Reachability Preservation Based Parameter Synthesis for Timed Automata

Étienne André\textsuperscript{1}, Giuseppe Lipari\textsuperscript{2}, Hoang Gia Nguyen\textsuperscript{1}, Youcheng Sun\textsuperscript{3}

\textsuperscript{1}LIPN, Université Paris 13, Sorbonne Paris Cité, CNRS, France
\textsuperscript{2}CRIStAL – UMR 9189, Université de Lille, USR 3380 CNRS, France
\textsuperscript{3}Scuola Superiore Sant’Anna, Pisa, Italy
Context: Formal Verification of Timed Systems

- Model checking

A model of the system

A property to be satisfied
**Context: Formal Verification of Timed Systems**

- **Model checking**

A model of the system

A property to be satisfied

**Question:** does the model of the system **satisfy** the property?
Context: Formal Verification of Timed Systems

- Model checking

A model of the system

\[ \text{A property to be satisfied} \]

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

Yes

No

Counterexample
Beyond Model Checking: Parameter Synthesis

- 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”

- 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?
Beyond Model Checking: Parameter Synthesis

- 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”

- 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?

- Parameter synthesis
  - Consider that timing constants are unknown constants (parameters)
  - Find good values for the parameters
Outline

1. Parametric Timed Automata
2. Reachability Preservation using PRP
3. EF-Synthesis Using PRPC
4. Experiments
5. Conclusion and Perspectives
Outline

1. Parametric Timed Automata
2. Reachability Preservation using PRP
3. EF-Synthesis Using PRPC
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)

$$\begin{align*}
x &:= 0 \\
y &:= 0 \\
y &:= 5 \\
press? &
\end{align*}$$

$$\begin{align*}
x &:= 0 \\
y &:= 8 \\
coee! &
\end{align*}$$
Timed Automaton (TA)

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

```
x := 0
y := 0

y = 5

press?

coffie!

cup!

press?```
Timed Automaton (TA)

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

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

```plaintext
y \leq 5
```

```plaintext
x \geq 1
```

```
express
```

```
press?
```

```
cup!
```

```
press?
```

```
express
```

```
press?
```
Timed Automaton (TA)

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

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

```
y ≤ 5
press?
```
```
x ≥ 1
cup!
```
```
y = 8
coffee!
```
```
y = 5
```
```
y ≤ 8
```

---

Press your favorite living room button and the coffee table will spring into action, and it will stay there until you press the second button.

Press the second button and the coffee table will disappear from your living room.
**Timed Automaton (TA)**

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

**Features**

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

```
x := 0
y := 0

x ≥ 1
y = 5
press?

y = 8
coffee!

