De l’analyse automatique de systèmes temporisés au contrôle de systèmes dynamiques

Patricia Bouyer-Decitre

LMF, Université Paris-Saclay, CNRS, ENS Paris-Saclay
Time-dependent systems

- We are interested in **timed systems**
Time-dependent systems

- We are interested in timed systems

- ... and in their analysis and control
Model-checking and control

system:

property:
Model-checking and control

**system:**

![System diagram](image)

**property:**

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

System:

Property:

Algorithm:

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

system:

property:

model-checking algorithm

AG(¬B.overfull ∧ ¬B.dried_up)

yes/no
Model-checking and control

system:

property:

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

control/synthesis algorithm
An 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>
</table>

| $\times$ | 3 picoseconds |

<table>
<thead>
<tr>
<th>energy</th>
<th>10 Watt</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>
</table>

| $\times$ | 7 picoseconds |

<table>
<thead>
<tr>
<th>energy</th>
<th>20 Watts</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>

An example: The task graph scheduling problem

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

$P_1$ (fast):

<table>
<thead>
<tr>
<th></th>
<th>time</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></th>
<th>energy</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></th>
<th>time</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></th>
<th>energy</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>

An example: The task graph scheduling problem

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

$P_1$ (fast):

- **time**
  - +: 2 picoseconds
  - ×: 3 picoseconds

- **energy**
  - idle: 10 Watt
  - in use: 90 Watts

$P_2$ (slow):

- **time**
  - +: 5 picoseconds
  - ×: 7 picoseconds

- **energy**
  - idle: 20 Watts
  - in use: 30 Watts

An example: The task graph scheduling problem

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

- **\( P_1 \) (fast):**
  - **time**:
    - + 2 picoseconds
    - \( \times \) 3 picoseconds
  - **energy**:
    - idle 10 Watt
    - in use 90 Watts

- **\( P_2 \) (slow):**
  - **time**:
    - + 5 picoseconds
    - \( \times \) 7 picoseconds
  - **energy**:
    - idle 20 Watts
    - in use 30 Watts

\[ \begin{align*}
D \times (C \times (A+B)) & (A+B) + (C \times D) \\
\text{Sch}_1 & \\
T_1 & T_2 & T_3 & T_4 & T_5 & T_6 \\
P_1 & P_2 \\
\text{Sch}_2 & \\
T_1 & T_2 & T_3 & T_4 & T_5 & T_6 \\
P_1 & P_2 \\
\text{Sch}_3 & \\
T_1 & T_2 & T_3 & T_4 & T_5 & T_6 \\
P_1 & P_2
\end{align*} \]

13 picoseconds
1.37 nanojoules

12 picoseconds
1.39 nanojoules

19 picoseconds
1.32 nanojoules

Outline

1. Timed automata
2. Weighted timed automata
3. Timed games
4. Weighted timed games
5. Tools
6. Towards applying all this theory to robotic systems
7. Conclusion
The model of timed automata

This run reads the timed word: 

\[(\text{problem}, x:=0) \rightarrow (\text{alarm}, y:=0) \rightarrow (\text{repairing}, x\leq 15, y:=0) \rightarrow (\text{fail safe}, 2\leq y\land x\leq 56, y:=0) \rightarrow \cdots \rightarrow (\text{safe}, 15\leq x\leq 16) \rightarrow (\text{done}, 22\leq y\leq 25) \rightarrow \cdots \]

The model of timed automata

The run reads the timed word:

\[(\text{problem, } x:=0)(\text{repair, } x\leq 15 \land y:=0)(\text{delayed, } y:=0)(\text{repair, } 2\leq y \land x\leq 56 \land y:=0)(\text{done, } 22\leq y\leq 25)\]
The model of timed automata

The diagram represents a timed automaton with states labeled as follows:

- **Safe**
- **Alarm**
- **Repairing**
- **Failsafe**

The transitions and conditions are as follows:

- **Safe** to **Alarm**: $x := 0$
- **Safe** to **Repairing**: $y := 0$
- **Safe** to **Failsafe**: $done, 22 \leq y \leq 25$
- **Alarm** to **Repairing**: $15 \leq x \leq 16$
- **Repairing** to **Failsafe**: $2 \leq y \land x \leq 56$
- **Repairing** to **Repairing**: $y := 0$

The table shows the initial values and the transitions:

<table>
<thead>
<tr>
<th>State</th>
<th>Initial Value</th>
<th>Transition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safe</td>
<td>$x = 0$, $y = 0$</td>
<td>$x := 0$</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$y := 0$</td>
<td></td>
</tr>
</tbody>
</table>

This run reads the timed word: 

- $(problem, 23)$
- $(delayed, 38.6)$
- $(repair, 40.9)$
- $(done, 63)$
The model of timed automata

This run reads the timed word (problem, 23)(delayed, 38.6)(repair, 40.9)(done, 63)
The model of timed automata

This run reads the timed word: \( \text{problem}, 23 \rangle \text{delayed}, 38 \rangle \text{repair}, 40 \rangle \text{done} \).
The model of timed automata

\[
\begin{align*}
\text{safe} & \xrightarrow{23} \text{safe} \quad \text{problem, } x:=0 \\
\text{alarm} & \xrightarrow{15.6} \text{alarm} \quad \text{repair, } y:=0 \land 15 \leq x \leq 16 \\
\text{failsafe} & \xrightarrow{15.6} \text{failsafe} \quad \text{delayed, } y:=0 \land 2 \leq y \land x \leq 56
\end{align*}
\]

This run reads the timed word \((\text{problem, } 23)(\text{delayed, } 38.6)(\text{repair, } 40.9)(\text{done, } 63)\)
The model of timed automata

This run reads the timed word \((\text{problem, } x:=0)(\text{alarm, } y:=0)(\text{delayed, } y:=0)(\text{repair, } x\leq 15)(\text{repair, } x\leq 16)(\text{done, } 22\leq y\leq 25)(\text{done, } 22\leq y\leq 25)(\text{failsafe})\)
The model of timed automata

This run reads the timed word:

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

<table>
<thead>
<tr>
<th>State</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>safe</td>
<td>0</td>
</tr>
<tr>
<td>problem</td>
<td>23</td>
</tr>
<tr>
<td>alarm</td>
<td>0</td>
</tr>
<tr>
<td>repair</td>
<td>15.6</td>
</tr>
<tr>
<td>delayed</td>
<td>15.6</td>
</tr>
<tr>
<td>failsafe</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>15.6</td>
</tr>
<tr>
<td>...</td>
<td>17.9</td>
</tr>
<tr>
<td>...</td>
<td>17.9</td>
</tr>
</tbody>
</table>
The model of timed automata

The run reads the timed word

\[
\text{problem, } x := 0 \quad \rightarrow \quad \text{alarm, } y := 0 \quad \rightarrow \quad \text{repair, } x \leq 15 \quad \rightarrow \quad \text{failsafe, } y := 0 \quad \rightarrow \quad \text{done, } 22 \leq y \leq 25
\]

\[
\begin{array}{ccccccc}
\text{safe} & \rightarrow & 23 & \rightarrow & \text{safe} & \rightarrow & \text{problem} & \rightarrow & \text{alarm} & \rightarrow & 15.6 & \rightarrow & \text{alarm} & \rightarrow & \text{delayed} & \rightarrow & \text{failsafe} \\
X & 0 & 23 & 0 & 15.6 & 15.6 & \ldots \\
y & 0 & 23 & 23 & 38.6 & 0 \\
\end{array}
\]
The model of timed automata

This run reads the timed word

\((\text{problem}, 23) (\text{delayed}, 38.6) (\text{repair}, 40) (\text{done}, 63)\)
# The model of timed automata

The model of timed automata is a formalism for specifying and verifying real-time systems. Timed automata are a type of automaton that incorporate the notion of time, allowing the modeling of systems with timing constraints.

## Timed Automata
- **Weighted timed automata**
- **Timed games**
- **Weighted timed games**
- **Tools**
- **Towards further applications**
- **Conclusion**

## The Model of Timed Automata

### Timed Automata

Timed automata are a type of automaton that incorporate the notion of time, allowing the modeling of systems with timing constraints.

### Weighted Timed Automata

Weighted timed automata extend the concept of timed automata by incorporating weights, which can represent, for example, costs or probabilities.

### Timed Games

Timed games are a type of game played on a timed automaton, where players make decisions that affect the evolution of time over the game.

### Weighted Timed Games

Weighted timed games extend the concept of timed games by allowing for the incorporation of weights, which can be used to model, for example, the cost of making a move or the probability of an event occurring.

### Tools

Tools for analyzing and verifying timed systems, such as model checking and simulation, are available to help ensure that the systems meet their timing requirements.

### Towards Further Applications

The model of timed automata has various applications, including real-time systems, communication protocols, and hardware design.

### Conclusion

In conclusion, the model of timed automata provides a powerful framework for specifying and verifying systems with timing constraints, enabling the design of reliable and efficient real-time systems.

---

### Example

#### Problem

- **Safe**
  - \( x := 0 \)
  - \( y := 0 \)

- **Alarm**
  - \( 15 \leq x \leq 16 \)
  - \( 2 \leq y \land x \leq 56 \)

- **Delayed**
  - \( y := 0 \)

- **Failed Safe**
  - \( 15.6 \leq y \leq 25 \)

- **Repairing**
  - \( 22 \leq y \leq 25 \)

- **Done**
  - \( 22 \leq y \leq 25 \)

#### Clock Valuation

<table>
<thead>
<tr>
<th>State</th>
<th>Value</th>
<th>Value</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>
<tr>
<td>( y )</td>
<td>15.6</td>
<td>38.6</td>
</tr>
<tr>
<td>( y )</td>
<td>15.6</td>
<td>0</td>
</tr>
<tr>
<td>( y )</td>
<td>0</td>
<td>22.1</td>
</tr>
</tbody>
</table>

---

### Diagram

The diagram illustrates the transitions between different states of the timed automaton, showing how time evolves and how the system transitions between different modes of operation.

---

---
The model of timed automata

This run reads the timed word

(problem, 23)(delayed, 38.6)(repair, 40.9)(done, 63)
The train crossing example
Modelling the train crossing example

Train\(_i\) with \(i = 1, 2, \ldots\)
The train crossing example – cont’d

The gate:

- **Open** → GoDown?, $H_g := 0$ → **Lowering, $H_g < 10$**
- **Open** ↓ \( H_g < 10, a \) → **Raising, $H_g < 10$**
- **Lowering, $H_g < 10$** ↑ \( H_g < 10, a \) → **Close**
- **Close** ↓ GoUp?, $H_g := 0$ → **Open**
The controller:

$c_1, H_c \leq 20$  $\xrightarrow{\text{Exit?}, H_c := 0} c_0$  $\xrightarrow{\text{App?}, H_c := 0} c_2, H_c \leq 10$

$H_c = 20, \text{GoUp!}$  $H_c \leq 10, \text{GoDown!}$
The train crossing example – cont’d

We use the synchronization function $f$:

<table>
<thead>
<tr>
<th>$\text{Train}_1$</th>
<th>$\text{Train}_2$</th>
<th>Gate</th>
<th>Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>App!</td>
<td>.</td>
<td>.</td>
<td>App?</td>
</tr>
<tr>
<td>.</td>
<td>App!</td>
<td>.</td>
<td>App</td>
</tr>
<tr>
<td>Exit!</td>
<td>.</td>
<td>.</td>
<td>Exit?</td>
</tr>
<tr>
<td>.</td>
<td>Exit!</td>
<td>.</td>
<td>Exit</td>
</tr>
<tr>
<td>a</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>a</td>
<td>.</td>
<td>a</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?
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?
Compute $D \times (C \times (A+B)) + (A+B) + (C \times D)$ using two processors:

**$P_1$ (fast):**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</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>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
</tr>
<tr>
<td>In use</td>
</tr>
</tbody>
</table>

**$P_2$ (slow):**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</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>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
</tr>
<tr>
<td>In use</td>
</tr>
</tbody>
</table>

Back to the task graph scheduling problem

<table>
<thead>
<tr>
<th>Sch1</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

13 picoseconds 1.37 nanojoules

<table>
<thead>
<tr>
<th>Sch2</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

12 picoseconds 1.39 nanojoules

<table>
<thead>
<tr>
<th>Sch3</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

19 picoseconds 1.32 nanojoules
Modelling the task graph scheduling problem
Modelling the task graph scheduling problem

- Processors

\(P_1:\)

\[\begin{align*}
\text{add}_1 & \quad \text{done}_1 \\
(x \leq 2) & \quad \text{idle} \quad x := 0 \\
\rightarrow & \quad \text{add}_1 \\
\text{mult}_1 & \quad \text{done}_1 \\
x = 2 & \quad \text{idle} \\
\rightarrow & \quad x := 0 \\
(x \leq 3) & \quad \text{idle} \quad x := 0
\end{align*}\]

\(P_2:\)

\[\begin{align*}
\text{add}_2 & \quad \text{done}_2 \\
(y \leq 5) & \quad \text{idle} \quad y := 0 \\
\rightarrow & \quad \text{add}_2 \\
\text{mult}_2 & \quad \text{done}_2 \\
y = 5 & \quad \text{idle} \\
\rightarrow & \quad y := 0 \\
(y \leq 7) & \quad \text{idle} \quad y := 0
\end{align*}\]
Modelling the task graph scheduling problem

- **Processors**

\[ P_1 : \]

\( P_1 : x = 2 \) → \( x = 0 \) \( add_1 \) \( done_1 \) idle \( x = 3 \) \( x = 0 \) \( mult_1 \) \( done_1 \) idle

\( P_1 : (x \leq 2) \) \( (x \leq 3) \)

\[ P_2 : \]

\( P_2 : y = 5 \) → \( y = 0 \) \( add_2 \) \( done_2 \) idle \( y = 7 \) \( y = 0 \) \( mult_2 \) \( done_2 \) idle

\( P_2 : (y \leq 5) \) \( (y \leq 7) \)

- **Tasks**

\[ T_4 : \]

\( T_4 : t_1 \land t_2 \) \( add_i \) \( done_i \) \( t_4 := 1 \)

\[ T_5 : \]

\( T_5 : t_3 \) \( add_i \) \( done_i \) \( t_5 := 1 \)
Modelling the task graph scheduling problem

- **Processors**

  \[ P_1: \]
  \[-\]
  \[ P_2: \]
  \[-\]

- **Tasks**

  \[ T_4: \]
  \[-\]
  \[ T_5: \]
  \[-\]

\( \leadsto \) 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**

  \( P_1: \)

  \[
  \begin{align*}
  x & \leq 2 \\
  x & = 2 \\
  \text{add}_1 & \\
  \text{done}_1 & \\
  x & = 0 \\
  \text{idle} & \\
  x & = 3 \\
  \text{done}_1 & \\
  \text{mult}_1 & \\
  \text{idle} & \\
  \end{align*}
  \]

- **Tasks**

  \( T_4: \)

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

  \( T_5: \)

  \[
  \begin{align*}
  t_3 & \\
  \text{add}_i & \\
  \text{done}_i & \\
  \end{align*}
  \]

  \( t_4 := 1 \)

  \( t_5 := 1 \)

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

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

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

- **Processors**
  
  \[ P_1: \]
  
  \[ P_2: \]
  
  \[ \text{idle} \]
  
  \[ x = 2 \]
  
  \[ x = 3 \]
  
  \[ y = 5 \]
  
  \[ y = 7 \]
  
  \[ \text{add}_1 \]
  
  \[ \text{mult}_1 \]
  
  \[ \text{add}_2 \]
  
  \[ \text{mult}_2 \]
  
  \[ (x \leq 2) \]
  
  \[ (y \leq 5) \]
  
  \[ (x \leq 3) \]
  
  \[ (y \leq 7) \]
  
  \[ x := 0 \]
  
  \[ y := 0 \]
  
  \[ \text{done}_1 \]
  
  \[ \text{done}_2 \]
  
  \[ \text{build the synchronized product of all these automata} \]
  
  \[ (P_1 \parallel P_2) \parallel_s (T_1 \parallel T_2 \parallel \cdots \parallel T_6) \]
  
  **A schedule:** a path in the global system which reaches \( t_1 \land \cdots \land t_6 \)

- **Tasks**
  
  \[ T_4: \]
  
  \[ t_4 := 1 \]
  
  \[ \text{add}_i \]
  
  \[ \text{done}_i \]
  
  \[ T_5: \]
  
  \[ t_5 := 1 \]
  
  \[ \text{add}_i \]
  
  \[ \text{done}_i \]

**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?
- \ldots
What we have so far

- A model which can adequately represent systems with real-time constraint...
- ... on which we can ask relevant questions
What we have so far

- A model which can adequately represent systems with real-time constraint...
- ... on which we can ask relevant questions

Interesting problems

- Which semantics? (and be aware of the limits of the choice)
- Algorithms for automatic verification
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

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

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

Example

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

\[\begin{align*}
x &\leq 2, \\
y &\geq 2,
\end{align*}\]

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

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

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

Example

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

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

Example

\[ x = 1 \quad y := 0 \]
\[ x \leq 2, \ y := 0 \]
\[ y \geq 2, \ y := 0 \]
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

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

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

Example

$\begin{align*}
x &= 1 \\
y &= 0
\end{align*}$

$\begin{align*}
x &\leq 2, \quad x := 0 \\
y &\geq 2, \quad y := 0
\end{align*}$
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

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

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

Example

We will focus on the continuous-time semantics, since this is an adequate abstraction of real-time systems.
Continuous-time semantics

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

Example

We will focus on the continuous-time semantics, since this is an adequate abstraction of real-time systems.

