Robust control of timed systems

Patricia Bouyer-Decitre

LSV, CNRS & ENS Cachan, France

Based on joint works with Nicolas Markey, Pierre-Alain Reynier and Ocan Sankur.
Acknowledgment to Nicolas and Ocan for slides.
Support from ERC project EQualIS.
Outline

1. Introduction

2. Robust “black-box” model-checking
   - Parameterized enlarged semantics
   - Parameterized shrunk semantics

3. Robust guided model-checking
   - Excess semantics
   - Conservative semantics

4. Conclusion
Time-dependent systems

- We are interested in timed systems
Time-dependent systems

- We are interested in timed systems
Model-checking and control

system:

property:

[http://www.embedded.com]
Model-checking and control

system:

property:

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

**system:**

![Diagram of a system with tanks and a pump showing states such as Full and Empty, with transitions labeled a! and b? and ¬B.overfull ∧ ¬B.dried_up as properties.]

**property:**

![Diagram of a property with states Full and Empty, with ¬B.overfull ∧ ¬B.dried_up as a label for the algorithm.]

Algorithm:

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

system:

property:

[http://www.embedded.com]

AG(¬B.overfull ∧ ¬B.dried_up)

model-checking algorithm

yes/no
Model-checking and control

system:

property:

control/synthesis algorithm:

AG(¬B.overfull ∧ ¬B.dried_up)
Reasoning about real-time systems

Timed automata [AD94]

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

Example (A computer mouse)

Reasoning about real-time systems

Timed automata [AD94]

A timed automaton is made of

- a finite automaton-based structure
- a set of clocks

Example (A computer mouse)

Reasoning about real-time systems

Timed automata [AD94]

A timed automaton is made of

- a finite automaton-based structure
- a set of clocks
- timing constraints on states and transitions

Example (A computer mouse)

Discrete-time semantics

...because computers are digital!

Discrete-time semantics

...because computers are digital!

Example [Alur91]

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

Discrete-time semantics

...because computers are digital!

Example [Alur91]

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

Discrete-time semantics

...because computers are digital!

Example [Alur91]

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

Discrete-time semantics

...because computers are digital!

Example [Alur91]

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

...because computers are digital!

Example [Alur91]

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

Discrete-time semantics

...because computers are digital!

Example [Alur91]

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

Continuous-time semantics

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

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

Example

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

Technical tool: region abstraction
Continuous-time semantics

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

Example

Theorem [AD94]

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

Technical tool: region abstraction
Continuous-time semantics

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

Example

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

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

Technical tool: region abstraction
Continuous-time semantics

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

Example

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

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

Example

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

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

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

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

Technical tool: region abstraction
Continuous-time semantics

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

Example

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

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

\begin{align*}
  x &= 0 \land \\
  y &\geq 2
\end{align*}

Theorem [AD94]

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

Technical tool: region abstraction
Continuous-time semantics

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

Example

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

Technical tool: region abstraction
Continuous-time semantics

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

Example

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

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

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

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

Technical tool: region abstraction
Continuous-time semantics

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

Example

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

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

Technical tool: region abstraction
Continuous-time semantics

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

Example

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

Theorem \[ AD94 \]
Reachability in timed automata is decidable (as well as many other important properties).

Technical tool: region abstraction
Continuous-time semantics

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

**Example**

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

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

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

Technical tool: region abstraction
Continuous-time semantics

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

Example

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

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

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

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

Technical tool: region abstraction
Introduction

Continuous-time semantics

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

Example

Theorem [AD94]

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

- Technical tool: region abstraction
Are we doing the right job?

The continuous-time semantics is an *idealization* of a physical system.
Are we doing the right job?

The continuous-time semantics is an **idealization** of a physical system.

- It might not be proper for *implementation*:
  - it assumes zero-delay transitions
  - it assumes infinite precision of the clocks
  - it assumes immediate communication between systems
Are we doing the right job?

The continuous-time semantics is an **idealization** of a physical system.

- It might not be proper for **implementation**: it assumes zero-delay transitions, it assumes infinite precision of the clocks, it assumes immediate communication between systems.
- It may generate **timing anomalies**.
Are we doing the right job?

The continuous-time semantics is an **idealization** of a physical system.

- It might not be proper for **implementation**:
  - it assumes zero-delay transitions
  - it assumes infinite precision of the clocks
  - it assumes immediate communication between systems
- It may generate **timing anomalies**
- It does not exclude **non-realizable behaviours**:
  - not only Zeno behaviours
  - many **convergence phenomena** are hidden

\[ \sim \] this requires infinite precision and might not be realizable
Are we doing the right job?

The continuous-time semantics is an **idealization** of a physical system.

- It might not be proper for implementation:
  - it assumes zero-delay transitions
  - it assumes infinite precision of the clocks
  - it assumes immediate communication between systems
- It may generate **timing anomalies**
- It does not exclude **non-realizable behaviours**:
  - not only Zeno behaviours
  - many **convergence phenomena** are hidden

  \( \sim \) this requires infinite precision and might not be realizable

**Important questions**

- Is the real system correct when it is proven correct on the model?
- Does actual work transfer to real-world systems? To what extent?
Example 1: Imprecision on clock values

Frame capture [ACS10]

Example 1: Imprecision on clock values

Frame capture [ACS10]

\[ 2 \text{ t.u.} \]

\[
\begin{array}{cccccc}
\text{frame 0} & \text{frame 1} & \text{frame 2} & \text{frame 3} & \text{frame 4} & \text{frame 5} \\
\end{array}
\]

\[
\begin{array}{cccccc}
\text{encod. 0} & \text{encod. 1} & \text{encod. 2} & \text{encod. 3} & \text{encod. 4} \\
\end{array}
\]

\[ 2 + \epsilon \]

\[ \leadsto \text{A frame will eventually be skipped} \]

Example 2: Strict timing constraints

Mutual exclusion protocol [KLL+97]

When $P_1$ and $P_2$ run in parallel (sharing variable $r$), the state where both of them are in is not reachable. This property is lost when $x_{id} > 2$ is replaced with $x_{id} \geq 2$.

