TESTING NON-FUNCTIONAL QUALITY CHARACTERISTICS OF CYBER-PHYSICAL SYSTEMS

Presented by Martin A. Schneider (Fraunhofer FOKUS)
Cyber-Physical Systems and Non-Functional Quality Characteristics

• CPS interact with their physical environment
  • through sensors, actuators and a communication infrastructure
• examples of CPS
  • sensor systems
  • connected cars and automated driving
  • factory of the future
  • medical devices
• reliability, safety and security are critical quality characteristics of CPS
CPS may be unreliable

- undesired behaviour of a CPS is observed at runtime
  - due to uncertainty in the digital x physical environment

- Challenge
  - How to find such scenarios efficiently in the infinite and complex space of the scenarios?

- Solution
  - Search algorithms
Search Algorithms

- search algorithms do not exactly how to create a solution to a problem
- derive candidate solutions
- assess them
- plenty of algorithms
  - genetic algorithm: mutation, crossover
- need to be tailored to the concrete problem
- search-based testing
Search-based Testing with a Genetic Algorithm

- Starting point
- Mutation
- Quality evaluation (fitness evaluation) & selection
- Candidate solutions
- Test case generation & execution
- Crossover
- Test case generation & execution

40
70
85
Functional Models as a Starting Point

- We use UML state machines that describe the CPS' expected behavior.
- States, transitions, guards, triggers, effects.
<table>
<thead>
<tr>
<th>Mutation Operator</th>
<th>Description</th>
<th>Constraints/Comments</th>
<th>Mutation Operator</th>
<th>Description</th>
<th>Constraints/Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Transition</td>
<td>Adds a new transition by duplicating an existing one and setting a new source and target state.</td>
<td></td>
<td>Remove Trigger</td>
<td>Transforms the transition to a completion transition.</td>
<td></td>
</tr>
<tr>
<td>Remove Transition</td>
<td>Completely removes the transition.</td>
<td>Transitions having an initial state as source or a final node as target must not be removed. Equivalent to ‘Change Guard: replace expression with false’.</td>
<td>Remove Guard</td>
<td>Removes the guard of a transition completely.</td>
<td>Equivalent to ‘Change Guard: replace expression with true’</td>
</tr>
<tr>
<td>Remove Transition (with State Merge)</td>
<td>Completely removes the transition. Merges the source and target state if the removed transition is the only one connecting them (optional: with the same direction). This avoid mutilated state machines which inhibit generating test cases.</td>
<td>Transitions having an initial state as source or a final node as target must not be removed. Equivalent to ‘Change Guard: replace expression with false’.</td>
<td>Change Trigger Operation</td>
<td>Changes the operation to another one of the same interface of the original operation.</td>
<td></td>
</tr>
<tr>
<td>Reverse Transition</td>
<td>Swaps source and target of the transition.</td>
<td>Transitions having an initial state as source or a final node as target must not be reversed. Optional: Transitions being the only one that connect source and target state must not be removed (optional: with the same direction). This avoid mutilated state machines which inhibit generating test cases.</td>
<td>Change Guard/Change Effect</td>
<td>- replace expression with true/false - negate expression - replace subexpression with true/false - negate subexpression - change logical operator - change relational operator - change arithmetic operator - change set operator - change quantifier - replace operand guard/effect mutation operators - remove statement - move statement - fix parameter/property of a called method or sent signal - change called method or sent signal - change operator - fix operand (replace it with a literal) - change operand (replace with variable, call parameter or signal property of the same type) - replace result: replace right-hand-side (RHS) expression with default value of left-hand-side (LHS)</td>
<td>Guards and effects are written in C#.</td>
</tr>
<tr>
<td>Change Source/Target</td>
<td>Move the source or the target of the transition to any other state.</td>
<td>In case the target state of the transition is changed, the target must not be the initial state. In case the source state of the transition is changed, the source must not be the final node.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Uncertainty describes what might go wrong in the environment. Testers bootstrap search-based testing with such knowledge. Uncertainty models are pointing to parts of the functional model. Mutation is guided by these uncertainty models. With further generations, chances increase that other kinds of mutations are introduced.
Search-based Testing with a Genetic Algorithm

- Starting point
- Candidate solutions
- State machines
- Mutation
- Quality evaluation (fitness evaluation & selection)
- Recombine mutations
- Test case generation & execution

Guided by uncertainty models
Example

Mutation Operator
Delete Transition

Initial1
\[ \text{initializing} \]
\[ \text{instantiate number of required} \]
\[ \text{calibrating} \]

waiting_for_signal
\[ \text{calibrate all required mounted}
\text{set calibrated orientation to actual one} \]

setPosition/
\text{store the position data in}

getUserPositionData
\[ \text{getAllPositions/}
\text{return all currently recorded} \]

mount

configureTag()

mount(locator0, 0)

mount(locator1, 90)

mount(locator2, 180)

mount(locator3, 270)

calibrating()

mount(locator1, 180)

setPosition(1)

getAllPositions()
Quality Evaluation: Fitness Factors

- Model-based description of quality indicating functions, e.g., provided by the CPS (mission-critical functionality)
- By the test environment, e.g., response time, resource consumption
- Fitness factors allow to specify if values should minimize, maximize, or approach a certain threshold value
- How they should relate to other values

Constraint 1:
![OCL] not self.kind.oclIsUndefined() implies self.kind = self.counterPart.kind

Constraint ActualMetaClass:
![OCL] self.base.NamedElement.oclIsKindOf(BehavioralFeature)
Conclusions

• small effort for testers
  • start from functional models (state machines)
  • add uncertainty models
• reduction of search space
  • search is guided by uncertainty models
• configurable
  • by uncertainty models
  • and model-based fitness factors
Conclusions

Thanks for your attention!
Questions!?

Martin Schneider
Senior Scientist
Fraunhofer FOKUS
martin.schneider@fokus.fraunhofer.de