Spring 2012 Course: CS 498, Section DM

Software Testing

Problem Set 5
Assigned: April 21, 2012
Due: Thursday, April 26, 2012 (at 12:30pm, beginning of class)

This problem set covers the material from Chapter 3 of the textbook. There are five problems, worth total of 120 points. You need only 100 points to get the maximum score for this problem set, which accounts for 15% of the final grade. If you have more than 100 points, you will get credit for the past problem sets and the project. For the subproblems labeled **[MP]**, submit your code to SVN. For the non-MP problems, you can either include your solutions in SVN (if so, please use simple ASCII documents to write your solutions) or you can bring your (hand-written or printed) solutions to the lecture. **Important note:** you cannot collaborate on the first four problems, but members of each group should collaborate on Problem 5, without groups collaborating with one another!

Problem 1 [32 points]: (This is a modified version of exercises after Section **3.2** (page 119, Chapter 3); the entire text is here.) For each of the two predicates $P_1=a\wedge(\neg b\vee c)$ and $P_2=(a\wedge b)\vee(b\wedge c)\vee(a\wedge c)$, do the following:
(a) [2 points]: Identify the clauses that go with the predicate P.
(b) [2 points]: For each clause X, show all values for other clauses that make X determine the value of P. (You can compute and simplify P_X as shown in the book, or you can use an ad-hoc approach.)
(c) [2 points]: Write the complete truth table for the predicate with rows going from TT…T to FF…F (changing first clause 'a' then 'b' then 'c'). Label your rows starting from 1. (Use the format in the example underneath the definition of combinatorial coverage in Section 3.2.) You should include columns for the conditions under which each clause determines the predicate and a column for the predicate itself.
(d) [2 points]: Identify all pairs of rows from your table that satisfy **GACC** with respect to each clause.
(e) [2 points]: Identify all pairs of rows from your table that satisfy **CACC** with respect to each clause.
(f) [2 points]: Identify all pairs of rows from your table that satisfy **RACC** with respect to each clause.
(g) [2 points]: Identify all 4-tuples of rows from your table that satisfy **GICC** with respect to each clause. Identify any infeasible GICC test requirements.
(h) [2 points]: Identify all 4-tuples of rows from your table that satisfy **RICC** with respect to each clause. Identify any infeasible RICC test requirements.

Problem 2 [20 points]: [MP] (This is a modified version of Exercise **1** after Section **3.3** (page 130, Chapter 3); the entire text is here.) Consider the method **checkIt** below:

```
public static void checkIt(boolean a, boolean b, boolean c) {
   if (a && (b || c)) {
      System.out.println("P is true.");
   } else {
      System.out.println("P is false.");
   }
}
```

(a) [4 points]: [MP] Transform **checkIt** to **checkItExpand**, a method where each **if** statement tests exactly one boolean variable as discussed in Section 3.3.1 (pages 127-129).
(b) [8 points]: [MP] Instrument **checkItExpand** and add additional code to record which edges are traversed by a **test set** not only one test. For example, your instrumentation should say that calling **checkItExpand** with {TTT, FFF} covers some edges say branch1-then, branch1-else, and branch2-then.

You can assign unique ids to edges instead of using descriptive names. **Hint**: you probably do not want to just add print statements to the code.
(c) [4 points]: [MP] Derive a GACC test set T_G for **checkIt**. Write your test set in JUnit.
(d) [4 points]: [MP] Derive an Edge Coverage test set T_E for **checkItExpand**. Build T_E so that it does **not** satisfy GACC on the predicate in **checkIt**, or show this to be impossible. Write your test set in JUnit.

Problem 3 [28 points]: (This is a modified version of Exercise **5** after Section **3.3** (page 131, Chapter 3); the entire text is here.) This problem considers the **TestPat** class from page page 56, Chapter 2. (This is the same class from Problem 4 in Problem Set 2, Problem 5 in Problem Set 3, and Problem 3 in Problem Set 4.) Identify the predicates and find test sets for the following coverage criteria:
(a) [7 points]: Predicate Coverage
(b) [7 points]: Clause Coverage
(c) [7 points]: Combinatorial Coverage
(d) [7 points]: Correlated Active Clause Coverage
Your tests should ensure reachability. You do **not** need to derive the expected outputs, but you can use the test cases from Table 2.5 on pages 59-60, which provide the expected outputs.

Problem 4 [20 points]: [MP] Consider the following specification for a **sort** method:

```
public static void sort(int[] s)
    // Precondition: s != null
    // Effects: sort elements of s (smaller elements first)
```

Suppose that **sort** is checked in the following way, with the code provided in SVN:
(1) Make a copy **t** of **s**.
(2) Apply the method **sort** to **s**.
(3) Verify that every element in **s** is also an element in **t**.
(4) Verify that every element in **t** is also an element in **s**.
(5) Verify that **s[i]<=s[i+1]** for appropriate values of **i**.
(a) [5 points]: [MP] Do we need to test **sort** with **s** being **null**? What is an expected "output" in that case? Provide **explanation** for your answers; simple "yes" or "no" is not sufficient.
(b) [5 points]: [MP] The provided code that implements the checks has a few problems with array index bounds. Identify and correct those problems.
(c) [5 points]: [MP] Find initial and "sorted" values for **s** such that all of the given checks (with correct implementations for array index bounds) succeed, but the sorted version of **s** is still wrong. Propose an additional check or modify existing checks to detect this (counter) example with "sorted" values.
(d) [5 points]: [MP] Write a specification for the **binarySearch** method provided in SVN and implement a simple generic check for that method.

Problem 5 [20 points]: For some part of JPF (it can be the Config class or the part you chose in Problem Set 4 or a completely new part), do the following:
(a) [5 points]: Describe at least 4 conditions when this part should produce some warning or error, e.g., "class name already exists" or "config parameter missing". Write a predicate for all your conditions.
(b) [5 points]: Based on your predicate, prepare test cases that satisfy some coverage criterion. Write your test cases in JUnit and/or external files, as appropriate; update bug counts on Wiki. Discuss what criteria are appropriate to use. How many test inputs would there be in "Clause Coverage" for the predicate that you created?
(c) [5 points]: Identify another situation in the JPF code where to apply logic coverage in a non-trivial manner (e.g., find an "if" statement with a predicate that has three or more clauses). Apply logic coverage and discuss how you applied it.
(d) [5 points]: Update the potential bug count for your group in the table on the Projects page on Wiki: https://wiki.engr.illinois.edu/display/cs498dmsp12/Projects Prepare the info for submitting at least 4 total bug reports to JPF. Write a simple text file with all you'd submit (but don't submit the reports yet).