On the optimal reachability problem in weighted timed automata and games

Patricia Bouyer-Decitre

LSV, CNRS & ENS Cachan, France
Time-dependent systems

- We are interested in timed systems
Time-dependent systems

- We are interested in timed systems

... and in their analysis and control
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):

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

\( P_2 \) (slow):

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

\[ [\text{BFLM10}] \text{ Bouyer, Fahrenberg, Larsen, Markey. Quantitative Analysis of Real-Time Systems using Priced Timed Automata (Communication of the ACM).} \]
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

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

<table>
<thead>
<tr>
<th>Energy</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>10 Watt</td>
</tr>
<tr>
<td>In use</td>
<td>90 Watts</td>
</tr>
</tbody>
</table>

$P_2$ (slow):

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

<table>
<thead>
<tr>
<th>Energy</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>20 Watts</td>
</tr>
<tr>
<td>In use</td>
<td>30 Watts</td>
</tr>
</tbody>
</table>

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):**

\[
\begin{array}{|c|c|}
\hline
\text{time} & \text{energy} \\
\hline
+ & \text{idle} \\
2 \text{ picoseconds} & 10 \text{ Watt} \\
\times & \text{in use} \\
3 \text{ picoseconds} & 90 \text{ Watts} \\
\hline
\end{array}
\]

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

\[
\begin{array}{|c|c|}
\hline
\text{time} & \text{energy} \\
\hline
+ & \text{idle} \\
5 \text{ picoseconds} & 20 \text{ Watts} \\
\times & \text{in use} \\
7 \text{ picoseconds} & 30 \text{ Watts} \\
\hline
\end{array}
\]

\[
\begin{align*}
\text{Compute} & \\
D \times (C \times (A+B)) + (A+B) + (C \times D) & \\
& \text{using two processors:}
\end{align*}
\]

**\( P_1 \) (fast):**

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

\[
\begin{align*}
\text{time} & \\
+ & 2 \text{ picoseconds} \\
\times & 3 \text{ picoseconds} \\
\text{energy} & \\
\text{idle} & 10 \text{ Watt} \\
\text{in use} & 90 \text{ Watts}
\end{align*}
\]

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

- **Safe**: $x = 0$, $y = 0$
- **Alarm**: $2 \leq y \wedge x \leq 56$
- **Failsafe**: $15 \leq x \leq 16$
- **Done**: $22 \leq y \leq 25$

Transition rules:
- **Repair**: $y = 0$
- **Delayed**: $y = 0$
- **Problem**: $x = 0$

States:
- **Safe**
- **Alarm**
- **Repairing**
- **Failsafe**
The model of timed automata

\[
\begin{align*}
safe &\xrightarrow{23} safe \\
x &: 0 \quad 23 \\
y &: 0 \quad 23 \\
problem &\xrightarrow{15.6} alarm \\
x &: 0 \quad 15.6 \\
y &: 23 \quad 38.6 \\
\text{alarm} &\xrightarrow{15.6} alarm \\
x &: 0 \quad 15.6 \\
y &: 23 \quad 38.6 \\
\text{done} &\xrightarrow{ } safe \\
22 \leq y \leq 25 &\xrightarrow{ } done \\
15 \leq x \leq 16 &\xrightarrow{ } delayed \\
y &: 0 \\
\text{delayed} &\xrightarrow{ } failsafe \\
x &: 0 \quad 22.1 \\
y &: 0 \quad 22.1 \\
\text{failsafe} &\xrightarrow{2.3} failsafe \\
x &: 15.6 \quad 17.9 \\
y &: 0 \quad 22.1 \\
\text{repair} &\xrightarrow{22.1} repairing \\
x &: 17.9 \quad 40 \\
y &: 0 \quad 22.1 \\
\text{repairing} &\xrightarrow{ } done \\
x &: 17.9 \quad 40 \\
y &: 0 \quad 22.1 \\
\text{done} &\xrightarrow{ } safe \\
\end{align*}
\]
Modelling the task graph scheduling problem
Modelling the task graph scheduling problem

- **Processors**

  \[ P_1: \]
  - \( x \leq 2 \)
  - \( x := 0 \)
  - \( \text{add}_1 \)
  - \( \text{done}_1 \)
  - \( x = 2 \)
  - \( x = 3 \)
  - \( \text{mul}_1 \)
  - \( \text{mul}_2 \)
  - \( (x \leq 3) \)

  \[ P_2: \]
  - \( y \leq 5 \)
  - \( y := 0 \)
  - \( \text{add}_2 \)
  - \( \text{done}_2 \)
  - \( y = 5 \)
  - \( y = 7 \)
  - \( \text{mul}_2 \)
  - \( (y \leq 7) \)
Modelling the task graph scheduling problem

- **Processors**

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

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

- **Tasks**

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

  \[ T_5: \]
  \[ \begin{align*}
  &t_3 \quad \text{add}_i \\
  &t_5 := 1 \quad \text{done}_i \\
  \end{align*} \]
Modelling the task graph scheduling problem