Example 2: Strict timing constraints

Mutual exclusion protocol [KLL+97]

\[ P_{\text{id}} \]

- When \( P_1 \) and \( P_2 \) run in parallel (sharing variable \( r \)), the state where both of them are in is not reachable.

Example 2: Strict timing constraints

Mutual exclusion protocol [KLL+97]

- When $\mathcal{P}_1$ and $\mathcal{P}_2$ run in parallel (sharing variable $r$), the state where both of them are in $\square$ is not reachable.
- This property is lost when $x_{id} > 2$ is replaced with $x_{id} \geq 2$.

Example 3: Scheduling and timing anomaly

- Scheduling analysis with timed automata [AAM06]
- **Goal**: analyze a *work-conserving* scheduling policy on given scenarios (no machine is idle if a task is waiting for execution)

**Example of a scenario**

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>M₁</td>
<td></td>
<td>A</td>
<td></td>
<td>D</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M₂</td>
<td>C</td>
<td></td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

with the dependency constraints: $A \rightarrow B$ and $C \rightarrow D, E$.

1. $A, D, E$ must be scheduled on machine $M₁$
2. $B, C$ must be scheduled on machine $M₂$
3. $C$ starts no sooner than 2 time units

Example 3: Scheduling and timing anomaly

Example of a scenario

\[ M_1 \]

\[ M_2 \]

\[ \sim \] Schedulable in 6 time units

\[ \sim \] Standard analysis does not capture this timing anomaly.
Example 3: Scheduling and timing anomaly

Example of a scenario

\[ \begin{array}{c|c|c|c|c|c|c|c} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline M_1 & A & & D & E \\ M_2 & & C & B & & & & & \\ \end{array} \]

\[ \sim \text{ Schedulable in 6 time units} \]

- Unexpectedly, the duration of A drops to 1.999
Example 3: Scheduling and timing anomaly

Example of a scenario

\[
\begin{array}{ccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
M_1 & A & & D & E & \\
M_2 & C & B & & & \\
\end{array}
\]

\(\sim\) Schedulable in 6 time units

- Unexpectedly, the duration of A drops to 1.999

\[
\begin{array}{ccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
M_1 & A & D & E & & \\
M_2 & C & B & & & \\
\end{array}
\]

is not work-conserving
Example 3: Scheduling and timing anomaly

Example of a scenario

∼ Schedulable in 6 time units
- Unexpectedly, the duration of A drops to 1.999

is not work-conserving

is work-conserving and completes in 7.999 t.u.
Example 3: Scheduling and timing anomaly

Example of a scenario

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>A</td>
<td>D</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_2$</td>
<td>C</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

〜 Schedulable in 6 time units

• Unexpectedly, the duration of $A$ drops to 1.999

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>A</td>
<td>D</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_2$</td>
<td>C</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

is not work-conserving

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>A</td>
<td>D</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_2$</td>
<td>B</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

is work-conserving and completes in 7.999 t.u.

〜 Standard analysis does not capture this **timing anomaly**
Example 4: Zeno behaviours

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

Those are easy to detect and can be handled; [HS11] They are easy to remove by construction.
Example 4: Zeno behaviours

Those are easy to detect and can be handled;

Example 4: Zeno behaviours

Those are easy to detect and can be handled;
They are easy to remove by construction.

Example 5: More complex convergence phenomena

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

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

Value of clock `x` when hitting `y := 0` is converging, even though global time diverges.
Example 5: More complex convergence phenomena

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

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

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

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

Value of clock $x$ when hitting is converging, even though global time diverges.
Example 5: More complex convergence phenomena

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

Value of clock $x$ when hitting $y$ is converging, even though global time diverges.
Example 5: More complex convergence phenomena

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

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

$x = 0 \land y \geq 2$

Value of clock $x$ when hitting $y := 0$ is converging, even though global time diverges.
Example 5: More complex convergence phenomena

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

Value of clock when hitting is converging, even though global time diverges.
Example 5: More complex convergence phenomena

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

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

Value of clock when hitting is converging, even though global time diverges.
Example 5: More complex convergence phenomena

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

Value of clock \(x\) when hitting \(y\) is converging, even though global time diverges.
Example 5: More complex convergence phenomena

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

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

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

\[ x = 0 \land y \geq 2 \]
Example 5: More complex convergence phenomena

\( x = 1 \) \\
\( y := 0 \)

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

\( x = 0 \land \ y \geq 2 \)

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

Value of clock is converging, even though global time diverges.
Example 5: More complex convergence phenomena

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

Value of clock when hitting is converging, even though global time diverges
Example 5: More complex convergence phenomena

$x = 1$
$y := 0$

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

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

$x = 0 \land y \geq 2$

The value of the clock is converging, even though global time diverges.
Example 5: More complex convergence phenomena

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

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

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

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

Value of clock when hitting is converging, even though global time diverges.
Example 5: More complex convergence phenomena

\[ x = 1 \]
\[ y := 0 \]
\[ x \leq 2, x := 0 \]
\[ y \geq 2, y := 0 \]
\[ x = 0 \land y \geq 2 \]
Example 5: More complex convergence phenomena

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

Value of clock is converging, even though global time diverges.
Example 5: More complex convergence phenomena

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

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

\[ y = 0 \]

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

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

\[ \sim \quad \text{Value of clock } x \text{ when hitting } \bigcirc \text{ is converging, even though global time diverges} \]
The goal

Add robustness to the theory of timed automata
The goal

Add robustness to the theory of timed automata

- We need to understand what is the real system behind the mathematical model, and also which implementation we have in mind, if any.
**The goal**

Add robustness to the theory of timed automata

- We need to understand what is the real system behind the mathematical model, and also which implementation we have in mind, if any.

