Verifying real-time systems under uncertainty

Étienne André

LIPN, Université Paris 13, CNRS, France

Joint work with Camille Coti, Giuseppe Lipari, Nguyên Hoàng Gia, Sun Youcheng
Reachability in timed systems
Reachability in timed systems
Reachability in timed systems
Reachability in timed systems
Reachability in timed systems
Reachability in timed systems
Liveness in timed systems
Liveness in timed systems
Liveness in timed systems
Liveness in timed systems
Context: Critical systems

- Need for early bug detection
  - Bugs discovered when final testing: expensive
  - Need for a thorough modeling and verification phase
Timed model checking (1/2)

A model of the system

A property to be satisfied

\[ y = \text{delay} \]
\[ x := 0 \]
\[ x < \text{period} \]
Timed model checking (1/2)

A model of the system

A property to be satisfied

Question: does the model of the system satisfy the property?
Timed model checking (1/2)

A model of the system

A property to be satisfied

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

Yes

No

Counterexample
Timed model checking (2/2)

- Timed systems are characterized by a set of timing constants
  - “The packet transmission lasts for 50 ms”
  - “The sensor reads the value every 10 s”

- Powerful model checking tools, e.g.:
  - UPPAAL
  - PAT

[Larsen et al., 1997]
[Sun et al., 2009]
Beyond timed model checking: parameter synthesis

- Verification for **one** set of constants does not usually guarantee the correctness for other values

- Challenges
  - Numerous verifications: is the system correct for any value within [40; 60]?
  - Optimization: until what value can we increase 10?
  - Robustness [Markey, 2011]: What happens if 50 is implemented with 49.99?
  - System incompletely specified: Can I verify my system even if I don’t know the period value with full certainty?
Beyond timed model checking: parameter synthesis

- Verification for one set of constants does not usually guarantee the correctness for other values

- Challenges
  - Numerous verifications: is the system correct for any value within $[40; 60]$?
  - Optimization: until what value can we increase 10?
  - Robustness [Markey, 2011]: What happens if 50 is implemented with 49.99?
  - System incompletely specified: Can I verify my system even if I don’t know the period value with full certainty?

- Parameter synthesis
  - Consider that timing constants are unknown constants (parameters)
timed model checking

A model of the system

A property to be satisfied

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

Yes

No

Counterexample
Parametric 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 Modeling and Verifying Real-Time Systems
3 Verifying a Real-time System under Uncertainty
4 Distributed Verification of Distributed Systems
5 Conclusion and Perspectives
Outline

1 Parametric Timed Automata

2 Modeling and Verifying Real-Time Systems

3 Verifying a Real-time System under Uncertainty

4 Distributed Verification of Distributed Systems

5 Conclusion and Perspectives
Timed automaton (TA)

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

- Finite state automaton (sets of locations and actions)

\[
\begin{align*}
  x &= 0 \\
  y &= 0 \\
  y &= 5 \\
  x &\geq 1
\end{align*}
\]
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
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
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

```latex
\begin{align*}
y &= 8 \\
\text{coffee!}
\end{align*}
```

```latex
\begin{align*}
\text{press?} \\
y &\leq 5 \\
x &\geq 1 \\
\text{press?} \\
y &= 5 \\
\text{cup!}
\end{align*}
```

```latex
\begin{align*}
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
  - Clock reset: some of the clocks can be set to 0 at each transition

```
\begin{align*}
  & x := 0 \\
  & y := 0 \\
  & y \leq 5 \\
  & x \geq 1 \\
  & \text{press?} \\
  & x := 0 \\
  & y = 5 \\
  & \text{cup!} \\
  & y \leq 8
\end{align*}
```
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

\begin{align*}
\text{x} &= 0 \\
\text{y} &= 0
\end{align*}
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>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>15.4</td>
<td>15.4</td>
</tr>
</tbody>
</table>

\[ y = 8 \]
\[ \text{coffee!} \]
\[ y \leq 5 \]
\[ x \geq 1 \]
\[ \text{cup!} \]

\[ x := 0 \]
\[ y := 0 \]
\[ x := 0 \]
\[ y := 0 \]
Examples of concrete runs

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td>15.4</td>
</tr>
<tr>
<td>y</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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>15.4</th>
<th>press?</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td>15.4</td>
<td>0</td>
<td>5</td>
</tr>
<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

```
x  y  15.4  0  5  5
0  0  15.4  0  5  5
```
Examples of concrete runs

Possible concrete runs for the coffee machine

Coffee with no sugar

\[
\begin{array}{ccccccc}
x & 0 & 15.4 & 0 & 5 & 5 & 8 \\
y & 0 & 15.4 & 0 & 5 & 5 & 8 \\
\end{array}
\]
Examples of concrete runs

Possible concrete runs for the coffee machine

- Coffee with no sugar
Examples of concrete runs

