Robustness in Timed Systems

Patricia Bouyer-Decitre

LSV, CNRS & ENS Cachan, France

Acknowledgment to Nicolas Markey and Ocan Sankur for slides
Outline

1. Introduction

2. Robust model-checking
   - Parameterized enlarged semantics
   - Automatic generation of an implementation
   - Implementation by shrinking

3. Robust realisability and control
   - Excess semantics
   - Strict 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:
Model-checking and control

system:

property:

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

system:

property:

algorithm

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

system:

property:

model-checking algorithm

yes/no

AG(¬B. overfull ∧ ¬B. dried up)
Model-checking and control

**System:**

![System Diagram]

**Property:**

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

**Control/Synthesis Algorithm**
The model of timed automata
The model of timed automata

**Problem:** x:=0

**Alarm:**
- 15 ≤ x ≤ 16
- y:=0

**Repair:**
- 2 ≤ y ∧ x ≤ 56
- y:=0

**Failsafe:**
- done, 22 ≤ y ≤ 25

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>safe</td>
<td>23</td>
<td>safe</td>
</tr>
<tr>
<td>problem</td>
<td>alarm</td>
<td>alarm</td>
</tr>
<tr>
<td>delayed</td>
<td>failsafe</td>
<td>failsafe</td>
</tr>
<tr>
<td>...</td>
<td>2.3</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Initial Value</th>
<th>Next Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td>23, 0</td>
</tr>
<tr>
<td>y</td>
<td>0</td>
<td>23, 23, 0</td>
</tr>
<tr>
<td>...</td>
<td>15.6</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>17.9</td>
<td>...</td>
</tr>
<tr>
<td>failsafe</td>
<td>repair</td>
<td>repairing</td>
</tr>
<tr>
<td>repair</td>
<td>22.1</td>
<td>repairing</td>
</tr>
<tr>
<td>done</td>
<td>40</td>
<td>safe</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Initial Value</th>
<th>Next Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>17.9</td>
<td>...</td>
</tr>
<tr>
<td>failsafe</td>
<td>17.9</td>
<td>40</td>
</tr>
<tr>
<td>0</td>
<td>22.1</td>
<td>22.1</td>
</tr>
</tbody>
</table>
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!

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

Technical tool: region abstraction

Efficient symbolic technics based on zones, implemented in tools
Continuous-time semantics

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Technical tool: region abstraction

Efficient symbolic technics based on zones, implemented in tools
Continuous-time semantics

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

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

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

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

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

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

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

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

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

Technical tool: region abstraction

Efficient symbolic technics based on zones, implemented in tools
Continuous-time semantics

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

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

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

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

Theorem [AD94]

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

- Technical tool: region abstraction
Continuous-time semantics

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

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

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

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

This is a finite time-abstract bisimulation!
Technical tool: Region abstraction – An example [AD94]
Technical tool: Region abstraction – An example [AD94]
Technical tool: Region abstraction – An example [AD94]
Technical tool: Region abstraction – Another example
Technical tool: Region abstraction – Another example

\[ x = 1 \quad y = 0 \quad x \leq 2, \; x := 0 \quad y \geq 2, \; y := 0 \quad x = 0 \wedge y \geq 2 \]
Technical tool: Zones and DBMs

$DBM = \text{Difference Bound Matrix}$
Technical tool: Zones and DBMs

DBM = Difference Bound Matrix

Zones, or DBMs...

... are used to represent sets of states of timed automata:
Technical tool: Zones and DBMs

DBM = Difference Bound Matrix

Zones, or DBMs...

... are used to represent sets of states of timed automata:

Zone: \((x_1 \geq 3) \land (x_2 \leq 5) \land (x_1 - x_2 \leq 4)\)
Technical tool: Zones and DBMs

DBM = Difference Bound Matrix

Zones, or DBMs...

... are used to represent sets of states of timed automata:

Zone: \((x_1 \geq 3) \land (x_2 \leq 5) \land (x_1 - x_2 \leq 4)\)

DBM:

\[
\begin{pmatrix}
  x_0 & x_1 & x_2 \\
  x_0 & \infty & -3 & \infty \\
  x_1 & \infty & \infty & 4 \\
  x_2 & 5 & \infty & \infty
\end{pmatrix}
\]
Technical tool: Zones and DBMs

DBM = Difference Bound Matrix

Zones, or DBMs...

... are used to represent sets of states of timed automata:

Zone: \((x_1 \geq 3) \land (x_2 \leq 5) \land (x_1 - x_2 \leq 4)\)

Zone:

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

DBM:

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

Diagram:

---

11/69
Technical tool: Zones and DBMs

They can be used to compute sets of states in timed automata

\[ \ell_0 \xrightarrow{x \geq 1 \land y \leq 2, y := 0} \ell_1 \]

\[
\begin{align*}
\text{Pre}_{\text{time}} & \left( \bigcap \text{Unreset}_y \right) \\
= & \text{Pre}_{\text{time}} \left( \bigcap \text{Unreset}_y \right)
\end{align*}
\]
Are we doing the right job?

The continuous-time semantics is adequate for abstract design and high-level analysis.
Example: The Patriot anti-ballistic-missile failure

28 soldiers died.
Example: The Patriot anti-ballistic-missile failure

28 soldiers died.

**Problem: clock drift**

Internal clock incremented by 1/10 every 1/10 s.

\[ x = 0.1, x := 0 \]
\[ \text{clock} += 0.1 \]
Example: The Patriot anti-ballistic-missile failure

28 soldiers died.

Problem: clock drift

Internal clock incremented by 1/10 every 1/10 s.

Clock stored in 24-bit register:

\[
\frac{1}{10} - \left\langle \frac{1}{10} \right\rangle_{24 \text{ bit}} \approx 10^{-7}
\]
Example: The Patriot anti-ballistic-missile failure

28 soldiers died.

Problem: clock drift

Internal clock incremented by 1/10 every 1/10 s.

Clock stored in 24-bit register:

\[
\frac{1}{10} - \left\langle \frac{1}{10} \right\rangle_{24 \text{ bit}} \approx 10^{-7}
\]

After 100 hours, the total drift was 0.34 seconds.
The incoming missile could not be destroyed.
Are we doing the right job?

The continuous-time semantics is adequate for abstract design and high-level analysis.
Are we doing the right job?

The continuous-time semantics is an \textit{idealization} of a physical system. It is adequate for abstract design and high-level analysis.
Are we doing the right job?

The continuous-time semantics is an **idealization** of a physical system. It is adequate for abstract design and high-level analysis.

However it suffers from multiple inaccuracies:
Are we doing the right job?

The continuous-time semantics is an idealization of a physical system. It is adequate for abstract design and high-level analysis.

However it suffers from multiple inaccuracies:

- 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 assumes infinite frequency
Are we doing the right job?

The continuous-time semantics is an idealization of a physical system. It is adequate for abstract design and high-level analysis.

However it suffers from multiple inaccuracies:

- It might not be proper for implementation:
- It may generate timing anomalies
Are we doing the right job?

The continuous-time semantics is an *idealization* of a physical system. It is adequate for abstract design and high-level analysis.

However it suffers from multiple inaccuracies:

- It might not be proper for *implementation*:
- It may generate *timing anomalies*
- It does not exclude *non-realizable behaviours*:
  - not only Zeno behaviours
  - many *convergence phenomena* are hidden

  ∼ 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 is adequate for abstract design and high-level analysis.

However it suffers from multiple inaccuracies:

- It might not be proper for **implementation**:
- It may generate **timing anomalies**
- It does not exclude **non-realizable behaviours**:

**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]

2 t.u.

frame 0  frame 1  frame 2  frame 3  frame 4  frame 5

2 t.u.

encod. 0  encod. 1  encod. 2  encod. 3  encod. 4

2 + \epsilon

A frame will eventually be skipped

Example 1: Imprecision on clock values

Frame capture [ACS10]

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 $P_{id}$ is not reachable. This property is lost when $x_{id} > 2$ is replaced with $x_{id} \geq 2$.

[151x266]Introduction Robust model-checking Robust realisability and control Conclusion

[9x243]Example 2: Strict timing constraints

[33x8][KLL+97] Kristoffersen, Laroussinie, Larsen, Pettersson, Yi. A compositional proof of a real-time mutual exclusion protocol (TAPSOFT’97).
Example 2: Strict timing constraints

Mutual exclusion protocol [KLL+\textsuperscript{97}]

- When $\mathcal{P}_1$ and $\mathcal{P}_2$ 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 $P_1$ and $P_2$ 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 $x_{id} \geq 2$.

