Software Verification and Validation Laboratory

svv.lu
SnT Centre for Security, Reliability and Trust

- Part of the University of Luxembourg
- Focused on industry-driven research and innovation
- 24 MEUR turnover in 2017 (70% competitive)
- Headcount >260, >45 nationalities
- Most highly-cited scientists in Luxembourg
- CS @ UL ranked 58 in the world
- 31 industry and public-service partners
- 60 industrial PhD candidates (60% of total)
- 50 Research Associates (60% of total)
- FNR Public-Private Partnership programs
SVV Lab Overview

- Established in 2012
- Requirements Engineering, Security Analysis, Design Verification, Automated Testing, Runtime Monitoring
- ~25 lab members
- Eight partnerships
- ERC Advanced Grant
- Budget 2016: ~2 Meuros
Mode of Collaboration

- Strong emphasis on applied research, driven by needs
- Tight, large-scale industrial collaborations
Meeting Objectives

- Discuss partnership on testing and verifying Simulink models (i.e., controllers, plant)
- Significant funding with ERC grant and co-funding with FNR (Luxembourg funding agency)
- Refine the objectives in current proposal
- Demo our existing tools
Meeting Agenda

• March 2\textsuperscript{nd}
  • 11:30am – 12:30pm  Discussing ASTech models
  • 2pm – 3pm  Meeting with Bjorn Ottersten, SnT Director
  • 3 pm – 5pm Technical presentations (Audi, SnT)

• March 3\textsuperscript{rd}
  • 9am – 11am Technical presentation (Cnted) + demos
  • 11 am – 12pm Discussions (collaboration, partnership)
  • 2pm – 4pm Discussions (collaboration, partnership)
ASTech Models
Outline

• Introduction
  • Testing cyber-physical systems (ERC advanced grant)
• Automated testing of embedded software systems
  • Testing closed-loop controllers
  • Testing Simulink models
  • Fault Localization of Simulink models
• Other projects in the automotive domain
• Proposals for ASTech-SnT collaboration
Testing Closed Loop Controllers
\[ \begin{align*}
\text{Plant Model} & \quad \text{output}(t) = \text{desired}(t) \\
\Sigma & \quad \text{actual}(t) = \text{desired}(t) - \text{output}(t) \\
P & \quad K_P e(t) \\
I & \quad K_I \int e(t) \, dt \\
D & \quad K_D \frac{de(t)}{dt}
\end{align*} \]

**Test Input**

<table>
<thead>
<tr>
<th>Time</th>
<th>Initial Desired Value</th>
<th>Final Desired Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>T/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Test Output**

<table>
<thead>
<tr>
<th>Time</th>
<th>Desired Value</th>
<th>Actual Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>T/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
We identify high risk behaviors by maximizing test objectives.
Requirements and Test Objectives

Requirement: As soon as braking is requested, the contact between caliper and disk should occur within 32ms

Min\{\max\{\max\{|x(t) - (x_0 + \epsilon)|, |x(t) - (x_0 - \epsilon)|\}\}_{t_0 \leq t \leq T}\}
Continuous Controller Tester

1. Exploration
   - HeatMap Diagram
   - Domain Expert

2. Single-State Search
   - List of Critical Regions

Worst-Case Scenarios

Objective Functions based on Requirements + Controller-plant model

Initial Desired Value

Final Desired Value
Objective Functions based on Requirements + Controller-plant model

1. Exploration

HeatMap Diagram

List of Critical Regions

Domain Expert

2. Single-State Search

Worst-Case Scenarios

Smoothness

Responsiveness

20% Overshoot

250 ms Response Time

20% Overshoot

250 ms Response Time
Finding Seeded Faults

Inject Bug

Figure 1
Summary

• We found several interesting test scenarios (worst cases) during Model-in-the-Loop (MiL) testing compared to what our partner had found so far.

• These scenarios are also run at the Hardware-in-the-Loop (HiL) level, where testing is much more expensive:

  MiL results -> test selection for HiL
Testing Simulink Models
(Closed Loop and Open Loop Controllers)
Objectives

- Testing Simulink/Stateflow models in their entirety
- Without requiring plant models
- Without requiring automated test oracles
Simulink Testing Challenge I

Incompatibility

Existing testing techniques are not applicable to simulation models (with time-continuous behaviors)
Incompatibility Challenge -- Example

Simulation Model

Not Applicable

Code Generation Model

Applicable
Simulink Testing Challenge II

Low Fault-Revealing Ability

Existing testing techniques make unrealistic assumptions about test oracles
Low Fault-Revealing Ability Example

