x_2 := 0 \]

\[ u = 1, \quad \star, \quad u := 0 \]

\[ x_0 = 2, \quad c, \quad x_2 := 0 \]

\[ u + x_2 = 1 \]

\[ u = 1, \quad x_0 = 2, \quad \star, \quad u := 0, \quad x_2 := 0 \]
Some extensions of the model

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

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

- **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

- **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 $x$**

- $z = 0$
- $z = 1$, $z := 0$
- $z = 0$, $y := y - 1$

**Decrementation of counter $x$**

- $z = 0$
- $x \geq 1$
- $z = 0$, $x := x - 1$
- $x = 0$
Some extensions of the model

Decidability

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

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 \( x :\sim \ y + c \)
  \( \max_x \leq \max_{y,z} + c, \max_{z,x} \geq \max_{z,y} - c \)
- update \( x :< c \)
  \( c \leq \max_x \)
  \( \max_x \geq c + \max_{z,x} \)

The constants \( (\max_x) \) and \( (\max_{x,y}) \) define a set of regions.
Some extensions of the model

Decidability (cont.)

\[
\begin{align*}
\text{max}_y & \geq 0 \\
\text{max}_x & \geq 0 + \text{max}_{x,y} \\
\text{max}_y & \geq 1 \\
\text{max}_x & \geq 1 + \text{max}_{x,y} \\
\text{max}_{x,y} & \geq 1
\end{align*}
\]

implies

\[
\begin{align*}
\text{max}_x & = 2 \\
\text{max}_y & = 1 \\
\text{max}_{x,y} & = 1 \\
\text{max}_{y,x} & = -1
\end{align*}
\]

The bisimulation property is met.
Some extensions of the model

What’s wrong when undecidable?

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

$$\max_x \leq \max_x - 1$$
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** \( x := x - 1 \)

\[
\max_x \leq \max_x - 1
\]
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** \( x := x - 1 \)

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

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>Undecidable</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]
Linear hybrid automata

A finite control structure + a set $X$ of *dynamic variables*

A transition is of the form:

$$\text{Act}_{\ell} \xrightarrow{g, a, \alpha} \text{Act}_{\ell'}$$

- $g$ is a linear constraint on variables
- $\alpha$ is a jump condition, i.e. an affine update of the form $X' = A.X + B$
- in each state, an activity function assigning a slope to each variable (for each $x \in X$, $\text{Act}(x) \in [\ell, u]$)
Some extensions of the model

LHA semantics

- $H = (\Sigma, L, X, Act)$ is a LHA

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

- **Timed Transition System:**
  - **action transition:** $(\ell, v) \xrightarrow{a} (\ell', v')$ if $\exists \ell \xrightarrow{g,a,J} \ell' \in A$ s.t.
    \[
    \begin{align*}
    v &\models g \\
    v' &\models \alpha(v)
    \end{align*}
    \]
  - **delay transition:** $(\ell, v) \xrightarrow{\delta(d)} (\ell, v + d Act(\ell))$ if $d \in T$
The gas burner may leak.

- each time a leakage is detected, it is repaired or stopped in less than 1s
- two leakages are separated by at least 30s

Is it possible that the gas burner leaks during a time greater than \( \frac{1}{20} \) of the global time after the 60 first minutes?

\[
AG (y \geq 60 \implies 20t \leq y)
\]
Some extensions of the model

What about decidability?

→ almost everything is undecidable

[Henzinger, Kopke, Puri, Varaiya 98]

**Theorem.** The class of LHA with clocks and only one variable having possibly two slopes \( k_1 \neq k_2 \) is undecidable.

**Theorem.** 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 & bibliography
Implementation of timed automata

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
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
Reachability analysis

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

![Diagram of reachability analysis](image-url)
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
Symbolic representation, symbolic computation

- Need of a symbolic representation:
  Finite representation of infinite sets of configurations
Symbolic representation, symbolic computation

Need of a symbolic representation:

Finite representation of infinite sets of configurations

in the plane, a line
represented by two points.
Symbolic representation, symbolic computation

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 $aa(aa)^*$
Symbolic representation, symbolic computation

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
Symbolic representation, symbolic computation

- 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
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...
Implementation of timed automata

Symbolic representation, symbolic computation

- 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...
Need of a symbolic representation:

- 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...

