NFA TO DFA CONVERSION

PROJECT 1 REPORT

OZLEM KUTLUEL

Introduction

This report contains the overview of dfaTranslator.java, and all accompanying files. The aim of the code is to transform non-deterministic finite automata (hereby referred to as NFA) into deterministic finite automata (hereby referred to as DFA). It reads a NFA input from a file in a pre-determined format, transforms it, and prints the resulting DFA to the terminal in the same format.

The application was written in Java and is run in terminal. The format of the given NFA is shown in the example below, and the transition states are read as {Initial state, transition alphabet, resulting state}. The specifications of how it is read is detailed later in the report.

```
ALPHABET
0
1
STATES
A
B
C
START
A
FINAL
C
TRANSITIONS
A 0 A
A 1 A
A 1 B
B 1 C
END
```

Data Types

Array Lists: The program uses String Array Lists, as well as an Array List of String Array Lists to conduct all storing and calculating of necessary information. Examples of the Array Lists, as well as the declarations, are given below.

```
//Creates the ArrayLists to store the values
List<String> alphabet = new ArrayList<>();
List<String> states = new ArrayList<>();
List<String> start = new ArrayList<>();
List<String> accept = new ArrayList<>();
List<String> transition = new ArrayList<>();

//Creates the array I will be working with, with as many arrays as there as
states
ArrayList<ArrayList<String>> myArray = new ArrayList<>(states.size());

//Example of a states array
{A, B, C}

//Example of a transitions array
[A 0 A, A 1 B, A 1 C, B 0 B, B 0 C, C 0 A, C 0 B, C 1 B]

//Example of a 'myArray' containing the transition table
[[A, BC], [BC, -5], [AB, B], [ABC, B], [ABC, BC], [ABC, BC]]
```

Algorithm

The algorithm begins by declaring the necessary Array Lists, then reads the input while placing the information into the appropriate Array Lists. It then places the information into a transition table (myArray), using the provided transitions. It works through the table and creates new states as necessary, and finally uses the table to print the final state of the finite automata, which is now deterministic. The specifications of each step are detailed below;

Reading the file

The scanner class is utilized to read in the necessary file. The file path is declared separately, and the file is read in line by line.

```
//Stores the location of the file
File fis = new File("./src/NFA2.txt");

//Creates a scanner to scan in the file
Scanner sc = new Scanner(fis);
```

As each line is read, a switch case determines what information is being read, while another places that information in the appropriate Array List. A while loop assures that every line in the file is read.

```
//Loops through until no lines are left to read
while(sc.hasNext())
{
    temp = sc.nextLine();
    //Checks what part of the information is being read in
    switch (temp){
        case "ALPHABET":
            temp = sc.nextLine();
            break;
        case "STATES":
            tracker = 1;
            temp = sc.nextLine();
            break;
        case "START":
            tracker = 2;
            temp = sc.nextLine();
            break;
        case "FINAL":
            tracker = 3;
            temp = sc.nextLine();
            break;
        case "TRANSITIONS":
            tracker = 4;
            temp = sc.nextLine();
            break;
        default:
            break;
    }
    //Adds the information to the appropriate ArrayList
    switch (tracker){
        case 0:
            alphabet.add(temp);
            break;
        case 1:
            states.add(temp);
            break;
        case 2:
            start.add(temp);
            break;
        case 3:
            accept.add(temp);
            break;
        case 4:
            transition.add(temp);
            break;
    }
}
```

Filling myArray

This Array List of String Array Lists is filled initially with -5 values, to represent empty spots. It is created as an Array List of X Array Lists, each with Y number of elements, where X is the number of States, and Y is the number of elements in the Language. It is then filled using the transition information, with the index of each alphabet letter and each state as a reference to location.

```
//Initializes the array with "-5" default value, as many as there are
alphabet characters
for(int i=0; i < states.size(); i++) {
    myArray.add(new ArrayList());
    for (int j = 0; j < alphabet.size(); j++){
        myArray.get(i).add("-5");
    }
}

//Reads the transitions and places the appropriate information into the array
for (int i = 0; i<transition.size(); i++){
    char source = transition.get(i).charAt(0);
    int index = states.indexOf(String.valueOf(source));
    source = transition.get(i).charAt(2);
    int index1 = alphabet.indexOf(String.valueOf(source));
    if (myArray.get(index).get(index1) != "-5" &&
!myArray.get(index).get(index1).contains(String.valueOf(transition.get(i).cha
rAt(4)))){
        myArray.get(index).set(index1,
myArray.get(index).get(index1).concat(String.valueOf(transition.get(i).charAt
(4))));

    }
    else if
(!myArray.get(index).get(index1).contains(String.valueOf(transition.get(i).ch
arAt(4)))){
        myArray.get(index).set(index1,
String.valueOf(transition.get(i).charAt(4)) );
    }
}
```

