Model checking and synthesis

system

property

A G (¬ B.overfull ∧ ¬ B.dried_up)

model-checking algorithm

oui/non
Model checking and synthesis

A system with two tanks (A and B) connected by a pump. The system has properties "Full" and "Empty". The goal is to synthesize a controller (A) such that the property $\text{AG}(\neg B.\text{overfull} \land \neg B.\text{dried} \text{up})$ is satisfied.

The synthesis algorithm takes as input the system model and generates a controller that satisfies the specified property.
Outline of the talk

1. Introduction: timed automata and timed games

2. Measuring other quantities in timed automata
   - Examples
   - Timed automata with stopwatches
   - Timed automata with observer variables

3. Cost-optimal strategies
   - Optimal reachability in priced timed automata
   - Optimal reachability in priced timed games

4. Conclusions and future works
Outline of the talk

1. Introduction: timed automata and timed games

2. Measuring other quantities in timed automata
   - Examples
   - Timed automata with stopwatches
   - Timed automata with observer variables

3. Cost-optimal strategies
   - Optimal reachability in priced timed automata
   - Optimal reachability in priced timed games

4. Conclusions and future works
Reasoning about real-time systems

Definition
A timed automaton is made of a transition system, a set of clocks, timing constraints on states and transitions.

Example (A computer mouse)

- **Idle state**
  - Transition on **left_button**
    - Action: `left_click!`
  - Transition on **left_button**
    - Action: `left_double_click!`
- **Left state**
  - Transition on any input
- **Right state**
  - Transition on **right_button**
    - Action: `right_click!`
  - Transition on **right_button**
    - Action: `right_double_click!`
Timed automata A **timed automaton** is made of

- a transition system,

**Example (A computer mouse)**

- State: **idle**
  - Transition from **left** on **left_button**?
  - Transition to **right** on **right_button**?

- **Left buttons**:
  - **left_click!** when **x ≤ 300**
  - **left_double_click!** when **x = 300**

- **Right buttons**:
  - **right_click!** when **x ≤ 300**
  - **right_double_click!** when **x = 300**
Reasoning about real-time systems

**Definition**

Timed automata A *timed automaton* is made of

- a transition system,
- a set of clocks,

**Example (A computer mouse)**

- **left**
  - left_button?
  - left_click!
  - left_double_click!

- **idle**
  - right_button?
  - right_click!
  - left_button?
  - left_double_click!

- **right**
  - right_button?
  - right_double_click!
Reasoning about real-time systems

Definition
Timed automata A **timed automaton** is made of
- a transition system,
- a set of clocks,
- timing constraints on states and transitions.

Example (A computer mouse)

- **left**
  - $x \leq 300$
  - $x = 300$
  - left_button?
  - left_double_click!

- **idle**
  - $x = 300$
  - left_button?
  - left_double_click!

- **right**
  - $x = 300$
  - right_button?
  - right_double_click!

- **right_button**
  - $x \leq 300$

- **left_button**
  - $x := 0$

- **left_click**
  - $x \leq 300$

- **right_click**
  - $x \leq 300$
Continuous-time semantics

Example

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

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

\[ y \geq 2, \quad y := 0 \]

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

\[ y = 0 \]

\[ 0 \leq x \leq 2, \quad y \geq 2 \]

Theorem (\cite{AD90, ACD93, ...})

Reachability in timed automata is decidable (as well as many other important properties).

\cite{AD90} Alur and Dill. Automata For Modeling Real-Time Systems. ICALP 1990.

\cite{ACD93} Alur, Courcoubetis, Dill. Model-Checking in Dense Real-Time. Inf. & Comp., 1993.
Continuous-time semantics

Example

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

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

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

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

Theorem ([AD90, ACD93, ...])

Reachability in timed automata is decidable (as well as many other important properties).


Continuous-time semantics

Example

Theorem ([AD90, ACD93, ...])
Reachability in timed automata is decidable (as well as many other important properties).

Continuous-time semantics

Example

Theorem ([AD90, ACD93, ...]) Reachability in timed automata is decidable (as well as many other important properties).


Continuous-time semantics

Example

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

Theorem (\[\text{AD90, ACD93, ...}\])

Reachability in timed automata is decidable (as well as many other important properties).


Continuous-time semantics

Example

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

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

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

Theorem (\cite{AD90,ACD93, ...})
Reachability in timed automata is decidable (as well as many other important properties).

\cite{AD90} Alur and Dill. Automata For Modeling Real-Time Systems. ICALP 1990.
\cite{ACD93} Alur, Courcoubetis, Dill. Model-Checking in Dense Real-Time. Inf. & Comp., 1993.
Continuous-time semantics

Example

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

Theorem ([AD90, ACD93, ...])
Reachability in timed automata is decidable (as well as many other important properties).

Continuous-time semantics

Example

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

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

\[
y \geq 2, \quad y := 0
\]

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

Theorem ([AD90, ACD93, ...])

Reachability in timed automata is decidable (as well as many other important properties).


Continuous-time semantics

Example

Reachability in timed automata is decidable (as well as many other important properties).

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

Continuous-time semantics

Example

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

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

\[ y \geq 2, \quad y := 0 \]

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

Theorem (AD90, ACD93, ...)

Reachability in timed automata is decidable (as well as many other important properties).


Continuous-time semantics

Example

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

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

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

Theorem (\cite{AD90,ACD93, ...})
Reachability in timed automata is decidable (as well as many other important properties).

\[ \text{Theorem (AD90, ACD93, ...)} \]
Reachability in timed automata is decidable (as well as many other important properties).

\[ \text{[AD90] Alur and Dill. Automata For Modeling Real-Time Systems. ICALP 1990.} \]
\[ \text{[ACD93] Alur, Courcoubetis, Dill. Model-Checking in Dense Real-Time. Inf. & Comp., 1993.} \]
Continuous-time semantics

Example

Theorem ([AD90, ACD93, ...])

Reachability in timed automata is decidable (as well as many other important properties).

Region automaton

Theorem

Reachability checking in timed automata is PSPACE-complete.
Region automaton

Theorem: Reachability checking in timed automata is \( \text{PSPACE} \)-complete.
Reachability checking in timed automata is PSPACE-complete.
Timed games

Definition (Timed games)
A **timed game** is made of
- a timed automaton;

Example

\[
\begin{align*}
l_0 \quad & (x \leq 2) \quad x \geq 1 \\
l_1 \quad & x < 1 \quad x = 0 \\
l_2 \quad & x < 1 \\
l_3 \quad & x \geq 2 \\
\end{align*}
\]

- In \( l_0 \): wait 0 goto \( l_3 \).
- In \( l_1 \): wait until \( x = 2 \) goto \( l_3 \).
- In \( l_2 \): wait until \( x = 1 \) goto \( l_3 \).
- In \( l_3 \): wait until \( x = 1 \) goto \( l_1 \).
Definition (Timed games)

A timed game is made of
- a timed automaton;
- a partition between controllable and uncontrollable transitions.

Example

A memoryless strategy in ($\ell_0$, $x=0$): wait 0 goto $\ell_3$.