```
<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15.4</td>
<td>15.4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>
```

- **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**

  | x   | 0 | 15.4 | 0 | 5 | 5 | 8 | coffee! |
  | 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

- Coffee with 2 doses of sugar
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 1: press? $\rightarrow y \leq 5$
  - transition 2: $x \geq 1$ cup!
  - transition 3: $y = 5$
  - transition 4: press? $\rightarrow y \leq 8$

  - states:
    - green: initial state
    - blue: press? state
    - red: coffee! state

  - values:
    - $x = 0$
    - $y = 5$
    - $y = 8$

  - transitions:
    - press? $\rightarrow y \leq 5$
    - $x \geq 1$ cup!
    - press? $\rightarrow y \leq 8$

- **Coffee with 2 doses of sugar**

  - initial state: $x := 0$, $y := 0$
  - transition 1: press? $\rightarrow 1.5$
  - transition 2: press? $\rightarrow 2.7$
  - transition 3: press? $\rightarrow 0$

  - values:
    - $x = 0$
    - $y = 4.2$

  - transitions:
    - press? $\rightarrow 1.5$
    - press? $\rightarrow 2.7$
    - press? $\rightarrow 0$
Examples of concrete runs

Possible concrete runs for the coffee machine

- Coffee with no sugar
  
<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

- Coffee with 2 doses of sugar
  
<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>1.5</td>
<td>4.2</td>
</tr>
<tr>
<td>4.2</td>
<td>4.2</td>
</tr>
<tr>
<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></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td>15.4</td>
<td></td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

- Coffee with 2 doses of sugar

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>press?</th>
<th>1.5</th>
<th>press?</th>
<th>2.7</th>
<th>press?</th>
<th>0.8</th>
<th>cup!</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.5</td>
<td>0</td>
<td>2.7</td>
<td>0</td>
<td>0.8</td>
<td>0.8</td>
</tr>
</tbody>
</table>
Examples of concrete runs

- Possible concrete runs for the coffee machine
  - Coffee with no sugar
    - Initial state: x = 0, y = 0
    - Transition: press? to x := 0
    - Transition: x ≥ 1 to press?
    - Transition: x := 0 to y ≥ 5
    - Transition: y ≤ 5 to y := 0
    - Transition: y ≤ 8 to y = 5
    - Transition: y = 5 to coffee!
  
  - Coffee with 2 doses of sugar
    - Initial state: x = 0, y = 0
    - Transition: press? to x := 0
    - Transition: press? to y ≥ 1
    - Transition: y := 0 to x ≥ 2
    - Transition: x = 2.7 to press?
    - Transition: press? to y := 0
    - Transition: y ≤ 8 to y = 5
    - Transition: y = 5 to coffee!

\[ x := 0 \]
\[ y := 0 \]
\[ x := 0 \]
\[ y := 8 \]
\[ x := 0 \]
\[ y := 5 \]
Examples of concrete runs

Possible concrete runs for the coffee machine

- **Coffee with no sugar**

  1. **x = 0**: 0 \rightarrow 15.4 \rightarrow press? \rightarrow 5 \rightarrow cup! \rightarrow 3 \rightarrow coffee!
  2. **y = 0**: 0 \rightarrow 15.4 \rightarrow press? \rightarrow 5 \rightarrow cup! \rightarrow 3 \rightarrow coffee!

- **Coffee with 2 doses of sugar**

  1. **x = 0**: 0 \rightarrow 0 \rightarrow 1.5 \rightarrow press? \rightarrow 2.7 \rightarrow press? \rightarrow 0.8 \rightarrow cup! \rightarrow 3 \rightarrow coffee!
  2. **y = 0**: 0 \rightarrow 0 \rightarrow 1.5 \rightarrow press? \rightarrow 2.7 \rightarrow press? \rightarrow 0.8 \rightarrow cup! \rightarrow 3 \rightarrow coffee!
Parametric timed automaton (PTA)

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

\[
\begin{align*}
\text{press?} & \quad x := 0 \\
y & \leq 5 \\
\quad y = 8 \\
\text{coffee!} \\
\text{press?} & \quad x := 0 \\
x & \geq 1 \\
\quad y = 5 \\
\text{cup!} \\
y & \leq 8
\end{align*}
\]
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

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

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

1. Parametric Timed Automata
2. Modeling and Verifying Real-Time Systems
3. Verifying a Real-time System under Uncertainty
4. Distributed Verification of Distributed Systems
5. Conclusion and Perspectives
Context: Hard real-time embedded systems

- Modern hard real-time embedded systems are distributed in nature
- Many of them have critical timing requirements:
  - automotive systems (modern cars have 10-20 embedded boards connected by one or more CAN bus)
  - avionics systems (several distributed control boards connected by one or more dedicated networks)

- To analyze the schedulability of such systems, it is very important to estimate the (worst-case) computation times of the tasks
- Estimating WCET is very difficult in modern architectures
Many real-time applications can be modeled as pipelines (also called transactions) of tasks.

- Executed on a distributed (or multicore) system
- Activated cyclically (periodic or sporadic)
Model

- A set of pipelines $\{P^{(1)}, \ldots, P^{(p)}\}$ distributed over $m$ nodes
- Each pipeline $P^{(i)}$ is a chain of $n_i$ tasks $\{\tau_{i,1}, \ldots, \tau_{i,n_i}\}$
- Pipeline $P^{(i)}$ has an end-to-end (E2E) deadline $D_i$ and period $T_i$

