CS3383 Theory of Automata

Project 1. Explore the Beauty of Automata

Instructor: Prof. Y Zhang

Team Members:

Bibek Dhungana (R11679703)

Dhan Limbu (R11718207)

Santona Subedi (R11737206)

Computer Science, *Texas Tech University*

TABLE OF CONTENTS

1. *Introduction* 3

2. *Data Structure* 3

3. *Algorithm* 3

4. *Test Cases* 6

5. *Acknowledgement* 8

1. INTRODUCTION

In real world, there are inputs that we have no control over it. This is why, we view things as non-deterministic. We call an automaton non-deterministic when there can be 0 or more state transitions for every input- state pair. For instance, in gaming applications, the best move is frequently unknown. However, it can be discovered by search with backtracking. When there are multiple options, we pick one and follow it, until it becomes obvious whether or not it was the greatest option. If not, we return to the previous decision point and consider the alternatives.

For this project, we chose to use a recursive backtracking algorithm to implement NFA. Every time, the automaton must make a decision, go through every option and perform a recursive call to see if any of them lead to a final state. If there is a cycle of 1 transition, this implementation might create an issue as it may get stuck in an infinite loop resulting in Stack Overflow. In order to solve this, we chose to keep a simple counter.

2. DATA STRUCTURE

Since, we are using recursive functions we do not have any explicit data structures used. However, recursive functions use call stack. Whenever the function is called, the function goes on the top of call stack. We have also used array (list in python). However, inbuilt data types like int, string and character are also used in the implementation.

3. ALGORITHM

An NFA is implemented by means of recursive search from start state for a path to a final state. The pseudo code for the algorithm is presented below:

Pseudo code:

Data:

given data

file: the file consisting of: a tuple (α, β) where α is a tuple for an NFA, β is a tuple (t1, . . ., tn) where for i ∈ 1..n, ti is a string over the alphabet of the NFA

unknown data

accept/reject for each input string.

result[i]: (i \in 1...n) is "Accepted" if input string i is accepted by the NFA, "Rejected." otherwise.

Side effect: print result[1] … result[n] to the screen.

Plan

// subproblem 1: check the given string data is in the NFA *alphabet*, if not print Rejected

Set the counter variable, *isRejected* as Boolean *true*.
Ask the user to choose file:
    If *file* exists:
        Read the contents of the *file* until *EOF*.
        If the string to be tested *not empty*:
            Extract the strings to be tested from file as *firststr, secondstr, thirdstr…*
            Call function *state_q0(string)* for all extracted strings
                If (*isRejected* is *true*):
                    Print (Rejected)
                Else:
                    Call other possible states based on computation tree.
                Endif
        Else:
            prompt the user to input string
            Call function *state_q0(string)* on the *input string*
                If (*isRejected* is *true*):
                    Print (Rejected)
                Else:
                    Call other possible states based on computation tree.
                Endif
    Endif

2. Pseudocode for function(s)

// Definition of function state_q0
Function name: state_q0(n)
    Input:
        n: string
    Output:
        n.a.
    Side effect: check if there exists transition or not for the alphabets and move accordingly.

Data

N.A.

Plan

Using global counter *isRejected*.
If length (n) = 0:
    isRejected = true
Else:
    If the next symbol in *n* transits from state *q0* to *itself* or some other state *q1* for any *accepted alphabet 'a':*
        Call *state_q0(n[n-1])*
        Call *state_q1(n[n-1])*
    Elseif next symbol in *n* transits from state *q0* to *itself* for any *accepted alphabet 'b':*
        Call *state_q0(n[n-1])*
    Endif
Endif

// Definition of function state_q1

Function name: state_q1(n)

Input:

n: string

Output:

n.a.

Side effect: check if there exists transition or not for the alphabets and move accordingly.

Data

N.A.

Plan

Using global counter *isRejected*.
If length (n) = 0 then
    isRejected = true
Else:
    If the next symbol in *n has no* transits from state *q1* to *any other states* for accepted *alphabet 'a':*
        Do nothing
    Elseif next symbol in *n* transits from state *q1* to *any other state q2* for any *accepted alphabet 'b':*
        Call *state_q2(n[n-1])*
    Endif
Endif

// Definition of function state_q2

Function name: state_q2(n)

Input:

n: string

Output:

n.a.

Side effect: Assuming final state which prints “accepted”

Data

N.A.

Plan

Declare global counter as *isRejected*.
If length (n) = 0 then
    isRejected = *false*
    Print(Accepted)
Else:
    Do nothing
Endif

4. TEST CASES

We used 3 different strings to test our program for its correctness.

