Empirical Evaluation of the Statement Deletion Mutation Operator

By Lin Deng, Jeff Offutt, Nan Li
George Mason University
ICST 2013

Presented by-
Maruf Ahmad
Computer Science, Stony Brook University.

---

Key Concepts Of this Paper

- Mutation Testing.
- A technique of mutation testing.

---

Motivation

- Do-fewer approaches
- Do-smarter approaches
- Do-faster approaches

---

APPROACH: THE STATEMENT DELETION MUTATION OPERATOR

- SDL (Statement deletion mutation operator) requires much more than statement coverage, because a killing test must cause the statement to affect the program's behavior.
- Here statement deletion will take place of a program with statement deletion operator.
- Then to observe, how efficient this is than the existing technique.
**APPROACH CONDITION**

- All possible cases: Every possible case must be considered.
- Boolean conditions: Most control structures have at least one Boolean condition, which should be deleted.
- Inner statements: Statements inside control structures must be deleted.
- Nested control structures: Nested control structures must be treated recursively.

**APPROACH DETAIL**

- SDL for general statement
- SDL for “while” statement
- SDL for “if” statement
- SDL for “for” statement
- SDL for “switch” statement
- SDL for “try-catch” block
- SDL for “return” statement

- For “single statement” When applied to control structures that include a block of statements, including “if” “while,” and “for” blocks, the entire block must be deleted.

- For “while” statements. Most while statements include a Boolean condition and a block of statements. The condition decides whether to execute the statements in the loop.

- For “if” statements. The if statement is complicated by the else clause, the “if” statement removes by replacing the condition with true.

- For “for” statements. Each statement inside the loop deletes here.

- For “switch” statements. The SDL operator works at two levels for the switch statement.

- For “try-catch” block” the SDL operator deletes the corresponding catch block unless another statement can raise the same exception inside the same try block.

- For “return” statement: It can not be done to delete a return statement in Java for compiler error so remove the effect of the return statement by changing the value to the default value of the appropriate primitive type.
Presentation on Empirical Evaluation of the Statement Deletion Mutation Operator

SDL for single statement

```java
public void test0() {
    int a, b;
    a = 1;
    b = 2;
}
```

```java
public void test0() {
    int a, b;
    if (a == 0) {
        b = 3;
        for (int i = 0; i < 5; ++i) {
            a = a + b + c;
            b = b + c;
        }
    }
    a = a + b + c;
    b = b + c;
}
```

SDL for "while" statement

```java
public void testWhile() {
    int a, b, c, t;
    while (a < 5) {
        t = t + b + c;
    }
}
```

```java
public void testWhile() {
    int a, b, c, t;
    if (a == 0) {
        t = t + b + c;
    } else {
        t = t + b + c;
    }
}
```

SDL for "for" statement

```java
public void testFor() {
    int a, b, c;
    for (int i = 3; i < 10; i++) {
        a = a + b + c;
        b = b + c;
    }
}
```

```java
public void testFor() {
    int a, b, c;
    if (a == 0) {
        for (int i = 0; i < 5; i++) {
            a = a + b + c;
            b = b + c;
        }
    }
    a = a + b + c;
    b = b + c;
}
```

SDL for "switch" statement

```java
public void testSwitch() {
    int x = 5;
    switch (x) {
    case 10: numbering is "Out";
        break;
    case 11: numbering is "Now";
        break;
    default: numbering is "Invalid";
        break;
    }
}
```

```java
public void testSwitch() {
    int x = 5;
    switch (x) {
    case 10: numbering is "Out";
        break;
    case 11: numbering is "Now";
        break;
    default: numbering is "Invalid";
        break;
    }
}
```
**SDL for “try-catch” block**

```java
public void testTry() {
    try {
        int i = in.readLine();
        input = in.parseInt(i); // subscript
    } catch (IOException e) {
        print("Could not read input");
    } catch (NumberFormatException e) {
        print("Must be numeric");
    }
}
```

**Example Of SDL equivalent mutant**

```java
private final static NameList[] NO_NAMES = new NameList[];
public NameList[] getNameList (String goodName) {
    List result = new ArrayList();
    for (int i = 0; i < theNames.size(); i++) {
        if (name.get(name).equals (goodName)) {
            result.add (name);
        }
    }
    return NO_NAMES;
    return (NameList[]) result.toArray (new NameList [result.size()]);
}
```

**Experiment evaluation**

- First, they implement the SDL operator for Java, which involves a number of subtle decisions on the language constructs.
- Second, they evaluate its benefits in terms of how well tests generated to kill only SDL mutants perform when run on all of muJava’s method-level mutants.
- For experiment the idea is implemented in muJava mutation system to facilitate a direct comparison.
- Here used 40 java classes as experimental subject.
- This is taken from textbook example and open source projects.
- Test to kill the SDL mutants were generated by hand.
- It is evaluated against all of muJava’s mutation operators.
Experiment Result

• 41% fewer equivalent mutant found than existing muJava technique.
• The mutation score is 12% higher than muJava technique.
• Better mutant killing rate than muJava.

Conclusion

• This is an effective approach as consumed less time than the existing approach.
• This approach can be added with the existing approach to give the tester a wide facility of testing.

Thank You

Any Question Please?