Covers the fault **and**
**is Likely** to reveal it

Covers the fault **but**
**is very unlikely** to reveal it

Faulty Model Output
Correct Model Output
Our Approach

Search-based Test Generation Driven by Output-Diversity and Anti-Patterns
Search-Based Test Generation

Search Procedure

S ← Initial Candidate Solution

Repeat

R ← Tweak (S)

if Fitness (R) > Fitness (S)

S ← R

Until maximum resources spent

Return S
Output-Based Heuristics

Failure Patterns

Output Diversity
Failure-based Test Generation

- Maximizing the likelihood of presence of specific failure patterns in output signals

**Instability**

**Discontinuity**
Output Diversity -- Vector-Based
Output Diversity -- Feature-Based

signal features

value

instant-value (v)
constant-value (n, v)

constant (n)
increasing (n)
decreasing (n)

derivative

sign-derivative (s, n)

extreme-derivatives

first-derivative with strict local optimum
1-sided discontinuity

second derivative

discontinuity

1-sided continuity with strict local optimum

increasing

discontinuity with strict local optimum
Evaluation

How does the **fault revealing ability** of our algorithm compare with that of Simulink Design Verifier?
Simulink Design Verifier (SLDV)

- Underlying Technique: Model Checking and SAT solvers
- Test objective: Testing is guided by structural coverage
Our Approach vs. SLDV

- Our approach outperformed SLDV in revealing faults

<table>
<thead>
<tr>
<th>Faults</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLDV</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Our Approach</td>
<td>20</td>
<td>16</td>
<td>20</td>
<td>20</td>
<td>14</td>
<td>17</td>
<td>20</td>
<td>15</td>
<td>15</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Faults</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLDV</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Our Approach</td>
<td>14</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>

× SLDV could not find the fault
✔ SLDV found the fault

# The number of fault-revealing runs of our algorithm (out of 20)
SimCoTest Tool

https://sites.google.com/site/simcotentool/  
https://sites.google.com/site/cocotentool/
Summary

- We evaluated SimCoTest on seven representative Delphi Simulink models and one model from Bosch research lab.

- Hands-on tutorial to ten Delphi engineers:
  - “SimCoTest is useful for early stages of controller design to identify and detect design flaws.”

- We found some issues in Delphi models.

- We plan to follow further development of SimCoTest and its commercialization.
Tool Demos
Case Study

- Electro-Mechanical Braking (EMB)

- A public-domain model developed by the Bosch Research lab (http://cps-vo.org/node/20289)

- EMB Simulink model
  - Consists of a physical plant model, a PID controller and a state-flow
  - Is simulated by a variable-step solver
  - Contains float variables and float computation
Demo of SimCoTest

- SimCoTest is able to identify the following error patterns in EMB output
  - Oscillation (marginal stability)
  - Discontinuity
  - Growth to infinity (instability and not marginally stable)
Fault Localization in Simulink Models
Fault Localization

A Simulink model

A test suite

A Simulink model
Statistical Debugging

For $O_2$:

<table>
<thead>
<tr>
<th>b1</th>
<th>b2</th>
<th>b3</th>
<th>b4</th>
<th>b5</th>
<th>b6</th>
<th>b7</th>
<th>b8</th>
<th>b9</th>
<th>b10</th>
<th>b11</th>
<th>b12</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Statistical Debugging

For $O_2$:

<table>
<thead>
<tr>
<th>b1</th>
<th>b2</th>
<th>b3</th>
<th>b4</th>
<th>b5</th>
<th>b6</th>
<th>b7</th>
<th>b8</th>
<th>b9</th>
<th>b10</th>
<th>b11</th>
<th>b12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Test Suite

Status

<table>
<thead>
<tr>
<th>t1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not executed</td>
</tr>
</tbody>
</table>

Output$_1$

Output$_2$
Statistical Debugging

Test Suite

<table>
<thead>
<tr>
<th>b₁</th>
<th>b₂</th>
<th>b₃</th>
<th>b₄</th>
<th>b₅</th>
<th>b₆</th>
<th>b₇</th>
<th>b₈</th>
<th>b₉</th>
<th>b₁₀</th>
<th>b₁₁</th>
<th>b₁₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

For O₂:

Status

- Not executed
- Executed, pass
- Executed, fail

<table>
<thead>
<tr>
<th>Output₁</th>
<th>t₁</th>
<th>t₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>b₁</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b₅</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b₈</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>b₁₀</td>
</tr>
<tr>
<td>b₁₁</td>
</tr>
<tr>
<td>b₁₂</td>
</tr>
</tbody>
</table>
Statistical Debugging

Test Suite

For O₂:

<table>
<thead>
<tr>
<th>b1</th>
<th>b2</th>
<th>b3</th>
<th>b4</th>
<th>b5</th>
<th>b6</th>
<th>b7</th>
<th>b8</th>
<th>b9</th>
<th>b10</th>
<th>b11</th>
<th>b12</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Status

- t1: Executed, pass
- t2: Executed, fail
- t3: Not executed

Output₁: b6, b7, b8
Output₂: b10, b11, b12
Statistical Debugging

Test Suite

Status

For $O_2$:

<table>
<thead>
<tr>
<th>b1</th>
<th>b2</th>
<th>b3</th>
<th>b4</th>
<th>b5</th>
<th>b6</th>
<th>b7</th>
<th>b8</th>
<th>b9</th>
<th>b10</th>
<th>b11</th>
<th>b12</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Not executed
Executed, pass
Executed, fail

Output₁
Output₂

t₁: ✓
t₂: ✓
t₃: X
• Simulink blocks are ranked according to likelihood of causing output failures

• Engineers inspect Simulink blocks using the generated ranking to identify faulty block(s)

<table>
<thead>
<tr>
<th>Block</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>b3</td>
<td>1.0</td>
</tr>
<tr>
<td>b7</td>
<td>0.6667</td>
</tr>
<tr>
<td>b11</td>
<td>0.6667</td>
</tr>
<tr>
<td>b1</td>
<td>0.5</td>
</tr>
<tr>
<td>b5</td>
<td>0.5</td>
</tr>
<tr>
<td>b6</td>
<td>0.5</td>
</tr>
<tr>
<td>b8</td>
<td>0.5</td>
</tr>
<tr>
<td>b12</td>
<td>0.5</td>
</tr>
<tr>
<td>b2</td>
<td>0.0</td>
</tr>
<tr>
<td>b4</td>
<td>0.0</td>
</tr>
<tr>
<td>b9</td>
<td>0.0</td>
</tr>
<tr>
<td>b10</td>
<td>0.0</td>
</tr>
</tbody>
</table>
Results

- We have developed, SimFL, a tool for automated fault localization of Simulink models.
- Our tool is able to help localize multiple faults in Simulink models.
- SimFL has been applied to three large Simulink models with 400 to 800 blocks containing two to five faults.
- Using SimFL engineers need to inspect less than 3% of the model blocks on average to localize faults.
Statistical Debugging: Drawbacks

- Faulty blocks may not be ranked high
- Many blocks may have the same score
- Engineers may have to inspect many blocks until they find the faulty block(s)
Performance Improvement

- Engineers may fail to find fault(s) after inspecting the top rank blocks

- Our improvements:
  - Extending the test suites with test cases targeted at revealing test cases
  - Identifying heuristics to help engineers know situations where statistical ranking cannot be further refined/improved
Tool Demo
ASTech-SnT collaboration
Topics

• Stability analysis of nonlinear systems
• Analysis of performance and simulation (execution) time of Simulink models
Stability Analysis

- Identifying the parts of the input space of a Simulink model that may lead to violation of system stability requirements
- Identifying root causes of erroneous/unstable behaviors
- Investigating whether the problematic input regions can be reduced by automatic parameter tuning
Analysis of Performance and Simulation Time

- Identifying inputs, configurations, or blocks that are responsible for simulation slowdown
- Severe performance degradations after changes in models (regression performance analysis)
- Proposing ways to improve the simulation performance
Automated Testing and Verification of Cyber-Physical Systems

Software Verification and Validation (SVV) Lab
Interdisciplinary Centre for Security, Reliability and Trust (SnT)
University of Luxembourg
Testing via Physics-based Simulation
Testing via Simulation (Limitations)

- Simulation scenarios are generated manually
- There are many simulation scenarios
- No guidance as to which scenarios should be selected to test the system
Our Approach

- Automated testing of complex ADAS via physics-based executable models of these systems and their environments

<table>
<thead>
<tr>
<th>Challenges</th>
<th>Our solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test input is large</td>
<td><strong>Metaheuristic search</strong> to focus testing on worst case/critical behaviors</td>
</tr>
<tr>
<td>Simulation takes time</td>
<td><strong>Surrogate models</strong> to predict the simulation outcome <em>without running simulations</em></td>
</tr>
</tbody>
</table>
PeVi Requirement

The PeVi system shall detect any pedestrian located in the Acute Warning Area (AWA) of a vehicle

- Test objectives **critical aspects:**
  - Time-To-Collision (TTC) is small
  - The pedestrian is located near the car
  - The pedestrian is at the boundary of the AWA
Our Search-based Test Generation