y ≤ 5

x := 0

y ≤ 8
cup!
```
Timed Automata: A Coffee Vending Machine

- Examples of concrete runs
Timed Automata: A Coffee Vending Machine

Examples of concrete runs

- Coffee with no sugar
Timed Automata: A Coffee Vending Machine

Examples of concrete runs

- Coffee with no sugar
Timed Automata: A Coffee Vending Machine

Examples of concrete runs

- Coffee with no sugar
Timed Automata: A Coffee Vending Machine

- Examples of concrete runs
  - Coffee with no sugar

<table>
<thead>
<tr>
<th>Action</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>press?</td>
<td>0</td>
</tr>
<tr>
<td>x := 0</td>
<td>0</td>
</tr>
<tr>
<td>y := 0</td>
<td>0</td>
</tr>
<tr>
<td>press?</td>
<td>5</td>
</tr>
<tr>
<td>y = 5</td>
<td>5</td>
</tr>
<tr>
<td>cup!</td>
<td>5</td>
</tr>
<tr>
<td>press?</td>
<td>0</td>
</tr>
<tr>
<td>x := 0</td>
<td>0</td>
</tr>
<tr>
<td>y := 0</td>
<td>0</td>
</tr>
<tr>
<td>press?</td>
<td>5</td>
</tr>
<tr>
<td>y = 5</td>
<td>5</td>
</tr>
<tr>
<td>cup!</td>
<td>5</td>
</tr>
</tbody>
</table>
**Timed Automata: A Coffee Vending Machine**

- Initial state: $y = 8$
- Transition: $x = 0$, $y = 0$
- Transition condition: $y \leq 5$
- Coffee output

Examples of concrete runs:

- **Coffee with no sugar**
  - Initial state: $x = 0$, $y = 0$
  - Transition: $x = 0$, $y = 5$
  - Transition condition: $y \geq 5$
  - Cup output
  - Final state: $x = 0$, $y = 8$

<table>
<thead>
<tr>
<th>Pressed</th>
<th>Cupped</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$y$</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>3</td>
<td>8</td>
</tr>
</tbody>
</table>
Timed Automata: A Coffee Vending Machine

y = 8
coffee!

y \leq 5

press?
x := 0
y := 0

y = 5
cup!
x \geq 1
press?
x := 0

Examples of concrete runs

Coffee with no sugar

x  y  press?  5  cup!  3  coffee!
0 0
0 0
5 5
5 5
8 3
8 8
Timed Automata: A Coffee Vending Machine

**Examples of concrete runs**

- **Coffee with no sugar**

- **Coffee with 2 doses of sugar**
Timed Automata: A Coffee Vending Machine

Examples of concrete runs

- Coffee with no sugar
- Coffee with 2 doses of sugar
Timed Automata: A Coffee Vending Machine

Examples of concrete runs

- **Coffee with no sugar**
  - Press? 5 Cup! 3 Coffee!
  - \( x = 0 \)
  - \( y = 5 \)
  - \( x = 0 \)
  - \( y = 8 \)

- **Coffee with 2 doses of sugar**
  - Press? 1.5
  - \( x = 0 \)
  - \( y = 0 \)
  - \( x = 0 \)
  - \( y = 1.5 \)
Timed Automata: A Coffee Vending Machine

\[
y = 8 \\
\text{coffee!}
\]

\[
y \leq 5 \\
\text{press?} \\
x := 0 \\
y := 0
\]

\[
y \geq 1 \\
\text{press?} \\
x := 0 \\
y = 5 \\
\text{cup!}
\]

Examples of concrete runs

- 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>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>1.5</td>
</tr>
<tr>
<td>1.5</td>
<td>0</td>
</tr>
</tbody>
</table>

x := 0
y := 0

x \geq 1
press?

x := 0
y = 5
cup!

y \leq 5
press?

x := 0
y := 0

y \leq 8

Timed Automata: A Coffee Vending Machine

Examples of concrete runs

- **Coffee with no sugar**

- **Coffee with 2 doses of sugar**
Timed Automata: A Coffee Vending Machine

Examples of concrete runs

- Coffee with no sugar

- Coffee with 2 doses of sugar
Timed Automata: A Coffee Vending Machine

- Examples of concrete runs
  - Coffee with no sugar
  - Coffee with 2 doses of sugar
Timed Automata: A Coffee Vending Machine

\[
\begin{align*}
 & \text{Press?} \\
 & y = 8 \\
 & \text{Coffee!}
\end{align*}
\]

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

\[
\begin{align*}
 & \text{Press?} \\
 & x \geq 1 \\
 & y = 5 \\
 & \text{Cup!}
\end{align*}
\]

Examples of concrete runs

- Coffee with no sugar

\[
\begin{align*}
 & \text{Press?} \quad 5 \quad \text{Cup!} \quad 3 \quad \text{Coffee!}
\end{align*}
\]

\[
\begin{align*}
 & x \quad 0 \quad 5 \quad 5 \quad 8 \quad 8 \\
 & y \quad 0 \quad 5 \quad 5 \quad 8 \quad 8
\end{align*}
\]

- Coffee with 2 doses of sugar

\[
\begin{align*}
 & \text{Press?} \quad 1.5 \quad \text{Press?} \quad 2.7 \quad \text{Press?} \quad 0.8 \quad \text{Cup!}
\end{align*}
\]

\[
\begin{align*}
 & x \quad 0 \quad 0 \quad 1.5 \quad 0 \quad 2.7 \quad 0 \quad 0.8 \quad 0.8 \\
 & y \quad 0 \quad 1.5 \quad 1.5 \quad 4.2 \quad 4.2 \quad 5 \quad 5
\end{align*}
\]
Timed Automata: A Coffee Vending Machine

**Examples of concrete runs**

- **Coffee with no sugar**

