Practical Verification of OSEK/VDX Operating System (Updated)

Toshiaki Aoki
School of Information Science/
Research Center for Software Verification
JAIST
Motivation

- Recently, the safety and reliability of automotive systems are becoming a big concern of our society.
  - Functional safety standards (IEC61508, ISO26262)
  - Unintended acceleration problem of Toyota's cars (suspect on electronic throttle control systems).
- We are working on practically applying formal methods to automotive systems.
  - Mainly focus on automotive operating systems conforming to OSEK/VDX (shortly, OSEK OS).
  - The operating systems are considered as important components in the evaluation of the safety of the automotive systems.
Objective

• Providing a high quality OS by applying formal methods.
  • Target: RX-OSEK850
    • developed by Renesas Micro Systems (RMS)
    • already used in current series of cars.

• Joint research project.
  • JAIST and DENSO since 2006.
    • DENSO is a user of RX-OSEK850.
  • RMS joined in 2009.
    • RMS is developing RX-OSEK850.
Approach

- We focus on design verification and testing.
  - Design verification: the design of RX-OSEK850 is verified by a model checking tool Spin.
  - Testing: we automatically generate test cases from the verified design.

```
 RX-OSEK850 Simulator
 Test Program                      .......... 
 RX-OSEK850                        Simulator
```

Sufficiently verify the design model

Putting an assumption that the design model is correct

Design model = Test oracle
Design of RX-OSEK850

- We focus on the scheduler of RX-OSEK850.
  - The most of functionalities provided by OSEK OS is to schedule tasks and ISRs.
  - The scheduler computes a task or ISR to be dispatched based on some data such as task queue, TCB and tables.
- It is easy to describe the scheduler in Promela.
  - Those data and computation can be straightforwardly described in Promela.
Design Verification

- OSEK OS is an open system.
  - OSEK OS does scheduling of the tasks if it gets stimulus such as system call invocations.
  - OSEK OS does not do anything if it does not get any stimulus.
- We need the description of the outside of OSEK OS to verify the design model.
  - The outside consists of multi-task applications which invoke the system calls and hardware which causes the interrupts.
- The outside of the verification target is called an environment.
Various environments can be considered.

- The number of tasks, the variation of priorities, number of the resources, invocation relations of the system calls, and so on.
- It is impossible to make all the descriptions of the environments manually.
- If all the environments are realized by one Promela description, that may cause state explosion problem.

We model the variations of the environments, then automatically generate the Promela descriptions of the environments.
Design Verification

- We constructed multiple environment models for multi-viewpoint analysis.
  - Separation of concerns of the verification.
    - task lifecycles, event synchronization, multiple activation, and so on.
  - Making the environment model simple and confident.
    - It should be more confident than the design model in an engineering sense.
- Environment generation.
  - Total: 786, Total Time: 81.4s, 0.1s per an environment
- Model checking.
  - Total Time: 8819.3s, 11s for each of them.
  - Almost of all of the time is for compiling pan.c.

<table>
<thead>
<tr>
<th>No.</th>
<th>Environment Model</th>
<th>Generated Environments</th>
<th>model checking</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>number</td>
<td>time(s)</td>
<td>lines</td>
</tr>
<tr>
<td>1</td>
<td>TaskDiff</td>
<td>26</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>TaskEq</td>
<td>14</td>
<td>0.4</td>
</tr>
<tr>
<td>3</td>
<td>CtDiff</td>
<td>26</td>
<td>0.6</td>
</tr>
<tr>
<td>4</td>
<td>CtEq</td>
<td>14</td>
<td>0.5</td>
</tr>
<tr>
<td>5</td>
<td>MultDiff</td>
<td>26</td>
<td>0.7</td>
</tr>
<tr>
<td>6</td>
<td>MultEq</td>
<td>14</td>
<td>1.6</td>
</tr>
<tr>
<td>7</td>
<td>ResDiff</td>
<td>248</td>
<td>38.3</td>
</tr>
<tr>
<td>8</td>
<td>ResEq</td>
<td>98</td>
<td>15.4</td>
</tr>
<tr>
<td>9</td>
<td>EvDiff</td>
<td>26</td>
<td>1.2</td>
</tr>
<tr>
<td>10</td>
<td>EvEq</td>
<td>14</td>
<td>2.3</td>
</tr>
<tr>
<td>11</td>
<td>IsrDiff</td>
<td>182</td>
<td>10.3</td>
</tr>
<tr>
<td>12</td>
<td>IsrEq</td>
<td>98</td>
<td>9.6</td>
</tr>
</tbody>
</table>
Approach

Firstly, we focus on design verification and testing.

