An introduction to timed automata

Patricia Bouyer-Decitre

LSV, CNRS & ENS Cachan, France
Outline

1 Introduction
   - Timed automata
   - Examples

2 Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata

3 Implementation and tools

4 Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)

5 Timed control
   - Timed games
   - Weighted timed games

6 Conclusion
Time-dependent systems

- We are interested in timed systems
Time-dependent systems

- We are interested in timed systems
Time-dependent systems

- We are interested in **timed systems**

- and in their **correctness**

  “Will the airbag open within 5ms after the car crashes?”
  “Will the robot explore a given area without getting out of energy?”
Model-checking and control

system:

property:
Model-checking and control

system:

property:

\[ AG(\neg B.\text{overfull} \land \neg B.\text{dried\_up}) \]
Model-checking and control

system:

property:

algorithm

\[ AG(\neg B.\text{overfull} \land \neg B.\text{dried up}) \]
Model-checking and control

system:

property:

model-checking algorithm

\text{AG(\neg B.\text{overfull} \land \neg B.\text{dried\_up})}

yes/no
Model-checking and control

**system:**

[Diagram of a system with two tanks and a pump, showing flow between them.

**property:**

[Diagram showing a property indicated by red crosses on the tanks.

**control/synthesis algorithm:**

AG(¬B.overfull ∧ ¬B.dried_up)
Outline

1 Introduction
   • Timed automata
   • Examples

2 Decidability of basic properties
   • The region abstraction
   • Extensions of timed automata
   • Weighted timed automata

3 Implementation and tools

4 Other verification problems
   • Equivalence (or preorder) checking
   • Verification of timed temporal logics (short)

5 Timed control
   • Timed games
   • Weighted timed games

6 Conclusion
Reasoning about real-time systems

A plethora of models

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

A plethora of models

- timed circuits,
- time\(\text{(d)}\) Petri nets,
- **timed automata,**
- timed process algebra,
- ...
Reasoning about real-time systems

The model of timed automata [AD94]

A timed automaton is made of

- a finite automaton-based structure

Example

```
safe → problem, done, repairing
      ^
      |    
      |    
alarm  repair, delayed, failsafe
      ^
      |    
      |    
      repair
```
Reasoning about real-time systems

The model of timed automata [AD94]

A timed automaton is made of
- a finite automaton-based structure
- a set of clocks

Example

\[
\begin{align*}
\text{safe} & \xrightarrow{\text{problem,}} \text{alarm} \\
\text{repairing} & \xrightarrow{\text{repair,}} \text{alarm} \\
& \xrightarrow{\text{delayed,}} \text{failsafe} \\
\text{done,} & \xrightarrow{} \text{repairing} \\
\end{align*}
\]
Reasoning about real-time systems

The model of timed automata [AD94]

A timed automaton is made of
- a finite automaton-based structure
- a set of clocks
- timing constraints and clock resets on transitions