  ```
  x  y  press?  5  cup!  3  coffee!
  0  0  0  5  5  8  8
  0  0  5  5  8  8
  ```

- **Coffee with 2 doses of sugar**

  ```
  x  y  press?  1.5  press?  2.7  press?  0.8  cup!  3
  0  0  0  1.5  0  2.7  0  0.8  0.8  3
  0  0  1.5  1.5  4.2  4.2  5  5  8
  ```
Timed Automata: A Coffee Vending Machine

Examples of concrete runs

- **Coffee with no sugar**
  - Press? 5
  - Cup! 3
  - Coffee!

- **Coffee with 2 doses of sugar**
  - Press? 1.5
  - Press? 2.7
  - Press? 0.8
  - Cup! 3
  - Coffee!
Parametric Timed Automaton (PTA)

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

Example of problems
- Do there exist parameter valuations such that one can never get a coffee?
- Yes! e.g.: \( p_1 = 2, p_2 = 10 \)
- What are all possible parameter valuations such that one can get a coffee with 3 doses of sugar?
  \[ p_2 \leq 8 \land p_2 \geq 3 \times p_1 \]
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

\[ y = 8 \]

coffee!

\[ y \leq p_2 \]

\[ x \geq p_1 \]

press?

\[ x := 0 \]

\[ y := 0 \]

\[ x := 0 \]

\[ y = p_2 \]

cup!

\[ y \leq 8 \]
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
    - $y = 8$
    - coffee!

Examples of problems
  - “Do there exist parameter valuations such that one can never get a coffee?”
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 &= 8 \\
\text{coffee!}
\end{align*}

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

\begin{align*}
\text{press?} \\
x &\geq p_1 \\
y &= p_2 \\
\text{cup!}
\end{align*}

■ Examples of problems
  - “Do there exist parameter valuations such that one can never get a coffee?” Yes! e.g.: $p_1 = 2$, $p_2 = 10$
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
    
    \[
    y = 8 \\
    \text{coffee!}
    \]

Examples of problems

- “Do there exist parameter valuations such that one can never get a coffee?” Yes! e.g.: \( p_1 = 2, p_2 = 10 \)
- “What are all possible parameter valuations such that one can get a coffee with 3 doses of sugar?”
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
    - $y = 8$
    - coffee!
  - $y \leq p_2$

Examples of problems

- “Do there exist parameter valuations such that one can never get a coffee?” Yes! e.g.: $p_1 = 2, p_2 = 10$
- “What are all possible parameter valuations such that one can get a coffee with 3 doses of sugar?” $p_2 \leq 8 \land p_2 \geq 3 \times p_1$
Valuation of a PTA

- A valuation $\pi$ of all the parameters of $P$ is called a point.

- Given a PTA $A$ and a point $\pi$, we denote by $A[\pi]$ the (non-parametric) timed automaton where all parameters are valuated by $\pi$. 
Objective: Reachability Synthesis

Problem (EF-emptiness)

Let $A$ be a PTA. Is the set of parameter valuations $\pi$ such that $A[\pi]$ reaches $l_{bad}$ empty?
Objective: Reachability Synthesis

Problem (EF-emptiness)

Let $A$ be a PTA. Is the set of parameter valuations $\pi$ such that $A[\pi]$ reaches $l_{\text{bad}}$ empty?

Theorem

The EF-emptiness problem is undecidable. [Alur et al., 1993]
Objective: Reachability Synthesis

Problem (EF-emptiness)

Let $A$ be a PTA. Is the set of parameter valuations $\pi$ such that $A[\pi]$ reaches $l_{\text{bad}}$ empty?

Theorem

The EF-emptiness problem is undecidable. [Alur et al., 1993]

Problem (EF-synthesis)

Let $A$ be a PTA. Compute the set of parameter valuations $\pi$ such that $A[\pi]$ reaches $l_{\text{bad}}$. 
Previous Works

- Semi-algorithm **EFsynth** proposed in [Alur et al., 1993]

- Synthesis of **integer** parameter valuations
  - Enumerative terminating algorithm for 2 subclasses of PTA (“L-PTA and U-PTA”) [Bozzelli and La Torre, 2009]
  - Symbolic terminating algorithm for general PTA with a bounded parameter domain [Jovanović et al., 2014]
