Computing Students Talks

26th January 2011

The Good, the Bad and the Unknown

Synthesis of Timing Parameters in Concurrent Systems

Étienne ANDRÉ

PAT Team

School of Computing, National University of Singapore
Verification of Real Time Systems

Motivation
Context: Model Checking Timed Systems (1/2)

- **Input**

A timed concurrent system
Introduction

Context: Model Checking Timed Systems (1/2)

- Input

A timed concurrent system
A good behavior expected for the system
Context: Model Checking Timed Systems (1/2)

- Input

A timed concurrent system

A good behavior expected for the system

- Question: does the system always behave well?
Context: Model Checking Timed Systems (2/2)

- Use of formal methods

A finite model of the system

$AG\neg\text{bad}$

A formula to be satisfied
Context: Model Checking Timed Systems (2/2)

- Use of formal methods

\[ \text{AG} \neg \text{bad} \]

A finite model of the system

A formula to be satisfied

- Question: does the model of the system satisfy the formula?
Context: Model Checking Timed Systems (2/2)

- Use of formal methods

A finite model of the system

A formula to be satisfied

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

Yes

No

Counterexample
Outline

1. A Coffee Vending Machine
2. A Parametric Coffee Vending Machine
3. Synthesis of Parameters
4. Conclusion
Outline

1. A Coffee Vending Machine
2. A Parametric Coffee Vending Machine
3. Synthesis of Parameters
4. Conclusion
Model

- Waiting
- Adding sugar
- Delivering coffee

A Coffee Vending Machine

press?

press?

Waiting
Adding sugar
Delivering coffee

cup!

coffee!

Example of runs

Coffee with no sugar
press?
cup!
coffee!

Coffee with 2 doses of sugar
press?
press?
press?
cup!
coffee!

And so on
Model

Example of runs
- Coffee with no sugar

Waiting
- Adding sugar
- Delivering coffee
**Model**

- **Waiting**
- **Adding sugar**
- **Delivering coffee**

**Example of runs**

- **Coffee with no sugar**
  - press?
  - cup!
  - coffee!

- **Coffee with 2 doses of sugar**
  - press?
  - press?
  - press?
  - cup!
  - coffee!
Model

- Waiting
- Adding sugar
- Delivering coffee

Example of runs

- Coffee with no sugar
  - press?
  - cup!
  - coffee!

- Coffee with 2 doses of sugar
  - press?
  - press?
  - press?
  - cup!
  - coffee!

- And so on
Temporal Logics

- Specify properties on the order between events

- Example: CTL (Computation Tree Logic)
  [Clarke and Emerson, 1982]
  - “After the button is pressed, a coffee is always eventually delivered.”
Temporal Logics

- Specify properties on the order between events

- Example: CTL (Computation Tree Logic)  
  [Clarke and Emerson, 1982]
  - “After the button is pressed, a coffee is always eventually delivered.” (×)
Temporal Logics

- Specify properties on the **order** between events

- Example: **CTL** (Computation Tree Logic)  
  [Clarke and Emerson, 1982]
  - “After the button is pressed, a coffee is always eventually delivered.” (✗)
  - “After the button is pressed, there exists an execution such that a coffee is eventually delivered.”
Temporal Logics

- Specify properties on the order between events

- Example: CTL (Computation Tree Logic)
  [Clarke and Emerson, 1982]
  - “After the button is pressed, a coffee is always eventually delivered.” (×)
  - “After the button is pressed, there exists an execution such that a coffee is eventually delivered.” (√)
Temporal Logics

- Specify properties on the order between events

- Example: CTL (Computation Tree Logic)

  [Clarke and Emerson, 1982]

  - “After the button is pressed, a coffee is always eventually delivered.” (✗)
  - “After the button is pressed, there exists an execution such that a coffee is eventually delivered.” (√)
  - “It is possible to get a coffee with 2 doses of sugar.”
Temporal Logics

