Timed Systems
— Model-checking and games —

Nicolas MARKEY

Lab. Spécification & Vérification
CNRS & ENS Cachan

June 23, 2008
Why verification?

Computers are everywhere!
Why verification?

Computers are everywhere!
Why verification?

Computers are everywhere!
Why verification?

Computers are everywhere!
Why verification?

Computers are everywhere!
Why verification?

Computers are everywhere!
Why verification?

Computers are everywhere!
Why verification?

Computers are everywhere!
Why verification?

Computers are everywhere!
Why verification?

Bugs are everywhere!
Why verification?

Bugs are everywhere!

Ariane 5
Why verification?

Bugs are everywhere!

Ariane 5

Mars Climate Obs.
Why verification?

Bugs are everywhere!

- Ariane 5
- Mars Climate Obs.
- Northeast blackout, 2003
Why verification?

Bugs are everywhere!

- Ariane 5
- Mars Climate Obs.
- Airport screens
- Night before night after
- Northeast blackout, 2003
Why *formal* verification?

- provides (partial) proof of correctness
- model-based methods;
- (more-or-less) exhaustive methods;
- (more-or-less) automatic techniques.
Why *formal* verification?

- provides (partial) proof of correctness
- model-based methods;
- (more-or-less) exhaustive methods;
- (more-or-less) automatic techniques.

Several techniques:

- (model-based) testing
Why *formal* verification?

- provides (partial) proof of correctness
- model-based methods;
- (more-or-less) exhaustive methods;
- (more-or-less) automatic techniques.

Several techniques:

- (model-based) testing
- theorem proving
Why *formal* verification?

- provides (partial) proof of correctness
- model-based methods;
- (more-or-less) exhaustive methods;
- (more-or-less) automatic techniques.

Several techniques:

- (model-based) testing
- theorem proving
- model checking
Why *formal* verification?

- provides (partial) proof of correctness
- model-based methods;
- (more-or-less) exhaustive methods;
- (more-or-less) automatic techniques.

Several techniques:

- (model-based) testing
- theorem proving
- model checking
- ...

Different techniques, often complementary...
Model checking

system:

property:

G (request ⇒ F grant)
Model checking

system:

⇒

property:

G(request ⇒ F grant)
Model checking

system:

property:

G(request ⇒ F grant)

model-checking algorithm

G(request ⇒ F grant)
Model checking

system:

⇒

property:

G (request ⇒ F grant)

model-checking algorithm

yes/no
Model checking and control

system:

⇒

property:

$$G(request \Rightarrow F\ grant)$$
Model checking and control

system:

⇒

property:

G(request ⇒ F grant)

control algorithm

yes/no
Model checking and control

System:

Property:

G(request → F grant)

Control algorithm

Yes/no
Model checking and control

system:

property:

G(request ⇒ F grant)

control algorithm

yes/no
The untimed setting

Model-checking finite-state systems

- **CTL**: low complexity, efficient data-structures;
- **LTL**: higher complexity, automata-based techniques.

*Well-understood*, but still several recent improvements...
The untimed setting

**Model-checking finite-state systems**

- CTL: low complexity, efficient data-structures;
- LTL: higher complexity, automata-based techniques.

*Well-understood, but still several recent improvements…*

**Control and games**

- more recent (in computer science), more difficult;
- ATL: efficient algorithms.

*Some work remains to be done (e.g. tool development).*
Time is important...

The ordering of events is not always sufficient!

Quantitative measurement of time is needed in many applications (e.g. for modelling timeouts, delays, ...); even if not absolutely needed, quantitative time allows more faithful modelling.

Checking quantitative properties on timed systems is also compulsory in many real systems (lifts, airbags, ...); it allows performance evaluation.
Time is important...

The ordering of events is not always sufficient!

- Quantitative measurement of time is needed in many applications (e.g. for modelling timeouts, delays, ...);
- even if not absolutely needed, quantitative time allows more faithful modelling.
Time is important...

The ordering of events is not always sufficient!

- **Quantitative measurement of time** is needed in many applications (e.g. for modelling timeouts, delays, ...);
- even if not absolutely needed, quantitative time allows more faithful modelling.

- Checking **quantitative properties** on timed systems is also compulsory in many real systems (lifts, airbags, ...);
- it allows performance evaluation.
Outline of the presentation

1. Introduction

2. Discrete-time systems
   - Model Checking Timed Transition Graphs
   - Playing Discrete-Time Games

3. Dense-time systems: timed automata
   - Model Checking Timed Automata
   - Dense-time Games

4. Hot topics...
   - Robustness of timed automata
   - Priced timed automata and priced timed games
   - Other related models

5. Conclusion
Outline of the presentation

1. Introduction

2. Discrete-time systems
   - Model Checking Timed Transition Graphs
   - Playing Discrete-Time Games

3. Dense-time systems: timed automata
   - Model Checking Timed Automata
   - Dense-time Games

4. Hot topics...
   - Robustness of timed automata
   - Priced timed automata and priced timed games
   - Other related models

5. Conclusion
Quantitative model checking on Kripke structures

- Basic idea: counting the number of transitions;
Quantitative model checking on Kripke structures

- Basic idea: counting the number of transitions;
- Counting only some transitions;
- Adding nonnegative weights on transitions;
A timed transition graph is a 4-tuple $\mathcal{T} = \langle Q, Q_0, T, \ell \rangle$

- $Q$ is a set of locations,
- $Q_0$ is the subset of initial locations,
- $T \subseteq Q \times \mathbb{R}^+ \times Q$ is the transition relation,
- $\ell: Q \rightarrow 2^{AP}$ is the labelling function.
A **timed transition graph** is a 4-tuple $\mathcal{T} = \langle Q, Q_0, T, \ell \rangle$

- $Q$ is a set of **locations**,
- $Q_0$ is the subset of **initial locations**,
- $T \subseteq Q \times \mathbb{R}^+ \times Q$ is the **transition relation**,
- $\ell : Q \rightarrow 2^{AP}$ is the **labelling function**.

A **finite timed transition graph** is a TTG such that

- $Q$ is finite,
- $T \subseteq Q \times \mathbb{N} \times Q$. 

**Definition**
Finite Timed Transition Systems

Definition

A *timed transition graph* is a 4-tuple \( \mathcal{T} = \langle Q, Q_0, T, \ell \rangle \)

- \( Q \) is a set of *locations*,
- \( Q_0 \) is the subset of *initial locations*,
- \( T \subseteq Q \times \mathbb{R}^+ \times Q \) is the *transition relation*,
- \( \ell : Q \rightarrow 2^{AP} \) is the *labelling function*.

A *finite timed transition graph* is a TTG such that

- \( Q \) is finite,
- \( T \subseteq Q \times \mathbb{N} \times Q \).

A *unitary timed transition graph* is a finite TTG such that

- \( T \subseteq Q \times \{0, 1\} \times Q \).
Finite Timed Transition Systems

Example
Discrete-time semantics (a.k.a. jump semantics)

Definition

A path in a TTG is a sequence of transitions $\langle (q_i, d_i, q'_i)_{i \in \mathbb{N}}, \ell \rangle$ s.t.
- $q_0$ is an initial location,
- $q_{i+1} = q'_i$ for all $i$. 

Example
Discrete-time semantics (a.k.a. *jump* semantics)

**Definition**

A *path* in a TTG is a sequence of transitions $\langle (q_i, d_i, q'_i)_{i \in \mathbb{N}}, \ell \rangle$ s.t.

- $q_0$ is an initial location,
- $q_{i+1} = q'_i$ for all $i$.

**Example**

```
idle 1 idea 3 idle 1 idea 6 write
```
Temporal logics

Definition

\[ U \]
Temporal logics

Definition

$U^-$
We use strict semantics of Until throughout this talk.

\[ \sim U \overset{\text{def}}{=} \lor (\land U) \]
Temporal logics

Definition

- Temporal operators: $\acf{U}$

Examples

- Example 1: $X \overset{\text{def}}{=} \text{false} \ac{U}$
Temporal logics

Definition

- $U$: 

Examples

- $X$: $\text{false} \cup X$
Temporal logics

Definition

\( U \): 

\[ \begin{align*} 
    X & \overset{\text{def}}{=} \text{false} \ U \\
    F & \overset{\text{def}}{=} \text{true} \ U 
\end{align*} \]
Temporal logics

**Definition**

- $U$:

**Examples**

- $X \overset{\text{def}}{=} \text{false} \cup \text{false} \uparrow$
- $F \overset{\text{def}}{=} \text{true} \cup \text{true} \uparrow$
Temporal logics

**Definition**

- $U$:

**Examples**

- $X \overset{\text{def}}{=} \text{false} \ U$
- $F \overset{\text{def}}{=} \text{true} \ U$
- $G \overset{\text{def}}{=} \neg F \ U$
Temporal logics

Definition

- $X \overset{\text{def}}{=}$ false $U$
- $F \overset{\text{def}}{=}$ true $U$
- $G \overset{\text{def}}{=} \neg F$

Examples

- $X \overset{\text{def}}{=}$ false $U$
- $F \overset{\text{def}}{=}$ true $U$
- $G \overset{\text{def}}{=} \neg F$
Temporal logics

**Definition**

- \( U \):

**Examples**

- \( X \) \( \overset{\text{def}}{=} \) false \( U \)
- \( F \) \( \overset{\text{def}}{=} \) true \( U \)
- \( G \) \( \overset{\text{def}}{=} \) \( \neg F \) \( \neg U \)
- \( R \) \( \overset{\text{def}}{=} \) \( \neg ( \neg U ) \) \( U ( \neg U ) \) \( = \) \( U ( \neg \land \neg U ) \) \( \lor G \)
Temporal logics

Definition

\[ U \leq k : \]

\[ U \geq k : \]

Definition

\[ U_{\leq k} : \]

\[ U_{\leq 17} : \]
Temporal logics

**Definition**

- $U$: 

- $U \leq k$: 

- $U \geq k$: 

**Definition**

- $U_{\leq k}$: 

- $U_{= k}$: 

- $U_{= 15}$: 

- $U_{= 15}$:
Temporal logics

Definition

\[ U \leq k : \]
\[ U = k : \]
\[ U \geq k : \]

\[ U \leq 17 : \]
\[ U = 15 : \]
\[ U \geq 6 : \]
Temporal logics

**Definition**

- $U \leq k$:
- $U = k$:
- $U \geq k$:

**Examples**

- $X \sim k \overset{\text{def}}{=} \text{false} U \sim k$
Temporal logics

**Definition**

- $\mathbf{U}_{\leq k}$
- $\mathbf{U}_{= k}$
- $\mathbf{U}_{\geq k}$

**Examples**

- $X_{\sim k} \overset{\text{def}}{=} \text{false} \ U_{\sim k}$
- $F_{\sim k} \overset{\text{def}}{=} \text{true} \ U_{\sim k}$
Temporal logics

**Definition**

- \( U_{\leq k} \):
- \( U_{= k} \):
- \( U_{\geq k} \):

**Examples**

\[
\begin{align*}
X_{\sim k} & \overset{\text{def}}{=} \text{false} U_{\sim k} \\
G_{\sim k} & \overset{\text{def}}{=} \neg F_{\sim k} \\
F_{\sim k} & \overset{\text{def}}{=} \text{true} U_{\sim k}
\end{align*}
\]
Temporal logics

**Definition**

- $U \leq k \defeq k - 0$
- $U = k$
- $U \geq k$

**Examples**

- $X \sim k \defeq \text{false} U \sim k$
- $F \sim k \defeq \text{true} U \sim k$
- $G \sim k \defeq \neg F \sim k$
- $R \sim k \defeq \neg (\neg \cdot) U \sim k (\neg \cdot)$
Temporal logics

**Definition**

- $U_{\leq k}$
- $U_{= k}$
- $U_{\geq k}$

**Examples**

- $X_{\sim k} \overset{\text{def}}{=} \text{false} \ U_{\sim k}$
- $F_{\sim k} \overset{\text{def}}{=} \text{true} \ U_{\sim k}$
- $G_{\sim k} \overset{\text{def}}{=} \neg F_{\sim k}$
- $R_{\sim k} \overset{\text{def}}{=} \neg (\neg U_{\sim k} (\neg)) \neq U_{\sim k} (\text{true} \wedge \text{true}) \lor G_{\sim k}$
Temporal logics

Definition

$$E \varphi$$
Temporal logics

**Definition**

$E\varphi$ def $\neg E\neg \varphi$

$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$

Diagram:

- Red circle with 0 and 3
- Blue circle with 2 and 1
- Yellow circle with 4 and 5

Transformed into a tree structure with multiple nodes.
Temporal logics

Definition

$E\varphi \triangleq \neg E \neg \varphi$
Temporal logics

Definition

$E \varphi \overset{\text{def}}{=} \neg E \neg \varphi$

$A \varphi \overset{\text{def}}{=} \neg E \neg \varphi$
Temporal logics

Definition

\[ E \varphi \]  
\[ A \varphi \stackrel{\text{def}}{=} \neg E \neg \varphi \]

\[ E \varphi \]

\[ A \varphi \]

\[ \neg E \neg \varphi \]
Temporal logics

**Definition**

MTL $\exists \varphi ::= \bigcirc | \bigcirc | \ldots | \neg \varphi | \varphi \lor \varphi | \varphi U_{\sim k} \varphi$

TCTL $\exists \varphi ::= \bigcirc | \bigcirc | \ldots | \neg \varphi | \varphi \lor \varphi | E \varphi U_{\sim k} \varphi | A \varphi U_{\sim k} \varphi$
## Temporal logics

### Definition

**MTL**
\[ \exists \varphi ::= \bigcirc | \bigcirc | \ldots | \neg \varphi | \varphi \lor \varphi | \varphi U_{\sim k} \varphi \]

**TCTL**
\[ \exists \varphi ::= \bigcirc | \bigcirc | \ldots | \neg \varphi | \varphi \lor \varphi | E\varphi U_{\sim k} \varphi | A\varphi U_{\sim k} \varphi \]

### Definition

**MTL_{\leq,\geq}**
\[ \exists \varphi ::= \bigcirc | \bigcirc | \ldots | \neg \varphi | \varphi \lor \varphi | \varphi U_{\leq k} \varphi | \varphi U_{\geq k} \varphi \]

**TCTL_{\leq,\geq}**
\[ \exists \varphi ::= \bigcirc | \bigcirc | \ldots | \neg \varphi | \varphi \lor \varphi | E\varphi U_{\leq k} \varphi | E\varphi U_{\geq k} \varphi | A\varphi U_{\leq k} \varphi | A\varphi U_{\geq k} \varphi \]
“It is always possible to publish your ideas within three months”

\[\mathbf{A} \mathbf{G}(\text{idea} \Rightarrow \mathbf{E} \mathbf{F}_{\leq 90} \text{publish})\]
Model checking $TCTL_{\leq, \geq}$

**Theorem**

*Model checking $TCTL_{\leq, \geq}$ over TTGs is P-complete.*
Model checking $TCTL_{\leq,\geq}$

**Theorem**

Model checking $TCTL_{\leq,\geq}$ over TTGs is $P$-complete.