Example

```
<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>safe</td>
<td>problem</td>
<td>x:=0</td>
</tr>
<tr>
<td>alarm</td>
<td>repair</td>
<td>x&lt;15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>y:=0</td>
</tr>
<tr>
<td>repairing</td>
<td>repair</td>
<td>2&lt;=y^x&lt;=56</td>
</tr>
<tr>
<td></td>
<td></td>
<td>y:=0</td>
</tr>
<tr>
<td>done</td>
<td></td>
<td>22&lt;=y&lt;=25</td>
</tr>
<tr>
<td>failsafe</td>
<td>delayed</td>
<td>delayed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>y:=0</td>
</tr>
</tbody>
</table>
```

```
Example – cont’d

This run reads the timed word (problem, 23)(delayed, 38.6)(repair, 40.9)(done, 63).

\[
\text{safe} \xrightarrow{\text{problem, } x:=0} \text{alarm} \xrightarrow{\text{repair, } x\leq 15} \text{repairing} \xrightarrow{\text{repair} \land 2\leq y \land x \leq 56} \text{failsafe} \xrightarrow{\text{done, } 22\leq y \leq 25} \text{safe}
\]
Example – cont’d

This run reads the timed word $(\text{problem}, 23)(\text{delayed}, 38.6)(\text{repair}, 40.9)(\text{done}, 63).$
Example – cont’d

This run reads the timed word

(probлем, 23)(delayed, 38.6)(repair, 40.9)(done, 63).

<table>
<thead>
<tr>
<th></th>
<th>23</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>0</td>
<td>23</td>
</tr>
<tr>
<td>( y )</td>
<td>0</td>
<td>23</td>
</tr>
</tbody>
</table>
Example – cont’d

This run reads the timed word

\((\text{problem}, 23)(\text{delayed}, 38.6)(\text{repair}, 40.9)(\text{done}, 63)\).

<table>
<thead>
<tr>
<th>safe</th>
<th>(\rightarrow)</th>
<th>safe</th>
<th>(\rightarrow)</th>
<th>alarm</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x)</td>
<td>0</td>
<td>23</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(y)</td>
<td>0</td>
<td>23</td>
<td>23</td>
<td></td>
</tr>
</tbody>
</table>
Example – cont’d

\[
\begin{align*}
\text{safe} & \rightarrow \text{problem, } x:=0 \rightarrow \text{alarm} \rightarrow \text{repair, } x \leq 15 \rightarrow \text{done, } 22 \leq y \leq 25 \\
\text{delayed, } y:=0 \rightarrow \text{failsafe} \\
\text{repair, } y:=0 \rightarrow \text{repairing} \rightarrow \text{repair, } 2 \leq y \land x \leq 56 \\
\text{done, } 22 \leq y \leq 25 \rightarrow \text{safe} \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>State</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>safe</td>
<td>0</td>
<td>23</td>
<td>0</td>
<td>15.6</td>
</tr>
<tr>
<td>alarm</td>
<td>23</td>
<td>23</td>
<td>38.6</td>
<td></td>
</tr>
</tbody>
</table>

This run reads the timed word \((\text{problem}, 23)(\text{delayed}, 38.6)(\text{repair}, 40.9)(\text{done}, 63)).\)
Example – cont’d

This run reads the timed word \((\text{problem}, 23)(\text{delayed}, 0.6)(\text{repair}, 40.9)(\text{done}, 63)\).

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>safe</td>
<td>23</td>
<td>safe</td>
<td>problem</td>
<td>alarm</td>
<td>alarm</td>
</tr>
<tr>
<td>x</td>
<td>0</td>
<td>23</td>
<td>0</td>
<td>15.6</td>
<td>15.6</td>
</tr>
<tr>
<td>y</td>
<td>0</td>
<td>23</td>
<td>23</td>
<td>38.6</td>
<td>15.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>failsafe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>15.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example – cont’d

This run reads the timed word

\[(\text{problem}, 23) (\text{delayed}, 38.6) (\text{repair}, 40.9) (\text{done}, 63).\]

<table>
<thead>
<tr>
<th></th>
<th>safe</th>
<th>23</th>
<th>safe</th>
<th>problem</th>
<th>alarm</th>
<th>15.6</th>
<th>alarm</th>
<th>delayed</th>
<th>failsafe</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td>23</td>
<td>0</td>
<td>0</td>
<td>15.6</td>
<td>15.6</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>0</td>
<td>23</td>
<td>23</td>
<td>38.6</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>failsafe</th>
<th>2.3</th>
<th>failsafe</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>15.6</td>
<td>17.9</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>0</td>
<td>2.3</td>
<td></td>
</tr>
</tbody>
</table>
Example – cont’d

This run reads the timed word

\((\text{problem}, 23)(\text{delayed}, 38.6)(\text{repair}, 40.9)(\text{done}, 63)\).

<table>
<thead>
<tr>
<th>State</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>safe</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>problem</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>alarm</td>
<td>0</td>
<td>23</td>
</tr>
<tr>
<td>alarm</td>
<td>15.6</td>
<td>38.6</td>
</tr>
<tr>
<td>delayed</td>
<td>15.6</td>
<td>0</td>
</tr>
<tr>
<td>failsafe</td>
<td>15.6</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>failsafe</td>
<td>15.6</td>
<td>17.9</td>
</tr>
<tr>
<td>repairing</td>
<td>17.9</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>15.6</td>
<td>2.3</td>
</tr>
<tr>
<td>...</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Example – cont’d

This run reads the timed word:

\[(\text{problem}, 23) (\text{delayed}, 38.6) (\text{repair}, 40.9) (\text{done}, 63).\]

<table>
<thead>
<tr>
<th>State</th>
<th>23</th>
<th>15.6</th>
<th>15.6</th>
<th>2.3</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>safe</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>alarm</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>failsafe</td>
<td>38.6</td>
<td>38.6</td>
<td>38.6</td>
<td>40</td>
<td>40</td>
</tr>
</tbody>
</table>

\[x = 0, y = 0, 2 \leq y \land x \leq 56, y = 0\]
Example – cont’d

This run reads the timed word:

\begin{align*}
\text{problem} &: (23) \\
\text{delayed} &: (38.6) \\
\text{repair} &: (40) \\
\text{done} &: (63)
\end{align*}

<table>
<thead>
<tr>
<th>safe</th>
<th>23</th>
<th>safe</th>
<th>problem</th>
<th>alarm</th>
<th>15.6</th>
<th>alarm</th>
<th>delayed</th>
<th>failsafe</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0</td>
<td>23</td>
<td>0</td>
<td>15.6</td>
<td>15.6</td>
<td>15.6</td>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>0</td>
<td>23</td>
<td>23</td>
<td>38.6</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>failsafe</th>
<th>2.3</th>
<th>failsafe</th>
<th>repair</th>
<th>repairing</th>
<th>22.1</th>
<th>repairing</th>
<th>done</th>
<th>safe</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>15.6</td>
<td>17.9</td>
<td>17.9</td>
<td>40</td>
<td>22.1</td>
<td>22.1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example – cont’d

This run reads the timed word

\[
\text{problem, } x:=0 \rightarrow \text{alarm, } \begin{cases} y:=0 & \text{repair, } x \leq 15 \\ 15 \leq x \leq 16 & \text{delayed, } y:=0 \end{cases} \rightarrow \text{failsafe}
\]

<table>
<thead>
<tr>
<th>safe</th>
<th>23</th>
<th>safe</th>
<th>problem</th>
<th>alarm</th>
<th>15.6</th>
<th>alarm</th>
<th>delayed</th>
<th>failsafe</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td>23</td>
<td>0</td>
<td>15.6</td>
<td>0</td>
<td></td>
<td>15.6</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>0</td>
<td>23</td>
<td>23</td>
<td>38.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>failsafe</th>
<th>2.3</th>
<th>failsafe</th>
<th>repair</th>
<th>repairing</th>
<th>22.1</th>
<th>repairing</th>
<th>done</th>
<th>safe</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>15.6</td>
<td>17.9</td>
<td>17.9</td>
<td>40</td>
<td></td>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>0</td>
<td>2.3</td>
<td>0</td>
<td>22.1</td>
<td></td>
<td>22.1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example – cont’d

This run reads the timed word
\[(\text{problem}, 23)(\text{delayed}, 38.6)(\text{repair}, 40.9)(\text{done}, 63).\]
Discrete-time semantics

...because computers are digital!

Discrete-time semantics

...because computers are digital!

Example [Alur91]

- under discrete-time, the output is always 0:

Discrete-time semantics

...because computers are digital!

Example [Alur91]

• under discrete-time, the output is always 0:

Discrete-time semantics

...because computers are digital!

Example [Alur91]

- under discrete-time, the output is always 0:

Discrete-time semantics

...because computers are digital!

Example [Alur91]

- under discrete-time, the output is always 0:

Discrete-time semantics

...because computers are digital!

Example [Alur91]

- under discrete-time, the output is always 0:

Discrete-time semantics

...because computers are digital!

Example [Alur91]

- under continuous-time, the output can be 1:

Discrete-time semantics

...because computers are digital!

Example [Alur91]

Finding the correct granularity (if one exists) is hard!

Continuous-time semantics

...real-time models for real-time systems!
Continuous-time semantics

...real-time models for real-time systems!

Example

\[ x = 1 \]
\[ y := 0 \]
\[ x \leq 2, \; x := 0 \]
\[ y \geq 2, \; y := 0 \]
Continuous-time semantics

...real-time models for real-time systems!

Example

We will focus on the continuous-time semantics, and discuss further its relevance at the end of the tutorial.
Continuous-time semantics

...real-time models for real-time systems!

Example

\[ \begin{align*}
   &x = 1, \quad y := 0 \\
   &x \leq 2, \quad x := 0 \\
   &y \geq 2, \quad y := 0
\end{align*} \]
Continuous-time semantics

...real-time models for real-time systems!

Example

\[ x = 1, \quad y := 0 \]
\[ x \leq 2, \quad x := 0 \]
\[ y \geq 2, \quad y := 0 \]
Continuous-time semantics

...real-time models for real-time systems!

Example

\begin{align*}
\text{Example} & \\
& x=1, y:=0 \\
& x\leq 2, x:=0 \\
& y\geq 2, y:=0
\end{align*}
Continuous-time semantics

...real-time models for real-time systems!

Example

\[
\begin{align*}
x &= 1, & y &= 0 \\
x &\leq 2, & x &= 0 \\
y &\geq 2, & y &= 0
\end{align*}
\]
Continuous-time semantics

...real-time models for real-time systems!

Example

\[
\begin{align*}
    x &= 1, \quad y := 0 \\
    x &\leq 2, \quad x := 0 \\
    y &\geq 2, \quad y := 0
\end{align*}
\]
Continuous-time semantics

...real-time models for real-time systems!

Example

\[ x = 1, \quad y := 0 \]
\[ x \leq 2, \quad x := 0 \]
\[ y \geq 2, \quad y := 0 \]
Continuous-time semantics

...real-time models for real-time systems!

Example

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

\[ x \leq 2, \quad x := 0 \]

\[ y \geq 2, \quad y := 0 \]
Continuous-time semantics

...real-time models for real-time systems!

Example

\[ x = 1, \quad y = 0 \]
\[ x \leq 2, \quad x := 0 \]
\[ y \geq 2, \quad y := 0 \]
Continuous-time semantics

...real-time models for real-time systems!

Example

\[
\begin{align*}
\text{Example} & \quad x=1 \\
y & :=0 \\
x \leq 2, \ x:=0 \\
y \geq 2, \ y:=0
\end{align*}
\]
Continuous-time semantics

...real-time models for real-time systems!

Example

We will focus on the continuous-time semantics, and discuss further its relevance at the end of the tutorial.
Outline

1 Introduction
   - Timed automata
   - Examples

2 Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata

3 Implementation and tools

4 Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)

5 Timed control
   - Timed games
   - Weighted timed games

6 Conclusion
The train crossing example

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

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

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

- **On, $x_i < 20$**
  - $x_i := 0$
The train crossing example – cont’d

The gate:

- **Open** → **Lowering, $H_g < 10$**
  - $H_g < 10, \text{a}$
  - GoDown?, $H_g := 0$

- **Lowering, $H_g < 10$** → **Close**
  - $H_g < 10, \text{a}$
  - GoUp?, $H_g := 0$

- **Raising, $H_g < 10$**
The train crossing example – cont’d

The controller:

- **c₁, H_c ≤ 20**
  - Exit?, H_c := 0
  - H_c = 20, GoUp!

- **c₀**
  - App?, H_c := 0

- **c₂, H_c ≤ 10**
  - Exit?

- **Exit?**
  - H_c ≥ 20, GoUp!
  - H_c ≤ 10, GoDown!
The train crossing example – cont’d

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>.</td>
<td>GoUp?</td>
<td>GoUp!</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>GoDown?</td>
<td>GoDown!</td>
</tr>
</tbody>
</table>

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

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

Some properties one could check:

- Is the gate closed when a train crosses the road?
The train crossing example – cont’d

Some properties one could check:

- Is the gate closed when a train crosses the road?
- Is the gate always closed for less than 5 minutes?
Another example: A mutual exclusion protocol

A mutual exclusion protocol with a shared variable \( id \) [AL94].

Another example: A mutual exclusion protocol

A mutual exclusion protocol with a shared variable \( id \) [AL94].

Process \( i \):

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

\( \leadsto \) a max. delay \( k_1 \) between \( a \) and \( b \)

\( \) a min. delay \( k_2 \) between \( b \) and \( c \)

Another example: A mutual exclusion protocol

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

Process $i$:

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

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

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

$\leadsto$ See the demo with the tool Uppaal

(can be downloaded on http://www.uppaal.com/)

Another example: The task graph scheduling problem

Compute \( D \times (C \times (A+B)) + (A+B) + (C \times D) \) using two processors:

\( P_1 \) (fast):

\[
\begin{array}{c|c}
\text{time} & \\
\hline
+ & 2 \text{ picoseconds} \\
\times & 3 \text{ picoseconds} \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{energy} & \\
\hline
\text{idle} & 10 \text{ Watt} \\
\text{in use} & 90 \text{ Watts} \\
\end{array}
\]

\( P_2 \) (slow):

\[
\begin{array}{c|c}
\text{time} & \\
\hline
+ & 5 \text{ picoseconds} \\
\times & 7 \text{ picoseconds} \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{energy} & \\
\hline
\text{idle} & 20 \text{ Watts} \\
\text{in use} & 30 \text{ Watts} \\
\end{array}
\]

\[
\begin{align*}
&+ \\
&\times \\
&+ \\
&\times \\
&+ \\
&\times
\end{align*}
\]
Another example: The task graph scheduling problem

Compute $D \times (C \times (A+B))+(A+B)+(C \times D)$ using two processors:

$P_1$ (fast):

<table>
<thead>
<tr>
<th>time</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>2 picoseconds</td>
</tr>
<tr>
<td>×</td>
<td>3 picoseconds</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>energy</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>10 Watt</td>
</tr>
<tr>
<td>in use</td>
<td>90 Watts</td>
</tr>
</tbody>
</table>

$P_2$ (slow):

<table>
<thead>
<tr>
<th>time</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>5 picoseconds</td>
</tr>
<tr>
<td>×</td>
<td>7 picoseconds</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>energy</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>20 Watts</td>
</tr>
<tr>
<td>in use</td>
<td>30 Watts</td>
</tr>
</tbody>
</table>

Sch1

0  5  10  15  20  25
T2  T3  T5  T6

Sch2

T1  T4

13 picoseconds 1.37 nanojoules
Another example: The task graph scheduling problem

Compute $D \times (C \times (A+B)) + (A+B) + (C \times D)$ using two processors:

**$P_1$ (fast):**

<table>
<thead>
<tr>
<th>Time</th>
<th>2 picoseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+$</td>
<td>3 picoseconds</td>
</tr>
</tbody>
</table>

**$P_2$ (slow):**

<table>
<thead>
<tr>
<th>Time</th>
<th>5 picoseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+$</td>
<td>7 picoseconds</td>
</tr>
</tbody>
</table>

- **Energy:***
  - **Idle:**
    - $P_1$: 10 Watts
    - $P_2$: 20 Watts
  - **In Use:**
    - $P_1$: 90 Watts
    - $P_2$: 30 Watts

**Sch1**:

- $T_1$: 13 picoseconds, 1.37 nanojoules
- $T_2$: 12 picoseconds, 1.39 nanojoules

**Sch2**:

- $T_1$: 13 picoseconds, 1.37 nanojoules