- Specify properties on the order between events

- Example: **CTL** (Computation Tree Logic)  
  [Clarke and Emerson, 1982]
  - “After the button is pressed, a coffee is always eventually delivered.” (✗)
  - “After the button is pressed, there exists an execution such that a coffee is eventually delivered.” (✓)
  - “It is possible to get a coffee with 2 doses of sugar.” (✓)
Timed Automaton

- Finite state automaton (sets of locations)
Timed Automaton

- Finite state automaton (sets of locations and actions)

```plaintext
<x> := 0
<y> := 0
<y> = 5
<x> ⩾ 1
```
Timed Automaton

- 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

Diagram:

- Location with initial state (green)
- Transition labeled with "press?"
- Location with final state (red)
- Transition labeled with "cup!"
- Transition labeled with "coffee!"

Equations:

- $x := 0$
- $y := 0$
- $y = 5$
- $x \geq 1$
Timed Automaton

- 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

![Diagram of a Timed Automaton with states and transitions](image)
Timed Automaton

- 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

![Timed Automaton Diagram]

- $y = 8$
- coffee!
- $y \leq 5$
- press?
- $x \geq 1$
- press?
- $y = 5$
- cup!
Timed Automaton

- 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

---

$y = 8$

coffee!

---

$y \leq 5$

$y = 5$

cup!
Timed Runs

- $y = 8$
- Press?
- $x := 0$
- $y := 0$
- Press?
- $x := 0$
- $y = 5$
- Cup!
- $y \leq 5$
- $x \geq 1$

Examples of timed runs
Timed Runs

- \( y = 5 \) coffee!
- \( x \geq 1 \) press?
- \( x := 0 \)
- \( y := 0 \)

Examples of timed runs

- Coffee with no sugar

\[ x = 0 \]
\[ y = 0 \]
Timed Runs

- press? $x := 0$
- $y := 0$

- press? $x := 0$
- $y = 5$
  - cup!

- $y \leq 5$

- $y = 8$
  - coffee!

Examples of timed 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>0</td>
</tr>
</tbody>
</table>
Timed Runs

- \( y = 8 \) coffee!
- \( y \leq 5 \)
- \( x \geq 1 \) cup!
- \( y = 5 \)
- \( x = 0 \)

Examples of timed runs

- Coffee with no sugar

<table>
<thead>
<tr>
<th>press?</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>0</td>
</tr>
<tr>
<td>( y )</td>
<td>0</td>
</tr>
</tbody>
</table>
Timed Runs

y = 8
coffee!
y ≤ 5

press?
x := 0
y := 0
x ≥ 1
press?
x := 0
y = 5
cup!

- Examples of timed runs
  - Coffee with no sugar
    - press?
    - 5
    - cup!
    - x: 0 0 5 5
    - y: 0 0 5 5
Timed Runs

$y = 8$
coffe!

When $y \leq 5$

$y = 5$
cup!

$x := 0$
y := 0

$x \geq 1$

Examples of timed runs

- Coffee with no sugar

$x := 0$
$y := 0$

Press?

Press?

$5$
cup!

$3$

$5$

$5$

$8$

$8$

$x$

$y$
Timed Runs

- $y = 8$
- coffee!

$y \leq 5$

- press?
- $x := 0$
- $y := 0$

$y = 5$

- cup!
- $x \geq 1$
- press?
- $x := 0$

Examples of timed runs

- Coffee with no sugar

$\begin{array}{c|c|c|c|c|c}
\text{press?} & 5 & \text{cup!} & 3 & \text{coffee!} \\
\hline
x & 0 & 0 & 5 & 5 & 8 \\
y & 0 & 0 & 5 & 8 & 8 \\
\end{array}$
Timed Runs

\[ y = 8 \] coffee!

\[ y \leq 5 \]

press?

\[ x := 0 \]

\[ y := 0 \]

\[ x \geq 1 \] cup!

\[ y = 5 \]

press?

\[ x := 0 \]

\[ y := 0 \]

Examples of timed runs

- Coffee with no sugar

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( x )</th>
<th>( y )</th>
<th>( x )</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>
</tbody>
</table>

Timed Runs

Examples of timed runs

- Coffee with no sugar
  
- Coffee with 2 doses of sugar
Timed Runs

- **Examples of timed runs**
  - **Coffee with no sugar**
    - Press? \( x := 0 \)
    - \( x \geq 1 \)
    - \( y := 0 \)
    - \( y = 5 \)
    - Coffee!