[KLL+97] Kristoffersen, Laroussinie, Larsen, Pettersson, Yi. A compositional proof of a real-time mutual exclusion protocol (*TAPSOFT’97*).
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>$M_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_2$</td>
<td></td>
<td></td>
<td></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 $M_1$
2. $B, C$ must be scheduled on machine $M_2$
3. $C$ starts no sooner than 2 time units

Example 3: Scheduling and timing anomaly

Example of a scenario

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

Schedulable in 6 time units
Example 3: Scheduling and timing anomaly

Example of a scenario

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

\(\sim\) Schedulable in 6 time units

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

Example of a scenario

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

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

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

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

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

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

→ Schedulable in 6 time units

- Unexpectedly, the duration of A drops to 1.999

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

Example of a scenario

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

… Schedulable in 6 time units

- Unexpectedly, the duration of A drops to 1.999

\[ \begin{array}{ccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
M_1 & A & D & E \\
M_2 & C & B \\
\end{array} \]

is not work-conserving

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

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

Example of a scenario

ynchronize 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.

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

\[ x < 1 \land y < 1 \]

\[ x := 0 \]

\[ y = 1 \]

\[ x < 1 \land y < 1 \]

\[ x := 0 \]

\[ y = 1 \]

\[ 0 \]

\[ 1 \]

\[ x \]

\[ y \]
Example 4: Zeno behaviours

Those are easy to detect and can be handled;

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

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

\[ x = 1 \quad \text{and} \quad y = 0 \quad \text{leads to} \quad x \leq 2, \quad x = 0 \quad \land \quad y \geq 2 \]

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

\[ x = 1 \overset{y := 0}{\longrightarrow} x \leq 2, \ x := 0 \overset{y \geq 2}{\longrightarrow} x = 0 \land y \geq 2 \overset{y := 0}{\longrightarrow} \]

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

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

Value of clock \( x \) when hitting \( y \geq 2 \) 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 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, & x := 0 \\
y \geq 2, & y := 0
\end{align*}

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

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

Value of clock \( x \) 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

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

  x &\leq 2, \ x := 0 \\
  y &\geq 2, \ y := 0 \\

  x &= 0 \land \\
  y &\geq 2
\end{align*}
\]
Example 5: More complex convergence phenomena
Example 5: More complex convergence phenomena

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

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

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

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

Value of clock \( x \) when hitting \( y \geq 2 \).

Even though global time diverges, convergence is observed.
Example 5: More complex convergence phenomena
Example 5: More complex convergence phenomena
Example 5: More complex convergence phenomena
Example 5: More complex convergence phenomena

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

Value of clock $x^2$ when hitting is converging, even though global time diverges.
Example 5: More complex convergence phenomena

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

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

Value of clock $x$ when hitting $\bigcirc$ 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

- Understand the real system behind the mathematical model
The goal

Add robustness to the theory of timed automata

- Understand the real system behind the mathematical model
- Describe frameworks and provide tools to develop robustly correct systems
The goal

Add robustness to the theory of timed automata

- Understand the real system behind the mathematical model
- Describe frameworks and provide tools to develop robustly correct systems
  - Notion of robustness may depend on application areas
The goal

Add robustness to the theory of timed automata

- Understand the real system behind the mathematical model
- Describe frameworks and provide tools to develop robustly correct systems
  \( \sim \) Notion of robustness may depend on application areas

Rest of the talk

- We present a couple of frameworks that have been developed recently
- We focus on the tolerance to slight timing perturbations, that is, to perturbations on time measurements and jitter
Outline

1 Introduction

2 Robust model-checking
   - Parameterized enlarged semantics
   - Automatic generation of an implementation
   - Implementation by shrinking

3 Robust realisability and control
   - Excess semantics
   - Strict semantics

4 Conclusion
Robust model-checking approach

Idea

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

Idea

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

Due to imprecisions,

“standard” correctness of $\mathcal{A} \not\Rightarrow$ correctness of $\mathcal{A}_{\text{real}}$
Robust 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 $A_{real}$

$\leadsto$ We aim at proposing frameworks in which the correctness of the real system will be ensured once the model is verified
Robust model-checking approach

Idea

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

Due to imprecisions,

“standard” correctness of $\mathcal{A} \not\Rightarrow$ correctness of $\mathcal{A}_{\text{real}}$

$\sim \Rightarrow$ We aim at proposing frameworks in which the correctness of the real system will be ensured once the model is verified

We describe two such frameworks:

1. either we implement $\mathcal{A}$ and we prove:

   “robust” correctness of $\mathcal{A} \Rightarrow$ correctness of $\mathcal{A}_{\text{real}}$
Robust model-checking approach

Idea

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

Due to imprecisions,

“standard” correctness of $A \nRightarrow$ correctness of $A_{\text{real}}$

$\sim$ We aim at proposing frameworks in which the correctness of the real system will be ensured once the model is verified

We describe two such frameworks:

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

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

1. Introduction

2. Robust model-checking
   - Parameterized enlarged semantics
   - Automatic generation of an implementation
   - Implementation by shrinking

3. Robust realisability and control
   - Excess semantics
   - Strict semantics

4. Conclusion
Parameterized enlarged semantics for timed automata

A transition can be taken at any time in $[t - \delta; t + \delta]$
Parameterized enlarged semantics for timed automata

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

Example

Given a parameter $\delta$,

The transition $x=1 \rightarrow x=0, y:=0$ is transformed into

Parameterized model $A_{\delta}$
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

[DDR04] De Wulf, Doyen, Raskin. Almost ASAP semantics: From timed models to timed implementations (HSCC’04).

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 $\mathcal{A}$
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

Methodology

- Design $A$
- Verify $A_\delta$ (better if $\delta$ is a parameter)
- Implement $A$

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

\[ \sim \text{ It adds extra behaviours, however small may be parameter } \delta \]
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 $\delta_0 > 0$ s.t. for every $0 \leq \delta \leq \delta_0$, $A_\delta | = \phi$.

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

Timed automata with parameters: undecidable in general.

Here, an extension of the region automaton will do the job!

Theorem

Robust model-checking of safety, LTL, CoflatMTL properties is decidable.

Complexities are those of standard non robust model-checking problems.

(An exponential bound on $\delta_0$ is proven)
It adds extra behaviours, however small may be parameter $\delta$.

Example

Parameterized enlarged semantics – Algorithmics

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

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

Timed automata with parameters: undecidable in general.

Here, an extension of the region automaton will do the job!

Theorem

Robust model-checking of safety, LTL, CoflatMTL properties is decidable.

Complexities are those of standard non-robust model-checking problems.

(An exponential bound on $\delta_0$ is proven)
Parameterized enlarged semantics – Algorithmics

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

**Example**

![Diagram](image)
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 $\delta_0 > 0$ s.t. for every $0 \leq \delta \leq \delta_0$, $A_\delta |_\phi = \phi$.

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

Timed automata with parameters: undecidable in general.

Here, an extension of the region automaton will do the job!

Theorem

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

(An exponential bound on $\delta_0$ is proven)
Parameterized enlarged semantics – Algorithmics

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

Example

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

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

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

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

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

Example

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

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

Timed automata with parameters: undecidable in general.

Here, an extension of the region automaton will do the job!

Theorem

Robust model-checking of safety, LTL, CoflatMTL properties is decidable.

Complexities are those of standard non robust model-checking problems.

(An exponential bound on $\delta_0$ is proven)
Parameterized enlarged semantics – Algorithmics

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

**Example**

![Example Diagram](image)

The (parameterized) robust model-checking problem asks whether there is \( \delta_0 > 0 \) s.t. for every \( 0 \leq \delta \leq \delta_0 \), \( \mathcal{A}_\delta \models \varphi \).

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

Timed automata with parameters: undecidable in general.

Theorem: Robust model-checking of safety, LTL, CoflatMTL properties is decidable. Complexities are those of standard non robust model-checking problems. (An exponential bound on \( \delta_0 \) is proven)
Parameterized enlarged semantics – Algorithmics

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

Example

The (parameterized) robust model-checking problem asks whether there is \( \delta > 0 \) s.t. for every \( 0 \leq \delta \leq \delta_0 \), \( A_\delta \models \phi \).