**Proof.**

$E \cup U_{\leq 10}$
Model checking $TCTL_{\leq, \geq}$

Theorem

Model checking $TCTL_{\leq, \geq}$ over TTGs is P-complete.

Proof.

- $\mathbf{E} \mathbf{U}_{\leq 10}$
  - restrict to $\bullet$- and $\circ$-states;

\[
\begin{array}{c}
1 \\
\downarrow \\
2 \\
\downarrow \\
3 \\
\downarrow \\
4 \\
\downarrow \\
5 \\
\downarrow \\
7
\end{array}
\]
Model checking $TCTL_{\leq, \geq}$

**Theorem**

Model checking $TCTL_{\leq, \geq}$ over TTGs is P-complete.

**Proof.**

- $E \bigcirc U_{\leq 10}$
  - restrict to $\bigcirc$- and $\bigcirc$-states;
  - compute shortest path to some $\bigcirc$-state;
Model checking $TCTL_{\leq,\geq}$

Theorem

Model checking $TCTL_{\leq,\geq}$ over TTGs is P-complete.

Proof.

- $E \circ U_{\leq 10} \circ$
  - restrict to blue and red states;
  - compute shortest path to some red state;
  - come back the the whole graph;
Model checking $TCTL_{\leq,\geq}$

**Theorem**

Model checking $TCTL_{\leq,\geq}$ over TTGs is P-complete.

**Proof.**

- $E \cup U_{\leq 10}$
  - restrict to $\bullet$- and $\bullet$-states;
  - compute shortest path to some $\bullet$-state;
  - come back the the whole graph;
  - compute shortest **non-trivial** path to some $\bullet$-state via $\bullet$-states.
Model checking $TCTL_{\leq, \geq}$

**Theorem**

Model checking $TCTL_{\leq, \geq}$ over TTGs is P-complete.

**Proof.**

- **E**\( U_{\geq10} \)
Model checking $TCTL_{\leq, \geq}$

**Theorem**

Model checking $TCTL_{\leq, \geq}$ over TTGs is P-complete.

**Proof.**

- $E \bigcirc U_{\geq 10} \bigcirc$
  - mark positive $\bigcirc$-cycles;

![Diagram](image)
Model checking $TCTL_{\leq, \geq}$

**Theorem**

Model checking $TCTL_{\leq, \geq}$ over TTGs is P-complete.

**Proof.**

- $E\ U_{\geq10}$
  - mark positive -cycles;
  - restrict to “acyclic” - and -states;
Model checking $\text{TCTL}_{\leq,\geq}$

**Theorem**

Model checking $\text{TCTL}_{\leq,\geq}$ over TTGs is P-complete.

**Proof.**

- $\mathbf{E} \mathcal{U}_{\geq 10}$
  - mark positive $\mathbf{E}$-cycles;
  - restrict to "acyclic" $\mathbf{U}$- and $\mathbf{E}$-states;
  - compute longest paths to some $\mathbf{E}$-states;
Model checking $\text{TCTL}_{\leq,\geq}$

**Theorem**

Model checking $\text{TCTL}_{\leq,\geq}$ over TTGs is P-complete.

**Proof.**

- $\mathbf{E} \mathbf{U}_{\geq 10}$
  - mark positive $\circled{+}$-cycles;
  - restrict to “acyclic” $\circled{-}$- and $\circled{+}$-states;
  - compute longest paths to some $\circled{-}$-states;
  - in the whole graph, compute non-trivial longest paths to some $\circled{-}$-state via acyclic $\circled{+}$-states.
Model checking $TCTL_{\leq, \geq}$

**Theorem**

*Model checking $TCTL_{\leq, \geq}$ over TTGs is P-complete.*

**Proof.**

- $E \bigcirc U_{\geq 10}$
  - mark positive $\bigcirc$-cycles;
  - restrict to “acyclic” $\bigcirc$- and $\bigcirc$-states;
  - compute longest paths to some $\bigcirc$-states;
  - in the whole graph, compute non-trivial longest paths to some $\bigcirc$-state via acyclic $\bigcirc$-states.
- tag $+\infty$ those states satisfying $E(\bigcirc U (\bigcirc \land E\bigcirc U \bigcirc))$
Model checking $TCTL_{\leq,\geq}$

**Theorem**

*Model checking $TCTL_{\leq,\geq}$ over TTGs is P-complete.*

*Proof.*

- Other modalities are handled via similar techniques.
Theorem

Model checking $TCTL_{\leq,\geq}$ over TTGs is P-complete.

Proof.

- other modalities are handled via similar techniques.
- P-hardness: CTL model-checking is P-hard.
Proposition

Model checking TCTL over TTGs is NP-hard.
Model checking TCTL

Proposition
Model checking TCTL over TTGs is NP-hard.

Proof.

**SUBSET-SUM**
Given positive integers $n_1, \ldots, n_k$ and $b$, do there exist $i_1 < \cdots < i_p$ in $[1, k]$ s.t.

$$\sum_{j\in[1,p]} n_{ij} = b?$$
Proposition

Model checking TCTL over TTGs is NP-hard.

Proof.

SUBSET-SUM

Given positive integers $n_1, \ldots, n_k$ and $b$, do there exist $i_1 < \cdots < i_p$ in $[1, k]$ s.t.

$$\sum_{j \in [1,p]} n_{i_j} = b?$$
Model checking TCTL

Proposition
Model checking TCTL over TTGs is NP-hard.

Proposition
Model checking $E\Box U_{\leq k}^\circ$ over TTGs can be done in NP.
Model checking TCTL

Proposition
Model checking TCTL over TTGs is NP-hard.

Proposition
Model checking $E \bigcirc U_{\leq k}$ over TTGs can be done in NP.

\[ E \bigcirc U_{=2008} \]
Proposition

Model checking TCTL over TTGs is NP-hard.

Proposition

Model checking \( E \bigcirc U \geq k \) over TTGs can be done in NP.

\( E \bigcirc U \geq 2008 \)

-uess number of times each transition will be fired;
Model checking TCTL

**Proposition**

Model checking TCTL over TTGs is NP-hard.

**Proposition**

Model checking $\mathbf{E}\bigcirc \mathbf{U}^{\leq k}$ over TTGs can be done in NP.

- $\mathbf{E}\bigcirc \mathbf{U}^{=2008}$
  - guess number of times each transition will be fired;
  - check that it corresponds to a valid path going to some $\blacksquare$-state via $\bigcirc$-states.

![Diagram of a TCTL model with numbers labeled on transitions and states.](image)
Model checking TCTL

Proposition
Model checking TCTL over TTGs is NP-hard.

Proposition
Model checking $E \bigcirc U_{\leq k}$ over TTGs can be done in NP.

- $E \bigcirc U_{\leq 2008}$
  - guess number of times each transition will be fired;
  - check that it corresponds to a valid path going to some state via states.
  - check that the total length is 2008.

$1 \times 6 + 666 \times 2 + 667 \times 1 + 1 \times 3 = 2008$
Model checking TCTL

Theorem

Model checking TCTL over TTGs is $\Delta^P_2$-complete.
Model checking TCTL

Theorem

Model checking TCTL over TTGs is $\Delta_2^P$-complete.
Model checking TCTL

**Theorem**

Model checking TCTL over TTGs is $\Delta^p_2$-complete.

**Proof.**

\[ EG_{\geq 2}(A\Box U_{=3} \Diamond) \land AF_{=5}(EG_{\leq 4} \Diamond) \]
Model checking TCTL

**Theorem**

Model checking TCTL over TTGs is $\Delta_2^p$-complete.

**Proof.**

\[
\begin{align*}
\mathbf{E} G_{\geq 2}(A \circ U = 3 \bullet) \land A F_{= 5}(E G \leq 4 \bullet)
\end{align*}
\]

NP oracle
Model checking TCTL

**Theorem**

Model checking TCTL over TTGs is $\Delta^p_2$-complete.

**Proof.**

$E \bigwedge \forall 2 (A \bigcirc U \geq 3 \bigcirc) \land A F \leq 5 (E G \leq 4 \bigcirc)$

↑
in P

NP oracle in P

NP oracle in P

\(\Delta^p_2\)-hardness: reduction from SN-SAT (omitted).
Model checking TCTL

**Theorem**

*Model checking TCTL over TTGs is $\Delta^P_2$-complete.*

**Proof.**

\[ \text{in P} \]

\[ \exists \text{EG} \geq 2(A \cup \geq 3 \land A \leq 5(E \leq 4)) \land \exists \text{AF} = 5(E \leq 4) \]
Model checking TCTL

Theorem

Model checking TCTL over TTGs is $\Delta^P_2$-complete.

Proof.

\[
E \bigwedge \left[ \begin{array}{c}
G_{\geq 2} \left( A \bigvee \bigwedge \left( U_{\leq 3} \bigvee A \bigvee \bigwedge \left( F_{\leq 4} \bigvee E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 5} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 6} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 7} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 8} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 9} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 10} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 11} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 12} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 13} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 14} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 15} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 16} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 17} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 18} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 19} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 20} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 21} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 22} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 23} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 24} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 25} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 26} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 27} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 28} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 29} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 30} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 31} \bigwedge \bigwedge \left( E \bigvee \bigwedge \left( G \right) \right) \right) \right) \bigwedge \bigwedge \left( A \bigvee \bigwedge \left( F_{\leq 32} \bigwedge \bigwedge \left( E \bigvew
Model checking TCTL

Theorem

Model checking TCTL over TTGs is $\Delta^p_2$-complete.

Proof.

$\exists \mathbf{G}_{\geq 2}(A_{\sqcap U = 3} \sqcap) \land A_{F = 5}(E G_{\leq 4} \sqcup)$

in $P$
Model checking TCTL

Theorem

Model checking TCTL over TTGs is $\Delta_2^P$-complete.

Proof.

- $\text{EG}_{\geq 2}(\text{A} \circ \text{U} \geq 3 \quad \text{E}) \land \text{AF}_5(\text{EG}_{\leq 4} \quad \text{E})$

- $\Delta_2^P$-hardness: reduction from SN-SAT (omitted).
Model checking TCTL over unitary TTGs

**Theorem**

*Model checking TCTL over 1TTGs is P-complete.*
Model checking TCTL over unitary TTGs

Theorem

Model checking TCTL over 1TTGs is P-complete.

Proof.

- $E \bullet U \leq k$

  - Compute “adjacency” matrices

  $$T_0 = \{(q, q') \mid q \models \bullet U_{=1} (q' \land \bullet)\}$$

  $$T_0 = \{(q, q') \mid q \models \bullet U_{=1} (q' \land \bullet)\}$$
Model checking TCTL over unitary TTGs

Theorem

Model checking TCTL over 1TTGs is P-complete.

Proof.

- $E \bigcirc U_{\geq k} \bigcirc$
  - compute “adjacency” matrices
    
    $T_\bigcirc = \{(q, q') \mid q \models \bigcirc U_{\leq 1} (q' \land \bigcirc)\}$
    
    $T_\bigcirc = \{(q, q') \mid q \models \bigcirc U_{\leq 1} (q' \land \bigcirc)\}$

- compute $T_\bigcirc^2$, $T_\bigcirc^4$, $\ldots$, $T_\bigcirc^{2^m}$ where $m = \lceil \log k \rceil$;
Model checking TCTL over unitary TTGs

Theorem

Model checking TCTL over 1TTGs is P-complete.

Proof.

1. Compute “adjacency” matrices

\[ T_0 = \{(q, q') \mid q \models \textcolor{blue}{\bigcirc} \textcolor{red}{U=1} (q' \land \textcolor{red}{\bigcirc})\} \]

\[ T_1 = \{(q, q') \mid q \models \textcolor{blue}{\bigcirc} \textcolor{red}{U=1} (q' \land \textcolor{red}{\bigcirc})\} \]

2. Compute \( T_2, T_4, \ldots, T_{2^m} \) where \( m = \lceil \log k \rceil \);

3. Compute \( T_{k-1} \cdot T_0 \).
TTGs with continuous semantics

**Definition**

\[ E \land F = (E \land F) \land (E \land F) \land (E \land \neg F) \]

Diagram:

```
      5
    /  \
   /    \
  3     3
   \    /  \\
    \  /    \
     V     V
```

Red and green nodes represent states, with edges indicating transitions. The numbers indicate state counts or transitions.
TTGs with continuous semantics

**Definition**

\[ E \land F = (E \land F) \land (E \land F) \]

\[ A \lor G = (E \land F) \lor (E \land F) \]

\[ E \land F \land (E \land \neg F) \]
TTGs with continuous semantics

Definition

Examples

\[ E F \equiv_2 (E F \land E F) \]
TTGs with continuous semantics

**Definition**

Example:

\[ EF \equiv_2 (EF \lor \neg EF) \]

\[ AG \equiv_2 (EF \lor \neg EF) \]
TTGs with continuous semantics

**Definition**

\[
E F =_2 (E F \land E F)
\]

\[
A G =_2 (E F \land E F)
\]

\[
E F (\land \neg E F)
\]
Model checking under continuous semantics

Theorem

Model checking $TCTL_{\leq,\geq}$ over continuous TTGs is P-complete.
Model checking under continuous semantics

**Theorem**

Model checking $TCTL_{\leq, \geq}$ over continuous TTGs is P-complete.

