Parametric Deadlock-Freeness Checking
Timed Automata

Étienne André\textsuperscript{1,2}

\textsuperscript{1} LIPN, Université Paris 13, Sorbonne Paris Cité, CNRS, France
\textsuperscript{2} École Centrale de Nantes, IRCCyN, CNRS, UMR 6597, France
Context: timed model checking

- Timed model checking

\[ y = \text{delay} \]
\[ x := 0 \]
\[ x < \text{period} \]

A model of the system

A property to be satisfied

\( \square \) is unreachable
Context: timed model checking

- Timed model checking

\[
y = \text{delay} \\
x := 0 \\
x < \text{period}
\]

A model of the system

A property to be satisfied

Question: does the model of the system satisfy the property?
Context: timed model checking

- Timed model checking

\[
y = \text{delay}
\]
\[
x := 0
\]
\[
x < \text{period}
\]

? \[
\]

- \text{is unreachable}

A model of the system

A property to be satisfied

- Question: does the model of the system satisfy the property?

Yes

No

Counterexample
Context: parametric timed model checking

- Timed model checking

A model of the system

A property to be satisfied

Question: for what values of the parameters does the model of the system satisfy the property?

Yes if...

2^{\text{delay}} > \text{period} \\
\land \text{period} < 20.46
Outline

1. Parametric Timed Automata
2. Deadlock-Checking
3. Backward Under-approximation
4. Experiments
5. Conclusion and Perspectives
Outline

1. Parametric Timed Automata
2. Deadlock-Checking
3. Backward Under-approximation
4. Experiments
5. Conclusion and Perspectives
Timed automaton (TA)

- Finite state automaton (sets of locations)
Timed automaton (TA)

- Finite state automaton (sets of locations and actions)
Timed automaton (TA)

- Finite state automaton (sets of locations and actions) augmented with a set $X$ of clocks [Alur and Dill, 1994, Henzinger et al., 1994]
- Real-valued variables evolving linearly at the same rate

\[
\begin{align*}
\text{press?} & \quad \rightarrow \quad \text{cup!} \\
& \quad \quad \quad \text{press?} \\
\end{align*}
\]
Timed automaton (TA)

- Finite state automaton (sets of locations and actions) augmented with a set $X$ of clocks [Alur and Dill, 1994, Henzinger et al., 1994]
  - Real-valued variables evolving linearly at the same rate

- Features
  - Location invariant: constraint to be verified to stay at a location

\[
\begin{align*}
  &\text{press?} \quad y \leq 5 \quad \text{cup!} \\
  &\text{press?} \quad y \leq 8 \\
\end{align*}
\]
Timed automaton (TA)

- Finite state automaton (sets of locations and actions) augmented with a set $X$ of clocks [Alur and Dill, 1994, Henzinger et al., 1994]
  - Real-valued variables evolving linearly at the same rate

- Features
  - Location invariant: constraint to be verified to stay at a location
  - Transition guard: constraint to be verified to enable a transition

\[
\begin{align*}
  y &= 8 \\
  \text{coffee!}
\end{align*}
\]

```
x := 0 \\
y := 0 \\
y = 5 \\
x >= 1 \\
pRESS? \\
cup! \\
y <= 8:
```

\[
\begin{align*}
  y &= 8 \\
  \text{coffee!}
\end{align*}
\]
Timed automaton (TA)

- Finite state automaton (sets of locations and actions) augmented with a set $X$ of clocks [Alur and Dill, 1994, Henzinger et al., 1994]
  - Real-valued variables evolving linearly at the same rate

- Features
  - Location invariant: constraint to be verified to stay at a location
  - Transition guard: constraint to be verified to enable a transition
  - Clock reset: some of the clocks can be set to 0 at each transition

```
press? 
x := 0
y := 0
```

```
press?
y = 8
cup!
```

```
x := 0
y := 0
```

```
x ≥ 1
```

```
y = 5
```

```
x := 0
```

```
y ≤ 8:
```

```
y ≤ 5
```

```
press?
```
Concrete semantics of timed automata

- **Concrete state** of a TA: pair \((l, w)\), where
  - \(l\) is a location,
  - \(w\) is a valuation of each clock