- $T_2$: 12 picoseconds, 1.39 nanojoules
Another example: The task graph scheduling problem

Compute $D \times (C \times (A+B)) + (A+B) + (C \times D)$ using two processors:

$P_1$ (fast):

<table>
<thead>
<tr>
<th>time</th>
<th>2 picoseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+$</td>
<td>2 picoseconds</td>
</tr>
<tr>
<td>$\times$</td>
<td>3 picoseconds</td>
</tr>
</tbody>
</table>

$P_2$ (slow):

<table>
<thead>
<tr>
<th>time</th>
<th>5 picoseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+$</td>
<td>5 picoseconds</td>
</tr>
<tr>
<td>$\times$</td>
<td>7 picoseconds</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>energy</th>
<th>idle</th>
<th>10 Watt</th>
</tr>
</thead>
<tbody>
<tr>
<td>$idle$</td>
<td>10 Watt</td>
<td></td>
</tr>
<tr>
<td>$in~use$</td>
<td>90 Watts</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>energy</th>
<th>idle</th>
<th>20 Watts</th>
</tr>
</thead>
<tbody>
<tr>
<td>$idle$</td>
<td>20 Watts</td>
<td></td>
</tr>
<tr>
<td>$in~use$</td>
<td>30 Watts</td>
<td></td>
</tr>
</tbody>
</table>

1. Compute $D \times (C \times (A+B)) + (A+B) + (C \times D)$ using two processors:

$P_1$ (fast):

- $P_1(1)$: 13 picoseconds
- $P_1(2)$: 1.37 nanojoules

$P_2$ (slow):

- $P_2(1)$: 12 picoseconds
- $P_2(2)$: 1.39 nanojoules

- $P_2(3)$: 19 picoseconds
- $P_2(4)$: 1.32 nanojoules
Modelling the task graph scheduling problem
Modelling the task graph scheduling problem

**Processors**

- **P₁**:  
  - Start state: \(x=0\)  
  - States: \(x=2\), \(x=3\)  
  - Transitions:  
    - \(x:=0\)  
    - \(done_1\), \(add_1\)  
    - \(x:=0\)  
    - \(done_1\), \(mult_1\)  
  - \((x \leq 2)\)  
  - \((x \leq 3)\)

- **P₂**:  
  - Start state: \(y=0\)  
  - States: \(y=5\), \(y=7\)  
  - Transitions:  
    - \(y:=0\)  
    - \(done_2\), \(add_2\)  
    - \(y:=0\)  
    - \(done_2\), \(mult_2\)  
  - \((y \leq 5)\)  
  - \((y \leq 7)\)

**Questions**

1. Can the computation be made in no more than 10 time units?
2. Is there a scheduling along which no processor is ever idle?
3. ...
Modelling the task graph scheduling problem

**Processors**

- **$P_1$:**
  - $x \leq 2$
  - $x := 0$
  - idle
  - $x = 2$
  - done$_1$
  - add$_1$
  - $x = 3$
  - done$_1$
  - mult$_1$
  - $x \leq 3$

- **$P_2$:**
  - $y \leq 5$
  - $x := 0$
  - idle
  - $y = 5$
  - done$_2$
  - add$_2$
  - $y = 7$
  - done$_2$
  - mult$_2$
  - $y \leq 7$

**Tasks**

- **$T_4$:**
  - $t_1 \land t_2$
  - t$_4 := 1$
  - add$_i$
  - done$_i$

- **$T_5$:**
  - $t_3$
  - t$_5 := 1$
  - add$_i$
  - done$_i$
Modelling the task graph scheduling problem

- **Processors**
  - \( P_1 \):
    - \( x = 2 \) \( \quad \text{done}_1 \quad \text{add}_1 \)
    - \( x = 3 \) \( \quad \text{done}_1 \quad \text{mult}_1 \)
  - \( x = 0 \) \( \quad \text{add}_1 \quad \text{done}_1 \)
  - \( x = 0 \) \( \quad \text{mult}_1 \quad \text{done}_1 \)

- \( x \leq 2 \) \( \quad \text{add}_1 \quad \text{done}_1 \) \( x \leq 3 \)

- \( y = 5 \) \( \quad \text{done}_2 \quad \text{add}_2 \)
  - \( x = 0 \) \( \quad \text{add}_2 \quad \text{done}_2 \)
  - \( x = 0 \) \( \quad \text{mult}_2 \quad \text{done}_2 \)

- \( y \leq 5 \) \( \quad \text{add}_2 \quad \text{done}_2 \) \( y \leq 7 \)

- **Tasks**
  - \( T_4 \):
    - \( t_1 \wedge t_2 \)
    - \( t_4 := 1 \)

  - \( T_5 \):
    - \( t_3 \)
    - \( t_5 := 1 \)

\( \leadsto \) build the synchronized product of all these automata

\[(P_1 \parallel P_2) \parallel_s (T_1 \parallel T_2 \parallel \cdots \parallel T_6)\]
Modelling the task graph scheduling problem

**Processors**

\[ P_1: \]
\[
\begin{align*}
\text{idle} & \xrightarrow{(x \leq 2)} \text{idle} \\
\text{add}_1 & \xrightarrow{x=2} \text{done}_1 \\
\text{mult}_1 & \xrightarrow{x=3} \times
\end{align*}
\]

\[ P_2: \]
\[
\begin{align*}
\text{idles} & \xrightarrow{(y \leq 5)} \text{idles} \\
\text{add}_2 & \xrightarrow{y=5} \text{done}_2 \\
\text{mult}_2 & \xrightarrow{y=7} \times
\end{align*}
\]

\[ P_1: \]
\[
\begin{align*}
\text{idle} & \xrightarrow{(x \leq 3)} \text{idle} \\
\text{add}_1 & \xrightarrow{x=0} \times \\
\text{mult}_1 & \xrightarrow{x=0} \times
\end{align*}
\]

\[ P_2: \]
\[
\begin{align*}
\text{idles} & \xrightarrow{(y \leq 7)} \text{idles} \\
\text{add}_2 & \xrightarrow{y=0} \times \\
\text{mult}_2 & \xrightarrow{y=0} \times
\end{align*}
\]

**Tasks**

\[ T_4: \]
\[
\begin{align*}
t_1 \land t_2 & \xrightarrow{\text{add}_i} \text{done}_i \\
t_4 & \xrightarrow{\text{done}_i} \text{done}_i
\end{align*}
\]

\[ T_5: \]
\[
\begin{align*}
t_3 & \xrightarrow{\text{add}_i} \text{done}_i \\
t_5 & \xrightarrow{\text{done}_i} \text{done}_i
\end{align*}
\]

\[ \sim \text{ build the synchronized product of all these automata} \]

\[(P_1 \parallel P_2) \parallel_s (T_1 \parallel T_2 \parallel \cdots \parallel T_6)\]

A schedule: a path in the global system which reaches \( t_1 \land \cdots \land t_6 \)
Modelling the task graph scheduling problem

**Processors**

\( P_1 : \)

\begin{align*}
& x = 2 \\
& \text{add}_1 \\
& \text{done}_1 \\
& (x \leq 2) \\
& x := 0
\end{align*}

\begin{align*}
& x = 3 \\
& \text{mul}_1 \\
& \text{done}_1 \\
& (x \leq 3) \\
& x := 0
\end{align*}

\( P_2 : \)

\begin{align*}
& y = 5 \\
& \text{add}_2 \\
& \text{done}_2 \\
& (y \leq 5) \\
& x := 0
\end{align*}

\begin{align*}
& y = 7 \\
& \text{mul}_2 \\
& \text{done}_2 \\
& (y \leq 7) \\
& x := 0
\end{align*}

\( \sim \) build the synchronized product of all these automata

\((P_1 \parallel P_2) \parallel_s (T_1 \parallel T_2 \parallel \cdots \parallel T_6)\)

A schedule: a path in the global system which reaches \( t_1 \land \cdots \land t_6 \)

**Tasks**

\( T_4 : \)

\begin{align*}
& t_1 \land t_2 \\
& \text{add}_i \\
& \text{done}_i \\
& t_4 := 1
\end{align*}

\( T_5 : \)

\begin{align*}
& t_3 \\
& \text{add}_i \\
& \text{done}_i \\
& t_5 := 1
\end{align*}

Questions one can ask

- Can the computation be made in no more than 10 time units?
- Is there a scheduling along which no processor is ever idle?
- \( \cdots \)
Outline

1. Introduction
   - Timed automata
   - Examples

2. Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata

3. Implementation and tools

4. Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)

5. Timed control
   - Timed games
   - Weighted timed games

6. Conclusion
Verification

Basic verification problems

- basic reachability/safety properties
- basic liveness properties
Verification

Basic verification problems

- basic reachability/safety properties (final states)
- basic liveness properties (ω-regular conditions)

Is the language accepted by a timed automaton empty?
Verification

Basic verification problems

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

Basic verification problems

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

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

Basic verification problems

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

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

**Theorem** [AD90,AD94]

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

An example [AD94]

Starting at $s_0$, can we visit $s_2$ and then $s_3$?

Method: construct a finite abstraction.

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

\[ x < 1, c \]

\[ x < 1, c \]

\[ y = 1, b \]

\[ y < 1, a, y := 0 \]

\[ x > 1, d \]
An example [AD94]

Starting at $s_0$, can we visit $s_2$ and then $s_3$?
An example [AD94]

Starting at $s_0$, can we visit $s_2$ and then $s_3$?

Method: construct a finite abstraction
Outline

1 Introduction
   - Timed automata
   - Examples

2 Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata

3 Implementation and tools

4 Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)

5 Timed control
   - Timed games
   - Weighted timed games

6 Conclusion
The region abstraction

only constraints:

\[ x \sim c \quad \text{with} \quad c \in \{0, 1, 2\} \]

\[ y \sim c \quad \text{with} \quad c \in \{0, 1, 2\} \]

The path \( x = 1 \) \( y = 1 \) - can be fired from

- cannot be fired from
The region abstraction

only constraints: $x \sim c$ with $c \in \{0, 1, 2\}$
$y \sim c$ with $c \in \{0, 1, 2\}$

“compatibility” between regions and constraints
The region abstraction

- "compatibility" between regions and constraints
- "compatibility" between regions and time elapsing
The region abstraction

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

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- Clocks $x$ and $y$

- "compatibility" between regions and constraints
- "compatibility" between regions and time elapsing

$\sim$ an equivalence of finite index
The region abstraction

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

\[ x \sim c \quad \text{with} \quad c \in \{0, 1, 2\} \]

\[ y \sim c \quad \text{with} \quad c \in \{0, 1, 2\} \]

\[ \text{The path } x=1, y=1 \text{ - can be fired from} \]

\[ \text{The path } x=1, y=1 \text{ - cannot be fired from} \]

\[ \sim \text{ an equivalence of finite index} \]

\[ \sim \text{ a time-abstract bisimulation} \]
Time-abstract bisimulation

This is a relation between • and • such that:
Time-abstract bisimulation

This is a relation between $\bullet$ and $\bullet$ such that:

$$\forall d \geq 0 \exists d' \geq 0 \delta(d) \ldots$$

and vice-versa (swap $\bullet$ and $\bullet$).

Consequence:

$$(\ell_1, v_1) \xrightarrow{a} (\ell_2, v_2)$$

$$\forall d_1, a_1 \rightarrow (\ell_1, R_1) \xrightarrow{a_1} (\ell_1, v_1') \xrightarrow{a_1} (d_1', a_1)$$
Time-abstract bisimulation

This is a relation between $\bullet$ and $\bullet$ such that:

$\forall a \exists a$

$\exists a$
Time-abstract bisimulation

This is a relation between \( \bullet \) and \( \bullet \) such that:

\[
\forall \quad \exists \quad a \\
\forall d \geq 0 \\
\delta(d)
\]
Time-abstract bisimulation

This is a relation between $\bullet$ and $\bullet$ such that:

\[
\forall \bullet \quad a \quad \exists \bullet \\
\exists \bullet \quad a \\
\forall d \geq 0 \quad \delta(d) \quad \exists d' \geq 0 \\
\delta(d')
\]
**Time-abstract bisimulation**

This is a relation between • and • such that:

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

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

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

... and vice-versa (swap • and •).
Time-abstract bisimulation

This is a relation between ● and ● such that:

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

... and vice-versa (swap ● and ●).

Consequence

\[
\forall (\ell_1, v_1) \xrightarrow{d_1,a_1} (\ell_2, v_2) \xrightarrow{d_2,a_2} (\ell_3, v_3) \xrightarrow{d_3,a_3} \ldots
\]
Time-abstract bisimulation

This is a relation between \( \bullet \) and \( \bullet \) such that:

\[
\forall \quad \bullet \xrightarrow{a} \bullet \quad \forall d \geq 0 \quad \bullet \xrightarrow{\delta(d)} \bullet
\]

\[
\exists \quad \bullet \xrightarrow{a} \bullet \quad \exists d' \geq 0 \quad \bullet \xrightarrow{\delta(d')} \bullet
\]

... and vice-versa (swap \( \bullet \) and \( \bullet \)).

Consequence

\[
\forall \quad (\ell_1, v_1) \xrightarrow{d_1, a_1} (\ell_2, v_2) \xrightarrow{d_2, a_2} (\ell_3, v_3) \xrightarrow{d_3, a_3} \cdots
\]

\[
(\ell_1, R_1) \xrightarrow{a_1} (\ell_2, R_2) \xrightarrow{a_2} (\ell_3, R_3) \xrightarrow{a_3} \cdots \quad \text{with } v_i \in R_i
\]
Time-abstract bisimulation

This is a relation between $\bullet$ and $\bullet$ such that:

\[ a \quad a \]

\[ \forall d \geq 0 \quad \exists d' \geq 0 \]

... and vice-versa (swap $\bullet$ and $\bullet$).

**Consequence**

\[ (\ell_1, v_1) \xrightarrow{d_1,a_1} (\ell_2, v_2) \xrightarrow{d_2,a_2} (\ell_3, v_3) \xrightarrow{d_3,a_3} \ldots \]

\[ (\ell_1, R_1) \xrightarrow{a_1} (\ell_2, R_2) \xrightarrow{a_2} (\ell_3, R_3) \xrightarrow{a_3} \ldots \quad \text{with } v_i \in R_i \]

\[ \forall v_1' \in R_1 \]
Time-abstract bisimulation

This is a relation between $\bullet$ and $\bullet$ such that:

$$
\forall \quad \left\{ \begin{array}{c}
\bullet \xrightarrow{a} \bullet \\
\bullet \xleftarrow{a} \bullet \\
\bullet \xrightarrow{\delta(d)} \bullet \\
\bullet \xleftarrow{\delta(d')} \bullet \\
\end{array} \right. \\
\forall d \geq 0 \quad \left\{ \begin{array}{c}
\exists d' \geq 0 \\
\end{array} \right.
$$

... and vice-versa (swap $\bullet$ and $\bullet$).

**Consequence**

$$
\forall \quad \left\{ \begin{array}{c}
(\ell_1, v_1) \xrightarrow{d_1, a_1} (\ell_2, v_2) \xrightarrow{d_2, a_2} (\ell_3, v_3) \xrightarrow{d_3, a_3} \cdots \\
(\ell_1, R_1) \xrightarrow{a_1} (\ell_2, R_2) \xrightarrow{a_2} (\ell_3, R_3) \xrightarrow{a_3} \cdots \quad \text{with } v_i \in R_i \\
\forall v'_1 \in R_1 \exists \quad \left\{ \begin{array}{c}
(\ell_1, v'_1) \xrightarrow{d'_1, a_1} (\ell_2, v'_2) \xrightarrow{d'_2, a_2} (\ell_3, v'_3) \xrightarrow{d'_3, a_3} \cdots \quad \text{with } v'_i \in R_i
\end{array} \right.
\end{array} \right.
$$
The region abstraction

- region $R$ defined by:
  \[
  \begin{cases}
  0 < x < 1 \\
  0 < y < 1 \\
  y < x
  \end{cases}
  \]
The region abstraction

- region $R$ defined by:
  \[
  \begin{cases}
    0 < x < 1 \\
    0 < y < 1 \\
    y < x
  \end{cases}
  \]

- time successors of $R$
The region abstraction

- region $R$ defined by:
  \[
  \begin{cases}
  0 < x < 1 \\
  0 < y < 1 \\
  y < x
  \end{cases}
  \]

- time successors of $R$

image of $R$ when resetting clock $x$
The construction of the region graph

It “mimicks” the behaviours of the clocks.
The construction of the region graph

It “mimicks” the behaviours of the clocks.
Region automaton $\equiv$ finite bisimulation quotient
Region automaton $\equiv$ finite bisimulation quotient

Timed automaton

Region graph

Region automaton

$y < 1, a, x := 0$
Region automaton $\equiv$ finite bisimulation quotient

\[
\begin{align*}
\text{timed automaton} & \quad \equiv \quad \text{region graph} \\
\text{region automaton} & \quad \equiv \quad \text{region automaton}
\end{align*}
\]

language(reg. aut.) = UNTIME(language(timed aut.))
An example [AD94]
An example [AD94]

\[
\begin{align*}
S_0 & \quad x > 0, a, y := 0 \\
S_1 & \quad x < 1, c, x < 1, c, y < 1, a, y := 0 \\
S_2 & \quad y = 1, b \\
S_3 & \quad x > 1, d \\
\end{align*}
\]
An example [AD94]

![Diagram of a formal model with states and transitions](Image)
A large (but finite) automaton, or region automaton, can be obtained from the initial timed automaton through a process involving finite bisimulation and quotienting. This approach allows for the decidable properties of the system to be checked, including reachability, safety, liveness (both Büchi and ω-regular properties), and LTL properties. Challenges such as Zeno behaviors (infinitely many actions in bounded time) can also be addressed within this framework.
large: exponential in the number of clocks and in the constants (if encoded in binary). The number of regions is:

$$
\prod_{x \in X} (2M_x + 2) \cdot |X|! \cdot 2^{|X|}
$$
large: exponential in the number of clocks and in the constants (if encoded in binary). The number of regions is:

\[ \prod_{x \in X} (2M_x + 2) \cdot |X|! \cdot 2^{|X|} \]

It can be used to check for:
- reachability/safety properties
- liveness properties (Büchi/ω-regular properties)
- LTL properties
large: exponential in the number of clocks and in the constants (if encoded in binary). The number of regions is:

\[ \prod_{x \in X} (2M_x + 2) \cdot |X|! \cdot 2^{|X|} \]

It can be used to check for:
- reachability/safety properties
- liveness properties ( Büchi/ω -regular properties)
- LTL properties
- Problems with Zeno behaviours?
  (infinitely many actions in bounded time)
Back to the example
Back to the example

