CS410 Project 3
Design Report

İsmail Can Yağmur

December 2022

Introduction

This report discuss the simulation of Turing Machine written in C++17 with g++ compiler. Turing Machines written in a specific format in text files are read and processed in design logic.

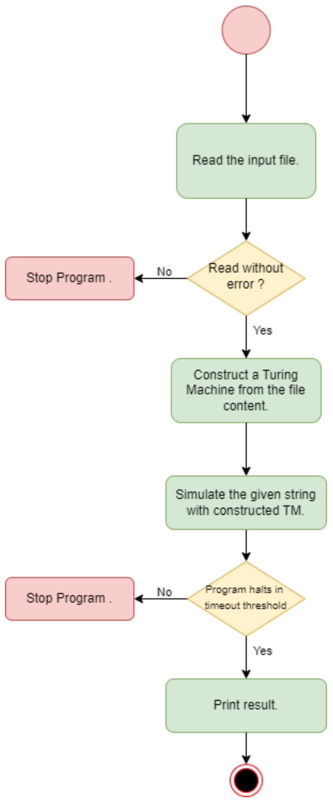
Objectives

The implementation follows below objectives.

1. Read the input file.
2. Construct a Turing Machine from the file content.
3. With the given input string, simulate the Turing Machine
4. Print the result if the program halts in timeout threshold, otherwise quit the program.

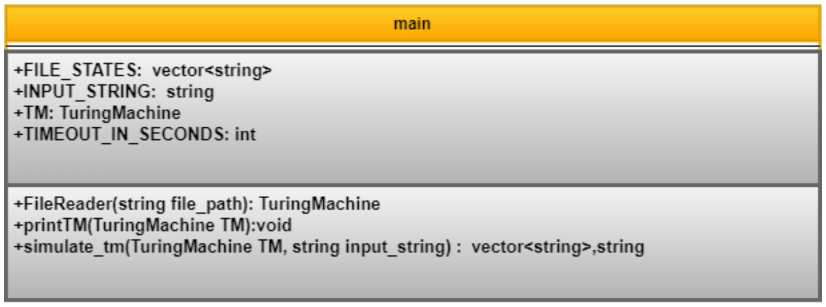
Software Architecture Overview

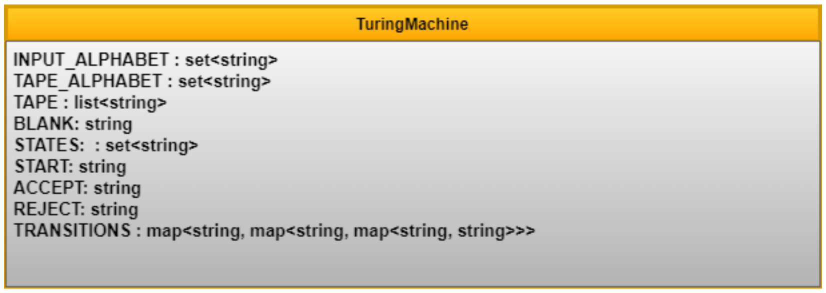
Activity Diagram


Read the input file.
Read without error ?
No
Stop Program .
Yes
Construct a Turing Machine from the file content.
Simulate the given string with constructed TM.
Program halts in timeout threshold
No
Stop Program .
Yes
Print result.

As can be seen from the activity diagram, the input file is read at the start of the program. If it is read with an error, the program prints the error to the console and exits. Otherwise, Turing Machine is constructed from the file content. After construction, TM is simulated with the given string. In the end, if the program halts in the timeout threshold, the result is printed. Otherwise, the program is terminated.

Class Diagram


main
+FILE_STATES: vector<string>
+INPUT_STRING: string
+TM: TuringMachine
+TIMEOUT_IN_SECONDS: int
+FileReader(string file_path): TuringMachine
+printTM(TuringMachine TM):void
+simulate_tm(TuringMachine TM, string input_string) : vector<string>,string


TuringMachine
INPUT_ALPHABET : set<string>
TAPE_ALPHABET : set<string>
TAPE : list<string>
BLANK: string
STATES: : set<string>
START: string
ACCEPT: string
REJECT: string
TRANSITIONS : map<string, map<string, map<string, string>>>