- **Concrete run**: alternating sequence of concrete states and actions or time elapse
Examples of concrete runs

- Possible concrete runs for the coffee machine
Examples of concrete runs

- Possible concrete runs for the coffee machine
  - Coffee with no sugar

```
- x := 0
- y := 0
```

```
x 0
y 0
```
Examples of concrete runs

Possible concrete runs for the coffee machine

- Coffee with no sugar

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>15.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>15.4</td>
</tr>
</tbody>
</table>
Examples of concrete runs

Possible concrete runs for the coffee machine

- Coffee with no sugar
Examples of concrete runs

Possible concrete runs for the coffee machine

- Coffee with no sugar

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>15.4</th>
<th>0</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>0</td>
<td>15.4</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>
Examples of concrete runs

- Possible concrete runs for the coffee machine
  - Coffee with no sugar

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>15.4</th>
<th>0</th>
<th>5</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>0</td>
<td>15.4</td>
<td>0</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Examples of concrete runs

Possible concrete runs for the coffee machine

Coffee with no sugar

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>15.4</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>15.4</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>
Examples of concrete runs

Possible concrete runs for the coffee machine

- Coffee with no sugar
**Examples of concrete runs**

![Diagram of a coffee machine automaton]

- **Possible concrete runs for the coffee machine**
  - **Coffee with no sugar**
    - 15.4
    - press?
    - 5
    - cup!
    - 3
    - coffee!
  - x: 0 15.4 0 5 5 8 8
  - y: 0 15.4 0 5 5 8 8

- **Coffee with 2 doses of sugar**
  - x: 0
  - y: 0
Examples of concrete runs

- Possible concrete runs for the coffee machine
  - Coffee with no sugar
    - x: 0, 15.4, 0, 5, 5, 8, 8
    - y: 0, 15.4, 0, 5, 5, 8, 8
  - Coffee with 2 doses of sugar
    - x: 0, 0
    - y: 0, 0
Examples of concrete runs

Possible concrete runs for the coffee machine

- Coffee with no sugar

- Coffee with 2 doses of sugar
Examples of concrete runs

Possible concrete runs for the coffee machine

- Coffee with no sugar

\[
\begin{array}{cccccc}
\text{press?} & 15.4 & \text{press?} & 5 & \text{cup!} & 3 \\
0 & 15.4 & 0 & 5 & 5 & 8 \\
0 & 15.4 & 0 & 5 & 5 & 8 \\
\end{array}
\]

- Coffee with 2 doses of sugar

\[
\begin{array}{ccc}
\text{press?} & 1.5 & \text{press?} \\
0 & 1.5 & 0 \\
0 & 1.5 & 1.5 \\
\end{array}
\]
Examples of concrete runs

Possible concrete runs for the coffee machine

- **Coffee with no sugar**

- **Coffee with 2 doses of sugar**

---

É. André (U. Paris 13)
Examples of concrete runs

![Diagram of a timed automaton with states and transitions]

- Possible concrete runs for the coffee machine
  - Coffee with no sugar
    - States and transitions with initial conditions:
      - \( x := 0 \)
      - \( y := 0 \)
      - Press?: \( x \geq 1 \)
      - \( x := 0 \)
      - \( y = 5 \)
      - \( \text{cup!} \)
      - \( y \leq 8 \)
      - \( y = 8 \)
      - \( \text{coffee!} \)

    - Table:
      | State | \( x \) | \( y \) |
      |-------|-------|-------|
      | 15.4  | 0     | 0     |
      | 5     | 5     | 5     |
      | 3     | 8     | 8     |
  - Coffee with 2 doses of sugar
    - States and transitions with initial conditions:
      - \( x := 0 \)
      - \( y := 0 \)
      - Press?: \( x = 0 \)
      - \( x := 0 \)
      - \( y = 0 \)
      - \( \text{coffee!} \)

    - Table:
      | State | \( x \) | \( y \) |
      |-------|-------|-------|
      | 1.5   | 0     | 0     |
      | 2.7   | 0     | 0     |
      | 4.2   | 4.2   | 4.2   |
Examples of concrete runs

Possible concrete runs for the coffee machine

- Coffee with no sugar

- Coffee with 2 doses of sugar
Examples of concrete runs

- Possible concrete runs for the coffee machine
  - Coffee with no sugar
    - Initial state: \( x := 0 \), \( y := 0 \)
    - Transition: press? \( x := 0 \), \( y := 0 \)
    - Transition: \( y \leq 5 \)
    - Transition: press? \( x \geq 1 \)
    - Transition: \( y = 5 \)
    - Transition: \( y \leq 8 \)
    - Final state: coffee!

