Specification Mutation

Vadim Okun
University of Maryland Baltimore County
Outline

1. What is Mutation?

2. How mutation is used?


4. Problems and Future Work.
What is Mutation?

• Define a set of mutation operators.

• Each operator is a pattern for a small syntactic change.

• Mutation operators model potential faults.

• Produce a mutant by applying an operator to the original program (specification).

• Applying operators repeatedly yields a set of mutants.
Sample Mutants for

\[ \text{AG (request} \rightarrow \text{AF state} = \text{busy)} \]

- Replace constant:
  \[ \text{AG (request} \rightarrow \text{AF state} = \text{ready)} \]

- Negate variable:
  \[ \text{AG (!request} \rightarrow \text{AF state} = \text{busy)} \]

- Replace logical operator:
  \[ \text{AG (request} \& \text{AF state} = \text{busy)} \]
Program Mutation (Original Use)

• Proposed by DeMillo, Lipton and Sayward (1978), Hamlet (1977).

• Procedure to measure test case adequacy.

• Each mutant is executed on each test case.

• A mutant is either dead or alive.
Early Uses of Specification Mutation

• Ajei Gopal and Tim Budd (1983) extended the concept of program mutation to the domain of program specifications.

• Specifications in predicate calculus form.

• Program must be executed on test input to obtain the output.

• M. Woodward (1993) investigated mutation operators for algebraic specifications.

• Algebraic specifications are considered as term-rewriting systems.

• Test adequacy is measured without executing the program.
Specification Mutation and Model Checking

- If inconsistency is found, attempt to generate a counterexample.

- Counterexamples have both stimulus and expected values.

- So can be automatically converted to complete test cases.

- What is mutated?
  - Constraints (passing tests) or
  - State machine (failing tests).
Categories of Mutants

Code Mutants

- Equivalent
- Nonequivalent

Logic Mutants

- Consistent
- Inconsistent
  - Falsifiable
  - Nonfalsifiable
Theoretical Foundations of Mutation.

- Competent programmer hypothesis:
  A programmer writes programs that are close to being correct.

- Competent specifier hypothesis:
  An analyst writes specifications which are close to what is desired.

- Coupling effect hypothesis:
  Simple and complex errors are coupled.
Mutation Operators for Specifications

- Operand Replacement Operator (ORO)
- Simple Expression Negation (SNO)
- Expression Negation Operator (ENO)
- Logical Operator Replacement (LRO)
- Relational Operator Replacement (RRO)
- Missing Condition Operator (MCO)
- Stuck-At Operators (STO)
- Associative Shift Operator (ASO)
An SMV Example

MODULE main
VAR
    request : boolean;
    state : {ready, busy};
ASSIGN
    init(state) := ready;
    next(state) := case
        state = ready & request : busy;
        1 : {ready, busy};
    esac;
SPEC AG (request → AF state = busy)
Mutants for

AG (request → AF state = busy)

<table>
<thead>
<tr>
<th>Oper</th>
<th>Example Mutants</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORO</td>
<td>AG (request → AF state = ready)</td>
</tr>
<tr>
<td>SNO</td>
<td>AG (!request → AF state = busy)</td>
</tr>
<tr>
<td></td>
<td>AG (request → AF (!state = busy))</td>
</tr>
<tr>
<td>ENO</td>
<td>AG (!(request → AF state = busy))</td>
</tr>
<tr>
<td>LRO</td>
<td>AG (request &amp; AF state = busy)</td>
</tr>
<tr>
<td></td>
<td>AG (request</td>
</tr>
<tr>
<td>MCO</td>
<td>AG AF state = busy</td>
</tr>
<tr>
<td>STA</td>
<td>AG (0 → AF state = busy)</td>
</tr>
<tr>
<td></td>
<td>AG (1 → AF state = busy)</td>
</tr>
<tr>
<td></td>
<td>AG (request → AF 0)</td>
</tr>
<tr>
<td></td>
<td>AG (request → AF 1)</td>
</tr>
</tbody>
</table>
Comparison of Mutation Operators

• Use D. R. Kuhn’s technique to analyze faults.

• SNO and ENO are not needed when ORO is used.

• Empirically evaluate mutation operators.

• ORO+ (ORO and RRO) has 100% coverage.
Problems and Future Work

• Define new mutation operators.
  Achieve high coverage with less mutants.

• Investigate program-based coverage.
  Evaluate tests using program mutants.

• Study state machine mutation.

• Use a different model checker.