- **Processors**
  - $P_1$: 
    - $x \leq 2$: $x := 0$ (idle)
    - $x = 2$: add$_1$, done$_1$
    - $x = 3$: mult$_1$, done$_1$
  - $P_2$: 
    - $y \leq 5$: $y := 0$ (idle)
    - $y = 5$: add$_2$, done$_2$
    - $y = 7$: mult$_2$, done$_2$

- **Tasks**
  - $T_4$: 
    - $t_1 \land t_2$: add$_i$, done$_i$
    - $t_4 := 1$
  - $T_5$: 
    - $t_3$: add$_i$
    - $t_5 := 1$

$\leadsto$ build the synchronized product of all these automata

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

- **Processors**
  - $P_1$: $x = 2$ \\
  - $P_2$: $y = 5$

- **Tasks**
  - $T_4$: $t_1 \land t_2$
  - $T_5$: $t_3$

\[ \sim \text{ build the synchronized product of all these automata} \]
\[ (P_1 \parallel P_2) \parallel_s (T_1 \parallel T_2 \parallel \cdots \parallel T_6) \]

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

- **Processors**
  
  $P_1$: 
  \[ \begin{array}{c}
  \text{idle} \\
  \text{add}_1 \\
  \text{done}_1 \\
  \text{add}_2 \\
  \text{done}_2 \\
  \text{mult}_1 \\
  \text{mult}_2 \\
  \text{idle} \\
  \end{array} \]
  \[ \begin{array}{c}
  x=2 \\
  x:=0 \\
  x:=0 \\
  y=5 \\
  y:=0 \\
  y:=0 \\
  (x \leq 2) \\
  (y \leq 5) \\
  \end{array} \]

  $P_2$: 
  \[ \begin{array}{c}
  \text{idle} \\
  \text{add}_1 \\
  \text{done}_1 \\
  \text{add}_2 \\
  \text{done}_2 \\
  \text{mult}_2 \\
  \text{mult}_1 \\
  \text{idle} \\
  \end{array} \]
  \[ \begin{array}{c}
  x=3 \\
  x:=0 \\
  x:=0 \\
  y=7 \\
  y:=0 \\
  y:=0 \\
  (x \leq 3) \\
  (y \leq 7) \\
  \end{array} \]

- **Tasks**
  
  $T_4$: 
  \[ \begin{array}{c}
  \text{add}_i \\
  \text{done}_i \\
  \end{array} \]
  \[ \begin{array}{c}
  t_1 \land t_2 \\
  t_4:=1 \\
  \end{array} \]

  $T_5$: 
  \[ \begin{array}{c}
  \text{add}_i \\
  \text{done}_i \\
  \end{array} \]
  \[ \begin{array}{c}
  t_3 \\
  t_5:=1 \\
  \end{array} \]

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

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

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

**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
Analyzing timed automata

\[ x = 1, y := 0 \quad \text{and} \quad y \geq 2 \]

\[ x \leq 2, x := 0 \quad \text{and} \quad y \geq 2 \]

\[ x = 0 \land y \geq 2 \quad \text{and} \quad y := 0 \]

Theorem [AD94] Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.
Analyzing timed automata

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

Theorem [AD94]: Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction

Efficient symbolic technics based on zones, implemented in tools
Analyzing timed automata

Theorem [AD94] Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction

Efficient symbolic technics based on zones, implemented in tools.
Analyzing timed automata

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

Theorem [AD94] Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction

Efficient symbolic technics based on zones, implemented in tools.
Analyzing timed automata

[Diagram of a timed automaton with transitions labeled by conditions on variables.]

Theorem [AD94]: Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction

Efficient symbolic technics based on zones, implemented in tools...
Analyzing timed automata

Theorem [AD94] Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction

Efficient symbolic technics based on zones, implemented in tools
Analyzing timed automata

\[
\begin{align*}
x &= 1, \\ y &= 0 \quad \Rightarrow \quad x \leq 2, \quad x := 0 \\
y \geq 2, \quad y := 0 \quad \Rightarrow \quad x = 0 \land y \geq 2
\end{align*}
\]

Theorem [AD94] Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction

Efficient symbolic technics based on zones, implemented in tools.
Analyzing timed automata

Theorem [AD94] Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction

Efficient symbolic technics based on zones, implemented in tools...
Analyzing timed automata

Theorem [AD94] Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction
Efficient symbolic technics based on zones, implemented in tools
Analyzing timed automata

Theorem [AD94] Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction

Efficient symbolic technics based on zones, implemented in tools.
Analyzing timed automata

Theorem [AD94] Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.
Analyzing timed automata

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

\[ x = 0 \land y \geq 2 \]

\[ y = 1 \]

Theorem [AD94] Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction

Efficient symbolic technics based on zones, implemented in tools.
Analyzing timed automata

Theorem [AD94] Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction

Efficient symbolic technics based on zones, implemented in tools.
Analyzing timed automata

Theorem [AD94]
Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

- Technical tool: region abstraction

Analyzing timed automata

Theorem [AD94]

Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

- Technical tool: region abstraction
- Efficient symbolic technics based on zones, implemented in tools

Technical tool: Region abstraction

The path $x = 1$ $y = 1$ - can be fired from
- cannot be fired from

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

This is a finite time-abstract bisimulation!
Technical tool: 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
Technical tool: Region abstraction

- “compatibility” between regions and constraints
- “compatibility” between regions and time elapsing
Technical tool: Region abstraction

- “compatibility” between regions and constraints
- “compatibility” between regions and time elapsing
Technical tool: Region abstraction

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

→ This is a finite time-abstract bisimulation!
Time-abstract bisimulation

This is a relation between ● and ● such that:

∀\(\delta(d)\)
\[\exists d \geq 0 \exists d' \geq 0 \delta(d')\]

... and vice-versa (swap ● and ●).

