Soutenance d’habilitation

Verification of Embedded Systems
— Algorithms and Complexity —

Nicolas Markey

LSV, CNRS & ENS Cachan, France

April 8, 2011
Verification of embedded systems

- Computers are everywhere
Verification of embedded systems

- Computers are everywhere

- Bugs are everywhere...
Verification of embedded systems

- Computers are everywhere

- Bugs are everywhere...

- Verification should be everywhere!
Formal verification

- provides (partial) proof of correctness;
- model-based methods;
- (more-or-less) exhaustive methods;
- (more-or-less) automated techniques.
**Formal verification**

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

**Different techniques**

- (model-based) testing
**Formal verification**

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

**Different techniques**

- (model-based) testing
- theorem proving
Formal verification

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

Different techniques

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

\[ \text{system: } G(\text{request} \Rightarrow F\text{grant}) \]

\[ \text{model-checking algorithm: } \text{yes/no} \]
Model checking

system:

⇒

property:

G(request ⇒ F grant)

model-checking algorithm

yes/no
Embedded system

An embedded system is a computer system designed to perform one or a few dedicated functions[1][2] often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical parts. By contrast, a general-purpose computer, such as a personal computer (PC), is designed to be flexible and to meet a wide range of end-user needs. Embedded systems control many devices in common use today.[3]

Embedded systems are controlled by one or more main processing cores that are typically either microcontrollers or digital signal processors (DSP).[4] The key characteristic, however, is being dedicated to handle a particular task, which may require very powerful processors. For example, air traffic control systems may usefully be viewed as embedded, even though they involve mainframe computers and dedicated regional and national networks between airports and radar sites (each radar probably includes one or more embedded systems of its own).

Characteristics

1. Embedded systems are designed to do some specific task, rather than be a general-purpose computer for multiple tasks. Some also have real-time performance constraints that must be met for reasons such as safety and usability; others may have low or no performance requirements, allowing the system hardware to be simplified to reduce costs.

2. Embedded systems are not always standalone devices. Many embedded systems consist of small, computerized parts within a larger device that serves a more general purpose. For example, the Gibson Robot Guitar features an embedded system for tuning the strings, but the overall purpose of the Robot Guitar is, of course, to play music.[5] Similarly, an embedded system in an automobile provides a specific function as a subsystem of the car itself.

3. The program instructions written for embedded systems are referred to as firmware, and are stored in read-only memory or flash memory chips. They run with limited computer hardware resources: little memory, small or non-existent keyboard and/or screen.
Embedded system

An embedded system is a computer system designed to perform one or a few dedicated functions often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical parts. By contrast, a general-purpose computer, such as a personal computer (PC), is designed to be flexible and to meet a wide range of end-user needs. Embedded systems control many devices in common use today.

Embedded systems are controlled by one or more main processing cores that are typically either microcontrollers or digital signal processors (DSP). The key characteristic, however, is being dedicated to handle a particular task, which may require very powerful processors. For example, air traffic control systems may usefully be viewed as embedded, even though they involve mainframe computers and dedicated regional and national networks between airports and radar sites (each radar probably includes one or more embedded systems of its own).

Characteristics

1. Embedded systems are designed to do some specific task, rather than be a general-purpose computer for multiple tasks. Some also have real-time performance constraints that must be met for reasons such as safety and usability; others may have low or no performance requirements, allowing the system hardware to be simplified to reduce costs.

2. Embedded systems are not always standalone devices. Many embedded systems consist of small, computerized parts within a larger device that serves a more general purpose. For example, the Gibson Robot Guitar is an embedded system for tuning the strings, but the overall purpose of the Robot Guitar is, of course, to play music. Similarly, an embedded system in an automobile provides a specific function as a subsystem of the car itself.

3. The program instructions written for embedded systems are referred to as firmware, and are stored in read-only memory or flash memory chips. They run with limited computer hardware resources: little memory, small or non-existent keyboard and/or screen.
Outline of the presentation

1. Introduction
2. Verification of Open Systems
3. Verification of Timed Systems
4. Modelling Resources in Timed Systems
5. Perspectives
Outline of the presentation

1. Introduction

2. Verification of Open Systems

3. Verification of Timed Systems

4. Modelling Resources in Timed Systems

5. Perspectives
Reasoning about open systems

Concurrent games

A concurrent game is made of

- a transition system;

```
qu_0
|   ^   |
|   |   |
|   v   |
qu_1
```

```
qu_0
|   ^   |
|   |   |
|   v   |
qu_2
```

