Revision of Testing/Debugging

CS4271
Abhik Roychoudhury
National University of Singapore

Exercise - 1

A Fibonacci sequence is given as follows.

- fib(0) = 1
- fib(1) = 1
- fib(n) = fib(n − 1) + fib(n − 2) for n ≥ 2.

Exercise - 1

- Following is a (buggy) program to compute the first n fibonacci numbers. What is the bug? Suggest a fix to the bug.
- Will the effect of the bug is observable in the printed values (note that you do not need to start computing all the printed values). Explain your answer.

The program in Ex. 1

```
int fib(int n){
   int f, f0 =1, f1 = 1;
   while (n>1){ n=n-1; f = f0+f1; f0 = f1; f1 = f;} return f;
}
void main(){
    int n = 99;
   while (n > 0){
     printf("fib(%d)=%d\n",n,fib(n)); n=n-1; }
}
```

Answer to Ex-1

Answer:

- The bug is in the lack of initialization of f in the fib procedure. It should be initialized to 1.
- fib(1) will print a garbage value, and this will be observable.

Ex-2

One method of software testing for inputs with large domains is called "equivalence partitioning". In this method, the domain of an input variable is partitioned into equivalence classes, so that from each equivalence class only one test input will be tried out. Now, there is a wide choice of when we define two test inputs to be "equivalent" and put them into an equivalence class. Suppose we define two test inputs to be equivalent when they produce the same path in the program.

- Give an example program, where such an equivalence partitioning will lead to efficient testing, that is, only few test cases to try.
- Give an example program, where such an equivalence partitioning will lead to inefficient testing, that is, too many test cases to try.

Answer to Ex-2

- A matrix multiplication program. Irrespective of the input matrices, there is only one program path to execute.
- An insertion sort program. The number of equivalence classes to try out is the number of

possible permutations in the input array

Ex-3

Consider the following program.

- if (x > 2) y = x - z else y = x + z;
- if (y > 0) return 0 else return 1;

(a) Give one sample test input for which the above code will return the value 0.

(b) Characterize the set of all test inputs which cause the above code to return the value 0. Explain your answer.

Answer to Ex-3

- x == 3, y == 2
- $(x > 2 \land x - z > 0) \lor (x \leq 2 \land x + z > 0)$
  - This is obtained by combining the path conditions of the paths which lead to the statement saying return 0

Exercise 4 (Hard!!)

- Suppose you want to use a model checker (such as the SPIN tool discussed in class) to generate test cases of a terminating sequential program written in a C-style imperative language. What are the temporal properties you can feed in to meet the statement coverage, edge coverage and condition coverage criteria for test generation?.

Answer – Ex 4

- Statement Coverage: We can employ model checking on a CFG like structure which is at the source code level and where every statement is identified. We will have a clearly marked end node in the CFG; we can put a self-loop here to make the traces non-terminating (a rather technical point). For each statement s, we can then check a property of the form
- $G(\neg s \lor G \neg end)$

Answer – Ex 4

- Edge coverage: Again, we can employ model checking on program CFG , check the following property for every edge e
  - $G(\neg e \lor G \neg end)$
- Condition coverage: Here, we need to employ model checking on a source level CFG. For every branch edge b in the control flow graph again we check
  - $G(\neg b \lor G \neg end)$