Scheduler Problem: Guarantee that all pipelines complete before their E2E deadlines
Activations, jitter, deadline

An example

<table>
<thead>
<tr>
<th>Task</th>
<th>Per.</th>
<th>E2E</th>
<th>Comp.</th>
<th>Resp.</th>
<th>Jitter</th>
<th>prio</th>
</tr>
</thead>
<tbody>
<tr>
<td>τ₁,₁</td>
<td>15</td>
<td></td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>HI</td>
</tr>
<tr>
<td>τ₁,₂</td>
<td>-</td>
<td></td>
<td>3</td>
<td>8</td>
<td>0-2</td>
<td>LO</td>
</tr>
<tr>
<td>τ₁,₃</td>
<td>-</td>
<td>15</td>
<td>2</td>
<td>13</td>
<td>3</td>
<td>LO</td>
</tr>
<tr>
<td>τ₂,₁</td>
<td>12</td>
<td></td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>HI</td>
</tr>
<tr>
<td>τ₂,₂</td>
<td>-</td>
<td>12</td>
<td>6</td>
<td>?</td>
<td>0-1</td>
<td>ME</td>
</tr>
</tbody>
</table>

Node A

Node B

τ₁,₁  →  τ₁,₂
τ₁,₁  →  τ₁,₃
τ₂,₂  →  τ₂,₁

É. André (Université Paris 13)  Verifying RTS under uncertainty  12 mai 2017  19 / 52
Activations, jitter, deadline

- An example

<table>
<thead>
<tr>
<th>Task</th>
<th>Per.</th>
<th>E2E</th>
<th>Comp.</th>
<th>Resp.</th>
<th>Jitter</th>
<th>prio</th>
</tr>
</thead>
<tbody>
<tr>
<td>τ_{1,1}</td>
<td>15</td>
<td></td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>HI</td>
</tr>
<tr>
<td>τ_{1,2}</td>
<td>-</td>
<td></td>
<td>3</td>
<td>8</td>
<td>0-2</td>
<td>LO</td>
</tr>
<tr>
<td>τ_{1,3}</td>
<td>-</td>
<td>15</td>
<td>2</td>
<td>13</td>
<td>3</td>
<td>LO</td>
</tr>
<tr>
<td>τ_{2,1}</td>
<td>12</td>
<td></td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>HI</td>
</tr>
<tr>
<td>τ_{2,2}</td>
<td>-</td>
<td>12</td>
<td>6</td>
<td>?</td>
<td>0-1</td>
<td>ME</td>
</tr>
</tbody>
</table>

Node A

- τ_{1,1}
- τ_{1,2}
- τ_{1,3}

Node B

- τ_{2,1}
- τ_{2,2}
Activations, jitter, deadline

- An example

<table>
<thead>
<tr>
<th>Task</th>
<th>Per.</th>
<th>E2E</th>
<th>Comp.</th>
<th>Resp.</th>
<th>Jitter</th>
<th>prio</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_{1,1}$</td>
<td>15</td>
<td></td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>HI</td>
</tr>
<tr>
<td>$\tau_{1,2}$</td>
<td>-</td>
<td>15</td>
<td>2</td>
<td>13</td>
<td>3</td>
<td>LO</td>
</tr>
<tr>
<td>$\tau_{1,3}$</td>
<td>-</td>
<td>15</td>
<td>6</td>
<td>14</td>
<td>0</td>
<td>LO</td>
</tr>
<tr>
<td>$\tau_{2,1}$</td>
<td>12</td>
<td></td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>HI</td>
</tr>
<tr>
<td>$\tau_{2,2}$</td>
<td>-</td>
<td>12</td>
<td>6</td>
<td>14</td>
<td>0</td>
<td>ME</td>
</tr>
</tbody>
</table>

Node A: $\tau_{1,1}$, $\tau_{1,2}$, $\tau_{1,3}$, $\tau_{2,2}$, $\tau_{2,1}$

Node B: $\tau_{1,1}$, $\tau_{1,2}$, $\tau_{1,3}$, $\tau_{2,2}$, $\tau_{2,1}$
Enriching the model with parameters

- A task is identified by three parameters:
  - $C_i$ is the **worst-case computation time** (or worst-case transmission time, in case it models a message)
  - $R_i$ is the **task worst-case response time**, i.e., the worst case finishing time of any task instance relative to the activation of its pipeline.
  - $J_i$ is the task **worst-case activation jitter**, i.e., the greatest time since its activation that a task must wait for all preceding tasks to complete their execution

- A parameter of major interest is the **computation time**
Modeling a task / pipeline

\[ \begin{align*}
\tau_1 & \text{ waiting urgent} \\
\tau_2 & \text{ waiting urgent} \\
\tau_1 & \text{ released} \\
\tau_2 & \text{ released}
\end{align*} \]

\[ \begin{align*}
\tau_1 & \text{ release} \\
\tau_1 & \text{ completed} \\
\tau_2 & \text{ release}
\end{align*} \]

\[ \begin{align*}
\mathcal{P}_1 & \text{ complete} \\
\mathcal{P}_1 & \text{ restart} \\
\mathcal{P}_1 & \text{ complete}
\end{align*} \]

\[ \begin{align*}
\chi_{\mathcal{P}_1} & = T_1 \\
\chi_{\mathcal{P}_1} & := 0 \\
\chi_{\mathcal{P}_1} & \leq T_1
\end{align*} \]
Modeling and Verifying Real-Time Systems

Modeling Real-time Systems under Uncertainty

Modeling the fixed priority scheduler (preemptive)

Actually a PTA extended with stopwatches [Sun et al., 2013]
an extension of stopwatch automata [Adbeddaïm and Maler, 2002]
Parametric verification of real-time systems

Many problems can be reduced to parametric reachability (EFsynth): find parameter valuations for which a given state is (un)reachable.

- This problem is undecidable for PTAs and many subclasses.

- But we can still compute part (and often all) of the solution.

Interesting problems:

- Find parameter valuations for which no deadline violation occurs (i.e., for which the system is schedulable).
- Compute the worst-case computation time.
- Find the parametric WCET when the jitter is unknown.
**IMITATOR**

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

**Verification**

- Computation of the symbolic state space
- (non-Zeno) parametric model checking (using a subset of TCTL)
- Language and trace preservation, and robustness analysis
- Parametric deadlock-freeness checking
- Behavioral cartography
IMITATOR

Under continuous development since 2008

[André et al., 2012]

A library of benchmarks

- Communication protocols
- Schedulability problems
- Asynchronous circuits
- ...and more