Known limits: robustness issues (we will comment on that later)
Analyzing timed automata

Can we reach state ○?
Analyzing timed automata

- **Problem:** the set of configurations is infinite
  \[ x \leq 2, \; x := 0 \]
  \[ y \geq 2, \; y := 0 \]

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

Can we reach state \( \bigcirc \)?
Crux idea: Region abstraction

Crux idea: 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

Crux idea: Region abstraction

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

Crux idea: Region abstraction

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

Crux idea: Region abstraction

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

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

Crux idea: Region abstraction

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

\[ \land \text{an equivalence of finite index} \]
\[ \land \text{a time-abstract bisimulation} \]

Time-abstract bisimulation

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

\[
\forall \delta(d) \exists d' \geq 0 \delta(d') \text{ and vice-versa (swap $\bullet$ and $\bullet$).}
\]

Consequence: $(\ell_1, v_1, d_1, a_1)(\ell_1, R_1) a_1(\ell_1, v_1', d_1', a_1)(\ell_2, v_2, d_2, a_2)(\ell_2, R_2) a_2(\ell_2, v_2', d_2', a_2)(\ell_3, v_3, d_3, a_3)(\ell_3, R_3) a_3(\ell_3, v_3', d_3', a_3) \ldots$
Time-abstract bisimulation

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

\[
\forall \quad \circ \xrightarrow{a} \bullet \\
\bullet \quad \bullet \quad \bullet
\]
Time-abstract bisimulation

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

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

... and vice-versa (swap \( \bullet \) and \( \bullet \)).
Time-abstract bisimulation

This is a relation between ● and ● such that:

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

\[
\exists \quad \rightarrow \quad a
\]

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

This is a relation between ● and ● such that:

∀ ● → ● [a]

∃ ● → ● [a]

∀ d ≥ 0

∃ d' ≥ 0
Time-abstract bisimulation

This is a relation between • and • such that:

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

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

\[ \forall \quad \exists \]

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

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

$\forall \quad \bullet \quad a \quad \bullet$

$\exists \quad \bullet \quad a \quad \bullet$

$\forall d \geq 0 \quad \bullet \quad \delta(d) \quad \bullet$

$\exists d' \geq 0 \quad \bullet \quad \delta(d') \quad \bullet$

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

Consequence

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

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

\[
\forall \quad \bullet \overset{a}{\rightarrow} \bullet \\
\exists \quad \bullet \overset{a}{\rightarrow} \bullet
\]

\[
\forall d \geq 0 \quad \bullet \overset{\delta(d)}{\rightarrow} \bullet \\
\exists d' \geq 0 \quad \bullet \overset{\delta(d')}{\rightarrow} \bullet
\]

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

Consequence

\[
\forall \quad (\ell_1, v_1) \overset{d_1, a_1}{\rightarrow} (\ell_2, v_2) \overset{d_2, a_2}{\rightarrow} (\ell_3, v_3) \overset{d_3, a_3}{\rightarrow} \cdots \\
(\ell_1, R_1) \overset{a_1}{\rightarrow} (\ell_2, R_2) \overset{a_2}{\rightarrow} (\ell_3, R_3) \overset{a_3}{\rightarrow} \cdots \quad \text{with } v_i \in R_i
\]
Time-abstract bisimulation

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

\[
\forall \begin{array}{c}
\bullet \\
\circ
\end{array} \xrightarrow{a} \begin{array}{c}
\bullet \\
\circ
\end{array} \quad \forall d \geq 0 \begin{array}{c}
\circ \\
\bullet
\end{array} \xrightarrow{\delta(d)} \begin{array}{c}
\circ \\
\bullet
\end{array}
\]

\[
\exists \begin{array}{c}
\circ \\
\bullet
\end{array} \xrightarrow{a} \begin{array}{c}
\circ \\
\bullet
\end{array} \quad \exists d' \geq 0 \begin{array}{c}
\bullet \\
\circ
\end{array} \xrightarrow{\delta(d')} \begin{array}{c}
\bullet \\
\circ
\end{array}
\]

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

Consequence

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

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

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

This is a relation between • and • such that:

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

... and vice-versa (swap • and •).

Consequence

\[
\forall (\ell_1, v_1) \xrightarrow{d_1,a_1} (\ell_2, v_2) \xrightarrow{d_2,a_2} (\ell_3, v_3) \xrightarrow{d_3,a_3} \cdots \\
(\ell_1, R_1) \xrightarrow{a_1} (\ell_2, R_2) \xrightarrow{a_2} (\ell_3, R_3) \xrightarrow{a_3} \cdots \quad \text{with } v_i \in R_i
\]

\[
\forall v_1' \in R_1 \exists (\ell_1, v_1') \xrightarrow{d_1',a_1} (\ell_2, v_2') \xrightarrow{d_2',a_2} (\ell_3, v_3') \xrightarrow{d_3',a_3} \cdots \quad \text{with } v'_i \in R_i
\]
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 \text{finite bisimulation quotient}
Region automaton \equiv \text{finite bisimulation quotient}
Region automaton $\equiv$ finite bisimulation quotient

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

Timed automaton

Region graph

Region automaton

language(reg. aut.) = UNTIME(language(timed aut.))
An example [AD94]
An example [AD94]
An example \[\text{[AD94]}\]
• A large (but finite) automaton (region automaton) can be obtained by taking the quotient of a timed automaton with respect to a finite bisimulation relation.

- 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)
• **large**: exponential in the number of clocks and in the constants (if encoded in binary). The number of regions is:

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

- **Weighted timed automata**
- **Timed games**
- **Weighted timed games**
- **Tools**
- **Towards further applications**
- **Conclusion**

---

**Timed automaton**

**finite bisimulation**

**quotient**

**large (but finite) automaton**

**(region automaton)**

- **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
Back to the example
Back to the example
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*).
[FJ13] Fearnley, Jurdziński. Reachability in two-clock timed automata is PSPACE-complete (*ICALP’13*).
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:

\[
\begin{align*}
0 < x < 1 \\
0 < y < 1 \\
y < x
\end{align*}
\]
Complexity issues

**Theorem** [AD90,AD94]

The emptiness problem for timed automata is decidable and \( \text{PSPACE-complete} \). It even holds for two-clock timed automata [FJ13]. It is \( \text{NLOGSPACE-complete} \) 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 \( \mathcal{M} \)
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 $\mathcal{M}$

![Diagram](image-url)

maximal number of cells in use: $N$

tape of $\mathcal{M}$

cell $C_i$

$x_i \leq 1$

cell $C_j$

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

- \(u:=0\)  
  - \(q,i\)  
  - \(x_1 \leq 4, x_1 := 0\)  
  - \(x_1 > 4\)

- \(u=2\)  
  -  
  - \(\cdots\)

- \(\cdots\)  
  -  
  - \(x_i \leq 4\)  
  - \(\cdots\)

- \(\cdots\)  
  -  
  - \(x_N \leq 4, x_N := 0\)  
  - \(x_N > 4\)

- \(u=3\)  
  -  
  - \(q', i+1\)

**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 \((q, a, b, q', \rightarrow)\):

\[
\begin{align*}
\text{constraint } x_j \leq 4: & \text{ cell } j \text{ contains an } a \\
\text{constraint } x_j > 4: & \text{ cell } j \text{ contains a } b \\
\text{reset of clock } x_j: & \text{ the new content is an } a
\end{align*}
\]
Example of the simulation of a rule \((q, a, b, q', \rightarrow)\):

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

- **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

- If only constants 0, 2, and 5 are used.
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
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)
• 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
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
- ...
• 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
  • ... 

• Note however that it might be hard to prove there is a finite bisimulation quotient!
What about the practice?

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

- Symbolic representation: zones

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

DBM: Difference Bound
Matrice [BM83,Dill89]
What about the practice?

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

- Symbolic representation: zones

\[ Z = (x_1 \geq 3) \land (x_2 \leq 5) \land (x_1 - x_2 \leq 4) \]
What about the practice?

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

- Symbolic representation: zones

\[ Z = (x_1 \geq 3) \land (x_2 \leq 5) \land (x_1 - x_2 \leq 4) \]
What about the practice?

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

• Symbolic representation: zones

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

\[ \begin{array}{ccc}
  x_0 & x_1 & x_2 \\
  0 & -3 & 0 \\
  9 & 0 & 4 \\
  5 & 2 & 0
\end{array} \]

"normal form"
What about the practice?

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

• Symbolic representation: zones
• Needs of (correct) extrapolation operators… [Bou04,BBLP06]

What about the practice?

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

• Symbolic representation: zones
• Needs of (correct) extrapolation operators… [Bou04,BBLP06]
• … or clever inclusion tests or simulation relations [HSW12,HSW13,GMS19,HSTW20]
What about the practice?

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

- Symbolic representation: zones
- Needs of (correct) extrapolation operators... \cite{Bou04,BBLP06}
- ... or clever inclusion tests or simulation relations \cite{HSW12,HSW13,GMS19,HSTW20}
- ... as well as abstraction-refinement techniques \cite{RSM19}
- ... and a good static analysis approach \cite{GMS20}
Before going further...

Which hypotheses did we make?

- timestamps taken in $\mathbb{R}_+$ (continuous-time semantics): only density is important, and they can be taken in $\mathbb{Q}_+$

- constants in clock constraints $x \sim c$: $c \in \mathbb{N}$; they could be taken in $\mathbb{Q}_+$, but not in $\mathbb{R}_+$!

- clock constraints of the form $x \sim c$ are fine as well

- no other kind of clock constraints!

- resets of clocks to 0 only; we can reset to integral values as well

- more involved updates can be used as well, but they don’t interact very well with diagonal constraints. So one needs to be careful
Before going further...

Which hypotheses did we make?

- timestamps taken in $\mathbb{R}_+$ (continuous-time semantics): only density is important, and they can be taken in $\mathbb{Q}_+$
- constants in clock constraints $x \sim c$: $c \in \mathbb{N}$; they could be taken in $\mathbb{Q}_+$, but not in $\mathbb{R}_+$!
Before going further...

### Which hypotheses did we make?

- Timestamps taken in $\mathbb{R}_+$ *(continuous-time semantics)*: only density is important, and they can be taken in $\mathbb{Q}_+$.
- Constants in clock constraints $x \sim c$: $c \in \mathbb{N}$; they could be taken in $\mathbb{Q}_+$, but not in $\mathbb{R}_+$!
- Clock constraints of the form $x \sim c$. 

32/103
Before going further...

Which hypotheses did we make?

- timestamps taken in $\mathbb{R}^+$ (continuous-time semantics): only density is important, and they can be taken in $\mathbb{Q}^+$.
- constants in clock constraints $x \sim c$: $c \in \mathbb{N}$; they could be taken in $\mathbb{Q}^+$, but not in $\mathbb{R}^+$!
- clock constraints of the form $x \sim c$
  - $x - y \sim c$ are fine as well
  - no other kind of clock constraints!
Before going further...

Which hypotheses did we make?

- Timestamps taken in $\mathbb{R}_+$ (continuous-time semantics): only density is important, and they can be taken in $\mathbb{Q}_+$.
- Constants in clock constraints $x \sim c$: $c \in \mathbb{N}$; they could be taken in $\mathbb{Q}_+$, but not in $\mathbb{R}_+$!
- Clock constraints of the form $x \sim c$
  - $x - y \sim c$ are fine as well
  - No other kind of clock constraints!
- Resets of clocks to 0 only; we can reset to integral values as well.
Before going further...

Which hypotheses did we make?

• timestamps taken in $\mathbb{R}_+$ (continuous-time semantics): only density is important, and they can be taken in $\mathbb{Q}_+$

• constants in clock constraints $x \sim c$: $c \in \mathbb{N}$; they could be taken in $\mathbb{Q}_+$, but not in $\mathbb{R}_+$!

• clock constraints of the form $x \sim c$
  • $x - y \sim c$ are fine as well
  • no other kind of clock constraints!

• resets of clocks to 0 only; we can reset to integral values as well
  • more involved updates can be used as well, but they don’t interact very well with diagonal constraints. So one needs to be careful
Limits of the model

• Any slight extension of the model is undecidable:
  • Richer clock constraints $x + y = c$, $2x \leq y$
  • Richer updates: $x := x + 1$
  • ...

Limits of the model
Limits of the model

• Any slight extension of the model is undecidable:
  • Richer clock constraints \( x + y = c, \ 2x \leq y \)
  • Richer updates: \( x := x + 1 \)
  • ...

• The inclusion problem
  \[ L(A) \subseteq L(B) \]

is undecidable [AD94]
Limits of the model

- Any slight extension of the model is undecidable:
  - Richer clock constraints $x + y = c$, $2x \leq y$
  - Richer updates: $x := x + 1$
  - ...

- The inclusion problem
  \[ L(A) \subseteq L(B) \]
  is undecidable [AD94]

- One cannot complement nor determinize timed automata

\[ a \]
\[ a, x := 0 \]
\[ a \]
\[ a \]
\[ a \]
\[ a, x := 1, a \]
An important issue: Robustness and implementability

\[ x = 1, \quad y = 0 \]
\[ x \leq 2, \quad x := 0 \]
\[ y \geq 2, \quad y := 0 \]

Value of clock \( x \) when hitting is converging, even though global time diverges.

Can we implement such a strategy?? No. But we can detect such behaviours, and give conditions for implementations!
An important issue: Robustness and implementability

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

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

\[ y \leq 1, \ y := 0 \]

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

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

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

\( x = 1 \)

\( y = 0 \)

\( y \leq 2, \ y := 0 \)

\( y \geq 2, \ y := 0 \)

\[ \therefore \text{Value of clock } x \text{ when hitting } \bigcirc \text{ is converging, even though global time diverges} \]
An important issue: Robustness and implementability

Value of clock $x$ when hitting $\bigcirc$ is converging, even though global time diverges

Can we implement such a strategy??

$\sim$
An important issue: Robustness and implementability

Value of clock $x$ when hitting $\circ$ is converging, even though global time diverges

Can we implement such a strategy??

No. But we can detect such behaviours, and give conditions for implementations!

An important issue: Robustness and implementability

Value of clock \( x \) when hitting \( \bigcirc \) is converging, even though global time diverges

Can we implement such a strategy??

No. But we can detect such behaviours, and give conditions for implementations!

A survey: [BMS13]

Theoretical recent developments

• Tree automata technics for timed automata analysis
  [AGK16,AGKS17]

  • Write behaviours as graphs with timing constraints
  • Realize that those graphs have bounded tree-width
  • Express properties using MSO and/or build directly tree automata

Theoretical recent developments

- Tree automata technics for timed automata analysis
  \[\text{[AGK16,AGKS17]}\]

- Write behaviours as graphs with timing constraints
- Realize that those graphs have bounded tree-width
- Express properties using MSO and/or build directly tree automata

- Compute and use the reachability relation \[\text{[CJ99,QSW17]}\]

Outline

1. Timed automata
2. Weighted timed automata
3. Timed games
4. Weighted timed games
5. Tools
6. Towards applying all this theory to robotic systems
7. Conclusion
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>

**$P_2$ (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>

**Energy:**

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

**Energy:**

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

**Schedule 1 ($Sch_1$):**

$P_1$: $T_2$, $T_3$, $T_5$, $T_6$

$P_2$: $T_1$, $T_4$

**Schedule 2 ($Sch_2$):**

$P_1$: $T_1$, $T_3$, $T_5$, $T_4$, $T_6$

$P_2$: $T_2$

**Schedule 3 ($Sch_3$):**

$P_1$: $T_1$, $T_3$, $T_4$

$P_2$: $T_2$, $T_5$, $T_6$
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):**

- \( + \): 2 picoseconds
- \( \times \): 3 picoseconds

**\( P_2 \) (slow):**

- \( + \): 5 picoseconds
- \( \times \): 7 picoseconds

**How to model energy consumption?**
Modelling resources in timed systems

- System resources might be relevant and even crucial information
Modelling resources in timed systems

- System **resources** might be relevant and even crucial information
  - energy consumption,
  - memory usage,
  - ...
  - price to pay,
  - bandwidth,
Modelling resources in timed systems

- System **resources** might be relevant and even crucial information
  - energy consumption,
  - memory usage,
  - ...
  
