Evaluating Mutation Operators for Test Case Generation from UML State Diagrams

Wolfgang Herzner, Rupert Schlick (Austrian Institute of Technology)

FMCO 2010, Graz, November 30, 2010
MOGENTES - Intro

- Objective: to significantly enhance testing and verification of dependable embedded systems
  - by means of automated generation of **efficient test cases**
  - *Goal: reduce testing effort by at least 20%*

- Approaches
  - model-based (UML state charts, Simulink)
  - (model) mutation, using fault models for
  - exploiting OOAS, LTS, CBMC, UPPAAL
  - FI (fault injection)

- Motivation
  - stronger test cases than if using only functional or code coverage metrics
Motivation - Example

context TriangType(a: int, b: int, c: int): String
pre: a < (b+c) and b < (a+c) and c < (a+b)
post: if ((a = b) and (b = c))
    then result = "equilateral"
else if ((a = b) or (a = c) or (b = c))
    then result = "isosceles"
else result = "scalene"
endif endif

- Test cases
  - $a = 1, b = 1, c = 1$, result = "equilateral"
  - $a = 2, b = 2, c = 1$, result = "isosceles"
  - $a = 2, b = 3, c = 4$, result = "scalene"

- would not detect the faults
  - post: if ((a = a) and (b = c))
    then result = "equilateral"
  - else if ((a = b) or (a = c) or (a = c))
    then result = "isosceles"
  - else result = "scalene"
  - endif endif

- Test case $a = 1, b = 2, c = 2$, result = "isosceles" detects both
Overview

MOGENTES – research core

MOGENTES – demonstrators
- Automotive
- Off-highway
- Railway signalling

Test Case Application (unit, system, SIL, HIL, FI)

Result
Data
Scenario

Test Cases

Test Case Generation

(Fault) Model

Safety Requirements (Standards)

User Requirements

Specification

Software / System

Safety Requirements (Standards)

User Requirements

Overview

MOGENTES – demonstrators
- Automotive
- Off-highway
- Railway signalling

Test Case Application (unit, system, SIL, HIL, FI)

Result
Data
Scenario

Test Cases

Test Case Generation

(Fault) Model

Safety Requirements (Standards)

User Requirements

Overview

MOGENTES – research core

MOGENTES – demonstrators
- Automotive
- Off-highway
- Railway signalling

Test Case Application (unit, system, SIL, HIL, FI)

Result
Data
Scenario

Test Cases

Test Case Generation

(Fault) Model

Safety Requirements (Standards)

User Requirements

Overview

MOGENTES – research core

MOGENTES – demonstrators
- Automotive
- Off-highway
- Railway signalling

Test Case Application (unit, system, SIL, HIL, FI)

Result
Data
Scenario

Test Cases

Test Case Generation

(Fault) Model

Safety Requirements (Standards)

User Requirements

Overview

MOGENTES – research core

MOGENTES – demonstrators
- Automotive
- Off-highway
- Railway signalling

Test Case Application (unit, system, SIL, HIL, FI)

Result
Data
Scenario

Test Cases

Test Case Generation

(Fault) Model

Safety Requirements (Standards)

User Requirements

Overview

MOGENTES – research core

MOGENTES – demonstrators
- Automotive
- Off-highway
- Railway signalling

Test Case Application (unit, system, SIL, HIL, FI)

Result
Data
Scenario

Test Cases

Test Case Generation

(Fault) Model

Safety Requirements (Standards)

User Requirements
Use Cases (Demonstrators)

- Automotive
  - Car-Alarm System
  - Steering Anti Catchup

- Off-highway vehicles
  - Implement-control (bucket) through human driver (ISOBUS-conform)

- Railway interlocking
  - Electronic signalling and interlocking control system ELEKTRA
  - Peripheral signals analyser (of electronic rail control desk)
Figures

- Duration:

- Costs:
  - 4.4 M€ Total
  - 3.1 M€ Funding

- Efforts:
  - 400.5 PM RTD
  - 48.5 Demo
  - 12.0 Mgmt

- Coordination:
  - AIT

More at
www.mogentes.eu

Partner

- Universities
  - Budapest University of Technology and Economics (HU)
  - ETH Zurich (CH) / Stanford University (UK)
  - Graz University of Technology (AT)

- Research Organisations
  - AIT – Austrian Institute of Technology
  - SP Technical Research Institute of Sweden

- Industrial Demonstrators
  - Ford Forschungszentrum Aachen (DE)
  - Prolan Irányítástechnikai ZRT (HU)
  - Re:Lab S.R.L. (IT)
  - Thales Rail Signalling Solutions GmbH (AT)

- Tool Developer
  - Prover Technology AB (SE)
Black box test case generation from UML State Machines

Motivation:
- Requirements coverage
- Test oracle needed
- No access to source
Mutation testing