Examples of systems: counter automata, pushdown systems, linear hybrid automata, timed automata, 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
Note on the backward analysis of TA

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

\[ \ell \xrightarrow{g, a, C := 0} \ell' \]
Note on the backward analysis of TA

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

\[ \ell, a, C := 0 \quad \ell' \]

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

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

\[ C \leftarrow 0 \]

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

\[ Z \]

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

\[ Z \]

\[ C := 0 \]
Note on the backward analysis of TA

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

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

\[ Z \]

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

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

The exact backward computation terminates and is correct!
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:

- $v \in \overleftarrow{R}$ (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 $v' \in \overleftarrow{R}$. 
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”

**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

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

zones \( Z \)

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

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

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

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

zones \( Z \)

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

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

zones \( Z \)

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

\[ Z \]

\[ \overline{Z} \]
Forward analysis of timed automata

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

zones \[ Z \]

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

\[ Z \]

\[ \overrightarrow{Z} \]

\[ \overrightarrow{Z} \cap g \]
Implementation of timed automata

Forward analysis of timed automata

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

zones

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

\[ Z \]

\[ \overrightarrow{Z} \]

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

\[ [y \leftarrow 0]\left(\overrightarrow{Z} \cap g\right) \]
Implementation of timed automata

Forward analysis of timed automata

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

zones \( Z \)

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

\( \rightarrow \) 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

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

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

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

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

→ 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 $Z$

  ➔ 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 \( Z \)

  \( \Rightarrow \) correct w.r.t. reachability

- **activity**: eliminate redundant clocks

  \( \Rightarrow \) correct w.r.t. reachability

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

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

→ “semi-correct” w.r.t. reachability
convex-hull approximation: if $Z$ and $Z'$ are computed then we overapproximate using \("Z \sqcup Z'\).

\(\Rightarrow\) “semi-correct” w.r.t. reachability

extrapolation, a widening operator on zones
### The DBM data structure

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

[Berthomieu, Menasche 1983] [Dill 1989]

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

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

DBM (Difference Bounded Matrice) data structure

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

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

- Existence of a normal form

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

DBM (Difference Bounded 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

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

Fix an integer $k$ ("\(\ast\)" represents an integer between $-k$ and $+k$)

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

• "intuitively", erase non-relevant constraints

→ ensures termination
The extrapolation operator

Fix an integer $k$  


greater than $k$ \(\sim\) \(\sim\)

intuitively”, erase non-relevant constraints

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

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

Fix an integer $k$ ($\ast$ represents an integer between $-k$ and $+k$)

\[
\begin{pmatrix}
\ast & \ast & \ast & > k \\
\ast & \ast & \ast & \ast \\
< -k & \ast & \ast & \ast
\end{pmatrix}
\sim
\begin{pmatrix}
\ast & \ast & \ast & \ast \\
\ast & \ast & \ast & \ast \\
- k & \ast & \ast & \ast
\end{pmatrix}
\]

"intuitively", erase non-relevant constraints

→ ensures termination
Challenge: choose a good constant for the extrapolation so that the forward computation is correct.
Challenge: choose a good constant for the extrapolation so that the forward computation is correct.

- 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.

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

**Theorem:** this algorithm is correct for diagonal-free timed automata.
Challenge: choose a good constant for the extrapolation so that the forward computation is correct.

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

Theorem: this algorithm is correct for diagonal-free timed automata.