\( q_0 \) \( q_1 \) \( q_2 \)
Reasoning about open systems

Concurrent games

A concurrent game is made of

- a transition system;
- a set of agents;

\begin{figure}
\centering
\begin{tikzpicture}
  \node[draw, circle, fill=yellow, inner sep=3pt] (q0) at (0,0) {$q_0$};
  \node[draw, circle, fill=green, inner sep=3pt] (q1) at (2,1) {$q_1$};
  \node[draw, circle, fill=red, inner sep=3pt] (q2) at (2,-1) {$q_2$};
  \draw[->] (q0) to (q1);
  \draw[->] (q0) to (q2);
  \draw[->, bend right] (q1) to (q0);
\end{tikzpicture}
\end{figure}
Reasoning about open systems

Concurrent games

A concurrent game is made of

- a transition system;
- a set of agents;
- a table indicating the transition to be taken given the actions of the players.

```
 player 1
```

```
<table>
<thead>
<tr>
<th>player 1</th>
<th>player 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>q0</td>
</tr>
<tr>
<td>q2</td>
<td>q2</td>
</tr>
<tr>
<td>q1</td>
<td>q1</td>
</tr>
</tbody>
</table>
```

Player 1

Player 2
Reasoning about open systems

Alternating-time Temporal Logic

ATL formulas are built inductively using atomic propositions, Boolean combinations, and

- **temporal modalities**: \( \Diamond U \Diamond \) expresses that a \( \Diamond \)-state will be reached, and only \( \Diamond \)-states are visited in the meantime.

- **strategy quantifiers**: \( \llbracket A \rrbracket \varphi \) expresses that agent (or coalition) \( A \) has a strategy to enforce \( \varphi \).
Alternating-time Temporal Logic

ATL formulas are built inductively using atomic propositions, Boolean combinations, and
- **temporal modalities**: \(\Diamond U \Box\) expresses that a \(\Box\)-state will be reached, and only \(\Diamond\)-states are visited in the meantime.
- **strategy quantifiers**: \(\langle A \rangle \varphi\) expresses that agent (or coalition) \(A\) has a strategy to enforce \(\varphi\).
Alternating-time Temporal Logic

ATL formulas are built inductively using atomic propositions, Boolean combinations, and

- **temporal modalities**: $\bigcirc U \bigcirc$ expresses that a $\bigcirc$-state will be reached, and only $\bigcirc$-states are visited in the meantime.
- **strategy quantifiers**: $\langle A \rangle \varphi$ expresses that agent (or coalition) $A$ has a strategy to enforce $\varphi$. 

\[ \checkmark \langle \square \rangle F \bigcirc \equiv \langle \square \rangle \text{true} U \bigcirc \]

\[ \times \langle \Diamond \rangle F \bigcirc \]
Reasoning about open systems

Alternating-time Temporal Logic

ATL formulas are built inductively using atomic propositions, Boolean combinations, and
- **temporal modalities**: ○ $U$ ○ expresses that a ○-state will be reached, and only ○-states are visited in the meantime.
- **strategy quantifiers**: $\langle A \rangle \varphi$ expresses that agent (or coalition) $A$ has a strategy to enforce $\varphi$.

$\checkmark$ $\langle \Box \rangle \mathbf{F} \circ \equiv \langle \Box \rangle \text{true} U \circ$

$\times$ $\langle \Diamond \rangle \mathbf{F} \circ$

$\langle \Box \rangle \mathbf{G}(\langle \Box \rangle \mathbf{F} \circ)$
Reasoning about open systems

Alternating-time Temporal Logic

ATL formulas are built inductively using atomic propositions, Boolean combinations, and
- **temporal modalities**: \( \bigcirc \mathbf{U} \bigcirc \) expresses that a \( \bigcirc \)-state will be reached, and only \( \bigcirc \)-states are visited in the meantime.
- **strategy quantifiers**: \( \langle \langle A \rangle \rangle \varphi \) expresses that agent (or coalition) \( A \) has a strategy to enforce \( \varphi \).

\[
\checkmark \quad \langle \langle \bigcirc \rangle \rangle \mathbf{F} \bigcirc \equiv \langle \langle \bigcirc \rangle \rangle \text{true} \mathbf{U} \bigcirc
\]

\[
\times \quad \langle \langle \Box \rangle \rangle \mathbf{F} \bigcirc
\]

\[
\langle \langle \bigcirc \rangle \rangle \mathbf{G}(\langle \langle \Box \rangle \rangle \mathbf{F} \bigcirc) \equiv \langle \langle \bigcirc \rangle \rangle \mathbf{G} p
\]

\(p\)
Reasoning about open systems

Alternating-time Temporal Logic

ATL formulas are built inductively using atomic propositions, Boolean combinations, and