When \( \delta \) is small, truth of \( \phi \) is independent of \( \delta \).

Timed automata with parameters: undecidable in general.

Here, an extension of the region automaton will do the job!

Theorem

Robust model-checking of safety, LTL, CoflatMTL properties is decidable. Complexities are those of standard non robust model-checking problems. (An exponential bound on \( \delta_0 \) is proven.)
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 $\delta_0 > 0$ s.t. for every $0 \leq \delta \leq \delta_0$, $A_{\delta} \models \varphi$.

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

Timed automata with parameters: undecidable in general

Here, an extension of the region automaton will do the job!

Theorem
Robust model-checking of safety, LTL, CoflatMTL properties is decidable.

Complexities are those of standard non robust model-checking problems.

(An exponential bound on $\delta_0$ is proven)
Parameterized enlarged semantics – Algorithmics

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

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

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

\[ y = 0 \]

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

\[ x \leq \delta \land y \geq 2 - \delta \]

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

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

\[ y := 0 \]

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

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

Example

The (parameterized) robust model-checking problem

It asks whether there is \( \delta_0 > 0 \) s.t. for every \( 0 \leq \delta \leq \delta_0 \),

\[ A_\delta | \phi \]

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

Timed automata with parameters: undecidable in general

Here, an extension of the region automaton will do the job!

Theorem

Robust model-checking of safety, LTL, CoflatMTL properties is decidable.

Complexities are those of standard non robust model-checking problems.

(An exponential bound on \( \delta_0 \) is proven)
Parameterized enlarged semantics – Algorithmics

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

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

\[ x \leq \delta \land y \geq 2 - \delta \]

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

It adds extra behaviours, however small may be parameter \( \delta \).
Parameterized enlarged semantics – Algorithmics

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

\[ y \]
\[ 3 \]
\[ 2 \]
\[ 1 \]
\[ 0 \]
\[ 1 \]
\[ 2 \]
\[ 3 \]
\[ x \]

It adds extra behaviours, however small may be parameter \( \delta \)
Parameterized enlarged semantics – Algorithmics

\[ y = 0 \quad x \leq 2 + \delta, \; x := 0 \quad x \leq \delta \land y \geq 2 - \delta \]

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

Example

\[ 1 - \delta \leq x \leq 1 + \delta \quad y := 0 \quad y \geq 2 - \delta, \; y := 0 \]
Parameterized enlarged semantics – Algorithmics

〜 It adds extra behaviours, however small may be parameter δ

Example

The (parameterized) robust model-checking problem
It asks whether there is δ₀ > 0 s.t. for every 0 ≤ δ ≤ δ₀,
A_δ |= ϕ.
When δ is small, truth of ϕ is independent of δ.
Timed automata with parameters: undecidable in general.
Here, an extension of the region automaton will do the job!

Theorem
Robust model-checking of safety, LTL, CoflatMTL properties is decidable.
Complexities are those of standard non robust model-checking problems.
(An exponential bound on δ₀ is proven)
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 \\
 x &\leq 2 + \delta, & x := 0 \\
x &\leq \delta \land y \geq 2 - \delta, & y := 0 \\
y &\geq 2 - \delta, & y := 0
\end{align*}
Parameterized enlarged semantics – Algorithmics

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

Example

$1 - \delta \leq x \leq 1 + \delta$

$y := 0$

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

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

$x \leq \delta \land y \geq 2 - \delta$
Parameterized enlarged semantics – Algorithmics

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

$\sim$ Example

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

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

Timed automata with parameters: undecidable in general.

Here, an extension of the region automaton will do the job!

Theorem

Robust model-checking of safety, LTL, CoflatMTL properties is decidable.

Complexities are those of standard non robust model-checking problems.

(An exponential bound on $\delta_0$ is proven)
Parameterized enlarged semantics – Algorithmics

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

The (parameterized) robust model-checking problem

It asks whether there is \( \delta_0 > 0 \) s.t. for every \( 0 \leq \delta \leq \delta_0 \), \( A_\delta \models \varphi \).
Parameterized enlarged semantics – Algorithmics

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

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

It asks whether there is \( \delta_0 > 0 \) s.t. for every \( 0 \leq \delta \leq \delta_0 \), \( A_\delta \models \varphi \).

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

---

[Mil00] Miller. Decidability and Complexity Results for Timed Automata and Semi-linear Hybrid Automata (HSCC’00).
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 \( \delta_0 > 0 \) s.t. for every \( 0 \leq \delta \leq \delta_0 \), \( A_\delta \models \varphi \).

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

\[
\delta_1 \leq \delta_2 \implies \text{Reach}(A_{\delta_1}) \subseteq \text{Reach}(A_{\delta_2})
\]
Parameterized enlarged semantics – Algorithmics

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

The (parameterized) robust model-checking problem

It asks whether there is \( \delta_0 > 0 \) s.t. for every \( 0 \leq \delta \leq \delta_0 \), \( A_\delta \models \varphi \).

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

\[
\begin{align*}
\delta_1 \leq \delta_2 & \implies \text{Reach}(A_{\delta_1}) \subseteq \text{Reach}(A_{\delta_2}) \\
& \implies (A_{\delta_2} \text{ safe } \implies A_{\delta_1} \text{ safe})
\end{align*}
\]
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 $\delta_0 > 0$ s.t. for every $0 \leq \delta \leq \delta_0$, $A_\delta \models \varphi$.

- When $\delta$ is small, truth of $\varphi$ is independent of $\delta$
- Timed automata with parameters: undecidable in general
Parameterized enlarged semantics – Algorithmics

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

The (parameterized) robust model-checking problem

It asks whether there is \( \delta_0 > 0 \) s.t. for every \( 0 \leq \delta \leq \delta_0 \), \( A_\delta \models \varphi \).

- When \( \delta \) is small, truth of \( \varphi \) is independent of \( \delta \)
- Timed automata with parameters: undecidable in general
- Here, an extension of the region automaton will do the job!
Parameterized enlarged semantics – Algorithmics

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

The (parameterized) robust model-checking problem

It asks whether there is \( \delta_0 > 0 \) s.t. for every \( 0 \leq \delta \leq \delta_0 \), \( A_\delta \models \varphi \).

- When \( \delta \) is small, truth of \( \varphi \) is independent of \( \delta \)
- Timed automata with parameters: undecidable in general
- Here, an extension of the region automaton will do the job!

Theorem

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

[BMR06] Bouyer, Markey, Reynier. Robust model-checking of timed automata (LATIN’06).
[BMR08] Bouyer, Markey, Reynier. Robust analysis of timed automata via channel machines (FoSSaCS’08).
Parameterized enlarged semantics – Algorithmics

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

The (parameterized) robust model-checking problem

It asks whether there is \( \delta_0 > 0 \) s.t. for every \( 0 \leq \delta \leq \delta_0 \), \( A_\delta \models \varphi \).

- When \( \delta \) is small, truth of \( \varphi \) is independent of \( \delta \)
- Timed automata with parameters: undecidable in general
- Here, an extension of the region automaton will do the job!