This program consists of 1 class and 1 struct. The struct named TuringMachine encapsulates the definition members of the Turing Machines. Turing Machine definition text files are read thanks to **FileReader** method. Each state in the text file representing which definition member of the grammar will be captured thanks to a vector of strings containing file states called **FILE_STATES**. While reading the file's content, **TuringMachine** structure is used to encapsulate the definition members of grammar. This structure gives the blueprint of the desired grammar. The code defines grammar members as follows: **INPUT_ALPHABET, TAPE_ALPHABET, TAPE, BLANK, STATES, ACCEPT, REJECT, and TRANSITIONS**. Also, **INPUT_ALPHABET, TAPE_ALPHABET, STATES** variables are encapsulated in the set data structure. **TAPE** variable is encapsulated in the list. **BLANK, START, ACCEPT, REJECT** variables are simply a string. Finally, **TRANSITIONS** variable is encapsulated in the 3-dimensional map data structure. After creating a TM from the text file, **simulate_tm** method is used for simulation with the given input. After the simulation, if the program halts in the predetermined timeout threshold, results are printed. Otherwise, the program stops.

Simulation Process

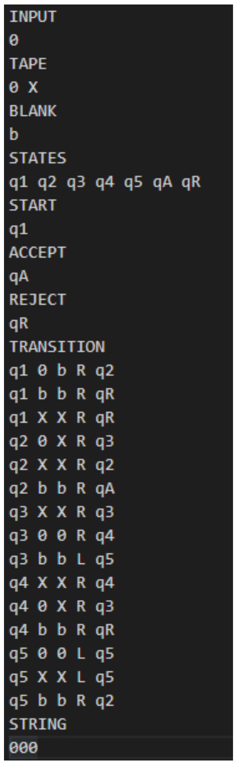
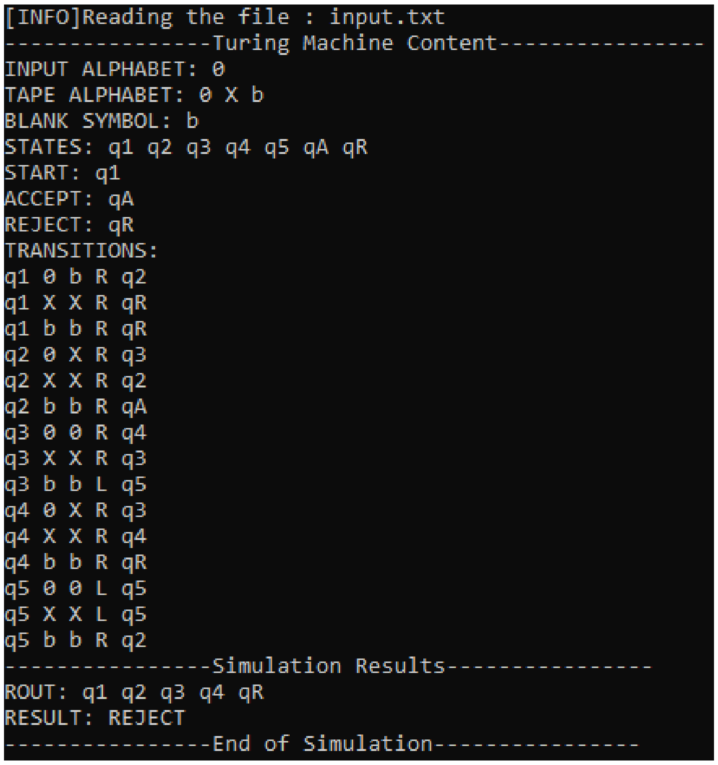
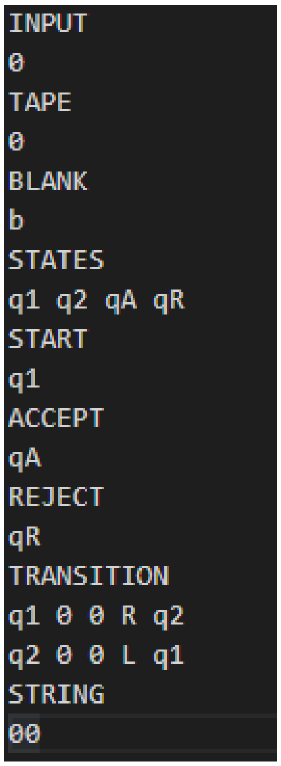
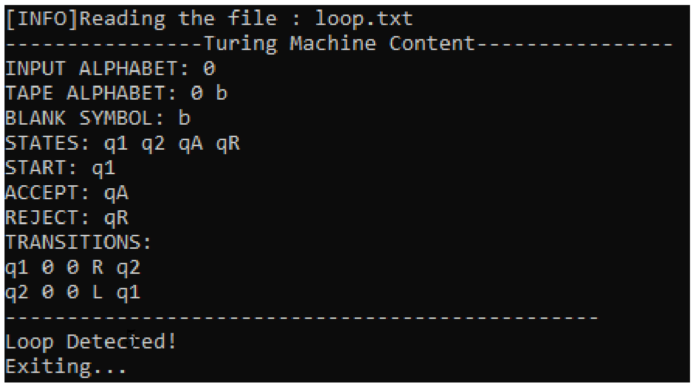
To simulate the Turing Machine with a given input string, **simulate_tm** function is executed with constructed TM and the input string. First, the tape of the TM is initialized by copying the content of the input string to the tape. Secondly, tape_head and current_state variables are created and set to the START state of TM. Also, a vector of strings typed variable namely route is created and START state is added to this vector. Thirdly, while loop is created within the condition of the current state is any state rather than ACCEPT or REJECT states. Inside the while loop, the current symbol is set to tape head and get the content of the Turing machine with respect to the current symbol. If the direction is right, the tape head is set to the right of the current tape content. If the tape is reached to end, the blank symbol is added to the end. If the direction is left, the tape head is set to the left of the current tape content. If the tape is reached to start, the tape head stayed at the same location. After deciding on the tape head, the current state is updated to a new state, and this new state is added to the route vector. At the end of the while loop, the route and final state are returned.