- **temporal modalities**: \( \Diamond U \Diamond \) expresses that a \( \Diamond \)-state will be reached, and only \( \Diamond \)-states are visited in the meantime.
- **strategy quantifiers**: \( \llbracket A \rrbracket \varphi \) expresses that agent (or coalition) \( A \) has a strategy to enforce \( \varphi \).

\[
\checkmark \quad \llbracket \Diamond \rrbracket F \Diamond \equiv \llbracket \Diamond \rrbracket \text{true} \ U \Diamond
\]

\[
\times \quad \llbracket \Box \rrbracket F \Diamond
\]

\[
\times \quad \llbracket \Diamond \rrbracket G(\llbracket \Box \rrbracket F \Diamond) \equiv \llbracket \Diamond \rrbracket G p
\]
Reasoning about open systems

Alternating-time Temporal Logic

ATL formulas are built inductively using atomic propositions, Boolean combinations, and

- **temporal modalities**: $\bigcirc \mathbf{U} \bigcirc$ expresses that a $\bigcirc$-state will be reached, and only $\bigcirc$-states are visited in the meantime.

- **strategy quantifiers**: $\langle A \rangle \varphi$ expresses that agent (or coalition) $A$ has a strategy to enforce $\varphi$.

Theorem (AHK02, LMO07)

ATL model checking is PTIME-complete (or $\Delta^p_3$-complete when the transition table is encoded symbolically).
Another semantics: ATL with strategy contexts [BDML09]

\[ \langle \Box \rangle \, G \, (\langle \Box \rangle \, F \, \Diamond) \]
Another semantics: ATL with strategy contexts \[ BDML09 \]

Evaluate the formula on the execution tree:

\( \langle \bigcirc \rangle \ G( \langle \square \rangle \ F \bigcirc ) \)
Another semantics: ATL with strategy contexts \[\text{BDML09}\]

Evaluate the formula on the execution tree:
- apply a strategy of Player \(\bigcirc\);
Another semantics: ATL with strategy contexts [BDML09]

Evaluate the formula on the execution tree:
- apply a strategy of Player \( \bigcirc \);
- in the remaining tree, check that Player \( \square \) can always enforce a visit to \( \bigcirc \).
What $\text{ATL}_{sc}$ can express

- All $\text{ATL}^*$ properties;
What $\text{ATL}_{sc}$ can express

- All $\text{ATL}^*$ properties;
- Client-server interactions for accessing a shared resource:

$$\langle \cdot \rangle G \left[ \bigwedge_{c \in \text{Clients}} \langle \cdot \rangle \mathbf{F} \text{access}_c \right] \land \left[ \bigwedge_{c \neq c'} \neg \text{access}_c \land \text{access}_{c'} \right]$$
What $\text{ATL}_{sc}$ can express

- All $\text{ATL}^*$ properties;
- Client-server interactions for accessing a shared resource:

$$\langle \cdot \rangle \text{G} \bigg[ \bigwedge_{c\in \text{Clients}} \langle c \rangle F \text{access}_c \bigwedge \neg \bigwedge_{c \neq c'} \text{access}_c \land \text{access}_c' \bigg]$$

- Existence of Nash equilibria:

$$\langle A_1, \ldots, A_n \rangle \bigwedge_i \left( \langle A_i \rangle \varphi_{A_i} \Rightarrow \varphi_{A_i} \right)$$
What $\text{ATL}_{sc}$ can express

- All $\text{ATL}^*$ properties;
- Client-server interactions for accessing a shared resource:

$$\langle \cdot \rangle \ G \left[ \bigwedge_{c \in \text{Clients}} \langle c \cdot \rangle \ F \text{access}_c \right.$$  

$$\left. \quad \land \quad \neg \bigwedge_{c \neq c'} \text{access}_c \land \text{access}_{c'} \right]$$

- Existence of Nash equilibria:

$$\langle A_1, \ldots, A_n \cdot \rangle \bigwedge_i (\langle A_i \cdot \rangle \varphi_{A_i} \Rightarrow \varphi_{A_i})$$

- Existence of dominating strategy:

$$\langle A \cdot \rangle \ [B] (\neg \varphi \Rightarrow [A] \neg \varphi)$$
Verifying $\text{ATL}_{sc}$ properties

**Theorem (DLM10)**

Given a CGS $\mathcal{C}$, a state $\ell_0$ and an $\text{ATL}_{sc}$ formula $\varphi$, we can build an alternating parity tree automaton $A$ s.t.

$$\mathcal{L}(A) \neq \emptyset \iff \mathcal{C}, \ell_0 \models \emptyset \varphi.$$  

$A$ has size $d$-exponential, where $d$ is the maximal number of nested quantifiers in $\varphi$.