- Mutants provide test efficiency measure
  - Evaluate test sets
  - Generate test cases
  - Can emulate all common coverage metrics

- Assumptions
  - „Competent programmer“ hypothesis
  - Fault coupling effect

- Stephan Weissleder, 2010, ” Satisfying yet unsupported Coverage Criteria or Simulated Satisfaction for MBT“
Mutation Operators

- Init values, constants, OCL literals, AGSL literals:
  - Constant Value Variation Step
  - Enum Value Exchange
  - Set Constant Value to Zero/Max

- States:
  - Remove State Entry/Exit Action
  - Exchange State Entry/Exit Action Method

- Transitions:
  - Remove Transition
  - Exchange Transition Source/Target
  - Exchange Transition Trigger

- Triggers
  - Exchange Trigger Signal
  - Minimally Change Time Trigger Duration
  - Substantially Change Time Trigger Duration
  - Minimize/Maximize Time Trigger Duration
  - Modify Change Trigger Expression
  - Exchange Call Trigger Method

- Guards:
  - Fix Guard Value
  - Invert Guard
  - Modify Guard Expression

- Effect:
  - Remove Effect
  - Exchange Effect Method
  - Modify Effect Body

- OCL:
  - Fix Expression
  - Negate Expression
  - Fix Subexpression
  - Negate Subexpression
  - Modify Boolean Operator
  - Modify Relational Operator
  - Modify Arithmetic Operator
  - Modify Set Operator
  - Modify Quantifier

- AGSL:
  - Remove Statement
  - Reorder Statement
  - Fix Parameter/Property
  - Modify Parameter/Property
  - Modify Operator
  - Fix Operand
  - Modify Operand
  - Fix Result
„All mutants are equal“

- Lots of possible mutants
- Checking test cases against mutants takes effort
- Generating is more expensive than checking
- Equivalent Mutants

Can we:

- leave out mutation operators or mutants?
- sort mutants/operators to reduce generation effort?
- sort mutants/operators to reduce checking effort?
Example: Car Alarm System (Ford) (1)

Car Alarm System

- car locked
- alarm sensor
- alarm armed
- acoustic alarm
- optical alarm

- example of a requirement

The anti-theft alarm system can be deactivated at any time – even when the alarm is sounding – by unlocking the vehicle from outside.
Example: Car Alarm System (Ford) (2)

- Slightly simplified automotive application (no sensor fault handling; based on behaviour from user manual)
- „Body control“ applications: central lock, window control, seat heating, seat position memory, cruise control, ..
Example: Car Alarm System (Ford) (3)

- 11 states
- 17 transitions

- Used Elements
  - Time trigger
  - Change trigger
  - OCL guards (minimal)
  - Activities (atomic or two lines, AGSL)
  - Nested states
  - Orthogonal regions
Mutants and test cases

- 15 applicable mutation operators,
- 20 locations
- 111 mutants
- 152 generated test cases (minimal search depth)

<table>
<thead>
<tr>
<th>Source</th>
<th>Manual Set</th>
<th>„Minimal“ generated set</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 use cases</td>
<td>8 mutants</td>
<td></td>
</tr>
<tr>
<td>I/O Steps</td>
<td>70</td>
<td>137</td>
</tr>
<tr>
<td>Accumulated time</td>
<td>470s</td>
<td>800s</td>
</tr>
<tr>
<td>Found mutants</td>
<td>83 / 75%</td>
<td>111 / 100%</td>
</tr>
</tbody>
</table>
Observations:

- Strongest generated test case covers 75% of mutants, weakest only 1 mutant
- Weakest test case is needed! (short, quiescence)
- Longest 4 test cases from minimal set from the 8 mutants cover 96%
- Test cases for "guard false" mutation operator cover over 90%
- 1 mutant is found by manual tests but not by guard false mut-op
- All mutants from minimal set are located at transitions
- Changing code chunks covers removing the containing code snippet
Caveats – Black box testing (1)

Requirements

/ show values; output values

error fixed / show values; output values

Working

Stopped

show safe state; set safe outputs /entry

error

100ms / recalculate; show values; output values

Refactored

Working

show values; output values /entry

Stopped

show safe state; set safe outputs /entry

error fixed

100ms / recalculate
Caveats – Black box testing (2)
Modeling for black box mutation testing

Rules for requirements e.g.

Negative statements and compound statements are prohibited.

What would be rules for building requirements models usable for mutation based test case generation?
And now?

- Next steps:
  - Add mutations
  - Verify and extend observations with 1 or 2 larger applications (other MOGENTES demonstrators)
  - Use results for prioritizing/discarding mutants for test case generation

- Further research ideas:
  - Rules for building test models from requirements
  - What do the relations of test cases and mutants tell us about an application?