  - **Coffee with 2 doses of sugar**
    - Press? \( x := 0 \)
    - \( x = 1.5 \)
    - Coffee!
Timed Runs

- $y = 8$
- **coffee!**

- $y \leq 5$

- press?
- $x := 0$
- $y := 0$

- press?
- $x \geq 1$
- $y = 5$
- **cup!**

- $x := 0$

Examples of timed 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>3</td>
<td>8</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>
<tr>
<td>1.5</td>
<td>1.5</td>
</tr>
</tbody>
</table>
Timed Runs

- \( y = 8 \) coffee!

- \( y \leq 5 \)

- \( y = 5 \) cup!

- \( x \geq 1 \)

- \( x = 0 \)

- \( y = 0 \)

Examples of timed runs

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

- **Coffee with 2 doses of sugar**
  - Press? \( x = 0 \)
  - 1.5
  - Press? \( x = 0 \)
  - 2.7
  - 1.5
  - 1.5
  - 4.2
Timed Runs

y = 8
coffee!

Examples of timed runs

Coffee with no sugar

Coffee with 2 doses of sugar
Timed Runs

- $y = 8$
  - coffee!

- $y \leq 5$
  - press?
    - $x := 0$
    - $y := 0$

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

Examples of timed runs

- Coffee with no sugar
  - $y = 8$
    - coffee!
  - $x = 0$
    - $y = 0$
  - $x = 5$
  - $y = 8$
  - $x = 8$
  - $y = 8$

- Coffee with 2 doses of sugar
  - $y = 8$
    - coffee!
  - $x = 0$
    - $y = 0$
  - $x = 1.5$
    - $y = 1.5$
  - $x = 2.7$
    - $y = 4.2$
  - $x = 0.8$
    - $y = 5$
### Timed Runs

- **y = 8**
  - Coffee!

- **y ≤ 5**

- **x := 0**
  - press?
  - y := 0
  - press?
  - x ≥ 1
  - y = 5
  - cup!

- **x := 0**

#### Examples of timed runs

- **Coffee with no sugar**

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

- **Coffee with 2 doses of sugar**

  - press? 1.5 press? 2.7 press? 0.8 cup!
  - 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
Timed Runs

- Examples of timed runs
  - Coffee with no sugar
    - Press? $x := 0$
    - $y := 0$
    - $y \leq 5$
    - Press? $x \geq 1$
    - $y = 5$
    - Coffee!
    - $x := 0$
    - $y = 8$
    - Coffee!

  - Coffee with 2 doses of sugar
    - Press? $x := 0$
    - $y := 0$
    - $y = 5$
    - Press? $x \geq 1$
    - $y = 8$
    - Coffee!
    - $x := 0$
    - $y = 8$
    - Coffee!

- $x \geq 1$
- $y \leq 5$
Timed Runs

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

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

Examples of timed runs

- Coffee with no sugar

\[ x \]
\[ y \]
\[ x := 0 \]
\[ y := 0 \]
\[ \text{press?} \]
\[ 5 \]
\[ \text{cup!} \]
\[ 3 \]
\[ \text{coffee!} \]

\[ x \]
\[ y \]
\[ 0 \]
\[ 0 \]
\[ 5 \]
\[ 5 \]
\[ 8 \]
\[ 8 \]

- Coffee with 2 doses of sugar

\[ x \]
\[ y \]
\[ 0 \]
\[ 0 \]
\[ 1.5 \]
\[ 1.5 \]
\[ 0 \]
\[ 2.7 \]
\[ 0 \]
\[ 0.8 \]
\[ 0.8 \]
\[ 3.8 \]
\[ 3.8 \]
Dense Time

- Time is **dense**: transitions can be taken anytime
  - Infinite number of timed runs
  - Model checking needs a **finite** structure!

