Semantic Mutation Testing

John A. Clark, Haitao Dan, Robert M Hierons

The 8th CREST Open Workshop, 27-10-2010
An example: cruise control
Question

• What happens in no_vehicle_in_front if brake and level=increase?
Another question

- What happens in no_vehicle_in_front if a vehicle is detected and level=increase?
The problem

• Traditional mutation operators introduce changes similar to ‘slips’.

• Sometimes a developer/user will make semantic mistakes:
  – They will misunderstand the semantics of part of the language they are using
Semantic Mutation

• A developer has been using language X with semantics L and moves to X with semantics L’.  

• How do we find test data to find resultant faults?
An alternative: switching between programming languages

• Developer moves between two languages at the same level of abstraction that have different semantics for a common construct.

• Example:
  – Logical connectives in C and Ada.
    • C uses short-circuit evaluation;
    • Ada has alternatives (with and without short-circuit evaluation)
Scenario: refinement/retrenchment

• Similar constructs can have different semantics.

• Examples:
  – integer division in Z and Ada
  – retrenching infinite types (issues with precision, bounds on the types)
A simple framework

- We have a syntactic entity $N$ in a language with semantics $L$.
- Traditional mutation operators transform $(N,L)$ to some $(N',L)$
- Semantic mutation operators transform $(N,L)$ to some $(N,L')$ [or maybe even $(N',L')$]
- They aim to find a different type of mistake.
Current status and future work

• Prototype tool being developed for C
• Some experiments being conducted to explore nature of semantic mutants:
  – How many are produced?
  – How do they relate to traditional syntactic mutants?
  – What are good operators?
  – Are there many trivial or equivalent mutants?
• More experiments
A Semantic Mutation Tool for C
GUI of SMT-C*
GUI of Test Runner*

- Running results of test suites and testcases: statistics and the result for each test suite and testcase with graphical highlight;
- Progress bar;
- Test error traces.
Mutant generation

Mutant generation -- support three different scopes
Tool Architecture

- GUI - SMT Console Viewer, SMT Mutant Viewer, SMT Test Viewer, SMT Result Viewer, Others
- Function Components - Mutant generator, Mutant Manager, Test Runner, Mutant Builder, Result Analyser
- Third-party software - Eclipse, CDT, Autotools Plug-ins, TXL, CHECK
Implementation Overview

• The tool is developed using Java and as Eclipse plug-ins.
• It also can be published as an independent testing tool based on Rich Client Platform (RCP) of Eclipse.
• For current version, TXL is used to drive the semantic mutation and Check is used to support mutant compilation and running tests.
TXL – as a prototyping mutation engine

• It is a generalized source-to-source translation system.
• It takes as input an piece of source code, and a set of transformation.
• It produces as output the transformed source code.
• Example:
  – txl source1.c tranform_rule.txl
Semantic Mutation Operators

• Thirteen semantic mutation operators have been implemented.
  – ASD, MFC_R, FTA_F...

• 6 traditional mutation operators were also implemented for conducting experiments to compare traditional and semantic mutation operators.
  – SCRB, SSWM, SSDL ...
CHECK

• A unit testing framework for C.
• Check is based on Autotools.
• Many advanced features: run in fork mode (allow signal and early exit), test fixture, multiple suites in one runner, looping tests, test timeouts, determining test coverage, xml logging etc.
Future work of SMT-C

• Implement more semantic mutation operators.
• Improve the GUI, better integration with C development process.
• Enhance mutant generation function: mutant management, function scope mutation and efficiency.
• Accelerate the mutation generation and testing processes.