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

Under continuous development since 2008

A library of benchmarks
- Communication protocols
- Schedulability problems
- Asynchronous circuits
- ... and more

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

Try it!

www.imitator.fr
Some success stories

- Modeled and verified an asynchronous memory circuit by ST-Microelectronics
  - Project ANR Valmem

- Parametric schedulability analysis of a prospective architecture for the flight control system of the next generation of spacecrafts designed at ASTRIUM Space Transportation [Fribourg et al., 2012]

- Formal timing analysis of music scores [Fanchon and Jacquemard, 2013]

- Solution to a challenge related to a distributed video processing system by Thales
Outline

1. Parametric Timed Automata
2. Modeling and Verifying Real-Time Systems
3. Verifying a Real-time System under Uncertainty
4. Distributed Verification of Distributed Systems
5. Conclusion and Perspectives
The FMTV 2015 Challenge (1/2)
Challenge by Thales proposed during the WATERS 2014 workshop
Solutions presented at WATERS 2015

System: an unmanned aerial video system with uncertain periods
- Period constant but with a small uncertainty (typically 0.01 %)
- Not a jitter!
The FMTV 2015 Challenge (2/2)

Goal

Compute the end-to-end BCET and WCET times for a buffer size of $n = 1$ and $n = 3$
The FMTV 2015 Challenge (2/2)

Goal

Compute the end-to-end BCET and WCET times for a buffer size of $n = 1$ and $n = 3$

😊 Not a typical parameter synthesis problem?

- No parameters in the specification
The FMTV 2015 Challenge (2/2)

Goal

Compute the end-to-end BCET and WCET times for a buffer size of \( n = 1 \) and \( n = 3 \)

🤔 Not a typical parameter synthesis problem?
- No parameters in the specification

😊 A typical parameter synthesis problem
- The end-to-end time can be set as a parameter... to be synthesized
- The uncertain period is typically a parameter (with some constraint, e.g., \( P1 \in [40 - 0.004, 40 + 0.004] \))
Methodology

1. Propose a PTA model with parameters for uncertain periods and the end-to-end time.
Methodology

1. Propose a PTA model with parameters for uncertain periods and the end-to-end time
2. Add a specific location corresponding to the correct transmission of the frame
Methodology

1. Propose a PTA model with parameters for uncertain periods and the end-to-end time
2. Add a specific location corresponding to the correct transmission of the frame
3. Run the reachability synthesis algorithm EFsynth (implemented in IMITATOR) w.r.t. that location

Note: not eliminating parameters allows one to know for which values of the periods the best / worst case execution times are obtained.
Methodology

1. Propose a PTA model with parameters for uncertain periods and the end-to-end time
2. Add a specific location corresponding to the correct transmission of the frame
3. Run the reachability synthesis algorithm EFsynth (implemented in IMITATOR) w.r.t. that location
4. Gather all constraints (in as many dimensions as uncertain periods + the end-to-end time)
Methodology

1. Propose a PTA model with parameters for uncertain periods and the end-to-end time
2. Add a specific location corresponding to the correct transmission of the frame
3. Run the reachability synthesis algorithm EFsynth (implemented in IMITATOR) w.r.t. that location
4. Gather all constraints (in as many dimensions as uncertain periods + the end-to-end time)
5. Eliminate all parameters but the end-to-end time

Note: not eliminating parameters allows one to know for which values of the periods the best / worst case execution times are obtained.
Methodology

1. Propose a PTA model with parameters for uncertain periods and the end-to-end time
2. Add a specific location corresponding to the correct transmission of the frame
3. Run the reachability synthesis algorithm EFsynth (implemented in IMITATOR) w.r.t. that location
4. Gather all constraints (in as many dimensions as uncertain periods + the end-to-end time)
5. Eliminate all parameters but the end-to-end time
6. Exhibit the minimum and the maximum

Note: not eliminating parameters allows one to know for which values of the periods the best / worst case execution times are obtained.
To build the PTA model

- Uncertainties in the system:
  - \( P1 \in [40 - 0.004, 40 + 0.004] \)
  - \( P3 \in \left[ \frac{40}{3} - \frac{1}{150}, \frac{40}{3} + \frac{1}{150} \right] \)
  - \( P4 \in [40 - 0.004, 40 + 0.004] \)
To build the PTA model

- Uncertainties in the system:
  - $P_1 \in [40 - 0.004, 40 + 0.004]$  
  - $P_3 \in \left[\frac{40}{3} - \frac{1}{150}, \frac{40}{3} + \frac{1}{150}\right]$  
  - $P_4 \in [40 - 0.004, 40 + 0.004]$  

- Parameters:
  - $P_1_{\text{uncertain}}$  
  - $P_3_{\text{uncertain}}$  
  - $P_4_{\text{uncertain}}$
To build the PTA model

- Uncertainties in the system:
  - $P_1 \in [40 - 0.004, 40 + 0.004]$  
  - $P_3 \in [\frac{40}{3} - \frac{1}{150}, \frac{40}{3} + \frac{1}{150}]$  
  - $P_4 \in [40 - 0.004, 40 + 0.004]$  

- Parameters:
  - $P_1_{\text{uncertain}}$  
  - $P_3_{\text{uncertain}}$  