- Some runs are equivalent
  - Taking the `press?` action at $t = 1.5$ or $t = 1.57$ is equivalent w.r.t. the possible actions

- Idea: reason with abstractions
  - Region automaton [Alur and Dill, 1994]
  - Example: in location $\blacklozenge$, all clock values in the following region are equivalent
    $$x \geq 1 \land y \leq 5 \land x = y$$
  - This abstraction is **finite**
Timed Temporal Logics

- Specify properties on the order and the delay between events

- Example: **TCTL** (Timed CTL) [Alur et al., 1993a]
  - “After the first time the button is pressed, a coffee is always eventually delivered within 10 units of time.”
Timed Temporal Logics

- Specify properties on the order and the delay between events

- Example: TCTL (Timed CTL) [Alur et al., 1993a]
  - “After the first time the button is pressed, a coffee is always eventually delivered within 10 units of time.” (√)
Timed Temporal Logics

- Specify properties on the order and the delay between events

- Example: TCTL (Timed CTL) [Alur et al., 1993a]
  - “After the first time the button is pressed, a coffee is always eventually delivered within 10 units of time.” (√)
  - “It must never happen that the button can be pressed twice within 1 unit of time.”
Timed Temporal Logics

- Specify properties on the order and the delay between events

- Example: TCTL (Timed CTL) [Alur et al., 1993a]
  - “After the first time the button is pressed, a coffee is always eventually delivered within 10 units of time.” (√)
  - “It must never happen that the button can be pressed twice within 1 unit of time.” (×)
Timed Temporal Logics

- Specify properties on the order and the delay between events

- Example: TCTL (Timed CTL) [Alur et al., 1993a]
  - “After the first time the button is pressed, a coffee is always eventually delivered within 10 units of time.” (✓)
  - “It must never happen that the button can be pressed twice within 1 unit of time.” (✗)
  - “It must never happen that the button can be pressed twice within a time strictly less than 1 unit of time.”
Timed Temporal Logics

- Specify properties on the order and the *delay* between events

- Example: **TCTL** (Timed CTL) [Alur et al., 1993a]
  - “After the first time the button is pressed, a coffee is always eventually delivered within 10 units of time.” (✓)
  - “It must never happen that the button can be pressed twice within 1 unit of time.” (✗)
  - “It must never happen that the button can be pressed twice within a time strictly less than 1 unit of time.” (✓)
Towards a Parametrization. . .

- **Interesting problems**
  - **Robustness**
    - Does the system still behave the same if one of the delays (slightly) changes?
  - **Optimization of timing constants**
    - Up to which value of the delay between two actions can I still order a coffee with 3 doses of sugar?
  - **Avoidance of numerous verifications**
    - If one of the timing delays of the model changes, should I model check again the whole system?
Towards a Parametrization...

- Interesting problems
  - Robustness
    - Does the system still behave the same if one of the delays (slightly) changes?
  - Optimization of timing constants
    - Up to which value of the delay between two actions press? can I still order a coffee with 3 doses of sugar?
  - Avoidance of numerous verifications
    - If one of the timing delays of the model changes, should I model check again the whole system?

- Idea: reason with parameters (unknown constants)
Outline

1. A Coffee Vending Machine
2. A Parametric Coffee Vending Machine
3. Synthesis of Parameters
4. Conclusion
A Parametric Coffee Vending Machine

Parametric Timed Automaton

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

```
y ≤ 5
coffee!
```

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

```
x ≥ 1
y = 5
cup!
```

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

```
x := 0
y := 5
press?
```

```
x := 0
y := 0
```
A Parametric Coffee Vending Machine

Parametric Timed Automaton

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

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

```
\begin{align*}
  y &= p_2 \\
  \text{cup!}
\end{align*}
```