  ~ timed automata are not powerful enough!

- price to pay,
- bandwidth,
Modelling resources in timed systems

• System resources might be relevant and even crucial information
  • energy consumption,
  • memory usage,
  • ...

→ timed automata are not powerful enough!

• A possible solution: use hybrid automata
  a discrete control (the mode of the system)
  + continuous evolution of the variables within a mode
Modelling resources in timed systems

- System **resources** might be relevant and even crucial information
  - energy consumption,
  - memory usage,
  - ...
  - price to pay,
  - bandwidth,

  \[ T \leq 19 \]
  \[ T \geq 21 \]

  \[ \dot{T} = -0.5T \quad (T \geq 18) \]
  \[ \dot{T} = 2.25 - 0.5T \quad (T \leq 22) \]

  \[ \sim \text{timed automata are not powerful enough!} \]

- A possible solution: use **hybrid automata**

The thermostat example
Modelling resources in timed systems

- System resources might be relevant and even crucial information
  - energy consumption,
  - memory usage,
  - ...
  \[ \sim \text{timed automata are not powerful enough!} \]

- A possible solution: use hybrid automata

The thermostat example

\[ \begin{align*}
\text{Off} & \quad \dot{T} = -0.5T \\
& \quad (T \geq 18) \\
\text{On} & \quad \dot{T} = 2.25 - 0.5T \\
& \quad (T \leq 22) \\
\end{align*} \]
Ok...
Ok...

Easy...
Ok...

Easy...
Ok... but?
Ok... but?
Modelling resources in timed systems

- System resources might be relevant and even crucial information
  - energy consumption,
  - memory usage,
  - ...
  \[\rightarrow\] timed automata are not powerful enough!

- A possible solution: use hybrid automata

**Theorem** [HKPV95]

The reachability problem is **undecidable** in hybrid automata. Even for the simplest, the so-called stopwatch automata (clocks can be stopped).

Modelling resources in timed systems

- System **resources** might be relevant and even crucial information
  - energy consumption,
  - memory usage,
  - ...

  \(\leadsto\) timed automata are not powerful enough!

- A possible solution: use **hybrid automata**

**Theorem** [HKPV95]

The reachability problem is **undecidable** in hybrid automata. Even for the simplest, the so-called stopwatch automata (clocks can be stopped).

- An alternative: **weighted/priced timed automata** [ALP01,BFH+01]

  \(\leadsto\) hybrid variables do not constrain the system
  hybrid variables are **observer variables**

[HKPV95] Henzinger, Kopke, Puri, Varaiya. What’s decidable about hybrid automata? *(STOC’95).*

[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).*
Modelling the task graph scheduling problem

- **Processors**
  
  \[ P_1: \]
  
  \[
  \begin{align*}
  &+ \quad \text{(x} \leq 2) \\
  &\xrightarrow{\text{done}_1} \quad \text{idle} \\
  &\xrightarrow{\text{add}_1} \quad x:=0 \\
  &\xrightarrow{\text{mult}_1} \quad (x \leq 3)
  \end{align*}
  \]

  \[ P_2: \]
  
  \[
  \begin{align*}
  &+ \quad \text{(y} \leq 5) \\
  &\xrightarrow{\text{done}_2} \quad \text{idle} \\
  &\xrightarrow{\text{add}_2} \quad y:=0 \\
  &\xrightarrow{\text{mult}_2} \quad (y \leq 7)
  \end{align*}
  \]

- **Tasks**
  
  \[ T_4: \]
  
  \[
  \begin{align*}
  &t_1 \wedge t_2 \quad \text{add}_i \quad \text{done}_i \\
  &\xrightarrow{\text{t}_4 := 1}
  \end{align*}
  \]

  \[ T_5: \]
  
  \[
  \begin{align*}
  &t_3 \quad \text{add}_i \quad \text{done}_i \\
  &\xrightarrow{\text{t}_5 := 1}
  \end{align*}
  \]
Modelling the task graph scheduling problem

• Processors

\[ P_1: \]
\[ \begin{align*}
+ & \quad x = 2 \\
\text{idle} & \quad x = 3 \\
\text{done}_1 & \quad x = 0 \\
\text{add}_1 & \quad x = 0 \\
\text{mult}_1 & \quad x \leq 2 \\
\text{mult}_1 & \quad x \leq 3 \\
\end{align*} \]

\[ P_2: \]
\[ \begin{align*}
+ & \quad y = 5 \\
\text{idle} & \quad y = 7 \\
\text{done}_2 & \quad y = 0 \\
\text{add}_2 & \quad y = 0 \\
\text{mult}_2 & \quad y \leq 5 \\
\text{mult}_2 & \quad y \leq 7 \\
\end{align*} \]

• Tasks

\[ T_4: \]
\[ \begin{align*}
t_1 \land t_2 & \quad t_4 := 1 \\
done_1 & \quad \text{add}_i \\
done_1 & \quad \text{done}_i \\
\end{align*} \]

\[ T_5: \]
\[ \begin{align*}
t_3 & \quad t_5 := 1 \\
\text{add}_1 & \quad \text{add}_i \\
\text{done}_1 & \quad \text{done}_i \\
\end{align*} \]

• Modelling energy

\[ P_1: \]
\[ \begin{align*}
+90 & \quad x = 2 \\
+10 & \quad x = 3 \\
+90 & \quad x = 0 \\
\text{done}_1 & \quad x = 0 \\
\text{add}_1 & \quad x \leq 2 \\
\text{mult}_1 & \quad x \leq 3 \\
\end{align*} \]

\[ P_2: \]
\[ \begin{align*}
+30 & \quad y = 5 \\
+20 & \quad y = 7 \\
+30 & \quad y = 0 \\
\text{done}_2 & \quad y = 0 \\
\text{add}_2 & \quad y \leq 5 \\
\text{mult}_2 & \quad y \leq 7 \\
\end{align*} \]

A good schedule is a path in the product automaton with a low cost.
Weighted/priced timed automata [ALP01,BFH+01]

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

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

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

\[
\begin{align*}
\ell_0 & \xrightarrow{5} \ell_0, \\
& \xrightarrow{1.3} \ell_0, \\
& \xrightarrow{c} \ell_1, \\
& \xrightarrow{c} \ell_1, \\
& \xrightarrow{u} \ell_1, \\
& \xrightarrow{u} \ell_1, \\
& \xrightarrow{u} \ell_3, \\
& \xrightarrow{0.7} \ell_3, \\
& \xrightarrow{c} \ell_3, \\
& \xrightarrow{c} \ell_3, \\
& \xrightarrow{c} \ell_3, \\
& \xrightarrow{c} \ell_3.
\end{align*}
\]

\[
\begin{align*}
x & \leq 2, c, y := 0, \\
x & = 2, c, y := 0, \\
x & = 2, c, y := 0.
\end{align*}
\]
**Weighted/priced timed automata** [ALP01,BFH+01]

\[\ell_0 \xrightarrow{+5} \ell_1 \xrightarrow{u} \ell_2 \xrightarrow{u} \ell_3 \xrightarrow{+1} \ell_0 \]

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

\[\ell_1 \rightarrow \ell_3\]

\[u \rightarrow \ell_2 \]

\[x = 2, c\]

\[\ell_3 \rightarrow \ell_1\]

\[x = 2, c\]

\[\ell_0 \rightarrow \ell_1\]

\[1.3\]

\[\ell_0 \rightarrow \ell_0\]

\[c\]

\[\ell_0 \rightarrow \ell_1\]

\[c\]

\[\ell_1 \rightarrow \ell_3\]

\[+10\]

\[\ell_2 \rightarrow \ell_3\]

\[+1\]

\[\ell_3 \rightarrow \ell_3\]

\[+1\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

\[c\]

\[\ell_3 \rightarrow \ell_3\]

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

\[
\begin{align*}
\ell_0 & \xrightarrow{1.3} \ell_0 & \ell_0 & \xrightarrow{c} \ell_1 & \ell_1 & \xrightarrow{u} \ell_3 & \ell_3 & \xrightarrow{0.7} \ell_3 & \ell_3 & \xrightarrow{c} \\scriptstyle{\smile} \\
\ell_0 & \xrightarrow{x \leq 2, c, y := 0} \ell_1 & (y = 0) & \xrightarrow{u} \ell_3 & \xrightarrow{x = 2, c} \ell_2 & \xrightarrow{x = 2, c} \ell_3 & \xrightarrow{c} \ell_3 & \xrightarrow{0.7} \ell_3 & \xrightarrow{0.7} \ell_3 & \xrightarrow{0} \ell_3 \\
\end{align*}
\]

\[
\begin{align*}
x & 0 & 1.3 & 1.3 & 1.3 & 2 & 2 & 2 \\
y & 0 & 1.3 & 0 & 0 & 0 & 0 & 0 \\
\end{align*}
\]

cost : 6.5

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

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

\[
\begin{array}{cccccc}
\ell_0 & 1.3 & \ell_0 & c & \ell_1 & u & \ell_3 & 0.7 & \ell_3 & c & \text{smile} \\
x & 0 & 1.3 & 1.3 & 1.3 & 2 & 0.7 & \\
y & 0 & 1.3 & 0 & 0 & \\
\text{cost} : & 6.5 & + & 0
\end{array}
\]

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata (HSCC'01).
Weighted/priced timed automata \([\text{ALP01, BFH+01}]\)

\[
\begin{align*}
\ell_0 & \overset{5}{\longrightarrow} \ell_1 \overset{c}{\longrightarrow} \ell_3 \overset{0.7}{\longrightarrow} \ell_3 \overset{c}{\longrightarrow} \text{smiley face} \\
\ell_0 & \overset{1.3}{\longrightarrow} \ell_0 \overset{c}{\longrightarrow} \ell_1 \overset{u}{\longrightarrow} \ell_3 \\
\ell_2 & \overset{+10}{\longrightarrow} \ell_2 \overset{u}{\longrightarrow} \ell_3 \overset{+1}{\longrightarrow} \\
\end{align*}
\]

\[
\begin{array}{cccccc}
\ell_0 & \ell_0 & \ell_1 & \ell_3 & \ell_3 & \text{smiley face} \\
0 & 1.3 & 1.3 & 1.3 & 2 & \\
y & 0 & 1.3 & 0 & 0 & 0.7 \\
\end{array}
\]

cost : \[6.5 + 0 + 0\]
Weighted/priced timed automata [ALP01,BFH+01]

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

\begin{align*}
x & 0 & 1.3 & 1.3 & 1.3 & 2 \\
y & 0 & 1.3 & 0 & 0 & 0.7
\end{align*}

\text{cost} : 6.5 + 0 + 0 + 0.7


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

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

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata (HSCC‘01).
Weighted/priced timed automata [ALP01,BFH+01]

\[
\begin{array}{ccc}
\ell_0 & \xrightarrow{1.3} & \ell_0 \\
\ell_0 & \xrightarrow{c} & \ell_1 \\
\ell_1 & \xrightarrow{u} & \ell_3 \\
\ell_3 & \xrightarrow{0.7} & \ell_3 \\
\ell_3 & \xrightarrow{c} & \text{\smiley} \\
\end{array}
\]

\[
\begin{array}{c|ccc}
x & 0 & 1.3 & 1.3 \\
y & 0 & 1.3 & 0 \\
\end{array}
\]

\[
\text{cost : } 6.5 + 0 + 0 + 0.7 + 7 = 14.2
\]


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

Question: what is the optimal cost for reaching 😊?
Weighted/priced timed automata [ALP01,BFH+01]

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

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

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

Question: what is the optimal cost for reaching 😊?

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


Weighted/priced timed automata \cite{ALP01,BFH+01}

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

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

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

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

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

\cite{ALP01} Alur, La Torre, Pappas. Optimal paths in weighted timed automata (HSCC’01).
\cite{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]

Question: what is the optimal cost for reaching 😊?

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

〜 strategy: leave immediately $\ell_0$, go to $\ell_3$, and wait there 2 t.u.
Optimal-cost reachability

**Theorem** [ALP01,BFH+01,BBBR07]

In weighted timed automata, the optimal cost is an integer and can be computed in PSPACE.

- Technical tool: a refinement of the regions, the corner-point abstraction

---

[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).*

Technical tool: the corner-point abstraction
Technical tool: the corner-point abstraction
Technical tool: the corner-point abstraction

Abstract time successors:
Technical tool: the corner-point abstraction

Abstract time successors:

Concrete time successors:
Technical tool: the corner-point abstraction

Abstract time successors:

Concrete time successors:
Technical tool: the corner-point abstraction

Abstract time successors:

Concrete time successors:
Technical tool: the corner-point abstraction

Abstract time successors:

Concrete time successors:
Technical tool: the corner-point abstraction

Abstract time successors:

Concrete time successors:
Technical tool: the corner-point abstraction

Time elapsing
Discrete transition
Technical tool: the corner-point abstraction

Cost rate 3
Discrete cost 7
Technical tool: the corner-point abstraction

Cost rate 3
Discrete cost 7

Optimal cost in the weighted graph
= optimal cost in the weighted timed automaton!
From timed to discrete behaviours

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

Optimal reachability as a linear programming problem

\[
\begin{align*}
& t_1 \\
& \quad \rightarrow \\
& t_2 \\
& \quad \rightarrow \\
& t_3 \\
& \quad \rightarrow \\
& t_4 \\
& \quad \rightarrow \\
& t_5 \\
& \quad \rightarrow \\
& \cdots
\end{align*}
\]
For $t_1$ to $t_2$ to $t_3$ to $t_4$ to $t_5$, and so on, \[ t_1 + t_2 \leq 2 \]
From timed to discrete behaviours

Optimal reachability as a linear programming problem

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

**Optimal reachability as a linear programming problem**

\[
\begin{align*}
T_1 & \quad T_2 & \quad T_3 & \quad T_4 & \quad T_5 \\
\begin{align*}
& t_1 \quad x \leq 2 \\
& y := 0 \quad t_2 \\
& t_3 \quad t_4 \\
& y \geq 5 \\
& t_5 \quad \ldots
\end{align*}
\end{align*}
\]

\[
\begin{align*}
& t_1 + t_2 \leq 2 \\
& T_2 \leq 2 \\
& t_2 + t_3 + t_4 \geq 5 \\
& T_4 - T_1 \geq 5
\end{align*}
\]

Lemma

Let \( Z \) be a bounded zone and \( f \) be a function

\[
f : (T_1, \ldots, T_n) \mapsto c_1 T_1 + \cdots + c_n T_n
\]

well-defined on \( Z \). Then \( \inf_Z f \) is obtained on the border of \( Z \) with integer coordinates.

For every finite path \( \pi \) in \( A \), there exists a path \( \Pi \) in \( A \) such that

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

(\( \Pi \) is a "corner-point projection" of \( \pi \))
From timed to discrete behaviours

**Optimal reachability as a linear programming problem**

\[
\begin{align*}
T_1 & \quad T_2 & \quad T_3 & \quad T_4 & \quad T_5 \\
\quad t_1 & \quad t_2 & \quad t_3 & \quad t_4 & \quad t_5 & \quad \ldots &
\end{align*}
\]

\[
\begin{align*}
y := 0 & \quad x \leq 2 & \quad y \geq 5 & \quad &
\begin{cases}
t_1 + t_2 & \leq 2 \\
t_2 + t_3 + t_4 & \geq 5 \\
T_2 & \leq 2 \\
T_4 - T_1 & \geq 5
\end{cases}
\end{align*}
\]

**Lemma**

Let \( Z \) be a bounded zone 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

\[
\begin{align*}
T_1 & \quad T_2 & \quad T_3 & \quad T_4 & \quad T_5 \\
\circ & \xrightarrow{t_1} \circ & \xrightarrow{t_2} \circ & \xrightarrow{t_3} \circ & \xrightarrow{t_4} \circ & \xrightarrow{t_5} \circ & \ldots
\end{align*}
\]

\[
\begin{align*}
t_1 + t_2 & \leq 2 \\
t_2 + t_3 + t_4 & \geq 5 \\
T_2 & \leq 2 \\
T_4 - T_1 & \geq 5
\end{align*}
\]

Lemma

Let \( Z \) be a bounded zone 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.

