Enhanced Distributed Behavioral Cartography of Parametric Timed Automata

Étienne André, Camille Coti, Hoang Gia Nguyen

LIPN, UMR 7030, Université Paris 13, Sorbonne Paris Cité, CNRS, France
Context: Formal Verification of Timed Systems

- Use formal methods

A model of the system

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

- Use formal methods

\[ \text{A model of the system} \quad \text{A property to be satisfied} \]

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

- Use formal methods

A model of the system

A property to be satisfied

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

Yes

No

Counterexample
Outline

1 Behavioral Cartography of Timed Automata
2 Distributing BC
3 Point-based Distribution Algorithms
4 Domain-based Distribution Algorithms
5 Experimental Validation
6 Conclusion and Perspectives
Outline

1 Behavioral Cartography of Timed Automata

2 Distributing BC

3 Point-based Distribution Algorithms

4 Domain-based Distribution Algorithms

5 Experimental Validation

6 Conclusion and Perspectives
Parametric Timed Automata (PTA)

A formalism to model and verify concurrent real-time systems
[Alur et al., 1993]

$x$: Clock

$p$: Parameters allow to represent unknown values (e.g., a transmission delay or a timeout)

Trace set = set of all sequences of (untimed) actions
**Behavioral Cartography (BC)**

- **BC**: Partitions a parameter domain (bounded rectangle) into tiles, i.e., parametric zones of uniform behavior [André and Fribourg, 2010]
- **Method**: Enumerate integer points and generate a tile using an existing algorithm (the inverse method IM)
- All parameter valuations in a tile have the same possible behaviors (same “trace set”), and verify the same linear-time properties
**Behavioral Cartography (BC)**

- **BC**: Partitions a parameter domain (bounded rectangle) into tiles, i.e., parametric zones of uniform behavior [André and Fribourg, 2010]
- **Method**: Enumerate integer points and generate a tile using an existing algorithm (the inverse method IM)
- All parameter valuations in a tile have the same possible behaviors (same “trace set”), and verify the same linear-time properties
Behavioral Cartography (BC)

- **BC**: Partitions a parameter domain (bounded rectangle) into tiles, i.e., parametric zones of uniform behavior [André and Fribourg, 2010]
- **Method**: Enumerate integer points and generate a tile using an existing algorithm (the inverse method IM)
- **All parameter valuations in a tile have the same possible behaviors (same “trace set”), and verify the same linear-time properties**
Behavioral Cartography (BC)

- **BC**: Partitions a parameter domain (bounded rectangle) into tiles, i.e., parametric zones of uniform behavior [André and Fribourg, 2010]
- **Method**: Enumerate integer points and generate a tile using an existing algorithm (the inverse method IM)
- All parameter valuations in a tile have the same possible behaviors (same “trace set”), and verify the same linear-time properties
Behavioral Cartography (BC)

- **BC:** Partitions a parameter domain (bounded rectangle) into tiles, i.e., parametric zones of uniform behavior [André and Fribourg, 2010]
- **Method:** Enumerate integer points and generate a tile using an existing algorithm (the inverse method IM)
- All parameter valuations in a tile have the same possible behaviors (same “trace set”), and verify the same linear-time properties
Behavioral Cartography (BC)

- **BC**: Partitions a parameter domain (bounded rectangle) into tiles, i.e., parametric zones of uniform behavior [André and Fribourg, 2010]
- **Method**: Enumerate integer points and generate a tile using an existing algorithm (the inverse method IM)
- All parameter valuations in a tile have the same possible behaviors (same “trace set”), and verify the same linear-time properties
Behavioral Cartography (BC)

- **BC**: Partitions a **parameter domain** (bounded rectangle) into **tiles**, i.e., parametric zones of uniform behavior [André and Fribourg, 2010]

- **Method**: Enumerate integer points and generate a tile using an existing algorithm (the inverse method **IM**)

- All parameter valuations in a tile have the **same possible behaviors** (same “trace set”), and verify the **same linear-time properties**
Behavioral Cartography (BC)

- **BC**: Partitions a parameter domain (bounded rectangle) into tiles, i.e., parametric zones of uniform behavior [André and Fribourg, 2010]

- **Method**: Enumerate integer points and generate a tile using an existing algorithm (the inverse method IM)

- All parameter valuations in a tile have the same possible behaviors (same “trace set”), and verify the same linear-time properties
Behavioral Cartography (BC) Problem

- But state space explosion is always painful! Especially for real-time systems.
Behavioral Cartography (BC) Problem

- But state space explosion is always painful! Especially for real-time systems.

