CS 311-01

Formal Language and Automata

Project# 1

Emily Le

Daniel Choi

11/2/15

Project Description

For this project, we were tasked with implementing a universal finite automata that would take in a finite automaton’s number of states, the final states, alphabet, transitions and the strings that will be tested. Our program would take in this data and perform a transition that would determine whether or not the string is accepted or rejected by the rules that were provided.

In order to accomplish this, we have decided to use an adjacency matrix that will be populated by the data provided. We have decided to use the indexes of the matrix as our means of dynamically populating and accessing the data within the matrix. The first array value would stand for the current state, while the second array value would stand for transition symbol that is being inputted in. The value within the array would stand for the next states that would be obtained from the transition symbol. We would then update the current state with the new state by shifting over the value within the right matrix to the left.

Our finite automata takes in data specifically in order to allow for proper assessment of the inputs. The data is accepted consecutively within this order: number of states, final states, alphabet, transitions and input to be used for checking. In order to denote each finite automata language pattern we have divided them using a space.

Our biggest challenge definitely lied within proper data parsing and assessment and conditional checking for whether or not the input would be either accepted or rejected. We accomplished this by parsing our data based on patterns that consistently occurred such as parentheses and spaces. In addition, to combat the problem of the conditionals, we used many symbols and translations in order to properly represent and transition within the states.

In conclusion, our program creates a definite finite automata through the use of an adjacency matrix and it’s indexes. The indexes signify the transitions that occur within a finite automata. Unfortunately, our program is very dependent on the strict compliance of the input. Our program will read in one file of text data and proceed accordingly.

Instructions

**Clarification: In the files that we have sent, the language.txt file is already configured for problems 1-4. If there is no need to add and test additional custom data, then just compile and run the program.**

**Data File:**

1. The name of the data file should be named “language.txt”.
2. The data file should be in the same directory as the program.
3. The data file will contain all finite automata separated by space.
4. The data shall be inputted in the order of Number of States, Final States, Alphabet, Transitions and Strings used for input.
5. Each of these will be put on its own individual line.
6. Each one will be separated by just one space between each other.
7. Transitions will be signified on each individual line using parentheses. E.g (0 0 0), the first number will symbolize the current state, while the middle will signify the transition, and the rightmost number will signify the state after the transition.
8. Finally, after inputting all proper strings, create a new line and repeat steps 1-8 for your new finite automata rules.

**Legend:**

Empty strings are denoted by using just one space.

**Z = 0**, when it needs to be differentiated from the other numbers; The transition shall go to 0. Thus, (1 0 1). This means that state one took in a Z and will go to state one.

**P = 1-9**; Transition will be towards 1.

**N = 0-9**; Transition will be towards 1.

**L = A-Z, a-z**; Transition will be towards 0

Use the symbols within the alphabet in the text file. E.g. Z P.

Symbols other than empty string, letters and numbers haven’t been accounted for.

**Program:**

1. Make sure that the “language.txt” is within the same directory as the Main.java.
2. Compile the program using either an ide or terminal: javac Main.java
3. Make sure that jdk is installed.
4. Run the program: java Main
5. If the data.txt file is properly created, then you should receive output that follows this format.

**Finite State Automaton #1.**

**1) number of states: 2**

**2) final states: 1**

**3) alphabet: 0, 1**

**4) transitions:**

**0 0 0**

**0 1 1**

**1 0 0**

**1 1 1**

**5) strings:**

**1000 Reject**

**10001 Accept**