ÖZYEĞİN UNIVERSITY

Automaton Theory and Formal Languages Fall 2022

CS 410 Project 3

Design Report

By

Yasemin Orhun

S018151

Abstract:

In this project I have designed a Turing Machine simulation. The Turing Machine simulation tells if a string will be accepted, rejected, or looped in each input file which contains the alphabet and the transition functions. In an input file, the number of variables of the input and the tape alphabet, the input and tape alphabet, the blank symbol, number of states, states, start state, accept state, reject state, the transitions, and the string to be detected should be given. My project first constructs an alphabet after reading the input file, creates a tape, simulates the Turing Machine, and writes the results into console.

Tools and Software Requirements:

For the Project I have used Java as my language.

Method:

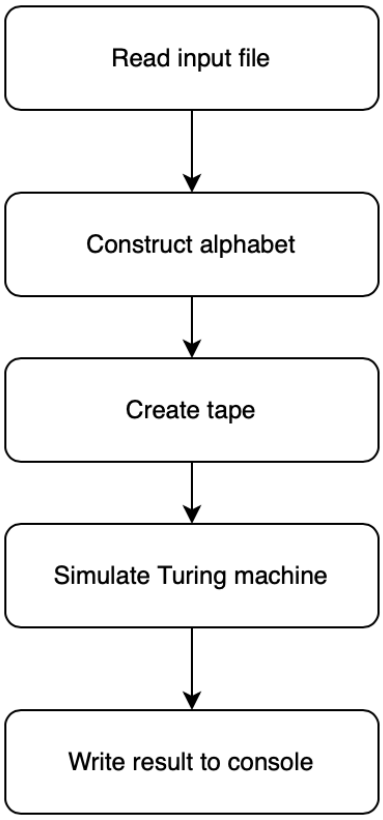


Figure 1.

1. The program read an input.txt file as input. To store the alphabet, an alphabet object is created, while reading the input, the information of the input file is stored in an Algorithm object.
2. The program creates a tape in List<String> format with the given string to be detected and the blank symbol. The first indexes are stored with the string to be detected information; the remaining indexes are set with the blank symbol.
3. After the tape and alphabet are created the program runs Turing Machine Algorithm. The algorithm checks all the transitions of the alphabet until the state reaches accept or reject or it loops. If the symbol to read on the transition is equal to tape head index and the transition's first state is equal to the current state it moves the tape index. If the direction is "L" it moves 1 step to left, if the direction is "R" it moves one step to right (the tape head index is incremented and decremented if it does not equal to 0). When the state reaches accept or reject the loops tops and the result is written to console. If the loop reaches a certain limit and does not reach accept or reject state, then it stops and says the string looped. For the loop to not get any error I put a limit number which is so much bigger than the given transitions amount. If the state does not get accept and reject while looking at all the transition it loops.

$\boldsymbol{simulateTuringMachineAlgorithm}(\boldsymbol{Alphabet}, \boldsymbol{tape})$:

1: $while$ ($state\ does\ not\ accept\ or\ reject$):
2: $for\ each\ transition\ in\ Alphabeth$:
3: $if\ (symbolToRead == tape.get(headIndex)\ and\ firstState == currentState)$
4: $headIndexOfTape = symbolToWrite$
5: $moveTape(headIndex)$
6: $route = route + currentState$
7: end
8: end
9: $return\ route$

4. After the route information is collected if the route is accepted the program writes to the console: "accepted", if rejected "rejected and if none of the two option it writes "looped" with the route information (the states visited).

Implementation:

For the implementation I created an input file.

Input File:

2 (number of variables in input alphabet)
0 1 (the input alphabet)
2 (number of variables in tape alphabet)
0 1 (the tape alphabet)
b (blank symbol)
13 (number of states)
q1 q2 q3 q4 q5 q6 q7 q8 q9 q10 q11 qA qR (states)
q1 (start state)
qA (accept state)
qR (reject state)
q1 0 b R q2 (the transitions)
q1 1 b L q3
q2 0 1 R q4
q2 1 0 R q5
q3 0 0 L q6
q3 1 1 L q7
q4 0 1 L q8
q4 1 0 L q9
q5 0 0 R q10
q5 1 1 R q11
q6 0 0 L q6
q6 1 1 L q7
q7 0 1 R q8
q7 1 0 L q9
q8 0 1 R q8
q8 1 0 L q9
q9 0 0 L q10
q9 1 1 L q11
q10 0 0 L q10
q10 1 1 R q11
q11 0 1 L qA
q11 1 0 R qR
STRING
0110011011011011011 (the string to be detected)

Explanation of Methods:

To store the alphabet, I created an Alphabet class which stores all the information given from the input file. I used List<String> for storing the transitions.In the Main class I have six methods:

1. ***readAlphabethFromInputFile(Alphabeth):*** reads the alphabet from the input.txt file and sets the variables.