**Proof.**

for each state $\bigcirc$ of $\mathcal{T}$ and each subformula $\psi$, define $Sat(\bigcirc, \psi)$ as the set of integers s.t.

$$n \in Sat(\bigcirc, \psi) \iff n \models \psi$$
Model checking under continuous semantics

**Theorem**

*Model checking $TCTL_{\leq,\geq}$ over continuous TTGs is P-complete.*

**Proof.**

- For each state $\bigcirc$ of $T$ and each subformula $\psi$, define $\text{Sat}(\bigcirc, \psi)$ as the set of integers s.t.
  
  $$n \in \text{Sat}(\bigcirc, \psi) \iff n \models \psi$$

- Prove by induction that:
  - $\text{Sat}(\bigcirc, \psi)$ is a union of at most $|T| \cdot |\psi|$ disjoint intervals.
  - $\text{Sat}(\bigcirc, \psi)$ can be computed in time $O(|T|^2 \cdot |\psi|^3)$. 

\[ \square \]
Model checking under continuous semantics

**Theorem**

*Model checking $TCTL_{\leq,\geq}$ over continuous TTGs is P-complete.*

**Theorem**

*Model checking $TCTL$ over continuous TTGs is PSPACE-complete.*
Model checking under continuous semantics

Theorem

Model checking $\text{TCTL}_{\leq,\geq}$ over continuous TTGs is $\text{P}$-complete.

Theorem

Model checking $\text{TCTL}$ over continuous TTGs is $\text{PSPACE}$-complete.

Proof.

- **PSPACE algorithm**: the continuous semantics can be encoded as a timed automaton, for which $\text{TCTL}$ model-checking is $\text{PSPACE}$-complete.
- **PSPACE hardness**: reduction form QBF (omitted).
Linear-time temporal logics

Theorem

Model checking MTL over TTGs is EXPSPACE-complete.

Model checking $MTL_{\leq, \geq}$ over TTGs is PSPACE-complete.
Linear-time temporal logics

**Theorem**

*Model checking MTL over TTGs is EXPSPACE-complete.*

*Model checking MTL$_{\leq,\geq}$ over TTGs is PSPACE-complete.*

**Proof.**

- membership in EXPSPACE: tableau construction.
Theorem

Model checking MTL over TTGs is EXPSPACE-complete.

Model checking MTL$_{\leq, \geq}$ over TTGs is PSPACE-complete.

Proof.

- membership in EXPSPACE: tableau construction.
  - build the closure of $\varphi$, such that
    - if some $U_{\sim k}$-formula appears in the closure, then so do the corresponding $U_{\sim l}$-formulas for all $l < k$.
    - the closure also contains atomic propositions $\text{time}_d$ for each integer $d$ appearing in the TTG;
  - build the tableau for $\varphi$, and its product with $\mathcal{T}$. 
Linear-time temporal logics

<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model checking MTL over TTGs is EXPSPACE-complete.</td>
</tr>
<tr>
<td>Model checking $MTL_{\leq,\geq}$ over TTGs is PSPACE-complete.</td>
</tr>
</tbody>
</table>

**Proof.**

- hardness in EXPSPACE: encode a $2^n$-tape Turing machine.
Linear-time temporal logics

**Theorem**

Model checking MTL over TTGs is \text{EXPSPACE}-complete.

Model checking MTL\(_{\leq,\geq}\) over TTGs is \text{PSPACE}-complete.

**Proof.**

- hardness in \text{EXPSPACE}: encode a \(2^n\)-tape Turing machine.
  - the sequence of configurations of the TM is encoded as a sequence of \(2^n\)-long segments of the execution;
  - enforce correct transitions using \(F_{\geq 2^n}\).
Outline of the presentation

1 Introduction

2 Discrete-time systems
   - Model Checking Timed Transition Graphs
   - Playing Discrete-Time Games

3 Dense-time systems: timed automata
   - Model Checking Timed Automata
   - Dense-time Games

4 Hot topics...
   - Robustness of timed automata
   - Priced timed automata and priced timed games
   - Other related models

5 Conclusion
Finite Durational Concurrent Game Structures

**Definition**

A *durational concurrent game structure* is an 8-tuple $\mathcal{G} = \langle Q, Q_0, T, \ell, \text{Agt}, \mathcal{M}, \text{Mv}, \text{Tab} \rangle$

- $\langle Q, Q_0, T, \ell \rangle$ is a TTG,
- $\text{Agt} = \{A_1, \ldots, A_n\}$ is a (finite) set of players,
- $\mathcal{M}$ is a set of moves of the agents,
- $\text{Mv}: Q \times \text{Agt} \to \mathcal{P}(\mathcal{M}) \setminus \{\emptyset\}$ indicates the set of moves available to each agent in each location,
- $\text{Tab}: Q \times \mathcal{M}^n \to T$ is the *transition table*. 

A *finite DCGS* is a DCGS such that $\langle Q, Q_0, T, \ell \rangle$ is a finite TTG, $\mathcal{M}$ is finite.
## Finite Durational Concurrent Game Structures

### Definition

A *durational concurrent game structure* is an 8-tuple \( G = \langle Q, Q_0, T, \ell, \text{Agt}, M, Mv, \text{Tab} \rangle \)

- \( \langle Q, Q_0, T, \ell \rangle \) is a TTG,
- \( \text{Agt} = \{ A_1, \ldots, A_n \} \) is a (finite) set of players,
- \( M \) is a set of moves of the agents,
- \( Mv: Q \times \text{Agt} \rightarrow \mathcal{P}(M) \setminus \{\emptyset\} \) indicates the set of moves available to each agent in each location,
- \( \text{Tab}: Q \times M^n \rightarrow T \) is the *transition table*.

A *finite DCGS* is a DCGS such that

- \( \langle Q, Q_0, T, \ell \rangle \) is a finite TTG,
- \( M \) is finite.
Finite Durational Concurrent Game Structures

Example

\[
\begin{aligned}
\langle a, a \rangle \langle a, b \rangle \\
\langle b, a \rangle \\
\langle b, b \rangle \\
\end{aligned}
\]

\[
\begin{aligned}
\text{Tab}(\langle a, a \rangle) = 2 \\
\text{Tab}(\langle a, b \rangle) = 2 \\
\text{Tab}(\langle b, a \rangle) = 5 \\
\text{Tab}(\langle b, b \rangle) = 3 \\
\end{aligned}
\]
Finite Durational Concurrent Game Structures

Example

\[
\begin{align*}
\text{Tab}(\langle a, a \rangle) & = 2 \\
\text{Tab}(\langle a, b \rangle) & = 2 \\
\text{Tab}(\langle b, a \rangle) & = 5 \\
\text{Tab}(\langle b, b \rangle) & = 3
\end{align*}
\]
Finite Durational Concurrent Game Structures

Example

\[
\begin{align*}
\langle a, a \rangle & \quad \langle a, b \rangle \\
\langle b, a \rangle & \quad \langle b, b \rangle
\end{align*}
\]

\[
\begin{align*}
\text{Tab}(\langle a, a \rangle) &= 2 \\
\text{Tab}(\langle a, b \rangle) &= 2 \\
\text{Tab}(\langle b, a \rangle) &= 5 \\
\text{Tab}(\langle b, b \rangle) &= 3
\end{align*}
\]
Strategies, outcomes

Definition

A *strategy* for player $A_i$ is a function

$$\sigma_i : T^* \rightarrow M_v$$
A strategy for player $A_i$ is a function

$$\sigma_i : T^* \rightarrow M_v$$

Example

\[
\begin{align*}
\langle a, a \rangle & \quad 2 \\
\langle a, b \rangle & \quad 5 \\
\langle b, a \rangle & \quad 3 \\
\langle b, b \rangle & 
\end{align*}
\]
Strategies, outcomes

**Definition**

A *strategy* for player $A_i$ is a function

$$\sigma_i : T^* \rightarrow M_v$$

**Example**

$$\sigma_1(\rightarrow \bullet) = b$$
A strategy for player $A_i$ is a function

$$\sigma_i : T^* \rightarrow Mv$$

Example

$$\sigma_1(\rightarrow \bullet) = b$$
Strategies, outcomes

**Definition**

A *strategy* for player $A_i$ is a function

$$\sigma_i : T^* \rightarrow Mv$$

**Example**

![Diagram](image-url)
Strategies, outcomes

**Definition**

A *strategy* for player $A_i$ is a function

$$\sigma_i : T^* \rightarrow M_v$$

**Example**

Is there a strategy for player 1 to reach $\bigcirc$ in exactly 10 time units?

![Diagram](image)
Strategies, outcomes

**Definition**

A *strategy* for player $A_i$ is a function

$$\sigma_i : T^* \rightarrow M_v$$

**Example**

Is there a strategy for player 1 to reach $\bigcirc$ in exactly 10 time units?

- $\sigma_1(\bigcirc) = a$
- $\sigma_1(\bigcirc) = b$
- $\sigma_1(\bigcirc) = a$
- $\sigma_1(\bigcirc) = a$
Strategies, outcomes

Definition

A strategy for player $A_i$ is a function

$$\sigma_i : T^* \rightarrow \mathcal{M}_v$$

Example

Is there a strategy for player 1 to reach $\bigcirc$ in exactly 10 time units?

$$\sigma_1(\bigcirc) = a$$
$$\sigma_1(\bigcirc) = b$$
$$\sigma_1(\bigcirc) = a$$
$$\sigma_1(\bigcirc) = a$$
Strategies, outcomes

Definition

A *strategy* for player $A_i$ is a function

$$\sigma_i: T^* \rightarrow M_v$$

Example

Is there a strategy for player 1 to reach $\bigcirc$ in exactly 10 time units?

$$\sigma_1(\bigcirc) = a$$
$$\sigma_1(\bigcirc) = b$$
$$\sigma_1(\bigcirc) = a$$
$$\sigma_1(\bigcirc) = a$$
Strategies, outcomes

**Definition**

A *strategy* for player $A_i$ is a function

$$\sigma_i : T^* \rightarrow M_v$$

**Example**

Is there a strategy for player 1 to reach $\bullet$ in exactly 10 time units?

$$\sigma_1(\bullet) = a$$

$$\sigma_1(\longrightarrow) = b$$

$$\sigma_1(\longrightarrow) = a$$

$$\sigma_1(\longrightarrow) = a$$
Strategies, outcomes

Definition

A *strategy* for player $A_i$ is a function

$$\sigma_i : T^* \rightarrow M_v$$

Example

Is there a strategy for player 1 to reach $\circ$ in exactly 10 time units?

$$\sigma_1(\circ) = a$$

$$\sigma_1(\circ) = b$$

$$\sigma_1(\circ) = a$$

$$\sigma_1(\circ) = a$$
Strategies, outcomes

**Definition**

A *strategy* for player $A_i$ is a function

$$\sigma_i : T^* \rightarrow M_v$$

**Definition**

\[
\text{TATL } \exists \varphi ::= \bigcirc | \smiley | \ldots | \neg \varphi | \varphi \lor \varphi |
\]

\[
\langle\langle C\rangle\rangle \varphi \ U_{\sim k} \varphi | \langle\langle C\rangle\rangle \varphi \ R_{\sim k} \varphi
\]

where $\langle\langle C\rangle\rangle \varphi$ reads:

“Coalition $C$ has a strategy to enforce $\varphi$.”
Strategies, outcomes

Definition

A *strategy* for player $A_i$ is a function

$$\sigma_i : T^* \rightarrow Mv$$

Definition

$$\exists \varphi ::= \bigcirc | \bigcirc | \ldots | \neg \varphi | \varphi \lor \varphi |$$

$$\langle \langle C \rangle \rangle \varphi \ U_{\sim k} \varphi \ | \ \langle \langle C \rangle \rangle \varphi \ R_{\sim k} \varphi$$

where $\langle \langle C \rangle \rangle \varphi$ reads:

“Coalition $C$ has a strategy to enforce $\varphi$”.

$\text{TATL}_{\leq, \geq}$ is TATL with no punctual constraints.
Model checking TATL

<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
</table>

Model checking $\text{TATL}_{\leq,\geq}$ over DCGSs (with positive durations) is P-complete.
Model checking TATL

**Theorem**

Model checking $\text{TATL}_{\leq, \geq}$ over DCGSs (with positive durations) is $\mathcal{P}$-complete.

**Proof.**

\[ \langle A_1 \rangle \cup_{\leq} U \leq k \]

**Lemma**

Memoryless strategies are sufficient for this kind of objectives.

\[ \rightsquigarrow \text{compute controllable predecessors together with minimal duration of reaching} \]

in less than $i$ steps.
Model checking TATL

**Theorem**

Model checking $\text{TATL}_{\leq, \geq}$ over DCGSs (with positive durations) is P-complete.

**Proof.**

- $\langle A_1 \rangle \cup U_{\leq k}$:
  - Compute controllable predecessors together with minimal duration of reaching in less than $i$ steps.

**Example**

- Diagram of a simple model with transitions labeled with pairs $(a,b)$ and $(b,a)$, and durations 1, 10, and 20.
- Table showing states $a, b, c, d, e$ with transition probabilities and duration.
Model checking TATL \( \leq, \geq \) over DCGSs (with positive durations) is P-complete.

Proof.

\[ \langle A_1 \rangle \cup U_{\leq k} \quad : \quad \text{compute controllable predecessors together with minimal duration of reaching} \quad \text{in less than} \quad i \text{ steps}. \]

Example

- (a, a) = 1
- (a, b) = 1
- (b, a) = 1
- (b, b) = 1
- (c, b) = 10
- (d, b) = 20

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>+∞</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>+∞</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>+∞</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Model checking TATL\(\leq,\geq\) over DCGSs (with positive durations) is \(\text{P-complete}\).

Proof.

\[\langle A_1 \rangle \cup \leq_k : \sim \text{ compute controllable predecessors together with minimal duration of reaching in less than } i \text{ steps.}\]

**Example**

![Example Diagram]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>+(\infty)</td>
<td>+(\infty)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(c)</td>
<td></td>
<td>+(\infty)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(d)</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(e)</td>
<td></td>
<td>+(\infty)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Model checking TATL

Theorem

Model checking $\text{TATL}_{\leq, \geq}$ over DCGSs (with positive durations) is $\text{P}$-complete.

Proof.

- $\langle A_1 \rangle \cup \leq_k \geq_k$:
  - compute controllable predecessors together with minimal duration of reaching in less than $i$ steps.

Example

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>+∞</td>
<td>+∞</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>+∞</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>+∞</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Theorem

Model checking $\text{TATL}_{\leq, \geq}$ over DCGSs (with positive durations) is $\mathsf{P}$-complete.

Proof.