Checking whether $\mathcal{C}, \ell_0 \models \emptyset \varphi$ is in $(d+1)$-EXPTIME.
Verifying $\text{ATL}_{sc}$ properties

Theorem (DLM10)

Given a CGS $C$, a state $\ell_0$ and an $\text{ATL}_{sc}$ formula $\varphi$, we can build an alternating parity tree automaton $A$ s.t.

$$\mathcal{L}(A) \neq \emptyset \iff C, \ell_0 \models_{\emptyset} \varphi.$$  

$A$ has size $d$-exponential, where $d$ is the maximal number of nested quantifiers in $\varphi$.

Checking whether $C, \ell_0 \models_{\emptyset} \varphi$ is in $(d+1)$-EXPTIME.

Proposition (DLM11 [unpublished])

Checking whether $C, \ell_0 \models_{\emptyset} \varphi$ is $(d-1)$-EXPSPACE-hard.
Verification of open systems: conclusions and perspectives

ATL model checking:
- revisiting of the basic setting; \cite{LMO07, LMO08}
- extension to the timed setting; \cite{LMO06, BLMO07}
- “strategy-context” semantics, useful for non-zero-sum objectives; \cite{BDLM09, DLM10}

Boolean Nash equilibria in concurrent (timed) games; \cite{BBM10a, BBM10b}

Permissive strategies. \cite{BDMR09, BMOU11}

Current research directions:
- Better understanding of ATL\(_{sc}\) and Strategy Logic;
- Quantitative Nash equilibria;
- Permissive strategies in the timed setting.
Outline of the presentation

1. Introduction
2. Verification of Open Systems
3. Verification of Timed Systems
4. Modelling Resources in Timed Systems
5. Perspectives
Reasoning about timed systems

Timed automata
A timed automaton is made of
- a transition system,

Example
Reasoning about timed systems

Timed automata

A timed automaton is made of
- a transition system,
- a set of clocks,

Example
Reasoning about timed systems

Timed automata
A timed automaton is made of
- a transition system,
- a set of clocks,
- a labelling of transitions with timing informations.

Example

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

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

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

\[
y \geq 2, \ y := 0
\]
Reasoning about timed systems

Timed automata

A timed automaton is made of
- a transition system,
- a set of clocks,
- a labelling of transitions with timing informations.

Example
Reasoning about timed systems

Timed automata

A timed automaton is made of
- a transition system,
- a set of clocks,
- a labelling of transitions with timing informations.

Example

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

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

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

Timed automata

A **timed automaton** is made of
- a transition system,
- a set of clocks,
- a labelling of transitions with timing informations.

Example

Graph showing transitions with timing conditions:
- Transition from state 1 to state 2 with conditions $x \leq 2$, $x:=0$.
- Transition from state 2 to state 1 with conditions $y \geq 2$, $y:=0$.
- Transition from state 1 to state 3 with conditions $x=0 \land y \geq 2$.

Graph with axes $x$ and $y$.
Reasoning about timed systems

**Timed automata**

A timed automaton is made of

- a transition system,
- a set of clocks,
- a labelling of transitions with timing informations.

**Example**

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

Timed automata

A timed automaton is made of

- a transition system,
- a set of clocks,
- a labelling of transitions with timing informations.

Example
Reasoning about timed systems

Timed automata

A timed automaton is made of

- a transition system,
- a set of clocks,
- a labelling of transitions with timing informations.

Example

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

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

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

Timed automata
A timed automaton is made of
- a transition system,
- a set of clocks,
- a labelling of transitions with timing informations.

Example

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

Timed automata

A timed automaton is made of

- a transition system,
- a set of clocks,
- a labelling of transitions with timing informations.

Example

\[ x = 1 \]
\[ y := 0 \]
\[ x \leq 2, \ x := 0 \]
\[ y \geq 2, \ y := 0 \]
\[ x = 0 \land y \geq 2 \]
Reasoning about timed systems

Timed automata

A timed automaton is made of

- a transition system,
- a set of clocks,
- a labelling of transitions with timing informations.

Example

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

Timed automata

A **timed automaton** is made of

- a transition system,
- a set of clocks,
- a labelling of transitions with timing informations.

Example

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

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

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

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

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

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

\[
x = 0 \land \\
y &\geq 2
\]
Reasoning about timed systems

**Timed automata**

A **timed automaton** is made of

- a transition system,
- a set of clocks,
- a labelling of transitions with timing informations.

**Example**

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

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

**Timed automata**

A **timed automaton** is made of

- a transition system,
- a set of clocks,
- a labelling of transitions with timing informations.