\( \leadsto \) 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 \( \pi \)]
Approximation of abstract paths:

For any path $\Pi$ of $A_{cp}$, 

From discrete to timed behaviours

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.

$$||\Pi - \pi_{\varepsilon}||_{\infty} < \varepsilon \Rightarrow |\text{cost}(\Pi) - \text{cost}(\pi_{\varepsilon})| < \eta$$
Use of the corner-point abstraction

It is a very interesting abstraction, that can be used in several other contexts:

• for mean-cost optimization \[\text{[BBL04,BBL08]}\]
• for discounted-cost optimization \[\text{[FL08]}\]
• for all concavely-priced timed automata \[\text{[JT08]}\]
• for deciding frequency objectives \[\text{[BBBS11,Sta12]}\]
• ...
Going further 1: mean-cost optimization

\[ \dot{C} = p, \quad \dot{R} = g \]

Going further 1: mean-cost optimization

\[ \dot{\mathcal{C}} = p \quad \dot{\mathcal{R}} = g \]

\[ x \leq D \]

\[ x = 0 \]

\[ \text{att?} \]

\[ \text{att!} \]

\[ z \geq S \]

\[ z := 0 \]

\[ \text{Op} \]

\[ \xrightarrow{\sim} \text{compute optimal infinite schedules that minimize} \]

\[ \text{mean-cost}(\pi) = \lim_{n \to +\infty} \sup_{\pi_n} \frac{\text{cost}(\pi_n)}{\text{reward}(\pi_n)} \]

Going further 1: mean-cost optimization

\[ \text{Low} \quad \dot{C} = p \quad \dot{R} = g \]

\[ \text{High} \quad (x \leq D) \quad \dot{C} = P \quad \dot{R} = G \]

\[ \text{att?}, x := 0 \quad \text{att?}, x := 0 \]

\[ \text{att?} \quad \text{att?} \]

\[ x = D \quad x = 0 \]

\[ \text{Op} \quad \text{Op} \]

\[ z \geq S \quad z := 0 \]

\[ z := 0 \quad z := 0 \]

\[ \sim \text{compute optimal infinite schedules that minimize} \]

\[ \text{mean-cost}(\pi) = \limsup_{n \to +\infty} \frac{\text{cost}(\pi_n)}{\text{reward}(\pi_n)} \]

Schedule with ratio \( \approx 1.455 \)

Schedule with ratio \( \approx 1.478 \)

Going further 1: mean-cost optimization

\[ \dot{C} = p \quad \dot{R} = g \]

\[ x \leq D \]

\[ \dot{x} = 0 \]

\[ x = D \]

\[ \dot{x} = 0 \]

\[ \text{att?} \]

\[ \text{att!} \]

\[ z \geq S \]

\[ z = 0 \]

\[ x := 0 \]

\[ x := 0 \]

\[ \text{att?} \]

\[ \text{att!} \]

\[ \text{Op} \]

\[ \sim \text{compute optimal infinite schedules that minimize} \]

\[ \text{mean-cost}(\pi) = \lim_{n \to +\infty} \frac{\text{cost}(\pi_n)}{\text{reward}(\pi_n)} \]

**Theorem [BBL08]**

In weighted timed automata, the optimal mean-cost can be compute in PSPACE.

\[ \sim \text{the corner-point abstraction can be used} \]

From timed to discrete behaviours

- **Finite behaviours:** based on the following property

**Lemma**

Let $Z$ be a bounded zone and $f$ be a function $f: (t_1, \ldots, t_n) \mapsto \frac{\sum_{i=1}^{n} c_i t_i + c}{\sum_{i=1}^{n} r_i t_i + r}$

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

- **Finite behaviours**: based on the following property

**Lemma**

Let $Z$ be a bounded zone and $f$ be a function

$$f : (t_1, ..., t_n) \mapsto \frac{\sum_{i=1}^n c_i t_i + c}{\sum_{i=1}^n r_i t_i + r}$$

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

$\leadsto$ for every finite path $\pi$ in $\mathcal{A}$, there exists a path $\Pi$ in $\mathcal{A}_{cp}$ s.t.

$$\text{mean-cost}(\Pi) \leq \text{mean-cost}(\pi)$$
From timed to discrete behaviours

• **Finite behaviours:** based on the following property

**Lemma**

Let $Z$ be a bounded zone and $f$ be a function

$$f : (t_1, \ldots, t_n) \mapsto \frac{\sum_{i=1}^{n} c_i t_i + c}{\sum_{i=1}^{n} r_i t_i + r}$$

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

\[ \rightsquigarrow \]

for every finite path $\pi$ in $\mathcal{A}$, there exists a path $\Pi$ in $\mathcal{A}_{cp}$ s.t.

mean-cost($\Pi$) \leq mean-cost($\pi$)

• **Infinite behaviours:** decompose each sufficiently long projection into cycles:

The (acyclic) linear part will be negligible!
From timed to discrete behaviours

- **Finite behaviours:** based on the following property

**Lemma**

Let $Z$ be a bounded zone and $f$ be a function

$$f : (t_1, ..., t_n) \mapsto \frac{\sum_{i=1}^{n} c_i t_i + c}{\sum_{i=1}^{n} r_i t_i + r}$$

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

$\leadsto$ for every finite path $\pi$ in $A$, there exists a path $\Pi$ in $A_{cp}$ s.t.

$$\text{mean-cost}(\Pi) \leq \text{mean-cost}(\pi)$$

- **Infinite behaviours:** decompose each sufficiently long projection into cycles:

The (acyclic) linear part will be negligible!

$\leadsto$ the optimal cycle of $A_{cp}$ is better than any infinite path of $A$!
From discrete to timed behaviours

**Approximation of abstract paths:**

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

Approximation of abstract paths:

For any path $\Pi$ of $A_{cp}$, for any $\varepsilon > 0$, 

\[
\|\Pi - \pi_\varepsilon\|_{\infty} < \varepsilon 
\]

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

\[
\|\Pi - \pi_\varepsilon\|_{\infty} < \varepsilon \Rightarrow |mean-cost(\Pi) - mean-cost(\pi_\varepsilon)| < \eta
\]
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.

$$||\Pi - \pi_\varepsilon||_\infty < \varepsilon \Rightarrow |\text{mean-cost}(\Pi) - \text{mean-cost}(\pi_\varepsilon)| < \eta$$
Going further 2: concavely-priced cost functions

A general abstract framework for quantitative timed systems

**Theorem [JT08]**

In concavely-priced timed automata, optimal cost is computable, if we restrict to quasi-concave cost functions. For the following cost functions, the (decision) problem is even PSPACE-complete:

- optimal-time and optimal-cost reachability;
- optimal discrete discounted cost;
- optimal mean-cost.

\( \sim \) the corner-point abstraction can be used

Going further 3: discounted-time cost optimization

Globally, \((z \leq 8)\)

- **High**
  - \((x \leq 3)\)
  - \(x = 3, x := 0\)
  - \(\text{deg}\)
  - \(\text{att}\)
  - \(z \geq 2, x, z := 0\)
  - \(+2\)

- **Med**
  - \((x \leq 3)\)
  - \(x = 3\)
  - \(\text{deg}\)
  - \(\text{att}\)
  - \(z \geq 2, z := 0\)
  - \(+5\)

- **Low**
  - \(x = 3\)
  - \(\text{deg}\)
  - \(\text{att}\)
  - \(z \geq 2, z := 0\)
  - \(+9\)

Going further 3: discounted-time cost optimization

Globally, \((z \leq 8)\)

\[ x = 3, x := 0 \quad \text{deg} \]

\[ z \geq 2, x, z := 0 \quad \text{att} \]

\[ x = 3, x := 0 \quad \text{deg} \]

\[ z \geq 2, z := 0 \quad \text{att} \]

\[ z \leq 8; \text{compute optimal infinite schedules that minimize discounted cost over time} \]

[FL08] Fahrenberg, Larsen. Discount-optimal infinite runs in priced timed automata (*INFINITY’08*).
Going further 3: discounted-time cost optimization

Globally, \((z \leq 8)\)

\[
\begin{array}{ccc}
\text{High} & \text{Med} & \text{Low} \\
\downarrow +2 & \text{deg} & \downarrow +9 \\
x = 3, x := 0 & \text{att} & z = 2, z := 0 \\
\downarrow -2 & \text{deg} & \downarrow +1 \\
 z \geq 2, x, z := 0 & z \geq 2, z := 0 \\
\end{array}
\]

\[\sim \text{ compute optimal infinite schedules that minimize}\]

\[
\text{discounted-cost}_\lambda(\pi) = \sum_{n \geq 0} \lambda^{T_n} \int_{t=0}^{\tau_{n+1}} \lambda^t \text{cost}(\ell_n) \, dt + \lambda^{T_{n+1}} \text{cost}(\ell_n \xrightarrow{a_{n+1}} \ell_{n+1})
\]

if \(\pi = (\ell_0, v_0) \xrightarrow{\tau_1, a_1} (\ell_1, v_1) \xrightarrow{\tau_2, a_2} \cdots\) and \(T_n = \sum_{i \leq n} \tau_i\)

Going further 3: discounted-time cost optimization

Globally, \((z \leq 8)\)

\[
\begin{align*}
(x \leq 3) & \quad \text{deg} \\
n(x=3, x:=0) & \quad \text{att} \\
z \geq 2, x, z:=0 & \quad +2 \\
\text{High} & \quad +2 \\
\end{align*}
\]

\[
\begin{align*}
(x \leq 3) & \quad \text{deg} \\
n(x=3) & \quad \text{att} \\
z \geq 2, z:=0 & \quad +1 \\
\text{Med} & \quad +1 \\
\end{align*}
\]

\[
\begin{align*}
(x \leq 3) & \quad \text{deg} \\
n(x=3) & \quad \text{att} \\
z \geq 2, z:=0 & \quad +9 \\
\text{Low} & \quad +9 \\
\end{align*}
\]

\(\leadsto\) compute optimal infinite schedules that minimize discounted cost over time

Going further 3: discounted-time cost optimization

Globally, \((z \leq 8)\)

\[
\begin{align*}
(x \leq 3) & \quad \text{High} \\
& \quad \text{deg} \quad +2 \\
& \quad \text{att} \quad +2 \\
& \quad z \geq 2, x, z := 0
\end{align*}
\]

\[
\begin{align*}
(x \leq 3) & \quad \text{Med} \\
& \quad \text{deg} \quad +5 \\
& \quad \text{att} \quad +1 \\
& \quad z \geq 2, z := 0
\end{align*}
\]

\[
\begin{align*}
(x \leq 3) & \quad \text{Low} \\
& \quad +9
\end{align*}
\]

\[\sim\] compute optimal infinite schedules that minimize discounted cost over time

If \(\lambda = e^{-1}\), the discounted cost of that infinite schedule is \(\approx 2.16\)

Going further 3: discounted-time cost optimization

Globally, \((z \leq 8)\)

\[x = 3, x := 0\]  \( (x \leq 3) \)  \( \text{deg} \)  \( \text{att} \)  \( z \geq 2, x, z := 0 \)

\[x = 3\]  \( (x \leq 3) \)  \( \text{deg} \)  \( \text{att} \)  \( z \geq 2, z := 0 \)

\(+2\)  \(+2\)  \(+5\)  \(+1\)

\(z \geq 2, x, z := 0\)

\(z \geq 2, z := 0\)

\(\leadsto\) compute optimal infinite schedules that minimize discounted cost over time

Theorem [FL08]

In weighted timed automata, the optimal discounted cost is computable in \(\text{EXPTIME}\).

\(\leadsto\) the corner-point abstraction can be used

[FL08] Fahrenberg, Larsen. Discount-optimal infinite runs in priced timed automata (\textit{INFINITY’08}).
And symbolically?

- Non-obvious in general...

[LBB+01] Larsen, Behrmann, Brinksma, Fehnker, Hune, Pettersson, Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata (CAV’01).

And symbolically?

- Non-obvious in general...
- Only for optimal reachability

\[ \zeta = 2 - x + 2y \]

[LBB+01] Larsen, Behrmann, Brinksma, Fehnker, Hune, Pettersson, Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata (CAV’01).

And symbolically?

- Non-obvious in general...
- Only for optimal reachability

**Priced zones**

priced zone = zone + affine cost function

- Efficient representation: DBM + offset cost + affine coefficient for each clock

represented by: zone $Z$
- offset cost: $+4$
- rate for $x$: $-1$
- rate for $y$: $+2$

---

[LBB+01] Larsen, Behrmann, Brinksma, Fehnker, Hune, Pettersson, Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata (CAV’01).

Results

**Theorem [LBB+01,RLS06]**

The forward algorithm with standard inclusion is correct and terminates for **bounded** timed automata with non-negative costs.

Termination: well-quasi-order on priced zones

---

[LBB+01] Larsen, Behrmann, Brinksma, Fehnker, Hune, Pettersson, Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata *(CAV’01)*.

Results

**Theorem** [LBB+01,RLS06]

The forward algorithm with standard inclusion is correct and terminates for **bounded** timed automata with non-negative costs.

Termination: well-quasi-order on priced zones

- Development of an (abstract) inclusion test $\sqsubseteq_M$ on priced zones

---

[LBB+01] Larsen, Behrmann, Brinksma, Fehnker, Hune, Pettersson, Romijn. As cheap as possible: Efficient cost- optimal reachability for priced timed automata (*CAV’01*).


[BCM16] Bouyer, Colange, Markey. Symbolic Optimal Reachability in Weighted Timed Automata (*CAV’16*).
Results

**Theorem [LBB+01,RLS06]**

The forward algorithm with standard inclusion is correct and terminates for **bounded** timed automata with non-negative costs.

Termination: well-quasi-order on priced zones

- Development of an (abstract) inclusion test $\sqsubseteq \mathcal{M}$ on priced zones
- $\mathcal{Z} \sqsubseteq \mathcal{M} \mathcal{Z}'$ reduces to several bilevel linear optimization problems

---

[LBB+01] Larsen, Behrmann, Brinksma, Fehnker, Hune, Pettersson, Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata (CAV’01).


Results

**Theorem [LBB+01,RLS06]**

The forward algorithm with standard inclusion is correct and terminates for **bounded** timed automata with non-negative costs.

Termination: well-quasi-order on priced zones

- Development of an (abstract) inclusion test $\sqsubseteq_{M}$ on priced zones
- $\mathcal{Z} \sqsubseteq_{M} \mathcal{Z}'$ reduces to several bilevel linear optimization problems

**Theorem [BCM16]**

The forward algorithm with inclusion test $\sqsubseteq_{M}$ is correct and terminates for timed automata with some conditions on the cost.

It is always better than standard inclusion for bounded timed automata.

---

[LBB+01] Larsen, Behrmann, Brinksma, Fehnker, Hune, Pettersson, Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata (*CAV’01*).


