HÁSKÓLI ÍSLANDS
Iðnaðarverkfræði-, vélaverkfræði- og tölvunarfræðideild

HBV205M: Prófun hugbúnaðar / Software Testing · Spring 2021
Dr. Helmut Neukirchen

Assignments 5/6 · Due 22.2.2021, 10:00

Assignment 5

Objectives: Train creating and annotating CFGs with def/use, achieving CFG-based and data-flow-based coverage, using Cyclomatic number as upper bound for branch coverage.

Consider the Java method `search` below.

1. Draw the control-flow graph (CFG) of method `search`. The nodes shall contain the corresponding line numbers of the source code (split `for` loops into multiple nodes such as '3init", "3cond", etc.). You have to think less, if you use nodes for closing brackets.
2. Enumerate (using a sequence of node numbers) CFG paths that you need to get 100% *statement coverage*.
3. Enumerate additional CFG paths that you need to get 100% *branch coverage*.
4. What is the upper bound of the number test cases needed to achieve branch coverage?
5. Enumerate additional CFG paths that you need to get 100% *boundary-interior path coverage* (i.e. exterior path, boundary path, interior path) if you have already 100 % branch coverage. Focus only on the number of loop iterations: you do not need to cover all further possible combinations that are added due to the possible paths inside the body of the loop – you can restrict on one sample path through the body of the loop.
6. What is the number of different paths that are possible through the CFG depending on the number of iterations? Provide the number as a function of parameter `length`, i.e. number of paths when doing *exactly* `length` iterations.
7. Add defs/uses annotations[1] to your CFG[2] and check (and justify) whether the *all p-uses* criterion is achieved by using your paths from 100% branch coverage. (No need to actually enumerate any additional paths.) *Hint:* Do not forget that incrementing a loop counter is a def that then influences the two p-use edges of the loop condition.

```
1 int search(int[] array, int length, int searchItem) {
2   int foundIndex=-1;
3   for (int i = 0; i < length; i++) {
4     if (array[i] == searchItem) {
5       foundIndex=i;
6     }
7    }
8   return foundIndex;
9 }
```

If you have a Google account, work online on a copy of the following template:
`https://docs.google.com/spreadsheets/d/1U4_YGU42Z8ldDcyyoeZ7ehZN5pxRLYRk2T2FB68F4vc/copy`

If you have no Google account, you can view and download a copy of the template from here:
`https://docs.google.com/spreadsheets/d/1U4_YGU42Z8ldDcyyoeZ7ehZN5pxRLYRk2T2FB68F4vc/edit?usp=sharing`

[1] Remember that def and c-use are annotations of nodes, p-use annotate all edges controlled by that p-use.

[2] Parameters (and global variables and fields of a class) can be considered to be defined (def) when a function or method is entered. An array index is –even when used within a predicate– typically considered as c-use.

Assignment 6

Objectives: Train condition coverage-based white-box testing involving short-circuit evaluation and conditions that are not feasible.

Consider the Java method `classify` on the next page that performs a classification of triangles. Derive test cases based on different condition coverage criteria:

1. Assume that a minimal set of test cases for 100% branch coverage (which is in this case the same as Decision coverage) has already been created. For which of the conditions do you have to create additional test cases to obtain 100% Basic Condition coverage (i.e. Branch Condition Testing)? Just name the conditions (and justify your answer)! – No need to actually create these test cases: we will do this partialy in item 2. below.

2. Now, consider from those conditions that you have identified in item 1. above that condition that comes first in the listing and create logical and concrete test cases to obtain 100% Basic Condition coverage (i.e. Branch Condition Testing) for that condition!

   Hints:

   (a) Java uses short-circuit evaluation of Boolean expressions. Hence, for a Boolean expression *(A or B)*, the term B will not be evaluated as soon as A is true. Hence, for 100% Basic Condition coverage, the logical test cases $TC_1 : A = true, B = -$; $TC_2 : A = false, B = true$; $TC_3 : A = false$, $B = false$ are required (in TC_1, B does not matter as it is anyway not evaluated, hence the "don't care" symbol "–" is used in the logical test case).

   (b) The considered Java function `classify` has dependencies between the final values of a, b, and c due to the conditions 1–3 (see source code); therefore, not all combinations of these final values are possible in the code in line 20 and later.