  - Table:
    | \( x \) | 0 | 15.4 | 0 | 5 | 5 | 8 | 8 |
    | \( y \) | 0 | 15.4 | 0 | 5 | 5 | 8 | 8 |

  - Coffee with 2 doses of sugar
    - Initial state: \( x := 0 \), \( y := 0 \)
    - Transition: press? \( x := 0 \), \( y := 0 \)
    - Transition: \( y \leq 5 \)
    - Transition: press? \( x \geq 1 \)
    - Transition: \( y = 5 \)
    - Transition: \( y \leq 8 \)
    - Final state: coffee!

  - Table:
    | \( x \) | 0 | 0 | 1.5 | 0 | 2.7 | 0 | 0.8 | 0.8 |
    | \( y \) | 0 | 0 | 1.5 | 1.5 | 4.2 | 4.2 | 5 | 5 |
Examples of concrete runs

- Possible concrete runs for the coffee machine
  - Coffee with no sugar
    - States: 15.4 → press? → 5 → cup! → 3 → coffee!
    - Variables:
      - \( x \): 0 → 15.4 → 0 → 5 → 5 → 8 → 8
      - \( y \): 0 → 15.4 → 0 → 5 → 5 → 8 → 8
  - Coffee with 2 doses of sugar
    - States: press? → 1.5 → press? → 2.7 → press? → 0.8 → cup! → 3 → coffee!
    - Variables:
      - \( x \): 0 → 0 → 1.5 → 0 → 2.7 → 0 → 0.8 → 0.8 → 3.8
      - \( y \): 0 → 0 → 1.5 → 1.5 → 4.2 → 4.2 → 5 → 5 → 8

\[ y = 8 \]
\[ \text{coffee!} \]
\[ y \leq 5 \]
\[ \text{press?} \]
\[ x := 0 \]
\[ y := 0 \]
\[ \text{cup!} \]
\[ y = 5 \]
\[ \text{press?} \]
\[ x \geq 1 \]
\[ x := 0 \]
Examples of concrete runs

- **Possible concrete runs for the coffee machine**

  **Coffee with no sugar**

  - Transition sequence: 15.4, press?, 5, cup!, 3, coffee!
  - States: x = 0, y = 0, x = 15.4, y = 0, x = 5, y = 5, x = 5, y = 8, x = 8, y = 8

  **Coffee with 2 doses of sugar**

  - Transition sequence: press?, 1.5, press?, 2.7, press?, 0.8, cup!, 3, coffee!
  - States: x = 0, y = 0, x = 0, y = 1.5, x = 0, y = 1.5, x = 0, y = 4.2, x = 0, y = 4.2, x = 0, y = 5, x = 0, y = 5, x = 0, y = 8, x = 0, y = 8
Parametric timed automaton (PTA)

- Timed automaton (sets of locations, actions and clocks)

![Diagram of a parametric timed automaton]

- Parameters: $y \leq 5$
- Actions: $x := 0$, $y := 0$
- Transitions:
  - From state 1 to state 2: $y = 8$, coffee!
  - From state 2 to state 3: $y = 5$, cup!

\[ x \geq 1 \]
\[ x := 0 \]
\[ y \leq 8 \]
Parametric timed automaton (PTA)

- Timed automaton (sets of locations, actions and clocks) augmented with a set $P$ of parameters [Alur et al., 1993]
  - Unknown constants used in guards and invariants

\[
\begin{align*}
  y &= p_3 \\
  \text{coffee!}
\end{align*}
\]

\[
\begin{align*}
  x &= 0 \\
  y &= 0 \\
  \text{press?}
\end{align*}
\]

\[
\begin{align*}
  y &\leq p_2 \\
  x &\geq p_1 \\
  \text{cup!}
\end{align*}
\]

\[
\begin{align*}
  x &= 0 \\
  \text{press?}
\end{align*}
\]
Valuation of a PTA

Given a PTA $\mathcal{A}$ and a parameter valuation $\nu$, we denote by $\nu(\mathcal{A})$ the (non-parametric) timed automaton where all parameters are valuated by $\nu$. 
Valuation of a PTA