- **Aim**: provide frameworks to build correct systems
The goal

Add robustness to the theory of timed automata

- We need to understand what is the real system behind the mathematical model, and also which implementation we have in mind, if any.
- **Aim:** provide frameworks to build robustly correct systems
The goal

Add robustness to the theory of timed automata

- We need to understand what is the real system behind the mathematical model, and also which implementation we have in mind, if any.

- **Aim:** provide frameworks to build robustly correct systems
  \[\rightsquigarrow\] Robustness calls for specific theories for each application area
The goal

Add robustness to the theory of timed automata

- We need to understand what is the real system behind the mathematical model, and also which implementation we have in mind, if any.

- **Aim:** provide frameworks to build robustly correct systems
  \[\rightsquigarrow\] Robustness calls for specific theories for each application areas

Rest of the talk

We present a couple of frameworks that have been developed recently in this context
Outline

1. Introduction

2. Robust “black-box” model-checking
   Parameterized enlarged semantics
   Parameterized shrunk semantics

3. Robust guided model-checking
   Excess semantics
   Conservative semantics

4. Conclusion
Robust “black-box” model-checking approach

Idea

Capture any real (or approximate) behaviours (e.g. the implementation) in the verification process.
## Robust “black-box” model-checking approach

### Idea

Capture any real (or approximate) behaviours (e.g. the implementation) in the verification process.

Due to imprecisions,

\[
\text{“standard” correctness of } \mathcal{A} \not\Rightarrow \text{ correctness of } \mathcal{A}_{\text{real}}
\]
Robust “black-box” model-checking approach

Idea
Capture any real (or approximate) behaviours (e.g. the implementation) in the verification process

Due to imprecisions,

"standard" correctness of $A \not\Rightarrow$ correctness of $A_{\text{real}}$

$\sim$ We aim at proposing frameworks in which we will ensure the correctness of the real behaviour of the system
## Robust “black-box” model-checking approach

### Idea

Capture any real (or approximate) behaviours (e.g. the implementation) in the verification process.

Due to imprecisions, 

\[
\text{“standard” correctness of } A \not\Rightarrow \text{ correctness of } A_{\text{real}}
\]

\[\leadsto\text{ We aim at proposing frameworks in which we will ensure the correctness of the real behaviour of the system}\]

We describe two such frameworks:

1. either we implement \( A \) and we prove:
   
   \[
   \text{“robust” correctness of } A \Rightarrow \text{ correctness of } A_{\text{real}}
   \]
Robust “black-box” model-checking approach

**Idea**

Capture any real (or approximate) behaviours (e.g. the implementation) in the verification process.

Due to imprecisions,

\[ \text{“standard” correctness of } A \not\Rightarrow \text{ correctness of } A_{\text{real}} \]

\[ \sim \text{ We aim at proposing frameworks in which we will ensure the correctness of the real behaviour of the system} \]

We describe two such frameworks:

1. either we implement \( A \) and we prove:
   \[ \text{“robust” correctness of } A \Rightarrow \text{ correctness of } A_{\text{real}} \]

2. or we build and implement \( B \), and we prove:
   \[ \text{correctness of } A \Rightarrow \text{ “robust” correctness of } B \Rightarrow \text{ correctness of } B_{\text{real}} \]
Outline

1. Introduction

2. Robust “black-box” model-checking
   Parameterized enlarged semantics
   Parameterized shrunk semantics

3. Robust guided model-checking
   Excess semantics
   Conservative semantics

4. Conclusion
Parameterized enlarged semantics for timed automata

A transition can be taken at any time in \([t - \delta; t + \delta]\)
A transition can be taken at any time in $[t - \delta; t + \delta]$.

**Example**

Given a parameter $\delta$,

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

is transformed into

- $1 - \delta \leq x \leq 1 + \delta, y := 0$
- $x \leq 2 + \delta, x := 0$
- $y \geq 2 - \delta, y := 0$
- $x \leq \delta \land y \geq 2 - \delta$
Parameterized enlarged semantics – Discussion

What is the relevance of this semantics?

- This is a worst-case approach
- This captures approximate behaviours of the system
- One can define program semantics such that for every $\epsilon > 0$:

$$A \subseteq \text{program}_\epsilon(A) \subseteq A_f(\epsilon)$$

$\epsilon$: parameters of the semantics


What is the relevance of this semantics?

- This is a worst-case approach
- This captures approximate behaviours of the system
- One can define program semantics such that for every $\epsilon > 0$:

$$A \subseteq \text{program}_\epsilon(A) \subseteq A_{f(\epsilon)}$$

$\epsilon$: parameters of the semantics

Methodology

- Design $A$
- Verify $A_\delta$ (better if $\delta$ is a parameter)
- Implement $A$
Parameterized enlarged semantics – Discussion

What is the relevance of this semantics?

- This is a worst-case approach
- This captures approximate behaviours of the system
- One can define program semantics such that for every $\epsilon > 0$:

\[ \mathcal{A} \subseteq \text{program}_\epsilon(\mathcal{A}) \subseteq \mathcal{A}_{f(\epsilon)} \]

$\epsilon$: parameters of the semantics

Methodology

- Design $\mathcal{A}$
- Verify $\mathcal{A}_\delta$ (better if $\delta$ is a parameter)
- Implement $\mathcal{A}$

$\leadsto$ This is good for designing systems with simple timing constraints (e.g. equalities).
Parameterized enlarged semantics – Algorithmics

\[ \leadsto \text{It adds extra behaviours, however small may be parameter } \delta \]
Parameterized enlarged semantics – Algorithmics

Let it adds extra behaviours, however small may be parameter $\delta$

Example

![Diagram]

$y = 0$

$x = 1$

$y := 0$

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

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

$x = 0 \land y \geq 2$
Parameterized enlarged semantics – Algorithmics

It adds extra behaviours, however small may be parameter $\delta$

Example

The (parameterized) robust model-checking problem

It asks whether there is some $\delta_0 > 0$ such that for every $0 \leq \delta \leq \delta_0$,

$A_\delta |\models \phi$.