- $\langle\langle A_1 \rangle\rangle \cup \leq_k$:
  - $\leadsto$ compute controllable predecessors together with minimal duration of reaching in less than $i$ steps.

Example

\begin{table}[h]
\begin{tabular}{c|ccc}
  & 0 & 1 & 2 & 3 \\
\hline
$\text{a}$ & $+\infty$ & $+\infty$ &  &  \\
$\text{b}$ & 0 & 0 &  &  \\
$\text{c}$ &  &  & $+\infty$ & 20 \\
$\text{d}$ &  &  &  &  \\
$\text{e}$ &  &  &  & $+\infty$ \\
\end{tabular}
\end{table}
Model checking TATL

Theorem

Model checking $\text{TATL}_{\leq, \geq}$ over DCGSs (with positive durations) is P-complete.

Proof.

- $\langle A_1 \rangle \cup U_{\leq k}$:
  - $\leadsto$ compute controllable predecessors together with minimal duration of reaching in less than $i$ steps.

Example

```
\begin{array}{c|c|c|c|c}
   & 0 & 1 & 2 & 3 \\
\hline
a  & +\infty & +\infty &   &   \\
b  & 0     & 0     &   &   \\
c  & +\infty & 20    &   &   \\
d  & 0     & 0     &   &   \\
e  & +\infty &       &   &   \\
\end{array}
```
Model checking \( \text{TATL}_{\leq, \geq} \) over DCGSs (with positive durations) is P-complete.

Proof.

\[ \langle A_1 \rangle \cup_{\leq_k} : \]
\[ \sim \text{ compute controllable predecessors together with minimal duration of reaching} \] in less than \( i \) steps.

Example

\[ \begin{array}{c|ccc|}
 & 0 & 1 & 2 & 3 \\
\hline
a & +\infty & +\infty &  &  \\
\hline
b & 0 & 0 &  &  \\
\hline
c & +\infty & 20 &  &  \\
\hline
d & 0 & 0 &  &  \\
\hline
e & +\infty & +\infty &  &  \\
\end{array} \]
Model checking TATL

Theorem

Model checking TATL\(_{\leq,\geq}\) over DCGSs (with positive durations) is P-complete.

Proof.

\[\langle A_1 \rangle \cup_{\leq} k : \sim \text{ compute controllable predecessors together with minimal duration of reaching in less than } i \text{ steps.}\]

Example

\[
\begin{array}{c|c|c|c|c}
& 0 & 1 & 2 & 3 \\
\hline
a & +\infty & +\infty & 21 \\
b & 0 & 0 & 0 \\
c & +\infty & 20 & 20 \\
d & 0 & 0 & 0 \\
e & +\infty & +\infty & +\infty \\
\end{array}
\]
Model checking TATL

**Theorem**

Model checking TATL$_{\leq, \geq}$ over DCGSs (with positive durations) is P-complete.

**Proof.**

- $\langle A_1 \rangle \bigcirc U_{\leq k}$
  - $\leadsto$ compute controllable predecessors together with minimal duration of reaching in less than $i$ steps.

**Example**

![Diagram with labels and transitions]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>+∞</td>
<td>+∞</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>+∞</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e</td>
<td>+∞</td>
<td>+∞</td>
<td>+∞</td>
<td>+∞</td>
</tr>
</tbody>
</table>
Model checking TATL

**Theorem**

Model checking \( \text{TATL}_{\leq, \geq} \) over DCGSs (with positive durations) is \( \text{P-complete} \).

**Proof.**

- \( \langle \langle A_1 \rangle \rangle U_{\leq k} \): compute controllable predecessors together with minimal duration of reaching in less than \( i \) steps.
- similar algorithms for other modalities.
Model checking TATL

Theorem

Model checking TATL over DCGSs (with positive durations) is EXPTIME-complete.
Model checking TATL

Theorem

Model checking TATL over DCGSs (with positive durations) is EXPTIME-complete.

Proof.

- $\langle A_1 \rangle$ $\bigcirc$ $U_{\geq k}$ $\bigcirc$:
  - Define the boolean table $T[\bigcirc, i]$ s.t.
    
    $$T[\bigcirc, i] = \top \iff \bigcirc \models \langle A_1 \rangle \bigcirc U_{\geq i} \bigcirc.$$

  - $T$ can be computed in time $O(|\text{Tab}| \times k)$.
Model checking TATL

Theorem

Model checking TATL over DCGSs (with positive durations) is EXPTIME-complete.

Proof.

- $\langle A_1 \rangle \cup U \leq k$:
  - Define the boolean table $T[\bigcirc, i]$ s.t.
    \[ T[\bigcirc, i] = \top \iff \bigcirc \models \langle A_1 \rangle \cup U \leq i. \]
  - $T$ can be computed in time $O(|\text{Tab}| \times k)$.

- EXPTIME-hardness: encode a linear-tape alternating Turing machine (omitted).
Conclusion for discrete-time

- a basic notion of time, still allowing quantitative reasoning;
- efficient algorithms as long as no punctual requirement is involved.

<table>
<thead>
<tr>
<th></th>
<th>unitary TTGs</th>
<th>jump semantics</th>
<th>continuous semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCTL$_{\leq,\geq}$</td>
<td>P-complete</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCTL</td>
<td>P-c.</td>
<td>$\Delta^p_2$-c.</td>
<td>PSPACE-c.</td>
</tr>
<tr>
<td>MTL$_{\leq,\geq}$</td>
<td></td>
<td></td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>MTL</td>
<td></td>
<td></td>
<td>EXPSPACE-complete</td>
</tr>
<tr>
<td>TATL$_{\leq,\geq}$</td>
<td>P-complete</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TATL</td>
<td>P-c.</td>
<td></td>
<td>EXPTIME-c.</td>
</tr>
</tbody>
</table>
Conclusion for discrete-time

- a basic notion of time, still allowing quantitative reasoning;
- efficient algorithms as long as no punctual requirement is involved.

<table>
<thead>
<tr>
<th></th>
<th>unitary TTGs</th>
<th>jump semantics</th>
<th>continuous semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCTL&lt;,&gt;</td>
<td></td>
<td>P-complete</td>
<td></td>
</tr>
<tr>
<td>TCTL</td>
<td></td>
<td>P-c.</td>
<td>$\Delta^p_2$-c.</td>
</tr>
<tr>
<td>MTL&lt;,&gt;</td>
<td></td>
<td></td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>MTL</td>
<td></td>
<td></td>
<td>EXPSPACE-complete</td>
</tr>
<tr>
<td>TATL&lt;,&gt;</td>
<td></td>
<td>P-complete</td>
<td></td>
</tr>
<tr>
<td>TATL</td>
<td></td>
<td>P-c.</td>
<td>EXPTIME-c.</td>
</tr>
</tbody>
</table>

But in some situations, discrete-time is not sufficient...
Outline of the presentation

1. Introduction

2. Discrete-time systems
   - Model Checking Timed Transition Graphs
   - Playing Discrete-Time Games

3. Dense-time systems: timed automata
   - Model Checking Timed Automata
   - Dense-time Games

4. Hot topics...
   - Robustness of timed automata
   - Priced timed automata and priced timed games
   - Other related models

5. Conclusion
Why *dense* time?

Example

- **Stable configurations:**
  - When $i = 0$, the output is $o = [0, 1, 0]$.
  - When $i = 1$, the output is $o = [1, 1, 0]$. 
Why *dense* time?

Example

Stable configurations:

- when $i = 0$, the output is $o = [101]$;
- when $i = 1$, the output is $o = [011]$. 
Why *dense* time?

Example

When $i$ changes from 0 to 1:

\[
\begin{align*}
[101] & \xrightarrow{o_2 \ | \ 1.2} [111] & \xrightarrow{o_3 \ | \ 2.3} [110] & \xrightarrow{o_1 \ | \ 2.6} [010] \quad & \xrightarrow{o_3 \ | \ 3.4} [011]
\end{align*}
\]
Why *dense* time?

Example
Why *dense* time?

**Example**

\[\begin{align*}
[11100000] & \xrightarrow{o_1} [01100000] & \xrightarrow{o_2, o_3, o_4, o_5} & [00000000]; \\
[11100000] & \xrightarrow{o_1, o_2} [00100000] & \xrightarrow{o_3, o_5, o_6} & [00001100] & \xrightarrow{o_5, o_6} & [00000000]; \\
[11100000] & \xrightarrow{o_1, o_3} [01000000] & \xrightarrow{o_2, o_4, o_6} & [00010100] & \xrightarrow{o_4, o_6} & [00000000]; \\
[11100000] & \xrightarrow{o_1, o_2, o_3} [00000000];
\end{align*}\]
Why dense time?

Example

\[
\begin{align*}
\begin{array}{c}
\text{\textcolor{red}{[11100000]}} \xrightarrow{o_1} \text{\textcolor{blue}{[01100000]}} \xrightarrow{o_2, o_5} \text{\textcolor{red}{[00000100]}} \\
\xrightarrow{o_5, o_7} \text{\textcolor{blue}{[00000010]}} \xrightarrow{o_7, o_8} \text{\textcolor{blue}{[00000001]}}
\end{array}
\end{align*}
\]
Why *dense* time?

**Theorem**

*For every $k \geq 1$, there exists a digital circuit such that the reachability set of states in dense-time is strictly larger than the one in discrete time (with granularity $1/k$).*
Why *dense* time?

**Theorem**

For every $k \geq 1$, there exists a digital circuit such that the reachability set of states in dense-time is strictly larger than the one in discrete time (with granularity $1/k$).

Finding a correct granularity might be as difficult as model checking the circuit in dense time.
Why *dense* time?

- dense-time semantics for timed automata;
Why *dense* time?

- dense-time semantics for timed automata;
- extensions, tools...
Timed automata

Definition

A **timed automaton** is an 8-tuple $A = \langle Q, Q_0, T, \ell, C, G, I, R \rangle$ such that:

- $\langle Q, Q_0, T, \ell \rangle$ is a finite transition system;
- $C$ is a finite set of **clocks**;
- $G: T \rightarrow \text{Constr}(C)$ imposes conditions for firing a transition;
- $I: Q \rightarrow \text{Constr}(C)$ imposes conditions for staying in a location;
- $R: T \rightarrow 2^C$ indicates which clocks are reset when firing a transition.

Definition

$$\text{Constr}(C) \ni g ::= x \sim n \mid g \land g$$

with $x \in C$, $\sim \in \{<, \leq, =, \geq, >\}$, $n \in \mathbb{N}$. 
Timed automata

Example

$y = 1, y := 0$

$y \leq 2$

$x \leq 2, x := 0$

$y \geq 2, y := 0$

$x = 0, y = 2$

$x \leq 1$

$x := 0$
Timed automata

Example

\[
\begin{align*}
&x \leq 3 \\
y \leq 2 \\
&x \leq 3 \\
&x \leq 1, \quad y := 0 \\
&x \leq 2, \quad x := 0 \\
&y \geq 2, \quad y := 0 \\
&x = 0, \quad y = 2 \\
&x = 0, \quad y = 0 \\
&x = 1, \quad y = 0.8 \\
&x = 0, \quad y = 0.6 \\
&x = 1.3, \quad y = 0 \\
&x = 1.9, \quad y = 0.6 \\
&x = 1, \quad y = 0.9 \\
&x = 0, \quad y = 0 \\
&x = 1, \quad y = 0.3 \\
&x = 0, \quad y = 0 \\
\end{align*}
\]
Timed automata

Example

\[
\begin{align*}
\text{\textcolor{red}{x}} & \leq 3 \\
\text{\textcolor{blue}{y}} & \leq 2 \\
\text{\textcolor{blue}{x}} & \leq 3 \\
\text{\textcolor{red}{y}} & \leq 1 \\
\text{\textcolor{red}{x}} & = 1 \\
\text{\textcolor{red}{y}} & = 1 \\
\text{\textcolor{blue}{x}} & = 1 \\
\text{\textcolor{gray}{y}} & = 1.8 \\
\text{\textcolor{gray}{x}} & = 0 \\
\text{\textcolor{gray}{y}} & = 0.8 \\
\text{\textcolor{gray}{x}} & = 1 \\
\text{\textcolor{red}{y}} & = 0 \\
\text{\textcolor{gray}{x}} & = 1 \\
\text{\textcolor{gray}{y}} & = 0.6 \\
\text{\textcolor{green}{x}} & = 1 \\
\text{\textcolor{green}{y}} & = 0 \\
\text{\textcolor{green}{x}} & = 0 \\
\text{\textcolor{green}{y}} & = 0 \\
\end{align*}
\]
Timed automata

Example

\begin{align*}
&x \leq 3, \\
y \leq 2, \\
x \leq 3, \\
y \leq 1, \\
y = 1, \\
x = 0, y = 0, \\
x = 0, y = 0.8, \\
x = 0, y = 0.6, \\
x = 1, y = 1, \\
x = 1, y = 0.9, \\
x = 1, y = 1.3, \\
x = 1, y = 0.8, \\
x = 0, y = 0.9, \\
x = 0, y = 0.6.
\end{align*}
Timed automata

Example

- State 1: $y = 1$, $y := 0$
- State 2: $y \leq 2$
- State 3: $x = 0$, $y = 2$
- State 4: $x \leq 1$

Transitions:
- From State 1 to State 2: $x \leq 2$, $x := 0$
- From State 2 to State 3: $y \geq 2$, $y := 0$
- From State 3 to State 4: $x = 0$, $y = 2$
- From State 4 to State 1: $x = 0$, $y = 0$

The question is: is the state $x = 0$, $y = 0$ reachable?
Timed automata

**Example**

- **State 1**: \( x = 1, y = 0 \)
  - Edge: \( y \leq 2 \)
  - Action: \( x \leq 2, x := 0 \)
- **State 2**: \( y \leq 2 \)
  - Edge: \( y \geq 2, y := 0 \)
- **State 3**: \( x = 0, y = 2 \)
  - Edge: \( x \leq 1 \)
  - Action: \( x := 0 \)

**State Transition Diagram**

- States: \( x = 0 \), \( y = 0 \)