```
s0 → s1
  x>0,a
  y:=0

s1 → s2
  y=1,b
  x<1,c
  x<1,c

s1 → s3
  y<1,a,y:=0

s3 → s1
  x>1,d
```
Back to the example

\[
x > 0, a \\
y := 0 \\
v = 1, b \\
x < 1, c \\
x < 1, c \\
x > 1, d \\
y < 1, a, y := 0
\]
Back to the example

Zeno cycles
Back to the example

Cycles with non-Zeno behaviours
Complexity issues

**Theorem** [AD90,AD94]

The emptiness problem for timed automata is decidable and PSPACE-complete. It even holds for two-clock timed automata [FJ13]. It is NLOGSPACE-complete for one-clock timed automata [LMS04].

---

[AD90] Alur, Dill. Automata for modeling real-time systems (ICALP’90).
[LMS04] Laroussinie, Markey, Schnoebelen. Model checking timed automata with one or two clocks (CONCUR’04).
Complexity issues

Theorem [AD90,AD94]

The emptiness problem for timed automata is decidable and PSPACE-complete. It even holds for two-clock timed automata [FJ13]. It is NLOGSPACE-complete for one-clock timed automata [LMS04].

- **PSPACE upper bound:** guess a path in the region automaton
 Complexity issues

**Theorem [AD90,AD94]**

The emptiness problem for timed automata is decidable and PSPACE-complete. It even holds for two-clock timed automata [FJ13]. It is NLOGSPACE-complete for one-clock timed automata [LMS04].

- **PSPACE upper bound:** guess a path in the region automaton

![Region R defined by:
\[
\begin{array}{l}
0 < x < 1 \\
0 < y < 1 \\
y < x
\end{array}
\]
Complexity issues

**Theorem [AD90,AD94]**

The emptiness problem for timed automata is decidable and PSPACE-complete. It even holds for two-clock timed automata [FJ13]. It is NLOGSPACE-complete for one-clock timed automata [LMS04].

- **PSPACE upper bound**: guess a path in the region automaton
- **PSPACE lower bound**: by reduction from a linearly-bounded Turing machine $M$
The emptiness problem for timed automata is decidable and PSPACE-complete. It even holds for two-clock timed automata [FJ13]. It is NLOGSPACE-complete for one-clock timed automata [LMS04].

- **PSPACE upper bound**: guess a path in the region automaton
- **PSPACE lower bound**: by reduction from a linearly-bounded Turing machine $M$

 maximal number of cells in use: $N$

<table>
<thead>
<tr>
<th>cell $C_i$</th>
<th>cell $C_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$b$</td>
</tr>
</tbody>
</table>

$\text{tape of } M$

$x_i \leq 1$

$x_j > 2$
Example of the simulation of a rule \((q, a, b, q', \rightarrow)\):
Example of the simulation of a rule \((q, a, b, q', \rightarrow)\):

\[
\begin{align*}
q, i &\quad \rightarrow \quad u := 0, u = 2 \\
x_1 \leq 4, x_1 := 0 &\quad \rightarrow \quad x_1 > 4 \\
x_1 > 4 &\quad \rightarrow \quad x_1 \leq 4 \\
x_1 \leq 4 &\quad \rightarrow \quad x_1 := 0 \\
x_1 = 0 &\quad \rightarrow \quad x_1 > 4 \\
x_1 > 4 &\quad \rightarrow \quad x_1 \leq 4 \\
\end{align*}
\]

\[x_j \leq 4: \text{cell } j \text{ contains an } a\]
Example of the simulation of a rule \((q, a, b, q', \rightarrow)\):

\[
\begin{align*}
&u := 0 & q, i \\
&u := 2 & \quad x_1 \leq 4, x_1 := 0 & x_1 > 4 \\
& \quad x_i \leq 4 & x_N \leq 4, x_N := 0 & \quad x_N > 4 \\
& \quad x_i > 4 & u = 3 & q', i + 1
\end{align*}
\]

constraint \(x_j \leq 4\): cell \(j\) contains an \(a\)  
constraint \(x_j > 4\): cell \(j\) contains a \(b\)
Example of the simulation of a rule \((q, a, b, q', \rightarrow)\):

- Initial state: \(q, i\), \(u = 0\)
- Transition: \(u = 2\)
- Next state: \(x_1 \leq 4, x_1 := 0\)
- Transition: \(x_1 > 4\)
- Next state: \(x_i \leq 4\)
- Transition: \(x_N \leq 4, x_N := 0\)
- Next state: \(x_N > 4\)
- Next state: \(q', i + 1\), \(u = 3\)

Constraint \(x_j \leq 4\): cell \(j\) contains an \(a\)
Constraint \(x_j > 4\): cell \(j\) contains a \(b\)
Reset of clock \(x_j\): the new content is an \(a\)
Example of the simulation of a rule \((q, a, b, q', \rightarrow)\):

\[
\begin{align*}
& u := 0 \\
& q, i \\
& \rightarrow \\
& u = 2 \\
& \rightarrow \\
& x_1 \leq 4, x_1 := 0 \\
& x_1 > 4 \\
& \rightarrow \\
& x_i \leq 4 \\
& \rightarrow \\
& x_N \leq 4, x_N := 0 \\
& x_N > 4 \\
& \rightarrow \\
& q', i+1
\end{align*}
\]

Constraint \(x_j \leq 4\): cell \(j\) contains an \(a\)

Constraint \(x_j > 4\): cell \(j\) contains a \(b\)

Reset of clock \(x_j\): the new content is an \(a\)

No reset of clock \(x_j\): the new content is a \(b\)
The case of single-clock timed automata
The case of single-clock timed automata

if only constants 0, 2 and 5 are used
Discussion

- This idea of a finite bisimulation quotient has been applied to many “timed” or “hybrid” systems:
  - various extensions of timed automata
Discussion

- This idea of a finite bisimulation quotient has been applied to many “timed” or “hybrid” systems:
  - various extensions of timed automata
  - model-checking of branching-time properties (TCTL, timed μ-calculus)
Discussion

- This idea of a finite bisimulation quotient has been applied to many “timed” or “hybrid” systems:
  - various extensions of timed automata
  - model-checking of branching-time properties (TCTL, timed \( \mu \)-calculus)
  - weighted/-priced timed automata (e.g. WCTL model-checking, optimal games)
Discussion

- This idea of a finite bisimulation quotient has been applied to many “timed” or “hybrid” systems:
  - various extensions of timed automata
  - model-checking of branching-time properties (TCTL, timed $\mu$-calculus)
  - weighted/priced timed automata (e.g. WCTL model-checking, optimal games)
  - o-minimal hybrid systems
Discussion

- This idea of a finite bisimulation quotient has been applied to many “timed” or “hybrid” systems:
  - various extensions of timed automata
  - model-checking of branching-time properties (TCTL, timed $\mu$-calculus)
  - weighted/priced timed automata (e.g. WCTL model-checking, optimal games)
  - o-minimal hybrid systems
  - ...
Discussion

- This idea of a finite bisimulation quotient has been applied to many "timed" or "hybrid" systems:
  - various extensions of timed automata
  - model-checking of branching-time properties (TCTL, timed μ-calculus)
  - weighted/priced timed automata (e.g. WCTL model-checking, optimal games)
  - o-minimal hybrid systems
  - ...

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

1. Introduction
   - Timed automata
   - Examples

2. Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata

3. Implementation and tools

4. Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)

5. Timed control
   - Timed games
   - Weighted timed games

6. Conclusion
What if we extend the clock constraints?

- Diagonal constraints \((i.e. \ x - y \leq 2)\)
What if we extend the clock constraints?

- Diagonal constraints \((i.e. \; x - y \leq 2)\)
  - decidable (with the same complexity)
What if we extend the clock constraints?

- Diagonal constraints \((i.e. \ x - y \leq 2)\)
  - Decidable (with the same complexity)

is also a time-abstract bisimulation!
What if we extend the clock constraints?

- **Diagonal constraints** \((i.e. \ x - y \leq 2)\)
  - **decidable** (with the same complexity)

\[
\begin{array}{c}
\begin{array}{ccc}
\bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet \\
\end{array}
\end{array}
\]

is also a time-abstract bisimulation!

- they can be removed (at an exponential price)
Removing diagonal constraints

Assume $c \geq 0$

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

\[ x - y \leq c \]

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

copy where "$x - y \leq c$"

copy where "$x - y > c$"
Removing diagonal constraints

Assume $c \geq 0$

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

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

Copy where \( x - y > c \)

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

\[
\begin{align*}
  &x := 0 \\
  &y := 0 \\
  &x - y > c
\end{align*}
\]
Removing diagonal constraints

Assume $c \geq 0$

- $x := 0$
- $y := 0$
- $x - y \leq c$

Copy where "$x - y \leq c$"

- $x := 0$
- $y := 0$

Copy where "$x - y > c$"
Removing diagonal constraints

Assume $c \geq 0$

- $x \leq y \leq c$
  - $x := 0$
  - $y := 0$
  - $x - y \leq c$

- $x > c$
  - $x := 0$
  - $y := 0$
  - $x - y > c$

Copy where "$x - y \leq c$"

Copy where "$x - y > c$"
Removing diagonal constraints

Assume $c \geq 0$

- Assume $x - y \leq c$
  - $x := 0$
  - $y := 0$

- Assume $x - y > c$
  - $x := 0$
  - $y := 0$

- Removes diagonal constraint $x - y = c$

- Utility function: $\frac{37}{100}$
Removing diagonal constraints

Assume $c \geq 0$

\begin{align*}
&x \leq c, y = 0 \\
&x > c, y = 0
\end{align*}

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

copy where \( "x - y > c" \)
What if we extend the clock constraints?

- **Diagonal constraints** *(i.e. $x - y \leq 2$)*
  - **decidable** *(with the same complexity)*

  is also a time-abstract bisimulation!

- they can be removed *(at an exponential price)*
What if we extend the clock constraints?

- **Diagonal constraints** *(i.e. \( x - y \leq 2 \))
  - **decidable** (with the same complexity)

  is also a time-abstract bisimulation!

- **Linear constraints** *(i.e. \( 2x + 3y \sim 5 \))

  they can be removed (at an exponential price)
What if we extend the clock constraints?

- **Diagonal constraints** \((i.e. \ x - y \leq 2)\)
  - **decidable** (with the same complexity)
    - is also a time-abstract bisimulation!
    - they can be removed (at an exponential price)

- **Linear constraints** \((i.e. \ 2x + 3y \sim 5)\)
  - **undecidable** in general
What if we extend the clock constraints?

- **Diagonal constraints** *(i.e. $x - y \leq 2$)*
  - **decidable** (with the same complexity)
    - is also a time-abstract bisimulation!
  - they can be removed (at an exponential price)

- **Linear constraints** *(i.e. $2x + 3y \sim 5$)*
  - **undecidable** in general
  - only **decidable** in few cases
What if we extend the clock constraints?

- **Diagonal constraints** (i.e. $x - y \leq 2$)
  - **decidable** (with the same complexity)
  - is also a time-abstract bisimulation!
  - they can be removed (at an exponential price)

- **Linear constraints** (i.e. $2x + 3y \sim 5$)
  - **undecidable** in general
  - only **decidable** in few cases
  - is a time-abstract bisimulation (when two clocks $x$ and $y$ and constraints $x + y \sim c$)!
What if we allow more operations on clocks?

- that can be **transfer operations** (i.e. $x := y$), or **reinitialization operations** (i.e. $x := 4$), or...

What if we allow more operations on clocks?

- that can be transfer operations (i.e. $x := y$), or reinitialization operations (i.e. $x := 4$), or ...

[BDFP04] Bouyer, Dufourd, Fleury, Petit. Updatable Timed Automata (*Theoretical Computer Science*).

<table>
<thead>
<tr>
<th></th>
<th>simple constraints</th>
<th>+ diagonal constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x := c$, $x := y$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x := x + 1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x := y + c$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x := x - 1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x &lt; c$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x &gt; c$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x \sim y + c$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y + c \leq x :&lt; y + d$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y + c \leq x :&lt; z + d$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What if we allow more operations on clocks?

- that can be transfer operations (i.e. $x := y$), or reinitialization operations (i.e. $x := 4$), or ...

[BDFP04] Bouyer, Dufourd, Fleury, Petit. Updatable Timed Automata (*Theoretical Computer Science*).

<table>
<thead>
<tr>
<th>expression</th>
<th>simple constraints</th>
<th>+ diagonal constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x := c$, $x := y$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x := x + 1$</td>
<td>decided</td>
<td></td>
</tr>
<tr>
<td>$x := y + c$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x := x - 1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x &lt; c$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x &gt; c$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x \sim y + c$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y + c \leq x \leq y + d$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y + c \leq x \leq z + d$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What if we allow more operations on clocks?

- that can be transfer operations (*i.e.* $x := y$), or reinitialization operations (*i.e.* $x := 4$), or ...

[BDFP04] Bouyer, Dufourd, Fleury, Petit. Updatable Timed Automata (*Theoretical Computer Science*).

<table>
<thead>
<tr>
<th></th>
<th>simple constraints</th>
<th>+ diagonal constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x := c, x := y$</td>
<td>decidable</td>
<td></td>
</tr>
<tr>
<td>$x := x + 1$</td>
<td>decidable</td>
<td></td>
</tr>
<tr>
<td>$x := y + c$</td>
<td></td>
<td>undecidable</td>
</tr>
<tr>
<td>$x := x - 1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x &lt; c$</td>
<td>decidable</td>
<td></td>
</tr>
<tr>
<td>$x &gt; c$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x \sim y + c$</td>
<td>decidable</td>
<td></td>
</tr>
<tr>
<td>$y + c &lt;= x &lt; y + d$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y + c &lt;= x &lt;= z + d$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$\sim$ need of being very careful when using more operations on clocks!
Visually...

The example of decrement updates $x := x - 1$

If we want a time-abstract bisimulation...
Visually...

The example of decrement updates $x := x - 1$

If we want a time-abstract bisimulation...
Visually...

The example of decrement updates $x := x - 1$

If we want a time-abstract bisimulation...
Visually...

The example of decrement updates $x := x - 1$

If we want a time-abstract bisimulation...
Visually...

The example of decrement updates \( x := x - 1 \)

If we want a time-abstract bisimulation...
Visually...

The example of decrement updates $x := x - 1$

If we want a time-abstract bisimulation...
Visually...

The example of decrement updates $x := x - 1$

If we want a time-abstract bisimulation...
Visually...

The example of decrement updates $x := x - 1$

If we want a time-abstract bisimulation...
Visually...

**The example of decrement updates** $x := x - 1$

If we want a time-abstract bisimulation...

$\leadsto$ an infinite number of regions
And formally...

- We can simulate a two-counter machine!
And formally...

- We can simulate a two-counter machine!

**Definition**

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

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

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

**Theorem** [Minsky 67]

The halting and recurring problems for two counter machines are undecidable.
And formally...

- We can simulate a two-counter machine!
- Clocks $x$ and $y$ store the two counters...
And formally...

- We can simulate a two-counter machine!
- Clocks $x$ and $y$ store the two counters...

\[ u := 0 \quad \rightarrow \quad u := 1 \]

\[ u := 0, \ y := y - 1 \]

Increment $x$

Decrement $x$
Back to the task-graph scheduling problem

Compute $D \times (C \times (A+B)) + (A+B) + (C \times D)$ using two processors:

**$P_1$ (fast):**

<table>
<thead>
<tr>
<th>time</th>
<th>2 picoseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+$</td>
<td></td>
</tr>
<tr>
<td>$\times$</td>
<td>3 picoseconds</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>energy</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>10 Watt</td>
</tr>
<tr>
<td>in use</td>
<td>90 Watts</td>
</tr>
</tbody>
</table>

**$P_2$ (slow):**

<table>
<thead>
<tr>
<th>time</th>
<th>5 picoseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+$</td>
<td></td>
</tr>
<tr>
<td>$\times$</td>
<td>7 picoseconds</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>energy</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>20 Watts</td>
</tr>
<tr>
<td>in use</td>
<td>30 Watts</td>
</tr>
</tbody>
</table>

![Task-Graph Schedules](https://via.placeholder.com/150)
Back to the task-graph scheduling problem

Compute $D \times (C \times (A+B)) + (A+B) + (C \times D)$ using two processors:

**$P_1$ (fast):**

<table>
<thead>
<tr>
<th>time</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+$</td>
<td>2 picoseconds</td>
</tr>
<tr>
<td>$\times$</td>
<td>3 picoseconds</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>energy</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>10 Watt</td>
</tr>
</tbody>
</table>

**$P_2$ (slow):**

<table>
<thead>
<tr>
<th>time</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+$</td>
<td>5 picoseconds</td>
</tr>
<tr>
<td>$\times$</td>
<td>7 picoseconds</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>energy</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>20 Watts</td>
</tr>
</tbody>
</table>

How to model energy consumption?
A note on hybrid automata (see more on Thursday)

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

[HKPV95] Henzinger, Kopke, Puri, Varaiya. What’s decidable wabout hybrid automata? (STOC’95)
A note on hybrid automata (see more on Thursday)

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

### The thermostat example

- **Off**
  - $\dot{T} = -0.5T$
  - $(T \geq 18)$
  - $T \leq 19$

- **On**
  - $\dot{T} = 2.25 - 0.5T$
  - $(T \leq 22)$

- $T \geq 21$

[HKPV95] Henzinger, Kopke, Puri, Varaiya. What’s decidable about hybrid automata? (*STOC’95*).
A note on hybrid automata (see more on Thursday)

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

The thermostat example

Theorem [HKPV95]
The reachability problem is **undecidable** in hybrid automata, even for stopwatch automata.

(stopwatch automata: timed automata in which clocks can be stopped)

Outline

1 Introduction
   - Timed automata
   - Examples

2 Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata

3 Implementation and tools

4 Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)

5 Timed control
   - Timed games
   - Weighted timed games

6 Conclusion
Weighted timed automata