Previous Works

- Semi-algorithm **EFsynth** proposed in [Alur et al., 1993]

- Synthesis of **integer** parameter valuations
  - Enumerative terminating algorithm for 2 subclasses of PTA ("L-PTA and U-PTA") [Bozzelli and La Torre, 2009]
  - Symbolic terminating algorithm for general PTA with a bounded parameter domain [Jovanović et al., 2014]

- Here: **reachability preservation-based** approach
  - For rational-valued parameter valuations
Previous Works

- Semi-algorithm EFsynth proposed in [Alur et al., 1993]

- Synthesis of integer parameter valuations
  - Enumerative terminating algorithm for 2 subclasses of PTA ("L-PTA and U-PTA") [Bozzelli and La Torre, 2009]
  - Symbolic terminating algorithm for general PTA with a bounded parameter domain [Jovanović et al., 2014]

- Here: reachability preservation-based approach
  - For rational-valued parameter valuations
  - 😊 ... and that can be distributed
Outline

1. Parametric Timed Automata
2. Reachability Preservation using PRP
3. EF-Synthesis Using PRPC
4. Experiments
5. Conclusion and Perspectives
“If we know a parameter valuation $\pi$ that reaches (resp. does not reach) $l_{bad}$, can we find other valuations around $\pi$ that reach (resp. do not reach) $l_{bad}$?”
Reachability Preservation

Key idea

“If we know a parameter valuation $\pi$ that reaches (resp. does not reach) $l_{bad}$, can we find other valuations around $\pi$ that reach (resp. do not reach) $l_{bad}$?”
Reachability Preservation

Key idea

“If we know a parameter valuation $\pi$ that reaches (resp. does not reach) $l_{bad}$, can we find other valuations around $\pi$ that reach (resp. do not reach) $l_{bad}$?”
Reachability Preservation: Undecidability

Problem (PREACH-emptiness)

Let $A$ be a PTA, and $\pi$ a parameter valuation. Does there exist $\pi' \neq \pi$ such that $A[\pi']$ preserves the reachability of $l_{bad}$ in $A[\pi]$?

Theorem
PREACH-emptiness is undecidable.

Proof.
Reachability Preservation: Undecidability

Problem (PREACH-emptiness)

Let $A$ be a PTA, and $\pi$ a parameter valuation. Does there exist $\pi' \neq \pi$ such that $A[\pi']$ preserves the reachability of $l_{bad}$ in $A[\pi]$?

Theorem

PREACH-emptiness is undecidable.
Reachability Preservation: Undecidability

Problem (PREACH-emptiness)

Let \( A \) be a PTA, and \( \pi \) a parameter valuation. Does there exist \( \pi' \neq \pi \) such that \( A[\pi'] \) preserves the reachability of \( l_{bad} \) in \( A[\pi] \)?

Theorem

PREACH-emptiness is undecidable.

Proof.
PRP: Parametric Reachability Preservation

Input: parameter valuation $\pi$
Output: constraint $K$ such that

1. $\pi \models K$, and
2. $\forall \pi' \models K$, $A[\pi']$ preserves the reachability of $l_{bad}$ in $A[\pi]$

Inspired by EFsynth [Alur et al., 1993, Jovanović et al., 2014] and IM$^K$ [André and Soulat, 2011]
PRP: Parametric Reachability Preservation

Input: parameter valuation $\pi$
Output: constraint $K$ such that

1. $\pi \models K$, and
2. $\forall \pi' \models K$, $A[\pi']$ preserves the reachability of $l_{bad}$ in $A[\pi]$
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 $A[\pi]$!
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 $A[\pi]$!
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 $A[\pi]$!

\[ \text{ when no successors, and if } l_{bad} \text{ was never met: } \neg \land \cdots \land \neg \]

Ensures a subset of the behaviors of $A[\pi]$, 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 $A[\pi]$!
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 $A[\pi]$!
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 $A[\pi]$!