Consequence
\[(\ell, v_1)\]
\[d_1, a_1(\ell, R_1) a_1(\ell, v'_1)\]
\[d', a_1(\ell, v'_2)\]
\[d_2, a_2(\ell, R_2) a_2(\ell, v'_3)\]
\[d_3, a_3(\ell, R_3) a_3(\ell, v'_3)\]
Time-abstract bisimulation

This is a relation between • and • such that:

∀ \rightarrow a \rightarrow

... and vice-versa (swap • and •).

Consequence

(ℓ₁, v₁) \rightarrow (ℓ₁, R₁) \rightarrow (ℓ₁, v′₁)

∀ ... with vᵢ ∈ Rᵢ ∀ v′ᵢ ∈ Rᵢ ...
Time-abstract bisimulation

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

\[
\forall \quad \exists
\]
Time-abstract bisimulation

This is a relation between • and • such that:

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

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

with \( v_i \in R_i \)
Time-abstract bisimulation

This is a relation between • and • such that:

∀ (red) • → a •

∃ (green) • ← a •

∀ (red) • ← δ(d) •

∃ (green) • → δ(d') •

∀ (red) • → δ(d) •

∃ (green) • ← δ(d') •

∀ (red) • ← δ(d) •

∃ (green) • → δ(d') •
Time-abstract bisimulation

This is a relation between ● and ○ 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 ● and ○).
Time-abstract bisimulation

This is a relation between red dot and green dot such that:

\[
\begin{align*}
\forall (\ell, v) &\xrightarrow{a} (\ell', v) \\
\exists (\ell, v) &\xrightarrow{a} (\ell', v')
\end{align*}
\]

\[
\begin{align*}
\forall d \geq 0 &\xrightarrow{\delta(d)} \\
\exists d' \geq 0 &\xrightarrow{\delta(d')}
\end{align*}
\]

... and vice-versa (swap red dot and green dot).

Consequence

\[
\begin{align*}
\forall (\ell_1, v_1) &\xrightarrow{d_1, a_1} (\ell_2, v_2) \\
&\xrightarrow{d_2, a_2} (\ell_3, v_3) \\
&\xrightarrow{d_3, a_3} \cdots
\end{align*}
\]
Time-abstract bisimulation

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

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

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

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

Consequence

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

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

This is a relation between • and • such that:

∀ \overset{a}{\rightarrow} \exists \overset{a}{\rightarrow}

∀ \delta(d) \geq 0 \overset{\delta(d)}{\rightarrow}

∃ \overset{a}{\rightarrow}

∃ d' \geq 0 \overset{\delta(d')}{\rightarrow}

... and vice-versa (swap • and •).

Consequence

\forall (\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 \text{ with } v_i \in R_i

∀ v'_1 \in R_1
Time-abstract bisimulation

This is a relation between • and • such that:

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

... and vice-versa (swap • and •).

**Consequence**

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

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

\[ \forall v_1' \in R_1 \exists \quad (\ell_1, v_1') \xrightarrow{d_1',a_1} (\ell_2, v_2') \xrightarrow{d_2',a_2} (\ell_3, v_3') \xrightarrow{d_3',a_3} \ldots \text{ with } v_i' \in R_i \]
The region automaton

\[
\begin{align*}
s_0 & \xrightarrow{x>0,a} s_1 \\
y := 0 & \xrightarrow{x<1,c} s_1 \\
y < 1, a, y := 0 & \xrightarrow{x<1,c} s_1 \\
y = 1, b & \xrightarrow{y = 1, b} s_2 \\
x > 1, d & \xrightarrow{x > 1, d} s_3 \\
y < 1, a, y := 0 & \xrightarrow{x > 1, d} s_3 \\
\end{align*}
\]
The region automaton

\[ s_0 \xrightarrow{x>0,a} s_1 \xrightarrow{y:=0} s_1 \xrightarrow{x<1,c} s_2 \xrightarrow{y=1,b} s_1 \xrightarrow{y<1,a,y:=0} s_3 \xrightarrow{x>1,d} s_3 \xrightarrow{y<1,c} s_2 \xrightarrow{y=1,b} s_1 \xrightarrow{y<1,a,y:=0} s_3 \xrightarrow{x>1,d} s_3 \xrightarrow{y<1,c} s_2 \xrightarrow{y=1,b} s_1 \xrightarrow{y<1,a,y:=0} s_3 \]
The region automaton
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
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!
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,
  - ...

→ timed automata are not powerful enough!

- A possible solution: use hybrid automata

The thermostat example

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

- $T \leq 19$
- $T \geq 21$
Modelling resources in timed systems

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

→ timed automata are not powerful enough!