Theorem

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

(An exponential bound on \( \delta_0 \) is proven)

---

[BMR06] Bouyer, Markey, Reynier. Robust model-checking of timed automata (LATIN’06).
[BMR08] Bouyer, Markey, Reynier. Robust analysis of timed automata via channel machines (FoSSaCS’08).
Technical tool: extended region automaton

Extended region automaton

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

- $\overline{r} \cap \overline{r'} \neq \emptyset$ and
- $(\ell, r')$ belongs to an SCC of $\mathcal{R}(A)$,

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

(under slight technical restrictions)
Technical tool: extended region automaton

Extended region automaton

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

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

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

(under slight technical restrictions)
Technical tool: extended region automaton

Extended region automaton

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

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

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

(under slight technical restrictions)
Technical tool: extended region automaton

Extended region automaton

For any location \( \ell \) and any two regions \( r \) and \( r' \), if
- \( \overline{r} \cap \overline{r'} \neq \emptyset \) and
- \( (\ell, r') \) belongs to an SCC of \( \mathcal{R}(A) \),

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

(under slight technical restrictions)
Technical tool: extended region automaton

Extended region automaton

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

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

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

(under slight technical restrictions)
Technical tool: extended region automaton

Extended region automaton

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

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

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

(under slight technical restrictions)
Technical tool: extended region automaton

Extended region automaton

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

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

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

(under slight technical restrictions)
Parameterized enlarged semantics – An example
Parameterized enlarged semantics – An example
Parameterized enlarged semantics – An example
Parameterized enlarged semantics – An example
Outline

1 Introduction

2 Robust model-checking
   - Parameterized enlarged semantics
   - **Automatic generation of an implementation**
   - Implementation by shrinking

3 Robust realisability and control
   - Excess semantics
   - Strict semantics

4 Conclusion
Automatic generation of an implementation

The (approx.) implementation synthesis problem

Given $\mathcal{A}$, build $\mathcal{A}'$ such that:

- $\mathcal{A}'$ ‘identical’ (e.g. bisimilar) to $\mathcal{A}$
- $\mathcal{A}'$ is ‘robust’ (that is, good enough for implementation)

Automatic generation of an implementation

The (approx.) implementation synthesis problem

Given $\mathcal{A}$, build $\mathcal{A}'$ such that:
- $\mathcal{A}'$ ‘identical’ (e.g. bisimilar) to $\mathcal{A}$
- $\mathcal{A}'$ is ‘robust’ (that is, good enough for implementation)

The second condition can be (for instance) read as $\mathcal{A}'$ is approximately the same as $\mathcal{A}'_\delta$, for small enough $\delta$.

Automatic generation of an implementation

The (approx.) implementation synthesis problem

Given $\mathcal{A}$, build $\mathcal{A}'$ such that:
- $\mathcal{A}'$ ‘identical’ (e.g. bisimilar) to $\mathcal{A}$
- $\mathcal{A}'$ is ‘robust’ (that is, good enough for implementation)

The second condition can be (for instance) read as $\mathcal{A}'$ is approximately the same as $\mathcal{A}'_{\delta}$, for small enough $\delta$.

Theorem

All timed automata are approximately implementable!
(for approx. bisimulation)

- Technical tool: region construction

Automatic generation of an implementation

Methodology

- Design and verify $\mathcal{A}$
- Implement $\mathcal{A}'$ (automatically generated)

Automatic generation of an implementation

Methodology

- Design and verify $\mathcal{A}$
- Implement $\mathcal{A}'$ (automatically generated)

- 😊 Separates design and implementation
- 😞 $\mathcal{A}'$ is much bigger than $\mathcal{A}$

Outline

1 Introduction

2 Robust model-checking
   - Parameterized enlarged semantics
   - Automatic generation of an implementation
   - Implementation by shrinking

3 Robust realisability and control
   - Excess semantics
   - Strict semantics

4 Conclusion
Parameterized shrunk semantics for timed automata

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

Parameterized shrunk semantics for timed automata

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

Why should we do that?

Models

Abstract model

Impl. model

1 \leq x \leq 2

Parameterized shrunk semantics for timed automata

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

Why should we do that?

- **Abstract model**
  - Models:
    - \(1 \leq x \leq 2\)
  - Impl. model:
    - \(1 - \Delta \leq x \leq 2 + \Delta\)

Parameterized shrunk semantics for timed automata

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

Why should we do that?

<table>
<thead>
<tr>
<th>Models</th>
<th>Abstract model</th>
<th>Impl. model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real-world</td>
<td>$1 \leq x \leq 2$</td>
<td>$1 - \Delta \leq x \leq 2 + \Delta$</td>
</tr>
<tr>
<td>$1 + \delta \leq x \leq 2 - \delta'$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Why should we do that?

$[1 + \delta - \Delta; 2 - \delta' + \Delta] \subseteq [1; 2]$ which is the case when $\delta, \delta' \geq \Delta$.

Parameterized shrunk semantics for timed automata

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

Why should we do that?

<table>
<thead>
<tr>
<th>Models</th>
<th>Abstract model</th>
<th>Impl. model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real-world</td>
<td>$1 \leq x \leq 2$</td>
<td>$1 - \Delta \leq x \leq 2 + \Delta$</td>
</tr>
<tr>
<td></td>
<td>$1 + \delta \leq x \leq 2 - \delta'$</td>
<td>$1 + \delta - \Delta \leq x \leq 2 - \delta' + \Delta$</td>
</tr>
</tbody>
</table>

Parameterized shrunk semantics for timed automata

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

Why should we do that?

<table>
<thead>
<tr>
<th>Models</th>
<th>Abstract model</th>
<th>Impl. model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real-world</td>
<td>1 \leq x \leq 2</td>
<td>1 - \Delta \leq x \leq 2 + \Delta</td>
</tr>
<tr>
<td></td>
<td>1 + \delta \leq x \leq 2 - \delta'</td>
<td>1 + \delta - \Delta \leq x \leq 2 - \delta' + \Delta</td>
</tr>
</tbody>
</table>

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 \([a + \delta; b - \delta']\)

Summary of the approach

\(\sim\) Shrink the clock constraints in the model, to prevent additional behaviours 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 $\delta$)

∽ This is a good approach 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 $\delta$)

$\rightsquigarrow$ This is a good approach for designing systems with strong/hard timing constraints

Make sure that no important behaviours are lost in $\mathcal{A}_{-k\delta}$!!
Parameterized shrinked 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 $F$)  
  [shrinkability w.r.t. untimed simulation]

- $A_{-k\delta}$ is non-blocking whenever $A$ is non-blocking  
  [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/

The case of non-blockingness

Non-blockingness

Whenever $\sigma$ is taken, either $\sigma'$ or $\sigma''$ is eventually firable.
The case of non-blockingness

**Non-blockingness**

Whenever $\sigma$ is taken, either $\sigma'$ or $\sigma''$ is eventually firable.

**Fix-point characterization**

Let $G_\sigma$ denote the **guards** of the timed automaton. It is non-blocking iff,

$$\forall \sigma, \quad \begin{bmatrix} G_\sigma \end{bmatrix} \subseteq \bigcup \text{Unreset}_{R_\sigma} (\text{Pre}_{\text{time}}(\begin{bmatrix} G_{\sigma'} \end{bmatrix})).$$
Technical tools: shrunk DBMs...

\[ [G_\sigma] \subseteq \text{Unreset}_{R_\sigma}(\text{Pre}_{\text{time}}([G_{\sigma'}])) \]
Technical tools: shrunk DBMs...

\[ [G_\sigma] \subseteq \text{Unreset}_{R_\sigma}(\text{Pre}_{\text{time}}([G_{\sigma'}])) \]

\[ \subseteq \text{Unreset}_y \text{Pre}_{\text{time}} \left( \begin{pmatrix} \text{Unreset}_y \end{pmatrix} \right) \]
Technical tools: shrunk DBMs...

\[
\begin{align*}
\sigma & \quad \sigma' \\
\begin{array}{c}
\mathcal{G}_\sigma \\
\subseteq \\
\text{Unreset}_{R_{\sigma}}(\text{Pre}_{\text{time}}(\mathcal{G}_{\sigma'}))
\end{array}
\end{align*}
\]
Technical tools: shrunk DBMs...

\[ [G_{\sigma}] \subseteq \text{Unreset}_{R_{\sigma}}(\text{Pre}_{\text{time}}([G_{\sigma'}])) \]
Technical tools: shrunk DBMs...

\[
\left[ \langle G_\sigma \rangle - \vec{k}\delta \right] \subseteq \text{Unreset}_{R_\sigma} (\text{Pre}_{\text{time}} (\left[ \langle G_{\sigma'} \rangle - \vec{k}\delta \right]))
\]

Determine \( \vec{k} \)
Technical tools: shrunk DBMs...

\[ \langle G_{\sigma} \rangle_{-k\delta} \subseteq \text{Unreset}_{R_{\sigma}}(\text{Pre}_{\text{time}}(\langle G_{\sigma'} \rangle_{-k\delta})) \]
Technical tools: shrunk DBMs...

\[
\left[ \langle G_{\sigma} \rangle_{-\bar{k}\delta} \right] \subseteq \text{Unreset}_{R_{\sigma}}(\text{Pre}_{\text{time}}(\left[ \langle G_{\sigma'} \rangle_{-\bar{k}\delta} \right]))
\]

\[
\subseteq \text{Unreset}_{y}
\]

\[
\begin{pmatrix}
(k_1 + k_3)\delta \\
(k_2 + k_4)\delta
\end{pmatrix}
\]
Technical tools: shrunk DBMs...