However, this theorem does not extend to timed automata using diagonal clock constraints...
A problematic automaton

\[ x_3 \leq 3, \quad x_1, x_3 := 0 \quad x_2 = 3, \quad x_2 := 0 \]

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

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

Error

\[ x_2 - x_1 > 2 \]

\[ x_4 - x_3 < 2 \]

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

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

The loop
A problematic automaton

\[
\begin{align*}
x_3 &\leq 3 \\
x_1, x_3 &:= 0 \\
x_1, x_3 &:= 0 \\
x_2 &:= 3 \\
x_2 &:= 0 \\
x_2 &:= 0 \\
x_2 &:= 0 \\
x_1 &:= 2 \\
x_1 &:= 0 \\
x_2 &:= 2, x_2 := 0 \\
x_1 &:= 3 \\
x_1 &:= 0 \\
x_2 &:= 2 \\
x_2 &:= 0 \\
x_4 &- x_3 < 2 \\
x_2 - x_1 > 2 \\
\end{align*}
\]

\[
\begin{cases}
v(x_1) = 0 \\
v(x_2) = d \\
v(x_3) = 2\alpha + 5 \\
v(x_4) = 2\alpha + 5 + d
\end{cases}
\]
A problematic automaton

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

Error

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

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

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

\[ [1; 3] \quad [2\alpha + 5] \quad [1; 3] \]

\[ x_1 \quad x_2 \quad x_3 \quad x_4 \]

implies

\[ x_1 - x_2 = x_3 - x_4. \]

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

\[ [1; 3] \]

\[ x_1 \quad x_2 \quad x_3 \quad x_4 \]

does not imply \( x_1 - x_2 = x_3 - x_4. \)
General abstractions

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

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

- easy computation

\textit{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
  \[ \text{[Effectiveness]} \]

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

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

[Effectiveness] 
[Termination] 
[Completeness] 
[Soundness]
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

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.

$\Rightarrow$ $k$ the max. constant defining one of the previous DBMs

We get that, for every zone $Z$,

$$Z \subseteq \text{Extra}_k(Z) \subseteq \text{Abs}(Z)$$
Open questions:  
- which conditions can be made weaker?  
- find a clever termination criterium?  
- use an other data structure than zones/DBMs?
Outline

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

5. Conclusion & bibliography
Discussion on complexity


<table>
<thead>
<tr>
<th>Kripke structures S</th>
<th>Timed automaton A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachability</td>
<td>NLOGSPACE-complete</td>
</tr>
<tr>
<td>CTL/TCTL</td>
<td>P-complete</td>
</tr>
<tr>
<td>AF-$\mu$-calc./$L_{\mu,\nu}$</td>
<td>P-complete</td>
</tr>
<tr>
<td>full $\mu$-calc./$L^{+}_{\mu,\nu}$</td>
<td>NP $\cap$ co-NP</td>
</tr>
</tbody>
</table>
## Discussion on complexity


<table>
<thead>
<tr>
<th></th>
<th>Kripke structures $S$</th>
<th>Timed automaton $A$ or $(S_1 \parallel \ldots \parallel S_n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachability</td>
<td>NLOGSPACE-complete</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>CTL/TCTL</td>
<td>P-complete</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>AF-$\mu$-calc./$L_{\mu,\nu}$</td>
<td>P-complete</td>
<td>EXPTIME-complete</td>
</tr>
<tr>
<td>full $\mu$-calc./$L_{\mu,\nu}^+$</td>
<td>NP $\cap$ co-NP</td>
<td>EXPTIME-complete</td>
</tr>
</tbody>
</table>

Timing constraints induce a complexity blowup!
## Discussion on complexity


<table>
<thead>
<tr>
<th></th>
<th>Kripke structures $S$</th>
<th>Timed automaton $A$ or $(S_1 \parallel \ldots \parallel S_n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachability</td>
<td>NLOGSPACE-complete</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>CTL/TCTL</td>
<td>P-complete</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>$\text{AF-}\mu\text{-calc.}/L_{\mu,\nu}$</td>
<td>P-complete</td>
<td>EXPTIME-complete</td>
</tr>
<tr>
<td>full $\mu\text{-calc.}/L^+_{\mu,\nu}$</td>
<td>NP $\cap$ co-NP</td>
<td>EXPTIME-complete</td>
</tr>
</tbody>
</table>

Timing constraints induce a complexity blowup!