- We use **multi-objective** search algorithm to generate test cases

- Three objectives: Minimum Distance to Car $\min\{D(P/\text{Car})\}$, Minimum Time To Collision $\min\{\text{TTC}\}$, and Minimum Distance to AWA $\min\{D(P/\text{AWA})\}$

- Input a vector (car-speed, person-speed, person-position $(x,y)$, person-orientation)

- Each search iteration calls simulation to compute objectives
Evaluation

• Given the same execution time, our approach is able to produce higher quality solutions than baseline methods (Random Search)

• We provided engineers with 20 scenarios representing risky situations (no detection in AWA, collision/no detection) by varying weather conditions, roadside objects and ramped/curved road

• Scenarios helped engineers identify several critical behaviors of PeVi that have not been previously identified by manual simulation

• The scenarios are available at: https://sites.google.com/site/testingpevi
An example critical scenario
Conclusion

• An automated effective testing approach for ADAS

• Formulated the generation of critical test cases as a multi-objective search problem using the NSGA2 algorithm

• Improved the search performance, while maintaining the accuracy, using surrogate models based on neural networks

• Generated some critical scenarios: no detection in the AWA, or collision and no detection
Model-based development of control systems

Model-in-the-Loop Stage
- Simulink Modeling
  - MiL Testing

Software-in-the-Loop Stage
- Code Generation and Integration
  - SiL Testing

Hardware-in-the-Loop Stage
- Software Running on ECU
  - HiL Testing
Simulink Testing Challenges

- Complex models with several hundreds of blocks
- Mix of continuous and discrete behaviour
- Captures both hardware/physical components and software/algorithms
- Mix of fixed-point and floating point computations
Controller Properties: Smoothness

Test Objective is to maximize $O_{sm}$
Controller Properties:
Responsiveness

Test Objective is to maximize $O_r$
We define a set of basic features characterizing distinguishable signal shapes.
Feature Functions

- For each feature $f$ in our feature classification, we define a feature function $F_f$

- For a signal $s_g$, the value of $F_f(s_g)$ quantifies the similarity between shapes of the signal $s_g$ and the feature $f$

Always increasing

\[
F_f(s_g) = \sum_{i=1}^{K} ((s_g(i \cdot \Delta t) - s_g((i - 1) \cdot \Delta t))) - |s_g(i \cdot \Delta t) - s_g((i - 1) \cdot \Delta t)|)
\]

The higher the value of $F_f(s_g)$, the more similar $s_g$ is to $f$
Output Diversity
Test Suite Generation Algorithm

Test Suite Generation Algorithm

P \leftarrow \text{Initial number of segments}

Best \leftarrow \text{TS} \leftarrow \text{Initial test suite}

\text{Repeat}\n
\quad \text{TS} \leftarrow \text{Tweak (TS, P)}

\quad \text{Execute (TS)}

\quad \text{If } O (\text{TS}) > O (\text{Best})

\quad \quad \text{Best} \leftarrow \text{TS}

\quad \text{If (Coverage has reached a plateau less than 100\%)}

\quad \quad P \leftarrow P + 1

\text{Until maximum resources spent}

\text{Return TS}

O \text{ is either } O_v \text{ or } O_f

A whole test suite generation algorithm
Failure patterns for Continuous Outputs

Instability

Discontinuity

Time

ClutchCtrlSig

Output

A

B

C

Instability

Discontinuity

Time

0.0

1.0

2.0

-1.0

-0.5

0.0

0.5

1.0

0.0

0.25

0.50

0.75

1.0

0.0

0.25

0.50

0.75

1.0
SimCoTest Maturity

- Hands-on tutorial to ten Delphi engineers:
  - “SimCoTest is useful for early stages of controller design to identify and detect design flaws.”

- We are currently applying SimCoTest to a number of newly developed (cutting edge) Delphi Controller models to help engineers with testing/verification, and to better demonstrate practical usefulness of SimCoTest
Failure-based Test Generation

- Maximizing the likelihood of presence of specific failure patterns in output signals

![Graphs showing Instability and Discontinuity](image)
Comparing SimCoTest with Simulink Design Verifier

<table>
<thead>
<tr>
<th>Fault No.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>#OurApproach</td>
<td>20</td>
<td>16</td>
<td>20</td>
<td>11</td>
<td>5</td>
<td>20</td>
<td>14</td>
<td>17</td>
<td>11</td>
<td>20</td>
<td>4</td>
</tr>
<tr>
<td>*SLDV</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fault No.</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>#OurApproach</td>
<td>5</td>
<td>14</td>
<td>2</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>*SLDV</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

# Number of Fault Revealing Runs of our algorithm (Out of 20)
* Did SLDV reveal the fault? (Yes/No)

The only fault found by SLDV (fault 17), was also found by SimCoTest with very high probability