When $\delta$ is small, truth of $\phi$ is independent of $\delta$

It can be computed using a simple extension of the region automaton

Theorem

Robust model-checking of reachability, Büchi, LTL, CoflatMTL properties

is decidable. Complexities are those of standard non robust

model-checking problems.
Parameterized enlarged semantics – Algorithmics

\[ \sim \text{ It adds extra behaviours, however small may be parameter } \delta \]

**Example**

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

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

\[\begin{align*}
    x &= 0 \land y \geq 2
\end{align*}\]
Parameterized enlarged semantics – Algorithmics

\[ \text{It adds extra behaviours, however small may be parameter } \delta \]

Example

The (parameterized) robust model-checking problem

It asks whether there is some \( \delta_0 > 0 \) such that for every \( 0 \leq \delta \leq \delta_0 \),

\[ A_\delta | = \varphi. \]

When \( \delta \) is small, truth of \( \varphi \) is independent of \( \delta \)

It can be computed using a simple extension of the region automaton

Theorem

Robust model-checking of reachability, Büchi, LTL, CoflatMTL properties

is decidable. Complexities are those of standard non robust model-checking problems.
Parameterized enlarged semantics – Algorithmics

↔ It adds extra behaviours, however small may be parameter $\delta$

Example

The (parameterized) robust model-checking problem asks whether there is some $\delta_0 > 0$ such that for every $0 \leq \delta \leq \delta_0$, $A_\delta |\phi$.

When $\delta$ is small, truth of $\phi$ is independent of $\delta$.

It can be computed using a simple extension of the region automaton.

Theorem

Robust model-checking of reachability, Büchi, LTL, CoflatMTL properties is decidable. Complexities are those of standard non-robust model-checking problems.
Parameterized enlarged semantics – Algorithmics

〜 It adds extra behaviours, however small may be parameter $\delta$

Example

The (parameterized) robust model-checking problem asks whether there is some $\delta_0 > 0$ such that for every $0 \leq \delta \leq \delta_0$, $A_\delta \models \varphi$.

When $\delta$ is small, truth of $\varphi$ is independent of $\delta$.

It can be computed using a simple extension of the region automaton.

Theorem: Robust model-checking of reachability, Büchi, LTL, CoflatMTL properties is decidable. Complexities are those of standard non-robust model-checking problems.
Parameterized enlarged semantics – Algorithmics

It adds extra behaviours, however small may be parameter \( \delta \)

Example

The (parameterized) robust model-checking problem

It asks whether there is some \( \delta_0 > 0 \) such that for every \( 0 \leq \delta \leq \delta_0 \),

\[ A_{\delta} \models \varphi. \]

When \( \delta \) is small, truth of \( \varphi \) is independent of \( \delta \)

It can be computed using a simple extension of the region automaton

Theorem

Robust model-checking of reachability, Büchi, LTL, CoflatMTL properties is decidable. Complexities are those of standard non robust model-checking problems.
Parameterized enlarged semantics – Algorithmics

〜 It adds extra behaviours, however small may be parameter $\delta$

Example

The (parameterized) robust model-checking problem asks whether there is some $\delta_0 > 0$ such that for every $0 \leq \delta \leq \delta_0$, $A_\delta |= \phi$.

When $\delta$ is small, truth of $\phi$ is independent of $\delta$.

It can be computed using a simple extension of the region automaton.

Theorem

Robust model-checking of reachability, Büchi, LTL, CoflatMTL properties is decidable. Complexities are those of standard non robust model-checking problems.
Parameterized enlarged semantics – Algorithmics

→ It adds extra behaviours, however small may be parameter $\delta$

Example

\[
\begin{align*}
1 - \delta & \leq x \leq 1 + \delta, \quad y := 0 \\
& x \leq 2 + \delta, \quad x := 0 \\
& x \leq \delta \land y \geq 2 - \delta \\
& y \geq 2 - \delta, \quad y := 0
\end{align*}
\]
Parameterized enlarged semantics – Algorithmics

〜 It adds extra behaviours, however small may be parameter $\delta$

Example

The (parameterized) robust model-checking problem asks whether there is some $\delta_0 > 0$ such that for every $0 \leq \delta \leq \delta_0$, $A_\delta |_\phi$.

When $\delta$ is small, truth of $\phi$ is independent of $\delta$.

It can be computed using a simple extension of the region automaton.

Theorem

Robust model-checking of reachability, Büchi, LTL, CoflatMTL properties is decidable. Complexities are those of standard non robust model-checking problems.
Parameterized enlarged semantics – Algorithmics

\[ \sim \text{It adds extra behaviours, however small may be parameter } \delta \]

Example

The (parameterized) robust model-checking problem asks whether there is some \( \delta_0 > 0 \) such that for every \( 0 \leq \delta \leq \delta_0 \), \( A_{\delta} \models \varphi \).

When \( \delta \) is small, truth of \( \varphi \) is independent of \( \delta \). It can be computed using a simple extension of the region automaton.