2. ***createTape(Alphabeth):*** creates a tape in List<String> format where the first strings are the string given symbols and 50 more blank symbols are added in representation of infinite number of blank symbols.

```
private static List<String> createTape(Alphabet alphabet) {
    String[] stringDetected = alphabet.getStringToBeDetected().split( regex: "");
    String[] tapeArray = new String[50 + stringDetected.length];
    for (int i = 0; i < 50 + stringDetected.length; i++) {
        if (i < stringDetected.length) {
            tapeArray[i] = stringDetected[i];
        } else {
            tapeArray[i] = alphabet.getBlankSymbol();
        }
    }
    List<String> tape = Arrays.asList(tapeArray);
    return tape;
}
```

3. ***simulateTuringMachine (Alphabeth,tape):*** simulates an algorithm of Turing Machine according to specific rules.

```
public static String simulateTuringMachine(Alphabet alphabet, List<String> tape) {
    String state = alphabet.getStartState();
    String route = "";

    int loopLimit = 100000;
    int tapeHeadIndex = 0;
    int currentLoopIndex = 0;

    for (currentLoopIndex = 0; currentLoopIndex < loopLimit; currentLoopIndex++) {
        if (!state.equals(alphabet.getAcceptState()) && !state.equals(alphabet.getRejectState())) {
            for (String currentTransition : alphabet.getGoalStates()) {
                String[] transitionInfo = currentTransition.split( regex: " ");

                String firstState = transitionInfo[0];
                String symbolToRead = transitionInfo[1];
                String symbolToWrite = transitionInfo[2];
                String direction = transitionInfo[3];
                String secondState = transitionInfo[4];
                String tapeNext = tape.get(tapeHeadIndex);

                if (symbolToRead.equals(tapeNext) && firstState.equals(state)) {
                    tape.set(tapeHeadIndex, symbolToWrite);
                    tapeHeadIndex = moveTapeIndex(tapeHeadIndex, direction);
                    state = secondState;
                    route = route.concat( str: state + " ");
                }
            }
        }
    }
    return state + "/" + route;
}
```

4. ***moveTapeIndex(int, String):*** moves the index of the tape according to the direction of the transition.

```java
private static int moveTapeIndex(int tapeHeadIndex, String direction) {
    if (direction.equals("R")) {
        tapeHeadIndex++;
    } else if (direction.equals("L") && tapeHeadIndex != 0) {
        tapeHeadIndex--;
    }
    return tapeHeadIndex;
}
```

5. ***writeResultsToConsole(Alphabet, String):*** writes the results of the route to the console.