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

```
\begin{align*}
  x &\geq p_1 \\
  \text{press?} \\
  x &:= 0
\end{align*}
```
Symbolic Exploration

- Iterative exploration of symbolic states
  - Symbolic state: location and constraint on the clocks and parameters
Symbolic Exploration: Coffee Machine

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

\[
\begin{align*}
y &\leq p_2 \\
\text{press?} \\
x &:= 0 \\
y &:= 0
\end{align*}
\]

\[
\begin{align*}
y &= p_2 \\
\text{cup!} \\
x &\geq p_1 \\
\text{press?} \\
x &:= 0
\end{align*}
\]

\[
\begin{align*}
x &= y
\end{align*}
\]
Symbolic Exploration: Coffee Machine

A Parametric Coffee Vending Machine

\[ y = p_3 \]

\[ y = p_2 \]

\[ \text{cup!} \]

\[ x := 0 \]

\[ y := 0 \]

\[ x \geq p_1 \]

\[ \text{press?} \]

\[ x := 0 \]

\[ y = p_3 \text{ coffee!} \]

\[ x = y \]

\[ 0 \leq y \leq p_2 \]

\[ \text{press?} \]

\[ x = y \]
Symbolic Exploration: Coffee Machine

\[ x = y \]

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

\[ \text{press?} \]

\[ y \leq p_2 \]

\[ x \geq p_1 \]

\[ \text{press?} \]

\[ x := 0 \]

\[ y = p_2 \]

\[ \text{cup!} \]

\[ x = y \]

- \[ 0 \leq y \leq p_2 \]

\[ \text{press?} \]

\[ y = p_2 \]

\[ y \geq p_2 \]

\[ \text{cup!} \]
Symbolic Exploration: Coffee Machine

\[ y = p_3 \]

\[ \text{coffee!} \]

\[ y \leq p_2 \]

\[ \text{press?} \]

\[ x := 0 \]

\[ y := 0 \]

\[ \text{press?} \]

\[ x \geq p_1 \]

\[ x := 0 \]

\[ y = p_2 \]

\[ \text{cup!} \]

\[ x = y \]

\[ \text{press?} \]

\[ 0 \leq y \leq p_2 \]

\[ \text{cup!} \]

\[ x = y \]

\[ y \geq p_2 \]

\[ \text{coffee!} \]

\[ x = y \]

\[ y \geq p_3 \]
Symbolic Exploration: Coffee Machine

\[ y = p_3 \]

coffee!

\[ y \leq p_2 \]

press?

\[ x := 0 \]

\[ y := 0 \]

cup!

\[ x \geq p_1 \]

press?

\[ x := 0 \]

\[ x = y \]

\[ 0 \leq y \leq p_2 \]

press?

\[ x = y \]

\[ y \geq p_2 \]

cup!

\[ x = y \]

\[ y \geq p_3 \]

coffee!
Symbolic Exploration: Coffee Machine

\[ y = p_3 \]

\[ \text{coffee!} \]

\[ y \leq p_2 \]

press?

\[ x := 0 \]

\[ y := 0 \]

\[ y = p_2 \]

\[ \text{cup!} \]

\[ x \geq p_1 \]

press?

\[ x := 0 \]

\[ x = y \]

\[ 0 \leq y \leq p_2 \]

press?

\[ x = y \]

\[ y \geq p_2 \]

cup!

\[ x = y \]

\[ y \geq p_3 \]

coffee!

\[ y - x \geq p_1 \]

\[ 0 \leq y \leq p_2 \]

press?

\[ \ldots \]

cup!
Symbolic Exploration: Coffee Machine

\[ y = p_3 \]
\[ \text{coffee!} \]

\[ y \leq p_2 \]