The model of weighted automata has hybrid variables, which are observer variables (they do not constrain a priori the system) and their weights model the energy consumption, bandwidth, price to pay, etc.
Weighted timed automata

The model of weighted automata

hybrid variables are observer variables (they do not constrain a priori the system)

$\sim$ models energy consumption, bandwidth, price to pay, etc.
Weighted/priced timed automata [ALP01,BFH+01]

\[
\ell_0 \xrightarrow{x \leq 2, c, y := 0} \ell_1 \\
\ell_1 \xrightarrow{} \ell_2 \\
\ell_2 \xrightarrow{x = 2, c} \ell_3 \\
\ell_3 \xrightarrow{x = 2, c} \ell_0
\]

Weighted/priced timed automata [ALP01,BFH+01]

\[ \ell_0 \xrightarrow{1.3} \ell_0 \xrightarrow{c} \ell_1 \xrightarrow{u} \ell_3 \xrightarrow{0.7} \ell_3 \xrightarrow{c} \text{ } \]

\[
\begin{array}{ccccccc}
\ell_0 & \ell_0 & c & l_1 & u & l_3 & u \\
x & 0 & 1.3 & 1.3 & 1.3 & 2 & 0.7 \\
y & 0 & 1.3 & 0 & 0 & 0.7 & \\
\end{array}
\]


Weighted/priced timed automata [ALP01,BFH+01]

\[
\begin{align*}
\ell_0 & \xrightarrow{1.3} \ell_0 \\
\ell_0 & \xrightarrow{c} \ell_1 \\
\ell_1 & \xrightarrow{u} \ell_3 \\
\ell_3 & \xrightarrow{0.7} \ell_3 \\
\ell_3 & \xrightarrow{c} \text{smile}
\end{align*}
\]

\[
\begin{array}{c|c|c|c|c|c|c}
\ell_0 & \ell_0 & \ell_1 & \ell_3 & \ell_3 & \ell_3 & \text{smile} \\
x & 0 & 1.3 & 1.3 & 1.3 & 2 & \\
y & 0 & 1.3 & 0 & 0 & 0.7 & \\
\end{array}
\]

cost :

Weighted/priced timed automata \([\text{ALP01, BFH+01}]\)

\[
\begin{align*}
\ell_0 & \xrightarrow{+5} \ell_0, \quad x \leq 2, c, y := 0 \\
\ell_1 & \xrightarrow{+10} \ell_2, \quad (y = 0) \\
\ell_2 & \xrightarrow{+1} \ell_3, \quad x = 2, c \\
\ell_3 & \xrightarrow{+7} \ell_3, \quad x = 2, c
\end{align*}
\]

\[
\begin{array}{ccccccc}
\ell_0 & \xrightarrow{1.3} & \ell_0 & \xrightarrow{c} & \ell_1 & \xrightarrow{u} & \ell_3 & \xrightarrow{0.7} & \ell_3 \\
x & 0 & 1.3 & 1.3 & 1.3 & 2 & 0 & 0 & 0.7 \\
y & 0 & 1.3 & 0 & 0 & \end{array}
\]

cost : 6.5

[\text{ALP01}] Alur, La Torre, Pappas. Optimal paths in weighted timed automata (HSCC’01).
[\text{BFH+01}] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata (HSCC’01).
Weighted/priced timed automata [ALP01,BFH+01]

\[
\begin{align*}
\ell_0 & \overset{1.3}{\longrightarrow} \ell_0 \\
\ell_0 & \overset{c}{\longrightarrow} \ell_1 \\
\ell_1 & \overset{u}{\longrightarrow} \ell_3 \\
\ell_3 & \overset{0.7}{\longrightarrow} \ell_3 \\
\ell_3 & \overset{c}{\longrightarrow} \text{smile}
\end{align*}
\]

Cost:

\[
\begin{align*}
x & : 0 & 1.3 & 1.3 & 1.3 & 2 \\
y & : 0 & 1.3 & 0 & 0 & 0.7 \\
\text{cost} & : 6.5 + 0
\end{align*}
\]


Weighted/priced timed automata [ALP01,BFH+01]

\[ \ell_0 + 5 \quad x \leq 2, c, y := 0 \quad \ell_1 \quad (y = 0) \quad \ell_2 + 10 \quad u \quad x = 2, c \quad +1 \quad \ell_3 + 1 \quad u \quad x = 2, c \quad +1 \quad \ell_0 \](y = 0)

\[
x = 0 \quad 1.3 \quad c \quad 1.3 \quad u \quad 1.3 \quad 0.7 \quad c \quad \smiley
\]

\[
x \quad 0 \quad 1.3 \quad 1.3 \quad 1.3 \quad 2 \quad 0.7
\]

\[
y \quad 0 \quad 1.3 \quad 0 \quad 0 \quad 0 \quad 0
\]

cost : \quad 6.5 \quad + \quad 0 \quad + \quad 0


Weighted/priced timed automata [ALP01,BFH+01]

\[ \ell_0 \xrightarrow{+5} \ell_0 \xrightarrow{x \leq 2, c, y := 0} \ell_1 \xrightarrow{u} \ell_2 \xrightarrow{x = 2, c} \ell_3 \xrightarrow{c} \ell_3 \xrightarrow{+1} \ell_3 \xrightarrow{+10} \ell_2 \xrightarrow{u} \ell_1 \xrightarrow{u} \ell_0 \]

<table>
<thead>
<tr>
<th></th>
<th>\ell_0</th>
<th>\ell_0</th>
<th>\ell_1</th>
<th>\ell_2</th>
<th>\ell_3</th>
<th>\ell_3</th>
<th>\ell_3</th>
<th>\ell_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>0</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>2</td>
<td>0</td>
<td>0.7</td>
<td>0</td>
</tr>
<tr>
<td>( y )</td>
<td>0</td>
<td>1.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cost</td>
<td>6.5</td>
<td>+</td>
<td>0</td>
<td>+</td>
<td>0</td>
<td>+</td>
<td>0.7</td>
<td>0</td>
</tr>
</tbody>
</table>
Weighted/priced timed automata [ALP01,BFH+01]

\[
\ell_0 \xrightarrow{1.3} \ell_0 \xrightarrow{c} \ell_1 \xrightarrow{u} \ell_3 \xrightarrow{0.7} \ell_3 \xrightarrow{c} \ell_3
\]

\[
\begin{array}{ccc}
\ell_0 & \ell_0 & \ell_1 & \ell_3 \\
x & 0 & 1.3 & 1.3 & 1.3 \\
y & 0 & 1.3 & 0 & 0
\end{array}
\]

\[
\text{cost} : 6.5 + 0 + 0 + 0.7 + 7
\]

Weighted/priced timed automata [ALP01,BFH+01]

\[
\begin{align*}
\ell_0 & \xrightarrow{1.3} \ell_0 & \ell_0 & \xrightarrow{c} \ell_1 & \ell_1 & \xrightarrow{u} \ell_3 & \ell_3 & \xrightarrow{0.7} \ell_3 & \ell_3 & \xrightarrow{c} \smileyface \\
x & 0 & 1.3 & 1.3 & 1.3 & 2 & & & \\
y & 0 & 1.3 & 0 & 0 & 0.7 & & & \\
\text{cost} : & 6.5 & + & 0 & + & 0 & + & 0.7 & + & 7 & = 14.2
\end{align*}
\]

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata \( (HSCC'01) \).

[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata \( (HSCC'01) \).
Weighted/priced timed automata [ALP01,BFH+01]

\[ \ell_0 \rightarrow_5 \ell_1 \quad \ell_1 \rightarrow_1 \ell_2 \quad \ell_2 \rightarrow_1 \ell_3 \quad \ell_3 \rightarrow \text{goal} \]

\[ x \leq 2, c, y := 0 \]

\[ \ell_1 \quad (y = 0) \]

\[ +10 \]

\[ \ell_2 \quad x = 2, c \]

\[ \ell_3 \quad x = 2, c \]

\[ +1 \]

\[ +7 \]

Question: what is the optimal cost for reaching \( \text{goal} \)?

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata (HSCC'01).

Weighted/priced timed automata [ALP01,BFH+01]

Question: what is the optimal cost for reaching 😊?

5t + 10(2 − t) + 1

Weighted/priced timed automata [ALP01,BFH+01]

Question: what is the optimal cost for reaching \(\text{😊} \) ?

\[ 5t + 10(2 - t) + 1, \quad 5t + (2 - t) + 7 \]

Weighted/priced timed automata [ALP01,BFH+01]

\[
\begin{align*}
\ell_0 & \xrightarrow{x \leq 2, c, y := 0} \ell_1 \\
\ell_1 & \xrightarrow{u} \ell_2 \\
\ell_2 & \xrightarrow{x = 2, c} \ell_3 \\
\ell_3 & \xrightarrow{c} \ell_2 \\
\ell_2 & \xrightarrow{+10} +10 \\
\ell_3 & \xrightarrow{+1} +1 \\
\end{align*}
\]

**Question:** what is the optimal cost for reaching \( \smiley \)?

\[
\min (5t + 10(2 - t) + 1, 5t + (2 - t) + 7)
\]
Weighted/priced timed automata [ALP01,BFH+01]

Question: what is the optimal cost for reaching 😊?

\[ \inf_{0 \leq t \leq 2} \min (5t + 10(2 - t) + 1, 5t + (2 - t) + 7) = 9 \]

Weighted/priced timed automata [ALP01,BFH+01]

Question: what is the optimal cost for reaching 😊?

\[
\inf_{0 \leq t \leq 2} \min \left( 5t + 10(2 - t) + 1, \ 5t + (2 - t) + 7 \right) = 9
\]

⇒ strategy: leave immediately ℓ₀, go to ℓ₃, and wait there 2 t.u.

The region abstraction is not fine enough

\[ \text{time elapsing} \]

\[ \text{reset to 0} \]
The corner-point abstraction
The corner-point abstraction

We can somehow discretize the behaviours...
From timed to discrete behaviours

Optimal reachability as a linear programming problem
From timed to discrete behaviours

Optimal reachability as a linear programming problem

\[ t_1 \rightarrow t_2 \rightarrow t_3 \rightarrow t_4 \rightarrow t_5 \rightarrow \ldots \]
From timed to discrete behaviours

Optimal reachability as a linear programming problem

\[
\begin{align*}
& t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad \cdots \\
\xrightarrow{x \leq 2} & \\
\{ & t_1 + t_2 \leq 2
\end{align*}
\]
From timed to discrete behaviours

Optimal reachability as a linear programming problem

\[ t_1 + t_2 \leq 2 \]
\[ t_2 + t_3 + t_4 \geq 5 \]
From timed to discrete behaviours

**Optimal reachability as a linear programming problem**

\[ \begin{align*}
  & t_1 \quad \text{y:=0} \quad t_2 \quad \text{x}\leq 2 \quad t_3 \quad y\geq 5 \quad t_4 \quad t_5 \quad \ldots \\
  & \quad \{ t_1 + t_2 \leq 2 \\
  & \quad t_2 + t_3 + t_4 \geq 5 \}
\end{align*} \]

**Lemma**

Let \( Z \) be a bounded constraint as above and \( f \) be a function

\[
f : (t_1, \ldots, t_n) \mapsto \sum_{i=1}^{n} c_i t_i + c
\]

well-defined on \( \overline{Z} \). Then \( \inf_Z f \) is obtained on the border of \( \overline{Z} \) with integer coordinates.
From timed to discrete behaviours

Optimal reachability as a linear programming problem

\[ \begin{align*}
  y := 0 & \quad t_1 & \quad x \leq 2 & \quad t_2 & \quad y \geq 5 & \quad t_3 & \quad t_4 & \quad t_5 & \quad \ldots \\
  t_1 + t_2 & \leq 2 & \quad t_2 + t_3 + t_4 & \geq 5
\end{align*} \]

Lemma

Let \( Z \) be a bounded constraint as above and \( f \) be a function

\[ f : (t_1, \ldots, t_n) \mapsto \sum_{i=1}^{n} c_i t_i + c \]

well-defined on \( \overline{Z} \). Then \( \inf_{Z} f \) is obtained on the border of \( \overline{Z} \) with integer coordinates.

\( \rightsquigarrow \) for every finite path \( \pi \) in \( A \), there exists a path \( \Pi \) in \( A_{cp} \) such that

\[ \text{cost}(\Pi) \leq \text{cost}(\pi) \]

[\( \Pi \) is a “corner-point projection” of \( \pi \)]
From discrete to timed behaviours

Approximation of abstract paths:

For any path $\Pi$ of $A_{cp}$,
Approximation of abstract paths:

For any path $\Pi$ of $A_{cp}$, for any $\varepsilon > 0$,
From discrete to timed behaviours

**Approximation of abstract paths:**

For any path $\Pi$ of $A_{cp}$, for any $\varepsilon > 0$, there exists a path $\pi_\varepsilon$ of $A$ s.t.

$$\|\Pi - \pi_\varepsilon\|_\infty < \varepsilon$$
From discrete to timed behaviours

Approximation of abstract paths:

For any path $\Pi$ of $A_{cp}$, for any $\varepsilon > 0$, there exists a path $\pi_\varepsilon$ of $A$ s.t.

$$\|\Pi - \pi_\varepsilon\|_\infty < \varepsilon$$

For every $\eta > 0$, there exists $\varepsilon > 0$ s.t.

$$\|\Pi - \pi_\varepsilon\|_\infty < \varepsilon \implies |\text{cost}(\Pi) - \text{cost}(\pi_\varepsilon)| < \eta$$
Theorem [ALP01,BFH+01,BBBR07]
The optimal-cost reachability problem is decidable (and PSPACE-complete) in weighted timed automata.
Further problems of interest

Relevant questions

- **Optimization questions:**
  - optimal reachability
  - optimal average consumption
  - ...

- **Management of resources:**
  - a lower bound global constraint (your bank account)
  - a lower and an upper bound global constraint (the tank of your car, the pressure in a pump)
  - ...

~ lots of developments, many open problems

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata (*HSCC’01*).
[BFLMS08] Bouyer, Fahrenberg, Larsen, Markey, Srba. Infinite runs in weighted timed automata with energy constraints (*FORMATS’08*).
Outline

1 Introduction
   - Timed automata
   - Examples

2 Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata

3 Implementation and tools

4 Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)

5 Timed control
   - Timed games
   - Weighted timed games

6 Conclusion
What about the practice?

- the region automaton is never computed
- instead, symbolic computations are performed
What about the practice?

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

What do we need?

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

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

What do we need?

- Need of a symbolic representation:

  Finite representation of infinite sets of configurations

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

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

What do we need?

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

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

What do we need?

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

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

What do we need?

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

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

What do we need?

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

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

What do we need?

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

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

- Need of abstractions, heuristics, etc...
Zones: A symbolic representation for timed systems

Example of a zone and its DBM representation

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

DBM: Difference Bound Matrice [BM83,Dill89]


Zones: A symbolic representation for timed systems

Example of a zone and its DBM representation

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

DBM: Difference Bound Matrice [BM83,Dill89]

Zones: A symbolic representation for timed systems

Example of a zone and its DBM representation

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

DBM: Difference Bound Matrice [BM83,Dill89]

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

Example of a zone and its DBM representation

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

DBM: Difference Bound Matrice [BM83,Dill89]

Backward computation
Backward computation
Backward computation
Backward computation
Backward computation
Note on the backward analysis of timed automata

\[
\begin{align*}
g, a, Y &:= 0 \\
\ell &\rightarrow \ell' \\
\left[ C \leftarrow 0 \right]^{-1} (Z \cap (C = 0)) \cap g &\cap Z
\end{align*}
\]
Note on the backward analysis of timed automata

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

\[
\ell \quad g, a, Y := 0 \quad \ell'
\]

\[
y \leftarrow 0 - 1 (Z \cap (C = 0)) \cap g
\]

\[
Z
\]

\[
x \quad y
\]

\[
Z
\]
Note on the backward analysis of timed automata

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

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

Diagram:

- For \( x \):
  - Blue region: \( Z \)
  - Red region: \( [y \leftarrow 0]^{-1}(Z \cap (y = 0)) \)

- For \( y \):
  - Blue region: \( Z \)
  - Red region: \( [y \leftarrow 0]^{-1}(Z \cap (y = 0)) \)
Note on the backward analysis of timed automata

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

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

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

\[ Z \]

\[ Z \]

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

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

\[ Z \]

\[ y \]
\[ x \]

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

\[ Z \]

\[ y \]
\[ x \]
Note on the backward analysis (cont.)

😊 All previous operations can be computed using DBMs!
Note on the backward analysis (cont.)

😊 All previous operations can be computed using DBMs!

- intersection: take the minimum of the two constraints
Note on the backward analysis (cont.)

All previous operations can be computed using DBMs!

- **intersection**: take the minimum of the two constraints
- **inverse reset w.r.t \( y \)**: relax constraints on \( y \) (on a DBM on normal form)
Note on the backward analysis (cont.)

😊 All previous operations can be computed using DBMs!

- **intersection**: take the minimum of the two constraints
- **inverse reset w.r.t $y$**: relax constraints on $y$ (on a DBM on normal form)
- **past**: relax lower bounds (on a DBM on normal form)
Note on the backward analysis (cont.)

😊 All previous operations can be computed using DBMs!

- **intersection**: take the minimum of the two constraints
- **inverse reset w.r.t. $y$**: relax constraints on $y$ (on a DBM on normal form)
- **past**: relax lower bounds (on a DBM on normal form)
- **emptiness**: check whether there is a negative cycle
Note on the backward analysis (cont.)

😊 All previous operations can be computed using DBMs!

- **intersection**: take the minimum of the two constraints
- **inverse reset w.r.t. \( y \)**: relax constraints on \( y \) (on a DBM on normal form)
- **past**: relax lower bounds (on a DBM on normal form)
- **emptiness**: check whether there is a negative cycle

The backward computation terminates

Because of the bisimulation property of the region abstraction:

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

😊 All previous operations can be computed using DBMs!

- **intersection**: take the minimum of the two constraints
- **inverse reset w.r.t $y$**: relax constraints on $y$ (on a DBM on normal form)
- **past**: relax lower bounds (on a DBM on normal form)
- **emptiness**: check whether there is a negative cycle

The backward computation terminates

Because of the bisimulation property of the region abstraction:

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

Let $R$ be a region. Assume:

- $v \in \hat{\downarrow} R$ (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 $v' \in \hat{\downarrow} R$. 
Note on the backward analysis (cont.)

😊 All previous operations can be computed using DBMs!

- **intersection**: take the minimum of the two constraints
- **inverse reset w.r.t $y$**: relax constraints on $y$ (on a DBM on normal form)
- **past**: relax lower bounds (on a DBM on normal form)
- **emptiness**: check whether there is a negative cycle

The backward computation terminates

Because of the bisimulation property of the region abstraction:

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

However the backward computation is not appropriate to manipulate other variables (think for instance of assignment $i := j.k + l$)
Forward computation
Forward computation
Forward computation
Forward computation
Forward computation
Forward analysis of timed automata

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

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

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

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

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

\[ \ell, a, Y := 0 \quad \rightarrow \quad \ell' \]

\[ Z \quad \rightarrow \quad [Y \leftarrow 0](\overrightarrow{Z} \cap g) \]

\[ Z \quad \overrightarrow{Z} \quad \overrightarrow{Z} \cap g \]
Forward analysis of timed automata

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

\[ Z \]

\[ [Y \leftarrow 0](\overrightarrow{Z} \cap g) \]
Note on the forward analysis (cont.)

😊 All previous operations can be computed using DBMs!

- **intersection**: take the minimum of the two constraints
- **reset w.r.t y**: set constraint if $y$ to 0 (on a DBM on normal form)
- **future**: relax upper bounds (on a DBM on normal form)
- **emptiness**: check whether there is a negative cycle
Note on the forward analysis (cont.)

😊 All previous operations can be computed using DBMs!

- intersection: take the minimum of the two constraints
- reset w.r.t $y$: set constraint if $y$ to 0 (on a DBM on normal form)
- future: relax upper bounds (on a DBM on normal form)
- emptiness: check whether there is a negative cycle

