Fundamentals of Software Testing
(A.Y. 2022/2023) – Solutions
June 16^{th}, 2023

Exercise 1.

Our company has been asked to develop a very simple components that should enable asynchronous communications among two systems A and B. In particular the developed component should permit to system A to store an information on our component that can be successively fetched by system B. Anyway after two consecutive store actions by system A our component should alert A that it is not possible to store the information until system B makes a fetch. On the other hand, in case system B tries to fetch an information while component A did not store any unread information, our component shold notify B with an error message. You are asked by the PM to deeply test the behaviour of the component.
Select the most suitable test case derivation strategy and provide a specification for the test case composing the test suite.

Solution:

The descrition suggests that the software component changes its behaviour according to an internal state. The following FSM can be used to represent the behaviour (the corresponding graphical representation is reported in Figure 1):

- $I = \{store, fetch\}$
- $O = \{ok, nok\}$
- $Q = \{empty, one, full\}$
- "*empty*" is the initial state
- $\delta = \{(empty, store, one), (one, store, full), (full, store, full), (full, fetch, one), (one, fetch, empty), (empty, fetch, empty)\}$
- $\omega = \{(empty, store, ok), (one, store, ok), (full, store, nok), (full, fetch, ok), (one, fetch, ok), (empty, fetch, nok)\}$

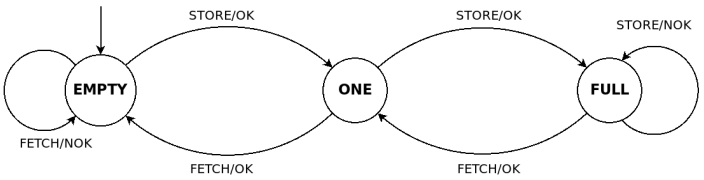

Figure 1: FSM for the two positions buffer component

Given the specification a possible test suite can be obtained applying the W-method strategy, as reported in the following:

Step 1. In order to reduce space the following syntactical substitutions are applied:

- $0 = emtpy, 1 = one, 2 = full, s = store, f = fetch, o = ok, n = nok$

Step 2. The k-equivalence partitions are reported in Table 1.

δ	s	f
0	1	0
1	2	0
2	2	1

ω	s	f	Part.
0	o	n	1
1	o	o	2
2	n	o	3

Table 1: k-equivalence partitions

Step 3. The resulting W set is derived as:

- $W = \{s, f\}$

Step 4. The transition cover set P is computed as:

- $P = \{\epsilon, s, f, sf, ss, ssf, sss\}$

Step 5. The resulting test suite is (where we assume number of state in the real system to be equal to 3 so that $Z = W$):

- $TS = P \times Z = \{s, f, ss, sf, fs, ff, sfs, sff, sss, ssf, ssfs, ssff, ssss, sssf\}$

Exercise 2.

Consider the following program:

```
1  enum Discount {nodisc, normal, high}
2  enum Course {antipasto, primo, secondo}
3
4  public double computeBill(Course[] order, Discount disc) {
5    int i = 0;
6    double total = 0.0;
7    double discount = 0.0;
8    while (i < order.length) {
9      if (order[i] == Course.antipasto) total = total + 10;
10     if (order[i] == Course.primo) total = total + 15;
11     if (order[i] == Course.secondo) total = total + 30;
12     i= i + 1;
13   }
14   if (disc == Discount.high) discount = 0.15;
15   else if (disc == Discount.normal) discount = 0.05;
16   total = total * (1 - discount);
17   return total;
18 }
19
```

- Provide a minimal test suite that fully satisfy the condition/decision coverage adequacy criterion.
- Derive a data-flow graph for the program above and discuss if the test suite derived at the previous step satisfies any data-flow adequacy criteria.
- Compute the c-use coverage for the test suite derived at the previous step and in case it is not equal to 1 discuss why and how a test suite satisfying the c-use criterion can be derived..

Solution

To answer to the first request the most simple strategy seems to be a manual analysis of the source code so to identifify the needed test cases. We have to note that all the decisions are simple and then decision coverage corresponds to condition coverage. We can note that the following test suite (among many others) satisfies the request:

- $TS = \{$

  $t_1 =<$ `[],Discount.nodisc` $>$

  $t_2 =<$ `[Course.antipasto,Course.primo,Course.secondo],Discount.high` $>$

  $t_3 =<$ `[Course.antipasto],Discount.normal` $>$

  $\}$

To answer to the second request we have, at first, to identify the blocks. In the following we report the identified blocks where some lines are split as they include both a condition (i_{cond}) and a command (i_{comm}):
$B_1 = \{4, 5, 6, 7\}$, $B_2 = \{8\}$, $B_3 = \{9_{cond}\}$, $B_4 = \{9_{comm}\}$, $B_5 = \{10_{cond}\}$, $B_6 = \{10_{comm}\}$, $B_7 = \{11_{cond}\}$, $B_8 = \{11_{comm}\}$, $B_9 = \{12\}$, $B_{10} = \{14_{cond}\}$, $B_{11} = \{14_{comm}\}$, $B_{12} = \{15_{cond}\}$, $B_{13} = \{15_{comm}\}$, $B_{14} = \{16, 17\}$.