**Observations**

- \( x = 0 \)
- \( y = 0 \)
Timed automata

Example

\begin{align*}
x \leq 2, & \quad x := 0 \\
y \geq 2, & \quad y := 0 \\
x = 0, & \quad y = 2 \\
x = 0 \\
y = 0, & \quad y = 1 \\
x = 1 \\
y = 1 \\
x = 1.3 \\
y = 2.1 \\
x = 1.9 \\
y = 0.6 \\
x = 0 \\
y = 0.
\end{align*}

\begin{tikzpicture}[node distance=2cm, every state/.style={draw}]  
  \node[state] (s0) at (0,0) {$y=1, \ y:=0$};  
  \node[state] (s1) at (2,0) {$y \leq 2$};  
  \node[state] (s2) at (4,0) {$x=0, \ y=2$};  
  \node[state] (s3) at (6,0) {$x \leq 1$};  
  \draw[->] (s0) -- (s1) node [midway, above] {$x \leq 2, \ x:=0$};  
  \draw[->] (s1) -- (s2) node [midway, left] {$y \geq 2, \ y:=0$};  
  \draw[->] (s2) -- (s3) node [midway, above] {$x=0, \ y=2$};  
  \draw[->] (s3) -- (s0) node [midway, right] {$x:=0$};  
\end{tikzpicture}
Timed automata

Example

- $x = 0, y = 0$
- $x = 1, y = 0$
- $x = 1, y = 0.6$
- $x = 1.3, y = 0$
- $x = 1.3, y = 0.8$
- $x = 0, y = 0$
- $x = 2, y = 0$
- $x = 0, y = 2$
- $x = 0, y = 1$
- $x = 0, y = 0$
- $x = 0, y = 0$

$y = 1, y := 0$

$y \leq 2$

$x \leq 2, x := 0$

$y \geq 2, y := 0$

$x = 0, y = 2$

$x \leq 1$

$x := 0$

0 1 2 3 4 5 6 7
Timed automata

Example

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

\[ x = 0 \quad x = 1 \quad x = 1.8 \]
\[ y = 0 \quad y = 0 \quad y = 0.8 \]
Timed automata

Example

- \( x \leq 2, x := 0 \)
- \( x := 0, y = 2 \)
- \( y \geq 2, y := 0 \)
- \( y := 0, y = 0.8 \)

Is the state reachable?
Timed automata

Example

\[
\begin{aligned}
&x \leq 2, \quad x := 0 \\
y \geq 2, \quad y := 0 \\
x = 0, \quad y = 0 \\
x = 0, \quad y = 2 \\
x = 1, \quad y = 0 \\
x = 1.3, \quad y = 0 \\
\end{aligned}
\]
Timed automata

Example

\begin{align*}
    & x \leq 2, \ x := 0 \\
    & y \leq 2 \\
    & x = 0, \ y = 2 \\
    & x \leq 1 \\
    & x := 0
\end{align*}

\begin{align*}
    & x = 0 \quad x = 1 \quad x = 0 \quad x = 1.3 \\
    & y = 0 \quad y = 0 \quad y = 0.8 \quad y = 0
\end{align*}
Timed automata

Example

$x \leq 2, \ x := 0$

$y \geq 2, \ y := 0$

$x = 0, \ y = 2$

$x \leq 1$

$x := 0$

0 1 2 3 4 5 6 7

$x = 0 \quad x = 1 \quad x = 0 \quad x = 1.3 \times 1.9$

$y = 0 \quad y = 0 \quad y = 0 \quad y = 0 \quad y = 0.8 \quad y = 0 \quad y = 0.6$
Timed automata

Example

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

\[
\begin{align*}
x &\leq 1,
\end{align*}
\]

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

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

Example

Is the state reachable?
Semantics of timed automata

Definition

Given a timed automaton \( \mathcal{A} = \langle Q, Q_0, T, \ell, C, G, I, R \rangle \), its semantics is the (infinite) TTG \( S_\mathcal{A} = \langle S, S_0, \delta, e \rangle \) s.t.

- \( S = \{ (q, v) \mid q \in Q, \ v: C \to \mathbb{R}_{\geq 0}, \ v \models I(q) \} \),
- \( S_0 = (Q_0 \times \{0\}) \cap S \);
- \( e(q, v) = \ell(q) \) for each \( (q, v) \in S \),
- \( \delta \) contains three types of transitions:
  - delay transitions: \((q, v) \xrightarrow{d} (q, v + d)\) provided that \( d \in \mathbb{R}_{>0} \), and \((q, v)\) and \((q, v + d)\) are in \( S \);
  - action transitions: \((q, v) \xrightarrow{a} (q', v')\) if there is a transition \((q, q')\) s.t. \( v \models g(q, q') \), \( v' = v[r(q, q') \leftarrow 0] \), and \((q, v)\) and \((q', v')\) are in \( S \);
  - mixed transitions: \((q, v) \xrightarrow{m} (q', v')\) if \((q, v) \xrightarrow{d} \xrightarrow{a} (q', v') \).
Timed automata

Example

- $y=1$, $y:=0$
- $x \leq 2$, $x:=0$
- $x=0$, $y=2$
- $x \leq 1$
- $x:=0$
- $y \geq 2$, $y:=0$

Is the $\bigcirc$-state reachable?
Timed automata

Example

![Timed automaton diagram]

Is the state reachable?
Timed automata

Example

is the \( \bigcirc \)-state reachable?
Timed automata

Example

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

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

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

\[ y = 1 \]
\[ y := 0 \]
\[ y = 2 \]
\[ y := 0 \]

is the \( \bigcirc \)-state reachable?
Timed automata

Example

is the ◦-state reachable?
Timed automata

Example

\[ x \leq 3 \]
\[ y \leq 2 \]

\[ x = 0, \quad y = 2 \]
\[ x = 1 \]

is the \( \circ \)-state reachable?
Timed automata

Example

\[ x \leq 3 \]
\[ y \leq 2 \]
\[ x \leq 1 \]
\[ y = 1, y := 0 \]
\[ x = 0, y = 2 \]
\[ x = 0 \]

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

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

\( \bullet \) is the \( \bigcirc \)-state reachable?
Timed automata

Example

\[ x \leq 3 \]
\[ y \leq 2 \]
\[ x \leq 3 \]
\[ x \leq 1 \]
\[ y \geq 2 \]
\[ y = 0 \]
\[ x = 0, y = 2 \]
\[ x = 0, y = 2 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]
\[ x = 0 \]

is the \( \bigcirc \)-state reachable?
Timed automata

Example

```
x ≤ 3
y ≤ 2
x ≤ 3
x ≤ 1
y = 1, y := 0
y ≤ 2
x = 0, y = 2
x = 0
y ≥ 2
y = 0
x = 0
x = 0
x = 0
x = 0
x = 0
y = 2
y = 0
x = 1
y = 1
x = 1
x = 1
y = 0
y = 0
y = 0
x = 2
x = 2
x = 2
y = 0
y = 0
y = 0
```

is the ○-state reachable?
Region equivalence

**Definition**

Let $A$ be a timed automaton, and $M$ be the largest integer appearing in $A$. Two states $(q, v)$ and $(q', v')$ are equivalent, written $(q, v) \sim (q', v')$, if the following conditions hold:

- $q = q'$,
- for any $x \in C$, either $v(x)$ and $v'(x)$ are larger than $M$, or
  \[ \lfloor v(x) \rfloor = \lfloor v'(x) \rfloor \]
- for any $x, y \in C$ with $v(x) \leq M$ and $v(y) \leq M$, we have
  \[ \text{fract}(v(x)) \leq \text{fract}(v(y)) \iff \text{fract}(v'(x)) \leq \text{fract}(v'(y)) \]
- for any $x \in C$ with $v(x) \leq M$, we have
  \[ \text{fract}(v(x)) = 0 \iff \text{fract}(v'(x)) = 0 \]
Region equivalence

Example
Region equivalence

Example
Region equivalence

Example
Region equivalence

Example
Region equivalence

Example
Region equivalence

Example
Region equivalence

Example
Region equivalence

Example
Region equivalence

Example
Lemma

Two equivalent states can reach the same set of regions, and can be reached from the same set of regions.
Lemma

Two equivalent states can reach the same set of regions, and can be reached from the same set of regions.

Definition

Given a timed automaton \( \mathcal{A} \), its \textit{region automaton} is the transition system \( \mathcal{R}_\mathcal{A} = \langle R, R_0, E, l \rangle \) where

- \( R \) is the set of regions of \( \mathcal{A} \);
- \( R_0 = Q_0 \times r_0 \);
- \((q, r) \rightarrow (q', r')\) if there exists \( v \in r \) and \( v' \in r' \) s.t. \((q, v) \rightarrow (q', v')\) in \( S_\mathcal{A} \);
- \( l(q, r) = \ell(q) \).
Region automaton

Example
Region automaton

Example

\[
\begin{align*}
  x & \leq 3 \\
  y & \leq 2 \\
  x & \leq 3 \\
  y & = 1 \\
  x & = 0 \\
  y & = 2 \\
  x & = 0
\end{align*}
\]
Reachability

Theorem

Deciding whether a location $q$ is reachable in a timed automaton $A$ is PSPACE-complete.
Reachability

**Theorem**

*Deciding whether a location $q$ is reachable in a timed automaton $A$ is PSPACE-complete.*

**Proof.**

- **Membership in PSPACE:**
  - there are finitely many regions, at most
  
  $$|Q| \times |C|! \cdot (4M + 4)^{|C|}$$

  - non-deterministically guess a sequence of regions leading to location $q$. 
Reachability

Proof.

- Hardness in PSPACE: simulate a linear-space Turing machine.
Reachability

Proof.

- Hardness in PSPACE: simulate a linear-space Turing machine.

  Cell $c_j$ on the tape contains 1 if $x_j = y_j$;
  Cell $c_j$ on the tape contains 0 if $x_j \neq y_j$. 
Reachability

Proof.

- Hardness in PSPACE: simulate a linear-space Turing machine.

Cell $c_j$ on the tape contains 1 if $x_j = y_j$;
Cell $c_j$ on the tape contains 0 if $x_j \neq y_j$.

For example: a transition $(q, 0, 1, \text{right}, q')$ is encoded as

- $x_j = 1, \ x_j := 0$
- $y_j = 1, \ y_j := 0$
- $x_i = 0, \ y_i > 0$
- $x_i := 0, \ y_i := 0$
- $x_j = 1, \ x_j := 0$
- $y_j = 1, \ y_j := 0$
Reachability

Proof.

- Hardness in PSPACE: simulate a linear-space Turing machine.

Cell $c_j$ on the tape contains 1 if $x_j = y_j$;
Cell $c_j$ on the tape contains 0 if $x_j \neq y_j$.

For example: a transition $(q, 0, 1, \text{right}, q')$ is encoded as

$x_j = 1, x_j := 0$
$x_i = 0, y_i > 0$
$x_i := 0, y_i := 0$
$y_j = 1, y_j := 0$

The Turing machine accepts iff location $q_{\text{acc}}$ is reachable.
Safety

Example

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

Does there exist a path never visiting the \( \bigcirc \)-state?
Safety

Example

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

Does there exist a path never visiting the \( \bigcirc \)-state?
Safety

Example

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

Does there exist a path never visiting the \( \bullet \)-state?
Safety

Theorem

Safety under non-Zenoness assumption is PSPACE-complete.

Proof.

Hardness: simulate a linear-space Turing machine;

Membership: add an extra tick-clock, which is reset each time it reaches 1; check existence of a safe path that resets the tick-clock infinitely many times.
Theorem

Safety under non-Zenoness assumption is PSPACE-complete.

Proof.

- **Hardness**: simulate a linear-space Turing machine;
Safety

Theorem

Safety under non-Zenoness assumption is \textit{PSPACE}-complete.

\textit{Proof}.

- **Hardness**: simulate a linear-space Turing machine;
- **Membership**:
  - add an extra \textit{tick-clock}, which is reset each time it reaches 1;
  - check existence of a \textit{safe path} that resets the \textit{tick-clock} infinitely many times.
Temporal Logics

Example

\[ x \leq 3 \]

\[ y \leq 2 \]

\[ x = 0, y = 2 \]

\[ x = 0 \]

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

\[ y \leq 2 \]

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

\[ x \leq 1 \]

\[ x := 0 \]
Temporal Logics

Example

- **jump (or pointwise) semantics:**
Temporal Logics

Example

- **Jump (or pointwise) semantics:**

- **Continuous semantics:**
Temporal Logics

Example

- *jump semantics:*

  $\neg (U \leq 2) \land \neg (F = 2 \land F = 3 \land F \leq 1)$

- *continuous semantics:*

  $\neg (\neg (U \leq 2))$
Temporal Logics

Example

- **jump semantics:**

  \[ \neg (F_{=2}) \]

- **continuous semantics:**

  \[ F_{=2} \]
Temporal Logics

Example

- **jump semantics:**

\[ \neg (F_3(F_{\leq 1} \circ)) \]

- **continuous semantics:**

\[ F_3(F_{\leq 1} \circ) \]
TCTL mode-checking

Theorem

*TCTL model-checking over timed automata is PSPACE-complete.*
TCTL mode-checking

**Theorem**

*TCTL model-checking over timed automata is PSPACE-complete.*

**Proof.**

- Hardness in PSPACE: follows from hardness of reachability;
Theorem

\textit{TCTL model-checking over timed automata is PSPACE-complete.}

Proof.

- Hardness in PSPACE: follows from hardness of reachability;
- Membership in PSPACE:

Lemma

\textit{Two states in the same region satisfy the same TCTL formulas.}
TCTL mode-checking

**Theorem**

*TCTL model-checking over timed automata is PSPACE-complete.*

**Proof.**

- Hardness in PSPACE: follows from hardness of reachability;
- Membership in PSPACE:

**Lemma**

*Two states in the same region satisfy the same TCTL formulas.*

\[ E F_{\geq 6} \equiv (t = 0).E F(\equiv t = 6). \]

- CTL algorithm on the extended region graph.
MTL model-checking

Theorem

MTL model-checking over timed automata is undecidable.
MTL model-checking

Theorem

MTL model-checking over timed automata is undecidable.

Proof.

Encode the halting problem of a Turing machine (assuming continuous semantics):

One time-unit = one configuration of the Turing machine
MTL model-checking

Theorem

MTL model-checking over timed automata is undecidable.

Proof.

Encode the halting problem of a Turing machine (assuming continuous semantics):

One time-unit = one configuration of the Turing machine
MTL model-checking

**Theorem**

*MTL model-checking over timed automata is undecidable.*

**Proof.**

Encode the halting problem of a Turing machine (assuming continuous semantics):

One time-unit = one configuration of the Turing machine
Theorem

MTL model-checking over timed automata is undecidable.