- Design verification: the design of RX-OSEK850 is verified by a model checking tool Spin.
- Testing: we automatically generate test cases from the verified design.

```
<table>
<thead>
<tr>
<th>Design Verification</th>
<th>Automatic Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RX-OSEK850</td>
<td>Simulator</td>
</tr>
</tbody>
</table>
```

Putting an assumption that the design model is correct

\[ \text{Design model} = \text{Test oracle} \]
Testing

- We adopted testing to check RX-OSEK850 implementation.
  - RX-OSEK850 is already implemented.
  - Automata Theoretic Conformance Testing.
- Design Model = Test Oracle
  - We put the assumption that the design model is correct.
    - We made much effort to check the design model by model checking.
- We extract test cases and expected results from the design model.
  - Exhaustive test cases are automatically generated by search of Spin model checker.
  - Expected results are computed in the design model.

state transition model (Promela)  test tree  test cases

\[
\begin{align*}
&\text{state transition model (Promela)} \\
&\begin{array}{c}
\text{Search by Spin}
\end{array} \\
&\text{Scan test tree}
\end{align*}
\]

\[
\begin{align*}
&\begin{array}{c}
\text{test tree}
\end{array} \\
&\text{TC} = \{ab, cd\}
\end{align*}
\]
# Execution of Testing

- Total number of test cases: 742,748

```latex
<table>
<thead>
<tr>
<th>Priorities</th>
<th>Priorities</th>
</tr>
</thead>
<tbody>
<tr>
<td>TaskA</td>
<td>1</td>
</tr>
<tr>
<td>TaskB</td>
<td>2</td>
</tr>
<tr>
<td>TaskC</td>
<td>3</td>
</tr>
<tr>
<td>ResouceA</td>
<td>1 1 2 3 2 1</td>
</tr>
<tr>
<td>ResouceB</td>
<td>2 3 3 2 1</td>
</tr>
<tr>
<td>#Test cases</td>
<td>12483 15077 26373 37127 25035 8495</td>
</tr>
<tr>
<td>pan exe. time</td>
<td>12.9 16.8 26.0 38.7 26.7 10.7</td>
</tr>
<tr>
<td>TCG exe. time</td>
<td>19.0 24.9 67.9 73.1 58.5 12.7</td>
</tr>
<tr>
<td>TPG exe. time</td>
<td>176.8 174.4 508.5 522.8 433.9 95.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Priorities</th>
<th>Priorities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task A</td>
<td>1</td>
</tr>
<tr>
<td>Task B</td>
<td>1</td>
</tr>
<tr>
<td>Task C</td>
<td>2</td>
</tr>
<tr>
<td>Resouce A</td>
<td>1 1 2 3 2 1</td>
</tr>
<tr>
<td>Resouce B</td>
<td>2 3 3 2 1</td>
</tr>
<tr>
<td>#Test cases</td>
<td>17151 22427 44723 60707 39457 10331</td>
</tr>
<tr>
<td>pan exe. time</td>
<td>19.0 22.7 45.0 60.5 40.2 11.5</td>
</tr>
<tr>
<td>TCG exe. time</td>
<td>35.6 44.7 117.6 179.7 99.4 16.8</td>
</tr>
<tr>
<td>TPG exe. time</td>
<td>290.7 320.2 799.8 1353.5 694.4 138.9</td>
</tr>
</tbody>
</table>
```
Testing Results

- The execution of testing is fully automated by TCG and TPG.
- It can be done in parallel, but, the number of simulator licenses is limited to 3.
  - The licenses are occupied by engineers of Renesas daytime.
  - We executed testing in parallel nighttime and weekends.
- It takes around 3 months to complete the testing.
  - We could not major precise time to complete the testing due to our ad-hoc parallelization of testing.
- Samples)
  - It takes 169.75 hours for 26,489 test cases (Total compile time: 42.5 hours, execution: 127.25 hours)
  - It takes 265 hours for 44,723 test cases (compile time: 80.25 hours, execution: 184.75 hours)
- The estimation of total time of testing based on those samples is around 4535 hours, that is, 189 days.
  - We succeeded in reducing the testing time by the parallelization.
Whole view of our approach

OSEK/VDX Specification

RX-OSEK850 Specification

Design Model (Promela)

Environment Model

EnvGen

Environment (Promela)

Spin
Whole view of our approach

OSEK/VDX Specification
RX-OSEK850 Specification

Design Verification

Environment Model → EnvGen → Environment (Promela) → Spin

Design Model (Promela)

Test Model (Promela) → Spin

TCG/TPG

Test Case Generation

Testing

RX(Binary) → Simulator

Test Programs
Results

- Surprisingly, all the test cases have been passed at a time!
  - All the execution sequences of the design model have been accepted by the implementation.
  - We did not need any modification of the design model during the testing.
- Although no new bug was found, we obtained the confidence in the correctness of RX-OSEK850.