- Given a PTA $\mathcal{A}$ and a parameter valuation $\nu$, we denote by $\nu(\mathcal{A})$ the (non-parametric) timed automaton where all parameters are valuated by $\nu$

\[
\nu = \begin{cases} 
  p_1 & \rightarrow 1 \\
  p_2 & \rightarrow 5 \\
  p_3 & \rightarrow 8 
\end{cases}
\]
Outline

1 Parametric Timed Automata

2 Deadlock-Checking

3 Backward Under-approximation

4 Experiments

5 Conclusion and Perspectives
Objective

Problem

Given a PTA $A$, synthesize valuations $v$ for which $v(A)$ is deadlock-free.
Objective

Problem

Given a PTA $A$, synthesize valuations $v$ for which $v(A)$ is **deadlock-free**.

Deadlock: state in which no **discrete action** may be taken in some state, even after elapsing some time.
**Objective**

**Problem**

Given a PTA $A$, synthesize valuations $\nu$ for which $\nu(A)$ is **deadlock-free**.

Deadlock: state in which no **discrete action** may be taken in some state, even after elapsing some time.

The mere emptiness of this valuation set is **undecidable** for PTAs, and even for subclasses known for other decidability results

[André and Lime, 2016]
Examples

Deadlock-free if \( p_1 + 5 \geq p_2 \leq 10 \)

Deadlock-free for no valuation

É. André (U. Paris 13)
Examples

Deadlock-free if
\[ p_1 + 5 \geq p_2 \leq 10 \]
Examples

Deadlock-free if \( p_1 + 5 \geq p_2 \leq 10 \)

Deadlock-free for no valuation
Symbolic semantics of PTAs

- Symbolic state $s = (l, Z)$: location + convex polyhedron constraining both clocks and parameters

  [André et al., 2009, Jovanović et al., 2015]

- Convex polyhedra obtained have a special form called parametric zone

  [Hune et al., 2002]

\[ Z_0 = \left\{ \begin{array}{l}
  x = y \\
  0 \leq y \leq p_1 \\
  p_1, p_2 \geq 0 \\
\end{array} \right. \quad \begin{array}{l}
  Z_1 = \left\{ \begin{array}{l}
  p_2 \leq x - y \leq p_1 \\
  p_2 \leq p_1 \\
  x, y, p_1, p_2 \geq 0 \\
\end{array} \right. \]
Symbolic semantics of PTAs

- Symbolic state $s = (l, Z)$: location + convex polyhedron constraining both clocks and parameters
  
  [André et al., 2009, Jovanović et al., 2015]

- Convex polyhedra obtained have a special form called parametric zone
  
  [Hune et al., 2002]

$$
Z_0 = \begin{cases} 
  x = y \\
  0 \leq y \leq p_1 \\
  p_1, p_2 \geq 0
\end{cases} 
$$

$$
Z_1 = \begin{cases} 
  p_2 \leq x - y \leq p_1 \\
  (p_2 \leq p_1) \\
  x, y, p_1, p_2 \geq 0
\end{cases} 
$$

Note: there is a potentially infinite number of symbolic states
A semi-algorithm for deadlock-existence

\[
\text{DSynth}(s, \text{Passed}) = \begin{cases} 
\text{return nothing if state met before } s \downarrow \text{Passed} \\
\uplus s' \in \text{Succ}(s) \text{\, \text{DSynth}(s', \text{Passed} \cup \{s\})} \\
\cup s \in \text{Succ}(s) \text{\, pass the guard } \bigwedge g(s', s) \downarrow \text{Passed} \\
\end{cases}
\]
A semi-algorithm for deadlock-existence

\[
\text{DSynth}(s, \text{Passed}) = \begin{cases} 
\text{return nothing if state met before} \\
\end{cases}
\]
A semi-algorithm for deadlock-existence

\[
DSynth(s, \text{Passed}) = \begin{cases} 
\text{return nothing if state met before} \\
\perp \text{ if } s \in \text{Passed}
\end{cases}
\]