Proof.

Encode the halting problem of a Turing machine (assuming continuous semantics):

One time-unit = one configuration of the Turing machine

\[
G \left[ (\circ \land \neg \circ \land \neg \left( (\neg \circ \land \neg \bullet) \mathbf{U} \circ \right) \right] \iff F_{=1} \circ] \land \ldots
\]
Relaxing punctuality

**Definition**

MITL is the fragment of MTL where equality is not allowed. CoFlatMTL is the fragment of MTL defined as:

\[ \varphi ::= \bigcirc | \ldots | \varphi \vee \varphi | \varphi \wedge \varphi | \varphi \, \text{U}_I \, \varphi | \varphi \, \text{U}_J \, \psi | \varphi \, \text{R}_I \, \varphi | \psi \, \text{R}_J \, \varphi \]

where \( I \) ranges over *bounded* intervals, and \( \psi \) ranges over MITL.
Relaxing punctuality

**Definition**

MITL is the fragment of MTL where equality is not allowed. CoFlatMTL is the fragment of MTL defined as:

\[ \varphi ::= \bigcirc \mid \ldots \mid \varphi \lor \varphi \mid \varphi \land \varphi \mid \varphi \ U_I \varphi \mid \varphi \ U_J \psi \mid \varphi \ R_I \varphi \mid \psi \ R_J \varphi \]

where \( I \) ranges over *bounded* intervals, and \( \psi \) ranges over MITL.

**Theorem**

MITL and CoFlatMTL model-checking over timed automata are EXPSPACE-complete.

*Proof.*

Omitted.
Outline of the presentation

1. Introduction

2. Discrete-time systems
   - Model Checking Timed Transition Graphs
   - Playing Discrete-Time Games

3. Dense-time systems: timed automata
   - Model Checking Timed Automata
   - Dense-time Games

4. Hot topics...
   - Robustness of timed automata
   - Priced timed automata and priced timed games
   - Other related models

5. Conclusion
Dense-time games

How to control time elapsing?
Dense-time games

How to control time elapsing?

- **turn-based timed games**: the player controlling the current location is also responsible for deciding the date of the next transition.

Example
Dense-time games

How to control time elapsing?

- **turn-based timed games**: the player controlling the current location is also responsible for deciding the date of the next transition.

- **concurrent timed games**: time is handled on a first-come-first-served basis; transitions are partitionned amongst the players.

Example

```
x ≤ 1
x ≤ 2
x ≥ 2
x := 0
x ≤ 3
```
Dense-time games

Definition

A *timed game automaton* is a 10-tuple $\mathcal{G} = \langle Q, Q_0, T, \ell, \text{Agt}, \text{Own}, C, G, I, R \rangle$ where

- $\langle Q, Q_0, T, \ell, C, G, I, R \rangle$ is a timed automaton,
- $\text{Agt} = \{A_1, \ldots, A_n\}$ is a (finite) set of players,
- $\text{Own}: T \rightarrow \text{Agt}$ partitions the set of transitions between the players.
Dense-time games

**Definition**

A *timed game automaton* is a 10-tuple $\mathcal{G} = \langle Q, Q_0, T, \ell, \text{Agt}, \text{Own}, \mathcal{C}, \mathcal{G}, I, R \rangle$ where

- $\langle Q, Q_0, T, \ell, \mathcal{C}, \mathcal{G}, I, R \rangle$ is a timed automaton,
- $\text{Agt} = \{A_1, \ldots, A_n\}$ is a (finite) set of players,
- $\text{Own}: T \rightarrow \text{Agt}$ partitions the set of transitions between the players.

**Example**

![Dense-time games example diagram](image-url)
Dense-time games

Example

\[ x \leq 3 \]
\[ x \leq 1 \]
\[ x \geq 2 \]

Player 2 has no available transition.

Player 1: "I want to fire at date 0."

Player 2: "I want to fire at date 2."

Player 1: "I want to fire at date 1."

Player 2: "I want to fire at date 1."

Player 1: "I want to fire at date 1."

Player 2: "I want to fire at date 2."

Player 1: "I want to fire at date 4."

Player 2 has no available transition.
Dense-time games

Example

Player 1: “I want to fire \( \cdot \rightarrow \bullet \) at date 0.7.”
Player 2 has no available transition.
Dense-time games

Example

Player 1: “I want to fire $\bullet \rightarrow \bigcirc$ at date 0.7.”
Player 2 has no available transition.

Player 1: “I want to fire $\bigcirc \rightarrow \bullet$ at date 0.9.”
Player 2 has no available transition.

Player 1: “I want to fire $\bigcirc \rightarrow \bullet$ at date 1.9.”
Player 2 has no available transition.

Player 1: “I want to fire $\bigcirc \rightarrow \bullet$ at date 2.9.”
Player 2 has no available transition.

Player 1: “I want to fire $\bigcirc \rightarrow \bullet$ at date 4.5.”

$x = 0.7$
Dense-time games

Example

Player 1: “I want to fire \( \bullet \rightarrow \bullet \) at date 1.4.”
Player 2: “I want to fire \( \bullet \rightarrow \bullet \) at date 0.9.”

\( x = 0.7 \)
Dense-time games

Example

Player 1: “I want to fire \( \bullet \rightarrow \bullet \) at date 1.4.”
Player 2: “I want to fire \( \bullet \rightarrow \bullet \) at date 0.9.”

\[ x = 0.9 \]
Dense-time games

Example

Player 1: “I want to fire at date 1.6.”
Player 2 has no available transition.

\[ x = 0.9 \]
Dense-time games

Example

Player 1: “I want to fire \( \bullet \rightarrow \bullet \) at date 1.6.”
Player 2 has no available transition.
Dense-time games

Example

Player 1: “I want to fire at date 1.9.”
Player 2: “I want to fire at date 1.8.”

\[ x = 1.6 \]
Dense-time games

Example

Player 1: “I want to fire at date 1.9.”
Player 2: “I want to fire at date 1.8.”

\[ x = 1.8 \]
Dense-time games

Example

Player 1: “I want to fire \( \rightarrow \) at date 1.9.”
Player 2: “I want to fire \( \rightarrow \) at date 2.1.”

\[ x = 1.8 \]
Dense-time games

Example

Player 1: “I want to fire at date 1.9.”
Player 2: “I want to fire at date 2.1.”
Dense-time games

Example

Player 1: “I want to fire \( \bullet \rightarrow \circ \) at date 4.5.”
Player 2 has no available transition.
Example

Player 1: “I want to fire $\rightarrow$ at date 4.5.”
Player 2 has no available transition.
Dense-time games

Definition

The **semantics of a timed game automaton** $G = \langle Q, Q_0, T, \ell, \text{Agt}, \text{Own}, C, G, I, R \rangle$ is the (infinite) Durational CGS $C = \langle S, S_0, \delta, e, \text{Agt}, M, Mv, \text{Tab} \rangle$ where

- $\langle S, S_0, \delta, e \rangle$ is the semantics of the underlying timed automaton;
- $M = (T \cup \{\lambda\}) \times \mathbb{R}_{\geq 0}$;
- $Mv: S \times \text{Agt} \rightarrow \mathcal{P}(M) \setminus \{\emptyset\}$ s.t. $(t, d) \in Mv((q, \nu), A_i)$ iff $\nu$ and $\nu + d$ are in $S$, $t$ belongs to $A_i$ and starts from $q$, and $\nu + d \models G(t)$.
- $\text{Tab}$ selects the player with the shortest delay and returns the corresponding transition.
Dense-time games

**Lemma**

If $X$ is a region, then $CPre_A(X)$ is a union of regions.
Dense-time games

Lemma
If $X$ is a region, then $\text{CPre}_A(X)$ is a union of regions.

Theorem
Reachability in timed game automata is EXPTIME-complete.
Dense-time games

Lemma

If $X$ is a region, then $\text{CPre}_A(X)$ is a union of regions.

Theorem

Reachability in timed game automata is EXPTIME-complete.

Proof.

- Membership in EXPTIME: P algorithm for games on the exponential-size region CGS.
Dense-time games

**Lemma**

If $X$ is a region, then $\text{CPre}_A(X)$ is a union of regions.

**Theorem**

*Reachability in timed game automata is EXPTIME-complete.*

*Proof.*

- Membership in EXPTIME: P algorithm for games on the exponential-size region $\text{CGS}$.
- Hardness in EXPTIME: follows from EXPTIME-hardness of discrete-time games.
Does Player 1 have a strategy to reach \( \bullet \)?
Zeno strategies

Example

Does Player 1 have a strategy to reach $\bullet$?

- clock $x$ never reaches 1 if Player 2 plays according to the following strategy:

  $$\sigma(\bigcirc, \nu) = (\lambda, (1 - \nu(x))/2).$$

  Thus Player 1 cannot win this game.
Does Player 1 have a strategy to reach \( \bullet \)?

- **clock \( x \) never reaches 1** if Player 2 plays according to the following strategy:
  
  \[
  \sigma(\bigcirc, v) = (\lambda, (1 - v(x))/2).
  \]

  Thus Player 1 cannot win this game.

- On the other hand, it is not wise to allow a player to “stop” time.
Zeno strategies

Example

Does Player 1 have a strategy to reach ⬤?
Zeno strategies

Example

Does Player 1 have a strategy to reach \( \bigcirc \)?

- a winning strategy necessarily satisfies

\[
\sigma(\bigcirc, x = 0) = (\bigcirc, 0)
\]
\[
\sigma(\bigcirc, x = 0) = (\bigcirc, 0)
\]
Zeno strategies

Example

Does Player 1 have a strategy to reach $\bullet$?

- A winning strategy necessarily satisfies

\[
\sigma(\bullet, x = 0) = (\bullet, 0)
\]

\[
\sigma(\circ, x = 0) = (\circ, 0)
\]

- Player 1 has a winning strategy, but it requires memory if we want it to be non-Zeno.
Ruling out Zeno strategies

Idea
If time converges along an outcome, then (at least) one of the players has played a Zeno strategy.
Ruling out Zeno strategies

Idea

If time converges along an outcome, then (at least) one of the players has played a Zeno strategy.

- at each step, assign a *blame* to the player who applies her strategy (she has chosen the smallest delay);
Ruling out Zeno strategies

Idea

If time converges along an outcome, then (at least) one of the players has played a Zeno strategy.

- at each step, assign a *blame* to the player who applies her strategy (she has chosen the smallest delay);

- we modify the winning condition:
  - if time converges along an outcome, players having finitely many blames are declared winners;
  - if time diverges, the original winning condition is used.
Ruling out Zeno strategies

Example

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

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

\[ \sigma(v) = (2 - v(x)) \]
Ruling out Zeno strategies

Example

\[ \sigma(\cdot, v) = (\cdot, 2 - v(x)) \]
Ruling out Zeno strategies

Example

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

\[ x > 1 \]
Ruling out Zeno strategies

Example

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

No winning strategy for Player 1.
Ruling out Zeno strategies

Example

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

\[ \sigma((v, x)) = (1 - v(x))/2 \]
Ruling out Zeno strategies

Example

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

\[ \sigma(\bullet, v) = (\bullet \rightarrow \circ, (1 - v(x))/2) \]
Theorem

Reachability without Zeno strategies in timed game automata is EXPTIME-complete.
Ruling out Zeno strategies

**Theorem**

*Reachability without Zeno strategies in timed game automata is EXPTIME-complete.*

*Proof.*

- Hardness: reachability is already EXPTIME-hard.
- Membership:
  - modify $\text{CPre}_A$ to handle ticks and blames;
  - $\text{CPre}_A^{Zeno}(X)$ is compatible with regions;
  - model-checking on the region CGS.
Conclusion for dense-time

- **timed automata** and **timed games** are very nice formalisms for reasoning about real-time systems;
- several interesting **decidability** results;
- **adapted data-structures** (DBMs...), **efficient** algorithms;
- **cutting-edge tools** (Uppaal, Kronos, HyTech, CMC...);
Conclusion for dense-time

- **timed automata** and **timed games** are very nice formalisms for reasoning about real-time systems;
- several interesting **decidability** results;
- adapted data-structures (**DBMs...**), efficient algorithms;
- cutting-edge tools (**Uppaal, Kronos, HyTech, CMC...**);

Still **many open problems** in timed automata and timed games...
Outline of the presentation

1. Introduction

2. Discrete-time systems
   - Model Checking Timed Transition Graphs
   - Playing Discrete-Time Games

3. Dense-time systems: timed automata
   - Model Checking Timed Automata
   - Dense-time Games

4. Hot topics...
   - Robustness of timed automata
   - Priced timed automata and priced timed games
   - Other related models

5. Conclusion
Model checking and control

system:

⇒

property:

G(request ⇒ F grant)

control algorithm

yes/no
Implementability of timed automata

Computers are not as precise as timed automata.
Implementability of timed automata

Computers are not as precise as timed automata.

- Timed automata are
  - infinitely punctual: exact synchronization is required when composing several TAs;
Implementability of timed automata

Timed automata are
infinitely punctual: exact synchronization is required when composing several TAs;
infinitely precise: different clocks are assumed to increase at the same rate in both the controller and the system.

Computers are not as precise as timed automata.
Implementability of timed automata

Computers are not as precise as timed automata.

- **Timed automata** are
  - infinitely punctual: exact synchronization is required when composing several TAs;
  - infinitely precise: different clocks are assumed to increase at the same rate in both the controller and the system.
  - infinitely fast: several actions can be performed in any non-zero time interval.
Implementability of timed automata

Computers are not as precise as timed automata.

- **Timed automata** are
  - infinitely punctual: exact synchronization is required when composing several TAs;
  - infinitely precise: different clocks are assumed to increase at the same rate in both the controller and the system.
  - infinitely fast: several actions can be performed in any non-zero time interval.

- **Computers** are:
  - digital: the clock of a computer is digital, and only finitely many operations can be performed in a finite amount of time;
  - imprecise: two different clocks can (more-or-less slightly) drift from one another.
Implementability of timed automata

Examples

- Zeno behaviors

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

- Zeno behaviors
- Non-robust automaton

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

Strict guards: Fischer's mutual exclusion protocol
Examples

- Zeno behaviors
- Non-robust automaton
- Strict guards: Fischer’s mutual exclusion protocol
Different approaches to implementability

- "trajectory tubes": this approach discards behaviours that have too strict constraints.
Different approaches to implementability

- "trajectory tubes": this approach discards behaviours that have too strict constraints.