Theorem

Robust model-checking of reachability, Büchi, LTL, CoflatMTL properties is decidable. Complexities are those of standard non-robust model-checking problems.
Parameterized enlarged semantics – Algorithmics

\[ \text{It adds extra behaviours, however small may be parameter } \delta \]

Example

\begin{align*}
1 - \delta \leq x \leq 1 + \delta, \quad & y := 0 \\
1 - \delta \leq x \leq 1 + \delta, \quad & y := 0 \\
x \leq 2 + \delta, \quad & x := 0 \\
x \leq \delta \land y \geq 2 - \delta \\
y \geq 2 - \delta, \quad & y := 0
\end{align*}
Parameterized enlarged semantics – Algorithmics

\[ \text{It adds extra behaviours, however small may be parameter } \delta \]

\textbf{Example}

\begin{align*}
&1 - \delta \leq x \leq 1 + \delta, \quad y := 0 \\
x \leq 2 + \delta, \quad x := 0 \\
x \leq \delta \land y \geq 2 - \delta \\
y \geq 2 - \delta, \quad y := 0
\end{align*}
Parameterized enlarged semantics – Algorithmics

It adds extra behaviours, however small may be parameter $\delta$

Example

The (parameterized) robust model-checking problem asks whether there is some $\delta_0 > 0$ such that for every $0 \leq \delta \leq \delta_0$, $A_\delta |= \varphi$.

When $\delta$ is small, truth of $\varphi$ is independent of $\delta$. It can be computed using a simple extension of the region automaton.

Robust model-checking of reachability, Büchi, LTL, CoflatMTL properties is decidable. Complexities are those of standard non robust model-checking problems.
Parameterized enlarged semantics – Algorithmics

\( \sim \) It adds extra behaviours, however small may be parameter \( \delta \)

**Example**

The (parameterized) robust model-checking problem asks whether there is some \( \delta_0 > 0 \) such that for every \( 0 \leq \delta \leq \delta_0 \), 

\[ A_\delta | \phi \]

When \( \delta \) is small, truth of \( \phi \) is independent of \( \delta \).
It can be computed using a simple extension of the region automaton.

**Theorem**

Robust model-checking of reachability, Büchi, LTL, CoflatMTL properties is decidable. Complexities are those of standard non-robust model-checking problems.
Parameterized enlarged semantics – Algorithmics

→ It adds extra behaviours, however small may be parameter $\delta$

Example

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

\[
\begin{align*}
1 - \delta &\leq x \leq 1 + \delta \\
y &:= 0
\end{align*}
\]
Parameterized enlarged semantics – Algorithmics

〜 It adds extra behaviours, however small may be parameter $\delta$

**Example**

![Diagram](image-url)

$1 - \delta \leq x \leq 1 + \delta$, $y := 0$

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

$x \leq \delta \land y \geq 2 - \delta$

$y \geq 2 - \delta$, $y := 0$
Parameterized enlarged semantics – Algorithmics

It adds extra behaviours, however small may be parameter $\delta$

Example

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

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

$$
\begin{align*}
x & \leq \delta \land y \geq 2 - \delta
\end{align*}
$$

$$
\begin{align*}
y & \geq 2 - \delta, \\y & := 0
\end{align*}
$$
Parameterized enlarged semantics – Algorithmics

It adds extra behaviours, however small may be parameter $\delta$

Example

```
1 - \delta \leq x \leq 1 + \delta
y := 0

x \leq 2 + \delta, x := 0

x \leq \delta \land y \geq 2 - \delta
y \geq 2 - \delta, y := 0
```
Parameterized enlarged semantics – Algorithmics

It adds extra behaviours, however small may be parameter $\delta$

Example

\[
\begin{align*}
1 - \delta \leq x & \leq 1 + \delta, \quad y := 0 \\
x \leq 2 + \delta, \quad x := 0 \\
x \leq \delta \land y \geq 2 - \delta \\
y \geq 2 - \delta, \quad y := 0
\end{align*}
\]
Parameterized enlarged semantics – Algorithmics

It adds extra behaviours, however small may be parameter $\delta$

The (parameterized) robust model-checking problem

It asks whether there is some $\delta_0 > 0$ such that for every $0 \leq \delta \leq \delta_0$, $A_\delta \models \varphi$. 
Parameterized enlarged semantics – Algorithmics

It adds extra behaviours, however small may be parameter $\delta$

The (parameterized) robust model-checking problem

It asks whether there is some $\delta_0 > 0$ such that for every $0 \leq \delta \leq \delta_0$, $A_\delta \models \varphi$.

- When $\delta$ is small, truth of $\varphi$ is independent of $\delta$
Parameterized enlarged semantics – Algorithmics

\[ \Delta \text{ It adds extra behaviours, however small may be parameter } \delta \]

**The (parameterized) robust model-checking problem**

It asks whether there is some \( \delta_0 > 0 \) such that for every \( 0 \leq \delta \leq \delta_0 \),

\[ A_\delta \models \varphi. \]

- When \( \delta \) is small, truth of \( \varphi \) is independent of \( \delta \)
- It can be computed using a simple extension of the region automaton
Parameterized enlarged semantics – Algorithmics

\[ \text{It adds extra behaviours, however small may be parameter } \delta \]

The (parameterized) robust model-checking problem

It asks whether there is some \( \delta_0 > 0 \) such that for every \( 0 \leq \delta \leq \delta_0 \),
\[ A_\delta \models \varphi. \]

- When \( \delta \) is small, truth of \( \varphi \) is independent of \( \delta \)
- It can be computed using a simple extension of the region automaton

Theorem

Robust model-checking of reachability, Büchi, LTL, CoflatMTL properties is decidable. Complexities are those of standard non robust model-checking problems.

Outline

1. Introduction

2. Robust “black-box” model-checking
   Parameterized enlarged semantics
   Parameterized shrunk semantics

3. Robust guided model-checking
   Excess semantics
   Conservative semantics

4. Conclusion
Parameterized shrunk semantics for timed automata

A constraint $[a, b]$ is shrunk to $[a + k\delta; b - h\delta]$
Parameterized shrunk semantics for timed automata

A constraint \([a, b]\) is shrunk to \([a + k\delta; b - h\delta]\)

Why should we do that?

Abstract model

Real-world model

1 \leq x \leq 2

Parameterized shrunk semantics for timed automata

A constraint $[a, b]$ is shrunk to $[a + k\delta; b - h\delta]$

Why should we do that?