[BCM16] Bouyer, Colange, Markey. Symbolic Optimal Reachability in Weighted Timed Automata (*CAV’16*).
Further problems: Energy management

Example

\[
\ell_0 \quad +6 \quad \ell_1 \quad -6
\]

\[
\ell_0 \quad \xrightarrow{x:=0} \quad \ell_1 \quad \xrightarrow{x=1} \quad \ell_2
\]
Further problems: Energy management

Example

\[ \ell_0 \xrightarrow{x=0} \ell_1 \xrightarrow{x=1} \ell_2 \]

-3 \quad +6 \quad -6

"energy is \geq 0"

- Lower-bound problem (L)

Skip energy management
Further problems: Energy management

Example

\[ \ell_0 -3 \xrightarrow{x:=0} \ell_1 +6 \xrightarrow{x=1} \ell_2 -6 \]

"energy is \geq 0"

- Lower-bound problem (L)
Further problems: Energy management

Example

\[ \begin{align*}
\ell_0 & \xrightarrow{x:=0} \ell_1 & \ell_1 & \xrightarrow{x=1} \ell_2
\end{align*} \]

-3 +6 -6

"energy is \( \geq 0 \)"

• Lower-bound problem (L)
Further problems: Energy management

Example

\[ \ell_0 - 3 \quad \ell_1 + 6 \quad \ell_2 - 6 \]

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

```
0 1 2 3 4
```

```
0 0.5 1
```

"energy is \geq 0"

- Lower-bound problem (L)
Further problems: Energy management

Example

-3 \rightarrow l_0 \Rightarrow x := 0 \rightarrow +6 \rightarrow l_1 \rightarrow -6 \rightarrow l_2 \Rightarrow x = 1

“energy is \geq 0”

- Lower-bound problem (L)

- Lower-and-upper-bound problem (L+U)

- Lower-and-weak-upper-bound problem (L+W)
Further problems: Energy management

Example

\[ \ell_0 \xrightarrow{x := 0} \ell_1 \xrightarrow{x = 1} \ell_2 \]

\[ \ell_0 -3 \quad \ell_1 +6 \quad \ell_2 -6 \]

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

“energy is in \([0,3]\)”

- Lower-bound problem \((L)\)
- Lower-and-upper-bound problem \((L+U)\)
Further problems: Energy management

Example

- Lower-bound problem ($L$)
- Lower-and-upper-bound problem ($L+U$)
Further problems: Energy management

Example

\[ \ell_0 - 3 \quad +6 \quad \ell_1 \quad -6 \quad \ell_2 \]

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

“energy is \( \geq 0 \)”

“energy is in \([0,3]\)”

- Lower-bound problem (\(L\))
- Lower-and-upper-bound problem (\(L+U\))
Further problems: Energy management

Example

- Lower-bound problem (L)
- Lower-and-upper-bound problem (L+U)
Further problems: Energy management

Example

-3
\[ l_0 \quad \xrightarrow{x:=0} \quad l_1 \quad \xrightarrow{x=1} \quad l_2 \]

\[ \ell_0 -3 \] \quad \[ +6 \] \quad \[ -6 \]

“energy is in [0,3]”

- Lower-bound problem (L)
- Lower-and-upper-bound problem (L+U)
Further problems: Energy management

Example

- Lower-bound problem ($L$)
- Lower-and-upper-bound problem ($L+U$)
Further problems: Energy management

Example

- Lower-bound problem (L)
- Lower-and-upper-bound problem (L+U)
Further problems: Energy management

Example

\[
\begin{align*}
\ell_0 &\xrightarrow{x:=0} \ell_1 & &\xrightarrow{x=1} \ell_2 \\
-3 & & +6 & & -6
\end{align*}
\]

“energy is in [0,2]”

- Lower-bound problem (L)
- Lower-and-upper-bound problem (L+U)
Further problems: Energy management

Example

\[ \ell_0 - 3 \xrightarrow{x:=0} \ell_1 + 6 \xrightarrow{x=1} \ell_2 - 6 \]

- Lower-bound problem (L)
- Lower-and-upper-bound problem (L+U)

“energy is in [0,2]”

lost!
Further problems: Energy management

Example

\[ \ell_0 - 3 \xrightarrow{x:=0} \ell_1 + 6 \xrightarrow{x=1} \ell_2 - 6 \]

"energy is \([0,2]\)"

- Lower-bound problem \((L)\)
- Lower-and-upper-bound problem \((L+U)\)
Further problems: Energy management

Example

\[
\begin{align*}
\ell_0 & \rightarrow -3 \rightarrow \ell_1 \rightarrow +6 \rightarrow \ell_2 \\
x := 0 & \quad x = 1
\end{align*}
\]

“energy is in \([0, 1]\)”

- Lower-bound problem (L)
- Lower-and-upper-bound problem (L+U)
Further problems: Energy management

Example

- Lower-bound problem (L)
- Lower-and-upper-bound problem (L+U)
Further problems: Energy management

Example

-3 \rightarrow l_0 \xrightarrow{x:=0} +6 \rightarrow l_1 \xrightarrow{x=1} -6 \rightarrow l_2

“energy is in \([0,1]\) with a weak upper bound”

- Lower-bound problem (L)
- Lower-and-upper-bound problem (L+U)
- Lower-and-weak-upper-bound problem (L+W)
The L-problem: use the corner-point abstraction?

**Idea:** delay in the most profitable location

\[ \sim \text{the corner-point abstraction} \]
The \textbf{L}-problem: use the corner-point abstraction?

**Idea:** delay in the most profitable location

\[ x := 0 \quad \leadsto \quad \text{the corner-point abstraction} \]

**Example**

![Diagram](image)
**The L-problem: use the corner-point abstraction?**

**Idea:** delay in the most profitable location

\[ x := 0 \]

\[ x = 1 \]

**Example**

![Diagram with states and transitions](image)
The **L**-problem: use the corner-point abstraction?

**Idea:** delay in the most profitable location

\[ x := 0 \]

\[ x = 1 \]

**Example**

\[ x = 0 \]
\[ e = 1 \]

\[ x = 0.3 \]
\[ e = 0.1 \]

\[ x = 0.3 \]
\[ e = 0.1 \]

\[ x = 0.8 \]
\[ e = 3.1 \]

\[ x = 0.8 \]
\[ e = 3.1 \]

\[ x = 1 \]
\[ e = 1.9 \]
The \textbf{L}-problem: use the corner-point abstraction?

\textbf{Idea:} delay in the most profitable location

\[\sim\text{ the corner-point abstraction}\]

\textbf{Example}

Figure showing a timed automaton with transitions labeled by delays and states.
The \textbf{L}-problem: use the corner-point abstraction?

**Idea:** delay in the most profitable location

\[ \leadsto \text{the corner-point abstraction} \]

**Theorem** [BFLMS08]

The corner-point abstraction is sound and complete for single-clock WTA with no discrete costs. Hence the existential L-problem is in PTIME in that case.

The $L$-problem: use the corner-point abstraction?

**Idea:** delay in the most profitable location

$\sim$ the corner-point abstraction

**Remark**

The corner-point abstraction is not correct with discrete costs.

\[ +2 \rightarrow -3 \rightarrow +4 \]

$x = 1, x := 0$
The L-problem: use the corner-point abstraction?

Idea: delay in the most profitable location

Remark

The corner-point abstraction is not correct with discrete costs.
The **L**-problem: use the corner-point abstraction?

**Idea:** delay in the most profitable location

\[ x = 1, x = 0 \]

**Remark**

The corner-point abstraction is not correct with discrete costs.

![Diagram of a graph with nodes labeled +2 and +4 connected by edges labeled +2 and -3, with a point marked as lost at the bottom right.]
The **L**-problem: use the corner-point abstraction?

**Idea:** delay in the most profitable location

\[ +2 \quad \rightarrow \quad -3 \quad \rightarrow \quad +4 \]

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

**Remark**

The corner-point abstraction is not correct with discrete costs.

The diagram illustrates the progression and decision points.

Lost!
The L-problem: use the corner-point abstraction?

**Idea:** delay in the most profitable location

\[ +2 \xrightarrow{+3} +4 \]

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

The corner-point abstraction is not correct with discrete costs.

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

\[ +2 \]

\[ \rightarrow \text{the corner-point abstraction} \]

\[ \sim \text{requires new developments!} \]
The **L**-problem: computing optimal delays

Example

\[
\begin{align*}
0 & \xrightarrow{x=0} 3 & \xrightarrow{+1} 6 & \xrightarrow{+1} 8 & \xrightarrow{0} 0 \\
\& \geq 0 & \geq 3 & \geq 7 & \geq 4
\end{align*}
\]

- compute optimal delays \( t_{\text{opt}} \) in \( \ell_1 \) to \( \ell_{n-1} \);
- compute optimal possible delays \( t^* \) in \( \ell_1 \) to \( \ell_{n-1} \);
- compute other points on the energy function curve.
The $L$-problem: computing optimal delays

Example

![Diagram showing a timed automaton with states 0, 3, 6, and 8, transitions x=0, x=1, and delays +1, ≥3, ≥7, ≥4, and final credit 8.]

- $t_{\text{opt}}$: $\frac{2}{3}$, $\frac{1}{2}$, 0, 0, 0

- compute optimal delays $t_{\text{opt}}$ in $\ell_1$ to $\ell_{n-1}$;
The **L**-problem: computing optimal delays

**Example**

![Diagram](image)

<table>
<thead>
<tr>
<th></th>
<th>$t_{opt}$</th>
<th></th>
<th>$t^*$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$x=0$</td>
<td></td>
<td>$\geq 0$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>+1</td>
<td></td>
<td>$\geq 3$</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>$0$</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>+1</td>
<td></td>
<td>$\geq 4$</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>$x=1$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- compute optimal delays $t_{opt}$ in $l_1$ to $l_{n-1}$;
- compute optimal possible delays $t^*$ in $l_1$ to $l_{n-1}$;
The L-problem: computing optimal delays

Example

\[
\begin{array}{cccccc}
0 & x=0 & +1 & \geq 0 & 3 & +1 \\
 & & & \geq 3 & 6 & 0 \\
 & & & \geq 7 & 8 & +1 \\
 & & & \geq 4 & x=1 & 0
\end{array}
\]

<table>
<thead>
<tr>
<th>( t_{opt} )</th>
<th>(-)</th>
<th>( \frac{2}{3} )</th>
<th>( \frac{1}{2} )</th>
<th>(-)</th>
<th>(-)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t^* )</td>
<td>(-)</td>
<td></td>
<td>0</td>
<td>(-)</td>
<td></td>
</tr>
</tbody>
</table>

- compute optimal delays \( t_{opt} \) in \( \ell_1 \) to \( \ell_{n-1} \);
- compute optimal possible delays \( t^* \) in \( \ell_1 \) to \( \ell_{n-1} \);
The L-problem: computing optimal delays

Example

```
0 → x=0 +1 ≥0 3 → +1 ≥3 6 → 0 ≥7 8 → +1 ≥4 x=1 0
```

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>3</th>
<th>6</th>
<th>8</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{opt}$:</td>
<td>—</td>
<td>$\frac{2}{3}$</td>
<td>$\frac{1}{2}$</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>$t^*$:</td>
<td>—</td>
<td>$\frac{1}{2}$</td>
<td>0</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

- compute optimal delays $t_{opt}$ in $l_1$ to $l_{n-1}$;
- compute optimal possible delays $t^*$ in $l_1$ to $l_{n-1}$;
The L-problem: computing optimal delays

Example

- compute optimal delays $t_{opt}$ in $\ell_1$ to $\ell_{n-1}$;
- compute optimal possible delays $t^*$ in $\ell_1$ to $\ell_{n-1}$;
The **L**-problem: computing optimal delays

**Example**

- Compute optimal delays \( t_{opt} \) in \( \ell_1 \) to \( \ell_{n-1} \);
- Compute optimal possible delays \( t^* \) in \( \ell_1 \) to \( \ell_{n-1} \);

Minimal initial credit required: \( \frac{1}{2} \), yields final credit 8.
The L-problem: computing optimal delays

Example

- compute optimal delays \( t_{\text{opt}} \) in \( \ell_1 \) to \( \ell_{n-1} \);
- compute optimal possible delays \( t^* \) in \( \ell_1 \) to \( \ell_{n-1} \);
- compute other points on the energy function curve.
The **L**-problem: computing optimal delays

**Example**

- **compute optimal delays** $t_{opt}$ in $\ell_1$ to $\ell_{n-1}$;
- **compute optimal possible delays** $t^*$ in $\ell_1$ to $\ell_{n-1}$;
- **compute other points on the energy function curve.**
The \textbf{L}-problem: computing optimal delays

Example

\begin{itemize}
  \item compute optimal delays $t_{opt}$ in $l_1$ to $l_{n-1}$;
  \item compute optimal possible delays $t^*$ in $l_1$ to $l_{n-1}$;
  \item compute other points on the energy function curve.
\end{itemize}
The **L**-problem: computing optimal delays

**Example**

![Diagram](image)

<table>
<thead>
<tr>
<th>$t_{\text{opt}}$</th>
<th>$- \quad \frac{2}{3} \quad \frac{1}{2} \quad - \quad -$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t^*$</td>
<td>$- \quad \frac{1}{2} \quad \frac{1}{2} \quad 0 \quad -$</td>
</tr>
</tbody>
</table>

| initial credit  | $\frac{1}{2} + \delta \quad \frac{1}{2} - \frac{\delta}{3} \quad \frac{1}{2}$ |

- compute **optimal delays** $t_{\text{opt}}$ in $\ell_1$ to $\ell_{n-1}$;
- compute **optimal possible delays** $t^*$ in $\ell_1$ to $\ell_{n-1}$;
- compute other points on the energy function curve.
The \textbf{L}-problem: computing optimal delays

Example

\begin{center}
\begin{tabular}{c|c|c|c|c|c}
\hline
 & $x=0$ & $+1$ & $+1$ & $0$ & $+1$ \\
\hline
0 & $\geq 0$ & $\geq 3$ & $\geq 7$ & $\geq 4$ & $x=1$ \\
\hline
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{c|c|c|c|c|c}
\hline
$t_{opt}$: & $-$ & $\frac{2}{3}$ & $\frac{1}{2}$ & $-$ & $-$ \\
\hline
$t^*$: & $-$ & $\frac{1}{2}$ & $\frac{1}{2}$ & 0 & $-$ \\
\hline
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{c|c|c|c|c}
\hline
initial credit & $\frac{1}{2} + \delta$ & $\frac{1}{2} - \frac{\delta}{3}$ & $\frac{1}{2}$ & $\frac{\delta}{3}$ \\
\hline
\end{tabular}
\end{center}

- compute optimal delays $t_{opt}$ in $\ell_1$ to $\ell_{n-1}$;
- compute optimal possible delays $t^*$ in $\ell_1$ to $\ell_{n-1}$;
- compute other points on the energy function curve.
The **L**-problem: computing optimal delays

**Example**

\[
\begin{array}{cccccc}
\text{Initial credit} & 0 & 3 & 6 & 8 & 0 \\
\text{Final credit} & 8 & 12 & 16 & \text{final credit} & \\
\frac{1}{2} + \delta & \frac{1}{2} - \frac{\delta}{3} & \frac{1}{2} & \frac{\delta}{3} & 8 + \frac{8}{3}\delta \\
\end{array}
\]

- compute **optimal delays** \(t_{\text{opt}}\) in \(\ell_1\) to \(\ell_{n-1}\);
- compute **optimal possible delays** \(t^*\) in \(\ell_1\) to \(\ell_{n-1}\);
- compute other points on the energy function curve.
Example

- compute optimal delays $t_{\text{opt}}$ in $l_1$ to $l_{n-1}$;
- compute optimal possible delays $t^*$ in $l_1$ to $l_{n-1}$;
- compute other points on the energy function curve.
The $L$-problem: computing optimal delays

Example

```
x=0 \quad \begin{array}{c} \geq 0 \\ +1 \quad \geq 3 \\ +1 \quad \geq 7 \\ 0 \quad \geq 4 \\ +1 \quad \geq 0 \end{array} \\
0 \rightarrow 3 \rightarrow 6 \rightarrow 8 \rightarrow 0 \rightarrow 0
```

<table>
<thead>
<tr>
<th>$t_{\text{opt}}$:</th>
<th>$\begin{array}{c} - \ \frac{2}{3} \ \frac{1}{2} \ - \ - \end{array}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t^*$:</td>
<td>$\begin{array}{c} - \ \frac{1}{2} \ \frac{1}{2} \ 0 \ - \end{array}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>initial credit</th>
<th>$2 + \delta$</th>
<th>$0$</th>
<th>$\frac{1}{2} - \delta$</th>
<th>$\frac{1}{2} + \delta$</th>
<th>final credit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$12 + \frac{8}{6}\delta$</td>
</tr>
</tbody>
</table>

- compute optimal delays $t_{\text{opt}}$ in $\ell_1$ to $\ell_{n-1}$;
- compute optimal possible delays $t^*$ in $\ell_1$ to $\ell_{n-1}$;
- compute other points on the energy function curve.
The $L$-problem: computing optimal delays

Example

![Diagram of timed automaton with transitions and delays]

<table>
<thead>
<tr>
<th>$t_{opt}$:</th>
<th>$\frac{2}{3}$</th>
<th>$\frac{1}{2}$</th>
<th>$-$</th>
<th>$-$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t^*$:</td>
<td>$\frac{1}{2}$</td>
<td>$\frac{1}{2}$</td>
<td>$0$</td>
<td>$-$</td>
</tr>
</tbody>
</table>

Initial credit: 5  0  0  1  
Final credit:    0  0  1  16

- compute optimal delays $t_{opt}$ in $\ell_1$ to $\ell_{n-1}$;
- compute optimal possible delays $t^*$ in $\ell_1$ to $\ell_{n-1}$;
- compute other points on the energy function curve.
The \( \mathbf{L} \)-problem: computing optimal delays

Example

\[
\begin{array}{c}
0 \quad \xrightarrow{x=0} \quad 3 \quad \xrightarrow{+1 \geq 0} \quad 6 \quad \xrightarrow{+1 \geq 3} \quad 8 \quad \xrightarrow{0 \geq 7} \quad 0 \\
\end{array}
\]

<table>
<thead>
<tr>
<th>( t_{\text{opt}} )</th>
<th>( t^* )</th>
<th>initial credit</th>
<th>final credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( - )</td>
<td>( - )</td>
<td>( 5 + \delta )</td>
<td>( 16 + \delta )</td>
</tr>
<tr>
<td>( \frac{2}{3} )</td>
<td>( \frac{1}{2} )</td>
<td>( 0 )</td>
<td>( 1 )</td>
</tr>
<tr>
<td>( \frac{1}{2} )</td>
<td>( \frac{1}{2} )</td>
<td>( 0 )</td>
<td>( 1 )</td>
</tr>
</tbody>
</table>

- compute optimal delays \( t_{\text{opt}} \) in \( \ell_1 \) to \( \ell_{n-1} \);
- compute optimal possible delays \( t^* \) in \( \ell_1 \) to \( \ell_{n-1} \);
- compute other points on the energy function curve.
Example

The L-problem: computing optimal delays

\[ \begin{align*}
0 & \xrightarrow{x=0} 3 \quad 3 & \xrightarrow{+1} 6 \quad 6 & \xrightarrow{+1} 8 \quad 8 & \xrightarrow{+1} 0 \\
\geq 0 & & \geq 3 & & \geq 7 & & \geq 4
\end{align*} \]

\[
\begin{array}{c|c|c}
\text{point} & w_{in} & w_{out} \\
\hline
\alpha & 1/2 & 8 \\
\beta & 2 & 12 \\
\gamma & 5 & 16 \\
\end{array}
\]
The \textbf{L}-problem: concluding

\textbf{Theorem}

Optimization, reachability and existence of infinite runs satisfying the constraint $\geq 0$ can be decided in EXPTIME in single-clock WTA.
The **L**-problem: concluding

**Theorem**

Optimization, reachability and existence of infinite runs satisfying the constraint $\geq 0$ can be decided in EXPTIME in single-clock WTA

- transform the automaton into an automaton *with energy functions*;
The **L**-problem: concluding

**Theorem**

Optimization, reachability and existence of infinite runs satisfying the constraint $\geq 0$ can be decided in EXPTIME in single-clock WTA

- transform the automaton into an automaton with energy functions;
The **L**-problem: concluding

**Theorem**

Optimization, reachability and existence of infinite runs satisfying the constraint \( \geq 0 \) can be decided in EXPTIME in single-clock WTA

- transform the automaton into an automaton with energy functions;
The L-problem: concluding

**Theorem**

Optimization, reachability and existence of infinite runs satisfying the constraint $\geq 0$ can be decided in EXPTIME in single-clock WTA

- transform the automaton into an automaton with energy functions;

![Diagram](image-url)
The L-problem: concluding

**Theorem**

Optimization, reachability and existence of infinite runs satisfying the constraint $\geq 0$ can be decided in EXPTIME in single-clock WTA.