**Theorem (AD90)**

*Reachability in timed automata is PSPACE-complete.*
Implementing timed automata

The semantics of timed automata is not realistic

<table>
<thead>
<tr>
<th></th>
<th>timed automata</th>
<th>real-life CPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>frequency</td>
<td>infinite</td>
<td>finite</td>
</tr>
<tr>
<td>precision</td>
<td>arbitrary</td>
<td>bounded</td>
</tr>
<tr>
<td>synchronization</td>
<td>perfect</td>
<td>delayed</td>
</tr>
</tbody>
</table>

Some properties may be lost at implementation.
Implementing timed automata

The semantics of timed automata is not realistic

<table>
<thead>
<tr>
<th></th>
<th>timed automata</th>
<th>real-life CPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>frequency</td>
<td>infinite</td>
<td>finite</td>
</tr>
<tr>
<td>precision</td>
<td>arbitrary</td>
<td>bounded</td>
</tr>
<tr>
<td>synchronization</td>
<td>perfect</td>
<td>delayed</td>
</tr>
</tbody>
</table>

Some properties may be lost at implementation.

Program semantics (DDR04)

- A different semantics modelling the behaviour on a CPU;
- over-approximated by the enlarged semantics:

\[ x \in [a, b] \sim x \in [a - \Delta, b + \Delta]. \]
Robust safety

Example

\[ x = 1 \quad \text{and} \quad y \geq 2 \]
Robust safety

Example

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

Example

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

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

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

\[ x \leq \Delta \land y \geq 2 - \Delta \]
Robust safety

Example

$x \in [1-\Delta, 1+\Delta]$

$y := 0$

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

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

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

$y \geq 2 - \Delta$, $y := 0$
Robust safety

Example

\[ x \in [1-\Delta, 1+\Delta] \]

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

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

\[ x \leq \Delta \land y \geq 2-\Delta \]
Robust safety

Example

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

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

\[ x \leq \Delta \wedge y \geq 2 - \Delta \]
Robust safety

Example

\[
x \in [1-\Delta, 1+\Delta] \\
y := 0 \\
x \leq 2+\Delta, \ x := 0 \\
y \geq 2-\Delta, \ y := 0 \\
x \leq \Delta \land y \geq 2-\Delta
\]
Robust safety

Example

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

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

\[x \leq \Delta \land y \geq 2-\Delta\]
\[y \geq 2-\Delta, \ y := 0\]
Robust safety

Example

\[ x \in [1 - \Delta, 1 + \Delta] \]

\[ y := 0 \]

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

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

\[ x \leq \Delta \land y \geq 2 - \Delta \]
Robust safety

Example

\[ x \in [1-\Delta, 1+\Delta] \]

\[ y := 0 \]

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

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

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

Example

$x \in [1-\Delta, 1+\Delta]$

$y := 0$

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

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

$x \leq \Delta \land y \geq 2-\Delta$
Robust safety

Example

\[ x \in [1-\Delta, 1+\Delta] \]

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

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

\[ x \leq \Delta \land y \geq 2-\Delta \]
Robust safety

Example

\[
x \in [1-\Delta, 1+\Delta]
\]

\[
y := 0
\]

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

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

\[
x \leq \Delta \land y \geq 2 - \Delta
\]
Robust safety

Example

\[ x \in [1-\Delta, 1+\Delta] \]
\[ y := 0 \]
\[ x \leq 2 + \Delta, x := 0 \]
\[ x \leq \Delta \land y \geq 2 - \Delta \]
\[ y \geq 2 - \Delta, y := 0 \]
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 $(\ell, r) \xrightarrow{\gamma} (\ell, r')$. 

\[ y = \begin{cases} 3 & \text{if } x = 0 \\ 2 & \text{if } 0 < x < 1 \\ 1 & \text{if } 1 \leq x < 2 \\ 0 & \text{if } x = 2 \\ 0 & \text{if } x = 3 \end{cases} \]
For any location $\ell$ and any two regions $r$ and $r'$, if
- $r \cap r' \neq \emptyset$ and
- $(\ell, r')$ belongs to an SCC of $\mathcal{R}(A)$,
then we add a transition $(\ell, r) \xrightarrow{\gamma} (\ell, r')$. 

![Diagram](image-url)
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 $(\ell, r) \xrightarrow{\gamma} (\ell, r')$. 
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 $(\ell, r) \xrightarrow{\gamma} (\ell, r')$. 
Checking robust safety – 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 $(\ell, r) \xrightarrow{\gamma} (\ell, r')$. 
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 $(\ell, r) \xrightarrow{\gamma} (\ell, r')$. 
Checking robust safety – 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 $(\ell, r) \xrightarrow{\gamma} (\ell, r')$.