- A possible solution: use hybrid automata

The thermostat example

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

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

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

\[ T \leq 22 \]
\[ T \geq 21 \]

22
21
19
18
2 4 6 8 10

Time
Ok...
Ok...

Easy...
Ok...

Easy...
Ok...
Ok... but?
Ok... but?

Easy...   

Hard!

Easy...
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

**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,
  - ...

→ 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]

→ hybrid variables do not constrain the system
hybrid variables are observer variables
Modelling the task graph scheduling problem

**Processors**

- **P₁**:  
  - Initial state: idle  
  - Transition: \( x=2 \) \( \rightarrow \) done₁ \( \rightarrow \) add₁ \( \rightarrow \) idle  
  - \( (x \leq 2) \)  
  - Transition: \( x:=0 \) \( \rightarrow \) done₁ \( \rightarrow \) mult₁ \( \rightarrow \) ×  
  - \( (x \leq 3) \)

- **P₂**:  
  - Initial state: idle  
  - Transition: \( y=5 \) \( \rightarrow \) done₂ \( \rightarrow \) add₂ \( \rightarrow \) idle  
  - \( (y \leq 5) \)  
  - Transition: \( y:=0 \) \( \rightarrow \) done₂ \( \rightarrow \) mult₂ \( \rightarrow \) ×  
  - \( (y \leq 7) \)

**Tasks**

- **T₄**:  
  - Initial state: idle  
  - Transition: \( t₁ \land t₂ \) \( \rightarrow \) add₁ \( \rightarrow \) done₁  
  - \( t₄ := 1 \)

- **T₅**:  
  - Initial state: idle  
  - Transition: \( t₃ \) \( \rightarrow \) add₁ \( \rightarrow \) done₁  
  - \( t₅ := 1 \)
Modelling the task graph scheduling problem

- **Processors**

  \[ \begin{align*}
  P_1: & \quad (x \leq 2) \quad x := 0 \\
  & \quad (x \leq 3) \quad x := 0 \\
  P_2: & \quad (y \leq 5) \quad y := 0 \\
  & \quad (y \leq 7) \quad y := 0 
  \end{align*} \]

- **Tasks**

  \[ \begin{align*}
  T_4 & \quad t_1 \land t_2 \quad t_4 := 1 \\
  T_5 & \quad t_3 \quad t_5 := 1 
  \end{align*} \]

- **Modelling energy**

  \[ \begin{align*}
  P_1: & \quad (x \leq 2) \quad x := 0 \\
  & \quad (x \leq 3) \quad x := 0 \\
  P_2: & \quad (y \leq 5) \quad y := 0 \\
  & \quad (y \leq 7) \quad y := 0 
  \end{align*} \]

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

\[ \ell_0 + 5 \xrightarrow{x \leq 2, c, y := 0} \ell_1 \]

\[ (y = 0) \]

\[ \ell_1 \xrightarrow{u} \ell_2 + 10 \]
\[ x = 2, c \]

\[ \ell_2 \xrightarrow{x = 2, c} + 1 \]

\[ \ell_3 \xrightarrow{u} + 1 \]

\[ x = 2, c \]

\[ \ell_3 \]

\[ \ell_3 \xrightarrow{c + 1} + 1 \]

\[ x = 2, c \]

\[ \ell_0 \]

\[ +5 \]

\[ (y = 0) \]

\[ \ell_1 \xrightarrow{u} \ell_2 + 10 \]

\[ x = 2, c \]

\[ +1 \]


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_2 & \xrightarrow{x=2, c} \ell_3 \\
\ell_3 & \xrightarrow{c} \ell_4
\end{align*}

\[
\begin{array}{cccccccc}
\ell_0 & 1.3 & \ell_0 & c & \ell_1 & u & \ell_3 & 0.7 & \ell_3 & c & \ell_4 \\
x & 0 & 1.3 & 1.3 & 1.3 & 2 & 2 & 0.7 & 0.7 & 0.7 & 0.7 \\
y & 0 & 1.3 & 0 & 0 & 0 & 0 & 0.7 & 0.7 & 0.7 & 0.7
\end{array}
\]

\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]

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

\[
\begin{array}{c|c|c|c|c|c}
\ell_0 & 1.3 & \ell_0 & c & \ell_1 & u & \ell_2 & 0.7 & \ell_3 & c & \text{\smiley} \\
x & 0 & 1.3 & 1.3 & 1.3 & 2 & 0.7 & 0.7 & 0.7 \\
y & 0 & 1.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
\]

cost :


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

\[ \ell_0 \rightarrow \ell_1 \quad x \leq 2, c, y := 0 \]

\[ \ell_1 \rightarrow \ell_2 \rightarrow \ell_3 \rightarrow \text{smiley} \]

\[ \ell_0 \rightarrow \ell_1 \rightarrow \ell_2 \rightarrow \ell_3 \rightarrow \text{smiley} \]

\[ \begin{array}{c|c|c|c|c|c|c|c|c}
   & \ell_0 & \ell_0 & \ell_1 & \ell_1 & \ell_3 & \ell_3 & \text{smiley} \\
   \text{\( x \)} & 0 & 1.3 & 1.3 & 1.3 & 1.3 & 2 & \\
   \text{\( y \)} & 0 & 1.3 & 0 & 0 & 0 & 0.7 & \\
   \text{cost} & & & & & & 6.5 & \\
\end{array} \]