5 goto in ($\ell_1$, $x$): wait until $x=2$ goto $\ell_3$.

in ($\ell_2$, $x\leq 1$): wait until $x=1$ goto $\ell_3$.
Timed games

Definition (Timed games)
A timed game is made of
- a timed automaton;
- a partition between controllable and uncontrollable transitions.

Example

A memoryless strategy
- in $(\ell_0, x = 0)$: wait 0.5, goto $\ell_3$
- in $(\ell_1, x)$: wait until $x = 2$, goto $\ell_3$
- in $(\ell_2, x \leq 1)$: wait until $x = 1$, goto $\ell_3$
- in $(\ell_3, x \leq 1)$: wait until $x = 1$, goto $\ell_1$
Timed games

Theorem ([AMPS98])

Deciding the winner in a timed game (e.g. for reachability objectives) is EXPTIME-complete.

Proof

Timed games

Theorem ([AMPS98])

Deciding the winner in a timed game (e.g. for reachability objectives) is EXPTIME-complete.

Proof

Timed games

Theorem ([AMPS98])

Deciding the winner in a timed game (e.g. for reachability objectives) is EXPTIME-complete.

Proof

\[ 1 \leq x \leq 2 \land y \geq 1 \]

\[ x = 1 \land 1 \leq y \leq 2 \]

Timed games

Theorem ([AMPS98])

Deciding the winner in a timed game (e.g. for reachability objectives) is EXPTIME-complete.

Proof

Theorem ([AMPS98])

Deciding the winner in a timed game (e.g. for reachability objectives) is EXPTIME-complete.

Proof

- regions are sufficient;
- the computation terminates.

Outline of the talk

1 Introduction: timed automata and timed games

2 Measuring other quantities in timed automata
   - Examples
   - Timed automata with stopwatches
   - Timed automata with observer variables

3 Cost-optimal strategies
   - Optimal reachability in priced timed automata
   - Optimal reachability in priced timed games

4 Conclusions and future works
Example: level of liquid in a tank

The aim is then to keep the level of resources within given bounds.
Compute $D \times (C \times (A+B)) + (A+B) + (C \times D)$ using two processors:

**$P_1$ (fast):**

<table>
<thead>
<tr>
<th></th>
<th>time</th>
<th>energy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+</td>
<td>2 ps</td>
</tr>
<tr>
<td></td>
<td>$\times$</td>
<td>3 ps</td>
</tr>
<tr>
<td>idle</td>
<td></td>
<td>10 Watt</td>
</tr>
<tr>
<td>in use</td>
<td></td>
<td>90 Watts</td>
</tr>
</tbody>
</table>

**$P_2$ (slow):**

<table>
<thead>
<tr>
<th></th>
<th>time</th>
<th>energy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+</td>
<td>5 ps</td>
</tr>
<tr>
<td></td>
<td>$\times$</td>
<td>7 ps</td>
</tr>
<tr>
<td>idle</td>
<td></td>
<td>20 Watts</td>
</tr>
<tr>
<td>in use</td>
<td></td>
<td>30 Watts</td>
</tr>
</tbody>
</table>

**Task Graph Scheduling:**

- $T_1$: $A + B$
- $T_2$: $C \times D$
- $T_3$: $A + B$
- $T_4$: $T_1 + T_2$
- $T_5$: $C \times T_3$
- $T_6$: $T_4 + T_5$
Example: task graph scheduling

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>$\times$</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>time</th>
<th>5 picoseconds</th>
</tr>
</thead>
<tbody>
<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>

Schedule:

$S_{ch_1}$:

- $P_1$:
  - $T_1$ (2 picoseconds)
  - $T_2$ (3 picoseconds)
  - $T_4$ (5 picoseconds)
  - $T_5$ (7 picoseconds)
  - $T_6$ (9 picoseconds)

$S_{ch_2}$:

- $P_2$:
  - $T_1$ (5 picoseconds)
  - $T_2$ (7 picoseconds)
  - $T_4$ (9 picoseconds)
  - $T_5$ (11 picoseconds)
  - $T_6$ (13 picoseconds)

- $T_1$ takes 13 picoseconds and 1.37 nanojoules.
- $T_2$ takes 12 picoseconds and 1.39 nanojoules.
- $T_3$ takes 19 picoseconds and 1.32 nanojoules.
Example: task graph scheduling

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

$P_1$ (fast):

<table>
<thead>
<tr>
<th></th>
<th>time</th>
<th>energy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+ 2 ps</td>
<td>idle 10 Watt</td>
</tr>
<tr>
<td></td>
<td>× 3 ps</td>
<td>in use 90 Watts</td>
</tr>
</tbody>
</table>

$P_2$ (slow):

<table>
<thead>
<tr>
<th></th>
<th>time</th>
<th>energy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+ 5 ps</td>
<td>idle 20 Watts</td>
</tr>
<tr>
<td></td>
<td>× 7 ps</td>
<td>in use 30 Watts</td>
</tr>
</tbody>
</table>

Task graph:

```
T_1 → T_2 → T_3 → T_4 → T_5 → T_6
```

Sched 1:

<table>
<thead>
<tr>
<th>Sch_1</th>
<th>P_1</th>
<th>P_2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T_1</td>
<td>T_2</td>
</tr>
<tr>
<td></td>
<td>T_3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T_5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T_6</td>
<td></td>
</tr>
</tbody>
</table>

Sched 2:

<table>
<thead>
<tr>
<th>Sch_2</th>
<th>P_1</th>
<th>P_2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T_1</td>
<td>T_2</td>
</tr>
<tr>
<td></td>
<td>T_3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T_5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T_4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T_6</td>
<td></td>
</tr>
</tbody>
</table>

For each schedule and processor:

- **13** picoseconds
- **1.37** nanojoules

- **12** picoseconds
- **1.39** nanojoules
Example: task graph scheduling

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

\[ \begin{align*}
P_1 \text{ (fast)}: \\
&\text{time} \\
+ & 2 \text{ picoseconds} \\
\times & 3 \text{ picoseconds} \\
\hline
& \text{energy} \\
\text{idle} & 10 \text{ Watt} \\
\text{in use} & 90 \text{ Watts} \\
\hline
P_2 \text{ (slow)}: \\
&\text{time} \\
+ & 5 \text{ picoseconds} \\
\times & 7 \text{ picoseconds} \\
\hline
& \text{energy} \\
\text{idle} & 20 \text{ Watts} \\
\text{in use} & 30 \text{ Watts} \\
\hline
\end{align*} \]

### Schedule 1

- \( P_1 \): \( T_2 \) (2 picoseconds)
- \( P_2 \): \( T_1 \) (3 picoseconds)
- \( T_3 \) (13 picoseconds, 1.37 nanojoules)
- \( T_5 \) (13 picoseconds, 1.37 nanojoules)
- \( T_6 \) (19 picoseconds, 1.32 nanojoules)

### Schedule 2

- \( P_1 \): \( T_1 \) to \( T_5 \)
- \( P_2 \): \( T_3 \) to \( T_6 \)
- \( T_4 \) (12 picoseconds, 1.39 nanojoules)

### Schedule 3

- \( P_1 \): \( T_1 \) to \( T_5 \)
- \( P_2 \): \( T_3 \) to \( T_6 \)
- \( T_4 \) (19 picoseconds, 1.32 nanojoules)
Timed automata with stopwatches

Definition

A stopwatch automaton is made of

- a timed automaton;

Example

```
idle  work
  \text{start, } x:=0 \quad \text{start, } x:=0

wait  exec  paused
 \dot{x} = 0 \quad x=1, \text{stop}
```

\text{resume}
Timed automata with stopwatches

Definition

A stopwatch automaton is made of
- a timed automaton;
- for each location, a set of clocks that are stopped in that location.