- transform the automaton into an automaton with energy functions;

- check if simple cycles can be iterated (or if a Zeno cycle can be reached...)

\[ f \]
Outline

1. Timed automata
2. Weighted timed automata
3. Timed games
4. Weighted timed games
5. Tools
6. Towards applying all this theory to robotic systems
7. Conclusion
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
Why (timed) games?

- to model uncertainty

Example of a processor in the taskgraph example

- 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
Why (timed) games?

- to model uncertainty

Example of a processor in the taskgraph example

- to model an interaction with the environment

Example of the gate in the train/gate example
Modelling the task graph scheduling problem

- **Processors**
  
  \[ \begin{align*}
  P_1: & \quad \begin{cases}
    x = 2 & \text{(idle)} \\
    x = 3 & \text{(idle)}
  \end{cases} \\
  \quad \begin{cases}
    \text{add}_1 & \text{done}_1 \\
    \text{mult}_1 & \text{done}_1
  \end{cases}
  \\
  (x \leq 2) & \quad x := 0 \\
  (x \leq 3) & \quad x := 0
  \\
  & \quad \times
  \\
  P_2: & \quad \begin{cases}
    y = 5 & \text{(idle)} \\
    y = 7 & \text{(idle)}
  \end{cases} \\
  \quad \begin{cases}
    \text{add}_2 & \text{done}_2 \\
    \text{mult}_2 & \text{done}_2
  \end{cases}
  \\
  (y \leq 5) & \quad y := 0 \\
  (y \leq 7) & \quad y := 0
  \\
  & \quad \times
  
  \end{align*} \]

- **Tasks**
  
  \[ \begin{align*}
  T_4: & \quad \begin{cases}
    t_1 \land t_2 & \text{add}_i \\
    t_4 := 1 & \text{done}_i
  \end{cases}
  \\
  & \quad \text{t}_3
  \\
  T_5: & \quad \begin{cases}
    t_3 & \text{add}_i \\
    t_5 := 1 & \text{done}_i
  \end{cases}
  
  \end{align*} \]

- **Modelling energy**
  
  \[ \begin{align*}
  P_1: & \quad \begin{cases}
    x = 2 & \text{add}_1 \\
    x = 3 & \text{add}_1
  \end{cases} \\
  \quad \begin{cases}
    \text{done}_1 & \text{mult}_1 \\
    \text{done}_1 & \text{mult}_1
  \end{cases}
  \\
  (x \leq 2) & \quad x := 0 \\
  (x \leq 3) & \quad x := 0
  \\
  & \quad +90
  \\
  P_2: & \quad \begin{cases}
    y = 5 & \text{add}_2 \\
    y = 7 & \text{add}_2
  \end{cases} \\
  \quad \begin{cases}
    \text{done}_2 & \text{mult}_2 \\
    \text{done}_2 & \text{mult}_2
  \end{cases}
  \\
  (y \leq 5) & \quad y := 0 \\
  (y \leq 7) & \quad y := 0
  \\
  & \quad +30
  
  \end{align*} \]
Modelling the task graph scheduling problem

- **Processors**
  - $P_1$: 
    - $x = 2 \rightarrow \text{idle}$, $x := 0$
    - $x = 3 \rightarrow \text{idle}$, $x := 0$
    - $(x \leq 2)$
    - $(x \leq 3)$

- $P_2$: 
  - $y = 5 \rightarrow \text{idle}$, $y := 0$
  - $y = 7 \rightarrow \text{idle}$, $y := 0$
  - $(y \leq 5)$
  - $(y \leq 7)$

- **Tasks**
  - $T_4$: 
    - $t_1 \land t_2 \rightarrow \text{add}_i$, $t_5 := 1$
    - $(x \geq 1)$
    - $(x \leq 3)$

- $T_5$: 
  - $t_3 \rightarrow \text{add}_i$, $t_5 := 1$
  - $(y \geq 3)$
  - $(y \leq 2)$

- **Modelling energy**
  - $P_1$: 
    - $+90$ \text{idle}$, $x := 0$
    - $+10$ \text{idle}$, $x := 0$
    - $(x \leq 2)$
    - $(x \leq 3)$

  - $P_2$: 
    - $+30$ \text{idle}$, $x := 0$
    - $+20$ \text{idle}$, $x := 0$
    - $(y \leq 5)$
    - $(y \leq 7)$

- **Modelling uncertainty**
  - $P_1$: 
    - $x \geq 1 \rightarrow \text{idle}$
    - $x := 0$
    - $(x \leq 2)$
    - $(x \leq 3)$

  - $P_2$: 
    - $y \geq 3 \rightarrow \text{idle}$
    - $y := 0$
    - $(x \leq 2)$
    - $(x \leq 3)$

A (good) schedule is a strategy in the product game (with a low cost).
Modelling the task graph scheduling problem

- **Processors**

  \( P_1 : \)
  - \( x \leq 2 \) and \( x \leq 3 \)
  - \( x = 2 \)
  - \( done_1 \)
  - \( add_1 \)
  - \( x = 3 \)
  - \( done_1 \)
  - \( mult_1 \)

  \( P_2 : \)
  - \( y \leq 5 \) and \( y \leq 7 \)
  - \( y = 5 \)
  - \( done_2 \)
  - \( add_2 \)
  - \( y = 7 \)
  - \( done_2 \)
  - \( mult_2 \)

- **Tasks**

  \( T_4 : \) \( t_1 \land t_2 \)
  - \( add_i \)
  - \( done_i \)

  \( T_5 : \) \( t_3 \)
  - \( add_i \)
  - \( done_i \)

- **Modelling energy**

  \( P_1 : \)
  - \( x \leq 2 \)
  - \( x = 2 \)
  - \( done_1 \)
  - \( add_1 \)
  - \( x = 3 \)
  - \( done_1 \)
  - \( mult_1 \)
  - \( +90 \)

  \( P_2 : \)
  - \( y \leq 5 \)
  - \( y = 5 \)
  - \( done_2 \)
  - \( add_2 \)
  - \( y = 7 \)
  - \( done_2 \)
  - \( mult_2 \)
  - \( +30 \)

- **Modelling uncertainty**

  \( P_1 : \)
  - \( x \geq 1 \)
  - \( done_1 \)
  - \( add_1 \)
  - \( idle \)
  - \( x \geq 1 \)
  - \( done_1 \)
  - \( mult_1 \)

  \( P_2 : \)
  - \( y \geq 3 \)
  - \( done_2 \)
  - \( add_2 \)
  - \( idle \)
  - \( y \geq 2 \)
  - \( done_2 \)
  - \( mult_2 \)

A (good) schedule is a strategy in the product game (with a low cost)
An example of a timed game

Rule of the game
- **Aim:** avoid 😞 and reach 😊

\[
\begin{align*}
\ell_0 &\xrightarrow{(x \leq 2)} \ell_1 \\
\ell_1 &\xrightarrow{x \geq 1, u_3} \ell_0, \ell_2 \\
\ell_2 &\xrightarrow{x < 1, c_1} \ell_1, \ell_3 \\
\ell_3 &\xrightarrow{x \leq 1, c_3} \ell_2 \\
\ell_1 &\xrightarrow{x \geq 2, c_4} \ell_3 \\
x < 1, u_2, x := 0 &
\end{align*}
\]
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:

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

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 \((0.5, c_1)\)
  - \(\sim\) 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 \((0.5, c_1)\)
  - 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 \((0.5, c_1)\)
  \[ \rightsquigarrow \text{can be preempted by } u_2 \]
- 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 \((0.5, c_1)\)
  - can be preempted by \(u_2\)
- from \((\ell_2, \ast)\), play \((1 - \ast, c_2)\)
- 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:

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

Problems to be considered

\[ (x \leq 2) \]
\[ x \geq 1, u_3 \]
\[ x \leq 1, c_1 \]
\[ x < 1, u_1 \]
\[ x < 1, u_2, x := 0 \]
\[ x \geq 2, c_4 \]
\[ x \leq 1, c_3 \]
\[ c_2 \]
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?
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.

→ classical regions are sufficient for solving such problems
   a region-closed attractor can be computed

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 a region-closed attractor can be computed


Decidability of timed games

Theorem \[\text{AMPS98,HK99}\]

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

$\leadsto$ classical regions are sufficient for solving such problems a region-closed attractor can be computed

Theorem \[\text{AM99,BHPR07,JT07}\]

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

Back to the example: computing winning states

\[ (x \leq 2) \]

\[ x \leq 1, c_1 \]

\[ x < 1, u_1 \]

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

\[ c_2 \]

\[ x \leq 1, c_3 \]

\[ x \geq 1, u_3 \]

\[ x \geq 2, c_4 \]

\[ \ell_0 \]

\[ \ell_1 \]

\[ \ell_2 \]

\[ \ell_3 \]
Back to the example: computing winning states

\[
\begin{align*}
\ell_0 &\xrightarrow{x \leq 2} \ell_1 \\
\ell_1 &\xrightarrow{x \leq 1, c_1} \ell_2 \\
\ell_2 &\xrightarrow{x \leq 1, c_3} \ell_3 \\
\ell_3 &\xrightarrow{x \geq 1, u_3, x < 1, u_1, x := 0} \ell_0
\end{align*}
\]
Back to the example: computing winning states
Back to the example: computing winning states

\[ \ell_0 \quad x \leq 2 \]
\[ \ell_1 \quad x \leq 1, c_1 \]
\[ \ell_2 \quad x < 1, u_1 \]
\[ \ell_3 \quad x \leq 1, c_3 \]
\[ (x \leq 2) \]
\[ x \geq 1, u_3 \]
\[ x > 1, u_2, x := 0 \]

\[ \ell_0 \quad x \geq 1, u_3 \]
\[ \ell_1 \quad x > 2, c_4 \]
\[ x > 1, c_2 \]
\[ x \geq 1, c_3 \]

\[ \ell_0 \]
\[ 0 \quad 1 \quad 2 \quad 3 \]
\[ \ell_1 \]
\[ 0 \quad 1 \quad 2 \quad 3 \]
\[ \ell_2 \]
\[ 0 \quad 1 \quad 2 \quad 3 \]
\[ \ell_3 \]
\[ 0 \quad 1 \quad 2 \quad 3 \]
Back to the example: computing winning states
Back to the example: computing winning states

\[(x \leq 2)\]

- \(\ell_0\) for \(x \geq 1, u_3\)
- \(\ell_1\) for \(x \leq 1, c_1\)
- \(\ell_2\) for \(x < 1, u_1\)
- \(\ell_3\) for \(x < 1, u_2, x := 0\)

- \(x \geq 2, c_4\)
- \(x \leq 1, c_3\)
- \(c_2\)
Back to the example: computing winning states

\[
\begin{align*}
\ell_0 & \quad (x \leq 2) \\
\ell_1 & \quad x \leq 1, c_1 \\
\ell_2 & \quad x < 1, u_1 \\
\ell_3 & \quad c_2 \\
\ell_0 & \quad x \geq 1, u_3 \\
\ell_1 & \quad x \geq 2, c_4 \\
\ell_2 & \quad x \leq 1, c_3 \\
\ell_3 & \quad x < 1, u_2, x := 0
\end{align*}
\]
Back to the example: computing winning states

Winning states

Losing states

\[ \ell_0 \]

\[ \ell_1 \]

\[ \ell_2 \]

\[ \ell_3 \]
Decidability via attractors
Decidability via attractors

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

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

- controllable and uncontrollable discrete predecessors:

  $$\text{cPred}(X) = \bigcup_{a \text{ cont.}} \text{Pred}^a(X)$$
  $$\text{uPred}(X) = \bigcup_{a \text{ uncont.}} \text{Pred}^a(X)$$
Decidability via attractors

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

- controllable and uncontrollable discrete predecessors:

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

- time controllable predecessors:

\[
\text{delay } t \text{ t.u.}
\]

should be safe
Decidability *via* attractors

- $\text{Pred}^a(X) = \{\bullet \mid \bullet \xrightarrow{a} \bullet \in X\}$
- controllable and uncontrollable discrete predecessors:
  
  \[
  c\text{Pred}(X) = \bigcup_{a \text{ cont.}} \text{Pred}^a(X) \quad \text{uPred}(X) = \bigcup_{a \text{ uncont.}} \text{Pred}^a(X)
  \]
- time controllable predecessors:

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

  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:

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

- 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:

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

- 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:

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

- ...
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:
  \[ \text{Attr}_2(😊) = \pi(\text{Attr}_1(😊)) \]

- ... 
- The states from which one can ensure 😊 in no more than \( n \) steps is:
  \[ \text{Attr}_n(😊) = \pi(\text{Attr}_{n-1}(😊)) \]
Timed games with a reachability objective

We write:

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

- 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 $c\text{Pred}(X)$ and $u\text{Pred}(X)$.

- 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?

\[
\begin{align*}
\text{cPred}(X) \\
\text{uPred}(\neg X) \\
\text{Pred}_\delta(\text{cPred}(X), \neg \text{uPred}(\neg X))
\end{align*}
\]
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!

$c\text{Pred}(X)$ $\text{uPred}(\neg X)$
$\text{Pred}_\delta(\text{cPred}(X), \neg \text{uPred}(\neg X))$
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? Yes!

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

\[
\text{cPred}(X)
\]
\[
\text{uPred}(\neg X)
\]
\[
\text{Pred}_\delta(\text{cPred}(X), \neg \text{uPred}(\neg X))
\]
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 $c\text{Pred}(X)$ and $u\text{Pred}(X)$.
- Does $\pi$ also preserve unions of regions? Yes!

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

$\leadsto$ the computation of $\pi^*(\bigcirc)$ 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!

(cPred($X$), $\neg$uPred($\neg X$))

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

$\leadsto$ the computation of $\pi^*$ terminates!

... and is correct
And in practice?