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

\[ \ell_2 \xrightarrow{x=2, c} \ell_2 \xrightarrow{x=2, c} \ell_2 \]

\[ \ell_0 + 5 \]

\[ \ell_1 \]

\[ \ell_2 \]

\[ \ell_3 \]

\[ \text{smiley} \]

\[ \text{cost} : 6.5 \]

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

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

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

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

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

<table>
<thead>
<tr>
<th>State</th>
<th>$x$</th>
<th>$y$</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell_0$</td>
<td>0</td>
<td>1.3</td>
<td>6.5</td>
</tr>
<tr>
<td>$\ell_1$</td>
<td>1.3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\ell_2$</td>
<td>1.3</td>
<td>1.3</td>
<td>0.7</td>
</tr>
<tr>
<td>$\ell_3$</td>
<td>2</td>
<td>0.7</td>
<td>0</td>
</tr>
<tr>
<td>$\smiley$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Weighted/priced timed automata \cite{ALP01,BFH+01}

\[
\begin{align*}
\ell_0 &\rightarrow \ell_1 & x \leq 2, c, y := 0 \\
\ell_1 &\rightarrow \ell_2 \quad \ell_3 & x = 2, c \\
\ell_1 &\rightarrow \ell_3 & x = 2, c
\end{align*}
\]

\[
\begin{array}{c|c|c|c|c|c|c}
\ell & l_0 & l_0 & l_1 & u & l_3 & c \\
\hline
x & 0 & 1.3 & 1.3 & 1.3 & 1.3 & 2 \\
y & 0 & 1.3 & 0 & 0 & 0.7 & \\
\hline
\text{cost} & 6.5 & + & 0 & + & 0 & + & 0.7
\end{array}
\]

\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]

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

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

\[
\begin{align*}
\ell_0 & \xrightarrow{\ell_1} (y=0) & \ell_1 & \xrightarrow{\ell_2} +10 \\ x \leq 2, c, y := 0 & \quad & & x = 2, c \\
\ell_2 & \xrightarrow{\ell_3} +1 & \ell_3 & \xrightarrow{\ell_4} +1 \\
& x = 2, c & & x = 2, c
\end{align*}
\]

\[
\begin{array}{c|ccc|ccc|c}
& \ell_0 & \ell_1 & \ell_2 & \ell_3 & \ell_4 & \ell_5 \\
x & 0 & 1.3 & 1.3 & 1.3 & 2 & \text{c} \\
y & 0 & 1.3 & 0 & 0 & 0.7 & 7 \\
cost : & 6.5 & + & 0 & + & 0 & + & 0.7 & + & 7 & = & 14.2
\end{array}
\]

\textbf{References:}
\begin{itemize}
\item \cite{ALP01} Alur, La Torre, Pappas. Optimal paths in weighted timed automata \textit{(HSCC’01)}.
\item \cite{BFH+01} Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata \textit{(HSCC’01)}.
\end{itemize}
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 😊?

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

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

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

**Question:** what is the optimal cost for reaching ☹?

\[
5t + 10(2 - t) + 1 \quad \text{or} \quad 5t + (2 - t) + 7
\]
Weighted/priced timed automata [ALP01,BFH+01]

Question: what is the optimal cost for reaching 😊?

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

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

*Question:* what is the optimal cost for reaching $\square$?

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

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

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

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

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

\(\sim\) *strategy:* leave immediately \(\ell_0\), go to \(\ell_3\), and wait there 2 t.u.

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

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

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
- Symbolic technics based on priced zones

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata (*HSCC’01*).
Technical tool: the corner-point abstraction
Technical tool: the corner-point abstraction
Technical tool: the corner-point abstraction

Concrete time successors:

Abstract time successors:

Optimal cost in the weighted graph = optimal cost in the weighted timed automaton!
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!
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
Modelling the task graph scheduling problem

**Processors**

\( P_1: \)

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

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

\[ \begin{align*}
    &\text{x} \leq 2 \\
    &\text{y} = 5 \\
\end{align*} \]

\[ \begin{align*}
    &\text{idle} \\
    &\text{add}_2 \\
    &\text{done}_2 \\
    &\text{x} = 0 \\
\end{align*} \]

\[ \begin{align*}
    &\text{idle} \\
    &\text{mult}_2 \\
    &\text{done}_2 \\
    &\text{y} = 0 \\
\end{align*} \]

\[ \begin{align*}
    &\text{y} \leq 5 \\
    &\text{y} = 7 \\
\end{align*} \]

**Tasks**

\( T_4: \)

\[ \begin{align*}
    &t_1 \land t_2 \\
    &\text{add}_i \\
    &\text{done}_i \\
    &\text{t}_4 := 1 \\
\end{align*} \]

\( T_5: \)

\[ \begin{align*}
    &t_3 \\
    &\text{add}_i \\
    &\text{done}_i \\
    &\text{t}_5 := 1 \\
\end{align*} \]