- return nothing if state met before
- \( \perp \) if \( s \in \text{Passed} \)
A semi-algorithm for deadlock-existence

\[ DSynth(s, Passed) = \begin{cases} 
\text{return nothing if state met before} \\
\quad \downarrow \text{ if } s \in Passed \\
\text{recursive call on the successors} \\
\text{otherwise:} \\
\text{valuations for which one may get stuck in } s
\end{cases} \]
A semi-algorithm for deadlock-existence

\[ \text{DSynth}(s, \text{Passed}) = \begin{cases} 
\text{return nothing if state met before} \\
\perp \text{ if } s \in \text{Passed} \\
\text{recursive call on the successors} \\
\text{otherwise: } \left( \bigcup_{s' \in \text{Succ}(s)} \text{DSynth}(s', \text{Passed} \cup \{s\}) \right) \\
\text{valuations for which one may get stuck in } s
\end{cases} \]
A semi-algorithm for deadlock-existence

\[
\text{DSynth}(s, \text{Passed}) = \begin{cases} 
\text{return nothing if state met before} \\
\begin{cases} 
\bot & \text{if } s \in \text{Passed} \\
\text{recursive call on the successors} \\
\text{otherwise: } \bigcup_{s' \in \text{Succ}(s)} \text{DSynth}(s', \text{Passed} \cup \{s\}) \\
\cup \text{valuations for which one may get stuck in } s
\end{cases}
\end{cases}
\]
A semi-algorithm for deadlock-existence

$$DSynth(s, \text{Passed}) = \begin{cases} 
\text{return nothing if state met before} \\
\bot \text{ if } s \in \text{Passed} \\
\text{recursive call on the successors} \\
\text{otherwise: } (\bigcup_{s' \in \text{Succ}(s)} DSynth(s', \text{Passed} \cup \{s\})) \\
\cup (s_C \setminus (\bigcup_{s' \in \text{Succ}(s)} (s_C \land g(s, s') \land s_C \downarrow \text{P})))) \downarrow \text{P} \\
\text{valuations for which one may get stuck in } s
\end{cases}$$
A semi-algorithm for deadlock-freeness

$$\text{PDFC}(\mathcal{A}) = \neg \text{DSynth}(s_0^\mathcal{A}, \emptyset)$$
A semi-algorithm for deadlock-freeness

\[
\text{PDFC}(\mathcal{A}) = \neg \text{DSynth}(s_0^\mathcal{A}, \emptyset)
\]

Semi-algorithm: result \textbf{correct} whenever it terminates (which is not guaranteed – undecidable problem)
Outline

1. Parametric Timed Automata
2. Deadlock-Checking
3. Backward Under-approximation
4. Experiments
5. Conclusion and Perspectives
A conservative under-approximation BwUS

Idea:

1. Generate a partial state space
2. Consider unknown states as unsafe (deadlocked)
3. Recompute the deadlock-freeness constraint in a backward manner until fixpoint

\[ S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_4 \rightarrow \times \]

unexplored successors
A conservative under-approximation \textbf{BwUS}

Idea:

1. Generate a \textit{partial} state space
2. Consider unknown states as unsafe (deadlocked)
3. Recompute the deadlock-freeness constraint in a \textit{backward manner} until fixpoint

\[
\begin{array}{c}
S_0 \rightarrow S_1 \\
S_1 \rightarrow S_2 \\
S_2 \rightarrow S_4 \\
S_0 \rightarrow S_1 \rightarrow S_3 \rightarrow S_4
\end{array}
\]
A conservative under-approximation BwUS

Idea:

1. Generate a partial state space
2. Consider unknown states as unsafe (deadlocked)
3. Recompute the deadlock-freeness constraint in a backward manner until fixpoint
A conservative under-approximation \textbf{BwUS}

Idea:

1. Generate a \textit{partial} state space
2. Consider unknown states as unsafe (deadlocked)
3. Recompute the deadlock-freeness constraint in a \textit{backward} manner until fixpoint

\begin{itemize}
    \item \textbf{S0}
    \item \textbf{S1}
    \item \textbf{S2}
    \item \textbf{S3}
    \item \textbf{S4}
\end{itemize}
A conservative under-approximation \textbf{BwUS}

Idea:

1. Generate a \textit{partial} state space
2. Consider unknown states as unsafe (deadlocked)
3. Recompute the deadlock-freeness constraint in a \textit{backward} manner until fixpoint
Outline

1. Parametric Timed Automata
2. Deadlock-Checking
3. Backward Under-approximation
4. Experiments
5. Conclusion and Perspectives
Implementation in IMITATOR

- A tool for modeling and verifying real-time systems with unknown constants modeled with parametric timed automata
  - Communication through (strong) broadcast synchronization
  - Integer-valued discrete variables
  - Stopwatches, to model schedulability problems with preemption

Under continuous development since 2008 [André et al., 2012]

Free and open source software: Available under the GNU-GPL license

www.imitator.fr
### Summary of the experiments

| Case study                        | \( |A| \) | | \( |X| \) | | \( |P| \) | | States | PDFC | BwUS | \( K \) | Soundness                  |
|----------------------------------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|------------------|
| Fig. 1a                          | 1      | 1      | 2      | 3      | 0.012  | -      |        | nncc   | exact            |
| Fig. 1b                          | 1      | 1      | 1      | 2      | 0.005  | -      |        | ⊥      | exact            |
| and–or circuit                   | 4      | 4      | 4      | 5,265  | TO     | 171    | [nncc\(^-\), nncc\(^+\)] | under/over-app |
| coffee machine 1                 | 1      | 2      | 3      | 9,042  | TO     | 8.4    | [nncc\(^-\), nncc\(^+\)] | under/over-app |
| coffee machine 2                 | 2      | 3      | 3      | 51     |        | 0.198  | -      | nncc   | exact            |
| CSMA/CD protocol                 | 3      | 3      | 3      | 38     | 0.105  | -      |        | ⊥      | exact            |
| flip-flop circuit                | 6      | 5      | 2      | 20     | 0.093  | -      |        | ⊥      | exact            |
| nuclear plant                    | 1      | 2      | 4      | 13     | 0.014  | -      |        | nncc   | exact            |
| RCP protocol                     | 5      | 6      | 5      | 2,091  | 10.63  | -      |        | ⊥      | exact            |
| SIMOP                            | 5      | 8      | 2      | 22,894 | TO     | 121    | nncc   | over-app                      |
| Train controller                 | 1      | 2      | 3      | 11     | 0.025  | -      |        | nncc   | exact            |
| WFAS                             | 3      | 4      | 2      | 14,614 | TO     | 69.1   | [nncc\(^-\), nncc\(^+\)] | under/over-app |
Outline

1. Parametric Timed Automata
2. Deadlock-Checking
3. Backward Under-approximation
4. Experiments
5. Conclusion and Perspectives
Conclusion and perspectives

Conclusion

- Semi-algorithm to guarantee the absence of deadlocks in timed automata with parametric timing constants
- Under-approximation with guaranteed termination

Perspectives

- Theory: decidable subclasses for deadlock-checking?
- Practice: efficient computation (multi-core)
- Reusability: Apply the idea behind BwUS to other algorithms
Bibliography
References I

A theory of timed automata.

Parametric real-time reasoning.
In _STOC_, pages 592–601. ACM.

An inverse method for parametric timed automata.

IMITATOR 2.5: A tool for analyzing robustness in scheduling problems.

Liveness in L/U-parametric timed automata.
Submitted.

Symbolic model checking for real-time systems.

Licensing
Source of the graphics used I

Title: Smiley green alien big eyes (aaah)
Author: LadyofHats
Source: https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg
License: public domain

Title: Smiley green alien big eyes (cry)
Author: LadyofHats
Source: https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg
License: public domain
License of this document

This presentation can be published, reused and modified under the terms of the license Creative Commons Attribution-ShareAlike 4.0 Unported (CC BY-SA 4.0)

(\LaTeX source available on demand)

Author: Étienne André

https://creativecommons.org/licenses/by-sa/4.0/