\[
\left[ \left\langle G_\sigma \right\rangle - \vec{k}\delta \right] \subseteq \text{Unreset}_{R_\sigma} (\text{Pre}_{\text{time}}(\left[ \left\langle G_{\sigma'} \right\rangle - \vec{k}\delta \right]))
\]

\[
\subseteq (k_1 + k_3)\delta
\]
Technical tools: shrunk DBMs...and max-plus equations

\[ \left[ \langle G_\sigma \rangle_{-\kappa \delta} \right] \subseteq \text{Unreset}_{R_\sigma} \left( \text{Pre}_{\text{time}} \left( \left[ \langle G_{\sigma'} \rangle_{-\kappa \delta} \right] \right) \right) \]

Then, \( \vec{k} \) should satisfy

\[ k_5 \geq k_1 + k_3 \quad \text{that is,} \quad k_5 = \max(k_5, k_1 + k_3) \]

In this case, the above inclusion equation holds for small enough \( \delta \)'s
\[
\left[ \langle G_\sigma \rangle - \vec{k} \delta \right] \subseteq \text{Unreset}_{R_\sigma}(\text{Pre}_{\text{time}}(\left[ \langle G_\sigma' \rangle - \vec{k} \delta \right])) \\
\iff \\
k_5 = \max(k_5, k_1 + k_3).
\]
\[
\left\lfloor \langle G_\sigma \rangle_{-\vec{k}\delta} \right\rfloor \subseteq \text{Unreset}_{R_\sigma}(\text{Pre}_{\text{time}}(\left\lfloor \langle G_\sigma' \rangle_{-\vec{k}\delta} \right\rfloor))
\]
\[\iff\]
\[k_5 = \max(k_5, k_1 + k_3).\]

**Key Theorem**

Let \( \vec{M} = f(\vec{M}) \) be a **fixpoint equation on zones**, and \( \vec{M} \) a solution. 

\( f \) uses \( \text{Pre}_{\text{time}}(), \cap, \text{Unreset}(). \)

For any \( \vec{k} \in \mathbb{N}^n > 0 \),

\[
\langle \vec{M} \rangle_{-\vec{k}\delta} = f(\langle \vec{M} \rangle_{-\vec{k}\delta}) \quad \forall \text{ small } \delta > 0
\]
\[\iff\]
\[\vec{k} = \varphi(\vec{k}),\]

where \( \varphi \) is a **max-plus expression**.
Key Theorem

Let \( \vec{M} = f(\vec{M}) \) be a fixpoint equation on zones, and \( \vec{M} \) a solution. 
\( f \) uses \( \text{Pre}_{\text{time}}(), \cap, \text{Unreset}(). \)
For any \( \vec{k} \in \mathbb{N}^n > 0 \),
\[
\langle \vec{M} \rangle_{-\vec{k}\delta} = f(\langle \vec{M} \rangle_{-\vec{k}\delta}) \quad \forall \text{ small } \delta > 0
\]
\[
\iff \vec{k} = \varphi(\vec{k}),
\]
where \( \varphi \) is a max-plus expression.

\( \bowtie \) Max-plus algebra: the above fixpoint equations can be solved in polynomial time
Solving max-plus equations

Max-plus graph

\[ k_1 \geq \max(1, 2 + k_2) \land k_3 \geq k_2 \land k_2 \geq \max(4, k_3) \]
Solving max-plus equations

Max-plus graph

\[ k_1 \geq \max(1, 2 + k_2) \land k_3 \geq k_2 \land k_2 \geq \max(4, k_3) \]
Solving max-plus equations

Max-plus graph

\[
\begin{align*}
6 \quad & k_1 \quad \text{max} \quad 6 \\
& + \quad 6 \quad k_2 \quad \text{max} \quad 4 \\
& + \quad 4 \quad k_3 \\
& + \quad 4 \\
\end{align*}
\]
Solving max-plus equations

Max-plus graph

No solution!
Summary of shrinkability

Deciding shrinkability

Apply theorem to following fix-point equations:

- **Non-blockingness:**

  \[ \forall \sigma, \quad [G_\sigma] \subseteq \bigcup_{l_1 \xrightarrow{\sigma} l_2 \xrightarrow{\sigma'} l_3} \text{Unreset}_{R_\sigma} (\text{Pre}_{\text{time}}([G_{\sigma'}])). \]

  (Do technical work to remove the union)

- **Time-abstract simulation** (\(\mathcal{A} \subseteq \text{t.a. } \mathcal{A}_{-\delta_k}\)):

  \[ [M_{l,r}] = \bigcap_{\sigma \in \Sigma} \bigcap_{(l,r) \xrightarrow{\sigma} (l',r')} \text{Pre}_{\text{time}}(\text{Unreset}_{R_\sigma} ([M_{l',r'}]) \cap [G_\sigma]), \]

  where \(M_{l,r}\) is the time-abstract simulator set of the region \((l, r)\).
Example

The largest shrunk automaton which is correct w.r.t. untimed simulation and non-blockingness (for all $\delta \in [0, 1/4]$) is:

- $y \leq 1 \land u \geq 0$
- $y \leq 1 \land 1 \leq x$
- $u \geq 0 \land y \leq 1$

$u, y := 0$

$y \leq 1 \land u \geq 0$

$y \leq 1 \land 1 \leq x$

$u \geq 0 \land y \leq 1$

$u, y := 0$

$u, x, y := 0$

$u, y := 0$

$u, x := 0$

$u, x := 0$

$u, x := 0$

$u, x := 0$
Example

The largest shrunk automaton which is correct w.r.t. untimed simulation and non-blockingness (for all $\delta \in [0, \frac{1}{4}]$) is:

\[ u \geq \delta \wedge y \leq 1 - \delta \wedge u \geq \delta \]
\[ y - x \leq 1 - 4\delta \wedge u \geq \delta \]
\[ u, y := 0 \]
Counter-example

\[ 0 \leq x, y \leq 1, \ x := 0 \]
Counter-example

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

There is no shrunk automaton which is correct w.r.t. non-blockingness. Indeed, the max-plus equations we obtain are:

\[
\begin{align*}
\cdots \\
k_8 &= \max(k_{17}, k_{11} + \max(k_{16}, k_2 + \max(k_7, k_8)))) \\
k_{11} &= \max(1, k_{11})
\end{align*}
\]

which has no solution!

(remember the max-plus graph with no solution)
Counter-example

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

There is no shrunk automaton which is correct w.r.t. non-blockingness. Indeed, the max-plus equations we obtain are:

\[
\begin{aligned}
\cdots \\
k_8 &= \max(k_{17}, k_{11} + \max(k_{16}, k_2 + \max(k_7, k_8))) \\
k_{11} &= \max(1, k_{11})
\end{aligned}
\]

which has no solution!

(remember the max-plus graph with no solution)
Partial conclusion

- We have presented three methods for verifying robust correctness, hence correct implementation
Partial conclusion

- We have presented three methods for verifying robust correctness, hence correct implementation
- Same complexities as standard model-checking!
Partial conclusion

- We have presented three methods for verifying robust correctness, hence correct implementation.

- **Same complexities** as standard model-checking!

- **Technical tools:**
  - Extended region automaton
  - Shrunk DBMs
  - And also characterization of reachability relations in timed automata
    (hidden in this presentation)
Partial conclusion

- We have presented three methods for verifying robust correctness, hence correct implementation

- **Same complexities** as standard model-checking!

