DIT635 - Assignment 3: Fault-Based Testing and Model-Based Testing and Verification

Due Date: Sunday, March 14th, 23:59 (Via Canvas)

There are two questions worth a total of 100 points. You may discuss these problems in your teams and turn in a single submission for the team (zipped archive) on Canvas. Answers must be original and not copied from online sources.

Cover Page: On the cover page of your assignment, include the name of the course, the date, your group name, and a list of your group members.

Peer Evaluation: All students must also submit a peer evaluation form. This is a separate, individual submission on Canvas. Not submitting a peer evaluation will result in a penalty of five points on this assignment.

Problem 1 - Mutation Testing (45 Points)

In this question, you will apply Mutation Testing to the code from the CoffeeMaker example from Assignment 2. The CoffeeMaker code can be found at:
https://canvas.gu.se/courses/42587/files/folder/Assignments?preview=4149962

1. Create six mutants for classes from the CoffeeMaker project (before you apply any of your fixes from Assignment 2). Your report should include the mutated code, noting how it differs from the original code. **(20 Points)**
    a. One mutant must be **invalid** (does not compile).
    b. One must be **equivalent** to the original code (you inserted a fault, but no test case can possibly yield a different solution to the original code).
    c. Two mutants must be **valid-but-not-useful** (all tests, or almost all tests, will expose this mutation).
    d. Two mutants must be **useful** (only a small number of specific tests will expose this mutation).
    e. You must apply at least four different mutation operators, and you must use at least one mutation operator from each of the three categories in the attached handout (for more information, see Chapter 16 of Software Testing and Analysis).
    f. You do not have to use the same classes or methods for all mutant categories. Try mutating different parts of the code. You may use any class except Main or the exceptions.
2. Assess your test suite that you created for Assignment 2, with respect to the set of mutants that you derived.Then, write additional tests that can expose the remaining non-equivalent mutants.
    a. Test cases that expose a mutant pass on the original code and fail on the mutated code.
    b. Identify and list which tests expose which mutants **(10 Points)**.
    c. Describe why the exposing tests expose each mutant. For any non-equivalent mutants not exposed, note why they were not exposed. **(10 Points)**
    d. Then, design additional tests that will detect the remaining mutants. Describe why they detect those mutants. **(5 Points)**

Problem 2 - Finite-State Verification (55 Points)

For this exercise, you are required to create a finite-state model of a traffic-light controller and verify its properties using the NuSMV symbolic model-checker (download from http://nusmv.fbk.eu/)

- Assume that the controller manages traffic and pedestrian lights at the intersection of two roads, both with two-way traffic.
- Pedestrians can request access to cross the road by pressing a “walk button”.
- Assume that the system has traffic sensors for each direction to detect if vehicles are present and waiting to pass through, which allows the system to manage traffic flow efficiently by varying the amount of time the lights are green for each road/direction based on demand. Your model should capture and represent this notion of varying time in some manner (i.e., do not completely abstract away time).
- There are emergency vehicle sensors for each direction which lets the system provide priority access for emergency vehicles by switching lights appropriately.

You may state and make any other reasonable simplifying assumptions that you need. A simplified traffic light model appears in the slides for Lecture 14. Understanding that model is a good first step in solving this problem.

In your submission, you must address the following:

1. Define the scope and the requirements for the system that you intend to model – a brief description of what you have modeled, any assumptions that you have made and the key requirements you expect the system to satisfy. **(10 Points)**
2. Build a finite state model of the system in the NuSMV language. Be sure to write sufficient comments. (Though not required, you may find drawing state diagrams helpful). **(20 Points)**
3. Write at least three **safety** properties (“something bad must never happen”) in temporal logic (CTL or LTL) that must be satisfied by the system. Explain your properties and state which system requirements those properties are derived from. **(10 Points)**
4. Write at least three **liveness** properties (“something good must eventually happen”) in temporal logic (CTL or LTL) that must be satisfied by the system. Explain your properties and state which system requirements those properties are derived from. **(10 Points)**
5. Verify your properties on your system using the NuSMV symbolic model checker and provide a transcript of your NuSMV session. **(5 Points)**