• A zone-based forward algorithm with backtracking

[CDF+05, BCD+07]
Outline

1. Timed automata
2. Weighted timed automata
3. Timed games
4. Weighted timed games
5. Tools
6. Towards applying all this theory to robotic systems
7. Conclusion
A simple timed game

The optimal cost while reaching $\ell_0$ is

$$\inf_{0 \leq t \leq 2} \max \left( 5t + 10(2-t) + 1, 5t + (2-t) + 7 \right) = 14 + \frac{1}{3};$$

strategy: wait in $\ell_0$, and when $t = \frac{4}{3}$, go to $\ell_1$. 
A simple weighted timed game

A simple weighted timed game

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

Question: what is the optimal cost we can ensure while reaching \(\ell_2\)?

\[\inf_{0 \leq t \leq 2} \max \left(5t + 10(2 - t) + 1, 5t + (2 - t) + 7\right) = 14 + \frac{1}{3}\]

Strategy: wait in \(\ell_0\), and when \(t = \frac{4}{3}\), go to \(\ell_1\).
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

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

**Question:** what is the optimal cost we can ensure while reaching \( \smileyface \)?
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

\[ \inf_{0 \leq t \leq 2} \max \left( 5t + 10(2 - t) + 1, 5t + (2 - t) + 7 \right) = 14 + \frac{1}{3} \]

**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 😊?

\[
\inf_{0 \leq t \leq 2} \max ( 5t + 10(2 - t) + 1 , \ 5t + (2 - t) + 7 ) = 14 + \frac{1}{3}
\]

\[\leadsto \text{strategy: wait in } \ell_0, \text{ and when } t = \frac{4}{3}, \text{ go to } \ell_1\]
Optimal reachability in weighted timed games (1)

This topic has been fairly hot these since the 2000’s

[LMM02,ABM04,BCFL04,BBR05,BBM06,BLMR06,Rut11]
[HIK13,BGK+14,BJM15,BMR17,BMR18,MPR20]
Optimal reachability in weighted timed games (1)

This topic has been fairly hot these since the 2000’s

[LMM02, ABM04, BCFL04, BBR05, BBM06, BLMR06, Rut11]
[HIM13, BGK+14, BJM15, BMR17, BMR18, MPR20]

[LMM02]
Tree-like weighted timed games can be solved in 2EXPTIME.
Optimal reachability in weighted timed games (1)

This topic has been fairly hot these since the 2000’s

[LMM02, ABM04, BCFL04, BBR05, BBM06, BLMR06, Rut11]
[HI13, BGK+14, BJM15, BMR17, BMR18, MPR20]

[LMM02]
Tree-like weighted timed games can be solved in 2EXPTIME.

[ABM04, BCFL04]
Depth-$k$ weighted timed games can be solved in EXPTIME. There is a symbolic algorithm to solve weighted timed games with a strongly non-Zeno cost.
**Optimal reachability in weighted timed games (2)**

[BBR05,BBM06,BJM15]

In weighted timed games, the optimal cost (and the value) **cannot be computed**, as soon as games have three clocks or more.
Optimal reachability in weighted timed games (2)

[BBR05, BBM06, BJM15]

In weighted timed games, the optimal cost (and the value) cannot be computed, as soon as games have three clocks or more.

[BLMR06, Rut11, HIM13, BGK+14]

Turn-based optimal timed games are decidable in EXPTIME (resp. PTIME) when automata have a single clock (resp. with two rates). They are PTIME-hard.
What is easier with a single clock?

- Memoryless strategies can be non-optimal...

\[\ell_0 \quad \text{(}x \leq 1\text{)} \quad +2 \quad x=1 \quad x>0\]

\[\ell_1 \quad x<1 \quad x:=0 \quad +1\]

What is easier with a single clock?

- Memoryless strategies can be non-optimal...

... but memoryless almost-optimal strategies will be sufficient.

[BLMR06] Bouyer, Larsen, Markey, Rasmussen. Almost-optimal strategies in one-clock priced timed automata (*FSTTCS’06*).
What is easier with a single clock?

- Memoryless strategies can be non-optimal...

  ![Diagram](attachment:image.png)

  ... but memoryless almost-optimal strategies will be sufficient.

- Key: resetting the clock somehow resets the history...

[BLMR06] Bouyer, Larsen, Markey, Rasmussen. Almost-optimal strategies in one-clock priced timed automata (*FSTTCS'06*).
What is easier with a single clock?

• Memoryless strategies can be non-optimal...

\[ (x \leq 1) \quad \ell_0 \quad +2 \quad \ell_0 \]

\[ (x < 1) \quad \ell_1 \quad x = 1 \]

\[ x > 0 \quad \ell_1 \quad \smile \]

... but memoryless almost-optimal strategies will be sufficient.

• Key: resetting the clock somehow resets the history...

• By unfolding and removing one by one the locations, we can synthesize memoryless almost-optimal winning strategies.

[BLMR06] Bouyer, Larsen, Markey, Rasmussen. Almost-optimal strategies in one-clock priced timed automata (*FSTTCS'06*).
What is easier with a single clock?

- Memoryless strategies can be non-optimal...

\[
\begin{align*}
\ell_0 & \xrightarrow{x=1} \ell_1 \\
& \xrightarrow{x<1} \ell_0 \\
& \xrightarrow{x=0} \ell_1 \\
& \xrightarrow{x>0} \text{smiley face}
\end{align*}
\]

... but memoryless almost-optimal strategies will be sufficient.

- Key: resetting the clock somehow resets the history...
- By unfolding and removing one by one the locations, we can synthesize memoryless almost-optimal winning strategies.
- Rather involved proofs of correctness

\[ \sigma(c_2, x) = \begin{cases} 
  c_2 & \text{if } 0 \leq x < 2/5 \\
  c_2 & \text{if } 2/5 \leq x < 1/2 \\
  u_2 & \text{if } 1/2 \leq x \leq 1 
\end{cases} \]
Computing the optimal cost: why is that hard?

Given two clocks $x$ and $y$, we can check whether $y = 2x$. 
Computing the optimal cost: 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 $1 - x_0$. 

Computing the optimal cost: why is that hard?

Given two clocks $x$ and $y$, we can check whether $y = 2x$. 

$\begin{align*}
\text{Add}^+(x) & \rightarrow \text{Add}^+(x) & \rightarrow \text{Add}^-(y) \\
\text{Add}^-(x) & \rightarrow \text{Add}^-(x) & \rightarrow \text{Add}^+(y)
\end{align*}$
Given two clocks $x$ and $y$, we can check whether $y = 2x$. 

- In $\widehat{\square}$, cost $= 2x_0 + (1 - y_0) + 2$
Computing the optimal cost: 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 $\bigtriangleup$, cost = $2(1 - x_0) + y_0 + 1$
Computing the optimal cost: 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 $> 3$
Computing the optimal cost: why is that hard?

Given two clocks $x$ and $y$, we can check whether $y = 2x$.

- If $x = x_0$, cost $= 2x_0 + (1 - y_0) + 2$
- If $y = y_0$, 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 $> 3$
Computing the optimal cost: 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 $\bigodot$, 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 $= 3$
Computing the optimal cost: 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 $= 3$

  $\leadsto$ player 2 can enforce cost $3 + |y_0 - 2x_0|$
Computing the optimal cost: 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

$\sim$ player 2 can enforce cost $3 + |y_0 - 2x_0|$.

- Player 1 has a winning strategy with cost $\leq 3$ iff $y_0 = 2x_0$
Computing the optimal cost: 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:

$$x = \frac{1}{2^{c_1}} \quad \text{and} \quad y = \frac{1}{2^{c_2}}$$
Computing the optimal cost: 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:

\[
    x = \frac{1}{2^{c_1}} \quad \text{and} \quad y = \frac{1}{2^{c_2}}
\]

The two-counter machine has a halting computation iff player 1 has a winning strategy to ensure a cost no more than 3.
Computing the optimal cost: 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:

$$x = \frac{1}{2^{c_1}} \quad \text{and} \quad y = \frac{1}{2^{c_2}}$$

The two-counter machine has a halting computation iff player 1 has a winning strategy to ensure a cost no more than 3.

Globally, $$(x \leq 1, y \leq 1, u \leq 1)$$

$$x=1, x:=0 \quad \lor \quad y=1, y:=0$$

Test $(u=0)$
Computing the optimal cost: 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:

$$x = \frac{1}{2^{c_1}} \quad \text{and} \quad y = \frac{1}{2^{c_2}}$$

The two-counter machine has a halting computation iff player 1 has a winning strategy to ensure a cost no more than 3.
Computing the optimal cost: 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:

$$x = \frac{1}{2c_1} \quad \text{and} \quad y = \frac{1}{2c_2}$$

The two-counter machine has a halting computation iff player 1 has a winning strategy to ensure a cost no more than 3.
Computing the optimal cost: 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:
  \[ x = \frac{1}{2c_1} \quad \text{and} \quad y = \frac{1}{2c_2} \]

The two-counter machine has a halting computation iff player 1 has a winning strategy to ensure a cost no more than 3.
Computing the optimal cost: 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:

\[
\begin{align*}
  x &= \frac{1}{2c_1} \\
  y &= \frac{1}{2c_2}
\end{align*}
\]

The two-counter machine has a halting computation iff player 1 has a winning strategy to ensure a cost no more than 3.
Computing the optimal cost: 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:

$$x = \frac{1}{2c_1}, \quad \text{and} \quad y = \frac{1}{2c_2}$$

The two-counter machine has a halting computation iff player 1 has a winning strategy to ensure a cost no more than 3.
Shape of the reduction
Shape of the reduction

Instruction

Test module (acyclic)
Shape of the reduction

- Instruction
- Test module (acyclic)
- Cost 0 within the core of the game
Some further subtlety

Value of the game = infimum of all costs of strategies
Value of the game = infimum of all costs of strategies

The value of the game is 3, but no strategy has cost 3.
Some further subtlety

Value of the game = infimum of all costs of strategies

The value of the game is 3, but no strategy has cost 3.
A snapshot on the undecidability proof

Instruction

Test module
A snapshot on the undecidability proof
A snapshot on the undecidability proof

Leave with cost $3 + 1/2^n$ ($n$: length of the path)
A snapshot on the undecidability proof

\( \mathcal{M} \) does not halt iff the value of \( G_\mathcal{M} \) is 3

Leave with cost \( 3 + \frac{1}{2^n} \) (\( n \): length of the path)
Are we done?
Are we done? No!
Are we done? No!

**Optimal cost is computable...**

... when cost is strongly non-zeno. \[ \text{[AM04, BCFL04]} \]

There is \( \kappa > 0 \) s.t. for every region cycle \( C \), for every real run \( \varrho \) read on \( C \),

\[
\text{cost}(\varrho) \geq \kappa
\]

**Optimal cost is not computable...**

... when cost is almost-strongly non-zeno.

There is \( \kappa > 0 \) s.t. for every region cycle \( C \), for every real run \( \varrho \) read on \( C \),

\[
\text{cost}(\varrho) \geq \kappa \quad \text{or} \quad \text{cost}(\varrho) = 0
\]
Optimal cost is computable... when cost is strongly non-zeno. [AM04,BCFL04]
There is $\kappa > 0$ s.t. for every region cycle $C$, for every real run $\rho$ read on $C$,

$$\text{cost}(\rho) \geq \kappa$$

Optimal cost is not computable... but is approximable! [BJM15]
... when cost is almost-strongly non-zeno.
There is $\kappa > 0$ s.t. for every region cycle $C$, for every real run $\rho$ read on $C$,

$$\text{cost}(\rho) \geq \kappa \quad \text{or} \quad \text{cost}(\rho) = 0$$

Optimal cost is computable...

... when cost is strongly non-zeno. \cite{AM04,BCFL04}
There is $\kappa > 0$ s.t. for every region cycle $C$, for every real run $\varrho$ read on $C$,

$$\text{cost}(\varrho) \geq \kappa$$

Optimal cost is not computable... but is approximable!

... when cost is almost-strongly non-zeno. \cite{BJM15}
There is $\kappa > 0$ s.t. for every region cycle $C$, for every real run $\varrho$ read on $C$,

$$\text{cost}(\varrho) \geq \kappa \quad \text{or} \quad \text{cost}(\varrho) = 0$$

- Almost-optimality in practice should be sufficient
- Even when we know how to compute the value, we are only able to synthesize almost-optimal strategies...

\cite{BJM15} Bouyer, Jaziri, Markey. On the value problem in weighted timed games \textit{(CONCUR'15)}. 
Approximation of the optimal cost

**Theorem**

Let $\mathcal{G}$ be a weighted timed game, in which the cost is almost-strongly non-zeno. For every $\epsilon > 0$, one can compute:

- two values $v_\epsilon^-$ and $v_\epsilon^+$ such that
  
  $$|v_\epsilon^+ - v_\epsilon^-| < \epsilon \quad \text{and} \quad v_\epsilon^- \leq \text{optcost}_\mathcal{G} \leq v_\epsilon^+$$
Theorem

Let $G$ be a weighted timed game, in which the cost is almost-strongly non-zeno. For every $\epsilon > 0$, one can compute:

- two values $v^-_\epsilon$ and $v^+_\epsilon$ such that
  \[
  |v^+_\epsilon - v^-_\epsilon| < \epsilon \quad \text{and} \quad v^-_\epsilon \leq \text{optcost}_G \leq v^+_\epsilon
  \]

- one strategy $\sigma_\epsilon$ such that
  \[
  \text{optcost}_G \leq \text{cost}(\sigma_\epsilon) \leq \text{optcost}_G + \epsilon
  \]

[it is an $\epsilon$-optimal winning strategy]
Approximation of the optimal cost

**Theorem**

Let $G$ be a weighted timed game, in which the cost is almost-strongly non-zeno. For every $\epsilon > 0$, one can compute:

- two values $v^-_\epsilon$ and $v^+_\epsilon$ such that
  $$|v^+_\epsilon - v^-_\epsilon| < \epsilon \quad \text{and} \quad v^-_\epsilon \leq \text{optcost}_G \leq v^+_\epsilon$$

- one strategy $\sigma_\epsilon$ such that
  $$\text{optcost}_G \leq \text{cost}(\sigma_\epsilon) \leq \text{optcost}_G + \epsilon$$

[It is an $\epsilon$-optimal winning strategy]
Approximation of the optimal cost

Theorem

Let $\mathcal{G}$ be a weighted timed game, in which the cost is almost-strongly non-zeno. For every $\epsilon > 0$, one can compute:

- two values $v_{\epsilon}^-$ and $v_{\epsilon}^+$ such that
  
  $$ |v_{\epsilon}^+ - v_{\epsilon}^-| < \epsilon \quad \text{and} \quad v_{\epsilon}^- \leq \text{optcost}_G \leq v_{\epsilon}^+ $$

- one strategy $\sigma_{\epsilon}$ such that

  $$ \text{optcost}_G \leq \text{cost}(\sigma_{\epsilon}) \leq \text{optcost}_G + \epsilon $$

  [it is an $\epsilon$-optimal winning strategy]

- Standard technics: unfold the game to get more precision, and compute two adjacency sequences
Approximation of the optimal cost

**Theorem**

Let $G$ be a weighted timed game, in which the cost is almost-strongly non-zeno. For every $\epsilon > 0$, one can compute:

- two values $v_\epsilon^-$ and $v_\epsilon^+$ such that
  \[ |v_\epsilon^+ - v_\epsilon^-| < \epsilon \quad \text{and} \quad v_\epsilon^- \leq \text{optcost}_G \leq v_\epsilon^+ \]
- one strategy $\sigma_\epsilon$ such that
  \[ \text{optcost}_G \leq \text{cost}(\sigma_\epsilon) \leq \text{optcost}_G + \epsilon \]

[it is an $\epsilon$-optimal winning strategy]

- Standard technics: unfold the game to get more precision, and compute two adjacency sequences
- This is not possible here
  There might be runs with prefixes of arbitrary length and cost 0 (e.g. the game of the undecidability proof)
Idea for approximation

Idea

Only partially unfold the game:

- Keep components with cost 0 untouched – we call it the kernel
- Unfold the rest of the game
Idea for approximation

**Idea**

Only partially unfold the game:
- Keep components with cost 0 untouched – we call it the **kernel**
- Unfold the rest of the game

First: split the game along regions!

\[
g, Y := 0
\]

\[
\begin{align*}
& r_1, Y := 0 \\
& r_2, Y := 0 \\
& r_3, Y := 0 \\
& r_4, Y := 0 \\
& r_5, Y := 0
\end{align*}
\]
Idea of the proof: Semi-unfolding

Hypothesis: cost > 0 implies cost ≥ κ

Conclusion: we can stop unfolding the game after finitely many steps
Idea of the proof: Semi-unfolding
Idea of the proof: Semi-unfolding

Hypothesis: \( \text{cost} > 0 \) implies \( \text{cost} \geq \kappa \)

Conclusion: we can stop unfolding the game after finitely many steps
Idea of the proof: Semi-unfolding

Hypothesis: cost > 0 implies cost ≥ κ
Idea of the proof: Semi-unfolding

Hypothesis:
\[ \text{cost} > 0 \implies \text{cost} \geq \kappa \]

Conclusion: we can stop unfolding the game after finitely many steps
Approximation scheme
Approximation scheme
Approximation scheme

Exact computation
Approximation scheme
Approximation scheme
First step: Tree-like parts

\[ O(\ell, v) = \inf_{t' | v + t' = g'} \max(\alpha), \sup_{t'' \leq t' | v + t'' = g''} t''c + c'' + O(\ell', v') \]

\[ v' = \left[ Y' \leftarrow 0 \right] (v + t') \]

\[ v'' = \left[ Y'' \leftarrow 0 \right] (v + t'') \]

\(~\Rightarrow\) Goes back to [LMM02]