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

\[ x \geq p_1 \]
\[ \text{press?} \]
\[ x := 0 \]

\[ x = y \]
\[ 0 \leq y \leq p_2 \]

\[ x = y \]
\[ y \geq p_2 \]

\[ x = y \]
\[ y \geq p_3 \]

\[ x = y \]
\[ 0 \leq y \leq p_2 \]

\[ y - x \geq p_1 \]
\[ 0 \leq y \leq p_2 \]

\[ y - x \geq 2p_1 \]
\[ 0 \leq y \leq p_2 \]

\[ \text{press?} \]

\[ \text{cup!} \]

\[ \text{coffee!} \]
Symbolic Exploration: Coffee Machine

\[ y = p_3 \]
\[ \text{coffee!} \]

\[ y \leq p_2 \]

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

\[ x \geq p_1 \]
\[ \text{press?} \]
\[ x := 0 \]

\[ y = p_2 \]
\[ \text{cup!} \]

\[ x = y \]
\[ 0 \leq y \leq p_2 \]

\[ x = y \]
\[ y \geq p_2 \]

\[ x = y \]
\[ y \geq p_3 \]

\[ y - x \geq p_1 \]
\[ 0 \leq y \leq p_2 \]

\[ y - x \geq 2p_1 \]
\[ 0 \leq y \leq p_2 \]

\[ \text{press?} \]
\[ \text{cup!} \]
\[ \text{coffee!} \]
Undecidability

- The symbolic exploration is infinite in general
- No possible abstraction like for Timed Automata
Undecidability

- The symbolic exploration is infinite in general
- No possible abstraction like for Timed Automata

Bad News

(Almost) all interesting problems are undecidable for Parametric Timed Automata.
Outline

1. A Coffee Vending Machine
2. A Parametric Coffee Vending Machine
3. Synthesis of Parameters
4. Conclusion
The good parameters problem

“Given a bounded parameter domain, find a set of parameter valuations of good behavior”
The good parameters problem

“Given a bounded parameter domain, find a set of parameter valuations of good behavior”
Synthesis of Parameters (1/2)

- The good parameters problem
  - “Given a bounded parameter domain, find a set of parameter valuations of good behavior”

- Interesting problem
The good parameters problem

“Given a bounded parameter domain, find a set of parameter valuations of good behavior”

Interesting problem

But undecidable!
Possible options to deal with undecidability

- **Semi algorithms**
  - Example: Computation of all the reachable states, and intersection with the bad states [Henzinger and Wong-Toi, 1996]

- **Approximations**
  - Example: Use of octahedra [Clarisó and Cortadella, 2007]

- **Restrictions to decidable subclasses**
  - Example: L/U automata [Hune et al., 2002]
An Inverse Method for PTAs

- Original method for synthesis of parameters [André et al., 2009]
  - “Given a reference parameter valuation $\pi_0$, find other valuations around $\pi_0$ of same (linear-time) behavior”

- Input
  - $p_1 = 1$
  - $p_2 = 5$
  - $p_3 = 8$
An Inverse Method for PTAs

- Original method for synthesis of parameters [André et al., 2009]
  - “Given a reference parameter valuation $\pi_0$, find other valuations around $\pi_0$ of same (linear-time) behavior”

Input
- $p_1 = 1$
- $p_2 = 5$
- $p_3 = 8$

Output
- $p_3 \geq p_2$
- $\land 6p_1 > p_2$
- $\land p_2 \geq 5p_1$
An Inverse Method for PTAs

- Original method for synthesis of parameters [André et al., 2009]
  - “Given a reference parameter valuation $\pi_0$, find other valuations around $\pi_0$ of same (linear-time) behavior”

\[
\begin{align*}
\text{Input} & \quad \text{Output} \\
p_1 &= 1 & p_3 & \geq p_2 \\
p_2 &= 5 & \land 6p_1 & > p_2 \\
p_3 &= 8 & \land p_2 & \geq 5p_1
\end{align*}
\]

- Properties
  - Semi-algorithm
  - Not complete

- Advantages
  - Exact method
  - Behaves well in practice!
Behavioral Cartography of Timed Automata (1/2)

- Idea: repeatedly call the inverse method
  - [André and Fribourg, 2010]
  - Cover the parametric space with tiles

