CS 410 – Project 3 Report

Turing Machine Simulation

By: Esrah Zahid (S020289)

Table of Contents

I. Introduction

A Turing machine is a theoretical machine that can be used to perform computations. It consists of an infinite tape that is divided into cells, a read/write head that can move left or right across the tape, and a set of states that the machine can be in. The machine can read and write symbols on the tape and move the head left or right based on the symbol it reads and the current state it is in. The behavior of the Turing machine is determined by a set of transitions that specify the actions to be taken based on the current state and the symbol read from the tape.

In this report, I will present a Java program that can simulate a Turing machine. Given an input file that specifies the input alphabet, tape alphabet, states, transitions, start state, accept state, reject state, and an input string, the program can simulate the behavior of the Turing machine and determine if the input string is accepted, rejected, or looped. It will also display the route taken by the simulation.

II. Input File

The input string I will be using should have the following format:

```
<number of variables in input alphabet>
<the input alphabet>
<number of variables in tape alphabet>
<the tape alphabet>
<blank symbol>
<number of states>
<states>
<start state>
<accept state>
<reject state>
<transition1>
<transition2>
<transitionN>
<string to be detected>
```

input format for .txt file

```
1
0
2
0 X
b
7
q1 q2 q3 q4 q5 qA qR
q1
qA
qR
q1 0 b R q2
q1 b b R qR
q1 X X R qR
q2 0 X R q3
q2 X X R q2
q2 b b R qA
q3 X X R q3
q3 0 0 R q4
q3 b b L q5
q4 X X R q4
q4 0 X R q3
q4 b b R qR
q5 0 0 L q5
q5 X X L q5
q5 b b R q2
000
```

An example of input.txt I will be using in this report

III. Method

To simulate the Turing machine, in my program code I will be implementing the following steps:

First step is to read the input file and parse its contents into variables that will be stored. This includes extracting the number of variables in the input and tape alphabets, the blank symbol, the number of states, the states themselves, the start, accept, and reject states, and the list of transitions.

Next, the Turing machine is initialized by creating a list to represent the tape and setting the tape head and current state to the start state. A set is also initialized to keep track of the states that have been visited.

The main loop of the simulation then begins. Within the loop, the current state is checked to see if it has been visited before. If it has, the simulation is terminated as the machine is deemed to be in a loop. If the current state has not been visited before, it is added to the set of visited states and the list of states visited in the route.

The program then searches for a transition that matches the current state and the symbol under the cursor. If a matching transition is found, the symbol on the tape is updated according to the transition and the cursor is moved left or right as specified. The current state is then updated to the next state specified in the transition. If no matching transition is found, the simulation is terminated.

After the simulation has terminated, the route and result are printed to the console. If the final state is the accept state, the result is "accepted". If the final state is the reject state, the result is "rejected". Otherwise, the result is "looped".

Example of the result that will be printed to the console for the input file specified above:

```
ROUTE: q1 q2 q3 q4 qR
RESULT: rejected

Process finished with exit code 0
```

IV. Implementation

The Java program I wrote consists of a single class called TuringMachineSimulator with a main method that is responsible for reading the input file, parsing the relevant information, and simulating the behavior of the Turing machine.

To read input.txt, I will be making use of BufferedReader

```
BufferedReader reader = new BufferedReader(new FileReader( fileName: "input.txt"));
List<String> lines = new ArrayList<>();
String line;
while ((line = reader.readLine()) != null) {
    lines.add(line);
}
```

and then parsing through all the lines to get the number of variables in the input and tape alphabets, the blank symbol, the number of states, the states, the start, accept, and reject states, and the list of transitions

The transitions will be stored in a list of lists

```
List<List<String>> transitions = new ArrayList<>();
for (int i = 10; i < lines.size() - 1; i++) {
    String[] transition = lines.get(i).split( regex: " ");
    transitions.add(Arrays.asList(transition));
}
String string = lines.get(lines.size() - 1);
```

The tape will be simulated using a List

```
// Initialize Turing machine
List<Character> tape = new ArrayList<>();
for (char c : string.toCharArray()) {
    tape.add(c);
}
tape.add(blankSymbol);
int tapeHeadIndex = 0;
```

to which I will be adding more items using .add as I move through each transition.

I try to move the tape head using the following method:

```
if (transition.get(3).equals("L")) {
    tapeHeadIndex--;
    if (tapeHeadIndex < 0) {
        tape.add( index: 0, blankSymbol);
        tapeHeadIndex = 0;
    }
} else {
    tapeHeadIndex++;
    if (tapeHeadIndex >= tape.size()) {
        tape.add(blankSymbol);
    }
}
```

I will be making use of a while loop to simulate the functioning of the Turing Machine. The while loop runs until the state of the Turing machine is either the accept state, the reject state, or a state that has already been visited (looped). At each iteration of the loop, the Turing machine checks for a transition that matches the current state and character at the tape head of the tape. If a matching transition is found, the character at the tape head is replaced with the character specified in the transition, the cursor is moved left or right as specified in the transition, and the state is updated to the state specified in the transition. If a matching transition is not found, the loop is exited. After the loop has exited, the route taken by the Turing machine and the result (accepted, rejected, or looped) are printed to the console by the following way:

```
// Printing the output to console
System.out.print("ROUTE: ");
for (String s : route) {
    System.out.print(s + " ");
}
System.out.println();

if (state.equals(acceptState)) {
    System.out.println("RESULT: accepted");
} else if (state.equals(rejectState)) {
    System.out.println("RESULT: rejected");
} else {
    System.out.println("RESULT: looped");
}
```

V. Result

If I run the code on the input.txt file for the input string: 000, the result that will be printed to the console is:

```
ROUTE: q1 q2 q3 q4 qR
RESULT: rejected

Process finished with exit code 0
```

This is because, when the Turing machine starts at the state q1, it will move to the state q2 based on the transition q1 0 b R q2, then moves to the state q3 based on the transition q2 0 X R q3, then moves to the state q4 based on the transition q3 0 0 R q4, and finally moves to the

reject state qR based on the transition q4 b b R qR.
Finally, The input string 000 is rejected because the Turing machine ends in the reject state.