Example

idle → work

start

work → idle

stop

wait → exec

\[ x = 1, \text{stop} \]

exec → wait

\[ \dot{x} = 0 \]

wait → exec

pause

exec → paused

\[ x = 0 \]

paused → exec

resume
Reachability in stopwatch automata

Theorem ([Čer92])

Reachability in stopwatch automata is undecidable.

Reachability in stopwatch automata

Theorem ([Čer92])

*Reachability in stopwatch automata is undecidable.*

**Proof**

Encode a two-counter machine using four stopwatches:

\[ c_1 = a_1 - b_1 \quad \quad c_2 = a_2 - b_2 \]

Already undecidable for one stopwatch and no diagonal constraints.

Reachability in stopwatch automata

Theorem ([Čer92])

Reachability in stopwatch automata is undecidable.

Proof

Encode a two-counter machine using four stopwatches:

\[
\begin{align*}
  c_1 &= a_1 - b_1 \\
  c_2 &= a_2 - b_2
\end{align*}
\]

Reachability in stopwatch automata

Theorem ([Čer92])

Reachability in stopwatch automata is undecidable.

Proof

Encode a two-counter machine using four stopwatches:

\[ c_1 = a_1 - b_1 \]

\[ c_2 = a_2 - b_2 \]

Reachability in stopwatch automata

Theorem ([Čer92])
Reachability in stopwatch automata is undecidable.

Proof
Encode a two-counter machine using four stopwatches:
\[ c_1 = a_1 - b_1 \]
\[ c_2 = a_2 - b_2 \]

Already undecidable for one stopwatch and no diagonal constraints.

Priced timed automata

Definition ([KPSY99, ALP01, BFH+01])

A priced timed automaton is made of
- a timed automaton;

Example

Priced timed automata

Definition ([KPSY99, ALP01, BFH+01])

A priced timed automaton is made of
- a timed automaton;
- the price of each transition and location.

Example

\[\begin{align*}
&\text{\(x := 0\)} \\
\end{align*}\]

Priced timed automata

**Definition ([KPSY99,ALP01,BFH+01])**

A priced timed automaton is made of:
- a timed automaton;
- the price of each transition and location.

**Example**

\[ x := 0 \]

\[ x = 1 \]

Priced timed automata

Definition ([KPSY99, ALP01, BFH⁺01])

A priced timed automaton is made of
- a timed automaton;
- the price of each transition and location.

Example

```
-3 ——— x:=0 ——— +6
|      |      |      |
|      |      |      |
| -6 ——— ——— x=1 ——— +2
|      |      |      |
|      |      |      |
| -3 ——— ——— 1/6 ——— -3
```

Priced timed automata

Definition ([KPSY99, ALP01, BFH+01])

A priced timed automaton is made of:
- a timed automaton;
- the price of each transition and location.

Example

**Priced timed automata**

**Definition ([KPSY99, ALP01, BFH+01])**

A priced timed automaton is made of
- a timed automaton;
- the price of each transition and location.

**Example**

![Diagram of a priced timed automaton]

Priced timed automata

Definition ([KPSY99,ALP01,BFH⁺01])

A priced timed automaton is made of
- a timed automaton;
- the price of each transition and location.

Example

Priced timed automata

Definition ([KPSY99, ALP01, BFH+01])

A priced timed automaton is made of
- a timed automaton;
- the price of each transition and location.

Example

Priced timed automata

Definition ([KPSY99, ALP01, BFH+01])

A priced timed automaton is made of
- a timed automaton;
- the price of each transition and location.

Example

\[ \begin{align*}
\text{x:=0} & \quad -3 \\
\text{\textcolor{red}{-6}} & \quad \text{\textcolor{red}{x=1}} \\
\text{\textcolor{red}{+2}} & \quad \text{+6} \\
\rightarrow & \quad \frac{1}{6} \\
\rightarrow & \quad \frac{1}{2} \\
\rightarrow & \quad -1 \\
\rightarrow & \quad \frac{1}{3} \\
\rightarrow & \quad -2 \\
\rightarrow & \quad \frac{1}{4} \\
\rightarrow & \quad +2
\end{align*} \]

Priced timed automata

Definition ([KPSY99, ALP01, BFH+01])

A priced timed automaton is made of
- a timed automaton;
- the price of each transition and location.

Example

Example: task graph scheduling

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 Watts, in use 90 Watts

$P_2$ (slow):
- **Time**: 5 picoseconds × 7 picoseconds
- **Energy**: idle 20 Watts, in use 30 Watts

### Scheduling

#### Sch1
- $T_2$ on $P_1$
- $T_1$ on $P_2$
- $T_3$ on $P_1$
- $T_4$ on $P_2$
- $T_5$ on $P_1$
- $T_6$ on $P_2$

**Time**: 13 picoseconds
**Energy**: 1.37 nanojoules

#### Sch2
- $T_1$ on $P_1$
- $T_2$ on $P_2$
- $T_3$ on $P_1$
- $T_4$ on $P_2$
- $T_5$ on $P_1$
- $T_6$ on $P_2$

**Time**: 12 picoseconds
**Energy**: 1.39 nanojoules

#### Sch3
- $T_1$ on $P_1$
- $T_2$ on $P_2$
- $T_3$ on $P_1$
- $T_4$ on $P_2$
- $T_5$ on $P_1$
- $T_6$ on $P_2$

**Time**: 19 picoseconds
**Energy**: 1.32 nanojoules
Modelling the task graph scheduling problem

Processors:

\[ + \quad \dot{c} = 90 \]
\[ x \leq 2 \]
\[ \text{add}_1 \]
\[ x := 0 \]
\[ \text{done}_1 \]

\[ \text{idle} \quad \dot{c} = 10 \]

\[ \times \quad \dot{c} = 90 \]
\[ x \leq 3 \]
\[ \text{mul}_1 \]
\[ x := 0 \]
\[ \text{done}_1 \]
Modelling the task graph scheduling problem

Processors:

\[ + \quad \dot{c} = 90 \]
\[ x \leq 2 \]
\[ \text{add}_1 \quad x := 0 \]
\[ \text{done}_1 \]

\[ \text{idle} \quad \dot{c} = 10 \]

\[ \times \quad \dot{c} = 90 \]
\[ x \leq 3 \]
\[ \text{mul}_1 \quad x := 0 \]
\[ \text{done}_1 \]

\[ + \quad \dot{c} = 30 \]
\[ x \leq 5 \]
\[ \text{add}_2 \quad x := 0 \]
\[ \text{done}_2 \]

\[ \text{idle} \quad \dot{c} = 20 \]

\[ \times \quad \dot{c} = 30 \]
\[ x \leq 7 \]
\[ \text{mul}_2 \quad x := 0 \]
\[ \text{done}_2 \]
Modelling the task graph scheduling problem

