CS2001 Project W03 – Finite State Machine

Overview

For this practical we were asked to design and implement an interpreter for Finite State Machines using Java. We are to take a file containing an FSM, the input and produce the appropriate output regarding the Finite State Machine and input provided.

A Finite State Machine is a machine that has a finite set of states. This means it can be interpreted and used to get an output when a valid input is given.

Design

For storing the Finite State Machine data, I decided to use a 2D-String array. I took this decision to utilize the functionality and methods that using a *String* provides compared to a *char*.

I decided to use the 2D-array approach as I believe this was the easiest way to represent the FSM data concisely. The data is easy to access and interpret this way.

When reading in the file containing the Finite State Machine data I decided to put each column (e.g. 'FROM', 'INPUT',…) into its own *ArrayList*. I decided to do this because as the length of the file is undetermined using an *ArrayList* made it easy to interpret a file of an unknown length.

To interpret the Finite State Machine I used my *getRange*() method to find the indexes at which the current states data was. As there are generally multiple inputs given in the transition table I created a method *findLine*() which found the specific index where the current state and the input are. This meant the output and next state were simple to get access to and then the process is repeated for all the inputs.

As the Finite State Machine data is stored in a 2D String and the way I have decided to interpret the data, it means that the data could be in any order and my program would still be able to interpret the data correctly.

Initially I made my program work using the assumption the states will always be in order and using the integer values of the states to interpret the Finite State Machine data I had a working solution. However, this was not an appropriate way to solve the problem and was inefficient. Relying on the Finite State Machine data to be set out a particular way in a certain order meant my program lacked flexibility and was quite rigid. As a result of this I decided to alter the way I interpreted the Finite State Machine, and I made the program that I have now.

Testing

I had some trouble using the stacscheck to test my program. Some of the tests would fail in the stacscheck but would work with the exact same FSM and input when testing myself.

So, for my testing I have added in all the tests that stacscheck has and some of my own tests.

What is being tested?	Method	Folder (all in /TestData)	Expected output	Actual Output	Pass/Fail
Program works for a standard simple FSM and input	interpretFSM()	Test1_simple	'eoo'	'eoo'	Pass
Program still works for a bigger FSM which has more states	interpretFSM()	Test2_bigger	'12300'	'12300'	Pass
Program recognizes that the FSM is malformed	malformedFSM()	Test3_description	'Bad description'	'12300'	Fail
Program recognizes that there is a missing input from FSM	malformedFSM()	Test4_missingInput	'Bad description'	'Bad description'	Pass
Program recognizes that the input from user is illegal	inputValidation()	Test5_illegal	'Bad input'	'Bad input'	Pass
Program is able to still correctly handle a very small and simple FSM	interpretFSM()	Test6_minimal	'10101010 111000101 010111000'	'10101010 111000101 010111000'	Pass

Program is able to detect that the FSM provided is malformed	malformedFSM()	Test7_alsoDescription	'Bad description'	'Bad description'	Pass
Program is able to handle when no file name is provided	fileValidation()	n/a	'No file provided'	'No file provided'	Pass
Program is able to handle no input being provided	main()	n/a	'Please enter an input or exit using 'exit' '	'Please enter an input or exit using 'exit' '	Pass

Evaluation

As shown from my tests above. My program can interpret the Finite State Machine data and be given an input and get the correct output. However, my program is sometimes unable to detect whether the Finite State Machine is malformed. From the tests shown above I believe my program has succeeded in what the aim of this project was, and I believe it has met the requirements given in the specification.

Given more time I would have liked to make sure my program 100% of the time is able to detect a malformed FSM.