Abstract model

Real-world model

1 $\leq x \leq 2$

$1 - \Delta \leq x \leq 2 + \Delta$

Parameterized shrunk semantics for timed automata

A constraint \([a, b]\) is shrunk to \([a + k\delta; b - h\delta]\)

Why should we do that?

Abstract model

1 \leq x \leq 2

\[1 + \delta' \leq x \leq 2 - \delta\]

Real-world model

1 - \Delta \leq x \leq 2 + \Delta


Parameterized shrunk semantics for timed automata

A constraint \([a, b]\) is shrunk to \([a + k\delta; b - h\delta]\)

Why should we do that?

<table>
<thead>
<tr>
<th>Abstract model</th>
<th>Real-world model</th>
</tr>
</thead>
<tbody>
<tr>
<td>([1, x, 2])</td>
<td>([1 - \Delta, x, 2 + \Delta])</td>
</tr>
<tr>
<td>([1 + \delta', x, 2 - \delta])</td>
<td>([1 + \delta' - \Delta, x, 2 - \delta + \Delta])</td>
</tr>
</tbody>
</table>

Why should we do that?

It is fine as soon as \([1 + \delta' - \Delta; 2 - \delta + \Delta]\) ⊆ \([1, 2]\)
which is the case when \(\delta, \delta' \geq \Delta\).

Parameterized shrunk semantics for timed automata

A constraint \([a, b]\) is shrunk to \([a + k\delta; b - h\delta]\)

Why should we do that?

Abstract model

\[1 \leq x \leq 2\]

\[1 + \delta' \leq x \leq 2 - \delta\]

Real-world model

\[1 - \Delta \leq x \leq 2 + \Delta\]

\[1 + \delta' - \Delta \leq x \leq 2 - \delta + \Delta\]

It is fine as soon as \([1 + \delta' - \Delta; 2 - \delta + \Delta]\) \(\subseteq [1; 2]\), which is the case when \(\delta, \delta' \geq \Delta\).

Parameterized shrunk semantics for timed automata

A constraint \([a, b]\) is shrunk to \([a + k\delta; b - h\delta]\)

Summary of the approach

Shrink the clock constraints in the model, to prevent additional behaviour in the implementation

- If \(B = A_{-k\delta}\), then

\[
B \subseteq \text{program}_\epsilon(B) \subseteq B_{f(\epsilon)} = A_{-k\delta + f(\epsilon)} \subseteq A
\]

Parameterized shrunk semantics – Discussion

What is the relevance of that approach?
Anticipate imprecisions to prevent additional behaviours in the real-world
Parameterized shrunk semantics – Discussion

What is the relevance of that approach?
Anticipate imprecisions to prevent additional behaviours in the real-world

Methodology
- Design and verify $\mathcal{A}$
- Implement $\mathcal{A}_{-k\delta}$ (parameters are $k$ and $\delta$)
Parameterized shrunk semantics – Discussion

What is the relevance of that approach?

Anticipate imprecisions to prevent additional behaviours in the real-world

Methodology

- Design and verify $\mathcal{A}$
- Implement $\mathcal{A}_{-k\delta}$ (parameters are $k$ and $\delta$)

$\sim$ This is good for designing systems with strong/hard timing constraints
Parameterized shrunk semantics – Discussion

What is the relevance of that approach?
Anticipate imprecisions to prevent additional behaviours in the real-world

Methodology
- Design and verify $\mathcal{A}$
- Implement $\mathcal{A}_{-k\delta}$ (parameters are $k$ and $\delta$)

$\sim$ This is good for designing systems with strong/hard timing constraints

⚠️ Problem
Make sure that no important behaviours are lost in $\mathcal{A}_{-k\delta}$!!
Parameterized shrunk semantics – Algorithmics

The (parameterized) shrinkability problem

Find parameters $k$ and $\delta$ such that:

- $A \sqsubseteq_{t.a.} A_{-k\delta}$ (or $F \sqsubseteq_{t.a.} A_{-k\delta}$ for some finite automaton $F$)
  \[\text{[shrinkability w.r.t. untimed simulation]}\]

- $A_{-k\delta}$ is non-blocking whenever $A$ is non-blocking
  \[\text{[shrinkability w.r.t. non-blockingness]}\]
Parameterized shrunk semantics – Algorithmics

The (parameterized) shrinkability problem

Find parameters $k$ and $\delta$ such that:

- $A \subseteq_{t.a.} A_{-k\delta}$ (or $F \subseteq_{t.a.} A_{-k\delta}$ for some finite automaton $F$) [shrinkability w.r.t. untimed simulation]
- $A_{-k\delta}$ is non-blocking whenever $A$ is non-blocking [shrinkability w.r.t. non-blockingness]

Theorem

Parameterized shrinkability can be decided (in exponential time).

- Challenge: take care of the accumulation of perturbations
- Technical tools: parameterized shrunk DBM, max-plus equations
- Tool Shrinktech developed by Ocan Sankur [San13]
  
  http://www.lsv.ens-cachan.fr/Software/shrinktech/

Example

The largest shrunk automaton which is correct w.r.t. untimed simulation and non-blockingness is:

\[
\begin{align*}
\delta & \leq x \land y \leq 1 - \delta \land u \geq \delta \\
y - x & \leq 1 - 4\delta \land u \geq \delta \land u, y := 0 \\
u, x := 0 & \\
y \leq 1 - 2\delta \land 1 + \delta & \leq x \land u \geq \delta \land x - y \geq 3\delta \\
u, y := 0 & \\
u, x, y := 0 \\
\end{align*}
\]
The largest shrunk automaton which is correct w.r.t. untimed simulation and non-blockingness is:

\[
\begin{align*}
3\delta \leq x \land y \leq 1 - \delta \land u \geq \delta \\
y - x \leq 1 - 4\delta \land u \geq \delta
\end{align*}
\]

\[
\begin{align*}
y \leq 1 - 2\delta \land 1 + \delta \leq x \\
u \geq \delta \land x - y \geq 3\delta
\end{align*}
\]

\[
\begin{align*}
y \leq 1 - \delta \land u \geq \delta \\
u, y := 0
\end{align*}
\]
Outline