**Processors:**

- Processor 1: +
  - $c = 90$
  - $x \leq 2$
  - State: done$_1$
  - Action: add$_1$
  - $x := 0$

- Processor 2: $	imes$
  - $c = 90$
  - $x \leq 3$
  - State: mul$_1$
  - $x := 0$

- Processor 3: +
  - $c = 30$
  - $x \leq 5$
  - State: done$_2$
  - Action: add$_2$
  - $x := 0$

- Processor 4: $	imes$
  - $c = 30$
  - $x \leq 7$
  - State: mul$_2$
  - $x := 0$

**Tasks:**

- $F_4$
  - $t_4 := 1$
- $T_4$
  - $t_1 \land t_2$

- $t_1 \land t_2$
- add$_1$
- done$_1$
- done$_2$
- $t_4 := 1$

Task graph:

1. $t_1 \land t_2$ to add$_1$
2. add$_1$ to done$_1$
3. done$_1$ to $F_4$
4. $t_1 \land t_2$ to add$_2$
5. add$_2$ to done$_2$
6. done$_2$ to $F_4$
Outline of the talk

1. Introduction: timed automata and timed games

2. Measuring other quantities in timed automata
   - Examples
   - Timed automata with stopwatches
   - Timed automata with observer variables

3. Cost-optimal strategies
   - Optimal reachability in priced timed automata
   - Optimal reachability in priced timed games

4. Conclusions and future works
Cost-optimal reachability in priced timed automata

Example

\[ \dot{p} = 5 \]
\[ y := 0 \]
\[ x \leq 2 \]

\[ \dot{p} = 6 \]
\[ y = 0 \]
\[ x \geq 3 \]
\[ p += 1 \]

\[ \dot{p} = 3 \]
\[ x \geq 3 \]
\[ p += 9 \]

The optimal schedule consists in waiting 2 time units in; going through.
Cost-optimal reachability in priced timed automata

Example

\[ \dot{p} = 5 \]
\[ x \leq 2 \quad y := 0 \]
\[ y = 0 \]

\[ \dot{p} = 6 \]
\[ x \geq 3 \quad p += 1 \]

\[ \dot{p} = 3 \]
\[ x \geq 3 \quad p += 9 \]

Minimal cost for reaching \( \ddot{\smiley} \):

\[ \text{inf}_{0 \leq t \leq 2} \min \left( 5t + 6(3-t) + 1, 5t + 3(3-t) + 9 \right) = 18 \]
Minimal cost for reaching $\smiley$:

$$5t + 6(3 - t) + 1$$
Example

Cost-optimal reachability in priced timed automata

Minimal cost for reaching $\bigcirc$:

$$5t + 6(3 - t) + 1$$
$$5t + 3(3 - t) + 9$$
Example

\[ \dot{p} = 5 \]
\[ y := 0 \]
\[ x \leq 2 \]

\[ \dot{p} = 6 \]
\[ y = 0 \]

\[ \dot{p} = 3 \]

\[ x \geq 3 \]
\[ p += 1 \]

\[ x \geq 3 \]
\[ p += 9 \]

Minimal cost for reaching \( \mathbb{Q} \):

\[
\min \left( \frac{5t + 6(3 - t) + 1}{5t + 3(3 - t) + 9} \right)
\]

The optimal schedule consists in waiting 2 time units in; going through.
Cost-optimal reachability in priced timed automata

Example

\[ \dot{p} = 5 \quad x \leq 2 \quad y := 0 \]
\[ \dot{p} = 6 \quad y = 0 \]
\[ \dot{p} = 3 \quad x \geq 3 \]
\[ x \geq 3 \quad p += 9 \]
\[ x \geq 3 \quad p += 1 \]

Minimal cost for reaching \( \bigcirc \):

\[
\inf_{0 \leq t \leq 2} \min \left( \frac{5t + 6(3 - t) + 1}{5t + 3(3 - t) + 9} \right)
\]

The optimal schedule consists in waiting 2 time units, then going through.
Cost-optimal reachability in priced timed automata

Example

Minimal cost for reaching \( \odot \):

\[
\inf_{0 \leq t \leq 2} \min \left( \frac{5t + 6(3 - t) + 1}{5t + 3(3 - t) + 9} \right) = 17
\]
Cost-optimal reachability in priced timed automata

Example

\[ \dot{p} = 5 \]
\[ x \leq 2 \]
\[ y := 0 \]

\[ \dot{p} = 6 \]
\[ x \geq 3 \]
\[ p += 1 \]

\[ \dot{p} = 3 \]
\[ x \geq 3 \]
\[ p += 9 \]

Minimal cost for reaching \( \mathbf{?option} \):

\[
\inf_{0 \leq t \leq 2} \min \left( \frac{5t + 6(3 - t) + 1}{5t + 3(3 - t) + 9} \right) = 17
\]

The optimal \textit{schedule} consists in:

- waiting 2 time units in \( \mathbf{\text{option}} \); 
- going through \( \mathbf{\text{option}} \).
Cost-optimal reachability in priced timed automata

Theorem ([BBBR07])

Optimal reachability in priced timed automata is PSPACE-complete.

Theorem ([BBBR07])

Optimal reachability in priced timed automata is PSPACE-complete.

Proof

- Regions are not precise enough;

Cost-optimal reachability in priced timed automata

Theorem ([BBBR07])

Optimal reachability in priced timed automata is PSPACE-complete.

Proof

- Regions are not precise enough;
- Use regions with corner-points:
Cost-optimal reachability in priced timed automata

Theorem ([BBBR07])

Optimal reachability in priced timed automata is PSPACE-complete.

Proof

- Regions are not precise enough;
- Use regions with corner-points:

Cost-optimal reachability in priced timed automata

Theorem ([BBBR07])

Optimal reachability in priced timed automata is PSPACE-complete.

Proof

- Regions are not precise enough;
- Use regions with corner-points:

Cost-optimal reachability in priced timed automata

Theorem ([BBBR07])

**Optimal reachability in priced timed automata is PSPACE-complete.**

Proof

- Regions are not precise enough;
- Use regions with corner-points:

Cost-optimal reachability in priced timed automata

Theorem ([BBBR07])

Optimal reachability in priced timed automata is PSPACE-complete.

Proof

- optimal schedule as a linear programming problem:

```
\begin{align*}
  & t_1 \\
\end{align*}
```

Cost-optimal reachability in priced timed automata

Theorem ([BBBR07])

Optimal reachability in priced timed automata is PSPACE-complete.

Proof

- optimal schedule as a linear programming problem:

\[\begin{align*}
t_1 & \rightarrow t_2 \\
x \leq 2 & \rightarrow t_3 \\
t_4 & \rightarrow t_5 \\
t_1 + t_2 & \leq 2
\end{align*}\]

Cost-optimal reachability in priced timed automata

**Theorem ([BBBR07])**

Optimal reachability in priced timed automata is PSPACE-complete.

**Proof**

- optimal schedule as a linear programming problem:

  \[
  \begin{align*}
  t_1 &: y := 0 \\
  t_2 &: x \leq 2 \\
  t_3 &: t_4 \geq 3 \\
  t_5 &:
  \end{align*}
  \]

  Minimize \[ \sum c_i \cdot t_i + C_{\text{disc}} \]

  Subject to:

  \[
  \begin{align*}
  t_1 + t_2 &\leq 2 \\
  t_2 + t_3 + t_4 &\geq 3 \\
  \end{align*}
  \]

  Infimum over bounded zone reached at a point on the frontier, with integer coordinates.

