SOFTWARE TESTING: SAMPLE ANSWERS TO TUTORIAL 4

Here are sample answers to the activities for tutorial 4 on mutation testing.

TEST SUITES

I've collected nine test suites from tutorials here, although I've been a little unfair and only taken the first three tests from each. I also record whether they achieve statement, branch, modified condition/decision or all-uses coverage. Notice how subsumption means that if you achieve MC/DC you always achieve branch coverage and how if you achieve branch coverage you always achieve statement coverage, *but* since there's no simple subsumption relation between these and all-uses coverage, it's possible to achieve all-uses coverage without necessarily achieving the others.

Test suite	Test number	t[]	l	u	Expected	Statement coverage	Branch coverage	MC/DC	All-uses coverage
0	1	{1, 2, 3, 4, 5}	1	5	3	✓	✓	✓	✓
	2	{}	0	1	0				
	3	{1, 2}	0	1	0				
1	1	{}	1	2	0	✓	✓	✓	×
	2	{1, 2}	1	2	0				
	3	{0, 1, 2}	0	2	1				
2	1	{5}	2	4	0	✓	×	×	✓
	2	{3}	2	4	1				
	3	{3, 4, 5}	2	6	3				
3	1	{}	1	10	0	✓	✓	✓	×
	2	{3, 4, 5}	1	2	0				
	3	{2, 4, 6}	3	5	1				
4	1	{}	1	10	0	✓	×	×	×
	2	{0}	0	0	0				
	3	{0}	-1	1	1				
5	1	{3, 4, 5, 6}	4	8	2	✓	✓	×	✓
	2	{}	4	8	0				
	3	{9}	9	10	0				
6	1	{}	1	1	0	×	×	×	×
	2	{2}	0	0	0				
	3	{2}	3	3	0				
7	1	{}	1	1	0	✓	✓	✓	✓
	2	{1, 2, 3, 4}	1	4	2				
	3	{5, 6, 7, 8}	1	2	0				
8	1	{0, 3, 5}	0	5	1	✓	✓	✓	×
	2	{0, 1, 2, 3}	1	3	1				
	3	{}	0	1	0				

MUTANTS

Here's the original code:

Segment(): count elements in range

```
public int Segment(int t[], int l, int u) {

   // Assumes t is in ascending order, and l < u.
   // Counts the length of the segment
   // of t with each element l < t[i] < u.

   int k = 0;

   for(int i = 0; i < t.length && t[i] < u; i++)
      if(t[i] > l)
```

```
        k++;

    return k;
}
```

Here are ten sample mutants. Note that the last two break the rules set out in Activity 2: they're from when we got desperate after all of our mutants got killed by well-written test suites…

- Mutant M8 takes advantage of the fact that test suites often focus on small number; using a `byte` will store counts up to 127, but any test which is supposed to return a larger number will fail.
- Mutant M9 is also trying to trick small test suites: using `k = 1 << k;` will generate the sequence of values 0, 1, 2, 4, 8, … for *k* instead of 0, 1, 2, 3, 4, … — this mutant will survive any test suite which doesn't contain a test with a return value larger than 2.

Mutant M0: change constants

```
for(int i = 1; i < t.length && t[i] < u; i++)
```

Mutant M1: < → <=

```
for(int i = 0; i <= t.length && t[i] < u; i++)
```

Mutant M2: < → <=

```
for(int i = 0; i < t.length && t[i] <= u; i++)
```

Mutant M3: > → >=

```
    if(t[i] >= l)
```

Mutant M4: < → >

```
for(int i = 0; i > t.length && t[i] < u; i++)
```

Mutant M5: < → >

```
for(int i = 0; i < t.length && t[i] > u; i++)
```

Mutant M6: > → <

```
    if(t[i] < l)
```

Mutant M7: statement duplication

```
        { k++; k++; }
```

Mutant M8: change types

```
byte k = 0;
```

Mutant M9: change expressions

```
        k = 1 << k;
```

Mutant M10: statement duplication

```
    if(t[i] > l) if(t[i] > l)
```

MUTATION ADEQUACY

Here I show how each test suite does against each of the mutants: a green tick (✓) indicates that the test suite kills the mutant, while a red cross (✕) indicates that the mutant survived the test suite. In the final column I record each test suite's mutation adequacy as a percentage.

Note that Mutant M10 survives *all* of the test suites, and on closer examination, unlike M8, it turns out to be an equivalent mutant (duplicating the `if` statement is like saying `if(A && A)` — exactly the same as `if(A)`, so long as `A` has no side effects). Consequently M10 is not counted when computing mutation adequacy (= *#killed_mutants* / (*#mutants* - *#equivalent_mutants*)).

Test suite	Statement coverage	Branch coverage	MC/DC	All-uses coverage	M0	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	Mutation adequacy
0	✓	✓	✓	✓	✕	✓	✓	✓	✓	✓	✓	✓	✕	✓	✕	80%
1	✓	✓	✓	✕	✕	✓	✓	✓	✓	✓	✓	✓	✕	✕	✕	70%

2	✓	×	×	✓	✓	✓	×	×	✓	✓	✓	✓	×	✓	×	70%
3	✓	✓	✓	×	×	✓	×	×	✓	✓	×	✓	×	×	×	40%
4	✓	×	×	×	✓	✓	×	×	✓	✓	✓	✓	×	×	×	60%
5	✓	✓	×	✓	×	✓	×	✓	✓	✓	✓	✓	×	×	×	60%
6	×	×	×	×	×	✓	×	×	×	✓	✓	×	×	×	×	30%
7	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓	✓	×	×	×	70%
8	✓	✓	✓	×	×	✓	✓	✓	✓	✓	✓	✓	×	×	×	70%

It's interesting to see that while the suite with the worst mutation adequacy has the worst coverage and the suite with the best mutation adequacy has full coverage, there's no simple correlation between test suite coverage and mutation adequacy in between these two.

Version 1.1, 2018/01/09 13:42:03

Informatics Forum, 10 Crichton Street, Edinburgh, EH8 9AB, Scotland, UK
Tel: +44 131 651 5661, Fax: +44 131 651 1426, E-mail: school-office@inf.ed.ac.uk