- "probabilistic semantics": discards unlikely trajectories.
Different approaches to implementability

- **“trajectory tubes”**: this approach discards behaviours that have too strict constraints.

- **“probabilistic semantics”**: discards unlikely trajectories.

- **“platform modeling”**: models the CPU as a timed automaton.

\[
\begin{align*}
x &= \Delta_L, x := 0 \\
time &= time + \Delta_L \\
y &\leq \Delta_P, y := 0 \\
tick! \\
x &\leq \Delta_L \\
y &\leq \Delta_P
\end{align*}
\]
Implementation and robust semantics

Implementation semantics

\[ x = \Delta_L, x := 0 \]
\[ time := time + \Delta_L \]
\[ y \leq \Delta_P, y := 0 \]
\[ tick! \]

\[ \mathcal{A}^{\text{Impl}}_{\Delta_P, \Delta_L} = \{ \text{executions of } \mathcal{A} \text{ under this semantics} \}. \]
Implementation and robust semantics

Implementation semantics

\[
\begin{align*}
&x = \Delta_L, x := 0 \\
&time := time + \Delta_L
\end{align*}
\]

\[
\begin{align*}
&x \leq \Delta_L \\
&y \leq \Delta_P, y := 0 \\
&tick!
\end{align*}
\]

\[
\mathcal{A}^{\text{Impl}}_{\Delta_P, \Delta_L} = \{\text{executions of } \mathcal{A} \text{ under this semantics}\}.
\]

Robust semantics

If \([g] = [a; b]\), then \([g]^{\text{AASAP}}_{\Delta} = [a - \Delta, b + \Delta]\).

\[
\mathcal{A}^{\text{Rob}}_{\Delta} = \{\text{executions of } \mathcal{A} \text{ under this semantics}\}.
\]
Definition

Let $\mathcal{A}$ be a timed automaton and $\phi$ be a path property.

$\mathcal{A}$ is implementable w.r.t. $\phi$ if, for some $\Delta_P > 0$ and $\Delta_L > 0$,

$$[\mathcal{A}]_{\Delta_P, \Delta_L}^{\text{impl}} \subseteq \mathcal{L}(\phi).$$

$\mathcal{A}$ robustly satisfies $\phi$, written $\mathcal{A} \models_{\text{Rob}} \phi$, if for some $\Delta > 0$,

$$[\mathcal{A}]_{\Delta}^{\text{Rob}} \subseteq \mathcal{L}(\phi).$$
Implementation and robust semantics

Definition
Let $\mathcal{A}$ be a timed automaton and $\phi$ be a path property.

$\mathcal{A}$ is implementable w.r.t. $\phi$ if, for some $\Delta_P > 0$ and $\Delta_L > 0$,

$$[\mathcal{A}]^{\text{impl}}_{\Delta_P, \Delta_L} \subseteq \mathcal{L}(\phi).$$

$\mathcal{A}$ robustly satisfies $\phi$, written $\mathcal{A} \models_{\equiv} \phi$, if for some $\Delta > 0$,

$$[\mathcal{A}]^{\text{Rob}}_{\Delta} \subseteq \mathcal{L}(\phi).$$

Theorem
If $\Delta > 3\Delta_L + 4\Delta_P$, then $[\mathcal{A}]^{\text{impl}}_{\Delta_P, \Delta_L} \subseteq [\mathcal{A}]^{\text{Rob}}_{\Delta}$. 

In other terms, implementability can be checked via robustness.
Implementation and robust semantics

Definition

Let $\mathcal{A}$ be a timed automaton and $\phi$ be a path property.

$\mathcal{A}$ is implementable w.r.t. $\phi$ if, for some $\Delta_P > 0$ and $\Delta_L > 0$,

$$[\mathcal{A}]_{\Delta_P, \Delta_L}^{\text{impl}} \subseteq \mathcal{L}(\phi).$$

$\mathcal{A}$ robustly satisfies $\phi$, written $\mathcal{A} \equiv_{\phi}$, if for some $\Delta > 0$,

$$[\mathcal{A}]_{\Delta}^{\text{Rob}} \subseteq \mathcal{L}(\phi).$$

Theorem

If $\Delta > 3\Delta_L + 4\Delta_P$, then $[\mathcal{A}]_{\Delta_P, \Delta_L}^{\text{impl}} \subseteq [\mathcal{A}]_{\Delta}^{\text{Rob}}$.

In other terms, implementability can be checked via robustness.
Model-checking robust safety

Example

\[ x \leq 2, \quad x := 0 \]
\[ y \geq 2, \quad y := 0 \]
Model-checking robust safety

Example
Model-checking robust safety

Example

\[
\begin{align*}
    x &\leq 2, \ x := 0 \\
    y &\geq 2, \ y := 0
\end{align*}
\]
Model-checking robust safety

Example

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

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

\[
x = 0, \quad y = 2
\]

\[
x := 0
\]
Model-checking robust safety

Example

\[
\begin{align*}
y &= 1, & y &= 0 \\
x &\leq 2, & x &= 0 \\
y &\geq 2, & y &= 0 \\
x &= 0, & y &= 2 \\
x &= 0, & y &= 2
\end{align*}
\]
Model-checking robust safety

Example

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

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

\[ x = 0, \ y = 2 \]
\[ x := 0 \]
Model-checking robust safety

Example

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

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

\[ x := 0 \]
Model-checking robust safety

Example

\[
1 - \Delta \leq y \leq 1 + \Delta \\
y := 0
\]

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

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

\[
x \leq \Delta \\
2 - \Delta \leq y \leq 2 + \Delta
\]

\[
x := 0
\]
Model-checking robust safety

Example

\[ x \leq 2 + \Delta, \quad x := 0 \]
\[ y \geq 2 - \Delta, \quad y := 0 \]
\[ x \leq \Delta \]
\[ 2 - \Delta \leq y \leq 2 + \Delta \]
\[ x := 0 \]
Model-checking robust safety

Example

\[1 - \Delta \leq y \leq 1 + \Delta\]
\[y := 0\]
\[x \leq 2 + \Delta, \; x := 0\]
\[y \geq 2 - \Delta, \; y := 0\]
\[2 - \Delta \leq y \leq 2 + \Delta\]
\[x \leq \Delta\]
\[x := 0\]
Model-checking robust safety

Example

\begin{align*}
1 - \Delta \leq y \leq 1 + \Delta \\
y &:= 0
\end{align*}

\begin{align*}
x \leq 2 + \Delta, & \quad x := 0
\end{align*}

\begin{align*}
\Delta \leq y \leq \Delta
\end{align*}

\begin{align*}
y &:= 0
\end{align*}

\begin{align*}
2 - \Delta \leq y \leq 2 + \Delta
\end{align*}

\begin{align*}
x \leq \Delta
\end{align*}
Model-checking robust safety

Example

\[1 - \Delta \leq y \leq 1 + \Delta, \quad y := 0\]

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

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

\[2 - \Delta \leq y \leq 2 + \Delta\]

\[x \leq \Delta\]

\[x := 0\]
Model-checking robust safety

Example

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

\[ 1 - \Delta \leq y \leq 1 + \Delta, \: y := 0 \]

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

\[ 2 - \Delta \leq y \leq 2 + \Delta, \: y := 0 \]

\[ x \leq \Delta, \: x := 0 \]
Model-checking robust safety

Example

\[ 1 - \Delta \leq y \leq 1 + \Delta, \quad y := 0 \]

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

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

\[ 2 - \Delta \leq y \leq 2 + \Delta \]

\[ x \leq \Delta, \quad x := 0 \]
Model-checking robust safety

Example

\[
\begin{align*}
1-\Delta & \leq y \leq 1+\Delta \\
y & := 0 \\
2-\Delta & \leq y \leq 2+\Delta \\
y & := 0 \\
x & \leq 2+\Delta, \ x := 0 \\
x & \leq \Delta \\
x & := 0
\end{align*}
\]
Model-checking robust safety

Example

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

\[ x \leq 2 + \Delta, \quad x := 0 \]
\[ 2 - \Delta \leq y \leq 2 + \Delta \]

\[ 1 - \Delta \leq y \leq 1 + \Delta \]
\[ y := 0 \]

\[ x \leq \Delta \]
\[ x := 0 \]
Model-checking robust safety

Example

\[ 1 - \Delta \leq y \leq 1 + \Delta \]
\[ y := 0 \]
\[ x \leq 2 + \Delta, \quad x := 0 \]
\[ y \geq 2 - \Delta, \quad y := 0 \]
\[ 2 - \Delta \leq y \leq 2 + \Delta \]
\[ x := 0 \]
Model-checking robust safety

Example

\[ 1 - \Delta \leq y \leq 1 + \Delta \]
\[ y := 0 \]

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

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

\[ 2 - \Delta \leq y \leq 2 + \Delta \]

\[ x := 0 \]
Model-checking robust safety

Example

\[ \begin{align*}
1 - \Delta \leq y & \leq 1 + \Delta, \quad y := 0 \\
2 - \Delta \leq x & \leq 2 + \Delta, \quad x := 0
\end{align*} \]
Model-checking robust safety

Example

\[ \begin{align*}
1 - \Delta & \leq y \leq 1 + \Delta \\
y & := 0
\end{align*} \]

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

\[ \begin{align*}
x & \leq \Delta \\
2 - \Delta & \leq y \leq 2 + \Delta
\end{align*} \]

\[ x := 0 \]
Model-checking robust safety

Example

\[
\begin{align*}
x & \leq 2 + \Delta, \quad x := 0 \\
y & \geq 2 - \Delta, \quad y := 0
\end{align*}
\]
Model-checking robust safety

Example

\[\begin{align*}
1 - \Delta \leq y & \leq 1 + \Delta \quad y := 0 \\
2 - \Delta \leq y & \leq 2 + \Delta \\
\end{align*}\]

\[\begin{align*}
x \leq 2 + \Delta, & \quad x := 0 \\
2 - \Delta \leq y & \leq 2 + \Delta \\
\end{align*}\]

\[\begin{align*}
x \leq \Delta, & \quad x := 0 \\
y \geq 2 - \Delta, & \quad y := 0 \\
\end{align*}\]
Model-checking robust safety

**Definition**

\[
\text{Reach}_\Delta(\mathcal{A}) = \{ \text{reachable states in } [\mathcal{A}]^{\text{Rob}}_\Delta \}.
\]

\[
\text{Reach}_{>0}(\mathcal{A}) = \bigcap_{\Delta > 0} \text{Reach}_\Delta(\mathcal{A})
\]
Model-checking robust safety

Definition

\[
\text{Reach}_\Delta(\mathcal{A}) = \{\text{reachable states in } \llbracket \mathcal{A} \rrbracket^\text{Rob}_\Delta\}.
\]

\[
\text{Reach}_{>0}(\mathcal{A}) = \bigcap_{\Delta > 0} \text{Reach}_\Delta(\mathcal{A})
\]

Lemma

For any timed automata \( \mathcal{A} \) and for any region \( B \),

\[
\text{Reach}_{>0}(\mathcal{A}) \cap B = \emptyset \iff \exists \Delta > 0. \text{Reach}_\Delta(\mathcal{A}) \cap B = \emptyset.
\]
Model-checking robust safety

Input: A Timed Automaton $\mathcal{A}$
Output: The set $\text{Reach}_\rightarrow 0(\mathcal{A})$
Model-checking robust safety

Input: A Timed Automaton $\mathcal{A}$
Output: The set $\text{Reach}_{\rightarrow 0}(\mathcal{A})$

1. build the region graph $G$ of $\mathcal{A}$;
Model-checking robust safety

Input: A Timed Automaton $\mathcal{A}$
Output: The set $\text{Reach}_{\rightarrow 0}(\mathcal{A})$

1. build the region graph $G$ of $\mathcal{A}$;
2. compute $\text{SCC}(G) =$ the set of strongly connected components of $G$;
Model-checking robust safety

Input: A Timed Automaton \( \mathcal{A} \)
Output: The set \( \text{Reach}_\rightarrow 0(\mathcal{A}) \)

1. build the region graph \( G \) of \( \mathcal{A} \);
2. compute \( \text{SCC}(G) = \) the set of strongly connected components of \( G \);
3. \( J := [(q_0)] \);

6. return \( (J) \);
Model-checking robust safety

Input: A Timed Automaton $A$
Output: The set $\text{Reach}_{\geq 0}(A)$

1. build the region graph $G$ of $A$;
2. compute $\text{SCC}(G) = \text{the set of strongly connected components of } G$;
3. $J := [(q_0)]$;
4. $J := [\text{Reach}(G, J)]$;

6. return($J$);
Model-checking robust safety

Input: A Timed Automaton $A$
Output: The set $\text{Reach} \rightarrow 0(A)$

1. build the region graph $G$ of $A$;
2. compute $\text{SCC}(G) =$ the set of strongly connected components of $G$;
3. $J := [(q_0)]$;
4. $J := [\text{Reach}(G, J)]$;
5. while $\exists S \in \text{SCC}(G)$. $S \nsubseteq J$ and $S \cap J \neq \emptyset$,
   $J := J \cup S$;
   $J := [\text{Reach}(G, J)]$;
6. return($J$);
Model-checking robust safety

**Input:** A Timed Automaton $\mathcal{A}$  
**Output:** The set $\text{Reach}_\rightarrow 0(\mathcal{A})$

1. build the region graph $G$ of $\mathcal{A}$;  
2. compute $\text{SCC}(G) =$ the set of strongly connected components of $G$;  
3. $J := [(q_0)]$;  
4. $J := \text{Reach}(G, J)$;  
5. while $\exists$ $S \in \text{SCC}(G)$. $S \not\subseteq J$ and $S \cap J \neq \emptyset$,  
   \hspace{1cm} $J := J \cup S$;  
   \hspace{1cm} $J := \text{Reach}(G, J)$;  
6. return($J$);

**Theorem**

*This algorithm is correct.*
Model-checking robust safety

Input: A Timed Automaton $\mathcal{A}$
Output: The set $\text{Reach}_{\rightarrow} 0(\mathcal{A})$

1. build the region graph $G$ of $\mathcal{A}$;
2. compute $\text{SCC}(G) =$ the set of strongly connected components of $G$;
3. $J := \langle (q_0) \rangle$;
4. $J := [\text{Reach}(G, J)]$;
5. while $\exists \ S \in \text{SCC}(G). \ S \not\subseteq J \text{ and } S \cap J \neq \emptyset$,
   
   $J := J \cup S$;
   $J := [\text{Reach}(G, J)]$;
6. return($J$);

Theorem

Robustness w.r.t. safety properties can be checked in PSPACE.
Extended region automaton

Our algorithm suggests to extend the region automaton:

For any location \( \ell \) and any two regions \( r \) and \( r' \), if

1. \([r] \cap [r'] \neq \emptyset\) and
2. \((\ell, r')\) belongs to an SCC of \( \mathcal{R}(A)\),

then we add a transition \((\ell, r) \xrightarrow{\gamma} (\ell, r')\).