[LMM02] La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata (TCS@02).
First step: Tree-like parts

\[ O(\ell, v) = \inf t' \mid v + t' = g' \]

\[ t' = c + c' + O(\ell', v') \]

\[ t'' = c'' + O(\ell'', v'') \]

\[ g', Y' \quad g'', Y'' \]

\[ c', c'' \]

\[ \ell', \ell'' \]

\[ \sim \text{ Goes back to [LMM02]} \]

[LM02] La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata (TCS@02).
First step: Tree-like parts

\[ O(\ell, v) = \]

\[ \sim \text{ Goes back to [LMM02]} \]

[LMM02] La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata (TCS@02).
First step: Tree-like parts

\[ O(\ell, v) = \inf_{t' \mid v + t' = g'} \]

Goes back to [LMM02]

[LMM02] La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata (TCS@02).
First step: Tree-like parts

\[ O(\ell, v) = \inf_{t' \mid v + t' = g'} \max(\ldots, \ldots) \]

\[ O(\ell', v') \quad O(\ell'', v'') \]

\[ g', Y' \quad c' \]

\[ g'', Y'' \quad c'' \]

Goes back to [LMM02]

[LMM02] La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata (TCS@02).
First step: Tree-like parts

\[ O(\ell, v) = \inf_{t' \mid v + t' = g'} \max((\alpha), \quad) \]

\[ (\alpha) = t' c + c' + O(\ell', v') \]

\[ v' = [Y' \leftarrow 0](v + t') \]

\[ \sim \text{ Goes back to [LMM02]} \]
First step: Tree-like parts

\[ g', Y', \ell' \quad O(\ell', v') \]
\[ g'', Y'', \ell'' \quad O(\ell'', v'') \]

\[ (\alpha) = t' c + c' + O(\ell', v') \]
\[ (\beta) = \sup_{t'' \leq t' | v + t''| = g''} t'' c + c'' + O(\ell'', v'') \]

\[ v' = [Y' \leftarrow 0](v + t') \]
\[ v'' = [Y'' \leftarrow 0](v + t'') \]

\[ O(\ell, v) = \inf_{t' | v + t'| = g'} \max((\alpha), (\beta)) \]

\[ \sim \text{ Goes back to [LMM02]} \]

[LMM02] La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata (TCS@02).
Second step: Kernels

Output cost functions $f$
Second step: Kernels

1. Refine the regions such that $f$ differs of at most $\epsilon$ within a small region

Output cost functions $f$
Second step: Kernels

1. Refine the regions such that $f$ differs of at most $\epsilon$ within a small region.

Output cost functions $f$
Second step: Kernels

1. Refine the regions such that $f$ differs of at most $\epsilon$ within a small region

Output cost functions $f$
Second step: Kernels

1. Refine the regions such that $f$ differs of at most $\epsilon$ within a small region.

2. Under- and over-approximate by piecewise constant functions $f^{-}_\epsilon$ and $f^{+}_\epsilon$.

Output cost functions $f$
Second step: Kernels

3 Refine/split the kernel along the new small regions and fix $f_\epsilon^-$ or $f_\epsilon^+$, write $f_\epsilon$

$f_\epsilon$: constant    $f_\epsilon$: constant
Second step: Kernels

3. Refine/split the kernel along the new small regions and fix $f_{\epsilon}^-$ or $f_{\epsilon}^+$, write $f_{\epsilon}$.

4. Since cost is 0 everywhere, the resulting game is nothing more than a reachability timed game with an order on target (output) edges (given by $f_{\epsilon}$).

- $f_{\epsilon}$: constant
- $f_{\epsilon}$: constant
Second step: Kernels

3. Refine/split the kernel along the new small regions and fix $f_\epsilon^-$ or $f_\epsilon^+$, write $f_\epsilon$

4. Since cost is 0 everywhere, the resulting game is nothing more than a reachability timed game with an order on target (output) edges (given by $f_\epsilon$)

5. Those can be solved using standard technics based on attractors: small regions are sufficient, and the local optimal cost (for output $f_\epsilon$) is constant within a small region

$f_\epsilon$: constant  $f_\epsilon$: constant
Second step: Kernels

3. Refine/split the kernel along the new small regions and fix $f_{\epsilon}^-$ or $f_{\epsilon}^+$, write $f_{\epsilon}$.

4. Since cost is 0 everywhere, the resulting game is nothing more than a reachability timed game with an order on target (output) edges (given by $f_{\epsilon}$).

5. Those can be solved using standard technics based on attractors: small regions are sufficient, and the local optimal cost (for output $f_{\epsilon}$) is constant within a small region.
Second step: Kernels

3. Refine/split the kernel along the new small regions and fix $f_{\epsilon^-}$ or $f_{\epsilon^+}$, write $f_{\epsilon}$

4. Since cost is 0 everywhere, the resulting game is nothing more than a reachability timed game with an order on target (output) edges (given by $f_{\epsilon}$)

5. Those can be solved using standard technics based on attractors: small regions are sufficient, and the local optimal cost (for output $f_{\epsilon}$) is constant within a small region

→ We have computed $\epsilon$-approximations of the optimal cost, which are constant within small regions. Corresponding strategies can be inferred
Outline

1. Timed automata
2. Weighted timed automata
3. Timed games
4. Weighted timed games
5. Tools
6. Towards applying all this theory to robotic systems
7. Conclusion
Tools for (weighted) timed automata and games

- Many tools and prototypes everywhere on earth...
Tools for (weighted) timed automata and games

- Many tools and prototypes everywhere on earth...

- **Tool-suite Uppaal**, developed in Aalborg (Denmark) and originally Uppsala (Sweden) since 1995
  - Uppaal for timed automata
  - Uppaal-TiGa for timed games
  - Uppaal-Cora for weighted timed automata

Uppaal url: [http://www.uppaal.org](http://www.uppaal.org)
Tools for (weighted) timed automata and games

- Many tools and prototypes everywhere on earth...
- Tool-suite Uppaal, developed in Aalborg (Denmark) and originally Uppsala (Sweden) since 1995

Uppaal url: http://www.uppaal.org
Tools for (weighted) timed automata and games

• Many tools and prototypes everywhere on earth...

• Tool-suite Uppaal, developed in Aalborg (Denmark) and originally Uppsala (Sweden) since 1995

• Our first tool TiAMo (Timed Automata Model-checker), developed in 2016 by Maximilien Colange (formerly at LSV), using code by Ocan Sankur (IRISA, France)
Tools for (weighted) timed automata and games

- Many tools and prototypes everywhere on earth...

- **Tool-suite Uppaal**, developed in Aalborg (Denmark) and originally Uppsala (Sweden) since 1995

- **Our first tool TiAMo** (Timed Automata Model-checker), developed in 2016 by Maximilien Colange (formerly at LSV), using code by Ocan Sankur (IRISA, France)

- **The new tool Tchecker**, developed within ANR Ticktac project mostly by Frédéric Herbreteau (LaBRI), Ocan Sankur (IRISA), Gérald Point (LaBRI), Philipp Schlehuber-Caissier (LRDE) and Alexandre Duret-Lutz (LRDE)

  \[\text{http://www.irisa.fr/sumo/ticktac/}\]
Outline

1. Timed automata
2. Weighted timed automata
3. Timed games
4. Weighted timed games
5. Tools
6. Towards applying all this theory to robotic systems
7. Conclusion
Very refreshing collaboration with Nicolas Markey (LSV at that time, now at IRISA), Nicolas Perrin (ISIR) and Philipp Schlehuber-Caissier (ISIR at that time, now at LRDE)
Example problem, objective and approach

**Example Problem**

- Infinitely many configurations
- Complex behaviour
- Mechanical constraints

**Objective:**

- Synthesize a controller:
  - Which robot handles an object
  - How to avoid collision
  - Don't miss any object

**Approach:**

- Discretization of the behaviour via a fixed set of continuous controllers
- Create an abstraction and use previous results

---


Example problem, objective and approach

- Infinitely many configurations
- Complex behaviour
- Mechanical constraints


Example problem, objective and approach

Goal: Synthesize a controller:
- Which robot handles an object
- How to avoid collision
- Don’t miss any object

- Infinitely many configurations
- Complex behaviour
- Mechanical constraints


Example problem, objective and approach

Goal: Synthesize a controller:
- Which robot handles an object
- How to avoid collision
- Don’t miss any object

Approach:
- Discretization of the behaviour via a fixed set of continuous controllers
- Create an abstraction and use previous results

- Infinitely many configurations
- Complex behaviour
- Mechanical constraints

Our approach

**Simplistic idea:** fixed set of reference trajectories + property
Our approach

**Simplistic idea:** fixed set of reference trajectories + property

Corresponding timed automaton:

\[ P_1(t) \]
\[ P_2(t) \]

Corresponding timed automaton:
More realistic idea: fixed set of funnels for control law + property
Our approach

**More realistic idea:** fixed set of funnels for control law + property

\[
\begin{align*}
F_1(t \in I_1) & \quad F_2(t \in I_2), \\
a_{12} & \leq t \leq b_{12} & \quad t := c_{12} \\
a_{21} & \leq t \leq b_{21} & \quad t := c_{21}
\end{align*}
\]
Control funnels

System with continuous dynamics $\dot{x} = f(x, t)$
Control funnels

System with continuous dynamics $\dot{x} = f(x, t)$

A (control) funnel is a trajectory $F(t)$ of a set in the state space such that, for any trajectory $x(t)$ of the dynamical system:

$$\forall t_0 \in \mathbb{R}, \ x(t_0) \in F(t_0) \Rightarrow \forall t \geq t_0, \ x(t) \in F(t)$$
Control funnels

System with continuous dynamics $\dot{x} = f(x, t)$

A (control) funnel is a trajectory $\mathcal{F}(t)$ of a set in the state space such that, for any trajectory $x(t)$ of the dynamical system:

$$\forall t_0 \in \mathbb{R}, \ x(t_0) \in \mathcal{F}(t_0) \Rightarrow \forall t \geq t_0, \ x(t) \in \mathcal{F}(t)$$
How to build funnels?

• Needs specific competences...
How to build funnels?

- Needs specific competences...
- Design funnels tracking trajectories:

\[
F_\alpha(t) = \{ x_{\text{ref}}(t) + x_\Delta | V(x_\Delta(t)) \leq \alpha \}
\]

with \( V \) a Lyapunov function.

\( F_\alpha \) is a fixed \( d \)-dimensional ellipsoid centered on the reference trajectory.

Those enjoy absorption properties:

\[
V(x_\Delta(t+\delta t)) \leq e^{-\beta \cdot \delta t} V(x_\Delta(t))
\]

\( F_\alpha_1 \delta_1, F_\alpha_2 \delta_2, F_\alpha_3 \delta_3, \ldots \) with \( \alpha_1 > \alpha_2 > \alpha_3 > \ldots \)

* Linear Quadratic Regulator
How to build funnels?

- Needs specific competences...
- Design funnels tracking trajectories:
  - First easy example: trajectory of the form $e^{-t} \cdot x_0$

For every $W \subseteq \mathbb{R}^d$ with $x_0 \in W$, $\mathcal{F}_W : t \mapsto \{e^{-t} \cdot w \mid w \in W\}$

* Linear Quadratic Regulator
How to build funnels?

- Needs specific competences...
- Design funnels tracking trajectories:
  - First easy example: trajectory of the form $e^{-t} \cdot x_0$
    
    For every $W \subseteq \mathbb{R}^d$ with $x_0 \in W$, $\mathcal{F}_W : t \mapsto \{e^{-t} \cdot w \mid w \in W\}$
  - More generally, LQR* funnels $\dot{x} = Ax + Bu$, with $x : \mathbb{R} \to \mathbb{R}^d$
    
    $\mathcal{F}_\alpha : t \mapsto \{x_{\text{ref}}(t) + x_\Delta \mid V(x_\Delta) \leq \alpha\}$

    with $V$ a Lyapunov function

* Linear Quadratic Regulator
How to build funnels?

- Needs specific competences...
- Design funnels tracking trajectories:
  - First easy example: trajectory of the form $e^{-t} \cdot x_0$
    
    For every $W \subseteq \mathbb{R}^d$ with $x_0 \in W$, $F_W : t \mapsto \{e^{-t} \cdot w | w \in W\}$
  - More generally, LQR* funnels $\dot{x} = Ax + Bu$, with $x : \mathbb{R} \rightarrow \mathbb{R}^d$
    
    $F_\alpha : t \mapsto \{x_{\text{ref}}(t) + x_\Delta | V(x_\Delta) \leq \alpha\}$

    with $V$ a Lyapunov function
    - $F_\alpha$ is a fixed $d$-dimensional ellipsoid centered on the reference trajectory

* Linear Quadratic Regulator
How to build funnels?

• Needs specific competences...
• Design funnels tracking trajectories:
  • First easy example: trajectory of the form $e^{-t} \cdot x_0$
    
    For every $W \subseteq \mathbb{R}^d$ with $x_0 \in W$, $\mathcal{F}_W : t \mapsto \{e^{-t} \cdot w \mid w \in W\}$
  
  • More generally, LQR* funnels $\dot{x} = Ax + Bu$, with $x : \mathbb{R} \rightarrow \mathbb{R}^d$
    
    $\mathcal{F}_\alpha : t \mapsto \{x_{\text{ref}}(t) + x_\Delta \mid V(x_\Delta) \leq \alpha\}$

with $V$ a Lyapunov function

• $\mathcal{F}_\alpha$ is a fixed $d$-dimensional ellipsoid centered on the reference trajectory
• Those enjoy absorption properties:
  
  $V(x_\Delta(t + \delta t)) \leq e^{-\beta \cdot \delta t} V(x_\Delta(t))$

  $\mathcal{F}_{\alpha_1} \sim \mathcal{F}_{\alpha_2} \sim \mathcal{F}_{\alpha_3} \sim \ldots \quad$ with $\alpha_1 > \alpha_2 > \alpha_3 > \ldots$

* Linear Quadratic Regulator
Example
Example

c_t: positional clock; c_h: local clock

$\alpha_1 \leq c_t \leq \beta_1$

$\alpha_2 \leq c_t \leq \beta_2$

$c_t := \gamma_1; c_h := 0$

$c_t := \gamma_2, c_h := 0$

$c_t := \gamma_1; c_h := 0$

$c_h \geq \Delta$

$c_t \in I_1^1$

$c_t \in I_2^1$

$c_t \in I_2^2$

$c_t \in I_3^1$
Summary

(huge) timed automata/games with weights, with few clocks → safe (good) controller ← winning (optimal) strategy
Summary

(huge) timed automata/games (with weights), with few clocks
Summary

\[ \sim (\text{huge}) \text{ timed automata/games (with weights), with few clocks} \]

\[ \leftarrow \text{winning (optimal) strategy} \]
(huge) timed automata/games (with weights), with few clocks

safe (good) controller  \rightleftharpoons \text{winning (optimal) strategy}
A pick-and-place example

1d point mass
A pick-and-place example

1d point mass

Funnel system

\( \dot{x}_{\text{lim}} \)

\( x^4_{\text{ref}} \)

\( x^3_{\text{ref}} \)

\( x^L3_{\text{ref}} \)

\( x \)

\( \mathcal{F}_{0-4}^{(t)} \)

\( \mathcal{F}_{Lk}^{(0-1)}(t) \)

\( x^2_{\text{ref}} \)

\( x^1_{\text{ref}} \)

\( x^{-1}_{\text{ref}} \)
Current challenges

For control people

- Handle more non-linear systems (automatically build control funnels)
Current challenges

For control people

- Handle more non-linear systems (automatically build control funnels)

For us

- Does not scale up very well so far (huge timed automata models)
  - Build the model on-demand?
    But, can we give guarantees (optimality) when only part of the model has been built?
  - Develop specific algorithms for the special timed automata we construct?
  - Note: Reachability is indeed in NLOGSPACE...
- Implement efficient approx. algorithm for weighted timed games
Outline

1. Timed automata
2. Weighted timed automata
3. Timed games
4. Weighted timed games
5. Tools
6. Towards applying all this theory to robotic systems
7. Conclusion
Conclusion

Summary of the talk

• Basics of timed automata verification
• Relevant extensions for applications: weights, games, mix of both
  • We looked at decidability and limits
  • We mentioned algorithmics and tools
• Timed automata can be used as abstractions for more complex systems
Conclusion

Current challenges

• Various theoretical issues
  • Decidability and approximability of weighted timed automata and games
  • New approaches (tree automata, reachability relations) might give a new light on the verification of timed systems
  • Robustness and implementability
• Continue working on algorithms, tools and benchmarks

Within ANR project Ticktac

• Implementation of (weighted) timed games (good data structures, abstractions, etc.)
• More applications with specific challenges (e.g. robotic problems)

ANR Ticktac webpage: http://www.irisa.fr/sumo/ticktac/