The Halting Problem

Since the halting problem is undecidable, we cannot ensure that the TM will halt with the given input string to it. Instead, we can set a timeout mechanism to make sure that program stops instead of running indefinitely. If the program does not halt in the timeout threshold set by **TIMEOUT_IN_SECONDS**, the program is terminated with an error stating loop is detected. This timeout mechanism is implemented using async function from future library in C++17.

Results

Here, I tested my algorithm with 2 text files that I created, input.txt and loop.txt. The program takes the path of the text file as an argument and prints the original TM and the simulation results in the desired format.

Turing Machine	Simulation
	
	

```
INPUT
0
TAPE
0 X
BLANK
b
STATES
q1 q2 q3 q4 q5 qA qR
START
q1
ACCEPT
qA
REJECT
qR
TRANSITION
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
STRING
000
```

```
[INFO]Reading the file : input.txt
----------------Turing Machine Content----------------
INPUT ALPHABET: 0
TAPE ALPHABET: 0 X b
BLANK SYMBOL: b
STATES: q1 q2 q3 q4 q5 qA qR
START: q1
ACCEPT: qA
REJECT: qR
TRANSITIONS:
q1 0 b R q2
q1 X X R qR
q1 b b R qR
q2 0 X R q3
q2 X X R q2
q2 b b R qA
q3 0 0 R q4
q3 X X R q3
q3 b b L q5
q4 0 X R q3
q4 X X R q4
q4 b b R qR
q5 0 0 L q5
q5 X X L q5
q5 b b R q2
----------------Simulation Results----------------
ROUT: q1 q2 q3 q4 qR
RESULT: REJECT
----------------End of Simulation----------------
```

```
INPUT
0
TAPE
0
BLANK
b
STATES
q1 q2 qA qR
START
q1
ACCEPT
qA
REJECT
qR
TRANSITION
q1 0 0 R q2
q2 0 0 L q1
STRING
00
```

```
[INFO]Reading the file : loop.txt
----------------Turing Machine Content----------------
INPUT ALPHABET: 0
TAPE ALPHABET: 0 b
BLANK SYMBOL: b
STATES: q1 q2 qA qR
START: q1
ACCEPT: qA
REJECT: qR
TRANSITIONS:
q1 0 0 R q2
q2 0 0 L q1
------------------------------------------------
Loop Detected!
Exiting...
```

Table 1: Table of results