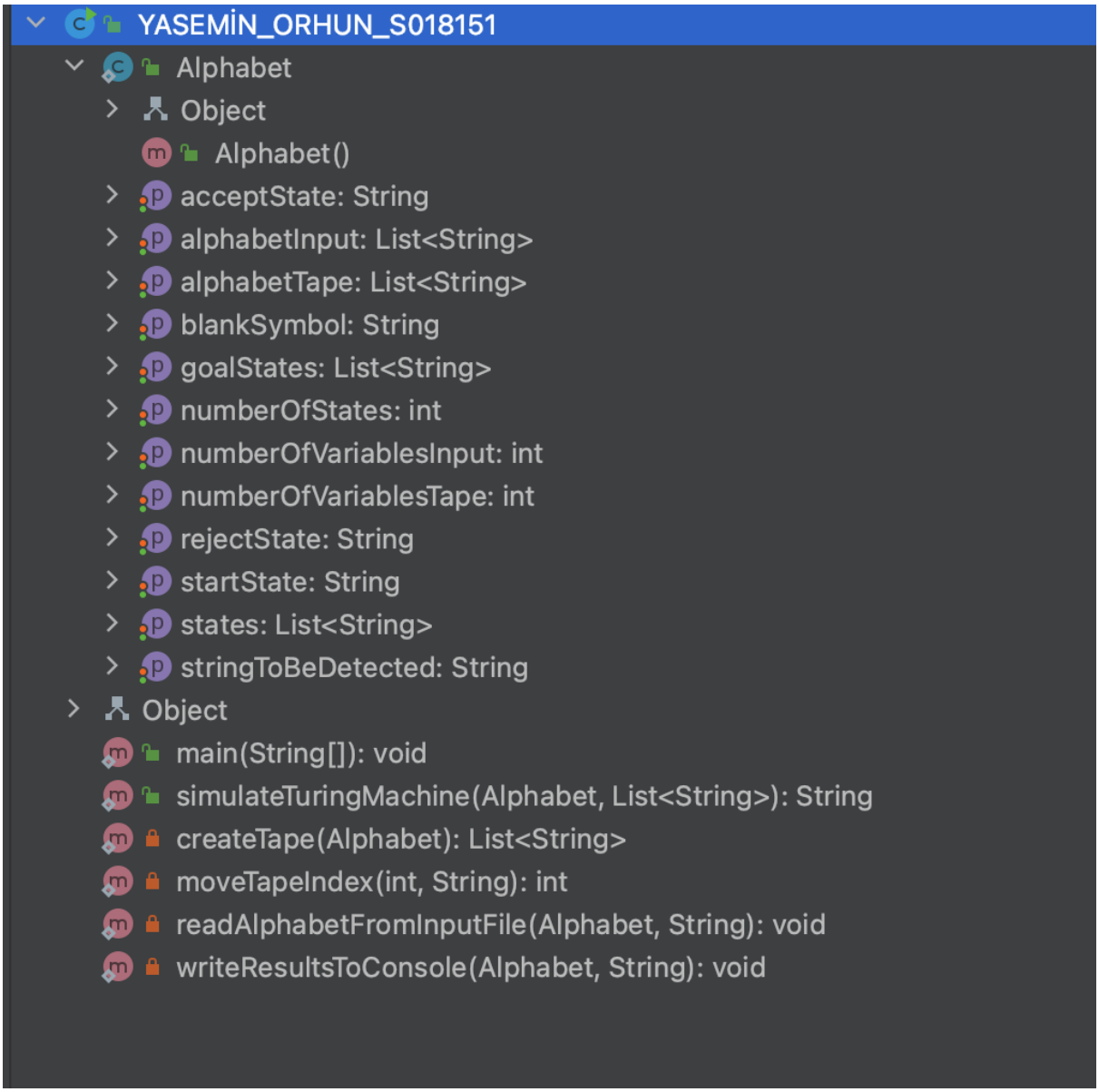
```java
1 usage
private static void writeResultsToConsole(Alphabet alphabet, String resultset) {
    String[] results = resultset.split( regex: "/");
    String state = results[0];
    String result = results[1];

    if (state.equals(alphabet.getAcceptState())) {
        System.out.println("ROUTE: " + alphabet.getStartState() + " " + result);
        System.out.println("RESULT: " + "accepted");
    } else if (state.equals(alphabet.getRejectState())) {
        System.out.println("ROUTE: " + alphabet.getStartState() + " " + result);
        System.out.println("RESULT: " + "rejected");
    } else {
        System.out.println("ROUTE: " + result);
        System.out.println("RESULT: " + "looped");
    }
}
```

6. ***main(String[][]):*** in the main method the methods are run.

```java
public class YASEMİN_ORHUN_S018151 {
    public static void main(String[] args) {
        try {
            Alphabet alphabet = new Alphabet();
            readAlphabetFromInputFile(alphabet, path: "Input_YASEMİN_ORHUN_S018151.txt");
            List<String> tape = createTape(alphabet);
            String result = simulateTuringMachine(alphabet, tape);
            writeResultsToConsole(alphabet,result);

        } catch (Exception e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
```


YASEMİN_ORHUN_S018151
Alphabet
Object
Alphabet()
acceptState: String
alphabetInput: List<String>
alphabetTape: List<String>
blankSymbol: String
goalStates: List<String>
numberOfStates: int
numberOfVariablesInput: int
numberOfVariablesTape: int
rejectState: String
startState: String
states: List<String>
stringToBeDetected: String
Object
main(String[]): void
simulateTuringMachine(Alphabet, List<String>): String
createTape(Alphabet): List<String>
moveTapeIndex(int, String): int
readAlphabetFromInputFile(Alphabet, String): void
writeResultsToConsole(Alphabet, String): void

Results and Conclusion:

1. For the **accept** scenario I tested with string "0110011011011011011"

String: 0110011011011011011

Result: ROUTE: q1 q2 q5 q11 qA
RESULT: accepted

2. For the **loop** scenario I tested with string "01"

String: 01

Result: ROUTE: q2 q5
RESULT: looped

3. For the **reject** scenario I teste with string "01111111"

String: 01111111

Result: ROUTE: q1 q2 q5 q11 qR
RESULT: rejected

As a conclusion my Turing Machine simulation can work as a Turing Machine with a given alphabet, transitions, and a string to be detected. It can give three possible results accept, reject or loop. It searches the possible route among the transitions given through a tape. If the tape reaches accept or reject it ends search, if it loops until some certain limit is reaches it states the string will be loop.