Theorem (DDMR04, DDMR08)

**Robust safety checking is PSPACE-complete.**

Theorem (BMR06)

**Robust LTL model checking is PSPACE-complete.**
Robust model checking – Channel automata

Channel automaton

A channel automaton with rewriting and occurrence testing is made of
- a transition system,
- an unbounded FIFO channel,
- a labelling of transition with channel read/write informations.

Example

\( a!,b! \)

\( s \)

\( \#! \)

\( \#? \)

\( a \rightarrow \{a,b\} \)

\( a?,b? \)

\( t \)

\( \#! \)

\( \#? \)

\( \text{zero}(a)\)
Robust model checking – Channel automata

**Channel automaton**

A channel automaton with rewriting and occurrence testing is made of
- a transition system,
- an unbounded FIFO channel,
- a labelling of transition with channel read/write informations.

**Proposition (BMOW07)**

*Cycle-bounded reachability in channel automata with rewriting and occurrence testing is PSPACE-complete.*
Robust model checking – Channel automata

Encoding timed automata as channel automata

one time unit = one cycle of the channel
Robust model checking – Channel automata

Encoding timed automata as channel automata

one time unit = one cycle of the channel

Example

\[ x \in [1-\Delta, 1+\Delta] \]

\[ y := 0 \]

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

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

\[ \lfloor x \rfloor = 0 \]

\[ \lfloor y \rfloor = 0 \]

state of \( C \)

\[ \begin{array}{cccccc}
  x, y & \Delta & \Delta & \Delta & \Delta \\
\end{array} \]
Robust model checking – Channel automata

Encoding timed automata as channel automata

one time unit = one cycle of the channel

Example

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

state of \( C \)

\[
\begin{array}{c|c|c|c|c|c}
\Delta & x, y & \Delta & \Delta & \Delta
\end{array}
\]
Robust model checking – Channel automata

Encoding timed automata as channel automata

one time unit = one cycle of the channel

Example

\[ x \in [1 - \Delta, 1 + \Delta] \]
\[ y := 0 \]
\[ x \leq 2 + \Delta, \; x := 0 \]
\[ y \geq 2 - \Delta, \; y := 0 \]

\[ \lfloor x \rfloor = 0 \]
\[ \lfloor y \rfloor = 0 \]

state of \( C \)
Robust model checking – Channel automata

Encoding timed automata as channel automata

one time unit = one cycle of the channel

Example

\[ x \in [1-\Delta, 1+\Delta] \]

\[ y := 0 \]

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

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

state of \( C \)

\[ |x| = 0 \]

\[ |y| = 0 \]

\[ \Delta \ \Delta \ \Delta \ x, y \ \Delta \]
Robust model checking – Channel automata

Encoding timed automata as channel automata

one time unit = one cycle of the channel

Example

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

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

state of $\mathcal{C}$
Robust model checking – Channel automata

Encoding timed automata as channel automata

one time unit = one cycle of the channel

Example

\[ x \in [1-\Delta, 1+\Delta] \]

\[ y := 0 \]

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

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

state of \( C \)

waiting for \( x \)

state of \( C \)

\[ \lfloor x \rfloor = 0 \]

\[ \lfloor y \rfloor = 0 \]
Robust model checking – Channel automata

Encoding timed automata as channel automata

one time unit = one cycle of the channel

Example

\[ \begin{align*}
  x \in [1 - \Delta, 1 + \Delta] \\
  y := 0 \\
  x \leq 2 + \Delta, \quad x := 0 \\
  y \geq 2 - \Delta, \quad y := 0 \\
  x \leq \Delta \land y \geq 2 - \Delta
\end{align*} \]
Robust model checking – Channel automata

Encoding timed automata as channel automata

one time unit = one cycle of the channel

Theorem (BMR08)

Robust model checking for CoFlatMTL is EXPSPACE-complete.

Theorem (BMS11 [unpublished])

Robust safety can be checked in PSPACE using channel automata.
Verification of timed systems: conclusions and perspectives

Timed automata are a well-established formalism for modelling real-time systems.

- separation of MTL and TPTL; [BCM05, BCM10]
- definition of a decidable extension of MITL with punctuality; [BMOW07, BMOW08]
- implementability issues. [DDMR04, BMR06, DDMR08, BMR08]

Current research directions:

- Study different approaches to implementability issues;
- Synthesis of implementable systems.
Outline of the presentation

1. Introduction
2. Verification of Open Systems
3. Verification of Timed Systems
4. Modelling Resources in Timed Systems
5. Perspectives
Modelling resources

Weighted timed automata

A **weighted timed automaton** is made of
  - a timed automaton;

Example

![Diagram of a weighted timed automaton](image)
Modelling resources

**Weighted timed automata**

A weighted timed automaton is made of:
- a timed automaton;
- cost variables;

**Example**

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

Cost:
- \( 1.3 \times 5 = 4.5 \)
- \( 1.7 \times 6 = 10.2 \)
Modelling resources

Weighted timed automata

A weighted timed automaton is made of

- a timed automaton;
- cost variables;
- cost information on states and transitions.