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 $\mathcal{A}[\pi]$!
PRP: Case 1

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

- Explore the symbolic state space
- But do not explore the behaviors not present in $A[\pi]$!

---

Explore the symbolic state space

But do not explore the behaviors not present in $A[\pi]$!
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 $A[\pi]$!
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 $A[\pi]$!
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 $A[\pi]$!
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 $A[\pi]$!
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 $A[\pi]$!
PRP: Case 1

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

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

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

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

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

- return $\neg \circ \land \cdots \land \neg \circ$
- Ensures a subset of the behaviors of $A[\pi]$, and hence guarantees the unreachability of $l_{bad}$
PRP: Case 1 (Remark)

Questions

How do we know the possible behaviors of $A[\pi]$?
How do we know that a symbolic state of $A$ corresponds to a behavior of $A[\pi]$?
PRP: Case 1 (Remark)

Questions

How do we know the possible behaviors of $A[\pi]$?
How do we know that a symbolic state of $A$ corresponds to a behavior of $A[\pi]$?

We could compute the zone graph of $A[\pi]$.
But this is not necessary.
In fact, we do not even need to know whether $A[\pi]$ reaches $l_{bad}$ or not.
PRP: Case 1 (Remark)

Questions

How do we know the possible behaviors of $A[\pi]$?
How do we know that a symbolic state of $A$ corresponds to a behavior of $A[\pi]$?

We could compute the zone graph of $A[\pi]$.
But this is not necessary.
In fact, we do not even need to know whether $A[\pi]$ reaches $l_{bad}$ or not.

Trick

A symbolic state $(l, C)$ corresponds to a behavior of $A[\pi]$ iff $\pi \models C$. 
PRP: Case 2

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

- But still without exploring the behaviors not present in $A[\pi]$
PRP: Case 2

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

- But still without exploring the behaviors not present in $A[\pi]$
PRP: Case 2

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

- But still without exploring the behaviors not present in $A[\pi]$
PRP: Case 2

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

- But still without exploring the behaviors not present in $A[\pi]$
PRP: Case 2

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

- But still without exploring the behaviors not present in $A[\pi]$. 

![Graph diagram showing reachability preservation using PRP.](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 $A[\pi]$. 
PRP: Case 2

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

- But still without exploring the behaviors not present in $A[\pi]$
**PRP: Case 2**

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

- But still without exploring the behaviors not present in $A[\pi]$
PRP: Case 2

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

- But still without exploring the behaviors not present in $A[\pi]$
PRP: Case 2

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

- But still without exploring the behaviors not present in $A[\pi]$
PRP: Case 2

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

- But still without exploring the behaviors not present in $A[\pi]$
PRP: Case 2

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

- But still without exploring the behaviors not present in $A[\pi]$
PRP: Case 2

