Model-based Runtime Verification Framework

Yuhong Zhao
Heinz Nixdorf Institute
University of Paderborn
Germany
Outline

- Motivation
  - Model-driven Engineering
  - Verification and validation techniques

- Model-based runtime verification framework
  - Problem Statement
  - Pipelined working principle
  - Model Checking Methodology
  - Game between Runtime Verification and System Execution
    - Pre-checking and post-checking
    - Speedup strategies
      - Enrich system model with probabilities
      - Enrich system model with additional information

- Conclusion
Motivation

- Model-driven Engineering (MDE)
  - Model system according to system specification
  - Verify system model against system specification
  - Synthesize system implementation (source code) from system model
Motivation

- Model-driven Engineering (MDE)
  - Model system according to system specification
  - Verify system model against system specification
  - Synthesize system implementation (source code) from system model
Motivation

- Verification and Validation Techniques
  - **Offline** Methods:
    - Model Checking (theorem proving)
      - Check all of the system behaviors
    - Simulation and Testing
      - Check some of the system behaviors
Motivation

- Verification and Validation Techniques
  - On-line Methods:
    - State-of-the-art runtime verification

[Diagram showing the process of system specification, runtime verification, and system model with arrows indicating Consistency Checking, Model Checking, Synthesize, and System implementation.]
Motivation

- Verification and Validation Techniques
  - On-line Methods:
    - State-of-the-art runtime verification
    - Model-based runtime verification
Model-based Runtime Verification Framework
Problem Statement

- As service of Real-time Operating System (RTOS)
- Application scenario
  Consider a real-time system model $M$ that
  ✓ contains $n$ modules: $M_1, M_2, \ldots, M_n$ working in parallel
  ✓ does reconfiguration at runtime by
    - case 1: $M - M_i$ (remove an existing module $M_i$)
    - case 2: $M + M_i'$ (add a new module $M_i'$)

Requirements:
- Send reconfiguration request in advance to RTOS (at time instant $t_r$)

Goal:
- Get answer before the reconfiguration is really done (at time instant $t_0 > t_r$) from runtime verification service
Model-based Runtime Verification Framework
Overview

Safety Checking

Consistency Checking

Büchi automaton

conform?

Abstract State

mapping function $\sigma$: abstract state $\leftarrow$ concrete state

Concrete State

conform?

Source Code

Execution Trace

Safety Property

System Model

Real-time ACTL/LTL

Real-time OCL Constraint

Real-time UML Model

FSM Model
Model-based Runtime Verification Framework

Basic Idea

Goal:
checking safety and consistency by looking ahead a subspace in system model that covers execution trace
Model-based Runtime Verification Framework
Pipelined Working Principle

Suppose

- Components and Protocols between Components are checked correct at design phase
- Implementations of the systems conform to the corresponding models
- Properties to be checked are ACTL and LTL formulas
- Processing speed of the verification is faster enough than that of the application
Model-based Runtime Verification Framework

ACTL/LTL Model Checking

Kripke structure of model $M$

Büchi Automaton of formula $f$

$G_i^m(0,t_i)$

$G_i^f(0,t_i)$

Time

$t_0 \leq t_d \leq t_c$

Note:

1. “$\Diamond$” stands for “$\leq$” (Simulation relation) for ACTL Model Checking; “$\Diamond$” stands for “$|=\$” (Satisfaction relation) for LTL Model Checking.

2. $G_i^m(0,t_i)$ ($1 \leq i \leq k$) denotes the subgraph of the Kripke structure (system model) reachable from initial states within $\Delta t_i$ steps.

3. $G_i^f(0,t_i)$ ($1 \leq i \leq k$) denotes the subgraph of the Büchi automaton (ACTL/LTL formula) composible with $G_i^m(0,t_i)$.

4. $t_d$ is the timing constraint required for verification.

5. $t_c$ is the minimum time difference between verification and application.
Goal: make runtime verification in pre-checking mode for as long time as possible in course of system execution
Game between Runtime Verification and System Execution: Speedup Strategies

- Enrich system model with probabilities

![Diagram showing concrete and abstract state spaces with simulation traces and critical states.](image-url)
- Enrich system model with probabilities
  ✓ Intentionally reduce state space to be explored
Enrich system model with additional information
**Conclusion**

- Model-driven Engineering
  - System specification $\rightarrow$ System model $\rightarrow$ System implementation

- Verification and Validation Techniques
  - **Off-line methods**
    - Model-checking
      - Check all of the system behaviors
  - Simulation and testing
    - Check some of the system behaviors
  - **On-line methods**
    - State-of-the-art runtime verification
      - System implementation $\rightarrow$ System specification
    - Model-based runtime verification
      - System implementation $\rightarrow$ system model $\rightarrow$ system specification
Thank You for Your Attention