Example

\[
\begin{align*}
\dot{p} &= 5, \\
x &\leq 2, y := 0, \\
p &= 2, \\
y &= 0, \\
p &= 2, \\
x &\geq 3, \\
p &= 1, \\
x &\geq 3, \\
p &= 7
\end{align*}
\]
Modelling resources

**Weighted timed automata**

A **weighted timed automaton** is made of
- a timed automaton;
- cost variables;
- cost information on states and transitions.

**Example**

![Diagram of a weighted timed automaton]

- \( \dot{p} = 5 \):
  - \( x \leq 2, y := 0 \)
  - \( p += 2 \)

- \( y = 0 \):
  - \( \dot{p} = 6 \)
  - \( x \geq 3 \)
  - \( p += 1 \)

- \( p = 1 \):
  - \( x \geq 3 \)
  - \( p += 7 \)

Cost:
- \( 1.3 \times 5 = 6.5 \)
- \( 1.7 \times 6 = 10.2 \)
Modelling resources

Weighted timed automata

A *weighted timed automaton* is made of
- a timed automaton;
- cost variables;
- cost information on states and transitions.

Example

\[
\begin{align*}
\dot{p} &= 5 \\
x \leq 2, & \quad y := 0 \\
p & \rightarrow p + = 2 \\

\dot{p} &= 6 \\
x \geq 3, & \quad p + = 1 \\

\dot{p} &= 1 \\
x \geq 3, & \quad p + = 7 \\
\end{align*}
\]

Cost:

\[
\begin{align*}
1.3 & \times 5 = 4.5 \\
1.7 & \times 6 = 10.2 \\
\end{align*}
\]
Modelling resources

**Weighted timed automata**

A *weighted timed automaton* is made of
- a timed automaton;
- cost variables;
- cost information on states and transitions.

**Theorem (ALP01, BFH⁺01, BBL04)**

*Optimal reachability is PSPACE-complete in weighted timed automata.*
Energy constraints – lower-bound constraints

Example

\[+2 \rightarrow -1 \rightarrow +8 \rightarrow -2\]
\[x := 0 \rightarrow x = 1\]

Theorem (BFLMS08, BFLM10)
Optimization under lower-bound constraint is decidable on one-clock weighted timed automata.
(also for exponential costs).
Energy constraints – lower-bound constraints

Example

\[ +2 \rightarrow -1 \rightarrow +8 \rightarrow -2 \]

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

\[ p = \frac{x}{1} \]

Theorem (BFLMS08, BFLM10)

Optimization under lower-bound constraint is decidable on one-clock weighted timed automata.

(also for exponential costs).
Energy constraints – lower-bound constraints

Example

\[ +2 \xrightarrow{-1} +8 \xrightarrow{-2} -2 \]

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

Theorem (BFLMS08, BFLM10)

Optimization under lower-bound constraint is decidable on one-clock weighted timed automata.

(also for exponential costs).
Energy constraints – lower-bound constraints

Example

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

Theorem (BFLMS08, BFLM10)

Optimization under lower-bound constraint is decidable on one-clock weighted timed automata.
(also for exponential costs).
Energy constraints – lower-bound constraints

Example

Theorem (BFLMS08, BFLM10) Optimizations under lower-bound constraint is decidable on one-clock weighted timed automata. (also for exponential costs.)
Energy constraints – lower-bound constraints

Example

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

Theorem (BFLMS08, BFLM10)

Optimization under lower-bound constraint is decidable on one-clock weighted timed automata.
Energy constraints – lower-bound constraints

Example

\[\begin{align*}
+2 & \rightarrow -1 \rightarrow +8 \rightarrow -2 \\
x := 0 & \quad x = 1
\end{align*}\]

\[\frac{dp}{dt} = 8 \cdot p\]

Theorem (BFLMS08, BFLM10)

Optimization under lower-bound constraint is decidable on one-clock weighted timed automata (also for exponential costs).
Energy constraints – Interval constraints

Example

Theorem (BFLMS08)
Reachability in one-clock weighted timed games is undecidable.
Energy constraints – Interval constraints

Example

\[ x := 0 \rightarrow \begin{array}{c} +2 \\ \downarrow \end{array} \rightarrow \begin{array}{c} -1 \\ \uparrow \end{array} \rightarrow \begin{array}{c} +8 \\ \downarrow \end{array} \rightarrow \begin{array}{c} -2 \\ \uparrow \end{array} \rightarrow \begin{array}{c} x = 1 \\ \downarrow \end{array} \rightarrow \begin{array}{c} -2 \\ \uparrow \end{array} \]