Cost-optimal reachability in priced timed automata

Theorem ([BBBR07])

Optimal reachability in priced timed automata is PSPACE-complete.

Proof

- optimal schedule as a linear programming problem:

Minimize

\[
\sum_i c_i \cdot t_i + C_{\text{disc}}
\]

\[t_1 + t_2 \leq 2\]
\[t_2 + t_3 + t_4 \geq 3\]
Cost-optimal reachability in priced timed automata

**Theorem ([BBBR07])**

*Optimal reachability in priced timed automata is PSPACE-complete.*

**Proof**

- optimal schedule as a linear programming problem:

  Minimize
  \[ \sum_i c_i \cdot t_i + C_{\text{disc}} \]

  subject to
  \[
  t_1 + t_2 \leq 2 \\
  t_2 + t_3 + t_4 \geq 3
  \]

  infimum over bounded zone reached at a point on the frontier, with integer coordinates.

Theorem ([BBBR07])

Optimal reachability in priced timed automata is PSPACE-complete.

Proof

optimal schedule as a linear programming problem:

$$\forall \pi. \exists \pi_{cp}. \text{cost}(\pi_{cp}) \leq \text{cost}(\pi).$$
Cost-optimal reachability in priced timed automata

Theorem ([BBBR07])

Optimal reachability in priced timed automata is PSPACE-complete.

Proof

- optimal schedule as a linear programming problem:

  \[ \forall \pi. \exists \pi_{cp}. \ cost(\pi_{cp}) \leq \ cost(\pi). \]

- approximate path in corner-point abstraction by a real run:

  \[ \forall \pi_{cp}. \exists \pi. \ cost(\pi) \leq \ cost(\pi_{cp}) + \epsilon. \]

Outline of the talk

1. Introduction: timed automata and timed games
2. Measuring other quantities in timed automata
   - Examples
   - Timed automata with stopwatches
   - Timed automata with observer variables
3. Cost-optimal strategies
   - Optimal reachability in priced timed automata
   - Optimal reachability in priced timed games
4. Conclusions and future works
Example: task graph scheduling

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

\[\begin{array}{|c|c|}\hline
P_1 & P_2 \\
\hline
\text{time} & \text{time} \\
\text{+} & \text{+} \\
2 \text{ picoseconds} & 5 \text{ picoseconds} \\
\times & \times \\
3 \text{ picoseconds} & 7 \text{ picoseconds} \\
\hline
\text{energy} & \text{energy} \\
\text{idle} & \text{idle} \\
10 \text{ Watt} & 20 \text{ Watts} \\
in \text{ use} & \text{in use} \\
90 \text{ Watts} & 30 \text{ Watts} \\
\hline
\end{array}\]

\[\begin{array}{|c|c|c|c|c|c|}
\hline
& T_1 & T_2 & T_3 & T_4 & T_5 & T_6 \\
\hline
\text{Sch}_1 \ P_1 & \text{Sch}_1 \ P_2 & \text{Sch}_2 \ P_1 & \text{Sch}_2 \ P_2 & \text{Sch}_3 \ P_1 & \text{Sch}_3 \ P_2 \\
\hline
\text{time} & 13 \text{ picoseconds} & 12 \text{ picoseconds} & 19 \text{ picoseconds} \\
\text{energy} & 1.37 \text{ nanojoules} & 1.39 \text{ nanojoules} & 1.32 \text{ nanojoules} \\
\hline
\end{array}\]
Using games to model uncertainty over delays

Processors with exact delays:

1. \( \dot{c} = 90 \) with \( x \leq 2 \):
   - \( x = 2 \): done\(_1\)
   - \( x := 0 \): add\(_1\)

2. \( \dot{c} = 10 \) (idle):
   - \( x = 3 \): done\(_1\)
   - \( x := 0 \): mul\(_1\)

3. \( \dot{c} = 90 \) with \( x \leq 3 \):
   - \( x \geq 2 \): done\(_1\)
   - \( x \geq 3 \): done\(_1\)
Cost-optimal reachability in priced timed games

Using games to model uncertainty over delays

Processors with exact delays:

\[ \dot{c} = 90 \quad x \leq 2 \]
\[ \dot{c} = 10 \quad x = 0 \]
\[ \dot{c} = 90 \quad x \leq 3 \]

Processors with approximative delays:

\[ \dot{c} = 90 \quad x \geq 2 \]
\[ \dot{c} = 10 \quad x = 0 \]
\[ \dot{c} = 90 \quad x \leq 4 \]
Cost-optimal reachability in priced timed games

Example

\[ \dot{p} = 5 \]
\[ y := 0 \]
\[ \dot{p} = 6 \]
\[ \dot{p} = 3 \]
\[ x \leq 2 \]
\[ y = 0 \]
\[ x \geq 3 \]
\[ p \geq 1 \]
\[ p \geq 9 \]

Minimal cost for reaching \( x \geq 3 \):
\[
\inf_{0 \leq t \leq 2} \max(5t + 6(3 - t) + 1, 5t + 3(3 - t) + 9) = 18.66
\]
(with \( t_{\text{opt}} = 1.3 \))
Cost-optimal reachability in priced timed games

Example

\[
\dot{p} = 5
\]

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

\[
\dot{p} = 6
\]

\[
x \geq 3
\]

\[
p += 1
\]

\[
\dot{p} = 3
\]

\[
x \geq 3
\]

\[
p += 9
\]

Minimal cost for reaching 😊:

\[
\inf_{0 \leq t \leq 2} \max(5t + 6(3-t) + 1, 5t + 3(3-t) + 9) = 18.66
\]

(with \( t_{\text{opt}} = 1.3 \))
Cost-optimal reachability in priced timed games

Example

\[ \dot{p}=5 \]
\[ y:=0 \]
\[ x \leq 2 \]

\[ \dot{y}=0 \]
\[ \dot{p}=6 \]
\[ \dot{p}=3 \]
\[ x \geq 3 \]

\[ x \geq 3 \]
\[ p+=1 \]
\[ p+=9 \]

Minimal cost for reaching ☺:

\[ \inf_{0 \leq t \leq 2} \max(5t + 6(3 - t) + 1, 5t + 3(3 - t) + 9) = 18.66 \]

(with \( t_{opt} = \frac{1}{3} \))
Cost-optimal reachability in priced timed games

Example

\[
\begin{align*}
\dot{p} &= 5 \quad (x \leq 2, y := 0) \\
\dot{p} &= 6 \quad (y = 0) \\
\dot{p} &= 3 \quad (x \geq 3, p += 1) \\
&\quad (x \geq 3, p += 9)
\end{align*}
\]

Minimal cost for reaching :)

\[
\begin{align*}
5t + 6(3 - t) + 1 \\
5t + 3(3 - t) + 9
\end{align*}
\]
Cost-optimal reachability in priced timed games