  - $P_4_{\text{uncertain}}$  

- The end-to-end latency (another parameter): $E_{2E}$
To build the PTA model

- **Uncertainties in the system:**
  - P1 ∈ [40 - 0.004, 40 + 0.004]
  - P3 ∈ [\(\frac{40}{3} - \frac{1}{150}, \frac{40}{3} + \frac{1}{150}\)]
  - P4 ∈ [40 - 0.004, 40 + 0.004]

- **Parameters:**
  - P1_uncertain
  - P3_uncertain
  - P4_uncertain

- **The end-to-end latency (another parameter):** E2E

- **Others:**
  - the register between task 2 and task 3: discrete variable reg_{2,3}
  - the buffer between task 3 and task 4: n = 1 or n = 3
Simplification

- T1 and T2 are synchronised; T1, T3 and T4 are asynchronous
- (exact modeling of the system behaviour is too heavy)
Simplification

- T1 and T2 are synchronised; T1, T3 and T4 are asynchronised
  - (exact modeling of the system behaviour is too heavy)

- We choose a single arbitrary frame, called the target one

- We assume the system is initially in an arbitrary status
  - This is our only uncertain assumption (in other words, can the periods deviate from each other so as to yield any arbitrary deviation?)
The initialization automaton

\[ \text{ckT1T2} = \text{WCET}_1 \]

\[ \text{camera0} \]
The initialization automaton

\[ \text{camera}_0 \quad \text{buffer}_{3,4} := 0 \]
\[ \quad \text{highest}_{3,4} := 0 \]

\[ \text{camera}_1 \quad \text{buffer}_{3,4} := 1 \]
\[ \quad \text{highest}_{3,4} := 1 \]
The initialization automaton

\[ \text{ckT1T2} = \text{WCET}_1 \]

- \( \text{buffer}_{3,4} := 0 \)  
  \( \text{highest}_{3,4} := 0 \)
- \( \text{buffer}_{3,4} := 1 \)  
  \( \text{highest}_{3,4} := 1 \)
- \( \text{frame}_{\text{in-}3} := 0 \)  
  \( \text{frame}_{\text{in-}3} := 2 \)
The initialization automaton

\[ \text{camera0} \quad \text{ckT1T2} = \text{WCET}_1 \]

- buffer\(_{3,4}\) := 0
- highest\(_{3,4}\) := 0

\[ \text{camera1} \quad \text{ckT1T2} = \text{WCET}_1 \]

- buffer\(_{3,4}\) := 1
- highest\(_{3,4}\) := 1
- frame\(_{in\_3}\) := 0
- frame\(_{in\_3}\) := 2

\[ \text{camera2} \quad \text{ckT1T2} = \text{WCET}_1 \]

- reg\(_{2,3}\) := 0
- reg\(_{2,3}\) := 3

\[ \text{camera3} \quad \text{ckT1T2} = \text{WCET}_1 \]
The initialization automaton

\[
\begin{align*}
&\text{camera0} : \quad \text{buffer}_{3,4} := 0 \\
&\quad \text{highest}_{3,4} := 0 \\
&\quad \text{frame}_\text{in}_3 := 0 \\
&\text{camera1} : \quad \text{buffer}_{3,4} := 1 \\
&\quad \text{highest}_{3,4} := 1 \\
&\quad \text{frame}_\text{in}_3 := 2 \\
&\text{camera2} : \quad \text{reg}_{2,3} := 0 \\
&\quad \text{reg}_{2,3} := 3 \\
&\text{start} \quad \text{T}_1\text{T}_2 : \quad \text{WCET}_1 + \text{WCL}_2 \geq \text{ckT}_1\text{T}_2
\end{align*}
\]
The initialization automaton

\[ \text{ckT1T2} = \text{WCET}_1 \]

\[ \begin{align*}
\text{camera0} & \quad \text{buffer}_{3,4} := 0 \\
& \quad \text{highest}_{3,4} := 0
\end{align*} \]

\[ \begin{align*}
\text{camera1} & \quad \text{buffer}_{3,4} := 1 \\
& \quad \text{highest}_{3,4} := 1
\end{align*} \]

\[ \begin{align*}
\text{camera1} & \quad \text{frame}\_\text{in}\_3 := 0 \\
& \quad \text{frame}\_\text{in}\_3 := 2
\end{align*} \]

\[ \begin{align*}
\text{camera2} & \quad \text{reg}_{2,3} := 0 \\
& \quad \text{reg}_{2,3} := 3
\end{align*} \]

\[ \begin{align*}
\text{camera2} & \quad \text{ckT1T2} = \text{WCET}_1 \quad \text{T2done}
\end{align*} \]

\[ \begin{align*}
\text{T1T2} & \quad \text{ckT1T2} \geq \text{WCET}_1 + \text{BCL}_2 \\
& \quad \text{T1T2done}
\end{align*} \]

\[ \begin{align*}
\text{T1T2} & \quad \text{reg}_{2,3} := \text{target}
\end{align*} \]
Task T3

\[ T3_{preinit} \]
Task T3

\[ \text{WCET}_3 \geq \text{ckT}_3 \]

\[ \text{start} \]

\[ \text{frame_in}_3 > \text{highest}_3,4 \]

\[ \text{T3_done} \]

\[ \text{write}_T3() \]

\[ \text{WCET}_3 \geq \text{ckT}_3 \]

\[ \text{frame_in}_3 \]

\[ \text{buer}_3,4 = 0 \]

\[ \text{WCET}_3 \geq \text{ckT}_3 \]

\[ \text{frame_in}_3 \]

\[ \text{buer}_3,4 \geq 0 \]

\[ \text{T3_done} \]

\[ \text{write}_T3() \]

\[ \text{WCET}_3 \geq \text{ckT}_3 \]

\[ \text{frame_in}_3 \]

\[ \text{buer}_3,4 = 0 \]

\[ \text{highest}_3,4 \geq \text{frame_in}_3 \]

\[ \text{T3_done} \]

\[ \text{write}_T3() \]

\[ \text{WCET}_3 \geq \text{ckT}_3 \]

\[ \text{frame_in}_3 \]
Task T3

**T3preinit**

$\text{WCET}_3 \geq c_kT_3$

**T3process**

$\text{WCET}_3 \geq c_kT_3$

**T3wait**

$P_{3\text{-}uncertain} \geq c_kT_3$
**Task T3**

- **T3preinit**
  - $\text{WCET}_3 \geq \text{ckT}_3$
  - $\text{start}$

- **T3process**
  - $\text{WCET}_3 \geq \text{ckT}_3$

- **T3wait**
  - $\text{P3\_uncertain} = \text{ckT}_3$
  - $\text{T3} \_\text{start}$
  - $\text{ckT}_3 := 0$
  - $\text{frame\_in\_3} := \text{reg}_{2,3}$

- $\text{P3\_uncertain} \geq \text{ckT}_3$
Task T3

\[
\begin{align*}
W C E T_3 & \geq ckT_3 \\
\text{start} & \\
W C E T_3 & \geq ckT_3 \\
\text{start} & \\
\end{align*}
\]

\[
\begin{align*}
P_3\_\text{uncertain} & = ckT_3 \\
T_3\_\text{start} & \\
ckT_3 & := 0 \\
\text{frame\_in\_3} & := \text{reg}_{2,3} \\
\end{align*}
\]

\[
\begin{align*}
W C E T_3 & = ckT_3 \\
\wedge \text{buffer}_{3,4} & = 0 \\
\wedge \text{frame\_in\_3} & > \text{highest}_{3,4} \\