- Technical tools:
  - Extended region automaton
  - Shrunk DBMs
  - And also characterization of reachability relations in timed automata
    (hidden in this presentation)

- What is missing:
  - A symbolic approach
  - A tool support
    - Shinktech is a prototype for the shrinking approach
  - Stochastic approach (see later)
Outline

1 Introduction

2 Robust model-checking
   - Parameterized enlarged semantics
   - Automatic generation of an implementation
   - Implementation by shrinking

3 Robust realisability and control
   - Excess semantics
   - Strict semantics

4 Conclusion
Robust realisability

Here, a strategy in a timed automaton is a way to resolve (time and action) non-determinism
Robust realisability

Here, 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 realisability

Here, 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
Robust realisability

Here, 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
Robust realisability

Here, 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 realisability

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

Idea of robust realisability

Synthesize strategies that realise some property, even under perturbations: strategies should adapt to previous imprecisions

\[ \sim \text{ develop a theory of robust strategies that tolerate errors/imprecisions and avoid convergence} \]
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 $(\ell', (v + d + \epsilon)[Y \leftarrow 0])$
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.
Two possible semantics

Consider a transition with guard $x \leq 3 \land y \geq 1$:

**excess semantics**

$y = 1$

$x = 3$

**strict semantics**

$y = 1$

$x = 3$
Outline

1 Introduction

2 Robust model-checking
   - Parameterized enlarged semantics
   - Automatic generation of an implementation
   - Implementation by shrinking

3 Robust realisability and control
   - Excess semantics
   - Strict semantics

4 Conclusion
The excess game semantics

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

The excess game semantics

Constraints may not be satisfied after the perturbation
only $\nu + d$ should satisfy $g$

Example

The excess game semantics

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

Example

The excess game semantics

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

Example

The excess game semantics

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

Example

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

The excess game semantics

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

Example

Example diagram with variables $x = y = 1$, $y := 0$.

The excess game semantics

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

Example

$\Rightarrow$ Allows simple design of constraints, ensures divergence of time, avoids convergence phenomena

The excess game semantics – Algorithmics

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

1. Accumulation of perturbations:

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

$x \leq 2$

$y := 0$

$x = 2$

$1 \leq x - y$

$\delta$
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

   $x = y = 1$
   $y := 0$
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.
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, shrunk DBMs
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, shrunk DBMs
- 🌟 Extends to two-player games (i.e. to real control problems)
- 😞 Only valid for reachability properties
The excess game semantics – Algorithm overview

1. (Forward) Construct an equivalent finite turn-based game $F(A)$ (based on regions)
2. Solve it
3. (Backward) Construct winning states in $G_\delta(A)$, and deduce $\delta_0$
The excess game semantics – Algorithm overview

1. (Forward) Construct an equivalent finite turn-based game $\mathbf{F}(\mathcal{A})$ (based on regions)
2. Solve it
3. (Backward) Construct winning states in $\mathcal{G}_\delta(\mathcal{A})$, and deduce $\delta_0$

Winning states will be described by shrinkings of regions:

$$r - \delta P$$

One can win from a region $r$ in $\mathbf{F}(\mathcal{A})$ $$\iff$$ one can win from a shrinking of $r$ in $\mathcal{G}_\delta(\mathcal{A})$
Construction of the finite turn-based game

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

\[ \ell, r_0 \rightarrow \ell', r_0' \rightarrow \ell', r_1 \]

region automaton:
Construction of the finite turn-based game

Extended region automaton:

Idea: We win from some shrinking of \( r_0 \), if, and only if we win from some shrinkings of \( r_1, r_2, r_3 \).
Construction of the finite turn-based game

Extended region automaton:

Idea: We win from some shrinking of $r_0$, if, and only if we win from some shrinkings of $r_1, r_2, r_3$. 

\[
x = y = 1 \\
y := 0
\]
Assume that we win from \textbf{some} shrinkings of $r_1, r_2, r_3$. 
Assume that we win from some shrinkings of $r_1, r_2, r_3$. 
Assume that we win from some shrinkings of $r_1$, $r_2$, $r_3$.

Can these be combined to a winning strategy from $r_0$?
Assume that we win from some shrinkings of $r_1, r_2, r_3$.

Can these be combined to a winning strategy from $r_0$? No: we don’t have a strategy for valuations around $r_1$. 
Solution: Look for a shrinking of some regions with constraints.

A constrained region is a region with some marked facets. A shrinking of a constrained region does not shrink from marked facets.
Solution: Look for a shrinking of some regions with constraints.

A constrained region is a region with some marked facets. A shrinking of a constrained region does not shrink from marked facets.

We win from $r_0$ iff we win from constrained shrinkings of $r_1, r_2, r_3$. 
Solution: Look for a shrinking of some regions with constraints.

A constrained region is a region with some marked facets.
A shrinking of a constrained region does not shrink from marked facets.

We win from $r_0$ iff we win from constrained shrinkings of $r_1, r_2, r_3$. 
**Solution:** Look for a shrinking of some regions with *constraints*.

A **constrained region** is a region with some marked facets.

A shrinking of a constrained region **does not shrink** from marked facets.

In fact,

\[ r_0 \rightarrow r'_0 \]

\[ r_2 \rightarrow r_1 \rightarrow r_3 \]
**Solution:** Look for a shrinking of some regions with constraints.

A constrained region is a region with some marked facets. A shrinking of a constrained region does not shrink from marked facets.

In fact,

OK, we have a strategy for all the points in the violet area.
Finite game $F(\mathcal{A})$

Shrinking constraint for region $r$ is represented by a boolean matrix $S_r$.

Controller wins in $G_\delta(\mathcal{A})$ for all $\delta \in [0, \delta_0]$ for some $\delta_0 > 0$

$\iff$

Controller wins in $F(\mathcal{A})$. 
Details on the definition of $F(\mathcal{A})$

\[
\ell, r_0, S_{r_0}
\]
Details on the definition of $F(\mathcal{A})$

$S_\varphi$ is defined such that:

Controller wins from some shrinking of $(\varphi, S_\varphi)$ iff
Controller wins from some shrinking of $(r_0, S_{r_0})$. 

$l, r_0, S_{r_0} \rightarrow l, \varphi, S_\varphi$
Details on the definition of $F(\mathcal{A})$

$S_\varphi$ is defined such that:

Controller wins from *some* shrinking of $(\varphi, S_\varphi)$ *iff*
Controller wins from *some* shrinking of $(r_0, S_{r_0})$. 
Details on the definition of $F(\mathcal{A})$

$S_\varphi$ is defined such that:

Controller wins from some shrinking of $(\varphi, S_\varphi)$ iff
Controller wins from some shrinking of $(r_0, S_{r_0})$. 

\[ F(\mathcal{A}) \text{ is defined such that:} \]

\[ \text{Controller wins from some shrinking of } (\varphi, S_\varphi) \iff \text{Controller wins from some shrinking of } (r_0, S_{r_0}). \]
Details on the definition of $F(\mathcal{A})$
Details on the definition of $F(\mathcal{A})$

Controller wins from some shrinking of $(\varphi, S\varphi)$ iff Controller wins from some shrinking of $(r_0, S_{r_0})$.
Details on the definition of $F(A)$

$F(A)$ is defined such that:

Controller wins from some shrinking of $(\varphi, S\varphi)$ iff Controller wins from some shrinking of $(r_0, S_{r_0})$.
Details on the definition of $F(\mathcal{A})$
Details on the definition of $F(\mathcal{A})$

$F(\mathcal{A})$ is defined such that:

Controller wins from some shrinking of $(\varphi, S_{\varphi})$ iff Controller wins from some shrinking of $(r_0, S_{r_0})$.
Constructing a winning strategy from $F(\mathcal{A})$
Constructing a winning strategy from $F(A)$
Constructing a winning strategy from $F(A)$

Each step of the backward propagation gives an upper bound on $\delta$. 

$\ell, r_0, S_{r_0} \rightarrow \ell, \varphi, S_\varphi \rightarrow \ell', r_1, S_{r_1} \rightarrow \ell', r_2, S_{r_2} \rightarrow \ell', r_3, S_{r_3}$

reset
Constructing a winning strategy from $F(A)$
Constructing a winning strategy from $F(A)$

▶ Each step of the backward propagation gives an upper bound on $\delta$. 
Usual semantics in timed automata can encode reachability in linearly bounded Turing machines (PSPACE-complete).

Robust semantics in timed automata can encode reachability in **alternating** linearly bounded Turing machines (EXPTIME-complete).
EXPTIME-hardness

Usual semantics in timed automata can encode reachability in linearly bounded Turing machines (PSPACE-complete).

Robust semantics in timed automata can encode reachability in \textbf{alternating} linearly bounded Turing machines (EXPTIME-complete).