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

- But still without exploring the behaviors not present in $A[\pi]$
PRP: Case 2

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

- But still without exploring the behaviors not present in $A[\pi]$
PRP: Case 2

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

- But still without exploring the behaviors not present in $A[\pi]$
PRP: Case 2

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

- But still without exploring the behaviors not present in $A[\pi]$
PRP: Case 2

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

- But still without exploring the behaviors not present in $A[\pi]$

When no successors, and if $l_{bad}$ was met:

- return $\bigvee \ldots \bigvee$

- Guarantees the reachability of $l_{bad}$
PRP: Early termination

Recall that PREACH-emptiness is undecidable
Hence PRP may not terminate.
PRP: Early termination

Recall that PREACH-emptiness is undecidable
Hence PRP may not terminate.

Proposition (Early termination)

If $\text{PRP}(A, \pi)$ does not terminate and is interrupted (e.g., after a timeout), the result is still a valid under-approximation provided $l_{bad}$ has been reached.

This is also true for EFsynth (in any case)
Outline

1. Parametric Timed Automata
2. Reachability Preservation using PRP
3. EF-Synthesis Using PRPC
4. Experiments
5. Conclusion and Perspectives
Perform EF-synthesis using PRP

Input: parameter bounded domain $V$
Output: constraints on the parameter such that $l_{bad}$ is / is not reachable in $A$

- The idea: reuse the “behavioral cartography” of parametric timed automata [André and Fribourg, 2010]
- Iterate on integer points, and call PRP on each point not covered by a constraint
  - If no termination: break, and keep result if possible (i.e., if $l_{bad}$ is reachable in this analysis)
PRPC: Reusing the Behavioral Cartography

Partition the domain $V$ into constraints where the reachability of $l_{bad}$ is uniform.

Method: done by calling PRP on integer points (parameter valuations) sequentially.
PRPC: Reusing the Behavioral Cartography

Partition the domain $V$ into constraints where the reachability of $l_{bad}$ is uniform.

Method: done by calling PRP on integer points (parameter valuations) sequentially.
PRPC: Reusing the Behavioral Cartography

Partition the domain $V$ into constraints where the reachability of $l_{bad}$ is uniform.

Method: done by calling PRP on integer points (parameter valuations) sequentially.
**PRPC: Reusing the Behavioral Cartography**

Partition the domain $V$ into constraints where the reachability of $l_{bad}$ is uniform.

Method: done by calling PRP on integer points (parameter valuations) sequentially.
**PRPC: Reusing the Behavioral Cartography**

Partition the domain $V$ into constraints where the reachability of $l_{bad}$ is uniform.

Method: done by calling PRP on integer points (parameter valuations) sequentially.
**PRPC: Reusing the Behavioral Cartography**

Partition the domain $V$ into constraints where the reachability of $l_{bad}$ is uniform.

Method: done by calling PRP on integer points (parameter valuations) sequentially.
PRPC: Reusing the Behavioral Cartography

Partition the domain $V$ into constraints where the reachability of $l_{bad}$ is uniform.

Method: done by calling PRP on integer points (parameter valuations) sequentially.
**PRPC: Reusing the Behavioral Cartography**

Partition the domain $\mathcal{V}$ into constraints where the reachability of $l_{bad}$ is uniform

Method: done by calling PRP on integer points (parameter valuations) sequentially
Result: “interval” under-approximation

- **PRPC** synthesizes:
  - An under-approximation of the bad constraints (reaching $l_{bad}$)
  - An under-approximation of the good constraints (avoiding $l_{bad}$)

- **EFsynth** synthesizes:
  - An under-approximation of the bad constraints

⇒ The result of **PRPC** is more valuable than **EFsynth**, at least when **EFsynth** does not terminate and is interrupted
Towards Distributed Parameter Synthesis

Idea

Calling sequentially PRP on various integer points in a bounded parameter domain looks like something that can be easily distributed.
Towards Distributed Parameter Synthesis

Idea

Calling sequentially PRP on various integer points in a bounded parameter domain looks like something that can be easily distributed.

Reuse the distributed algorithms to compute the behavioral cartography of parametric timed automata [A., Coti, Evangelista, 2014]
Master Worker Scheme

Master-Worker distribution scheme:

- **Workers**: ask the master for a point, calls PRP on that point, and send the result (constraint) to the master
- **Master**: is responsible for *smart repartition* of data between the workers
  - (Note: not trivial at all)
Dynamic Decomposition of BC

Most efficient distributed algorithm for BC (so far!): “Domain decomposition” scheme [work in progress]

- **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. Receives the subdomain from the master
  2. Calls PRP on the points of this subdomain
  3. Sends the results (list of constraints) back to the master
  4. Asks for more work
Domain Decomposition: Initial Splitting

- Prevent to choose close points
- Prevent bottleneck phenomenon at the master side
  - Master only responsible for gathering constraints and splitting subdomains
Domain Decomposition: Dynamic Splitting

Master can balance workload between workers
Outline

1 Parametric Timed Automata
2 Reachability Preservation using PRP
3 EF-Synthesis Using PRPC
4 Experiments
5 Conclusion and Perspectives
Implementation in IMITATOR

- **IMITATOR** [A., Fribourg, Kühne, Soulat, 2012]
  - 26,000 lines of OCaml code
  - Development started in 2009... in Hilton Pasadena!
  - Relies on the PPL library for operations on polyhedra [Bagnara et al., 2008]
  - Available under the GNU-GPL license
  - Latest version (2.7) implements distributed algorithms

- Distributed version of IMITATOR relying on **MPI**
  - Using the OcamlMPI library
Implementation in IMITATOR

- **IMITATOR** [A., Fribourg, Kühne, Soulat, 2012]
  - 26,000 lines of OCaml code
  - Development started in 2009... in Hilton Pasadena!
- Relies on the PPL library for operations on polyhedra [Bagnara et al., 2008]
- Available under the GNU-GPL license
- Latest version (2.7) implements distributed algorithms

- Distributed version of IMITATOR relying on MPI
  - Using the OcamlMPI library

http://www.imitator.fr/
### PRPC: experiments

<table>
<thead>
<tr>
<th>Case study</th>
<th></th>
<th></th>
<th>EFsynth</th>
<th>BC</th>
<th>PRPC</th>
<th>PRPC distr(12)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>2</td>
<td>2,601</td>
<td>0.401*</td>
<td>TO</td>
<td>0.078*</td>
<td>0.050*</td>
</tr>
<tr>
<td>Sched1</td>
<td>13</td>
<td>6,561</td>
<td>TO</td>
<td>TO</td>
<td>1,595</td>
<td>219</td>
</tr>
<tr>
<td>Sched2.50.0</td>
<td>6</td>
<td>3,321</td>
<td>9.25</td>
<td>990</td>
<td>14.55</td>
<td>4.77</td>
</tr>
<tr>
<td>Sched2.50.2</td>
<td>6</td>
<td>3,321</td>
<td>662</td>
<td>TO</td>
<td>213</td>
<td>84</td>
</tr>
<tr>
<td>Sched2.100.0</td>
<td>6</td>
<td>972,971</td>
<td>21.4</td>
<td>2,093</td>
<td>116</td>
<td>10.1</td>
</tr>
<tr>
<td>Sched2.100.2</td>
<td>6</td>
<td>972,971</td>
<td>3,757</td>
<td>TO</td>
<td>4,557</td>
<td>1,543</td>
