CSCE 747 - Final
Name:

This is a 150-minute exam. On all essay type questions, you will receive points based on the quality of the answer - not the quantity.

Make an effort to write legibly. Illegible answers will not be graded and awarded 0 points.

There are a total of 10 questions and 100 points available on the test.

Question 1

On multiple choice questions, more than one solution may be correct. Select all that apply.

1. A program may be correct, yet not reliable.
    a. True
    b. False

2. If a system is on an average down for a total 30 minutes during any 24-hour period:
    a. Its availability is about 98% (approximated to the nearest integer)
    b. Its reliability is about 98% (approximated to the nearest integer)
    c. Its mean time between failures is 23.5 hours
    d. Its maintenance window is 30 minutes

3. In general, we need either stubs or drivers but not both, when testing a module.
    a. True
    b. False

4. Which of the following may be logically inferred from the post-condition of a sorting routine, sort(array, size) that sorts elements in ascending order?
    a. size > 0
    b. ∃ i, j, 0 <= i < j < size : array[i] = array[j]
    c. ∀ i, j, 0 <= i < j < size : array[i] < array[j]
    d. ∀ i, j, 0 <= i < j < size : array[i] <= array[j]

5. If a temporal property holds for a finite-state model of a system, it holds for any implementation that conforms to the model.
    a. True
    b. False

6. Self-check oracles do not require the expected output for judging whether a program passed or failed a test.
    a. True
    b. False

7. Object-oriented design and implementation typically have an impact on verification such that OO specific approaches are required for:
    a. Unit Testing
    b. Integration Testing
    c. System Testing
    d. Acceptance Testing

Question 2

Metaheuristic search techniques can be divided into local and global search techniques. Define what a “local” search and a “global” search is. Contrast the two approaches. What are the strengths and weaknesses of each?

Question 3

Consider the following function:

```
bSearch(A, value, start, end) {
        if (end <= start)
                return -1;
        mid = (start + end) / 2;
        if (A[mid] > value) {
                return bSearch(A, value, start, mid);
        } else if (value > A[mid]) {
                return bSearch(A, value, mid+1, end);
        } else {
                return mid;
        }

}
```

Give an example, with a brief justification, for each of the following kinds of mutants that may be derived from the code by applying mutation operators of your choice. Do not reuse a mutation, even if it fits multiple categories.

1. Equivalent Mutant
2. Invalid Mutant
3. Valid, but not Useful
4. Useful Mutant

Question 4

Suppose that finite state verification of an abstract model of some software exposes a counter-example to a property that is expected to hold for true for the system. Briefly describe what follow-up actions would you take and why?

Question 5

You are building a web store that you feel will unseat Amazon as the king of online shops. Your marketing department has come back with figures stating that - to accomplish your goal - your shop will need an **availability** of at least 99%, a **probability of failure on demand** of less than 0.1, and a **rate of fault occurrence** of less than 2 failures per 8-hour work period.

You have recently finished a testing period of one week (seven full 24-hour days). During this time, 972 requests were served to the page. The product failed a total of 64 times. 37 of those resulted in a system crash, while the remaining 17 resulted in incorrect shopping cart totals. When the system crashes, it takes 2 minutes to restart it.

1. What is the rate of fault occurrence?
2. What is the probability of failure on demand?
3. What is the availability?
4. What additional information would you need to calculate the mean time between failures?
5. Is the product ready to ship? If not, why not?

Question 6

Temporal Operators: A quick reference list.

- G p: p holds globally at every state on the path
- F p: p holds at some state on the path
- X p: p holds at the next (second) state on the path
- p U q: q holds at some state on the path and p holds at every state before the first state at which q holds.
- A: for all paths from a state
- E: for some path from a state

Consider a finite state model of a traffic-light controller similar to the one discussed in the homework, with a pedestrian crossing and a button to request right-of-way to cross the road.

State variables:

- **traffic_light: {RED, YELLOW, GREEN}**
- **pedestrian_light: {WAIT, WALK, FLASH}**
- **button: {RESET, SET}**

Initially: **traffic_light = RED, pedestrian_light = WAIT, button = RESET**

Transitions:
pedestrian_light:

- **WAIT → WALK if traffic_light = RED**
- **WAIT → WAIT otherwise**
- **WALK → {WALK, FLASH}**
- **FLASH → {FLASH, WAIT}**

traffic_light:

- **RED → GREEN if button = RESET**
- **RED → RED otherwise**
- **GREEN → {GREEN, YELLOW} if button = SET**
- **GREEN → GREEN otherwise**
- **YELLOW→ {YELLOW, RED}**

button:

- **SET → RESET if pedestrian_light = WALK**
- **SET → SET otherwise**
- **RESET → {RESET, SET} if traffic_light = GREEN**
- **RESET → RESET otherwise**

1. Briefly describe a safety-property (nothing “bad” ever happens) for this model and formulate it in CTL.
2. Briefly describe a liveness-property (something “good” eventually happens) for this model and formulate it in LTL.
3. Write a trap-property that can be used to derive a test case using the model-checker to exercise the scenario “pedestrian obtains right-of-way to cross the road after pressing the button”.

Question 7

Consider the following C function:

```
int sum(int arr[], int n){
      int s = 0;
      while (n > 0){
            n = n - 1;
            s = s + arr[n];
      }
      return s;
}
```

1. What are the pre and post-conditions of this method?
2. For each line of code, derive the state predicates that would be collected by symbolic execution.

Question 8

Search-based test generation generally does not use coverage criteria directly in generation, but instead makes use of a scoring function to determine how clost tests are to achieving the current generation goal (such as coverage of the criterion). This is called the fitness function, or objective function.

Explain the fitness function formulation used by search-based generation to achieve branch coverage of a program. It may be helpful to come up with a code example.

Question 9

Consider a simple microwave controller modeled as a finite state machine using the following state variables:

- Door: {Open, Closed} -- sensor input indicating state of the door
- Button: {None, Start, Stop} -- button press (assumes at most one at a time)
- Timer: 0...999 -- (remaining) seconds to cook
- Cooking: Boolean -- state of the heating element

Formulate the following informal requirements in CTL:

1. The microwave shall never cook when the door is open.
2. The microwave shall cook only as long as there is some remaining cook time.
3. If the stop button is pressed when the microwave is not cooking, the remaining cook time shall be cleared.

Formulate the following informal requirements in LTL:

1. It shall never be the case that the microwave can continue cooking indefinitely.
2. The only way to initiate cooking shall be pressing the start button when the door is closed and the remaining cook time is not zero.
3. The microwave shall continue cooking when there is remaining cook time unless the stop button is pressed or the door is opened.

Question 10

You are testing the following method:
```
public double max(double a, double b);
```

Devise four executable test cases for this method in the JUnit notation.