We write \( \mathcal{R}^*(A) \) for the resulting automaton.
Extended region automaton

Our algorithm suggests to extend the region automaton:

For any location $\ell$ and any two regions $r$ and $r'$, if

- $[r] \cap [r'] \neq \emptyset$ and
- $(\ell, r')$ belongs to an SCC of $R(A)$,

then we add a transition $(\ell, r) \xrightarrow{\gamma} (\ell, r')$.

We write $R^*(A)$ for the resulting automaton.

**Theorem**

The set $\text{Reach}_{>0}(A)$ is the set of reachable regions in $R^*(A)$.
Extended region automaton

Example
Extended region automaton

Example
Extended region automaton

Example
Extended region automaton

Example
Robust model-checking

**Theorem**

_LTL robust model-checking is PSPACE-complete._

_Proof._

- Hardness: from robust reachability;
- Membership: _via_ Büchi automata.
Robust model-checking

Theorem

$LTL$ robust model-checking is $PSPACE$-complete.

Proof.

- Hardness: from robust reachability;
- Membership: via Büchi automata.

Theorem

$CoFlatMTL$ robust model-checking is $EXPSPACE$-complete.

Proof.

- Hardness: from FlatMTL satisfiability;
- Membership: via channel automata.
Outline of the presentation

1. Introduction

2. Discrete-time systems
   - Model Checking Timed Transition Graphs
   - Playing Discrete-Time Games

3. Dense-time systems: timed automata
   - Model Checking Timed Automata
   - Dense-time Games

4. Hot topics...
   - Robustness of timed automata
   - Priced timed automata and priced timed games
   - Other related models

5. Conclusion
Weighted timed automata

Definition

A *priced timed automaton* is a 9-tuple $\mathcal{A} = \langle Q, Q_0, T, \ell, C, G, I, R, W \rangle$ where

- $\langle Q, Q_0, T, \ell, C, G, I, R \rangle$ is a timed automaton;
- $W : Q \cup T \to \mathbb{Z}$ indicate
  - the cost-per-time-unit for delaying in locations,
  - the cost of firing transitions.
**Weighted timed automata**

**Definition**

A *priced timed automaton* is a 9-tuple $\mathcal{A} = \langle Q, Q_0, T, \ell, C, G, I, R, W \rangle$ where

- $\langle Q, Q_0, T, \ell, C, G, I, R \rangle$ is a timed automaton;
- $W: Q \cup T \rightarrow \mathbb{Z}$ indicate
  - the cost-per-time-unit for delaying in locations,
  - the cost of firing transitions.

**Example**

![Diagram of a priced timed automaton with transitions and guards]
Weighted timed automata

**Definition**

A *priced timed automaton* is a 9-tuple $\mathcal{A} = \langle Q, Q_0, T, \ell, C, G, I, R, W \rangle$ where

- $\langle Q, Q_0, T, \ell, C, G, I, R \rangle$ is a timed automaton;
- $W : Q \cup T \to \mathbb{Z}$ indicate
  - the cost-per-time-unit for delaying in locations,
  - the cost of firing transitions.

**Example**

Diagram showing transitions with conditions:
- $x \leq 2, y := 0$ from the initial state to $y = 0$.
- $y = 0$ transitions to $x \geq 3$.
- $x \geq 3$ from $y = 0$.
- Further transitions with conditions $x \geq 3$ and $x \geq 3$.
Weighted timed automata

Definition

A \textit{priced timed automaton} is a 9-tuple \( \mathcal{A} = \langle Q, Q_0, T, \ell, C, G, I, R, W \rangle \) where

- \( \langle Q, Q_0, T, \ell, C, G, I, R \rangle \) is a timed automaton;
- \( W: Q \cup T \rightarrow \mathbb{Z} \) indicate
  - the cost-per-time-unit for delaying in locations,
  - the cost of firing transitions.

Example

\[
\begin{align*}
\dot{p} &= 5 \\
y &= 0 \\
\dot{p} &= 6 \\
\dot{p} &= 1 \quad (x \geq 3) \\
p &= 1 \\
p &= 2 \\
p &= 7 \quad (x \geq 3)
\end{align*}
\]
Weighted timed automata

Definition

A *priced timed automaton* is a 9-tuple \( \mathcal{A} = \langle Q, Q_0, T, \ell, C, G, I, R, W \rangle \) where

- \( \langle Q, Q_0, T, \ell, C, G, I, R \rangle \) is a timed automaton;
- \( W: Q \cup T \to \mathbb{Z} \) indicate
  - the cost-per-time-unit for delaying in locations,
  - the cost of firing transitions.

Example

[Diagram of a priced timed automaton with transitions between states labeled with conditions and weights]
Weighted timed automata

Example

\[
\begin{align*}
\dot{p} &= 5, \quad x \leq 2, y := 0 \\
p &= 6, \quad x \geq 3 \\
p &= 1, \quad x \geq 3 \\
\dot{p} &= 0, \quad y := 0 \\
p &= 2, \quad p := 1 \\
p &= 7, \quad p := 7
\end{align*}
\]
Weighted timed automata

Example

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

\[ p_+ = 2 \]
\[ x \geq 3 \]
\[ p_+ = 1 \]
\[ p_+ = 7 \]

\[ 0 \leq 2, y := 0 \]
\[ y = 0 \]
\[ x \geq 3 \]

\[ 0 \leq 2, y := 0 \]
\[ y = 0 \]
\[ x \geq 3 \]
Weighted timed automata

Example

\[
\begin{align*}
\dot{p} &= 5, \\
\dot{y} &= 0, \\
\dot{p} &= 6, \\
\dot{p} &= 1, \\
x &\leq 2, \\
y &:= 0, \\
x &\geq 3, \\
p &+= 2, \\
p &+= 1, \\
x &\geq 3, \\
p &+= 7, \\
x &\geq 3,
\end{align*}
\]
Weighted timed automata

Example

\[ \begin{align*}
\dot{p} &= 5 \\
\dot{p} &= 6 \\
\dot{p} &= 1 \\
x &\geq 3 \\
p &+= 1 \\
p &+= 7 \\
y &= 0
\end{align*} \]

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

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

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

\[ 6.5 \quad 2 \quad 0 \quad 14.4 \quad 1 \]

\[ (1.3 \times 5) \quad (2.4 \times 6) \]

0 \quad 1 \quad 2 \quad 3 \quad 4

E F
Weighted timed automata

Example

\[ \dot{p} = 5 \]

\[ y = 0 \]

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

\[ p += 2 \]

\[ \dot{p} = 6 \]

\[ x \geq 3 \]

\[ p += 1 \]

\[ \dot{p} = 1 \]

\[ x \geq 3 \]

\[ p += 7 \]

\[ p \leq 24 \]

\[ \text{(1.3\times5)} \]

\[ 6.5 \]

\[ 2 + 0 \]

\[ \text{(2.4\times6)} \]

\[ 14.4 \]

\[ 1 \]

\[ = 23.9 \]
Weighted timed automata

Example

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

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

\[ \dot{p} = 1 \]
\[ x \geq 3 \]
\[ p += 7 \]

\[ 0 + 6.5 + 2 + 0 + (1.3 \times 5) + (2.4 \times 6) + 1 = 23.9 \]

\[ \textbf{E F}_{p \leq 24} \]
Weighted timed automata

Theorem

If we only allow nonnegative weights,

- optimal infinite behaviours are computable (in PSPACE).
- $WCTL$ is decidable on WTAs with less than 1 clock (in PSPACE).
- $WCTL$ is undecidable on WTAs with 3 or more clocks.
- $WMTL$ is undecidable on WTAs.
Weighted timed automata

Example

\[ -3 + 6 = -6 \]

\[ x := 0 \quad \text{and} \quad x := 1 \]
Weighted timed automata

Example

\[ x := 0 \]

\[ x = 1 \]

lower-bound problem
Weighted timed automata

Example

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

lower-bound problem
Weighted timed automata

Example

\[ x := 0 \rightarrow -3 \rightarrow +6 \rightarrow -6 \quad x := 1 \]

lower-bound problem
Weighted timed automata

Example

\[ x := 0 \quad \Rightarrow \quad -3 \quad \xrightarrow{} \quad +6 \quad \xrightarrow{} \quad -6 \]

\[ x = 1 \]

lower-bound problem
Weighted timed automata

Example

\[ x := 0 \quad \xrightarrow{} \quad +6 \quad \xrightarrow{} \quad -6 \]

\[ x = 1 \]

lower-bound problem
Weighted timed automata

Example

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

interval-bound problem
Weighted timed automata

Example

\[ -3 + 6 = 3 \]

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

interval-bound problem
Weighted timed automata

Example

$\begin{align*}
-3 & \xrightarrow{x:=0} +6 & \xrightarrow{x=1} -6
\end{align*}$

interval-bound problem
Weighted timed automata

Example

\[ -3 + 6 - 6 = 1 \]

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

interval-bound problem
Weighted timed automata

Example

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

Theorem

Deciding the existence of infinite runs in 1-clock WTAs under lower-bound constraint is in P.

Deciding the existence of infinite runs in 0-clock WTAs under interval-bound constraint is in PSPACE (and NP-hard).
Weighted timed games

Example

\[
\dot{p} = 5 \\
y = 0 \\
\dot{p} = 6 \\
x \geq 3 \\
p += 1 \\
\dot{p} = 1 \\
x \geq 3 \\
p += 7 \\
\]

Minimal cost for reaching \( x \leq 2 \): \( y := 0 \):

\[
\inf_{0 \leq t \leq 2} \max (5t + 6(3-t) + 1, 5t + (3-t) + 7) = 17.2
\]

(when \( t = 1.8 \))
Weighted timed games

Example

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

\[ \dot{p} = 1, \quad x \geq 3 \quad \text{if} \quad x \geq 3 \]

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

Minimal cost for reaching 😊:

\[
\inf_{0 \leq t \leq 2} \max(5t + 6(3 - t) + 1, 5t + (3 - t) + 7) = 17.
\]
Weighted timed games

Example

Minimal cost for reaching 😊:

\[ 5t + 6(3 - t) + 1 \]
Weighted timed games

Example

\[\dot{p} = 5, \quad y = 0, \quad \dot{p} = 6, \quad \dot{p} = 1, \quad x \geq 3, \quad p += 1, \quad p += 7\]

Minimal cost for reaching \(\smiley\):

\[5t + 6(3 - t) + 1, \quad 5t + (3 - t) + 7\]
Weighted timed games

Example

Minimal cost for reaching 😊:

\[
\max (5t + 6(3 - t) + 1, 5t + (3 - t) + 7)
\]
Weighted timed games

Example

\[ \dot{p} = 5 \quad y = 0 \quad \dot{p} = 6 \quad p += 1 \quad \dot{p} = 1 \quad p += 7 \]

Minimal cost for reaching \( \smiley \):

\[
\inf_{0 \leq t \leq 2} \max \left( \frac{5}{2} t + 6(3 - t) + 1, \frac{5}{2} t + (3 - t) + 7 \right)
\]
Weighted timed games

Example

Minimal cost for reaching 😊:

$$\inf_{0 \leq t \leq 2} \max (5t + 6(3 - t) + 1, 5t + (3 - t) + 7) = 17.2$$

(when $t = 1.8$)
Weighted timed games

Example

Theorem

If we only allow nonnegative weights,

- computing (almost) optimal strategies is decidable for 1-clock turn-based WTGs is decidable (in 3EXPTIME).
- it is undecidable for 3-clock WTGs.
Weighted timed games

Example

Theorem

Computing strategies for staying alive in 1-clock WTGs under interval-bound constraint is undecidable.
Outline of the presentation

1. Introduction

2. Discrete-time systems
   - Model Checking Timed Transition Graphs
   - Playing Discrete-Time Games

3. Dense-time systems: timed automata
   - Model Checking Timed Automata
   - Dense-time Games

4. Hot topics...
   - Robustness of timed automata
   - Priced timed automata and priced timed games
   - Other related models

5. Conclusion
Other related models

- **hybrid systems**: Goran’s talk this afternoon;

- **probabilistic timed systems**: Dave’s talk on Thursday morning;

- **time(d) Petri nets**...
Outline of the presentation

1 Introduction

2 Discrete-time systems
   - Model Checking Timed Transition Graphs
   - Playing Discrete-Time Games

3 Dense-time systems: timed automata
   - Model Checking Timed Automata
   - Dense-time Games

4 Hot topics...
   - Robustness of timed automata
   - Priced timed automata and priced timed games
   - Other related models

5 Conclusion
Conclusion and perspectives

Timed systems are now rather well understood:

- **discrete-time:**
  - not very faithful modeling;
  - efficient algorithms.

- **dense-time:**
  - very expressive;
  - higher complexity;
  - efficient data-structures and tools.
Conclusion and perspectives

Timed systems are now rather well understood:

- **discrete-time:**
  - not very faithful modeling;
  - efficient algorithms.

- **dense-time:**
  - very expressive;
  - higher complexity;
  - efficient data-structures and tools.

Very active field, many interesting extensions:

- implementability: extension to games...
- priced timed automata and games...
- probabilistic timed systems...
- distributed timed systems...
References – Discrete-time

— Model Checking —


— Games —

References – Dense-time

— Model Checking —


— Games —

L. de Alfaro, M. Faella, T.A. Henzinger, R. Majumdar, M. Stoelinga The Element of Surprise in Timed Games. CONCUR’03, LNCS 2761, p. 142-156.
References – Recent Topics

— Robustness —


— Priced Timed Automata —