Question?
Advice?
### Runtime invariant checking

<table>
<thead>
<tr>
<th>Modell</th>
<th>Typ</th>
<th>Zustand</th>
<th>Transition</th>
<th>Durch. Grad</th>
<th>Maximaler Ausg. Grad</th>
<th>BFS Höhe</th>
<th>Max. Stack</th>
<th>Bootsche Variable</th>
<th>Transition Schritt (ms)</th>
<th>Min. Vor. ausschau</th>
<th>Max. Vor. ausschau</th>
<th>Dur. Vor. ausschau</th>
</tr>
</thead>
<tbody>
<tr>
<td>sorter_1</td>
<td>Controller</td>
<td>20544</td>
<td>30697</td>
<td>1,5</td>
<td>5</td>
<td>198</td>
<td>617</td>
<td>36</td>
<td>1</td>
<td>40</td>
<td>299</td>
<td>103</td>
</tr>
<tr>
<td>collision_1</td>
<td>Kommunikations protokoll</td>
<td>55993</td>
<td>10792</td>
<td>1,9</td>
<td>5</td>
<td>57</td>
<td>617</td>
<td>25</td>
<td>1</td>
<td>26</td>
<td>81</td>
<td>48,7</td>
</tr>
<tr>
<td>synapse_2</td>
<td>Protokoll</td>
<td>61048</td>
<td>125334</td>
<td>2,1</td>
<td>18</td>
<td>41</td>
<td>2349</td>
<td>46</td>
<td>1</td>
<td>7</td>
<td>28</td>
<td>21,5</td>
</tr>
<tr>
<td>driving_phils_2</td>
<td>Mutual exclusion algorithm</td>
<td>33173</td>
<td>61064</td>
<td>2,5</td>
<td>9</td>
<td>150</td>
<td>5702</td>
<td>27</td>
<td>1</td>
<td>31</td>
<td>97</td>
<td>65,7</td>
</tr>
<tr>
<td>blocks_1</td>
<td>Planung und Scheduling</td>
<td>7057</td>
<td>18552</td>
<td>2,6</td>
<td>6</td>
<td>19</td>
<td>4263</td>
<td>23</td>
<td>1</td>
<td>8</td>
<td>21</td>
<td>14</td>
</tr>
<tr>
<td>peterson_1</td>
<td>Mutual Exclusion Algorithm</td>
<td>12456</td>
<td>33369</td>
<td>2,7</td>
<td>5</td>
<td>54</td>
<td>1862</td>
<td>30</td>
<td>1</td>
<td>13</td>
<td>36</td>
<td>31,7</td>
</tr>
<tr>
<td>szymanski_1</td>
<td>Mutual Exclusion Algorithm</td>
<td>20264</td>
<td>58701</td>
<td>2,8</td>
<td>3</td>
<td>72</td>
<td>2084</td>
<td>27</td>
<td>1</td>
<td>13</td>
<td>90</td>
<td>49,7</td>
</tr>
<tr>
<td>hanoi_1</td>
<td>Puzzle</td>
<td>6561</td>
<td>19680</td>
<td>3</td>
<td>3</td>
<td>256</td>
<td>4376</td>
<td>36</td>
<td>1</td>
<td>56</td>
<td>103</td>
<td>75,9</td>
</tr>
<tr>
<td>iprotocol_2</td>
<td>Kommunikations protokoll</td>
<td>2994</td>
<td>100489</td>
<td>3,4</td>
<td>7</td>
<td>91</td>
<td>443</td>
<td>39</td>
<td>1</td>
<td>18</td>
<td>451</td>
<td>50</td>
</tr>
<tr>
<td>phil_3</td>
<td>Mutual Exclusion Algorithm</td>
<td>729</td>
<td>2916</td>
<td>4</td>
<td>6</td>
<td>17</td>
<td>518</td>
<td>18</td>
<td>1</td>
<td>156</td>
<td>357</td>
<td>265</td>
</tr>
<tr>
<td>cyclic_scheduler_1</td>
<td>Protokoll</td>
<td>4007</td>
<td>20430</td>
<td>4,4</td>
<td>8</td>
<td>55</td>
<td>1819</td>
<td>40</td>
<td>1</td>
<td>23</td>
<td>437</td>
<td>278</td>
</tr>
<tr>
<td>rushhour_1</td>
<td>Puzzle</td>
<td>1048</td>
<td>5448</td>
<td>5,2</td>
<td>9</td>
<td>73</td>
<td>555</td>
<td>28</td>
<td>1</td>
<td>66</td>
<td>245</td>
<td>150,7</td>
</tr>
<tr>
<td>rushhour_2</td>
<td>Puzzle</td>
<td>2242</td>
<td>12633</td>
<td>5,6</td>
<td>10</td>
<td>80</td>
<td>906</td>
<td>32</td>
<td>1</td>
<td>36</td>
<td>408</td>
<td>116,4</td>
</tr>
<tr>
<td>pouring_1</td>
<td>Puzzle</td>
<td>503</td>
<td>4481</td>
<td>8,9</td>
<td>9</td>
<td>13</td>
<td>348</td>
<td>16</td>
<td>1</td>
<td>42</td>
<td>101</td>
<td>71,9</td>
</tr>
<tr>
<td>reader_writer_2</td>
<td>Protokoll</td>
<td>4104</td>
<td>43190</td>
<td>12</td>
<td>18</td>
<td>13</td>
<td>4097</td>
<td>25</td>
<td>1</td>
<td>4</td>
<td>16</td>
<td>9,3</td>
</tr>
<tr>
<td>pouring_2</td>
<td>Puzzle</td>
<td>51624</td>
<td>1232712</td>
<td>23,9</td>
<td>25</td>
<td>16</td>
<td>44509</td>
<td>18</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

**Note:**
1) Transition in system model represents 1 millisecond;
2) Platform: Linux, Pentium 4 processor 3.00 Ghz, 1 G RAM.
Runtime LTL Checking

- Model: the driving philosophers
- LTL formula: $G(ac0 \rightarrow Fgr0)$ --- If process 0 requests a resource it will be granted to him eventually

Note:
1) Transition in system model represents 1 millisecond;
2) Platform: Linux, Pentium 4 processor 3.00 Ghz, 1 G RAM.