Adding a new state for each one encountered

Each resulting spot in the array is checked, alphabetized, and recurring letter deleted. For example, ACBA becomes ABC. For each new string that is encountered, it is placed in the states Array List, and a new String Array List is added to myArray in the corresponding index. It is initially filled with '-5' default value, and is then filled with the appropriate results using the already existing information in myArray.

```
//Count to keep track of how many new states there are
int newCount = 0;
//Adds a new state to the ArrayList for each new state encountered
for (int i = 0; i < myArray.size(); i++){
    for (int j = 0; j < myArray.get(i).size(); j++){
        if(myArray.get(i).get(j) != "-5"){
```

```
            String outputString = sortString(myArray.get(i).get(j));
            myArray.get(i).set(j, outputString);
        }
        if (!states.contains(myArray.get(i).get(j)) && myArray.get(i).get(j)
!= "-5"){
            states.add(myArray.get(i).get(j));
            newCount++;
        }
    }
}

//Loops through each new state and creates a new ArrayList in myArray, finds
and adds the appropriate information
for (int i = states.size()-newCount; i < states.size(); i++){
    myArray.add(new ArrayList());
    for (int j = 0; j < alphabet.size(); j++){
        myArray.get(i).add("-5");
        for (int f = 0; f < states.get(i).length(); f++){
            char source = states.get(i).charAt(f);
            int index = states.indexOf(String.valueOf(source));
            String temp1 = myArray.get(index).get(j);
            if (myArray.get(i).get(j) != "-5" && temp1 != "-5" &&
!myArray.get(i).get(j).contains(temp1)){
                myArray.get(i).set(j, myArray.get(i).get(j).concat(temp1));
            }
            else if (temp1 != "-5" &&
!myArray.get(i).get(j).contains(temp1)){
                myArray.get(i).set(j, temp1);
            }
        }
    }
}
```

Finding accept states

Each state is checked to see if any of them contain the accept state as specified by the initial file. If they do, it is added to the accept states Array List.

```
for (int i = 0; i < states.size(); i++){
    for (int f = 0; f < accept.size(); f++) {
        if (states.get(i).contains(accept.get(f)) &&
!accept.contains(states.get(i))){
            accept.add(states.get(i));
        }
    }
}
```

Printing

The print section takes all the information that has been amassed and prints it to the console in the required format. The following contains both the implementation and an example output.

```
System.out.println("ALPHABET");
for (int i = 0; i < alphabet.size(); i++){
    System.out.println(alphabet.get(i));
}

System.out.println("STATES");
for (int i = 0; i < states.size(); i++){
    System.out.println(states.get(i));
}

System.out.println("START");
for (int i = 0; i < start.size(); i++){
    System.out.println(start.get(i));
}

System.out.println("FINAL");
for (int i = 0; i < accept.size(); i++){
    System.out.println(accept.get(i));
}

System.out.println("TRANSITIONS");

for (int i = 0; i < states.size(); i++){
    for (int j = 0; j < alphabet.size(); j++){
        if (myArray.get(i).get(j) != "-5"){
            System.out.print(states.get(i));
            System.out.print(" ");
            System.out.print(alphabet.get(j));
            System.out.print(" ");
            System.out.print(myArray.get(i).get(j));
            System.out.println("");
        }
    }
}
System.out.println("END");


//Example output
ALPHABET
0
1
STATES
A
B
C
BC
AB
ABC
START
A
FINAL
C
```

```
BC
ABC
TRANSITIONS
A 0 A
A 1 BC
B 0 BC
C 0 AB
C 1 B
BC 0 ABC
BC 1 B
AB 0 ABC
AB 1 BC
ABC 0 ABC
ABC 1 BC
END
```