Software testing 2014

ASSIGNMENT 2 - REPORT

November 19, 2014

Stephane KI
Abo Akademy University
Student Number 71364
I.T - Software Engineering
`stephane.ki@abo.fi`

Table of contents

1 Part 1 - Traditional mutants

Assignment : Chose 5 of the mutants that could not be killed, compute the RIP conditions for each, and tell if any of them could be weakly killed.

2 Mutants

2.1 Mutant 1- AOIS_19

```
rslt=rslt * left;
//BECOMES
rslt=rslt++ * left;
```

1. Reachability

   ```
   right != 0 && right >=2
   ```

2. Infection

   ```
   false
   ```

3. Propagation

   ```
   false
   ```

   Infection and propagation are false because after the post incrementation of the variable, the value is override.

   Example: left=3 and right=2. The loop will be executed one time and the statement *rslt=rslt++ * left* can be translated like that:

   ```
   int u= rslt*left;
   rslt=rslt+1;
   rslt=u;
   return rslt;
   ```

   So the mutant is equivalent.

2.2 Mutant 2 - AOIS_20

```
rslt=rslt * left;
//BECOMES
rslt=rslt-- * left;
```

1. Reachability

```
right != 0 && right >=2
```

2. Infection

```
false
```

3. Propagation

```
false
```

Infection and propagation are false because after the post decrementation of the variable, the value is override.

Example: left=3 and right=2. The loop will be executed one time and the statement *rslt=rslt- - * left* can be translated like that:

```
int u= rslt*left;
rslt=rslt -1;
rslt=u;
return rslt;
```

So the mutant is equivalent.

2.3 Mutant 3 - AOIS_25

```
return rslt;
//BECOMES
return rslt++;
```

1. Reachability

```
true
```

2. Infection

```
true
```

3. Propagation

```
false
```

The propagation is *false* because the variable is incremenented after the return statement ; so the returned result is not modified.

2.4 Mutant 4 - AOIS_256

```
return rslt;
//BECOMES
return rslt--;
```

1. Reachability

   ```
   true
   ```

2. Infection

   ```
   true
   ```

3. Propagation

   ```
   false
   ```

   The propagation is *false* because the variable is decremenented after the return statement ; so the returned result is not modified.

2.5 Mutant 5 - ROR_4

```
if (right == 0) {
}
//BECOMES
if (right <= 0) {
}
```

1. Reachability

   ```
   true
   ```

2. Infection

   ```
   right < 0;
   ```

   But knowing that the precondition says that *right* should be greater or eqaul to zero so the previous condition is false.

   The Mutant is equivalent!

3. Propagation (Does not mater ! ! !)

3 Part 2 - Class-level mutants

4 Mutants

4.1 Mutant 1- JID

```
static double result = 0;
//BECOMES
static double result;
```

This mutant is equivalant because initializing an attribute to null or 0 is redundant as in Java, primitives default to 0 (or false) and object references default to null. This is not the case for local variables however.

So in our case the Java Initialization Deletion mutant and the original code are the same.