😢 The forward computation may not terminate...

$x \geq 1, y = 1$

$y := 0$
Note on the forward analysis (cont.)

😊 All previous operations can be computed using DBMs!
- intersection: take the minimum of the two constraints
- reset w.r.t \( y \): set constraint if \( y \) to 0 (on a DBM on normal form)
- future: relax upper bounds (on a DBM on normal form)
- emptiness: check whether there is a negative cycle

😢 The forward computation may not terminate...

\[
x \geq 1, y = 1 \\
y := 0
\]
Note on the forward analysis (cont.)

😊 All previous operations can be computed using DBMs!
- **intersection**: take the minimum of the two constraints
- **reset w.r.t. y**: set constraint if $y$ to 0 (on a DBM on normal form)
- **future**: relax upper bounds (on a DBM on normal form)
- **emptiness**: check whether there is a negative cycle

😞 The forward computation may not terminate...

$x \geq 1, y = 1$

$y := 0$

$x \geq 0, y = 0$

$y := 0$
Note on the forward analysis (cont.)

巡回 Intersection: take the minimum of the two constraints
- Reset w.r.t $y$: set constraint if $y$ to 0 (on a DBM on normal form)
- Future: relax upper bounds (on a DBM on normal form)
- Emptiness: check whether there is a negative cycle

The forward computation may not terminate...

$x \geq 1, y = 1$
$y := 0$
Note on the forward analysis (cont.)

😊 All previous operations can be computed using DBMs!
- intersection: take the minimum of the two constraints
- reset w.r.t \( y \): set constraint if \( y \) to 0 (on a DBM on normal form)
- future: relax upper bounds (on a DBM on normal form)
- emptiness: check whether there is a negative cycle

😢 The forward computation may not terminate...

\[ x \geq 1, \ y = 1 \]
\[ y := 0 \]
Note on the forward analysis (cont.)

😊 All previous operations can be computed using DBMs!

- intersection: take the minimum of the two constraints
- reset w.r.t. $y$: set constraint if $y$ to 0 (on a DBM on normal form)
- future: relax upper bounds (on a DBM on normal form)
- emptiness: check whether there is a negative cycle

😢 The forward computation may not terminate...

$x \geq 1, y = 1$
$y := 0$
Note on the forward analysis (cont.)

😊 All previous operations can be computed using DBMs!
- intersection: take the minimum of the two constraints
- reset w.r.t. $y$: set constraint if $y$ to 0 (on a DBM on normal form)
- future: relax upper bounds (on a DBM on normal form)
- emptiness: check whether there is a negative cycle

😢 The forward computation may not terminate...

$x \geq 1, y = 1$
$y := 0$

$\xrightarrow{\sim} \text{an infinite number of steps...}$
An abstraction: the extrapolation operator

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

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

\rightarrow \text{ The extrapolation operator ensures termination of the computation!}
An abstraction: the extrapolation operator

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

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

$\Rightarrow$ The extrapolation operator ensures termination of the computation!
The extrapolation: correctness

**Theorem**

The algorithm using the extrapolation w.r.t. the maximal constant is correct for timed automata with only rectangular constraints.  
*Note*: the hypothesis on the constraints is crucial.

The extrapolation: correctness

**Theorem**

The algorithm using the extrapolation w.r.t. the maximal constant is correct for timed automata with only rectangular constraints.

*Note:* the hypothesis on the constraints is crucial.

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

Improving the classical algorithm

- the extrapolation operator can be made coarser:
  - local extrapolation constants [BBFL03];
  - distinguish between lower- and upper-bounded contraints [BBLP03,BBLP06,HSW12,HSW13]

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

- the extrapolation operator can be made coarser:
  - local extrapolation constants \([\text{BBFL03}]\);
  - distinguish between lower- and upper-bounded contraints \([\text{BBLP03,BBLP06,HSW12,HSW13}]\);
- over-approximations can be used
  - convex-hull

---

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

- the extrapolation operator can be made coarser:
  - local extrapolation constants [BBFL03];
  - distinguish between lower- and upper-bounded constraints [BBLP03,BBLP06,HSW12,HSW13]
- over-approximations can be used
  - convex-hull
- heuristics can be added
  - order for exploration
  - symmetry reduction [HBL+03]

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

- the extrapolation operator can be made coarser:
  - local extrapolation constants \([\text{BBFL03}]\);
  - distinguish between lower- and upper-bounded contraints \([\text{BBLP03,BBLP06,HSW12,HSW13}]\)
- over-approximations can be used
  - convex-hull
- heuristics can be added
  - order for exploration
  - symmetry reduction \([\text{HBL+03}]\)
- the representation of zones can be improved \([\text{DHLP06}]\)

---

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

1. Introduction
   - Timed automata
   - Examples

2. Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata

3. Implementation and tools

4. Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)

5. Timed control
   - Timed games
   - Weighted timed games

6. Conclusion
Outline

1. Introduction
   - Timed automata
   - Examples

2. Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata

3. Implementation and tools

4. Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)

5. Timed control
   - Timed games
   - Weighted timed games

6. Conclusion
Strong timed (bi)simulation

This is a relation between $\bullet$ and $\bullet$ such that:
Strong timed (bi)simulation

This is a relation between • and • such that:

∀ • a •

∀ \exists \delta(d)
\forall d > 0 \exists \delta(d)

... and vice-versa (swap • and •) for the bisimulation relation.

Theorem
Strong timed (bi)simulation between timed automata is decidable and EXPTIME-complete.
(see later for a simple proof of the upper bound)
Strong timed (bi)simulation

This is a relation between • and • such that:

∀

∃

a

a
Strong timed (bi)simulation

This is a relation between • and • such that:

\[ \forall \exists \quad a \quad d > 0 \quad \delta(d) \]
Strong timed (bi)simulation

This is a relation between • and • such that:

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

\[
\exists \forall \quad a \\
\exists \forall \quad \delta(d)
\]
Strong timed (bi)simulation

This is a relation between ● and ● such that:

\[
\forall \exists \quad \begin{array}{c}
\text{a} \\
\text{a}
\end{array} \\
\forall \exists \quad \begin{array}{c}
\text{δ}(d) \\
\text{δ}(d)
\end{array}
\]

... and vice-versa (swap ● and ●) for the bisimulation relation.
Strong timed (bi)simulation

This is a relation between • and • such that:

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

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

\[ a \quad \delta(d) \]

... and vice-versa (swap • and •) for the bisimulation relation.

**Theorem**

Strong timed (bi)simulation between timed automata is decidable and EXPTIME-complete.

(see later for a simple proof of the upper bound)
Language (or trace) equivalence and inclusion

Question
Given two timed automata $A$ and $B$, is $L(A) = L(B)$ (resp. $L(A) \subseteq L(B)$)?

Theorem [AD90, AD94]
Language equivalence and language inclusion are undecidable in timed automata.
Language (or trace) equivalence and inclusion

Question

Given two timed automata \( A \) and \( B \), is \( L(A) = L(B) \) (resp. \( L(A) \subseteq L(B) \))?

Theorem [AD90,AD94]

Language equivalence and language inclusion are undecidable in timed automata.

... as a special case of the universality problem (are all timed words accepted by the automaton?).

[AD90] Alur, Dill. Automata for modeling real-time systems (ICALP’90).
Language (or trace) equivalence and inclusion

Question

Given two timed automata $A$ and $B$, is $L(A) = L(B)$ (resp. $L(A) \subseteq L(B)$)?

Theorem [AD90, AD94]

Language equivalence and language inclusion are undecidable in timed automata.

... as a special case of the universality problem (are all timed words accepted by the automaton?).

$\sim$ Proof by reduction from the recurring problem of a two-counter machine

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

**Theorem** [AD90,AD94]

Universality of timed automata is undecidable.
Undecidability of universality

**Theorem [AD90,AD94]**

Universality of timed automata is undecidable.

- one configuration is encoded in one time unit
- number of \(c\)'s: value of counter \(c\)
- number of \(d\)'s: value of counter \(d\)
- one time unit between two corresponding \(c\)'s (resp. \(d\)'s)
Undecidability of universality

**Theorem [AD90,AD94]**

Universality of timed automata is undecidable.

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

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

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

\[
\begin{align*}
S_0 & \xrightarrow{b_i, x := 0} S_1 & x < 1, c, x := 0 & \xrightarrow{x = 1, \neg c} S_2 \\
S_1 & \xrightarrow{x \neq 1} S_2
\end{align*}
\]
Example

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

The union of all small modules is not universal iff

The two-counter machine has a recurring computation
## Bad consequences

- Language inclusion is **undecidable**  
  (Bad news for the application to verification)  
- Complementability is **undecidable**  
- ...  

---

[Tri03] Tripakis. Folk theorems on the determinization and minimization of timed automata (*FORMATS’03*).  
[Fin06] Finkel. Undecidable problems about timed automata (*FORMATS’06*).
Bad consequences

- Language inclusion is undecidable
  (Bad news for the application to verification)
- Complementability is undecidable
- ...

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

\[ s_0 \xrightarrow{a, x := 0} s_1 \xrightarrow{x = 1, a} s_2 \]

[Tri03] Tripakis. Folk theorems on the determinization and minimization of timed automata (FORMATS’03).
[Fin06] Finkel. Undecidable problems about timed automata (FORMATS’06).
Bad consequences

- Language inclusion is **undecidable**
  (Bad news for the application to verification)
- Complementability is **undecidable**
- ...

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

\[
\begin{align*}
S_0 \xrightarrow{a, b} S_0 & \quad \text{a, } x := 0 \\
S_0 \xrightarrow{a, x \neq 1, a, b} S_1 \xrightarrow{a, x := 0} S_1
\end{align*}
\]

---

[Tri03] Tripakis. Folk theorems on the determinization and minimization of timed automata (*FORMATS’03*).
[Fin06] Finkel. Undecidable problems about timed automata (*FORMATS’06*).
Bad consequences

- Language inclusion is **undecidable**  
  (Bad news for the application to verification)  
- Complementability is **undecidable**  
- ...

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

\[
\begin{align*}
S_0 & \xrightarrow{a,x:=0} S_0 \\
S_0 & \xrightarrow{a,b} S_1 \\
S_1 & \xrightarrow{x \neq 1,a,b} S_1
\end{align*}
\]

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

---

[Tri03] Tripakis. Folk theorems on the determinization and minimization of timed automata (*FORMATS’03*).

[Fin06] Finkel. Undecidable problems about timed automata (*FORMATS’06*).

Outline

1. Introduction
   - Timed automata
   - Examples

2. Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata

3. Implementation and tools

4. Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)

5. Timed control
   - Timed games
   - Weighted timed games

6. Conclusion
Timed temporal logics

- **Branching-time: TCTL**

  $$\text{TCTL} \ni \phi ::= a \mid \neg \phi \mid \phi \lor \phi \mid \phi \land \phi \mid E \phi U_I \phi$$

  where $I$ is an interval with integral bounds.

- **Linear-time: MTL**

  $$\text{MTL} \ni \phi ::= a \mid \neg \phi \mid \phi \lor \phi \mid \phi \land \phi \mid \phi U_I \phi$$

  where $I$ is an interval with integral bounds.

- **Alternative:** add variables (clocks) to the logics, e.g. TPTL

---

Timed temporal logics

- **Branching-time:** TCTL

\[
\text{TCTL} \ni \varphi ::= a \mid \neg \varphi \mid \varphi \lor \varphi \mid \varphi \land \varphi \mid E\varphi \ U I \varphi
\]

where \( I \) is an interval with integral bounds.

- **Linear-time:** MTL

\[
\text{MTL} \ni \varphi ::= a \mid \neg \varphi \mid \varphi \lor \varphi \mid \varphi \land \varphi \mid \varphi \ U I \varphi
\]

where \( I \) is an interval with integral bounds.

- **Alternative:** add variables (clocks) to the logics, e.g. TPTL

\[\sim \] interpreted over **signals**

Timed temporal logics

- **Branching-time:** TCTL

\[
\text{TCTL} \ni \varphi ::= \ a \mid \neg \varphi \mid \varphi \lor \varphi \mid \varphi \land \varphi \mid E\varphi \ U_I \varphi
\]

where \( I \) is an interval with integral bounds.

- **Linear-time:** MTL

\[
\text{MTL} \ni \varphi ::= \ a \mid \neg \varphi \mid \varphi \lor \varphi \mid \varphi \land \varphi \mid \varphi \ U_I \varphi
\]

where \( I \) is an interval with integral bounds.

- **Alternative:** add variables (clocks) to the logics, e.g. TPTL

\[ \sim \text{ interpreted over } \textbf{signals} \overset{}{\sim} \text{ (or over timed words) } \]

\[ (\bullet, .6) (\bullet, 1.1) (\bullet, 1.2) (\bullet, 1.3) \ldots \]

Timed temporal logics

- **Branching-time:** TCTL

  \[
  \text{TCTL} \ni \varphi ::= a | \neg \varphi | \varphi \lor \varphi | \varphi \land \varphi | E\varphi U_i \varphi
  \]

  where \( i \) is an interval with integral bounds.

- **Linear-time:** MTL \[Koy90\]

  \[
  \text{MTL} \ni \varphi ::= a | \neg \varphi | \varphi \lor \varphi | \varphi \land \varphi | \varphi U_i \varphi
  \]

  where \( i \) is an interval with integral bounds.

- **Alternative:** add variables (clocks) to the logics, e.g. TPTL

  \[
  \sim \text{interpreted over signals (or over timed words)}
  \]

\[Koy90\] Koymans. Specifying real-time properties with metric temporal logic (Real-time systems, 1990).
Examples

\[ \begin{array}{c}
0 & 1 & 2 & 3 & 4 \\
\in [1, 2] \\
\end{array} \]

\[ \models \ U_{[1, 2]} \]
Examples

\[ 0 \quad 1 \quad 2 \quad 3 \quad 4 \]

\[ \in [1,2] \]

\[ G(\text{problem} \rightarrow (F \leq 56 \text{alarm})) \]

\[ G(\text{problem} \rightarrow (F \leq 15 \text{repair} \lor G[12,15) \text{alarm})) \]

\[ \models \bullet U_{[1,2]} \bullet \]

\[ \not\models G_{[2,3]} \bullet \]
Examples

• “Every problem is followed within 56 time units by an alarm”

\[ G(\text{problem} \rightarrow F_{\leq 56} \text{alarm}) \]
Examples

- “Every problem is followed within 56 time units by an alarm”
  \[ G(\text{problem} \rightarrow F_{\leq 56} \text{alarm}) \]

- “Each time there is a problem, it is either repaired within the next 15 time units, or an alarm rings during 3 time units 12 time units later”
  \[ G(\text{problem} \rightarrow (F_{\leq 15} \text{repair} \vee G_{[12,15]} \text{alarm})) \]
Model-checking timed temporal logics

Branching-time logic TCTL [ACD93]

The model-checking of TCTL is PSPACE-complete!
(The region abstraction can be used, with an extra clock for the formula)

Model-checking timed temporal logics

**Branching-time logic TCTL** [ACD93]
The model-checking of TCTL is PSPACE-complete!
(The region abstraction can be used, with an extra clock for the formula)

**Linear-time logic MTL** [AFH96,OW05]
The model-checking of MTL is undecidable/NPR.
Some fragments with decidable model-checking have been designed.

[OW05] Ouaknine, Worrell. On the decidability of metric temporal logic (LICS’05).
Model-checking timed temporal logics

Technics: alternating timed automata, channel machines, small-model properties

[OW06] Ouaknine, Worrell. Safety Metric Temporal Logic is Fully Decidable (TACAS’06).
Model-checking timed temporal logics

**Branching-time logic TCTL [ACD93]**

The model-checking of TCTL is PSPACE-complete!
(The region abstraction can be used, with an extra clock for the formula)

**Linear-time logic MTL [AFH96,OW05]**

The model-checking of MTL is undecidable/NPR.
Some fragments with decidable model-checking have been designed.

[OW05] Ouaknine, Worrell. On the decidability of metric temporal logic (LICS’05).
A focus on MITL

The nightmare of timed temporal logics
Requiring too much precision, and hence too many clocks!!
A focus on MITL

The nightmare of timed temporal logics
Requiring too much precision, and hence too many clocks!!

Example

$G(\mathbf{\bullet} \rightarrow F_{=1} \mathbf{\bullet})$

- each time an $\mathbf{\bullet}$ occurs, start a new clock, and check that a $\mathbf{\bullet}$ occurs 1 time unit later
- this requires an unbounded number of clocks
A focus on MITL

The nightmare of timed temporal logics

Requiring too much precision, and hence too many clocks!!

Example

\[ G(\bullet \rightarrow F_{\geq 1} \bullet) \]

- each time an \( \bullet \) occurs, start a new clock, and check that a \( \bullet \) occurs 1 time unit later
- this requires an unbounded number of clocks

The logic MITL

- Bans “punctual” constraints
- Consequences:
  - we can bound the variability of signals
  - an MITL formula defines a timed regular language
Formula $G_{(0,1)}(a \rightarrow F_{[1,2]} b)$
Formula $G_{(0,1)}(a \rightarrow F_{[1,2]} b)$
Formula $G_{(0,1)}(a \rightarrow F_{[1,2]} b)$
Formula $G_{(0,1)}(a \to F_{[1,2]} b)$
Formula $G_{(0,1)}(a \rightarrow F_{[1,2]} b)$

This idea can be extended to any formula in MITL.
Outline

1 Introduction
   - Timed automata
   - Examples

2 Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata

3 Implementation and tools

4 Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)

5 Timed control
   - Timed games
   - Weighted timed games

6 Conclusion
Why (timed) games?

- to model uncertainty

Example of a processor in the taskgraph example

![Diagram of a processor in a taskgraph example](image)
Why (timed) games?

- to model uncertainty

Example of a processor in the taskgraph example
Why (timed) games?

- to model uncertainty

Example of a processor in the taskgraph example

Example of the gate in the train/gate example
Why (timed) games?

- to model uncertainty

Example of a processor in the taskgraph example

