DIT635 Software Quality and Testing

Assignment 3:

Fault-Based Testing and Model-Based Testing and Verification

Group 20:

Erdem Halil
George-Vlad Liteanu
Taofik Arnouk

13/03/2022

5 Pages

Problem 1 - Mutation Testing (45 Points)

1. Create six mutants for classes from the CoffeeMaker project. Your report should include the mutated code, noting how it differs from the original code. **(20 Points)**

The mutations are in the code.

2. Assess the test suite that you created for **Assignment 2** using the set of mutants that you derived.
    - Identify and list which tests expose which mutants, knowing that test cases that expose a mutant pass on the original code and fail on the mutated code. (10 Points).
    - Describe why the exposing tests expose each mutant. For any non-equivalent mutants not exposed, note why they were not exposed. (10 Points)

Mutant 1 (changing sugar to something that does not exist): this will affect all if not the majority of the tests, since changing sugar to something else will lead to the whole program not compiling. Sugar is needed throughout the whole program.

Mutant 3 (changing >= to <=): The test would fail since we are changing the relational operator, in which we would be adding sugar when we have none or negative amounts of it (First test in addSugarStructural.java file).

Mutant 4 (Changing int amtPaid = 0 to amtPaid = 1): This will affect the tests where the amtPaid is used in the ordering of drinks and etc... This will cause the amount given to be off the original given (so 70 will become 71), for example, this would cause the expected change to be different from the given (change should be 20 but the user gets 21). (This applies to the tests in the makeCoffeeStructural.java).

    - Describe why the exposing tests expose each mutant. For any non-equivalent mutants not exposed, note why they were not exposed. (10 Points)

Mutant 6 (addition of && amtPaid > 0): the test does not fail since we are adding a mutant that is going to be equivalent to the default condition.

Mutant 2 (changing >= 0 to > -1): This will lead to no change in the tests.

Mutant 5 (Add "&& !name.equals(" ")"): This will only lead to an exception if the recipe name is empty.

- Then, design additional tests that will detect the remaining mutants and describe why they succeed in detecting them, highlighting what differentiates the new test from others. **(5 Points)**

For mutant 6: this test will throw an error since we would be running and paying with a negative amount (balance)

```
@Test
public void testMakeCoffee_negative() throws RecipeException {
    CoffeeMaker cm = new CoffeeMaker();
    Recipe r = new Recipe();
    r.setPrice("50");
    cm.addRecipe(r);
    assertEquals(50, cm.makeCoffee(0, -1));
}
```

For mutant 2: this test will not throw an error since the mutation is equivalent to the original code.

Mutant 5: This test will throw an exception if the recipe name is empty.

```
@Test
public void testAddRecipe() {
    Recipe r5 = new Recipe();
    try {
        r5.setName("");
        r5.setAmtChocolate("0");
        r5.setAmtCoffee("0");
        r5.setAmtMilk("4");
        r5.setAmtSugar("1");
        r5.setPrice("40");
        cm.addRecipe(r5);
    } catch (RecipeException e) {
        fail("RecipeException should not be thrown.");
    }
    Recipe[] recipes = cm.getRecipes();
    assertEquals(4, recipes.length);
    assertEquals(r5, recipes[0]);
}
```

Problem 2 - Finite-State Verification (55 Points)

1. We have modelled a traffic light controller for a crossroad with two-way traffic on each side.

- The system includes synced traffic lights on the North/South and East/West side of the road.
- At a time, lights can be red, yellow, or green.
- Under normal circumstances, green/red light persists for 45 seconds whereas a yellow one - for 3 seconds.
- The system has sensors to detect cars on the North/South and East/West side in order to avoid traffic jams.
- The system also tracks any emergency car (ambulance, police car, etc). If an emergency car is nearby, all the traffic in the crossroad is stopped, meaning that lights go red immediately. This is done under the assumption that the system is going to be used in a civilized country where drivers are aware of the “zipper” method where drivers move to the nearest side of the road (the ones on the left lane move to the left, the ones on the right lane move to the right) in order to make way for the emergency car to pass in the middle. For that, we also assume that the roads are wide enough for at least 3 cars.
- Due to time and design constraints, our system does not include buttons for pedestrians to request passage. What is more, we couldn’t think of a real-life application of this in a four-way crossroad. We believe it’s not done as it might mess with the natural and optimal cycle of lights at the crossroad.
- Traffic lights on the North/South and East/West sides of the road can’t be green at the same time
- If the pedestrian lights on the East/West side are green, then the traffic flow must be from North to South (or from South to North, depending on your perspective). This means that traffic lights on East/West must be red.
- If there’s more traffic on one side of the road, the system should be able to react to that and decrease the “green” time for the side of the road where there isn’t much traffic.

2. You can find our Finite State Model in the .smv file.

3. Safety properties are included at the very end of the .smv file.

4. Liveness properties are included at the very end of the .smv file.

5. Below is a transcript of our NuSMV session. Only one liveness property does not hold.

```
*** This is NuSMV 2.6.0 (compiled on Thu Mar  4 12:49:32 2021)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

-- specification AG (traffic_light_NS = GREEN -> AF traffic_light_EW != GREEN)  is true
-- specification AG (emergency_sensor = TRUE -> AF (traffic_light_NS != GREEN & AF traffic_light_EW != GREEN))  is true
-- specification EF (traffic_light_NS = GREEN -> EF traffic_light_EW = RED)  is true
-- specification (EF traffic_light_NS = YELLOW & EF traffic_light_EW = YELLOW)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    traffic_light_NS = GREEN
    traffic_light_EW = RED
    ped_light_NS = STOP
    ped_light_EW = WALK
    current_traffic_flow = NS
    time_left_green = 45
    time_left_yellow = 0
    time_left_red = 0
    car_sensor_NS = FALSE
    car_sensor_EW = FALSE
    car_waiting_time_reduction = FALSE
    emergency_sensor = FALSE
-- specification AG (traffic_light_NS = GREEN -> AF ped_light_NS != WALK)  is true
-- specification EF (ped_light_NS = STOP -> EF ped_light_EW = WALK)  is true
```