From a complexity point of view, adding clocks = adding components!
## Discussion on complexity


| Kripke structures $S$ | Timed automaton $A$
|----------------------|----------------------|
| Reachability | NLOGSPACE-complete or $(S_1 \parallel \ldots \parallel S_n)$ or $(A_1 \parallel \ldots \parallel A_n)$
| CTL/TCTL | P-complete or PSPACE-complete
| AF-$\mu$-calc.$/L_{\mu,\nu}^-$ | P-complete or EXPTIME-complete
| full $\mu$-calc.$/L_{\mu,\nu}^+$ | NP $\cap$ co-NP or EXPTIME-complete

Timing constraints induce a complexity blowup!

From a complexity point of view, adding clocks = adding components!
State explosion problem

- due to parallel composition
- due to timing constraints
State explosion problem

- due to parallel composition
- due to timing constraints

From a complexity point of view:

no double complexity gap!
State explosion problem

- due to parallel composition
- due to timing constraints

From a complexity point of view:

no double complexity gap!
State explosion problem

- due to parallel composition
- due to timing constraints

From a complexity point of view:

no double complexity gap!

In practice:

- BDD-like techniques try to avoid discrete state explosion problem in untimed systems
  ➔ SMV verifies very large systems
State explosion problem

- due to parallel composition
- due to timing constraints

From a complexity point of view:

no double complexity gap!

In practice:

- BDD-like techniques try to avoid discrete state explosion problem in untimed systems ➔ SMV verifies very large systems

- **Timed systems**: problems to deal with both explosions. Much smaller systems can be analyzed in practice.
State explosion problem

- due to parallel composition
- due to timing constraints

From a complexity point of view:

no double complexity gap!

In practice:

- BDD-like techniques try to avoid discrete state explosion problem in untimed systems
  ➜ SMV verifies very large systems

- **Timed systems:** problems to deal with both explosions. Much smaller systems can be analyzed in practice.
State explosion problem

- due to parallel composition
- due to timing constraints

From a complexity point of view:

no double complexity gap!

In practice:

- BDD-like techniques try to avoid discrete state explosion problem in untimed systems  ➔ SMV verifies very large systems
- **Timed systems:** problems to deal with both explosions. Much smaller systems can be analyzed in practice.

**Tools for timed systems:** Uppaal, HyTech, Kronos, etc...
Conclusion & Further Work

- Decidability is quite well understood.

- Needs to understand better the **geometry** of the reachable state space.
  - clever (and correct) implementation of timed automata
  - accelerate verification of timed automata

- Data structures for both **dense** and **discrete** parts
Conclusion & Further Work

- Decidability is quite well understood.

- Needs to understand better the *geometry* of the reachable state space.
  - clever (and correct) implementation of timed automata
  - accelerate verification of timed automata

- Data structures for both *dense* and *discrete* parts

*To be continued...*
Conclusion & Further Work

- Decidability is quite well understood.

- Needs to understand better the geometry of the reachable state space.
  - clever (and correct) implementation of timed automata
  - accelerate verification of timed automata

- Data structures for both dense and discrete parts

To be continued...

- Some other current challenges:
  - controller synthesis
  - implementability issues (program synthesis)
  - optimal computations
  - ...

(see Kim’s talk)
Bibliography I


Bibliography II


[BDFP00a] Bouyer, Dufourd, Fleury, Petit. Are Timed Automata Updatable? CAV’00 (LNCS 1855).


Bibliography III


[DY96] Daws, Yovine. **Reducing the Number of Clock Variables of Timed Automata.** RTSS’96.


**Hytech:** [http://www-cad.eecs.berkeley.edu:80/~tah/HyTech/](http://www-cad.eecs.berkeley.edu:80/~tah/HyTech/)

**Kronos:** [http://www-verimag.imag.fr/TEMPORISE/kronos/](http://www-verimag.imag.fr/TEMPORISE/kronos/)

**Uppaal:** [http://www.uppaal.com/](http://www.uppaal.com/)