

**Problem 3.1.** (5 points) Which of the following statements are true?

1. 100% path coverage implies 100% branch coverage
2. 100% statement coverage implies 100% branch coverage
3. Suppose test suite  $T_1$  gets 100% branch coverage and test suite  $T_2$  gets 100% statement coverage. There can be an execution path explored in  $T_2$  but not by  $T_1$ .
4. There is a program such that no test suite can get 100% branch coverage.
5. There are incomparable coverage metrics; i.e., there are two coverage metrics such that 100% coverage for metric 1 does not imply 100% coverage for metric 2 and vice versa.

**Problem 3.2.** (10 points) Many programs (e.g., for communication protocols) are written as finite state machines, for example, in a language like Statecharts. The program has a set of “states”, a set of “input events”, and a set of “output events”. The program starts in some initial state. At each state, the program waits for an input event. Depending on the current state and the input event received, the program transitions to a new state, and can optionally emit an output event.

1. (5 points) Consider the following problem. We want to design an elevator controller for a building with  $N$  floors. A panel of  $N$  buttons is located inside the elevator car to request that the elevator move to a given floor. Each floor in the building also has a button panel, which has an up and a down button to request that the elevator stop at the floor and move in the corresponding direction. The elevator must service all the requests in one direction before it can move in the opposite direction. When the elevator arrives at a floor en route to another destination and no request has been made inside the elevator for that floor, nor has a request been made at that floor’s button

panel for movement in the same direction, the elevator continues on to its next destination without stopping or opening the door. If such a request has been made, however, then the elevator stops and opens the door. The door is always opened for a duration of  $T_{stop}$  at which point it closes (you can assume there is an event  $T_{stop}$  that says  $T_{stop}$  time has passed since the door was opened). When the elevator arrives at a floor that is the last request in its direction of movement, the door opens and then its behavior depends on the situation in the building. If the button panel at the elevator's location has requested movement in the same direction, the user must get in and push the desired floor on the elevator's button panel before the door has finished closing. Otherwise, the elevator is free to move in the opposite direction to service another request, if one exists.

Design a finite state machine controller for the elevator. Assume  $N=3$ . Note that when the floor is 1, then "down" should be ignored, and when the floor is 3, "up" should be ignored.

You can use either the UML Statecharts notation or a simple "circle and arrow" notation for the finite state machine.

2. (5 points) Suggest some test coverage metrics for the finite state machine programming model. Can you get 100% coverage on these metrics for your elevator controller?

**Problem 3.3.** (5 points) Here is the binary search code that was implemented in Java's JDK and used widely for many years:

```
1: public static int binarySearch(int[] a, int key) {
2:     int low = 0;
3:     int high = a.length - 1;
4:
5:     while (low <= high) {
6:         int mid = (low + high) / 2;
7:         int midVal = a[mid];
8:
9:         if (midVal < key)
10:            low = mid + 1
11:         else if (midVal > key)
12:            high = mid - 1;
13:         else
```

```
14:             return mid; // key found
15:         }
16:         return -(low + 1); // key not found.
17: }
```

(a) Find tests to achieve 100% branch coverage. (b) This code has a bug. Can you find a test that demonstrates the bug?