**Modelling energy**

\( P_1: \)

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

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

\[ \begin{align*}
    &\text{x} \leq 2 \\
    &\text{y} = 5 \\
\end{align*} \]

\[ \begin{align*}
    &\text{idle} \\
    &\text{add}_2 \\
    &\text{done}_2 \\
    &\text{x} = 0 \\
\end{align*} \]

\[ \begin{align*}
    &\text{idle} \\
    &\text{mult}_2 \\
    &\text{done}_2 \\
    &\text{y} = 0 \\
\end{align*} \]

\[ \begin{align*}
    &\text{y} \leq 5 \\
    &\text{y} = 7 \\
\end{align*} \]
Modelling the task graph scheduling problem

- **Processors**

  - $P_1$: 
    - $x = 2$  
    - $x = 3$  
  
    - $add_1$  
    - $mult_1$  
    
    - $done_1$  
    - $x = 0$  
  
    - $(x \leq 2)$  
    - $(x \leq 3)$  

  - $P_2$: 
    - $y = 5$  
    - $y = 7$  
  
    - $add_2$  
    - $mult_2$  
    
    - $done_2$  
    - $x = 0$  
  
    - $(y \leq 5)$  
    - $(y \leq 7)$

- **Tasks**

  - $T_4$: 
    - $t_1 \land t_2$  
    - $t_4 := 1$  
    - $add_i$  
    - $done_i$  

  - $T_5$: 
    - $t_3$  
    - $t_5 := 1$  

- **Modelling energy**

  - $P_1$: 
    - $x = 2$  
    - $x = 3$  
  
    - $add_1$  
    - $mult_1$  
    
    - $done_1$  
    - $x = 0$  
  
    - $(x \leq 2)$  
    - $(x \leq 3)$  

  - $P_2$: 
    - $y = 5$  
    - $y = 7$  
  
    - $add_2$  
    - $mult_2$  
    
    - $done_2$  
    - $x = 0$  
  
    - $(y \leq 5)$  
    - $(y \leq 7)$

- **Modelling uncertainty**

  - $P_1$: 
    - $x = 2$  
    - $x = 3$  
  
    - $add_1$  
    - $mult_1$  
    
    - $done_1$  
    - $x = 0$  
  
    - $(x \leq 2)$  
    - $(x \leq 3)$  

  - $P_2$: 
    - $y = 5$  
    - $y = 7$  
  
    - $add_2$  
    - $mult_2$  
    
    - $done_2$  
    - $x = 0$  
  
    - $(x \leq 2)$  
    - $(x \leq 3)$
Modelling the task graph scheduling problem

- **Processors**
  - $P_1$:
    - $x = 2$ (idle)
    - $x = 3$ (idle)
    - $x = 0$
    - $x = 0$
    - $y = 5$
    - $y = 7$
    - $y = 0$
    - $y = 0$
  - $P_2$:
    - $y = 5$
    - $y = 7$
    - $y = 0$

- **Tasks**
  - $T_4$:
    - $t_1 \land t_2$
    - $t_4 := 1$
    - $add_i$
    - $done_i$
  - $T_5$:
    - $t_3$
    - $t_5 := 1$
    - $add_i$
    - $done_i$

- **Modelling energy**
  - $P_1$:
    - $x = 2$ (idle)
    - $x = 3$ (idle)
    - $x = 0$
    - $x = 0$
    - $y = 5$
    - $y = 7$
    - $y = 0$
    - $y = 0$
  - $P_2$:
    - $y = 5$
    - $y = 7$
    - $y = 0$

- **Modelling uncertainty**
  - $P_1$:
    - $x \geq 1$ (idle)
    - $x \geq 1$ (idle)
    - $x = 0$
    - $x = 0$
    - $y \geq 3$
    - $y \geq 2$
    - $y = 0$
    - $y = 0$
  - $P_2$:
    - $y \geq 2$
    - $y \geq 3$
    - $y = 0$
    - $y = 0$

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 😊

(x ≤ 2) → \( x ≥ 1, u_3 \)

- \( x ≤ 1, c_1 \)

- \( x ≤ 1, u_1 \)

- \( c_2 \)

- \( x ≤ 1, c_3 \)

- \( x ≥ 2, c_4 \)

- \( x < 1, u_2, x := 0 \)
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 \((0.5, c_1)\)
  \(\leadsto\) can be preempted by \(u_2\)
- from \((\ell_2, \star)\), play \((1 - \star, c_2)\)
An example of a timed game

Rule of the game

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

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

A (memoryless) winning strategy

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

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

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

Rule of the game

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

Problems to be considered
An example of a timed game

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

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

Problems to be considered
- Does there exist a winning strategy?
An example of a timed game

Rule of the game

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

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

Problems to be considered

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

**Theorem [AMPS98,HK99]**

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

Theorem [AMPS98, HK99]

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

\[ \sim \text{classical regions are sufficient for solving such problems} \]
\[ \text{a region-closed attractor can be computed} \]
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

**Theorem [AM99,BHPR07,JT07]**

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

[AM99] Asarin, Maler. As soon as possible: time optimal control for timed automata (*HSCC’99*).