Example

\[ \dot{y} = 0 \]
\[ x \leq 2 \]
\[ y := 0 \]

\[ \dot{p} = 5 \]

\[ y = 0 \]

\[ \dot{p} = 3 \]
\[ x \geq 3 \]
\[ p += 9 \]

\[ \dot{p} = 6 \]
\[ x \geq 3 \]
\[ p += 1 \]

Minimal cost for reaching \( \smile \):

\[
\max \left( \frac{5t + 6(3 - t) + 1}{5t + 3(3 - t) + 9} \right)
\]

\[
\begin{align*}
\inf_{0 \leq t \leq 2} & \max \left( 5t + 6(3 - t) + 1, 5t + 3(3 - t) + 9 \right) \\
& = 18.66 \\
& \text{(with } t_{\text{opt}} = 1.3) \\
\end{align*}
\]
Cost-optimal reachability in priced timed games

Example

\[
\begin{align*}
\dot{p} &= 5 \\
y &= 0 \\
\dot{p} &= 6 \\
x &\leq 2 \\
y &= 0 \\
\dot{p} &= 3 \\
x &\geq 3 \\
p &= 9
\end{align*}
\]

Minimal cost for reaching \( \smiley \):

\[
\inf_{0 \leq t \leq 2} \max \left( \frac{5t + 6(3 - t) + 1}{5t + 3(3 - t) + 9} \right) = 18.66
\]
Cost-optimal reachability in priced timed games

Example

\[ \dot{p} = 5 \quad y := 0 \quad x \leq 2 \]

\[ \dot{p} = 6 \quad y = 0 \]

\[ \dot{p} = 3 \]

Minimal cost for reaching \( \ddot{\circ} \):

\[
\inf_{0 \leq t \leq 2} \max \left( \frac{5t + 6(3 - t) + 1}{5t + 3(3 - t) + 9} \right) = 18.66
\]
Cost-optimal reachability in priced timed games

Example

\[ \begin{align*}
\dot{p} &= 5 \\
y &= 0 \\
x &\leq 2 \\
\dot{p} &= 6 \\
\dot{p} &= 3 \\
x &\geq 3 \\
p &+ = 1 \\
p &+ = 9
\end{align*} \]

Minimal cost for reaching \(\ddot{\otimes}\):

\[
\inf_{0 \leq t \leq 2} \max \left( \begin{array}{c}
5t + 6(3 - t) + 1 \\
5t + 3(3 - t) + 9
\end{array} \right) = 18.66 \\
\text{(with } t_{\text{opt}} = \frac{1}{3})
\]
Cost-optimal reachability in priced timed games

**Theorem ([BBR05,BBM06])**

Optimal reachability in priced timed games is **undecidable**.
Theorem ([BBR05,BBM06])

Optimal reachability in priced timed games is undecidable.

Proof

Encode a two-counter machine as a priced timed game.

Cost-optimal reachability in priced timed games

Theorem ([BBR05,BBM06])

Optimal reachability in priced timed games is **undecidable**.

Proof

Encode a two-counter machine as a priced timed game.

- add the value of clock $x$ to the accumulated cost

![Diagram](image.png)


Cost-optimal reachability in priced timed games

Theorem ([BBR05,BBM06])

Optimal reachability in priced timed games is undecidable.

Proof

Encode a two-counter machine as a priced timed game.

- add the value of clock $x$ to the accumulated cost
- add $1 - x$ to the accumulated cost

\[ \begin{align*}
\dot{p} &= 1 \\
\dot{p} &= 0 \\
x &= 0 \\
y &= 1, \quad y := 0 \\
z &= 0 \\
\end{align*} \]

Add $^+$ (x)

Cost-optimal reachability in priced timed games

Theorem ([BBR05,BBM06])

*Optimal reachability in priced timed games is undecidable.*

Proof

Encode a two-counter machine as a priced timed game.

- add the value of clock $x$ to the accumulated cost
- add $1 - x$ to the accumulated cost
- check that $y = 2x$

Test($y=2x$)
Cost-optimal reachability in priced timed games

**Theorem ([BBR05, BBM06])**

*Optimal reachability in priced timed games is undecidable.*

**Proof**

Encode a two-counter machine as a priced timed game.

- add the value of clock \( x \) to the accumulated cost
- add \( 1 - x \) to the accumulated cost
- check that \( y = 2x \)

\[
\begin{align*}
\dot{p} &= 0 \\
\dot{z} &= 0 \\
Test(y=2x) &
\end{align*}
\]

Graph: (Diagram of priced timed game with transitions and cost calculations.)
Optimal reachability in priced timed games is undecidable.

**Proof**

Encode a two-counter machine as a priced timed game.

- add the value of clock $x$ to the accumulated cost
- add $1 - x$ to the accumulated cost
- check that $y = 2x$
- divide clock $x$ by 2

![Diagram of the priced timed game](image)
Cost-optimal reachability in priced timed games

Theorem ([BBR05,BBM06])

Optimal reachability in priced timed games is undecidable.

Proof

Encode a two-counter machine as a priced timed game.

- add the value of clock $x$ to the accumulated cost
- add $1 - x$ to the accumulated cost
- check that $y = 2x$
- divide clock $x$ by 2

$\sim$ We can use the following encoding:

$$x_1 = \frac{1}{2^{c_1}} \quad \quad \quad \quad \quad x_2 = \frac{1}{2^{c_2}}$$
Cost-optimal reachability in priced timed games

Theorem ([BBR05,BBM06])

Optimal reachability in priced timed games is undecidable.

Proof

Encode a two-counter machine as a priced timed game. We can use the following encoding:

\[ x_1 = \frac{1}{2c_1} \quad \quad x_2 = \frac{1}{2c_2} \]

Lemma

The halting state is reachable if, and only if, there is an optimal strategy in the priced timed game.


Cost-optimal reachability in priced timed games

Theorem ([BBR05,BBM06])

**Optimal reachability in priced timed games is undecidable.**

**Proof**

Encode a two-counter machine as a priced timed game.

We can use the following encoding:

\[ x_1 = \frac{1}{2c_1} \quad \text{and} \quad x_2 = \frac{1}{2c_2} \]

**Lemma**

The halting state is reachable if, and only if, there is an optimal strategy in the priced timed game.

reach terminal location with total weight at most 3


What about almost-optimal strategies?

Optimal strategies need not exist...

Optimal strategies may need memory...

\[ \dot{p} = 2, \quad x < 1, \quad x = 0, \quad x > 0; \]

We'd better look for almost-optimal strategies...
What about almost-optimal strategies?

Optimal strategies need not exist...

Optimal strategies may need memory...
What about almost-optimal strategies?

Optimal strategies need not exist...

Optimal strategies may need memory...

\[ \dot{p} = 2 \quad \dot{p} = 1 \]

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

We'd better look for almost-optimal strategies...
The value of a game

Definition

Cost of a path: $\text{cost}(\pi) = \text{sum of costs of all transitions until target location}$

Cost of a strategy: $\text{cost}(\sigma) = \sup \{ \text{cost}(\pi) | \pi \text{ outcome of } \sigma \}$

Optimal cost in a priced timed game: $\text{optcost}_G = \inf \{ \text{cost}(\sigma) | \sigma \text{ winning strategy in } G \}$

The existence of a strategy with cost less than $k$ is undecidable.