```
+ (x ≤ 2)
 \( x ≥ 1 \) done
 add \( x := 0 \)
 idle
 \( x ≥ 1 \) done
 mult \( x := 0 \)
 × (x ≤ 3)
```

- to model an interaction with the environment

Example of the gate in the train/gate example
Why (timed) games?

- to model uncertainty

Example of a processor in the taskgraph example

Example of the gate in the train/gate example
Outline

1. Introduction
   - Timed automata
   - Examples

2. Decidability of basic properties
   - The region abstraction
   - Extensions of timed automata
   - Weighted timed automata

3. Implementation and tools

4. Other verification problems
   - Equivalence (or preorder) checking
   - Verification of timed temporal logics (short)

5. Timed control
   - Timed games
   - Weighted timed games

6. Conclusion
An example of a timed game

Rule of the game

- **Aim**: avoid 😞 and reach 😊

![Diagram of a timed game with nodes labeled as $\ell_0$, $\ell_1$, $\ell_2$, and $\ell_3$. The transitions are labeled with conditions such as $x \leq 2$, $x \leq 1, c_1$, $x < 1, u_1$, $x < 1, u_2, x := 0$, $x \geq 1, u_3$, $x \geq 2, c_4$, and $x \leq 1, c_3$. The transitions are directed with arrows indicating the flow of the game.]
An example of a timed game

Rule of the game

- **Aim:** avoid 😞 and reach 😊
- **How do we play?** According to a strategy:
An example of a timed game

Rule of the game

- **Aim:** avoid 😞 and reach 😊
- **How do we play?** According to a strategy:

\[ f : \text{history} \mapsto (\text{delay, cont. transition}) \]
An example of a timed game

Rule of the game

- **Aim:** avoid 😞 and reach 😊
- **How do we play?** According to a strategy:

  \[ f : \text{history} \mapsto (\text{delay}, \text{cont. transition}) \]

A (memoryless) winning strategy

- from \((\ell_0, 0)\), play \((0.5, c_1)\)
An example of a timed game

Rule of the game
- **Aim:** avoid 😞 and reach 😊
- **How do we play?** According to a strategy:
  
  \[ f : \text{history} \mapsto (\text{delay, cont. transition}) \]

A (memoryless) winning strategy
- from \((\ell_0, 0)\), play \((0.5, c_1)\)
  \[ \rightsquigarrow \text{can be preempted by } u_2 \]
An example of a timed game

Rule of the game

- **Aim**: avoid 🙁 and reach 🙂
- **How do we play?** According to a strategy:

  \[ f : \text{history} \mapsto (\text{delay}, \text{cont. transition}) \]

A (memoryless) winning strategy

- from \((\ell_0, 0)\), play \((0.5, c_1)\)
  \[ \sim \text{ can be preempted by } u_2 \]
- from \((\ell_2, \ast)\), play \((1 - \ast, c_2)\)
An example of a timed game

Rule of the game

- **Aim:** avoid 😞 and reach 😊
- **How do we play?** According to a strategy:

  \[ f : \text{history} \mapsto (\text{delay, cont. transition}) \]

A (memoryless) winning strategy

- from \((\ell_0, 0)\), play \((0.5, c_1)\)
  \(\sim\) can be preempted by \(u_2\)
- from \((\ell_2, \star)\), play \((1 - \star, c_2)\)
- from \((\ell_3, 1)\), play \((0, c_3)\)
An example of a timed game

Rule of the game
- **Aim:** avoid 😞 and reach 😊
- **How do we play?** According to a strategy:

  \[ f : \text{history} \mapsto (\text{delay, cont. transition}) \]

A (memoryless) winning strategy
- from \((\ell_0, 0)\), play \((0.5, c_1)\)
  \(\sim\) can be preempted by \(u_2\)
- from \((\ell_2, \star)\), play \((1 - \star, c_2)\)
- from \((\ell_3, 1)\), play \((0, c_3)\)
- from \((\ell_1, 1)\), play \((1, c_4)\)
An example of a timed game

Rule of the game

- **Aim:** avoid ☹ and reach 😊
- **How do we play?** According to a strategy:

  \[ f : \text{history} \mapsto (\text{delay, cont. transition}) \]

Problems to be considered
An example of a timed game

Rule of the game

- **Aim**: avoid 😞 and reach 😊
- **How do we play?** According to a strategy:

  \[ f : \text{history} \mapsto (\text{delay}, \text{cont. transition}) \]

Problems to be considered

- Does there exist a winning strategy?
An example of a timed game

Rule of the game

- **Aim:** avoid 😞 and reach 😊
- **How do we play?** According to a strategy:

  \[ f : \text{history} \mapsto (\text{delay, cont. transition}) \]

Problems to be considered

- Does there exist a winning strategy?
- If yes, compute one (as simple as possible).
Decidability of timed games

Theorem [AMPS98, HK99]

Reachability and safety timed games are decidable and EXPTIME-complete. Furthermore memoryless and “region-based” strategies are sufficient.


Decidability of timed games

**Theorem [AMPS98,HK99]**

Reachability and safety timed games are decidable and EXPTIME-complete. Furthermore memoryless and “region-based” strategies are sufficient.

\[ \leadsto \text{classical regions are sufficient for solving such problems} \]
Decidability of timed games

Theorem [AMPS98,HK99]

Reachability and safety timed games are decidable and EXPTIME-complete. Furthermore memoryless and “region-based” strategies are sufficient.

∽ classical regions are sufficient for solving such problems

Theorem [AM99,BHPR07,JT07]

Optimal-time reachability timed games are decidable and EXPTIME-complete.

Back to the example: computing winning states
Back to the example: computing winning states

\[ (x \leq 2) \]

\[ x \geq 1, u_3 \]

\[ x \leq 1, c_1 \]

\[ x < 1, u_1 \]

\[ x < 1, u_2, x := 0 \]

\[ x \geq 2, c_4 \]

\[ x \leq 1, c_3 \]

\[ c_2 \]
Back to the example: computing winning states

- \( x \leq 2 \)
- \( x \geq 1, u_3 \)
- \( x \leq 1, c_1 \)
- \( x \geq 2, c_4 \)
- \( x < 1, u_1 \)
- \( x < 1, u_2, x := 0 \)
- \( x \leq 1, c_3 \)
- \( c_2 \)
- \( \ell_0 \)
- \( \ell_1 \)
- \( \ell_2 \)
- \( \ell_3 \)
Back to the example: computing winning states

\[ \begin{align*}
&\ell_0: x \leq 2, x<1, u_1, x=0, u_2 \\
&\ell_1: x \leq 1, c_1, x \geq 2, c_4 \\
&\ell_2: x \leq 1, c_3 \\
&\ell_3: c_2
\end{align*} \]
Back to the example: computing winning states

\begin{equation}
\begin{aligned}
\ell_0 &\xrightarrow{x \leq 1, c_1} \ell_1 \\
&\xrightarrow{x \geq 1, u_3} \ell_0 \\
\ell_1 &\xrightarrow{x < 1, u_1} \ell_2 \\
&\xrightarrow{x \geq 2, c_4} \ell_1 \\
\ell_2 &\xrightarrow{c_2} \ell_3 \\
\ell_3 &
\end{aligned}
\end{equation}

\begin{itemize}
\item \(x \leq 2\)
\item \(x < 1, u_2, x := 0\)
\item \(x < 1, u_1\)
\item \(x \leq 1, c_3\)
\end{itemize}

\text{Winning states:} \(\ell_0, \ell_1, \ell_3\)

\text{Losing states:} \(\ell_2\)
Back to the example: computing winning states

\( \ell_0 \)

\( x \leq 2 \)

\( x > 1, u_3 \)

\( x \leq 1, c_1 \)

\( x < 1, u_1 \)

\( x < 1, u_2, x := 0 \)

\( x \leq 1, c_3 \)

\( c_2 \)

\( \ell_1 \)

\( x > 2, c_4 \)

\( \ell_2 \)

\( \ell_3 \)
Back to the example: computing winning states
Back to the example: computing winning states

Winning states

Losing states

\begin{align*}
\ell_0 & : (x \leq 2) \\
\ell_1 & : x \leq 1, c_1 \\
\ell_2 & : x < 1, u_1 \\
\ell_3 & : c_2
\end{align*}
Decidability via attractors
Decidability via attractors

\[ \text{Pred}^a(X) = \{ \bullet \mid \bullet \xrightarrow{a} \bullet \in X \} \]
Decidability via attractors

- $\text{Pred}^a(X) = \{\bullet | \bullet \overset{a}{\rightarrow} \bullet \in X\}$

- Controllable and uncontrollable discrete predecessors:

\[
\begin{align*}
\text{cPred}(X) &= \bigcup_{a \text{ cont.}} \text{Pred}^a(X) \\
\text{uPred}(X) &= \bigcup_{a \text{ uncont.}} \text{Pred}^a(X)
\end{align*}
\]
Decidability via attractors

- \( \text{Pred}^a(X) = \{ \bullet \mid \bullet \xrightarrow{a} \bullet \in X \} \)

- controllable and uncontrollable discrete predecessors:

\[
\text{cPred}(X) = \bigcup_{a \text{ cont.}} \text{Pred}^a(X) \quad \text{uPred}(X) = \bigcup_{a \text{ uncont.}} \text{Pred}^a(X)
\]

- time controllable predecessors:

\[
\text{Pred}_\delta(X, \text{Safe}) = \{ \bullet \mid \exists t \geq 0, \bullet \xrightarrow{\delta(t)} \text{Safe} \}\quad \text{and} \quad \forall 0 \leq t' \leq t, \bullet \xrightarrow{\delta(t')} \text{Safe}
\]

\[
\text{delay } t \text{ t.u.} \quad \Rightarrow \text{should be safe}
\]
Decidability via attractors

- \( \text{Pred}^a(X) = \{ \bullet \mid \bullet \xrightarrow{a} \bullet \in X \} \)
- Controllable and uncontrollable discrete predecessors:

\[
\text{cPred}(X) = \bigcup_{a \text{ cont.}} \text{Pred}^a(X) \quad \text{and} \quad \text{uPred}(X) = \bigcup_{a \text{ uncont.}} \text{Pred}^a(X)
\]

- Time controllable predecessors:

\[
\text{Pred}_\delta(X, \text{Safe}) = \{ \bullet \mid \exists t \geq 0, \bullet \xrightarrow{\delta(t)} \bullet \}
\]

and \( \forall 0 \leq t' \leq t, \bullet \xrightarrow{\delta(t')} \bullet \in \text{Safe} \)
Timed games with a reachability objective

We write:

$$\pi(X) = X \cup \text{Pred}_\delta(\text{cPred}(X), \neg \text{uPred}(\neg X))$$
Timed games with a reachability objective

We write:

$$\pi(X) = X \cup \text{Pred}_\delta(\text{cPred}(X), \neg\text{uPred}(\neg X))$$

- The states from which one can ensure 😊 in no more than 1 step is:

$$\text{Attr}_1(😊) = \pi(😊)$$
Timed games with a reachability objective

We write:

\[ \pi(X) = X \cup \text{Pred}_\delta(c\text{Pred}(X), \neg\text{uPred}(\neg X)) \]

- The states from which one can ensure ☺ in no more than 1 step is:

  \[ \text{Attr}_1(☺) = \pi(☺) \]

- The states from which one can ensure ☺ in no more than 2 steps is:

  \[ \text{Attr}_2(☺) = \pi(\text{Attr}_1(☺)) \]
Timed games with a reachability objective

We write:

\[ \pi(X) = X \cup \text{Pred}_\delta(\text{cPred}(X), \neg \text{uPred}(\neg X)) \]

- The states from which one can ensure ☹ in no more than 1 step is:

  \[ \text{Attr}_1(☺) = \pi(☺) \]

- The states from which one can ensure ☹ in no more than 2 steps is:

  \[ \text{Attr}_2(☺) = \pi(\text{Attr}_1(☺)) \]

- ...
Timed games with a reachability objective

We write:

\[ \pi(X) = X \cup \text{Pred}_\delta(\text{cPred}(X), \neg \text{uPred}(\neg X)) \]

- The states from which one can ensure \( \square \) in no more than 1 step is:

\[ \text{Attr}_1(\square) = \pi(\square) \]

- The states from which one can ensure \( \square \) in no more than 2 steps is:

\[ \text{Attr}_2(\square) = \pi(\text{Attr}_1(\square)) \]

- \ldots

- The states from which one can ensure \( \square \) in no more than \( n \) steps is:

\[ \text{Attr}_n(\square) = \pi(\text{Attr}_{n-1}(\square)) \]
Timed games with a reachability objective

We write:

$$\pi(X) = X \cup \text{Pred}_\delta(c\text{Pred}(X), \neg u\text{Pred}(\neg X))$$

- The states from which one can ensure 😊 in no more than 1 step is:
  
  $$\text{Attr}_1(😊) = \pi(😊)$$

- The states from which one can ensure 😊 in no more than 2 steps is:

  $$\text{Attr}_2(😊) = \pi(\text{Attr}_1(😊))$$

- ...  

- The states from which one can ensure 😊 in no more than $n$ steps is:

  $$\text{Attr}_n(😊) = \pi(\text{Attr}_{n-1}(😊)) = \pi^n(😊)$$
Stability w.r.t. regions

- if $X$ is a union of regions, then:
  - $\text{Pred}_a(X)$ is a union of regions,
  - and so are $\text{cPred}(X)$ and $\text{uPred}(X)$.
Stability w.r.t. regions

- if $X$ is a union of regions, then:
  - $\text{Pred}_a(X)$ is a union of regions,
  - and so are $\text{cPred}(X)$ and $\text{uPred}(X)$.
- Does $\pi$ also preserve unions of regions?
Stability w.r.t. regions

- if $X$ is a union of regions, then:
  - $\text{Pred}_a(X)$ is a union of regions,
  - and so are $\text{cPred}(X)$ and $\text{uPred}(X)$.
- Does $\pi$ also preserve unions of regions?
Stability w.r.t. regions

- if $X$ is a union of regions, then:
  - $\text{Pred}_a(X)$ is a union of regions,
  - and so are $\text{cPred}(X)$ and $\text{uPred}(X)$.
- Does $\pi$ also preserve unions of regions?
Stability w.r.t. regions

- if $X$ is a union of regions, then:
  - $\text{Pred}_a(X)$ is a union of regions,
  - and so are $\text{cPred}(X)$ and $\text{uPred}(X)$.

- Does $\pi$ also preserve unions of regions?
Stability w.r.t. regions

- if \( X \) is a union of regions, then:
  - \( \text{Pred}_a(X) \) is a union of regions,
  - and so are \( \text{cPred}(X) \) and \( \text{uPred}(X) \).
- Does \( \pi \) also preserve unions of regions? Yes!

\[
\begin{align*}
\text{cPred}(X) \\
\text{uPred}(\neg X) \\
\text{Pred}_\delta(\text{cPred}(X), \neg \text{uPred}(\neg X))
\end{align*}
\]
Stability w.r.t. regions

- if $X$ is a union of regions, then:
  - $\text{Pred}_a(X)$ is a union of regions,
  - and so are $\text{cPred}(X)$ and $\text{uPred}(X)$.

- Does $\pi$ also preserve unions of regions? Yes!

(cPred($X$)

\text{uPred}(\neg X)

\text{Pred}_\delta(\text{cPred}(X), \neg \text{uPred}(\neg X))

(but it generates non-convex unions of regions...)
Stability w.r.t. regions

- if $X$ is a union of regions, then:
  - $\text{Pred}_a(X)$ is a union of regions,
  - and so are $c\text{Pred}(X)$ and $u\text{Pred}(X)$.
- Does $\pi$ also preserve unions of regions? Yes!

$$c\text{Pred}(X) \quad u\text{Pred}(\neg X) \quad \text{Pred}_\delta(c\text{Pred}(X), \neg u\text{Pred}(\neg X))$$

(but it generates non-convex unions of regions...)

$\leadsto$ the computation of $\pi^*$ terminates!
Stability w.r.t. regions

- if $X$ is a union of regions, then:
  - $\text{Pred}_a(X)$ is a union of regions,
  - and so are $\text{cPred}(X)$ and $\text{uPred}(X)$.

- Does $\pi$ also preserve unions of regions? Yes!

\[ \text{cPred}(X) \]
\[ \text{uPred}(\neg X) \]
\[ \text{Pred}_\delta(\text{cPred}(X), \neg \text{uPred}(\neg X)) \]

(but it generates non-convex unions of regions...)

$\mapsto$ the computation of $\pi^*$(🙂) terminates!
... and is correct
Timed games with a safety objective

- We can use operator \( \tilde{\pi} \) defined by

\[
\tilde{\pi}(X) = \text{Pred}_\delta(X \cap \text{cPred}(X), \neg \text{uPred}(\neg X))
\]

instead of \( \pi \), and compute \( \tilde{\pi}^*(\neg \ominus) \).
Timed games with a safety objective

- We can use operator $\tilde{\pi}$ defined by

$$\tilde{\pi}(X) = \text{Pred}_\delta(X \cap c\text{Pred}(X), \neg u\text{Pred}(\neg X))$$

instead of $\pi$, and compute $\tilde{\pi}^*(\neg \ominus)$

- It is also stable w.r.t. regions.
Some remarks

The model

Our games are control games,
Some remarks

The model

Our games are control games, and in particular they:

- are asymmetric
  - the environment can preempt any decision of the controller
  - we take the point-of-view of the controller
- are neither concurrent nor turn-based
- do not take into account Zenoness considerations
  ~ can be done adding a Büchi winning condition

The model

Our games are control games, and in particular they:

- are asymmetric
  - the environment can preempt any decision of the controller
  - we take the point-of-view of the controller
- are neither concurrent nor turn-based
- do not take into account Zenoness considerations
  ~ can be done adding a Büchi winning condition
Some remarks

The model

Our games are control games, and in particular they:
- are asymmetric
  - the environment can preempt any decision of the controller
  - we take the point-of-view of the controller
- are neither concurrent nor turn-based
- do not take into account Zenoness considerations
  \[\sim\]
  can be done adding a Büchi winning condition

Implementation

Uppaal-Tiga implements a forward algorithm to compute winning states and winning strategies \[\text{CDF}+05,\text{BCD}+07\]

Application of timed games to strong timed bisimulation

This is a relation between ● and ● such that:
Application of timed games to strong timed bisimulation

This is a relation between $\bullet$ and $\bullet$ such that:

$$\forall \quad \begin{array}{c} \bullet \\ \cdots \\ \bullet \end{array} \quad a \quad \begin{array}{c} \bullet \\ \cdots \\ \bullet \end{array}$$

Theorem
Strong timed (bi)simulation between timed automata is decidable and EXPTIME-complete.
Application of timed games to strong timed bisimulation

This is a relation between $\bullet$ and $\bullet$ such that:

$\forall \quad \exists$

$\Rightarrow \quad a$

$\Rightarrow \quad a$

Theorem

Strong timed (bi)simulation between timed automata is decidable and EXPTIME-complete.
Application of timed games to strong timed bisimulation

This is a relation between ● and ● such that:

∀                   a                    ∀d ≥ 0

∃                   a

δ(d)

∀d ≥ 0
Application of timed games to strong timed bisimulation

This is a relation between ● and ● such that:

\[ \forall \quad \exists \]

\[ a \quad \delta(d) \]

\[ \forall d \geq 0 \quad \exists \]

\[ a \quad \delta(d) \]
Application of timed games to strong timed bisimulation

This is a relation between $\bullet$ and $\bullet$ such that:

\[
\begin{align*}
\forall & \quad \bullet & \xrightarrow{a} & \bullet \\
\exists & \quad \bullet & \xleftarrow{a} & \bullet \\
\forall d \geq 0 & \quad \bullet & \xrightarrow{\delta(d)} & \bullet \\
\exists & \quad \bullet & \xleftarrow{\delta(d)} & \bullet
\end{align*}
\]

... and vice-versa (swap $\bullet$ and $\bullet$) for the bisimulation relation.

Theorem
Strong timed (bi)simulation between timed automata is decidable and EXPTIME-complete.
Application of timed games to strong timed bisimulation

This is a relation between $\bullet$ and $\bullet$ such that:

$$\forall \quad \exists$$

... and vice-versa (swap $\bullet$ and $\bullet$) for the bisimulation relation.

**Theorem**

Strong timed (bi)simulation between timed automata is decidable and EXPTIME-complete.
Timed games: WTG

\[ p, q, p', q' \]

\[ g_1, a, Y_1 := 0 \]

\[ g_2, a, Y_2 := 0 \]

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

\[ g_1 \land (z = 0), a, Y_1 := 0 \]

\[ g_2 \land (z = 0), a, Y_2 := 0 \]

\[ g \land (z = 0), a, Y := 0 \]

\[ !g \land (z = 0) \]
**Timed automaton $A$**

- Node $p$ transitions to $p'$ with $g_1, a, Y_1 := 0$
- Node $p$ transitions to $p'_2$ with $g_2, a, Y_2 := 0$

**Timed automaton $B$**

- Node $q$ transitions to $q'$ with $g, a, Y := 0$

---

**Tester**

- Node $p, q$
timed automaton \( A \)

- \( p \) to \( p'_1 \) via \( g_1, a, Y_1 := 0 \)
- \( g_2, a, Y_2 := 0 \) from \( p' \)

\[
g_1, a, Y_1 := 0
\]

\[
g_2, a, Y_2 := 0
\]

\[
tester\]

\[
prover
\]

timed automaton \( B \)

- \( q \) to \( q' \) via \( g, a, Y := 0 \)

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

\[
(p,q)
\]

\[
(z=0)
\]

\[
(z=0)
\]

\[
(z=0)
\]

\[
(z=0)
\]
Introduction

Decidability

Implementation

Other problems

Timed control

Conclusion

Timed games

WTG

\[ p, q \]

\[ p \rightarrow g_1, a, Y_1 := 0 \]

\[ p_1' \]

\[ g_2, a, Y_2 := 0 \]

\[ p_2' \]

\[ \text{timed automaton } A \]