[JT07] Jurdziński, Trivedi. Reachability-time games on timed automata (*ICALP’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

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

Question: what is the optimal cost we can ensure while reaching \( \ell_1\)?

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

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

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

\[\ell_1 \xrightarrow{\ell_1} (y = 0) \]

\[\ell_2 \xrightarrow{\ell_2} x = 2, c \]

\[\ell_3 \xrightarrow{\ell_3} x = 2, c \]

\[\text{strategy: wait in } \ell_0, \text{ and when } t = 4 \]

\[\text{inf } 0 \leq t \leq 2 \max (5t + 10(2 - t) + 1, 5t + (2 - t) + 7) = 14 + \frac{1}{3} \]
A simple weighted timed game

Question: what is the optimal cost we can ensure while reaching 😊?
A simple weighted timed game

Question: what is the optimal cost we can ensure while reaching 😊?

$5t + 10(2 - t) + 1$
A simple weighted timed game

**Question:** what is the optimal cost we can ensure while reaching 😊?

\[ 5t + 10(2 - t) + 1, \quad 5t + (2 - t) + 7 \]
A simple weighted timed game

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

**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

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

**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 😊?

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

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

This topic has been fairly hot these last fifteen years...

[LMM02, ABM04, BCFL04, BBR05, BBM06, BLMR06, Rut11, HIM13, BGK+14]
Optimal reachability in weighted timed games (1)

This topic has been fairly hot these last fifteen years...

[LMM02, ABM04, BCFL04, BBR05, BBM06, BLMR06, Rut11, HIM13, BGK+14]

[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 last fifteen years...

[LMM02,ABM04,BCFL04,BBR05,BBM06,BLMR06,Rut11,HIM13,BGK+14]

[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.
Computing the optimal cost: why is that hard?

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

The cost is increased by $x_0$.
Computing the optimal cost: why is that hard?

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

In $\text{Add}^-(x)$, cost = $2x_0 + (1 - y_0) + 2$.

In $\text{Add}^+(y)$, cost = $2(1 - x_0) + y_0 + 1$.

If $y_0 < 2x_0$, player 2 chooses the first branch: cost $> 3$.

If $y_0 > 2x_0$, player 2 chooses the second branch: cost $> 3$.

If $y_0 = 2x_0$, in both branches, cost = 3; player 2 can enforce cost 3 $+$ $|y_0 - 2x_0|$.

Player 1 has a winning strategy with cost $\leq 3$ iff $y_0 = 2x_0$. 

\[ x = x_0 \]
\[ y = y_0 \]
Computing the optimal cost: why is that hard?

Given two clocks \( x \) and \( y \), we can check whether \( y = 2x \).

\[
\begin{align*}
&x = x_0 \\
y = y_0 \\
z = 0
\end{align*}
\]

\[
\begin{align*}
\text{Add}^+(x) & \quad \text{Add}^+(x) & \quad \text{Add}^-(y) \\
\text{Add}^-(x) & \quad \text{Add}^-(x) & \quad \text{Add}^+(y)
\end{align*}
\]

\[
\begin{align*}
+2 \\
+1
\end{align*}
\]

\[
\text{In } \bigcirc, \text{ cost } = 2x_0 + (1 - y_0) + 2
\]
Computing the optimal cost: why is that hard?

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

\[
\begin{align*}
\text{In } \smiley, & \quad \text{cost } = 2x_0 + (1 - y_0) + 2 \\
\text{In } \sad, & \quad \text{cost } = 2(1 - x_0) + y_0 + 1
\end{align*}
\]
Computing the optimal cost: why is that hard?

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

\[
\begin{align*}
\text{In } & \text{, cost } = 2x_0 + (1 - y_0) + 2 \\
\text{In } & \text{, cost } = 2(1 - x_0) + y_0 + 1 \\
\text{if } & y_0 < 2x_0, \text{ player 2 chooses the first branch: cost > 3}
\end{align*}
\]
Computing the optimal cost: why is that hard?

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

- In the green state, cost = $2x_0 + (1 - y_0) + 2$
- In the pink state, 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 $\bullet$, 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 $3 + \mid y_0 - 2x_0 \mid$
Computing the optimal cost: why is that hard?

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

- In ☺, cost $= 2x_0 + (1 - y_0) + 2$
- In 😞, cost $= 2(1 - x_0) + y_0 + 1$

- if $y_0 < 2x_0$, player 2 chooses the first branch: cost $> 3$
  - if $y_0 > 2x_0$, player 2 chooses the second branch: cost $> 3$
  - if $y_0 = 2x_0$, in both branches, cost $= 3$
    - $\leadsto$ 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:

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

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.

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

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

Test $y(x = 2z)$
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:

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

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:

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

The two-counter machine has a halting computation iff player 1 has a winning strategy to ensure a cost no more than 3.
Are we done?
Are we done? No!
Are we done? No!

Optimal cost is computable...

... when cost is strongly non-zeno.

\[ \text{There is } \kappa > 0 \text{ s.t. for every region cycle } C, \text{ for every real run } \varrho \text{ read on } C, \]
\[ \text{cost}(\varrho) \geq \kappa \]

Optimal cost is not computable...