- Parametric zones with uniform behavior

Example: Root Contention Protocol
Idea: repeatedly call the inverse method
[André and Fribourg, 2010]

$\leadsto$ Cover the parametric space with tiles

- Parametric zones with uniform behavior

Example: Root Contention Protocol
Behavioral Cartography of Timed Automata (1/2)

- Idea: repeatedly call the inverse method
  [André and Fribourg, 2010]
  \[\sim\] Cover the parametric space with tiles

  - Parametric zones with uniform behavior

- Remarks
  - Tiles may be infinite
  - The cartography may not cover the whole real-valued space

Example: Root Contention Protocol
Partition into good and bad subspaces

- **Good** if the property is true
- **Bad** if the property is false
- **Unknown** if not covered by any tile

---

Behavioral Cartography of Timed Automata (2/2)
Partition into good and bad subspaces
- **Good** if the property is true
- **Bad** if the property is false
- Unknown if not covered by any tile

Advantages
- **Independent** of the property
  (Only the partition depends on the property)
- Covers much of the parametric space in practice
Summary

- **Synthesis of parameters for real-time concurrent systems**
  - Important
    - Robustness
    - Optimization of timing constants
    - Avoidance of numerous verifications

- **Undecidable in the general case**
  - Possible solutions
    - Semi algorithms
    - Approximations
    - Restrictions to decidable subclasses

- Still ongoing work!
References I


Summary of Experiments: IM

- Computation times of various case studies
- Experiments conducted on an Intel Core2 Duo 2.4 GHz with 2 Gb

| Example          | PTAs | loc./PTA | $|X|$ | $|P|$ | iter. | $|K_0|$ | states | trans. | Time  |
|------------------|------|----------|------|------|-------|--------|--------|--------|-------|
| SR-latch         | 3    | [3, 8]   | 3    | 3    | 5     | 2      | 4      | 3      | 0.007 |
| Flip-flop        | 5    | [4, 16]  | 5    | 12   | 9     | 6      | 11     | 10     | 0.122 |
| And–Or           | 3    | [4, 8]   | 4    | 12   | 14    | 4      | 13     | 13     | 0.15  |
| Latch circuit    | 7    | [2, 5]   | 8    | 13   | 12    | 6      | 18     | 17     | 0.345 |
| CSMA/CD          | 3    | [3, 8]   | 3    | 3    | 19    | 2      | 219    | 342    | 1.01  |
| RCP              | 5    | [6, 11]  | 6    | 5    | 20    | 2      | 327    | 518    | 2.3   |
| BRP              | 6    | [2, 6]   | 7    | 6    | 30    | 7      | 429    | 474    | 34    |
| SIMOP            | 5    | [5, 16]  | 8    | 7    | 53    | 9      | 1108   | 1404   | 67    |
Summary of Experiments: *BC*

- Computation time for the cartography algorithm
- Experiments conducted on an Intel Core2 Duo 2.4 GHz with 2 Gb

| Example      | PTAs | loc./PTA | |X| | |P| | |V₀| | tiles | states | trans. | Time (s) |
|--------------|------|----------|----|---|---|---|---|---|---|----|------|------|------|------|
| SR-latch     | 3    | [3, 8]   | 3  | 3 |   |   |   |   |   | 6   | 5    | 4    | 0.3  |
| Flip-flop    | 5    | [4, 16]  | 5  | 2 |   |   |   |   |   | 8   | 15   | 14   | 3    |
| Latch circuit| 7    | [2, 5]   | 8  | 4 |   |   |   |   |   | 5   | 21   | 20   | 96.3 |
| And–Or       | 3    | [4, 8]   | 4  | 6 |   |   |   |   |   | 4   | 64   | 72   | 118  |
| CSMA/CD      | 3    | [3, 8]   | 3  | 3 |   |   |   |   |   | 140 | 349  | 545  | 269  |
| RCP          | 5    | [6, 11]  | 6  | 3 |   |   |   |   |   | 19  | 5688 | 9312 | 7018 |