3. Now, check whether you need to add further logical test cases to achieve Minimal Multiple Condition Coverage for the condition considered in item 2. based on the coverage obtained by your test cases created in item 2. (No justification needed.) If yes, add logical and concrete test cases to obtain 100% Minimal Multiple Condition Coverage.

   Hint:

   - For a composed Boolean condition that contains no nested parentheses, only all atomic expressions and the overall expression need to be considered, but no other combinations. Hence, for a composed Boolean expression *(A or B or C)*, only the four parts A; B; C; and *(A or B or C)* need to be considered.

If you have a Google account, work online on a copy of the following template:
`https://docs.google.com/spreadsheets/d/1CsXz_33SqezgWHf3Cep4ermwvXRt4tEjSW8bLQSCIHw/copy`

If you have no Google account, you can view and download a copy of the template from here:
`https://docs.google.com/spreadsheets/d/1CsXz_33SqezgWHf3Cep4ermwvXRt4tEjSW8bLQSCIHw/edit?usp=sharing`

```
1  public static int classify( int a, int b, int c ) {
2    int type;
3    // Sort values to achieve that ((a <= b <= c))
4    int help;
5    if (c < a) { // Condition 1
6      help = c;
7      c = a;
8      a = help;
9    }
10   if (c < b) { // Condition 2
11     help = c;
12     c = b;
13     b = help;
14   }
15   if (b < a) { // Condition 3
16     help = b;
17     b = a;
18     a = help;
19   }
20   // After the above lines, we have guaranteed that ((a <= b <= c))
21
22   // Classify triangle
23   if ((a < 0) || (b < 0) || (c < 0) || ((c-a) > b)) {  // Condition 4
24     type = 0; // NO TRIANGLE, ILLEGAL VALUES
25   }
26   else {
27     if ((a == 0) || (b == 0) || (c == 0) || ((c-a) == b)) { // Condition 5
28       type = 1; // NO TRIANGLE, BUT IN FACT A LINE
29     }
30     else {
31       if ((a == b) || (b == c)) { // Condition 6
32         if (a == c) { // Condition 7
33           type = 2; // EQUILATERAL TRIANGLE
34         }
35         else {
36           type = 3; // ISOSCELES TRIANGLE
37         }
38       }
39       else {
40         type = 4; // SCALENE TRIANGLE
41       }
42     }
43   }
44   return type;
45 }
```

To keep in mind during flipped classroom sessions

- Have audio working (use smartphone with Zoom app in case of problems, also non-audio problems. In case of crashes, download/install latest Zoom version).
- Switch on video to create a more personal atmosphere.
- First: Solution of last week's assignment presented.
- Next: Questions concerning slides/videos discussed.
- Then: New assignments introduced.
- Finally: Work on assignments in breakout rooms (random allocation of 2 students each).
  - Breakout rooms do not get recorded.
  - First thing to do:
    1. Remember breakout room number: in case you need to reconnect, Helmut can then add you again to your old breakout room.
       * **if Helmut remembers to enable, you can also join breakout rooms yourself** / if you join late, you can add yourself to an empty breakout room or one that has only one student.
    2. Exchange contact details (email/phone), e.g. via chat in each breakout room, so that you can continue in case of technical problems/after class finishes.
  - Create some shared document to work together on the solution. You can also share your screen inside your breakout room.
  - Use "Ask for help" button (question mark icon) to make Helmut join (but may take time if he is busy in another breakout room).
    * Redo "Ask for help" if Helmut does not come after a few minutes (the requests do not queue up, but Helmut sees them only displayed once). In case of being idle, Helmut will come along each breakout room.
  - Submit one PDF per team to Gradescope (unless you disagree on solution): use Gradescope's group submission feature. Gradescope is reachable via Canvas. You have until Monday for submission (re-submission possible).
  - In case of some announcement for all: Helmut sends a chat message to all or ends all breakout room sessions for a video session with all. (If Helmut does not mess it up, should be possible to continue in the old breakout room.)
- There is no wrap-up at end of class, i.e. if you are finished with your assignment, there is no need to wait – you are welcome to leave the class.
- Typically, there are no Wednesday classes: use the time to read new slide/watch videos as preperation for next Monday.
  - But: **next week Monday, 22.2.2021, elementary schools have vaction and Helmut (and maybe some students) are be busy with their kids: we might rather use the Wednesday, 24.2.2021, 8:30(!) for teaching instead, but assignment will be published already on Monday... (To be announced later).**