CSC615M MP3: Two-Way Tape Automata

Ryan Austin Fernandez
CS-ST Student
2401 Taft Avenue,
Manila, Philippines 1004
(+63)917-621-6469
ryan_fernandez@dlsu.edu.ph

ABSTRACT

Formal Machines can be generalized as a tape automaton that has an input tape, output tape, and storage tape. Manipulating the presence and form of access to these tapes changes the power of the machine. A simple variety of the Tape Automata is a Two-Way Accepter, which is a Tape Automaton with an input tape, but no storage or output tape. The input tape can be scanned freely, meaning the head of the machine can go left and right. The goal of this project is to implement a Two-Way Accepter as a step towards implementing more complex tape automata, such as a complex Turing Machine. The model was designed to represent the states and the machine as a collection of states. Each state stores all their program instructions. A few sample machines were tested and they produced correct results. In the end, the project was successfully implemented and can be used partially in implementing more complex machines.

I. INTRODUCTION

Finite State Automata is a simplified representation of formal machines. To generalize formal machines, the input, storage, and output can be generalized as tapes, which can be used by the finite state control. The power of these machines to recognize languages or in producing output is now affected depending on the presence of the storage tape and how the storage tape is accessed i.e. scanning the storage tape freely or being restricted to the last symbol read like a stack.

A simple variety of tape automata is the two-way accepter. This machine has no storage tape and no output tape. The input tape can be accessed freely, meaning the head can be freely moved left and right. This machine has no more power than finite state automata (Denning, P.J, Dennis, J.B., & Qualitz, J.E., 1978).

This project's aim is to simulate the operation of a two-way accepter. This project is limited to deterministic two way accepters. This is a step towards moving to more powerful types of machines such as the Turing Machine.

II. DESIGN

A two-way accepter is defined by

$$M = (Q,T,P,q_I,F)$$

Where
Q = finite set of states
T = finite set of tape symbols
P = program containing finite set of instructions of the form
q] scanleft(t,q') or q] scanright(t,q')
q_I = initial state
F = finite set of final states, $F \subseteq Q$

The input is bounded by two "#" symbols on each side. The machine either accepts the string, by halting in a final state, or rejects the string, by entering an infinite loop or halting in a non-final state.

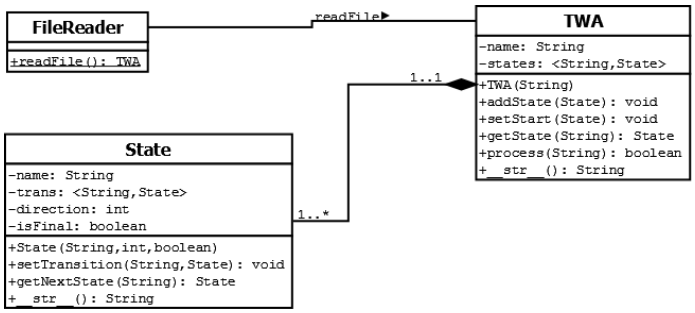

Figure 2.1. – UML Diagram for the TWA Project

The system has 3 main classes, as shown in Figure 2.1. The FileReader class reads a file and returns a TWA object. The file format is simply

<state 1>,<instruction 1>,<symbol 1>,<new state 1>
<state 2>,<instruction 2>,<symbol 2>,<new state 2>
•••
<state n>,<instruction n>,<symbol n>,<new state n>

Q, T, and P are inferred from the input. qI is denoted as "start". Trap states are denoted as "halt". Accept states are denoted as "accept". A restriction is put on P such that a single state can only have one type of instruction: either scanleft or scanright.

The other two classes are TWA and State. All elements of Q are represented as State objects. P is stored in each State object by mapping input symbols to states. The collection of all State objects defines the TWA (Denning, P.J, et al., 1978).

In running the program, the head's position is denoted by an index. When the head scans right, the index is incremented. When the head scans left, the index is decremented. The pseudocode for running the machine is given in Table 2.1.

Table 2.1. – Pseudocode for running a TWA

```
run(machine, input) returns TRUE if string is accepted, FALSE
otherwise
    index = 0
    halted = FALSE
    while not halted
        curr = machine.state
        if curr.direction is LEFT
            if index == 0 //if at leftmost part of tape
                input = "#" + input
```

```
            index = 1
        if input[index – 1] has a transition in curr
            machine.state = curr.nextState(input[index – 1])
            index--
        else //go to trap state
            halted = TRUE
    else //if direction is RIGHT
        if index == input.length – 1 //if at rightmost part of tape
            input = input + "#"
        if input[index + 1] has a transition in curr
            machine.state = curr.nextState(input[index + 1])
            index++
        else //go to trap state
            halted = TRUE
return machine.state.isFinal
```

III. IMPLEMENTATION

The system was implemented in Python. The project had four modules: TWA.py for modelling the actual machine, State.py to model the states, FileReader.py to read the file, and Main3.py to act as the driver.