Theorem (BFLMS08)
Reachability in one-clock weighted timed games is undecidable.
Energy constraints – Interval constraints

Example

Theorem (BFLMS08)

Reachability in one-clock weighted timed games is undecidable.
Energy constraints – Interval constraints

Example

Theorem (BFLMS08)

Reachability in one-clock weighted timed games is undecidable.
Undecidability proof: encoding of a two-counter machine

\[
x := 0 \rightarrow -6 \rightarrow +1 \rightarrow 30 \rightarrow -1 \rightarrow -n \rightarrow x = 1
\]
Undecidability proof: encoding of a two-counter machine
Undecidability proof: encoding of a two-counter machine

\[ x := 0 \quad -6 \quad p = 0 \quad 30 \quad p = 5 \quad -n \quad x = 1 \]

\[ p = 0 \quad 5 - e \quad x = 1 \quad 0 \]
Undecidability proof: encoding of a two-counter machine

\[ x := 0 \quad \rightarrow \quad -6 \quad \rightarrow \quad p = 0 \quad \rightarrow \quad 30 \quad \rightarrow \quad p = 5 \quad \rightarrow \quad -n \quad \rightarrow \quad x = 1 \]

\[ \begin{align*}
  x &:= 0 \\
  p &:= 0 \\
  5 - e &:= 0 \\
  \frac{5 - e}{6} &:= 0 \\
  x &:= 1 \\
  n &:= 0
\end{align*} \]
Undecidability proof: encoding of a two-counter machine

\[\begin{align*}
&x := 0 \\
&-6 \quad p = 0 \\
&30 \quad p = 5 \\
&-n \quad x = 1
\end{align*}\]
Undecidability proof: encoding of a two-counter machine

\[
\begin{align*}
x &= 0 & -6 & \quad p &= 0 & 30 & \quad p &= 5 & -n & \quad x &= 1 \\
\end{align*}
\]
Undecidability proof: encoding of a two-counter machine

\[ x := 0 \rightarrow \begin{array}{c} -6 \\ p = 0 \end{array} \rightarrow \begin{array}{c} 30 \\ p = 5 \end{array} \rightarrow \begin{array}{c} -n \\ x = 1 \end{array} \]

\[ e = \frac{1}{2c_1 \cdot 3c_2} \]

- \( n = 3 \): increment \( c_1 \)
- \( n = 2 \): increment \( c_2 \)
- \( n = 12 \): decrement \( c_1 \)
- \( n = 18 \): decrement \( c_2 \)
Quantitative verification: conclusions and perspectives

- Weighted timed automata are a natural framework for modelling and reasoning about resource consumption.
  - WCTL model checking is undecidable, except when restricting to the one-clock setting; \[\text{[BBM06, BLM07]}\]
  - Same results for computing optimal strategies in weighted timed games; \[\text{[BBM06, BLMR06]}\]
  - Energy constraints: some decidability results. \[\text{[BFLMS08, BFLM10]}\]

Current research directions:
- Optimization under lower-bound constraints:
  - decidability in the general case;
  - extension to games;
- Interval constraints:
  - Can we compute an upper bound?
  - What about weak upper bound?
Outline of the presentation

1. Introduction
2. Verification of Open Systems
3. Verification of Timed Systems
4. Modelling Resources in Timed Systems
5. Perspectives
Perspectives – Open systems

**ATL with strategy contexts**

- Several remaining open questions:
  - satisfiability;
  - corresponding behavioural equivalence, ...
- Extension to randomized strategies;
- Extension to the timed setting.

~~ Ph.D. thesis of Arnaud Da Costa-Lopes ~

**Nash equilibria**

- Specialized algorithms for computing Nash equilibria;
  \[\text{[BBM10a, BBM10b]}\]
- Extension to quantitative objectives, randomized strategies, other kinds of equilibria, ...

~~ Ph.D. thesis of Romain Brenguier ~
### Perspectives – Implementability of timed systems

#### Robustness and implementability
- Robust-controller synthesis;
- Permissive strategies in a timed setting.

#### Efficient algorithms
- Symbolic, zone-based algorithms.

#### Develop new approaches to implementability
- Probabilistic approach to robustness (instead of worst-case);
- Shrinkable timed automata.

~ PhD. thesis of Ocan Sankur
Perspectives – Robustness in weighted timed automata

Undecidability proofs require arbitrary precision

- Energy constraints under imprecision: $[0,1] \rightarrow [2,4] \rightarrow [-6,-3]$

$x:=0, x=1, \text{update}$

$\sim$ revisit all undecidable problems in an imprecise setting.