What about deciding if $\text{optcost}_G \leq k$?
The value of a game

Definition

Cost of a path:

\[
\text{cost}(\pi) = \text{sum of costs of all transitions until target location}
\]
The value of a game

**Definition**

**Cost of a path:**

\[ \text{cost}(\pi) = \text{sum of costs of all transitions until target location} \]

**Cost of a strategy:**

\[ \text{cost}(\sigma) = \sup \{ \text{cost}(\pi) \mid \pi \text{ outcome of } \sigma \} \]
The value of a game

**Definition**

**Cost of a path:**

\[ \text{cost}(\pi) = \text{sum of costs of all transitions until target location} \]

**Cost of a strategy:**

\[ \text{cost}(\sigma) = \sup \{ \text{cost}(\pi) \mid \pi \text{ outcome of } \sigma \} \]

**Optimal cost in a priced timed game:**

\[ \text{optcost}_G = \inf \{ \text{cost}(\sigma) \mid \sigma \text{ winning strategy in } G \} \]
The value of a game

Definition

Cost of a path:
\[ \text{cost}(\pi) = \text{sum of costs of all transitions until target location} \]

Cost of a strategy:
\[ \text{cost}(\sigma) = \sup \left\{ \text{cost}(\pi) \mid \pi \text{ outcome of } \sigma \right\} \]

Optimal cost in a priced timed game:
\[ \text{optcost}_G = \inf \left\{ \text{cost}(\sigma) \mid \sigma \text{ winning strategy in } G \right\} \]

The existence of a strategy with cost less than \( k \) is undecidable.

What about deciding if \( \text{optcost}_G \leq k \)?
Undecidability of the value problem

Trying to reuse the previous reduction...

\[ q_0 \rightarrow q_4 \quad \text{or} \quad q_1 \rightarrow q_8 \]

\[ q_2 \rightarrow q_6 \quad \text{or} \quad q_3 \rightarrow q_7 \]

\[ q_1 \rightarrow q_9 \quad \text{or} \quad q_2 \rightarrow q_8 \]

\[ q_6 \rightarrow q_0 \quad \text{or} \quad q_7 \rightarrow q_4 \]

\[ q_0 \rightarrow q_2 \quad \text{or} \quad q_3 \rightarrow q_6 \]

\[ q_4 \rightarrow q_1 \quad \text{or} \quad q_6 \rightarrow q_7 \]

\[ q_2 \rightarrow q_5 \quad \text{or} \quad q_7 \rightarrow q_8 \]

\[ q_5 \rightarrow q_9 \quad \text{or} \quad q_8 \rightarrow q_1 \]

\[ q_9 \rightarrow q_0 \quad \text{or} \quad q_1 \rightarrow q_2 \]

\[ c_1 += 2 \quad \text{or} \quad c_2 += 2 \]

\[ c_2 == 0 \quad \text{or} \quad c_1 == 0 \]

\[ c_1 > 0 \quad \text{or} \quad c_2 > 0 \]

The value of the game is 3, but there is no optimal strategy...
Undecidability of the value problem

Trying to reuse the previous reduction...

\[ c_1 =+ = 2 \]
\[ c_2 =+ = 2 \]
\[ c_2 > 0 \]
\[ c_2 = = 0 \]
\[ c_2 =+ = 2 \]
\[ c_1 =+ = 2 \]
\[ c_1 = = 0 \]
\[ c_1 > 0 \]

\[ c_2 = = 0 \]
\[ c_1 = = 0 \]
\[ c_2 > 0 \]
\[ c_2 = = 0 \]
\[ c_1 =+ = 2 \]

\[ c_1 =+ = 2 \]
\[ c_2 = = 0 \]

The value of the game is 3, but there is no optimal strategy...
Undecidability of the value problem

Trying to reuse the previous reduction...

Diagram:

- States: $q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9$
- Transitions:
  - $q_0$ to $q_4$: $c_2 = 0$
  - $q_4$ to $q_5$: $c_1 = 0$
  - $q_0$ to $q_3$: $c_2 > 0$
  - $q_3$ to $q_2$: $c_2 = 0$
  - $q_2$ to $q_6$: $c_1 = 2$
  - $q_6$ to $q_4$: $c_1 = 2$
  - $q_8$ to $q_9$: $c_2 = 2$
  - $q_9$ to $q_8$: $c_1 > 0$

Equations:

- $c_1 + 2$
- $c_2 = 0$
- $c_1 = 0$
- $c_2 > 0$
- $c_1 = 2$
- $c_2 = 2$
- $c_1 > 0$

The value of the game is 3, but there is no optimal strategy...
Undecidability of the value problem

Trying to reuse the previous reduction...

\[ c_1 + = 2 \]
\[ c_2 - - \]
\[ q_0 \]
\[ c_2 + = 2 \]
\[ c_1 + = 0 \]
\[ c_2 > 0 \]
\[ q_6 \]
\[ q_7 \]
\[ q_8 \]
\[ q_9 \]
\[ q_3 \]
\[ q_5 \]
\[ q_4 \]

The value of the game is 3, but there is no optimal strategy...
Undecidability of the value problem

Trying to reuse the previous reduction...

The value of the game is 3, but there is no optimal strategy...
Undecidability of the value problem

Adapting the previous reduction...

$q_{\text{halt}}$

exit nodes: $\mathcal{P}_n$, cost $3 + \frac{1}{n}$

if $M$ does not halt:
- Player 1 simulates correctly until $2^n > 1 + \epsilon$.
- cost($\sigma$) $\leq 3 + \epsilon$.

if $M$ halts:
- correct simulation for finite duration.
- cost($\sigma$) $\geq 3 + \alpha_M$ for all $\sigma$. 

Undecidability of the value problem

Adapting the previous reduction...

$q_{halt}$

Instr.

Test

Instr.

Test

Instr.

Test

Exit

exit nodes: cost 3 + 1

$n$ (length of path)

if $M$ does not halt: Player 1 simulates correctly until $2^n > 1 + \epsilon$.

if $M$ halts: correct simulation for finite duration.

$\text{cost}(\sigma) \leq 3 + \epsilon$ for all $\sigma$
Undecidability of the value problem

Adapting the previous reduction...

If $M$ does not halt:
- Player 1 simulates correctly until $2^n > 1 + \epsilon$.
- Cost($\sigma$) $\leq 3 + \epsilon$.

If $M$ halts:
- Correct simulation for finite duration.
- Cost($\sigma$) $\geq 3 + \alpha_M$ for all $\sigma$.

Exit nodes: cost $3 + \frac{1}{2^n}$
($n = \text{length of path}$)
Undecidability of the value problem

Adapting the previous reduction...

exit nodes: cost $3 + \frac{1}{2^n}$

($n = \text{length of path}$)
Undecidability of the value problem

Adapting the previous reduction...

**if M does not halt:**
Player 1 simulates correctly until $2^n > \frac{1}{\epsilon}$.

$\sim \text{cost}(\sigma) \leq 3 + \epsilon$

exit nodes: cost $3 + \frac{1}{2^n}$
($n = \text{length of path}$)
Undecidability of the value problem

Adapting the previous reduction...