The State class used a Python dictionary to store the mappings from input symbols to states. The TWA class used a dictionary to map the names of the states to the states themselves. Each state, when added to the TWA, has the TWA's name prepended to the state name. The TWA adapts the name of the file it originated from.

IV. TESTING

Testing was done mainly on two machines or variations of machines discussed in class: the machine that accepts a string in (0 U 1)* such that the number of 0's is odd and the number of 1's is divisible by 3.

This machine is illustrated in Figure 4.1.

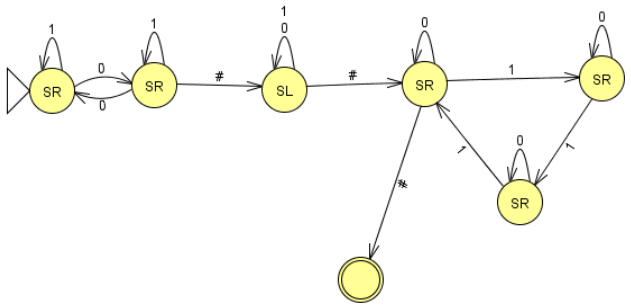

Figure 4.1. – State Diagram for the Machine that Accepts an Odd Number of 0's and a Number of 1's Divisible by 3

Strings used in testing this include:

001011, 00111,00101, 001

The expected outputs are:

Accept, Reject, Reject, Reject

The program's outputs are shown in Figure 4.2.

```
Enter filename of Two-Way Accepter (Leave blank to exit): odd0three1.txt
Enter string (Leave blank to exit): 001011
String was accepted!
Enter string (Leave blank to exit): 00111
String was rejected!
Enter string (Leave blank to exit): 00101
String was rejected!
Enter string (Leave blank to exit): 001
String was rejected!
Enter string (Leave blank to exit):
```

Figure 4.2. – Actual Output for the Test Cases for the sample machine

Which matches the expected output for the test cases.

The next machine to use in testing is the 234 machine, which accepts strings in (a U b U c)* that have less than or equal to two a's, and each a is followed by 3 b's and 4 c's in any order. The state diagram is shown in Figure 4.3.

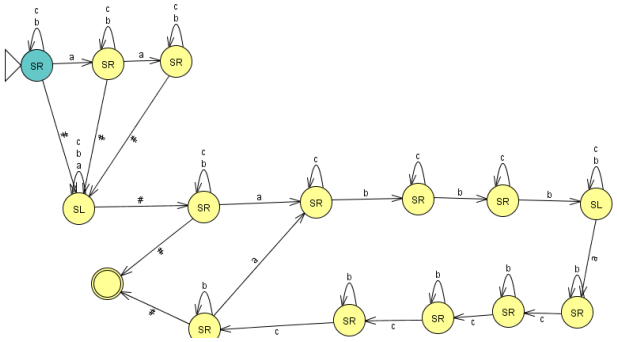

Figure 4.3. – State Diagram for the 234 Machine

The test cases for this include:

bcbbcbcbbc, bcbcacbcbcbc, acbcbcbcacbcbcbcacbcbcbc, acbcbcbcacbcbcb, acbcbcbcacbcbcbcacb

The expected output is

Accept, Accept, Reject, Reject, Reject

The actual output is shown in Figure 4.4.

```
Enter filename of Two-Way Accepter (Leave blank to exit): 234.txt
Enter string (Leave blank to exit): bcbbcbcbbc
String was accepted!
Enter string (Leave blank to exit): bcbcacbcbcbc
String was accepted!
Enter string (Leave blank to exit): acbcbcbcacbcbcbcacbcbcbc
String was rejected!
Enter string (Leave blank to exit): acbcbcbcacbcbcb
String was rejected!
Enter string (Leave blank to exit): acbcbcbcacbcbcbcacb
String was rejected!
```

Figure 4.4. – Actual Results for the Test Cases for the 234 Machine

These two machines deal with a large scope of the functionality of a TWA. Since the project passes the test cases, it could be said that it can successfully simulate a Two-Way Accepter.

V. CONCLUSION

In conclusion, the project was successfully implemented. The program can successfully simulate any given deterministic two-way accepter. It can be used as a basis for implementing combinations of Turing Machines.

APPENDIX A – SELF-ASSESSMENT RUBRIC

Criterion	Grade
I/O Modules	29
Core Process	30
Quality of Testing	18
Documentation	18
TOTAL	95

REFERENCES

[1] Denning, P.J., Dennis, J.B., & Qualitz, J.E. (1978). *Machines, languages, and computation*. New Jersey: Prentice Hall.