T_3\_\text{done} & \\
\text{write\_by\_T3()} & \\
\end{align*}
\]

\[
\begin{align*}
P_3\_\text{uncertain} & \geq ckT_3 \\
\end{align*}
\]
Task T3

The PTA model for $n = 1$

- P3_uncertain = ckT3
- T3_start = 0
- frame_in_3 := reg2,3

WCET$_3$ ≥ ckT3

- WCET$_3$ = ckT3
- buffer$_3,4$ = 0
- frame_in_3 > highest$_3,4$
- T3_done
- write_by_T3()

- WCET$_3$ = ckT3
- buffer$_3,4$ > 0
- T3_done

- P3_uncertain ≥ ckT3
Task T3

\[
\begin{align*}
\text{WCET}_3 & \geq \text{ckT}_3 \\
\text{P}_3\_\text{uncertain} & = \text{ckT}_3 \\
\text{buffer}_{3,4} & = 0 \\
\text{highest}_{3,4} & \geq \text{frame\_in\_3} \\
\text{T}_3\_\text{done} & \\
\text{frame\_in\_3} & := \text{reg}_{2,3}
\end{align*}
\]

\[
\begin{align*}
\text{WCET}_3 & = \text{ckT}_3 \\
\text{buffer}_{3,4} & = 0 \\
\text{frame\_in\_3} & > \text{highest}_{3,4} \\
\text{write\_by\_T}_3() & \\
\text{T}_3\_\text{done} & \geq \text{frame\_in\_3}
\end{align*}
\]

\[
\begin{align*}
\text{P}_3\_\text{uncertain} & \geq \text{ckT}_3
\end{align*}
\]

\[
\begin{align*}
\text{WCET}_3 & = \text{ckT}_3 \\
\text{buffer}_{3,4} & > 0 \\
\text{T}_3\_\text{done} & \\
\text{highest}_{3,4} & \geq \text{frame\_in\_3}
\end{align*}
\]

É. André (Université Paris 13)  Verifying RTS under uncertainty  12 mai 2017  34 / 52
Task T4

\[ P_{4, uncertain} \geq ckT_4 \]
Task T4

\[ P_{4\text{ uncertain}} = ckT4 \]
\[ \land buffer_{3,4} > 0 \]
\[ ckT4 := 0 \]
\[ \text{read\_by\_T4()} \]
Task T4

\[ \text{P4\_uncertain} = \text{ckT4} \]
\[ \quad \land \text{buffer}_{3,4} = 0 \]
\[ \quad \text{ckT4} := 0 \]

\[ \text{P4\_uncertain} = \text{ckT4} \]
\[ \quad \land \text{buffer}_{3,4} > 0 \]
\[ \quad \text{ckT4} := 0 \]
\[ \quad \text{read\_by\_T4()} \]

\[ \text{P4\_uncertain} \geq \text{ckT4} \]

\[ 10 \geq \text{ckT4} \]
Task T4

\[ P_{4\_uncertain} = ckt_4 \land \text{buffer}_{3,4} = 0 \]
\[ \text{ckT}_4 := 0 \]

\[ P_{4\_uncertain} = ckt_4 \land \text{buffer}_{3,4} > 0 \]
\[ \text{ckT}_4 := 0 \]
\[ \text{read\_by\_T}_4() \]

\[ P_{4\_uncertain} \geq \text{ckT}_4 \]
\[ 10 = \text{ckT}_4 \land \text{frame\_in\_4} \neq \text{target} \]
**Task T4**

\[
P_{4\_uncertain} = ckt_4 \\
\land \text{buffer}_{3,4} = 0 \\
\land \text{buffer}_{3,4} > 0 \\
\land \text{buffer}_{3,4} = 0 \\
\land \text{read\_by\_T4}() \\
\land \text{frame\_in\_4} \neq \text{target} \\
\land \text{frame\_in\_4} = \text{target} \\
\land \text{E2E} \\
\land \text{ckT1T2} = 0 \\
\land \text{ckT4} = 0
\]

10 = ckT4 \\
\land \text{frame\_in\_4} = \text{target} \\
\land \text{ckT1T2} = \text{E2E} \\
\land \text{ckT4} := 0
Results

E2E latency results for $n = 1$ and $n = 3$

<table>
<thead>
<tr>
<th></th>
<th>$n = 1$</th>
<th>$n = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>min E2E</td>
<td>63 ms</td>
<td>63 ms</td>
</tr>
<tr>
<td>max E2E</td>
<td>145.008 ms</td>
<td>225.016 ms</td>
</tr>
</tbody>
</table>

Results obtained using IMITATOR in a few seconds
Outline

1. Parametric Timed Automata
2. Modeling and Verifying Real-Time Systems
3. Verifying a Real-time System under Uncertainty
4. Distributed Verification of Distributed Systems
5. Conclusion and Perspectives
Why distributed algorithms?

Algorithms for parameter synthesis for PTA are very costly
- time
- memory

Some reasons:
- expensive operations on polyhedra
  - In IMITATOR: PPL
  
  [Bagnara et al., 2008]

- no known efficient data structure (such as BDDs or DBMs for timed automata)
Why distributed algorithms?

Algorithms for parameter synthesis for PTA are very costly

- time
- memory

Some reasons:

- expensive operations on polyhedra
  - In IMITATOR: PPL
- no known efficient data structure (such as BDDs or DBMs for timed automata)

Idea: benefit from the power of clusters

- Cluster: large set of nodes (computers with their own memory and processor)
- Communication between nodes over a network

[Bagnara et al., 2008]
A first naive approach

Naive approach to distribute EFsynth:
- Each node handles a subpart of the parameter domain
- Each node launches EFsynth on its parameter domain

Drawback: bad performances if the analysis is much more costly in some subdomains than in others
A more elaborate master-worker approach

Workers: run a “hybrid” algorithm

- **PRP**: parametric reachability preservation
- based on (integer) points: generalizes the reachability of the bad state as in the reference valuation (point)
- inspired by both EFsynth (to look for bad valuations) and TPsynth (to only explore a limited part of the symbolic state space, while “imitating” a reference valuation)
  - TPsynth: semi-algorithm for untimed language preservation
- guarantees the coverage of all integer points (but rational-valued points may be missing)

Master: responsible for gathering results and distributing reference valuations (“points”) among workers
**PRP: Case 1**

As long as $l_{bad}$ is not met...

- Explore the symbolic state space
- But do not explore the behaviors not present in $v(A)$!
PRP: Case 1

As long as $l_{bad}$ is not met...

- Explore the symbolic state space
- But do not explore the behaviors not present in $v(A)$!