Perturbator has a strategy to choose between any of the two branches.
- Top branch: make the first transition earlier
- Bottom branch: delay the first transition
Outline

1 Introduction

2 Robust model-checking
   - Parameterized enlarged semantics
   - Automatic generation of an implementation
   - Implementation by shrinking

3 Robust realisability and control
   - Excess semantics
   - Strict semantics

4 Conclusion
The strict game semantics

Constraints have to be satisfied after the perturbation:
\[ v + d + \epsilon \text{ should satisfy } g \text{ for every } \epsilon \in [-\delta; +\delta] \]
The strict game semantics

Constraints have to be satisfied after the perturbation:
\[ v + d + \epsilon \text{ should satisfy } g \text{ for every } \epsilon \in [-\delta; +\delta] \]

Example

The strict game semantics

Constraints have to be satisfied after the perturbation:

\[ v + d + \epsilon \text{ should satisfy } g \text{ for every } \epsilon \in [-\delta; +\delta] \]

Example

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

Constraints have to be satisfied after the perturbation:
\[ v + d + \epsilon \text{ should satisfy } g \text{ for every } \epsilon \in [-\delta; +\delta] \]

Example

The strict game semantics

Constraints have to be satisfied after the perturbation:
\[ v + d + \epsilon \text{ should satisfy } g \text{ for every } \epsilon \in [-\delta; +\delta] \]

Example
The strict game semantics

Constraints have to be satisfied after the perturbation:
\[ v + d + \epsilon \text{ should satisfy } g \text{ for every } \epsilon \in [-\delta; +\delta] \]

Example

The strict game semantics

Constraints have to be satisfied after the perturbation:
\[ v + d + \epsilon \text{ should satisfy } g \text{ for every } \epsilon \in [-\delta; +\delta] \]

Example

The strict game semantics

Constraints have to be satisfied after the perturbation:
\[ v + d + \epsilon \text{ should satisfy } g \text{ for every } \epsilon \in [-\delta; +\delta] \]

Example

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

The strict game semantics – Algorithmics

The (parameterized) synthesis problem

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

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:
The problem consists in finding cycles that do not become blocked.

- A converging phenomena:

- No convergence:

Tools for solving the synthesis problem

- Orbit graphs, forgetful cycles [AB11]
- Forgetful orbit graph ⇔ no convergence phenomena
  ∼ strong relation with thick automata.

Technical tool: the (folded) orbit graph

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

\[ y \geq 2, \ y := 0 \]
Technical tool: the (folded) orbit graph

A region cycle:

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

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

The corresponding (folded) orbit graph:
Technical tool: the (folded) orbit graph

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

A region cycle:

The corresponding orbit graph:

\[ \sim \] stores the reachability relation between vertices of the regions
Technical tool: the (folded) orbit graph

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

A region cycle:

The corresponding (folded) orbit graph:
Understanding the folded orbit graph

\[ \nu = \vec{\lambda} \cdot \vec{v} \text{ (convex combination of the vertices)} \]

Understanding the folded orbit graph

\[ \nu = \vec{\lambda} \cdot \vec{v} \] (convex combination of the vertices)

**Reachability relation** [Pur00]

Given a region cycle \( \rho \), and valuation \( \nu = \vec{\lambda} \cdot \vec{v} \),

\[ \vec{\lambda} \cdot \vec{v} \xrightarrow{\rho} \vec{\lambda}' \vec{v} \iff \vec{\lambda}' \text{ is computed by distributing each } \lambda_v \text{ to its successors following a probability distribution} \]

---


64/69
Understanding the folded orbit graph

Reachability relation [Pur00]

Given a region cycle $\rho$, and valuation $\nu = \vec{\lambda} \cdot \vec{v}$,

\[
\vec{\lambda} \cdot \vec{v} \xrightarrow{\rho} \vec{\lambda}' \vec{v} \iff \vec{\lambda}' \text{ is computed by distributing each } \lambda_v \text{ to its successors following a probability distribution}
\]

Understanding the folded orbit graph

\[ \lambda' = p \lambda_1 \]

Reachability relation [Pur00]

Given a region cycle \( \rho \), and valuation \( \nu = \lambda \cdot \vec{v} \),

\[ \lambda' \cdot \vec{v} \stackrel{\rho}{\rightarrow} \lambda' \nu \quad \Leftrightarrow \quad \lambda' \text{ is computed by distributing each } \lambda_v \text{ to its successors following a probability distribution} \]


Understanding the folded orbit graph

Reachability relation \[\text{[Pur00]}\]

Given a region cycle \(\rho\), and valuation \(\nu = \vec{\lambda} \cdot \vec{v}\),

\[
\vec{\lambda} \cdot \vec{v} \xrightarrow{\rho} \vec{\lambda'} \vec{v} \iff \vec{\lambda'} \text{ is computed by distributing}
\]

each \(\lambda_v\) to its successors

following a probability distribution

\[\begin{cases}
\lambda'_1 = p\lambda_1 \\
\lambda'_2 = (1 - p - q)\lambda_1 + \lambda_2
\end{cases}\]

\[\nu = \vec{\lambda} \cdot \vec{v}
\]

\[\begin{array}{c}
\lambda'_1 = p\lambda_1 \\
\lambda'_2 = (1 - p - q)\lambda_1 + \lambda_2
\end{array}\]

\[\text{[Pur00]}\] Puri. Dynamical properties of timed automata \(\text{(Discrete Event Dynamic Systems, 2010)}\).

\[\text{[AB11]}\] Asarin, Basset. Thin and thick timed regular languages \(\text{(FORMATS’11)}\).
Understanding the folded orbit graph

Reachability relation \([\text{Pur00}]\)

Given a region cycle \(\rho\), and valuation \(\nu = \vec{\lambda} \cdot \vec{v}\),

\[
\begin{align*}
\vec{\lambda} \cdot \vec{v} \xrightarrow{\rho} \vec{\lambda}' \vec{v} & \iff \vec{\lambda}' \text{ is computed by distributing} \\
& \text{each } \lambda_v \text{ to its successors} \\
& \text{following a probability distribution}
\end{align*}
\]

\[\begin{align*}
\lambda_1' &= p \lambda_1 \\
\lambda_2' &= (1 - p - q) \lambda_1 + \lambda_2 \\
\lambda_3' &= q \lambda_1 + \lambda_3
\end{align*}\]

\[\nu = \vec{\lambda} \cdot \vec{v}\]

\([\text{Pur00}]\) Puri. Dynamical properties of timed automata (\textit{Discrete Event Dynamic Systems, 2010}).

\([\text{AB11}]\) Asarin, Basset. Thin and thick timed regular languages (\textit{FORMATS’11}).
Understanding the folded orbit graph

\[ \nu = \lambda \cdot \bar{v} \] (convex combination of the vertices)

\[ \begin{align*}
\lambda'_1 &= p \lambda_1 \\
\lambda'_2 &= (1 - p - q) \lambda_1 + \lambda_2 \\
\lambda'_3 &= q \lambda_1 + \lambda_3
\end{align*} \]

Reachability relation [Pur00]

Given a region cycle \( \rho \), and valuation \( \nu = \lambda' \cdot \bar{v}, \)

\[ \lambda' \cdot \bar{v} \xrightarrow{\rho} \lambda' \bar{v} \iff \lambda' \text{ is computed by distributing each } \lambda_v \text{ to its successors following a probability distribution} \]

Understanding the folded orbit graph

\[ \nu = \vec{\lambda} \cdot \vec{v} \] (convex combination of the vertices)

\[ \begin{align*}
\lambda_1' &= p \lambda_1 \\
\lambda_2' &= (1 - p - q) \lambda_1 + \lambda_2 \\
\lambda_3' &= q \lambda_1 + \lambda_3
\end{align*} \]

\( \lambda_1 + \lambda_2 \) is non-increasing and \( \lambda_3 \) is non-decreasing

Understanding the folded orbit graph

The reachability relation along one cycle is complete iff its folded orbit graph is complete.

Understanding the folded orbit graph

Generalization

- The reachability relation along one cycle is complete iff its folded orbit graph is complete. [Pur00]
- If the folded orbit graph is connected but not strongly connected, then there is some convergence phenomenon in the direction of the hyperplane $\sum_{v \in I} \lambda_v$. [AB11]

Understanding the folded orbit graph

Classification of cycles

A cycle is **aperiodic** if all its iterations are strongly connected.

