Piacente Cristian 866020

Homework H10 – Mutants

Software Quality, Academic Year 2023-2024, University of Milan – Bicocca

Generate one invalid, one valid-but-not-useful, and one useful mutants for the following program using the provided mutant operators. For each mutant, show the test cases in the provided test-suite that contribute to its killing (This is pseudocode: assume that the == operator between string returns true if the two string are the same. Example: “ciao”==“ciao”-> true, “ciao” == “caiocaio” -> false).

For this exercise, assume that a mutant is useful only if it is killed by at most 1 given test case.

```
1   boolean f(String[] a, String b[]) {
2       for (int i = 0; i < a.length; i++) {
3           String s1 = a[i];
4           boolean found = false;
5           for (int j = 0; j < b.length; j++) {
6               String s2 = b[j];
7               if (s1 == s2) {
8                   found = true;
                }
            }
9           if (!found) {
10              return false;
            }
        }
11      return true;
    }
```

MUTANT OPERATORS:

A op B --> B op A, where op is a comparison operator
A op1 B --> A op2 B, where op1 and op2 are 2 different comparison operators
++A --> A++
c1 --> c2, where c1 and c2 are two different constants
TRUE --> FALSE
FALSE --> TRUE
== --> =
= --> ==

TEST SUITE:

[“software”], [“software”] --> TRUE
[“software”, “quality”], [“software”] --> FALSE
[“usi”, “quality”, “software”], [“software”, “quality”] --> FALSE

Let’s start generating an invalid mutant for the given program: a mutant is invalid if it’s not syntactically correct.

We can obtain such variant by applying the mutant operator `== --> =` on **line 7**.
Here is the resulting **invalid mutant**, which would generate a compilation error in many programming languages (Java for sure):

```
1   boolean f(String[] a, String b[]) {
2       for (int i = 0; i < a.length; i++) {
3           String s1 = a[i];
4           boolean found = false;
5           for (int j = 0; j < b.length; j++) {
6               String s2 = b[j];
7               if (s1 = s2) {
8                   found = true;
                }
            }
9           if (!found) {
10              return false;
            }
        }
11      return true;
    }
```

Piacente Cristian 866020

Homework H10 – Mutants

Software Quality, Academic Year 2023-2024, University of Milan – Bicocca

A **valid-but-not-useful mutant** is a syntactically correct mutant which is killed by more than 1 given test case.

We can generate such mutant by applying the mutant operator **FALSE --> TRUE** on **line 10**.
Here is the resulting **valid-but-not-useful mutant**:

```
1   boolean f(String[] a, String b[]) {
2       for (int i = 0; i < a.length; i++) {
3           String s1 = a[i];
4           boolean found = false;
5           for (int j = 0; j < b.length; j++) {
6               String s2 = b[j];
7               if (s1 == s2) {
8                   found = true;
                }
            }
9           if (!found) {
10              return true;
            }
        }
11      return true;
    }
```

It's simple to show that it's not a useful mutant: **the method always returns true**, so the **2nd and** the **3rd test case** in the test suite (shown below) **kill the mutant**, which means that the mutant is killed by 2 given test cases.

TEST SUITE:
["software"], ["software"] --> TRUE
["software", "quality"], ["software"] --> **TRUE**
["usi", "quality", "software"], ["software", "quality"] --> **TRUE**

We can now generate the last mutant, which must be a **useful mutant** (killed by at most 1 given test case).

We can create such mutant by applying the mutant operator **TRUE --> FALSE** on **line 11**.
Here is the resulting **useful mutant**:

```
1   boolean f(String[] a, String b[]) {
2       for (int i = 0; i < a.length; i++) {
3           String s1 = a[i];
4           boolean found = false;
5           for (int j = 0; j < b.length; j++) {
6               String s2 = b[j];
7               if (s1 == s2) {
8                   found = true;
                }
            }
9           if (!found) {
10              return false;
            }
        }
11      return false;
    }
```

It's simple to show that it's a useful mutant, if we notice that **the method always returns false**.
It means that **only** the **1st test case** in the test suite (shown below) **kills the mutant**, so it's useful.

TEST SUITE:
["software"], ["software"] --> **FALSE**
["software", "quality"], ["software"] --> FALSE
["usi", "quality", "software"], ["software", "quality"] --> FALSE