# CS2001 W03 Practical Report

## Tutor: Susmit Sarkar

Matriculation ID: 200014517

2021-09-29

## Overview

In this practical I was asked to create a program that:

- Could read in a file to make a Finite State Machine (FSM)
- Run the FSM taking in input and giving the correct output according to the FSM

For the most part I have achieved that.

## Design

For this practical I made 2 java files with 2 classes in them. These were fsminterpreter.java and State.java. The FSM interpreter had the main method that read in and ran the FSM and the other just contained information about each state.

### Fsminterpreter.java

First the imports. Then there's the attributes for the FSM file path, test file path, and array list of state objects that makes up the FSM. Then a Boolean for if a test file has been provided.

Then the main method that takes in arguments from the command line. It first checks if the correct number of arguments were passed. If not, the program displays an error message and quits.

The FSM file is parsed from the arguments. Then the program checks if there are 2 arguments. If not, the Boolean is set to false, if there are 2 however, the program takes in the argument and stores it into the string for test file paths. Then the program calls the read FSM method and then the run FSM method.

The read FSM method returns an array list of state objects and is given the parameter of the name of the FSM file being read. First an array list and scanner object are instantiated to read in the file and store it. Then the programs tries to create a file object and scan it in. Then the program loops through each line of the file.

The loop first splits the current row by spaces and stores it a string array. Then the program checks if the input is of the correct format, if not it displays bad description and quits the program. Otherwise, the program stores the input from the file in separate variables for each of the 4 aspects of the state, the state number, the input, the output, and the transition state number. The program then checks if the state has been instantiated before.

If not, the program increments the index and instantiates a new state object then adds that new state to the FSM. Then it adds the input, output, and transition state to the corresponding state object into the new state. Otherwise, it just adds the input, output, and transition state to the newest state. The program then returns the array list once it has looped through all lines in the file.

Then the run FSM method, takes in an array list of states objects (the FSM) and instantiated a string array for the user input, a scanner to read in the user input, an initial state index to start in, the next state number and the current state. Then if there was a test file to be used the program makes the scanner object scan the file and split the user input by character. It then loops through the input.

The program then gets an integer that will be either -1 for not in the accepted input or it will be the index of the matched input in the current state object. It does this by running the check input method.

Then it instantiates a string of the input being used. If the index is less than 0 then the input was bad, and the program displays an error and quits. Otherwise, the program gets the correct input and displays it. It then gets the next state by running the find next state method.

If there was no test file, then the program should do the same code, but the scanner instead takes in input from the command line.

The check input method takes in the current state and the user input as parameters and returns the index of the matched inputs. The method starts off by getting the accepted inputs of the state and storing them in a string array list. The program then loops through each of the inputs and checks them against the user input. If a match is found the index of the loop is returned.

The find next state method takes in the FSM, the current state, and the number of the next state to be transitioned to. It returns the state object of

the stat being transitioned to. It then loops for each state in the FSM and if it finds the next state then it returns that state.

### State.java

The state class is quite simple it has 4 attributes, the state number then array lists for the accepted inputs and their corresponding outputs and transition states.

There are 2 constructors. One for a basic state object to be instantiated with all attributes and the main one that instantiated the object and the number of the state. Then there's basic getter methods for each attribute. And finally, there is the add to state method which adds a given input, output, and transition state number to their respective array lists.

## Testing

Due to time constraints stacscheck was only used to test the program and it passes most of its tests, seen below in figure 1. It does take input and display it correctly with the FSM rules but it doesn't have enough bad

description catches so it lets through some bad inputs. One of them is caught by the bad input check however so it doesn't break the program.

```
rdn1@klovia:~/Documents/year2/cs2001/practicals/w03Practical $ stacscheck /cs/studres/CS2001/Practicals/W03-FSM/Tests/
Testing CS2001 Week 3 Practical (FSM)
- Looking for submission in a directory called 'src': found in current directory
* BUILD TEST - basic/build : pass
* COMPARISON TEST - basic/Test01_simple/progRun-expected.out : pass
* COMPARISON TEST - basic/Test02_bigger/progRun-expected.out : pass
* COMPARISON TEST - basic/Test03_description/progRun-expected.out : fail
--- expected output ---
Bad description
--- submission output ---
12300
---

* COMPARISON TEST - basic/Test04_missinginput/progRun-expected.out : fail
--- expected output ---
Bad description
--- submission output ---
12311
---

* COMPARISON TEST - basic/Test05_illegal/progRun-expected.out : pass
* COMPARISON TEST - basic/Test06_minimal/progRun-expected.out : pass
* COMPARISON TEST - basic/Test07_alsodescription/progRun-expected.out : fail
--- expected output ---
Bad description
--- submission output ---
Bad input
---

5 out of 8 tests passed
```

*Figure 1: Stacscheck results.*

## Evaluation

I think my submission meets most of the expectations it was supposed too, the only problem is it lets through some bad description files. It does everything else in the specification, so I believe it is perfectly serviceable.

## Conclusion

I honestly found this practical quite tough and if given more time I would test a lot more and fix the current issue with the program. Perhaps if I understood the conceptual parts of finite state machines I would have done better.