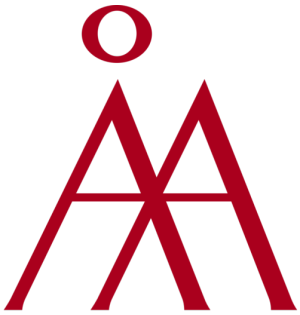

Åbo Akademi University

Software Testing

Assignment 3

Luis Araújo(2004624)

May 18, 2021

Contents

Chapter 1

Task 1

1.1 Implement the finite state machine model of the Theft Alarm in ModelJunit. Name the file TheftAlarmModel.java

First, to create the finite state machine model it was needed to develop all action methods:

- **variables & constructor**

```
public class TheftAlarmModel implements FsmModel {

    private enum State {DO,DC,AC,AO};

    private State state;

    private boolean ready;


    public TheftAlarmModel () {
        state = State.DO;
        ready = false;
    };
```

- **arm**

```
@Action
public int arm() {
    int res = 1;
    ready = true;

    if(state == State.AC || state == State.AO || state == State.DO) res = 0;
        else state = State.AC;

    return res;
};
```

- **disarm**

```
@Action
public int disarm() {
    int res = 2;

    if(state == State.DC || state == State.DO) res = 0;

    if(state == State.AO) state = State.DO;
    else if(state == State.AC) {
        state = State.DC;
        res = 1;
    }

    ready = false;

    return res;
};
```

- **closeDoors**

```
@Action
public int closeDoors() {
    int res = 1;
    if(state == State.AC || state == State.DC) res = -1;

    if(state == State.DO && !ready) {
        state = State.DC;
        res = 0;
    }

    if(state == State.DO || state == State.AO) state = State.AC;

    return res;
};

public boolean closeDoorsGuard() {
    return state != State.DC && state != State.AC;
}
```

- **openDoors**

```
@Action
public int openDoors() {
    int res = 0;

    if(state == State.DO || state == State.AO) res = -1;

    if(state == State.AC) {
        state = State.AO;
        res = 27;
    }

    if(state == State.DC) state = State.DO;

    return res;
};

public boolean openDoorsGuard() {
    return state != State.DO && state != State.AO;
}
```

- **reset & get method**

```
public void reset(boolean b) {
    state = State.DO;
    ready = false;
}

@Override
public Object getState() {
    return state;
}
```

After, the finite state machine model could be created:

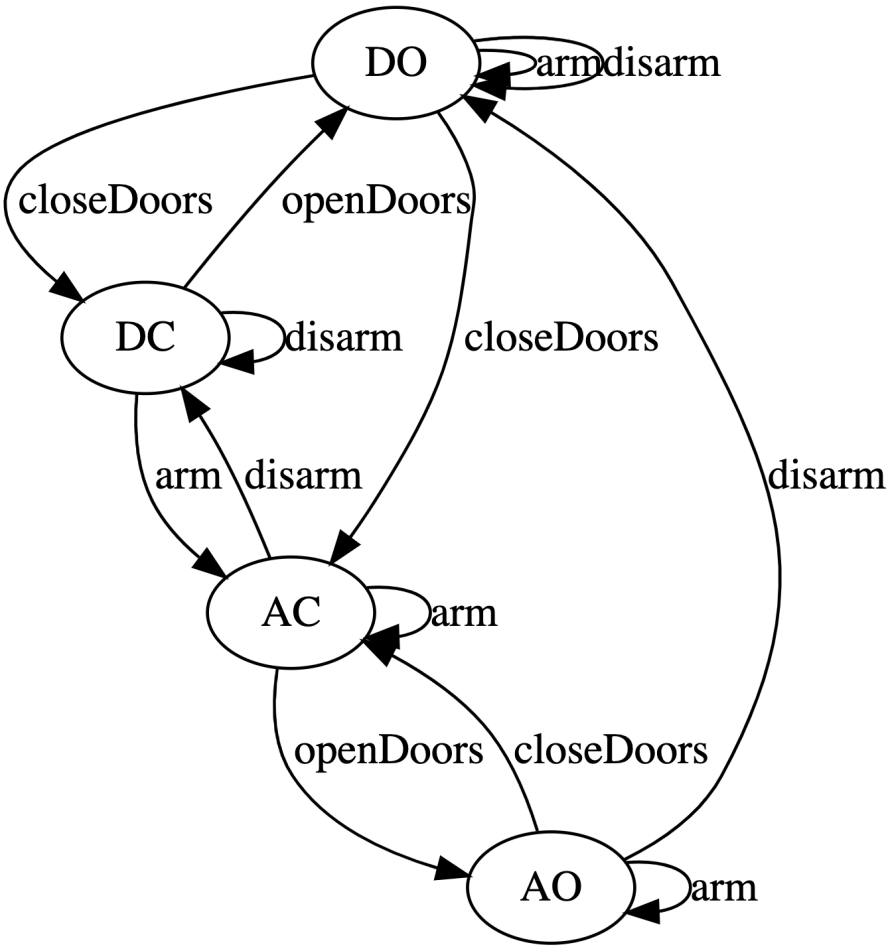

Figure 1.1: Finite State Machine Model of the Theft Alarm

Chapter 2

Task 2

2.1 Generate abstract tests and report model coverage achieved

In this second task, the two testers were created:

```
@Test
public void randomTester() throws Exception {
    System.out.println("_____Random_____");
    TheftAlarmModel theft = new TheftAlarmModel();

    Tester tester = new RandomTester(theft);
    tester.buildGraph();
    tester.buildGraph().printGraphDot("randomModel.dot");
    System.out.println(tester.buildGraph().getGraph());


    tester.addListener(new VerboseListener());
    //tester.addListener(new StopOnFailureListener());

    tester.addCoverageMetric(new ActionCoverage());
    tester.addCoverageMetric(new StateCoverage());
    tester.addCoverageMetric(new TransitionCoverage());
    tester.addCoverageMetric(new TransitionPairCoverage());

    System.out.println("\n_____Generate tests_____");
    tester.generate(2);

    System.out.println("\n_____Coverage report_____");
    tester.printCoverage();

};
```

Figure 2.1: Random Tester

```
@Test
public void greedyTester() throws Exception {
    System.out.println("_____Greedy_____");
    TheftAlarmModel theft = new TheftAlarmModel();

    Tester tester = new GreedyTester(theft);
    tester.buildGraph();
    tester.buildGraph().printGraphDot("greedyModel.dot");
    System.out.println(tester.buildGraph().getGraph());


    tester.addListener(new VerboseListener());
    //tester.addListener(new StopOnFailureListener());

    tester.addCoverageMetric(new ActionCoverage());
    tester.addCoverageMetric(new StateCoverage());
    tester.addCoverageMetric(new TransitionCoverage());
    tester.addCoverageMetric(new TransitionPairCoverage());

    System.out.println("\n_____Generate tests_____");
    tester.generate(2);

    System.out.println("\n_____Coverage report_____");
    tester.printCoverage();

};
```

Figure 2.2: Greedy Tester

After the creation of the test, the next step was to monitor the coverage of the criterias suggested by the teacher:

Radom					
NoSteps	2	16	32	64	2048
Action coverage	2/4	4/4	4/4	4/4	4/4
State Coverage	2/4	3/4	4/3	4/4	4/4
Transition Coverage	2/13	8/13	8/13	13/13	13/13
Edge pair coverage	1/43	12/43	14/43	27/43	39/43

Greedy					
NoSteps	2	16	32	64	2048
Action coverage	2/4	4/4	4/4	4/4	4/4
State Coverage	1/4	4/4	4/4	4/4	4/4
Transition Coverage	2/13	11/13	11/13	13/13	13/13
Edge pair coverage	1/43	14/43	18/43	28/43	39/43

Chapter 3

Task 3

3.1 Generate concrete tests and report errors

The next and final steps was to implement an online test adapter, generate and execute the tests for each test method.

```
public class TheftAlarmTestAdapter {
    Alarm sut = new Alarm();

      public void armSUT(int expected) {
          assertEquals(expected,sut.arm());
      }

      public void disarmSUT(int expected) {
          assertEquals(expected,sut.disarm());
      }

      public void closeDoorsSUT(int expected) {
          assertEquals(expected,sut.closeDoors());
      }

      public void openDoorsSUT(int expected) {
          assertEquals(expected,sut.openDoors());
      }

      public void resetSUT() {
          sut.reset();
      }
```

Figure 3.1: Test Adapter

The first error can be encountered when executing the function disarm from the state DC. It is expected 0 from the output but it obtained a 1.

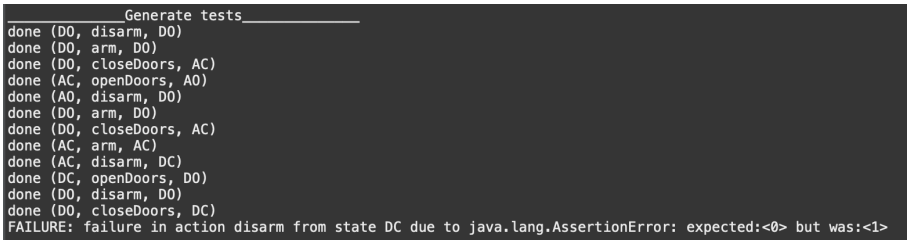

Figure 3.2: Error