```
When no successors, and if $l_{bad}$ was never met:
return $\neg \land \cdots \land \neg$
```

Ensures a subset of the behaviors of $v(A)$, and hence guarantees the unreachability of $l_{bad}$.
PRP: Case 1

As long as $l_{bad}$ is not met...

- Explore the symbolic state space
- But do not explore the behaviors not present in $\nu(A)$!
PRP: Case 1

As long as $l_{bad}$ is not met...

- Explore the symbolic state space
- But do not explore the behaviors not present in $\nu(A)$!
PRP: Case 1

As long as $l_{bad}$ is not met...

- Explore the symbolic state space
- But do not explore the behaviors not present in $\nu(A)$!
**PRP: Case 1**

As long as $l_{bad}$ is not met...

- Explore the symbolic state space
- But do not explore the behaviors not present in $\nu(A)$!
PRP: Case 1

As long as $l_{bad}$ is not met...

- Explore the symbolic state space
- But do not explore the behaviors not present in $v(A)$!
PRP: Case 1

As long as $l_{bad}$ is not met...

- Explore the symbolic state space
- But do not explore the behaviors not present in $v(A)$!
PRP: Case 1

As long as \( l_{bad} \) is not met...

- Explore the symbolic state space
- But do not explore the behaviors not present in \( v(A) \)!
PRP: Case 1

As long as $l_{bad}$ is not met...

- Explore the symbolic state space
- But do not explore the behaviors not present in $v(A)$!
**PRP: Case 1**

As long as $l_{bad}$ is not met...

- Explore the symbolic state space
- But do not explore the behaviors not present in $\nu(A)$!

![Diagram showing symbolic state space exploration and behavior exclusion](image-url)
**PRP: Case 1**

As long as $l_{bad}$ is not met...

- Explore the symbolic state space
- But do not explore the behaviors not present in $v(A)$!
PRP: Case 1

As long as $l_{bad}$ is not met...

- Explore the symbolic state space
- But do not explore the behaviors not present in $\nu(A)$!
PRP: Case 1

As long as $l_{bad}$ is not met...

- Explore the symbolic state space
- But do not explore the behaviors not present in $v(A)$!
**PRP: Case 1**

As long as \( l_{bad} \) is not met...

- Explore the symbolic state space
- But do not explore the behaviors not present in \( v(A) \)!

![Diagram showing symbolic state space and behaviors]

\( \neg \land \cdots \neg \) Ensures a subset of the behaviors of \( v(A) \), and hence guarantees the unreachability of \( l_{bad} \).
PRP: Case 1

As long as $l_{bad}$ is not met...

- Explore the symbolic state space
- But do not explore the behaviors not present in $\nu(A)$!
PRP: Case 1

As long as $l_{bad}$ is not met...

- Explore the symbolic state space
- But do not explore the behaviors not present in $v(A)!$
**PRP: Case 1**

As long as $l_{bad}$ is not met...

- Explore the symbolic state space
- But do not explore the behaviors not present in $\nu(A)$!
PRP: Case 1

As long as $l_{bad}$ is not met...

- Explore the symbolic state space
- But do not explore the behaviors not present in $\nu(A)$!
PRP: Case 1

As long as $l_{bad}$ is not met...

- Explore the symbolic state space
- But do not explore the behaviors not present in $\nu(A)$!
PRP: Case 1

As long as $l_{bad}$ is not met...

- Explore the symbolic state space
- But do not explore the behaviors not present in $v(A)$!

When no successors, and if $l_{bad}$ was never met:

- return $\neg\Box \land \cdots \land \neg\Box$
- Ensures a subset of the behaviors of $v(A)$, and hence guarantees the unreachability of $l_{bad}$
**PRP: Case 2**

When $l_{bad}$ is met, switch to an EFsynth-like algorithm...

- But still without exploring the behaviors not present in $\nu(A)$

![Diagram](image_url)
PRP: Case 2

When $l_{bad}$ is met, switch to an EFsynth-like algorithm...

- But still without exploring the behaviors not present in $\nu(A)$
PRP: Case 2

When $l_{bad}$ is met, switch to an EFsynth-like algorithm...

\[ \text{But still without exploring the behaviors not present in } v(A) \]
PRP: Case 2

When $l_{bad}$ is met, switch to an EFsynth-like algorithm...

- But still without exploring the behaviors not present in $v(A)$
**PRP: Case 2**

When $l_{bad}$ is met, switch to an EFsynth-like algorithm...

- But still without exploring the behaviors not present in $v(A)$
PRP: Case 2

When $l_{bad}$ is met, switch to an EFsynth-like algorithm...

- But still without exploring the behaviors not present in $v(A)$
PRP: Case 2

When $l_{bad}$ is met, switch to an EFsynth-like algorithm...

- But still without exploring the behaviors not present in $v(A)$
PRP: Case 2

When $l_{bad}$ is met, switch to an EFsynth-like algorithm...

- But still without exploring the behaviors not present in $v(A)$
**PRP: Case 2**

When $l_{bad}$ is met, switch to an EFsynth-like algorithm...

- But still without exploring the behaviors not present in $v(A)$
PRP: Case 2

When $l_{bad}$ is met, switch to an EFsynth-like algorithm...

- But still without exploring the behaviors not present in $v(A)$
PRP: Case 2

When $l_{bad}$ is met, switch to an EFsynth-like algorithm...

- But still without exploring the behaviors not present in $v(A)$
PRP: Case 2

When $l_{bad}$ is met, switch to an EFsynth-like algorithm...

- But still without exploring the behaviors not present in $v(A)$
PRP: Case 2

When $l_{bad}$ is met, switch to an EFsynth-like algorithm...

- But still without exploring the behaviors not present in $\nu(A)$
**PRP: Case 2**

When $l_{bad}$ is met, switch to an EFsynth-like algorithm...

- But still without exploring the behaviors not present in $\nu(A)$
PRP: Case 2

When $l_{bad}$ is met, switch to an EFsynth-like algorithm...

- But still without exploring the behaviors not present in $\nu(A)$
PRP: Case 2

When $l_{bad}$ is met, switch to an EFsynth-like algorithm...

- But still without exploring the behaviors not present in $\nu(A)$
**PRP: Case 2**

When \( l_{bad} \) is met, switch to an EFsynth-like algorithm...

- But still without exploring the behaviors not present in \( \nu(\mathcal{A}) \)

When no successors, and if \( l_{bad} \) was met:

- return \( \square \lor \cdots \lor \square \)

- Guarantees the reachability of \( l_{bad} \)
Parametric reachability preservation cartography (PRPC)
Parametric reachability preservation cartography (PRPC)
Parametric reachability preservation cartography (PRPC)
Parametric reachability preservation cartography (PRPC)
Parametric reachability preservation cartography (PRPC)
Parametric reachability preservation cartography (PRPC)
Parametric reachability preservation cartography (PRPC)
Parametric reachability preservation cartography (PRPC)
Parametric reachability preservation cartography (PRPC)
Parametric reachability preservation cartography (PRPC)
Parametric reachability preservation cartography (PRPC)
Parametric reachability preservation cartography (PRPC)
Parametric reachability preservation cartography (PRPC)
Parametric reachability preservation cartography (PRPC)
Parametric reachability preservation cartography (PRPC)
Parametric reachability preservation cartography (PRPC)
Parametric reachability preservation cartography (PRPC)