1. Introduction

2. Robust “black-box” model-checking
   Parameterized enlarged semantics
   Parameterized shrunk semantics

3. Robust guided model-checking
   Excess semantics
   Conservative semantics

4. Conclusion
Robust strategy synthesis

In this talk, a strategy in a timed automaton is a way to resolve (time and action) non-determinism
Robust strategy synthesis

In this talk, a strategy in a timed automaton is a way to resolve (time and action) non-determinism

Example

Strategy: in location \( \bigcirc \) with value \( x \), delay \( \frac{2-x}{2} \)
Robust guided model-checking

Robust strategy synthesis

In this talk, a strategy in a timed automaton is a way to resolve (time and action) non-determinism

Example

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

Strategy: in location \( \bigcirc \) with value \( x \), delay \( \frac{2-x}{2} \)

- This strategy requires infinite precision
Robust strategy synthesis

In this talk, a strategy in a timed automaton is a way to resolve (time and action) non-determinism

Example

Strategy: in location \( \textcircled{1} \) with value \( x \), delay \( \frac{2-x}{2} \)

- This strategy requires infinite precision
- In practice, when \( x \) is close to 2, no additional delay is supported: the run is theoretically infinite, but it is actually blocking
Robust strategy synthesis

In this talk, a strategy in a timed automaton is a way to resolve (time and action) non-determinism

Example

Strategy: in location $\bigcirc$ with value $x$, delay $\frac{2-x}{2}$

- This strategy requires infinite precision
- In practice, when $x$ is close to 2, no additional delay is supported: the run is theoretically infinite, but it is actually blocking
- And that is unavoidable
Robust strategy synthesis

In this talk, a strategy in a timed automaton is a way to resolve (time and action) non-determinism.

Idea

Add robustness to strategies, and adapt the behaviour of the system to previous imprecisions.

〜 develop a theory of robust strategies that tolerate errors/imprecisions and avoid convergence.
Game semantics $G_\delta(\mathcal{A})$ of timed automaton $\mathcal{A}$...

... between **Controller** and **Perturbator**:
- from $(\ell, v)$, **Controller** suggests a delay $d \geq \delta$ and a next edge $e = (\ell \xrightarrow{g,Y} \ell')$ that is available after delay $d$
- **Perturbator** then chooses a perturbation $\epsilon \in [-\delta; +\delta]$
- Next state is $(\ell', (v + d + \epsilon)[Y \leftarrow 0])$

Note: when $\delta = 0$, this is the standard semantics of timed automata.

A $\delta$-robust strategy for **Controller** is then a strategy that satisfies the expected property, whatever plays **Perturbator**.
Game semantics of a timed automaton

Game semantics $G_\delta(A)$ of timed automaton $A$...

... between Controller and Perturbator:

- from $(\ell, v)$, Controller suggests a delay $d \geq \delta$ and a next edge $e = (\ell \xrightarrow{g,Y} \ell')$ that is available after delay $d$
- Perturbator then chooses a perturbation $\epsilon \in [-\delta; +\delta]$
- Next state is $(\ell', (v + d + \epsilon)[Y \leftarrow 0])$

Note: when $\delta = 0$, this is the standard semantics of timed automata.
Game semantics of a timed automaton

Game semantics $G_\delta(A)$ of timed automaton $A$...

... between Controller and Perturbator:

- from $(\ell, v)$, Controller suggests a delay $d \geq \delta$ and a next edge $e = (\ell \xrightarrow{g,Y} \ell')$ that is available after delay $d$
- Perturbator then chooses a perturbation $\epsilon \in [-\delta; +\delta]$
- Next state is $(\ell', (v + d + \epsilon)[Y \leftarrow 0])$

Note: when $\delta = 0$, this is the standard semantics of timed automata.

A $\delta$-robust strategy for Controller is then a strategy that satisfies the expected property, whatever plays Perturbator.
Outline

1. Introduction

2. Robust “black-box” model-checking
   Parameterized enlarged semantics
   Parameterized shrunk semantics

3. Robust guided model-checking
   Excess semantics
   Conservative semantics

4. Conclusion
The excess game semantics

Constraints may not be satisfied after the perturbation: that is, only $v + d$ should satisfy $g$
The excess game semantics

Constraints may not be satisfied after the perturbation: that is, only $v + d$ should satisfy $g$

Example

The excess game semantics

Constraints may not be satisfied after the perturbation: that is, only $v + d$ should satisfy $g$

Example

The excess game semantics

Constraints may not be satisfied after the perturbation: that is, only $v + d$ should satisfy $g$

Example

The excess game semantics

Constraints may not be satisfied after the perturbation: that is, only $v + d$ should satisfy $g$.

Example

$\text{Example}$

$\text{Constraints may not be satisfied after the perturbation: that is, only } v + d \text{ should satisfy } g$

The excess game semantics

**Constraints may not be satisfied after the perturbation: that is, only** \( v + d \) **should satisfy** \( g \)**

**Example**

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

\[ \sim \text{ Allows simple design of constraints, ensures divergence of time, avoids convergence phenomena} \]

---

The (parameterized) synthesis problem

Synthesize $\delta > 0$ and a $\delta$-robust strategy that achieves a given goal.
The excess game semantics – Algorithmics

The (parameterized) synthesis problem

Synthesize $\delta > 0$ and a $\delta$-robust strategy that achieves a given goal.

Two challenges

- **Accumulation of perturbations:**

![Diagram](image-url)
The excess game semantics – Algorithmics

The (parameterized) synthesis problem

Synthesize $\delta > 0$ and a $\delta$-robust strategy that achieves a given goal.