- One solution:
  - Extend to distributed fashion
Outline

1. Behavioral Cartography of Timed Automata
2. Distributing BC
3. Point-based Distribution Algorithms
4. Domain-based Distribution Algorithms
5. Experimental Validation
6. Conclusion and Perspectives
Distributing BC

- Problem: **BC** is very slow! (up to several hours)
- Goal: distribute **BC** on a cluster to increase the computation speed
Distributing BC: Problems

The general shape of the Cartography is unknown and the time to compute each tile varies a lot (more or less complex trace sets) → Load balancing problem

Since tiles are unpredictable, two close points will very probably yield the same tile → Choosing point problem
Outline

1. Behavioral Cartography of Timed Automata
2. Distributing BC
3. Point-based Distribution Algorithms
4. Domain-based Distribution Algorithms
5. Experimental Validation
6. Conclusion and Perspectives
Master Workers Scheme

Traditional Master-Worker communication scheme: [André, Coti, Evangelista, 2014]

- **Workers**: ask the master for a point, and send the result ("tiles") to the master
- **Master**: is responsible for smart repartition of data between the workers
Point-based **BC** Algorithms

1. **Sequential**: Each point is sent to a worker sequentially.
2. **Random**: Points selected *randomly*, then switches to **Sequential**.
3. **Shuffle**: Similar to the **Sequential**, but the master must *statically compute* the array of all points, then shuffle all points, then store them back in array *(new)*.
Outline

1. Behavioral Cartography of Timed Automata
2. Distributing BC
3. Point-based Distribution Algorithms
4. Domain-based Distribution Algorithms
5. Experimental Validation
6. Conclusion and Perspectives
Domain-based BC Algorithm Scheme

- **Master**
  1. initially splits the parameter domain into sub-domains and send them to the workers
  2. when a worker has completed its sub-domain, the master splits another sub-domain, and sends it to the idle worker

- **Workers**
  1. receives the sub-domain from the master
  2. calls IM on the points of this sub-domain
  3. sends the results (tiles) back to the master
  4. asks for more work
Domain-based Distribution Algorithms

Static:
- One of Workers splits the domain then sends to other workers and gathers result of all workers
- **Drawback:** computing time depends on the slowest process

Dynamic:
- **Master** is only responsible for gathering tiles and splitting domain/sub-domains
- Master monitors all Workers then it can balance workload (by splitting) between workers
Domain-based BC Algorithms

Static:
- One of Workers splits the domain then sends to other workers and gathers result of all workers
- Drawback: computing time depends on the slowest process

Dynamic:
- Master is only responsible for gathering tiles and splitting domain/sub-domains
- Master monitors all Workers then it can balance workload (by splitting) between workers
Domain-based BC Algorithms

Static:
- One of Workers splits the domain then sends to other workers and gathers result of all workers
- Drawback: computing time depends on the slowest process

Dynamic:
- Master is only responsible for gathering tiles and splitting domain/sub-domains
- Master monitors all Workers then it can balance workload (by splitting) between workers
Redundancy Detection – Heuristic (For Dynamic Only)

- **Redundancy detection**: a mechanism to detect and stop process which calling point is in a covered tile

- **Example**: better to stop immediately when the reference point (“π₂”) is covered by another tile (“K₁”). Note that: There is non-determinism, so a same point can yield different tiles
Outline

1 Behavioral Cartography of Timed Automata
2 Distributing BC
3 Point-based Distribution Algorithms
4 Domain-based Distribution Algorithms
5 Experimental Validation
6 Conclusion and Perspectives
Implementation in IMITATOR

- **IMITATOR** [André, Fribourg, Kühne, Soulat, 2012]
  - 26,000 lines of OCaml code
    - Including > 3,000 lines for the distribution algorithms
  - Relies on the PPL library for operations on polyhedra [Bagnara et al., 2008]
  - Available under the GNU-GPL license at www.imitator.fr

- Distributed version of IMITATOR relying on MPI
  - Using the OcamlMPI library for passing messages between Master and Workers

- Implementation of the new algorithms: not trivial at all in practice
Experimental Validation

Experimental conducted on a 2 clusters of Grid’5000:

1. **Pastel** (Located in Toulouse, FR): 140 nodes
2. **Griffon** (Located in Nancy, FR): 92 nodes

<table>
<thead>
<tr>
<th>Each node’s</th>
<th>Pastel</th>
<th>Griffon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processors</td>
<td>2x Dual-core AMD Opteron <a href="mailto:2218@2.6GHz">2218@2.6GHz</a></td>
<td>2x Quad-core Intel Xeon <a href="mailto:L5420@2.5GHz">L5420@2.5GHz</a></td>
</tr>
<tr>
<td>Memory (GB)</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>Interconnection network</td>
<td>GigaEthernet</td>
<td>GigaEthernet + 20G InfiniBand</td>
</tr>
</tbody>
</table>
### Experiment Summary