The Data-Flow Graph reported in Figure 2 permits to answer to the second request. From the graph we can derive the following conclusions:

- All the p-uses for the definition `order` are covered (no c-uses are included for `order`)

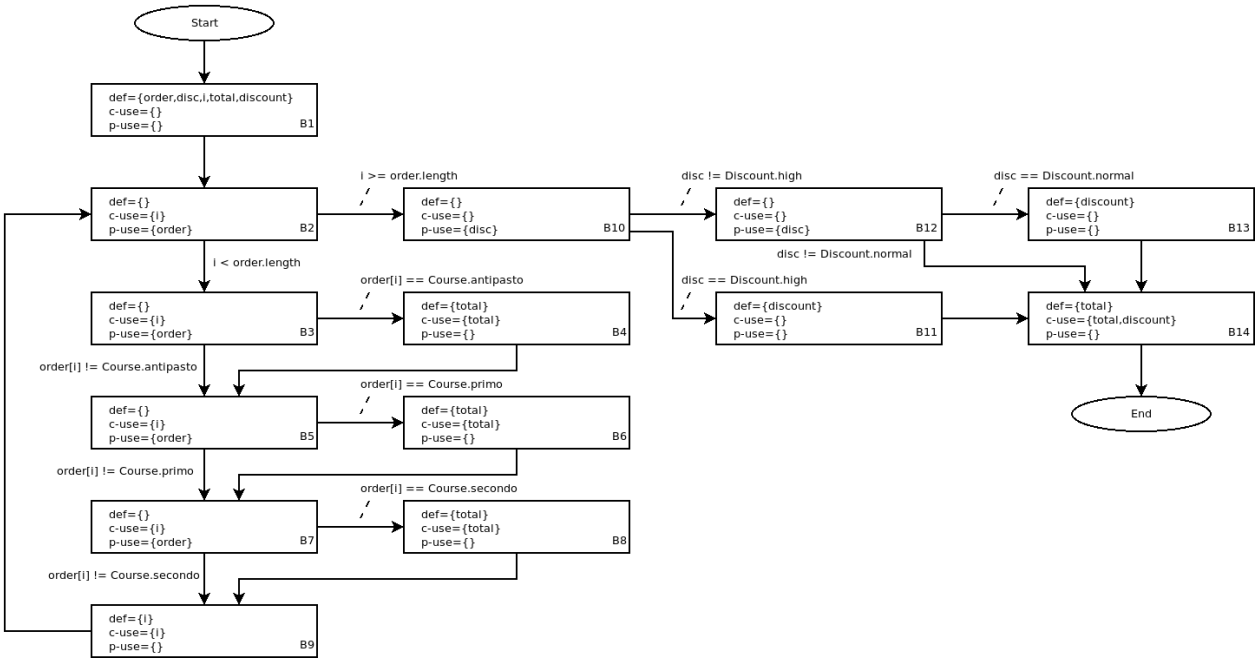

Figure 2: Data Flow Graph

- All the p-uses for the definition of `disc` are covered (no c-uses are included for `disc`)
- All the c-uses for the definition of `i` are covered (no p-uses are included for `i`)
- All the c-uses for the definitions of `discount` are covered (no p-uses are included for `discount`)
- Not all the c-uses are covered for `total` in particular there are dcu corresponding to different iterations of the `while` statement that are not covered. For instance the definition of `total` at node 1 is live at node 6 and this path is not covered by the test suite. Similarly for the definition at node 4 that has a path in which the definition is live at node 8 and this is not covered by the test suite.

In summary we can declare that the test suite fully satisfy the "p-use" criterion but not the "c-use" criterion.

The last point of the exercise asks to derive a value for the adequacy in relation to the c-use criterion. In order to do this we have to identify all the def-use pairs for `total` and to count the covered ones (see Table 2).

Defined at node (n)	**dcu**(`total`,n)
1	{4,6,8,14}
4	{6,8,14}
6	{4,8,14}
8	{4,6,14}

Table 2: dcu for the `total` variable

Table 3 reports the def-use pairs for `total` (in bold) that are covered by the test cases included in the test suite. From the data reported in the table we can conclude that the c-use coverage is equal to $\frac{5}{13} = 0,385$.

Defined at node (n)	**dcu**(`total`,n)
1	{**4**,6,8,**14**}
4	{**6**,8,14}
6	{4,**8**,14}
8	{4,6,**14**}

Table 3: dcu for the `total` variable

To fully satisfy the "c-uses" coverage criterion we add the following tests:

$t_4 =<$ `[Course.primo,Course.antipasto,Course.secondo,Course.primo],Discount.nodisc` $>$

$t_5 =<$ `[Course.secondo,Course.antipasto],Discount.nodisc` $>$