Two challenges

- Accumulation of perturbations:

\[ x \leq 2 \]
\[ y := 0 \]
\[ 1 \leq x - y \]

\[ x = 2 \]

\[ y \]

\[ y \]

\[ x \]

\[ x \]

\[ x \]

\[ y \]

\[ y \]

\[ x \]

\[ x \]
The excess game semantics – Algorithmics

The (parameterized) synthesis problem

Synthesize $\delta > 0$ and a $\delta$-robust strategy that achieves a given goal.

Two challenges

1. Accumulation of perturbations:

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

2. New regions become reachable

   $$x = y = 1$$
   $$y := 0$$

$$2\delta$$
The excess game semantics – Algorithmics

The (parameterized) synthesis problem

Synthesize $\delta > 0$ and a $\delta$-robust strategy that achieves a given goal.

Theorem

The parameterized synthesis problem for reachability properties is decidable and EXPTIME-complete. Furthermore, uniform winning strategies (w.r.t. $\delta$) can be computed.

- Technical tool: a region-based refined game abstraction
- ☑ Extends to two-player games (i.e. to real control problems)
- ☹ Only valid for reachability properties
Outline

1. Introduction

2. Robust “black-box” model-checking
   Parameterized enlarged semantics
   Parameterized shrunk semantics

3. Robust guided model-checking
   Excess semantics
   Conservative semantics

4. Conclusion
The conservative game semantics

Constraints have to be satisfied after the perturbation: that is, $v + d + \epsilon$ should satisfy $g$ for every $\epsilon \in [-\delta; +\delta]$.

The conservative game semantics

Constraints have to be satisfied after the perturbation: that is, 
\( v + d + \epsilon \) should satisfy \( g \) for every \( \epsilon \in [-\delta; +\delta] \)

Example

\[ 1 < x < 2 \]
\[ y := 0 \]
The conservative game semantics

Constraints have to be satisfied after the perturbation: that is, \( v + d + \epsilon \) should satisfy \( g \) for every \( \epsilon \in [-\delta; +\delta] \)

Example

Strongly ensures timing constraints, ensures divergence of time, prevents converging phenomena

The conservative game semantics

Constraints have to be satisfied after the perturbation: that is, $v + d + \epsilon$ should satisfy $g$ for every $\epsilon \in [-\delta; +\delta]$

Example

\[ y := 0 \]

\[ 1 < x < 2 \]

Strongly ensures timing constraints, ensures divergence of time, prevents converging phenomena

The conservative game semantics

Constraints have to be satisfied after the perturbation: that is, 
$v + d + \epsilon$ should satisfy $g$ for every $\epsilon \in [-\delta; +\delta]$

Example

\[ y := 0 \]

Strongly ensures timing constraints, ensures divergence of time, prevents converging phenomena

The (parameterized) synthesis problem

Synthesize $\delta > 0$ and a $\delta$-robust strategy that achieves a given goal.
The (parameterized) synthesis problem

Synthesize $\delta > 0$ and a $\delta$-robust strategy that achieves a given goal.

Theorem

The synthesis problem for Büchi properties is decidable and PSPACE-complete. Furthermore, $\delta$ is at most doubly-exponential, and uniform winning strategies (w.r.t. $\delta$) can be computed.
The problem consists in finding cycles that do not become blocked.
The problem consists in finding cycles that do not become blocked.

- A converging phenomena:
The problem consists in finding cycles that do not become blocked.

- A converging phenomena:

- No convergence:

No such constraining half-spaces.
The problem consists in finding cycles that do not become blocked.

- **A converging phenomena:**

- **No convergence:**

  No such constraining half-spaces.

---

**Tools for solving the synthesis problem**

- Orbit graphs, forgetful cycles \[AB11\]
- Forgetful (that is, strongly connected) orbit graph \(\Leftrightarrow\) no convergence phenomena
  \(\leadsto\) strong relation with thick automata.

---

Example

Robust guided model-checking

A region cycle:

Delay

The corresponding (folded) orbit graph:
Example

A region cycle:

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

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

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

Robust guided model-checking

A region cycle:

The corresponding (folded) orbit graph:
Example

The cycle is not forgetful (that is, not strongly connected), Perturbator can enforce convergence:
Outline

1. Introduction

2. Robust “black-box” model-checking
   Parameterized enlarged semantics
   Parameterized shrunk semantics

3. Robust guided model-checking
   Excess semantics
   Conservative semantics

4. Conclusion
Conclusion

- **Timed automata**: a nice mathematical model for real-time systems with interesting decidability properties and algorithmics solutions.

- Not always easy to transfer correctness proven in this model to real behaviours of the system.

- We have shown several frameworks for robustness that can be used to ensure correctness in the real-world.
Conclusion

- **Timed automata**: a nice mathematical model for real-time systems with interesting decidability properties and algorithmics solutions.
- Not always easy to transfer correctness proven in this model to real behaviours of the system.
- We have shown several frameworks for robustness that can be used to ensure correctness in the real-world.

- Extension of these works to richer models seems unfortunately hard [BMS13]
- A quantitative approach to robustness: Perturbator plays randomly
- Symbolic algorithms?
Conclusion

- **Timed automata:** a nice mathematical model for real-time systems with interesting decidability properties and algorithmics solutions.
- Not always easy to transfer correctness proven in this model to real behaviours of the system.
- We have shown several frameworks for **robustness** that can be used to ensure correctness in the real-world.

- Extension of these works to richer models seems unfortunately hard [BMS13]
- A quantitative approach to robustness: Perturbator plays randomly
- Symbolic algorithms?

- This list of possible approaches is not exhaustive:
  - tube acceptance [GHJ97]
  - turn any automaton into a robust one [BLM+11]
  - sampling approach [KP05, BLM+11]
  - probabilistic approach [BBB+08, BBJM12]
  - ...