<table>
<thead>
<tr>
<th>Case study</th>
<th>Flip-flop4</th>
<th>RCP</th>
<th>Sched3-2</th>
<th>Sched3B-2</th>
<th>Sched3B-3</th>
<th>Sched5</th>
<th>SiMoP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model</strong></td>
<td><strong>Clocks</strong></td>
<td><strong>Parameters</strong></td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>21</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>386400</td>
<td>3050</td>
<td>286</td>
<td>14746</td>
<td>530856</td>
<td>1681</td>
<td>10201</td>
<td></td>
</tr>
<tr>
<td><strong>Cartography</strong></td>
<td><strong># Tiles</strong></td>
<td><strong>N_{max}</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>190</td>
<td>19</td>
<td>59</td>
<td>71</td>
<td>378</td>
<td>177</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td>128</td>
<td>64</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Execution time at N_{max}</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Static</td>
<td>33.0</td>
<td>2108.0</td>
<td>4.0</td>
<td>26.6</td>
<td>181.0</td>
<td>213.0</td>
<td>21.4</td>
</tr>
<tr>
<td>Seq</td>
<td>2059.0</td>
<td>653.0</td>
<td>4.6</td>
<td>11.0</td>
<td>810.0</td>
<td>219.0</td>
<td>36.1</td>
</tr>
<tr>
<td>Random</td>
<td>652.0</td>
<td>635.0</td>
<td>3.6</td>
<td>8.4</td>
<td>524.0</td>
<td>148.0</td>
<td>23.6</td>
</tr>
<tr>
<td>Shuffle</td>
<td>670.0</td>
<td>624.0</td>
<td>3.1</td>
<td>7.6</td>
<td>243.0</td>
<td>140.0</td>
<td>18.7</td>
</tr>
<tr>
<td>Subdomain</td>
<td>48.0</td>
<td>1286.0</td>
<td>7.2</td>
<td>15.8</td>
<td>217.0</td>
<td>273.0</td>
<td>32.4</td>
</tr>
<tr>
<td>Subdomain + H</td>
<td>24.0</td>
<td>622.0</td>
<td>4.0</td>
<td>11.0</td>
<td>81.0</td>
<td>199.0</td>
<td>23.2</td>
</tr>
<tr>
<td>Hybrid</td>
<td>24.0</td>
<td>624.0</td>
<td>3.1</td>
<td>7.6</td>
<td>81.0</td>
<td>140.0</td>
<td>18.7</td>
</tr>
</tbody>
</table>

- **Shuffle** and **Subdomain + H** (Heuristic) are the fastest algorithms.
- **Hybrid:**
  - <100k points: Uses **Shuffle**
  - >100k points: Uses **Subdomain + H**
Outline

1. Behavioral Cartography of Timed Automata
2. Distributing BC
3. Point-based Distribution Algorithms
4. Domain-based Distribution Algorithms
5. Experimental Validation
6. Conclusion and Perspectives
Conclusion and Perspectives

- Conclusion:
  - Proposed a new efficient distributed algorithm for Behavioral Cartography
  - Proposed a new heuristic approach improving all BC distribution algorithms
  - Implemented the new algorithms in IMITATOR

- Future works:
  - Design an autonomous distribution scheme for BC
    - No master!
  - Try BC in GPU’s or CPU+GPU’s environment
  - ... and formally prove the deadlock-freeness of our master-worker communication scheme!
Bibliography
References I

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

Distributed behavioral cartography of timed automata.

André, É. and Friboung, L. (2010).
Behavioral cartography of timed automata.

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

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

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

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

Title: Polyhedron
Author: Robert Webb
Source: http://commons.wikimedia.org/wiki/File:Uniform_polyhedron-53-s012.png
License: public domain

Title: MPI logo
Author: Unknown
Source: http://www.open-mpi.org
License: Unknown
Source of the graphics used II

Title: Ocaml logo  
Author: Amir Chaudhry  
Source: https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg  
License: CC BY-SA 4.0

Title: IMITATOR logo (Typing Monkey)  
Author: Kater Begemot  
Source: https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg  
License: CC BY-SA 3.0

Title: Logo Grid’5000  
Author: Unknown  
Source: https://www.lri.fr/~fci/Grid5000/images/logo.jpg  
License: Unknown

Title: Grid’5000  
Author: Unknown  
Source: https://www.grid5000.fr/mediawiki/index.php/File:Renater5-g5k.jpg  
License: Unknown
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)

Authors: Nguyen Hoang Gia and Étienne André

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