ECE 153A/253, CS 153A - Homework 4 Solutions

1. (a) (5 points)

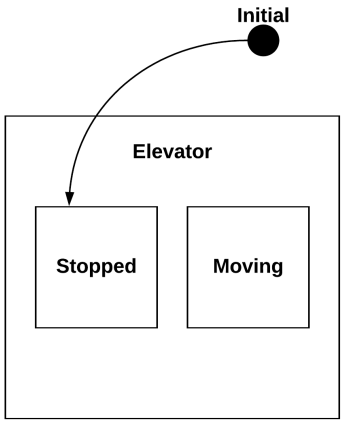

Figure 1: State Hierarchy

(b) Input signals: F1_SIG, F2_SIG, F3_SIG, F4_SIG, F5_SIG, TICK_SIG, PRINT_SIG, TERMINATE_SIG
State variables: floor_req_curr[5], floor_pen[5], curr_dir, curr_floor, stop_time, move_time, floor_curr_call_time[5], floor_calls[5], floor_total_time[5] (5 points)

(c) Q_TRAN() is used to transition to a new state, Q_SUPER() denotes the parent state which is queried if the current state doesn't have defined actions for an input, Q_HANDLED() is used to indicate that an input signal has been handled, but no state transitions are needed. (5 points)

(d) If a call to that floor is not already pending, it is marked as requested, updated to pending, the floor call time is recorded and number of calls to that floor is incremented. (5 points)

(e) First, pending floors are checked. If the current floor is pending, it is checked if the elevator has been stopped for 10 seconds. If 10 seconds have not passed, the state variable that counts the number of seconds stopped at a floor (stop_time) is incremented. If the current floor is pending but the elevator has been stopped for 10 seconds, stop_time is reset to 0, the pending flag for that floor is cleared, **it is checked if any other floor is pending, and if yes, the elevator transitions to the moving state. If the current floor is not pending, it is checked if any other floor is pending, and if yes, he elevator transitions to the moving state.** (5 points)

(f) It is checked if the elevator has been moving for 5 seconds. If 5 seconds have not passed, the state variable that counts the number of seconds moving between floors (move_time) is incremented. If 5 seconds have passed, move_time is reset to 0, the current floor is updated (by adding direction to the value of the current floor), **it is checked if the current floor is pending, and if yes, the elevator transitions to the stopped state.** If 5 seconds have passed, but the current floor is not pending, all other requested floors are updated to pending and the elevator stays in the moving state. (5 points)

2. The average times to service floor requests for different floor call frequencies are shown below: (20 points)

Table 1

Call Time (s)	Average Service Time (s)				
	Floor-1	Floor-2	Floor-3	Floor-4	Floor-5
200	10.0	7.0	6.0	7.0	10.0
100	10.0	7.0	6.0	7.0	10.0
50	10.0	7.0	6.0	7.0	10.0
20	15.1	11.1	10.2	11.1	15.1
10	43.4	27.2	21.5	27.2	43.4

3. After modeling a door that causes the stop time to be a random variable between 6 and 15 seconds, average times to service floor requests for different floor call frequencies are shown below: (25 points)

Table 2

Call Time (s)	Average Service Time (s)				
	Floor-1	Floor-2	Floor-3	Floor-4	Floor-5
200	10.0	7.0	6.0	7.0	10.0
100	10.0	7.0	6.0	7.0	10.0
50	10.0	7.0	6.0	7.0	10.0
20	16.4	12.4	11.2	12.4	16.4
10	47.7	30.7	24.2	30.6	47.7

4. The average time to service an emergency is 12.3 seconds. If the elevator is already stopped at a floor (other than floor-1) when an emergency occurs, but has been stopped for less than the stop time (after modeling the door), it was modeled that it will stay stopped for its scheduled stopped time, allowing all people to enter/exit the elevator at that floor safely. On the other hand, if the elevator is moving when an emergency occurs, it continues to move in its current direction, arrives at the next floor, does not open the door and then proceeds to floor-1. (20 points)

5. The current code shows several issues such as:

   - The actions taken on call to different floors are very close to being identical, but the code is redundant and the actions are re-written. A better way to organize the code would be to dispatch a common signal for call to any floor, and use a state variable to indicate the floor number. This would also make the HFSM scalable to any number of floors.
   - Several of the actions taken on call to different floors inside the two states “moving” and “stopped” are also common. By using hierarchy and executing the common actions inside of the parent state would further improve the code organization and make the code more compact.
   - The code uses too many complex guards which makes it difficult to understand at several places. Using more states to handle distinct behavior as opposed to complex guard statements would also improve the code organization and make it easier to modify and debug.
   - It would also help to have a more clear distinction between the basic elevator functionality and data gathering parts of the code. (5 points)