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

\[ \text{There is } \kappa > 0 \text{ s.t. for every region cycle } C, \text{ for every real run } \varrho \text{ read on } C, \]
\[ \text{cost}(\varrho) \geq \kappa \text{ or cost}(\varrho) = 0 \]
Are we done? No!

**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 $\varrho$ read on $C$,

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

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

... when cost is almost-strongly non-zeno. [BJM15]

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

$$\text{cost}(\varrho) \geq \kappa \quad \text{or} \quad \text{cost}(\varrho) = 0$$
Are we done? No!

Optimal cost is computable... [AM04,BCFL04]

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

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

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 $\varrho$ 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...

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
Idea of the proof: Semi-unfolding
Idea of the proof: Semi-unfolding

Hypothesis:
\[ \text{cost} > 0 \implies \text{cost} \geq \kappa \]
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
Approximation scheme
Approximation scheme
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](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 new tool TiAMo**, developed by Maximilien Colange (LSV), using code by Ocan Sankur (IRISA)

TiAMo = Timed Automata Model-checker

Uppaal url: [http://www.uppaal.org](http://www.uppaal.org)
TiAMo url: [https://git.lsv.fr/colange/tiamo](https://git.lsv.fr/colange/tiamo)

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 new tool TiAMo**, developed by Maximilien Colange (LSV), using code by Ocan Sankur (IRISA)

**TiAMo** = **Timed Automata Model-checker**

- Timed automata:
  (time-optimal) reachability
- Weighted timed automata:
  optimal reachability

Uppaal url: [http://www.uppaal.org](http://www.uppaal.org)
TiAMo url: [https://git.lsv.fr/colange/tiamo](https://git.lsv.fr/colange/tiamo)
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 new tool TiAMo**, developed by Maximilien Colange (LSV), using code by Ocan Sankur (IRISA)

\[ \text{TiAMo} = \text{Timed Automata Model-checker} \]

- Timed automata: (time-optimal) reachability
- Weighted timed automata: optimal reachability
- Aims at being a platform for experiments (**open source!**)  
- Aims at asserting and comparing algorithms

**Uppaal url:** [http://www.uppaal.org](http://www.uppaal.org)  
**TiAMo url:** [https://git.lsv.fr/colange/tiamo](https://git.lsv.fr/colange/tiamo)  
**[BCM16]** Bouyer, Colange, Markey. Symbolic optimal reachability in weighted timed automata (**CAV’16**).
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
Example problem, objective and approach

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:
Our approach

More realistic idea: fixed set of funnels for control law + property

\[ F_1(t \in I_1), F_2(t \in I_2), a_{12} \leq t \leq b_{12}, t := c_{12} \]

\[ a_{21} \leq t \leq b_{21}, t := c_{21} \]
Our approach

More realistic idea: fixed set of funnels for control law + property

Corresponding timed automaton:
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:

$$\forall t_0 \in \mathbb{R}, \ x(t_0) \in \mathcal{F}(t_0) \Rightarrow \forall t \geq t_0, \ x(t) \in \mathcal{F}(t)$$
Example

\[
\begin{align*}
\mathcal{F}_1^1 (c_t \in I_1) \\
\mathcal{F}_2^1 (c_t \in I_2) \\
\mathcal{F}_3^1 (c_t \in I_3) \\
\mathcal{F}_2^2 (c_h \geq \Delta c_h) \\
obstacle \\
\alpha_1 \leq c_t \leq \beta_1, c_t := \gamma_1; c_h := 0 \quad c_h \geq \Delta c_h := 0 \\
\alpha_2 \leq c_t \leq \beta_2, c_t := \gamma_2, c_h := 0
\end{align*}
\]
Example

\[ F_1(\alpha_1) \leq c_t \leq F_1(\beta_1) \]
\[ F_2(\gamma_1) \leq c_t \leq F_2(\gamma_2) \]

\[ c_t: \text{positional clock}; \ c_h: \text{local clock} \]

\[ (c_t \in I_1) \quad (c_t \in I_2) \quad (c_t \in I_2) \quad (c_t \in I_3) \]
Summary
Summary

(huge) timed automata/games (with weights), with few clocks
Summary

(Huge) timed automata/games (with weights), with few clocks

← winning (optimal) strategy
Summary

(huge) timed automata/games (with weights), with few clocks

safe (good) controller ← winning (optimal) strategy
A pick-and-place example
A pick-and-place example
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?
- 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

- Overview of results concerning the optimal reachability problem in weighted timed automata and games
- Our new tool TiAMo
## Conclusion

### Summary of the talk

- Overview of results concerning the optimal reachability problem in weighted timed automata and games
- Our new tool TiAMo

### Future work

- Various theoretical issues
  - Apply further the idea of approximation
  - Robustness issues
Conclusion

Summary of the talk

- Overview of results concerning the optimal reachability problem in weighted timed automata and games
- Our new tool TiAMo

Future work

- Various theoretical issues
  - Apply further the idea of approximation
  - Robustness issues
- Continue working on TiAMo
  - Implementation of (weighted) timed games (good data structures, abstractions, etc.)
  - More applications with specific challenges (e.g. robotic problems)