\[ q \rightarrow g, a, Y := 0 \]

\[ q' \]

\[ \text{timed automaton } B \]

\[ (z = 0) \]

\[ (z = 0) \]

\[ (z = 0) \]

\[ (z = 0) \]

\[ (z = 0) \]

\[ p, q, Y := 0, z := 0 \]

\[ \text{tester} \]

\[ \text{prover} \]
Timed games WTG

\[ p, q, Y_1 := 0, z := 0 \quad \Rightarrow \quad p_1, q' \quad \text{prover} \]

\[ p, q, Y_2 := 0, z := 0 \quad \Rightarrow \quad p_2, q' \quad \text{prover} \]

\[ p, a, Y_1 := 0, z := 0 \quad \Rightarrow \quad p_1, q' \quad \text{prover} \]

\[ p, a, Y_2 := 0, z := 0 \quad \Rightarrow \quad p_2, q' \quad \text{prover} \]

\[ g, a, Y_1 := 0 \quad \Rightarrow \quad q' \quad \text{tester} \]

\[ g, a, Y_2 := 0 \quad \Rightarrow \quad q' \quad \text{tester} \]

\[ g_1, a, Y_1 := 0 \quad \Rightarrow \quad p_1 \quad \text{prover} \]

\[ g_2, a, Y_2 := 0 \quad \Rightarrow \quad p_2 \quad \text{prover} \]

\[ g_1 \land (z = 0), a, Y_1 \quad \Rightarrow \quad p_1, q' \quad \text{prover} \]

\[ g_2 \land (z = 0), a, Y_2 \quad \Rightarrow \quad p_2, q' \quad \text{prover} \]

\[ g_1 \land (z = 0), a, Y \quad \Rightarrow \quad p_1, q' \quad \text{prover} \]

\[ g_2 \land (z = 0), a, Y \quad \Rightarrow \quad p_2, q' \quad \text{prover} \]

\[ (z = 0) \land \neg g, a \quad \Rightarrow \quad \text{responder} \]
\( \mathcal{A} \) and \( \mathcal{B} \) are strongly timed bisimilar iff the prover \( \bigcirc \) has a winning strategy to avoid \( \frown \).
Outline

1 Introduction
   • Timed automata
   • Examples

2 Decidability of basic properties
   • The region abstraction
   • Extensions of timed automata
   • Weighted timed automata

3 Implementation and tools

4 Other verification problems
   • Equivalence (or preorder) checking
   • Verification of timed temporal logics (short)

5 Timed control
   • Timed games
   • Weighted timed games

6 Conclusion
A simple weighted timed game

\[\begin{align*}
\ell_0 & \xrightarrow{x \leq 2, c, y := 0} \ell_1 \\
\ell_1 & \xrightarrow{(y = 0)} \ell_2 \\
\ell_2 & \xrightarrow{x = 2, c} \ell_3 \\
\ell_3 & \xrightarrow{x = 2, c} \text{goal}
\end{align*}\]

Question: What is the optimal cost we can ensure while reaching the goal?

\[
\inf_{0 \leq t \leq 2} \max \left( 5t + 10(2-t) + 1, 5t + (2-t) + 7 \right) = 14 + \frac{1}{3};
\]

Strategy: Wait in \(\ell_0\), and when \(t = \frac{93}{100}\), go to \(\ell_1\).
A simple weighted timed game

\[
\begin{align*}
\text{\(l_0\)} & \quad +5 \\
\text{\(l_1\)} & \quad (y=0) \\
\text{\(l_2\)} & \quad +10 \\
\text{\(l_3\)} & \quad +1 \\
\end{align*}
\]

\[
x \leq 2, c, y := 0
\]

\[
x = 2, c
\]

\[
x = 2, c
\]

\[
\inf_{0 \leq t \leq 2} \max (5t + 10(2-t) + 1, 5t + (2-t) + 7) = 14 + \frac{1}{3};
\]

strategy: wait in \(l_0\), and when \(t = \frac{4}{3}\), go to \(l_1\)
A simple weighted timed game

Question: what is the optimal cost we can ensure while reaching 😊?
A simple weighted timed game

Question: what is the optimal cost we can ensure while reaching 😊?

$$5t + 10(2 - t) + 1$$
A simple weighted timed game

Question: what is the optimal cost we can ensure while reaching 😊?

\[ 5t + 10(2 - t) + 1, \quad 5t + (2 - t) + 7 \]
A simple weighted timed game

Question: what is the optimal cost we can ensure while reaching 😊?

\[
\max\left(5t + 10(2 - t) + 1, 5t + (2 - t) + 7\right)
\]
A simple weighted timed game

Question: what is the optimal cost we can ensure while reaching 😊?

$$\inf_{0 \leq t \leq 2} \max \left( 5t + 10(2 - t) + 1, 5t + (2 - t) + 7 \right) = 14 + \frac{1}{3}$$
A simple weighted timed game

Question: what is the optimal cost we can ensure while reaching 😊?

\[
\inf_{0 \leq t \leq 2} \max \left( 5t + 10(2 - t) + 1, 5t + (2 - t) + 7 \right) = 14 + \frac{1}{3}
\]

~ strategy: wait in \( \ell_0 \), and when \( t = \frac{4}{3} \), go to \( \ell_1 \)
Optimal reachability in weighted timed games

This topic has been fairly hot these last ten years...

[LMM02,ABM04,BCFL04,BBR05,BBM06,BLMR06,Rut11,HIM13,BGK+14]

[BBR05] Brihaye, Bruyère, Raskin. On optimal timed strategies (FORMATS’05).
[BBM06] Bouyer, Brihaye, Markey. Improved undecidability results on weighted timed automata (Information Processing Letters).
[BGK+14] Brihaye, Geeraerts, Krishna, Manasa, Monmege, Trivedi. Adding Negative Prices to Priced Timed Games (CONCUR’14).
Optimal reachability in weighted timed games

This topic has been fairly hot these last ten years...

[La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata (TCS@02).
Alur, Bernardsky, Madhusudan. Optimal reachability in weighted timed games (ICALP’04).
Bouyer, Cassez, Fleury, Larsen. Optimal strategies in priced timed game automata (FSTTCS’04).
Brihaye, Bruyère, Raskin. On optimal timed strategies (FORMATS’05).
Bouyer, Brihaye, Markey. Improved undecidability results on weighted timed automata (Information Processing Letters).
Bouyer, Larsen, Markey, Rasmussen. Almost-optimal strategies in one-clock priced timed automata (FSTTCS’06).
Rutkowski. Two-player reachability-price games on single-clock timed automata (QAPL’11).
Hansen, Ibsen-Jensen, Miltersen. A faster algorithm for solving one-clock priced timed games (CONCUR’13).
Brihaye, Geeraerts, Krishna, Manasa, Monmege, Trivedi. Adding Negative Prices to Priced Timed Games (CONCUR’14).]

**Theorem** [Brihaye, Bruyère, Raskin. On optimal timed strategies (FORMATS’05).
Bouyer, Brihaye, Markey. Improved undecidability results on weighted timed automata (Information Processing Letters).
Bouyer, Larsen, Markey, Rasmussen. Almost-optimal strategies in one-clock priced timed automata (FSTTCS’06).
Rutkowski. Two-player reachability-price games on single-clock timed automata (QAPL’11).
Hansen, Ibsen-Jensen, Miltersen. A faster algorithm for solving one-clock priced timed games (CONCUR’13).
Brihaye, Geeraerts, Krishna, Manasa, Monmege, Trivedi. Adding Negative Prices to Priced Timed Games (CONCUR’14).]

Optimal timed games are **undecidable**, as soon as automata have three clocks or more.
Optimal reachability in weighted timed games

This topic has been fairly hot these last ten years...

[LMM02, ABM04, BCFL04, BBR05, BBM06, BLMR06, Rut11, HIM13, BGK+14]

**Theorem** [BBR05, BBM06, recent work]

Optimal timed games are **undecidable**, as soon as automata have three clocks or more.

**Theorem** [BLMR06, Rut11, HIM13, BGK+14]

Turn-based optimal timed games are **decidable** in EXPTIME (resp. PTIME) when automata have a single clock (with two rates). They are PTIME-hard.

[LMM02] La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata (TCS@02).
[BBR05] Brihaye, Bruyère, Raskin. On optimal timed strategies (FORMATS’05).
[BBM06] Bouyer, Brihaye, Markey. Improved undecidability results on weighted timed automata (Information Processing Letters).
[BGK+14] Brihaye, Geeraerts, Krishna, Manasa, Monmege, Trivedi. Adding Negative Prices to Priced Timed Games (CONCUR’14).
The positive side (one-clock case)

- Key: resetting the clock somehow resets the history...
The positive side (one-clock case)

- Key: resetting the clock somehow resets the history...
- Memoryless strategies can be non-optimal...

\[
\begin{align*}
\ell_0 & \quad x \leq 1 \\
\ell_1 & \quad x < 1, x := 0 \\
\ell_1 & \quad +1 \\
\end{align*}
\[
\begin{align*}
\ell_0 & \quad +2 \\
\ell_0 & \quad x = 1 \\
\end{align*}
\[
\begin{align*}
\ell_1 & \quad x > 0
\end{align*}
\]
The positive side (one-clock case)

- Key: resetting the clock somehow resets the history...
- Memoryless strategies can be non-optimal...

![Diagram](image)

- However, by unfolding and removing one by one the locations, we can synthesize memoryless almost-optimal winning strategies.
The positive side (one-clock case)

- Key: resetting the clock somehow resets the history...
- Memoryless strategies can be non-optimal...

However, by unfolding and removing one by one the locations, we can synthesize memoryless almost-optimal winning strategies.

Rather involved proof of correctness for a simple algorithm.
The negative side: why is that hard?

Given two clocks $x$ and $y$, we can check whether $y = 2x$. 
The negative side: why is that hard?

Given two clocks $x$ and $y$, we can check whether $y = 2x$.

The cost is increased by $x_0$.

The cost is increased by $1 - x_0$. 

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

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

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

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

\[ y = 1, y := 0 \]
\[ x = 1, x := 0 \]
\[ z = 0, z := 0 \]
The negative side: why is that hard?

Given two clocks $x$ and $y$, we can check whether $y = 2x$. 

![Diagram](image-url)
The negative side: why is that hard?

Given two clocks $x$ and $y$, we can check whether $y = 2x$.

- In $\smile$, cost = $2x_0 + (1 - y_0) + 2$
The negative side: why is that hard?

Given two clocks $x$ and $y$, we can check whether $y = 2x$. 

In $\circlearrowleft$, cost $= 2x_0 + (1 - y_0) + 2$

In $\mathbf{☺}$, cost $= 2(1 - x_0) + y_0 + 1$
The negative side: why is that hard?

Given two clocks $x$ and $y$, we can check whether $y = 2x$.

- In $\square$, cost $= 2x_0 + (1 - y_0) + 2$
- In $\square$, cost $= 2(1 - x_0) + y_0 + 1$
- if $y_0 < 2x_0$, player 2 chooses the first branch: cost $> 3$
The negative side: why is that hard?

Given two clocks $x$ and $y$, we can check whether $y = 2x$.

\[
\begin{align*}
\text{In } &\bigcirc, \text{ cost } = 2x_0 + (1 - y_0) + 2 \\
\text{In } &\bigcirc, \text{ cost } = 2(1 - x_0) + y_0 + 1 \\
\text{if } &y_0 < 2x_0, \text{ player 2 chooses the first branch: cost } > 3 \\
\text{if } &y_0 > 2x_0, \text{ player 2 chooses the second branch: cost } > 3
\end{align*}
\]
The negative side: why is that hard?

Given two clocks $x$ and $y$, we can check whether $y = 2x$.

- In 😊, cost = $2x_0 + (1 - y_0) + 2$
- In 😎, cost = $2(1 - x_0) + y_0 + 1$

- If $y_0 < 2x_0$, player 2 chooses the first branch: cost $> 3$
- If $y_0 > 2x_0$, player 2 chooses the second branch: cost $> 3$
- If $y_0 = 2x_0$, in both branches, cost = 3
The negative side: why is that hard?

Given two clocks $x$ and $y$, we can check whether $y = 2x$.

- In $\bigcirc$, cost = $2x_0 + (1 - y_0) + 2$
- In $\bigcirc$, cost = $2(1 - x_0) + y_0 + 1$

- if $y_0 < 2x_0$, player 2 chooses the first branch: cost $> 3$
  - if $y_0 > 2x_0$, player 2 chooses the second branch: cost $> 3$
  - if $y_0 = 2x_0$, in both branches, cost = 3

- Player 1 has a winning strategy with cost $\leq 3$ iff $y_0 = 2x_0$
The negative side: why is that hard?

Player 1 will simulate a two-counter machine:
- each instruction is encoded as a module;
- the counter values $c_1$ and $c_2$ are encoded by two clocks:

\[
x = \frac{1}{2^c_1} \quad \text{and} \quad y = \frac{1}{3^c_2}
\]
The negative side: why is that hard?

Player 1 will simulate a two-counter machine:
- each instruction is encoded as a module;
- the counter values $c_1$ and $c_2$ are encoded by two clocks:

$$x = \frac{1}{2c_1} \quad \text{and} \quad y = \frac{1}{3c_2}$$

The two-counter machine has an halting computation iff player 1 has a winning strategy to ensure a cost no more than 3.
The negative side: why is that hard?

Player 1 will simulate a two-counter machine:
- each instruction is encoded as a module;
- the counter values $c_1$ and $c_2$ are encoded by two clocks:

$$x = \frac{1}{2c_1} \quad \text{and} \quad y = \frac{1}{3c_2}$$

The two-counter machine has an halting computation iff player 1 has a winning strategy to ensure a cost no more than 3.

Globally, $(x \leq 1, y \leq 1, u \leq 1)$

Test $y(x=2z)$

$$\begin{cases} x=1, x:=0 \\
\vee \ y=1, y:=0 \end{cases} \quad \begin{cases} x=1, x:=0 \\
\vee \ y=1, y:=0 \end{cases}$$

$$\begin{pmatrix} x=\frac{1}{2c} \\
y=\frac{1}{3d} \\
z=\star \end{pmatrix} \quad \begin{pmatrix} x=\frac{1}{2c} \\
y=\frac{1}{3d} \\
z=\alpha \end{pmatrix}$$

$u:=0$ $z:=0$ $u=1, u:=0$ $(u=0)$
Outline

1 Introduction
   • Timed automata
   • Examples

2 Decidability of basic properties
   • The region abstraction
   • Extensions of timed automata
   • Weighted timed automata

3 Implementation and tools

4 Other verification problems
   • Equivalence (or preorder) checking
   • Verification of timed temporal logics (short)

5 Timed control
   • Timed games
   • Weighted timed games

6 Conclusion
Conclusion

- The model of **timed automata**:
Conclusion

- The model of timed automata:
  - Some nice properties (decidability of many structural properties, symbolic algorithms, ...)

Some nice properties (decidability of many structural properties, symbolic algorithms, ...)

Not all good properties though... (e.g. inclusion undecidable)
Conclusion

- The model of timed automata:
  - Some nice properties (decidability of many structural properties, symbolic algorithms, ...)
  - Not all good properties though... (e.g. inclusion undecidable)
Conclusion

- The model of timed automata:
  - ☑ Some nice properties (decidability of many structural properties, symbolic algorithms, …)
  - ☹ Not all good properties though…
    (e.g. inclusion undecidable)
  - ☑ Sucessfully used!!
Conclusion

- The model of **timed automata**:
  - Some nice properties (decidability of many structural properties, symbolic algorithms, ...)
  - Not all good properties though... (e.g. inclusion undecidable)
  - Successfully used!!

- Many extensions have been studied, which allows more accurate modelling of real systems:
  - Weighted timed automata
  - Timed games
  - Probabilistic/stochastic timed automata
  - Alternating timed automata
  - Hybrid automata
  - ...
Conclusion

- The model of **timed automata:**
  - Some nice properties (decidability of many structural properties, symbolic algorithms, ...)
  - Not all good properties though... (e.g. inclusion undecidable)
  - Successfully used!!

- Many extensions have been studied, which allows more accurate modelling of real systems:
  - Weighted timed automata
  - Timed games
  - Probabilistic/stochastic timed automata
  - Alternating timed automata
  - Hybrid automata
  - ...

- Going further in the use of timed automata in verification...
Conclusion

- The model of timed automata:
  - Some nice properties (decidability of many structural properties, symbolic algorithms, ...)
  - Not all good properties though... (e.g. inclusion undecidable)
  - Successfully used!!

- Many extensions have been studied, which allows more accurate modelling of real systems:
  - Weighted timed automata
  - Timed games
  - Probabilistic/stochastic timed automata
  - Alternating timed automata
  - Hybrid automata
  - ...

- Going further in the use of timed automata in verification... 
  ... requires to think about the accurateness of the (mathematical) model we analyze w.r.t. the real-world system
Just a taste...

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

\[ x \leq 2, \quad x := 0 \]

\[ y \geq 2, \quad y := 0 \]
elect...
Just a taste...

\[ x = 1, \quad y = 0, \quad x \leq 2, \quad x := 0 \]

\[ y \geq 2, \quad y := 0 \]

Value of clock is converging, even though global time diverges.

Can we implement such a strategy?

100/100
Just a taste...

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

\[ x \leq 2, \ x := 0 \]

\[ y \geq 2, \ y := 0 \]

\[ y \]

\[ 0 \quad 1 \quad 2 \]

\[ 0 \quad 1 \quad 2 \]

The value of clock \( x \) when hitting is converging, even though global time diverges.
Just a taste...

$x = 1$, $y := 0$

$x \leq 2$, $x := 0$

$y \geq 2$, $y := 0$

Value of clock $x$ when hitting is converging, even though global time diverges.

Can we implement such a strategy??

Lecture of Pierre-Alain tomorrow afternoon!
Just a taste...

\[ x = 1, y = 0 \]
\[ x \leq 2, x := 0 \]
\[ y \geq 2, y := 0 \]

Value of clock is converging, even though global time diverges.

Can we implement such a strategy??

Lecture of Pierre-Alain tomorrow afternoon!
Just a taste...

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

\[ x \leq 2, \quad x := 0 \]

\[ y \geq 2, \quad y := 0 \]

Value of clock \( x \) when hitting is converging, even though global time diverges.

Can we implement such a strategy??
Just a taste...

\[ x = 1, \ y := 0 \]
\[ x \leq 2, \ x := 0 \]
\[ y \geq 2, \ y := 0 \]

Value of clock is converging, even though global time diverges.

Can we implement such a strategy??
Just a taste...

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

\[ x \leq 2, \quad x := 0 \]

\[ y \geq 2, \quad y := 0 \]

Value of clock is converging, even though global time diverges.

Can we implement such a strategy??

Lecture of Pierre-Alain tomorrow afternoon!
Just a taste...

$x = 1, y := 0$

$x \leq 2, x := 0$

$y \geq 2, y := 0$

Value of clock $x_1$ when hitting $y_0$ is converging, even though global time diverges.

Can we implement such a strategy??

lecture of Pierre-Alain tomorrow afternoon!
Just a taste...

\[ x = 1, y = 0 \]

\[ x \leq 2, x := 0 \]

\[ y \geq 2, y := 0 \]

Value of clock is converging, even though global time diverges.

Can we implement such a strategy??
Just a taste...

\[
\begin{align*}
  x &= 1, \quad y := 0 \\
  x \leq 2, \quad x := 0 \\
  y \geq 2, \quad y := 0
\end{align*}
\]

Value of clock \( x \) when hitting is converging, even though global time diverges.

Can we implement such a strategy??

Lecture of Pierre-Alain tomorrow afternoon!
Just a taste...

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

\[
x \leq 2, \quad x := 0
\]

\[
y \geq 2, \quad y := 0
\]
Just a taste...

\[ x = 1, y = 0 \]
\[ x \leq 2, x := 0 \]
\[ y \geq 2, y := 0 \]
Just a taste...

\[ x = 1 \]
\[ y = 0 \]
\[ x \leq 2, \ x := 0 \]
\[ y \geq 2, \ y := 0 \]

\[ y \]
\[ x \]
\[ 0 \]
\[ 1 \]
\[ 2 \]
\[ 0 \]
\[ 1 \]
\[ 2 \]

Value of clock $x$ when hitting $\bigcirc$ is converging, even though global time diverges.
Just a taste...

\[ x = 1, \quad y = 0 \]
\[ x \leq 2, \quad x := 0 \]
\[ y \geq 2, \quad y := 0 \]

Value of clock \( x \) when hitting \( \bullet \) is converging, even though global time diverges

Can we implement such a strategy??
Just a taste...

\[ \begin{align*}
  x &= 1 \\
y &= 0
\end{align*} \quad x \leq 2, \quad x := 0 \\
\begin{align*}
  y &\geq 2, \quad y := 0
\end{align*} \]

\[ y_0 \]

Value of clock \( x \) when hitting \( \bigcirc \) is converging, even though global time diverges

Can we implement such a strategy??

lecture of Pierre-Alain tomorrow afternoon!