Parametric reachability preservation cartography (PRPC)
Parametric reachability preservation cartography (PRPC)
Master worker scheme

Master-worker distribution scheme:

- **Workers**: ask the master for a point (integer parameter valuation), calls PRP on that point, and send the result (constraint) to the master.
- **Master**: is responsible for **smart repartition** of data between the workers.
  - Not trivial at all
Distributing PRPC: general scheme

Example with 2 parameters ($p_1$ and $p_2$) and 4 nodes:

- 1 master
- 3 workers
Distributing PRPC: general scheme

Example with 2 parameters ($p_1$ and $p_2$) and 4 nodes:

- 1 master
- 3 workers
Distributing PRPC: general scheme

Example with 2 parameters ($p_1$ and $p_2$) and 4 nodes:

- 1 master
- 3 workers
Distributing PRPC: general scheme

Example with 2 parameters ($p_1$ and $p_2$) and 4 nodes:

- 1 master
- 3 workers
Distributing PRPC: general scheme

Example with 2 parameters ($p_1$ and $p_2$) and 4 nodes:
- 1 master
- 3 workers
Distributing PRPC: general scheme

Example with 2 parameters ($p_1$ and $p_2$) and 4 nodes:

- 1 master
- 3 workers
Distributing PRPC: general scheme

Example with 2 parameters \((p_1 \text{ and } p_2)\) and 4 nodes:

- 1 master
- 3 workers
Distributing PRPC: general scheme

Example with 2 parameters ($p_1$ and $p_2$) and 4 nodes:

- 1 master
- 3 workers
Distributing PRPC: general scheme

Example with 2 parameters ($p_1$ and $p_2$) and 4 nodes:

- 1 master
- 3 workers
Distributing PRPC: general scheme

Example with 2 parameters ($p_1$ and $p_2$) and 4 nodes:

- 1 master
- 3 workers
Distributing PRPC: general scheme

Example with 2 parameters ($p_1$ and $p_2$) and 4 nodes:

- 1 master
- 3 workers
Distributing PRPC: general scheme

Example with 2 parameters ($p_1$ and $p_2$) and 4 nodes:

- 1 master
- 3 workers
Distributing PRPC: general scheme

Example with 2 parameters ($p_1$ and $p_2$) and 4 nodes:

- 1 master
- 3 workers
Distributing PRPC: general scheme

Example with 2 parameters ($p_1$ and $p_2$) and 4 nodes:

- 1 master
- 3 workers
Distributing PRPC: general scheme

Example with 2 parameters \((p_1 \text{ and } p_2)\) and 4 nodes:

- 1 master
- 3 workers
Distributing PRPC: general scheme

Example with 2 parameters ($p_1$ and $p_2$) and 4 nodes:
- 1 master
- 3 workers
Distributing PRPC: general scheme

Example with 2 parameters (p₁ and p₂) and 4 nodes:

- 1 master
- 3 workers
Distributing PRPC: general scheme

Example with 2 parameters ($p_1$ and $p_2$) and 4 nodes:

- 1 master
- 3 workers
Distributing PRPC: general scheme

Example with 2 parameters ($p_1$ and $p_2$) and 4 nodes:

- 1 master
- 3 workers
Distributing PRPC: general scheme

Example with 2 parameters ($p_1$ and $p_2$) and 4 nodes:

- 1 master
- 3 workers
Distributing PRPC: general scheme

Example with 2 parameters \((p_1 \text{ and } p_2)\) and 4 nodes:
- 1 master
- 3 workers

Problem:
- redundant computations
Dynamic domain decomposition

Most efficient distributed algorithm (so far!):
“Domain decomposition” scheme

**Master**

1. initially splits the parameter domain into subdomains and send them to the workers
2. when a worker has completed its subdomain, the master splits another subdomain, and sends it to the idle worker

**Workers**

1. receive the subdomain from the master
2. call PRP on the points of this subdomain
3. send the results (list of constraints) back to the master
4. ask for more work
Domain decomposition: Initial splitting

- Prevent choosing close points
- Prevent bottleneck phenomenon at the master’s side
  - Master only responsible for gathering constraints and splitting subdomains
Domain decomposition: Dynamic splitting

- Master can balance workload between workers
Implementation in IMITATOR

Implemented in IMITATOR using the MPI paradigm (message passing interface)

Distributed version up to 44 times faster using 128 nodes than the monolithic EFsynth

[André et al., 2015a]
Outline

1. Parametric Timed Automata
2. Modeling and Verifying Real-Time Systems
3. Verifying a Real-time System under Uncertainty
4. Distributed Verification of Distributed Systems
5. Conclusion and Perspectives
Summary

- **Parametric timed automata**: a powerful (though undecidable) formalism to model and verify real-time systems
  - with preemption
  - with *unknown* or *uncertain* periods, jitters or WCETs
Perspectives

Address harder problems

- Thales challenge: what is the minimum time between two lost frames? (due to the uncertain periods)
  - Requires to model check thousands of frame processings

Improve the efficiency of parameter synthesis techniques

- Promising heuristics: approximations using the integer hull
  
  [Jovanović et al., 2015, André et al., 2015b]

- Distributed parameter synthesis
  
  - Multi-core synthesis
  
  - Distributed synthesis based on locations

  [Laarman et al., 2013]
  
  [Zhang et al., 2016]
Bibliography


References II


References IV

PAT: Towards flexible verification under fairness.
In CAV, volume 5643 of Lecture Notes in Computer Science, pages 709–714. Springer.

Parametric schedulability analysis of fixed priority real-time distributed systems.
In FTSCS, volume 419 of Communications in Computer and Information Science, pages 212–228. Springer.

Distributed algorithms for time optimal reachability analysis.
Additional explanation
Explanation for the 4 pictures in the beginning

Allusion to the Northeast blackout (USA, 2003)
Computer bug
Consequences: 11 fatalities, huge cost
(Picture actually from the Sandy Hurricane, 2012)

Error screen on the earliest versions of Macintosh

Allusion to the sinking of the Sleipner A offshore platform (Norway, 1991)
No fatalities
Computer bug: inaccurate finite element analysis modeling
(Picture actually from the Deepwater Horizon Offshore Drilling Platform)

Allusion to the MIM-104 Patriot Missile Failure (Iraq, 1991)
28 fatalities, hundreds of injured
Computer bug: software error (clock drift)
(Picture of an actual MIM-104 Patriot Missile, though not the one of 1991)
Licensing
Source of the graphics used I

Title: Hurricane Sandy Blackout New York Skyline
Author: David Shankbone
Source: https://commons.wikimedia.org/wiki/File:Hurricane_Sandy_Blackout_New_York_Skyline.JPG
License: CC BY 3.0

Title: Sad mac
Author: Przemub
Source: https://commons.wikimedia.org/wiki/File:Sad_mac.png
License: Public domain

Title: Deepwater Horizon Offshore Drilling Platform on Fire
Author: ideum
Source: https://secure.flickr.com/photos/ideum/4711481781/
License: CC BY-SA 2.0

Title: DA-SC-88-01663
Author: imcomkorea
Source: https://secure.flickr.com/photos/imcomkorea/3017886760/
License: CC BY-NC-ND 2.0
Source of the graphics used II

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

Title: Renault Twizy
Author: Citron
Source: https://commons.wikimedia.org/wiki/File:Renault_Twizy.jpg
License: CC BY-SA 3.0

Title: Airbus A380
Author: Axwel
Source: https://commons.wikimedia.org/wiki/File:Airbus_A380_blue_sky.jpg
License: CC BY 2.0
Source of the graphics used III
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/