---

[AB11] Asarin, Basset. Thin and thick timed regular languages (*FORMATS’11*).
Understanding the folded orbit graph

Classification of cycles

A cycle is \textbf{aperiodic} if all its iterations are strongly connected. Then:

- aperiodic cycle: no convergence phenomenon
  (some iterate is complete)

- non-aperiodic cycle: convergence phenomenon
  (convergence phenomenon from the non strongly connected iterate)

[AB11] Asarin, Basset. Thin and thick timed regular languages (\textit{FORMATS’11}).
Back to robustness

**Characterization**

There exists $\delta > 0$ such that **Controller** has a $\delta$-robust strategy ensuring a Büchi condition in $G_\delta(\mathcal{A})$ if, and only if there is a reachable aperiodic cycle in $\mathcal{A}$ which satisfies the Büchi condition.
Characterization

There exists $\delta > 0$ such that **Controller** has a $\delta$-robust strategy ensuring a Büchi condition in $G_\delta(A)$ if, and only if there is a reachable aperiodic cycle in $A$ which satisfies the Büchi condition.

- Non aperiodic cycle: **Perturbator** can enforce rapid decrease of $\sum_{v \in I} \lambda_v$.

\[ \sum_{v \in I} \lambda_v \geq \epsilon \]
Back to robustness

Characterization

There exists $\delta > 0$ such that Controller has a $\delta$-robust strategy ensuring a Büchi condition in $G_\delta(\mathcal{A})$ if, and only if there is a reachable aperiodic cycle in $\mathcal{A}$ which satisfies the Büchi condition.

- Non aperiodic cycle: Perturbator can enforce rapid decrease of $\sum_{v \in I} \lambda_v$, hence he can enforce convergence, and Controller gets stuck.
Back to robustness

**Characterization**

There exists $\delta > 0$ such that Controller has a $\delta$-robust strategy ensuring a Büchi condition in $G_\delta(A)$ if, and only if there is a reachable aperiodic cycle in $A$ which satisfies the Büchi condition.

- Non aperiodic cycle: Perturbator can enforce rapid decrease of $\sum_{v \in I} \lambda_v$.
- Aperiodic cycle $\pi$: Controller can target the middle of the regions and stay far from the borders.
Back to robustness

Characterization

There exists $\delta > 0$ such that Controller has a $\delta$-robust strategy ensuring a Büchi condition in $G_\delta(A)$ if, and only if there is a reachable aperiodic cycle in $A$ which satisfies the Büchi condition.

- Non aperiodic cycle: Perturbator can enforce rapid decrease of $\sum_{v \in I} \lambda_v$.
- Aperiodic cycle $\pi$: Controller can target the middle of the regions and stay far from the borders.
Back to robustness

Characterization

There exists $\delta > 0$ such that Controller has a $\delta$-robust strategy ensuring a Büchi condition in $G_\delta(\mathcal{A})$ if, and only if there is a reachable aperiodic cycle in $\mathcal{A}$ which satisfies the Büchi condition.

- Non aperiodic cycle: Perturbator can enforce rapid decrease of $\sum_{\nu \in I} \lambda_{\nu}$.
- Aperiodic cycle $\pi$: Controller can target the middle of the regions and stay far from the borders.

Remember shrunk DBMs: preimage of $s$ by $\pi$ under $\delta$-perturbations is $r - \delta Q$ ($Q$ fixed) for small $\delta$’s.

$\leadsto$ from $r - \delta Q$, Controller has a strategy to ensure $s$.
Back to robustness

Characterization

There exists $\delta > 0$ such that Controller has a $\delta$-robust strategy ensuring a Büchi condition in $G_\delta(A)$ if, and only if there is a reachable aperiodic cycle in $A$ which satisfies the Büchi condition.

- Non aperiodic cycle: Perturbator can enforce rapid decrease of $\sum_{v \in I} \lambda_v$.
- Aperiodic cycle $\pi$: Controller can target the middle of the regions and stay far from the borders.

$\Rightarrow$ Remember shrunk DBMs: preimage of $s$ by $\pi$ under $\delta$-perturbations is $r - \delta Q$ ($Q$ fixed) for small $\delta$’s

$\Rightarrow$ from $r - \delta Q$, Controller has a strategy to ensure $s$

$\Rightarrow$ Property of $s$: $s \subseteq r - \delta Q$ for small $\delta$’s

$\Rightarrow$ we can repeat the above strategy
Characterization

There exists $\delta > 0$ such that Controller has a $\delta$-robust strategy ensuring a Büchi condition in $G_\delta(A)$ if, and only if there is a reachable aperiodic cycle in $A$ which satisfies the Büchi condition.

- Non aperiodic cycle: Perturbator can enforce rapid decrease of $\sum_{v \in I} \lambda_v$.
- Aperiodic cycle $\pi$: Controller can target the middle of the regions and stay far from the borders.

- Remember shrunk DBMs: preimage of $s$ by $\pi$ under $\delta$-perturbations is $r - \delta Q$ ($Q$ fixed) for small $\delta$’s.
- $\sim$ from $r - \delta Q$, Controller has a strategy to ensure $s$.
- Property of $s$: $s \subseteq r - \delta Q$ for small $\delta$’s.
- $\sim$ we can repeat the above strategy.
  $\Rightarrow$ Robust strategy: enforce $s$ at each cycle.
Going further [ORS14]

Extension to two-player games

- New rules: **Controller** chooses a delay and an action, and **Perturbator** perturbs the delay and resolves the non-determinism, if any.
- Robustness under strict semantics can be solved in this case as well (EXPTIME)

Going further [ORS14]

Extension to two-player games
- New rules: Controller chooses a delay and an action, and Perturbator perturbs the delay and resolves the non-determinism, if any
- Robustness under strict semantics can be solved in this case as well (EXPTIME)

Beyond worst-case robustness
- Assume perturbations are randomized!
  (uniform distributions over $[d - \delta; d + \delta]$)

Going further [ORS14]

Extension to two-player games
- New rules: Controller chooses a delay and an action, and Perturbator perturbs the delay and resolves the non-determinism, if any.
- Robustness under strict semantics can be solved in this case as well (EXPTIME).

Beyond worst-case robustness
- Assume perturbations are randomized! (uniform distributions over \([d - \delta; d + \delta]\))
- Existence of an almost-sure winning strategy for Controller can be decided in EXPTIME. Furthermore there is a dichotomy:
  - either Controller wins almost-surely
  - or Perturbator wins almost-surely

We have presented a possible approach to the robust realizability and control problems.

- There are two natural semantics (excess or strict).
- Interesting relation between non-convergent cycles and robust cycles.
- Interesting complexities as well!
We have presented a possible approach to the robust realizability and control problems.

- There are two natural semantics (excess or strict)
- Interesting relation between non-convergent cycles and robust cycles
- Interesting complexities as well!

Technical tools:

- Regions
- Shrunken DBMs
- Orbit graphs
Partial conclusion

- We have presented a possible approach to the robust realizability and control problems
  - There are two natural semantics (excess or strict)
  - Interesting relation between non-convergent cycles and robust cycles
  - Interesting complexities as well!

- Technical tools:
  - Regions
  - Shrunk DBMs
  - Orbit graphs

- What is missing:
  - A symbolic approach
  - A tool support
  - Stochastic approach at the beginning only
Outline

1. Introduction

2. Robust model-checking
   - Parameterized enlarged semantics
   - Automatic generation of an implementation
   - Implementation by shrinking

3. Robust realisability and control
   - Excess semantics
   - Strict semantics

4. Conclusion
Conclusion

- **Timed automata**: a nice mathematical model for real-time systems with interesting decidability properties and algorithmics solutions.
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.
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.
- We have seen several tools that are useful in this context (*eg*, shrunk DBMs, orbit graphs)
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.
- We have seen several tools that are useful in this context (*eg*, shrunk DBMs, orbit graphs)
- Extension of these works to richer models seems unfortunately hard [*BMS13*]
- Probabilistic perturbations [*ORS14*]: compute/give lower bound average-time to failure?
- 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.
- We have seen several tools that are useful in this context (e.g., shrunk DBMs, orbit graphs)

- Extension of these works to richer models seems unfortunately hard [BMS13]
- Probabilistic perturbations [ORS14]: compute/give lower bound average-time to failure?
- Symbolic algorithms?

- This list of possible approaches is not exhaustive:
  - tube acceptance [GHJ97]
  - sampling approach [KP05, BLM⁺11]
  - probabilistic approach [BBB⁺08, BBJM12]
  - …