Description of muJava's Method-level Mutation Operators

Yu-Seung Ma
Electronics and Telecommunications Research Institute, Korea
ysma@etri.re.kr

Jeff Offutt
Software Engineering
George Mason University
offutt@gmu.edu

November 29, 2005
Correction December 2011
Update July 2016

This document provides a brief description of method-level mutation operators for Java used by muJava.

When designing method-level mutation operators for Java, we followed the selective approach [3]. The selective results found that the traditional operatorso of modifying operands and statements give little effectiveness to mutation testing. Therefore, we only consider mutation operators that modify expression by replacing, deleting, and inserting primitive operators. muJava provides six kinds of primitive operators; (1) arithmetic operator, (2) relational operator, (3) conditional operator, (4) shift operator, (5) logical operator, and (6) assignment. For some of them, muJava provides short-cut operators. This section presents designs of mutation operators for those six kinds of primitive operators. We try to design mutation operators that replace, insert, and delete the primitive operators. We defined total 12 method-level operators in Table 1. The detailed description for the operators are described in the following subsections, according to each primitive operator.

Furthermore, some of the operators are subdivided into two or three, according to the number and type of operand. For example, the AOR operator is subdivided into AOR_B (binary) and AOR_S (short-cut). (*Update 2016: the AOR_U (unary) operator was removed.*)

1 Arithmetic Operators

The Java programming language supports five arithmetic operators for all floating-point and integer numbers; (1) +, (2) -, (3) *, (4) /, and (5) %. These operators are all binary. However, both + and - have unary versions. Four short-cut arithmetic operators are defined; (1) op++, (2) ++op, (3) op−−, and (4) −−op.

- **AOR_B** : Arithmetic Operator Replacement
  Replace basic binary arithmetic operators with other binary arithmetic operators.
- ~~**AOR_U : Arithmetic Operator Replacement**~~
  ~~Replace basic unary arithmetic operators with other unary arithmetic operators.~~
  (*The **AOR_U** operator was removed in 2015.*)
- **AOR_S** : Arithmetic Operator Replacement
  Replace short-cut arithmetic operators with other unary arithmetic operators.

Operator	Description
AOR	Arithmetic Operator Replacement
AOI	Arithmetic Operator Insertion
AOD	Arithmetic Operator Deletion
ROR	Relational Operator Replacement
COR	Conditional Operator Replacement
COI	Conditional Operator Insertion
COD	Conditional Operator Deletion
SOR	Shift Operator Replacement
LOR	Logical Operator Replacement
LOI	Logical Operator Insertion
LOD	Logical Operator Deletion
ASR	Assignment Operator Replacement
Deletion operators added in 2013	
SDL	Statement DeLetion
VDL	Variable DeLetion
CDL	Constant DeLetion
ODL	Operator DeLetion

Table 1: The Method-level Mutation Operators for muJava

- **AOI**$_U$: Arithmetic Operator Insertion
  Insert basic unary arithmetic operators.
- **AOI**$_S$: Arithmetic Operator Insertion
  Insert short-cut arithmetic operators.
- **AOD**$_U$: Arithmetic Operator Deletion
  Delete basic unary arithmetic operators.
- **AOD**$_S$: Arithmetic Operator Deletion
  Delete short-cut arithmetic operators.

2 Relational Operators

A relational operator compares two values and determines the relationship between them. Java provide six kinds of relational operators; (1) >, (2) >=, (3) <, (4) <=, (5) ==, and (6) ! =. Because these operators take two operands, only replacement is allowed for the relational operators.

- **ROR** : Relational Operator Replacement
  Replace relational operators with other relational operators, and replace the entire predicate with *true* and *false*.

3 Conditional Operators

The Java programming language supports six conditional operators; five binary and one unary. Five binary conditional operators are (1) &&, (2) ||, (3) &, (4) |, and (5) ^. The one unary conditional operator is '!'.

- **COR** : Conditional Operator Replacement
  Replace binary conditional operators with other binary conditional operators.

- **COI** : Conditional Operator Insertion
  Insert unary conditional operators.
- **COD** : Conditional Operator Deletion
  Delete unary conditional operators.

4 Shift Operators

Java provides three shift operators; (1) >>, (2) <<, and (3) >>>>. A shift operator performs bit manipulation on data by shifting the bits of its first operand right or left. The shift operators should take two operand like the relational operators. Therefore, only replace mutation operators are defined.

- **SOR** : Shift Operator Replacement Replace shift operators with other shift operators.

5 Logical Operators

Java provides four logical operators to perform bitwise functions on their operands; three are binary and one is unary. Three binary logical operators are (1) &, (2) |, and (3) ^. One unary logical operator is ~.

- **LOR** : Logical Operator Replacement
  Replace binary logical operators with other binary logical operators.
- **LOI** : Logical Operator Insertion
  Insert unary logical operator.
- **LOD** : Logical Operator Delete
  Delete unary logical operator.

6 Assignment Operators

The basic assignment operator assigns the value of the right side expression (op2) to the left side variable (op1). In addition to the basic assignment operation, the Java programming language defines eleven short cut assignment operators that perform an operation and an assignment using one operator: (1) +=, (2) -=, (3) *=, (4) /=, (5) %=, (6) &=, (7) | =, (8) ^=, (9) <<=, (10) >>=, and (11) >>>=, are defined.

- **ASR**$_S$: Short-Cut Assignment Operator Replacement
  Replace short-cut assignment operators with other short-cut operators of the same kind.

7 Deletion Operators

The statement deletion operator was added in 2013 for Deng's ICST paper [2], and the other deletion operators were added after Delamaro et al.'s 2014 ICST paper [1]. The deletion operators delete statements, variables, constants, and objects. Note that this achieves much more than statement coverage. Statement coverage simply requires that each statement be reached, an SDL mutant can only be killed by a test that not only reaches the deleted statement, but that also causes the statement to have an effect on the output of the program.

- **SDL** : Statement DeLetion
  SDL delets each executable statement by commenting them out. It does not delete declarations. When applied to control structures that include a block of statements (for example, *if*, *while*, and *for*), the entire block is deleted, as well as each statement. Full details with examples are given in Deng et al.'s paper [2].

- **VDL** : Variable DeLetion
  All occurrences of variable references are deleted from every expression. When needed to preserve compilation, operators are also deleted.

- **CDL** : Constant DeLetion
  All occurrences of constant references are deleted from every expression. When needed to preserve compilation, operators are also deleted.

- **ODL** : Operator DeLetion
  Each arithmetic, relational, logical, bitwise, and shift operator is deleted from expressions and assignment operators. When removed from assignment operators (for example, "`X += 1`"), a plain assignment is left ("`X = 1`"). When a binary operator is removed, an operand must also be removed so the expression remains well formed (compilable). Thus deleting a binary operator produces two mutants; one where the left operand is deleted, and another where the right operand is deleted.

References

[1] Márcio E. Delamaro, Jeff Offutt, and Paul Ammann. Designing deletion mutation operators. In 7th IEEE International Conference on Software Testing, Verification, and Validation (ICST 2014), Cleveland, Ohio USA, March 2014.

[2] Lin Deng, Jeff Offutt, and Nan Li. Empirical evaluation of the statement deletion mutation operator. In 6th IEEE International Conference on Software Testing, Verification, and Validation (ICST 2013), pages 80–93, Luxembourg, March 2013.

[3] A. J. Offutt, A. Lee, G. Rothermel, R. Untch, and C. Zapf. An experimental determination of sufficient mutation operators. ACM Transactions on Software Engineering Methodology, 5(2):99–118, April 1996.