</tr>
<tr>
<td>Sched5</td>
<td>21</td>
<td>1,681</td>
<td>352</td>
<td>TO</td>
<td>TO</td>
<td>917</td>
</tr>
<tr>
<td>SPSMALL</td>
<td>11</td>
<td>3,082</td>
<td>7.49</td>
<td>587</td>
<td>118</td>
<td>11.2</td>
</tr>
</tbody>
</table>

**IMITATOR version:** 2.6.2, build 845

* experiment run using `-depth-limit 10` (does not terminate in general)

Experiments available at [http://www.imitator.fr/static/NFM15/](http://www.imitator.fr/static/NFM15/)
Outline

1. Parametric Timed Automata
2. Reachability Preservation using PRP
3. EF-Synthesis Using PRPC
4. Experiments
5. Conclusion and Perspectives
Summary

PRP
- Given a parameter valuation $\pi$ and a location $l_{bad}$, outputs a dense set of parameter valuations around $\pi$ that preserve the (un)reachability of $l_{bad}$

PRPC
- Computes an under-approximated set of parameter valuations reaching / not reaching $l_{bad}$
- Can be distributed
- Often outperforms EFsynth, especially when distributed
Perspectives

- Improvement: always return both good and bad constraints (for both PRP and EFsynth)
  
- Combine with integer hull to ensure termination
  [Jovanović et al., 2014]
  
  - At least for integer valuations

- Combine with multi-core techniques [Laarman et al., 2013]

- Verify the communication scheme in the distributed IMITATOR for an arbitrary number of nodes
  
  - Using parametric verification techniques?
Perspectives

- Improvement: always return both good and bad constraints (for both PRP and EFsynth)

- Combine with integer hull to ensure termination [Jovanović et al., 2014]
  - At least for integer valuations

- Combine with multi-core techniques [Laarman et al., 2013]

- Verify the communication scheme in the distributed IMITATOR for an arbitrary number of nodes
  - Using parametric verification techniques?

- Extend to compositional verification
Bibliography
References I


References II

Synthesis of timing parameters satisfying safety properties.

The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems.

Decision problems for lower/upper bound parametric timed automata.

Integer parameter synthesis for timed automata.
IEEE Transactions on Software Engineering (TSE).
To appear.

Additional explanation
PRP: The Algorithm

Algorithm 1: PRP(A, \pi)

input : PTA \ A of initial state \ s_0, parameter valuation \ \pi
output : Constraint over the parameters

\begin{verbatim}
S \leftarrow \emptyset; \ S_{\text{new}} \leftarrow \{s_0\}; \ Bad \leftarrow \text{false}; \ K_{\text{good}} \leftarrow \top; \ K_{\text{bad}} \leftarrow \bot; \ i \leftarrow 0
while true do
    foreach \ \pi\text{-incompatible state} \ (l, C) \ in \ S_{\text{new}} \ do
        S_{\text{new}} \leftarrow S_{\text{new}} \setminus \{(l, C)\}
        if Bad = false then
            Select a \ \pi\text{-incompatible inequality} \ J \ in \ C_{\downarrow P} \ (i.e., \ s.t. \ \pi \not\models J)
            K_{\text{good}} \leftarrow K_{\text{good}} \land \neg J
        endforeach
    foreach bad state \ (l_{\text{bad}}, C) \ in \ S_{\text{new}} \ do
        Bad \leftarrow \text{true}; \ K_{\text{bad}} \leftarrow K_{\text{bad}} \lor C_{\downarrow P}; \ S_{\text{new}} \leftarrow S_{\text{new}} \setminus \{(l_{\text{bad}}, C)\}
    if S_{\text{new}} \subseteq S \ then
        if Bad = true then return \ K_{\text{bad}} \ else return \ K_{\text{good}} ;
    S \leftarrow S \cup S_{\text{new}}; \ S_{\text{new}} \leftarrow \text{Succ}(S_{\text{new}}); \ i \leftarrow i + 1
\end{verbatim}

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)

(L\LaTeX source available on demand)

Authors: Étienne André

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