- **if $\mathcal{M}$ does not halt:**
  Player 1 simulates correctly until $2^n > \frac{1}{\epsilon}$.
  \[ \sim \text{ cost}(\sigma) \leq 3 + \epsilon \]

- **if $\mathcal{M}$ halts:**
  correct simulation for finite duration.
  \[ \sim \text{ cost}(\sigma) \geq 3 + \alpha_M \]
  for all $\sigma$

exit nodes: cost $3 + \frac{1}{2^n}$
($n = \text{length of path}$)
Undecidability of the value problem

**Theorem ([BJM15])**

The value problem is undecidable in priced timed automata.

Undecidability of the value problem

**Theorem ([BJM15])**

The value problem is *undecidable* in priced timed automata.

**Remark**

- blue nodes and intermediary instruction modules have cost zero everywhere;
- positive weights only occur in acyclic parts.

Approximation of the optimal cost

Definition

A priced timed game $\mathcal{G}$ is almost-strongly non-Zeno if there exists $\kappa > 0$ for any run $\rho$ that starts and ends in the same region:

$$\text{cost}(\rho) \geq \kappa \quad \text{or} \quad \text{cost}(\rho) = 0$$
Approximation of the optimal cost

**Definition**

A priced timed game $G$ is **almost-strongly non-Zeno** if there exists $\kappa > 0$ for any run $\rho$ that starts and ends in the same region:

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

**Theorem ([BJM15])**

*The optimal cost of almost-strongly non-Zeno priced timed automata can be approximated.*

Approximation of the optimal cost

Definition

A priced timed game $G$ is almost-strongly non-Zeno if there exists $\kappa > 0$ for any run $\rho$ that starts and ends in the same region:

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

Theorem ([BJM15])

The optimal cost of almost-strongly non-Zeno priced timed automata can be approximated: for every $\epsilon > 0$, we can compute values $v_\epsilon^+$ and $v_\epsilon^-$ such that

$$|v_\epsilon^+ - v_\epsilon^-| < \epsilon \quad \quad v_\epsilon^- \leq \text{optcost}_G \leq v_\epsilon^+$$

a strategy $\sigma_\epsilon$ such that

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

Approximation of the optimal cost

Proof

- **semi-unfolding** of region automaton (seen as a timed game)

Only cost 0
Kernel $\mathcal{K}$

Only cost 0
Kernel $\mathcal{K}$

Hypothesis: $\text{cost} > 0 \downarrow \text{cost} \geq \kappa$; bounded depth
compute exact optimal cost in tree-like parts
compute approximate optimal cost in kernels
Approximation of the optimal cost

Proof

- **semi-unfolding** of region automaton (seen as a timed game)

Only cost 0
Kernel $\mathcal{K}$

Hypothesis: $\text{cost} > 0$ \(\Downarrow\) $\text{cost} \geq \kappa$;
bounded depth

compute
exact
optimal cost in tree-like parts

compute
approximate
optimal cost in kernels
Approximation of the optimal cost

Proof

- **semi-unfolding** of region automaton (seen as a timed game)

  \[(\ell, r)\]

Only cost 0
Kernel \(K\)

Hypothesis:
\[\text{cost} > 0 \Rightarrow \text{cost} \geq \kappa; \text{bounded depth}\]

compute exact optimal cost in tree-like parts
compute approximate optimal cost in kernels
Approximation of the optimal cost

Proof

- **semi-unfolding** of region automaton (seen as a timed game)

Hypothesis:

\[
\text{cost} > 0 \\
\downarrow \\
\text{cost} \geq \kappa
\]
Approximation of the optimal cost

Proof:

- **semi-unfolding** of region automaton (seen as a timed game)

Hypothesis:

\[
\text{cost} > 0 \\
\Downarrow \\
\text{cost} \geq \kappa
\]

\(\sim\) bounded depth
Approximation of the optimal cost

Proof

- **semi-unfolding** of region automaton (seen as a timed game)
- compute **exact** optimal cost in tree-like parts

![Diagram showing the process of approximation](image)
Approximation of the optimal cost

Proof

- **semi-unfolding** of region automaton (seen as a timed game)
- compute **exact** optimal cost in tree-like parts
Approximation of the optimal cost

Proof

- **semi-unfolding** of region automaton (seen as a timed game)
- compute **exact** optimal cost in tree-like parts

1. [Diagram showing approximate and exact optimal costs]
Approximation of the optimal cost

Proof

- **semi-unfolding** of region automaton (seen as a timed game)
- **compute exact** optimal cost in tree-like parts
Approximation of the optimal cost

Proof

- **semi-unfolding** of region automaton (seen as a timed game)
- compute **exact** optimal cost in tree-like parts
- compute **approximate** optimal cost in kernels

Output cost functions $f$

Under- and over-approximate by $f - \epsilon$ and $f + \epsilon$; reachability timed game in small regions
Approximation of the optimal cost

Proof

- **semi-unfolding** of region automaton (seen as a timed game)
- compute exact optimal cost in tree-like parts
- compute approximate optimal cost in kernels

Output cost functions $f$

Under- and over-approximate by piecewise constant functions $f_{\epsilon}^{-}$ and $f_{\epsilon}^{+}$
Approximation of the optimal cost

Proof

- **semi-unfolding** of region automaton (seen as a timed game)
- compute **exact** optimal cost in tree-like parts
- compute **approximate** optimal cost in kernels

Output cost functions $f$

Under- and over-approximate by piecewise constant functions $f_\epsilon^-$ and $f_\epsilon^+$
Approximation of the optimal cost

Proof

- **semi-unfolding** of region automaton (seen as a timed game)
- compute **exact** optimal cost in tree-like parts
- compute **approximate** optimal cost in kernels

Output cost functions $f$

Under- and over-approximate by piecewise constant functions $f^-_\epsilon$ and $f^+\epsilon$

$\sim$ reachability timed game in small regions
Approximation of the optimal cost

Proof
- **semi-unfolding** of region automaton (seen as a timed game)
- compute **exact** optimal cost in tree-like parts
- compute **approximate** optimal cost in kernels

Output cost functions $f$

Under- and over-approximate by piecewise constant functions $f_{\epsilon}^-$ and $f_{\epsilon}^+$

$\leadsto$ reachability timed game in small regions
Outline of the talk

1. Introduction: timed automata and timed games
2. Measuring other quantities in timed automata
   - Examples
   - Timed automata with stopwatches
   - Timed automata with observer variables
3. Cost-optimal strategies
   - Optimal reachability in priced timed automata
   - Optimal reachability in priced timed games
4. Conclusions and future works
Conclusions and future directions

Priced timed automata and games

- convenient for modelling resources;
- 1-player setting remains tractable (sort of);
- 2-player setting undecidable, but approximable.
- approximation algorithms are a convenient trade-off.

Future work

- improve approximation technique (in terms of complexity);
- extend approximation to whole class of priced timed games;
- average energy and energy constraints;
- robust analysis of priced timed games.
Conclusions and future directions

Priced timed automata and games
- convenient for modelling resources;
- 1-player setting remains tractable (sort of);
- 2-player setting undecidable, but approximable.
- approximation algorithms are a convenient trade-off.

Future work
- improve approximation technique (in terms of complexity);
- extend approximation to whole class of priced timed games;
- average energy and energy constraints;
- robust analysis of priced timed games.