1. We used a string that is accepted by our machine.
2. We used a string that is rejected by our machine.
   Below are the results of our test cases:

String	RESULT
110001	Accepted
000010	Rejected
10110001	Accepted
0001	Rejected

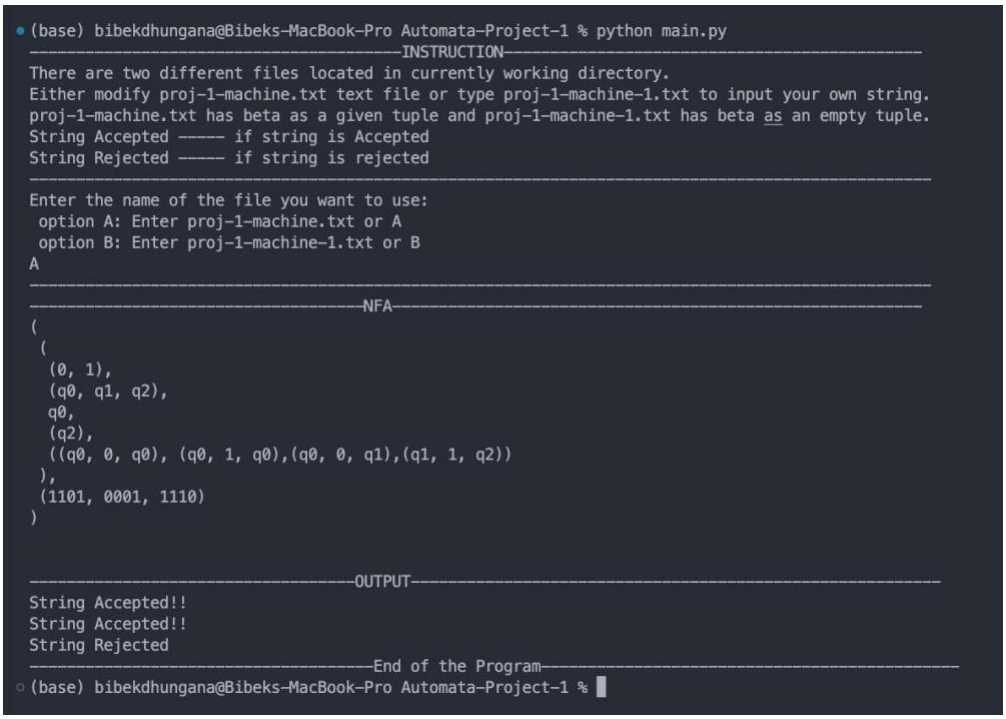

Fig 1. Output for proj-1-machine.txt

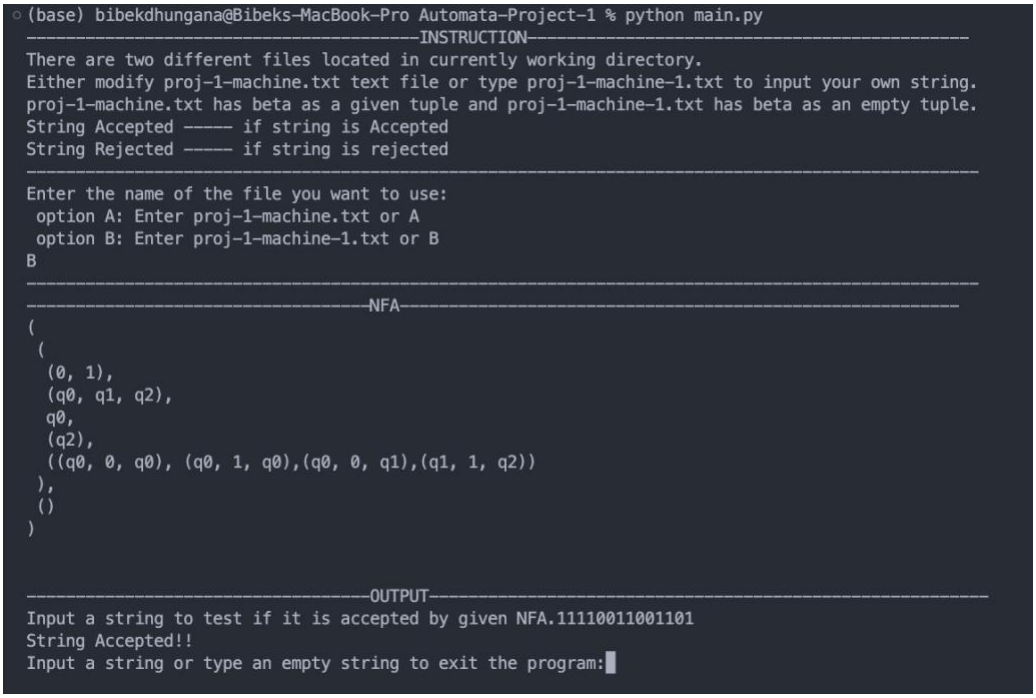

Fig 2. Output for proj-1-machine-1.txt

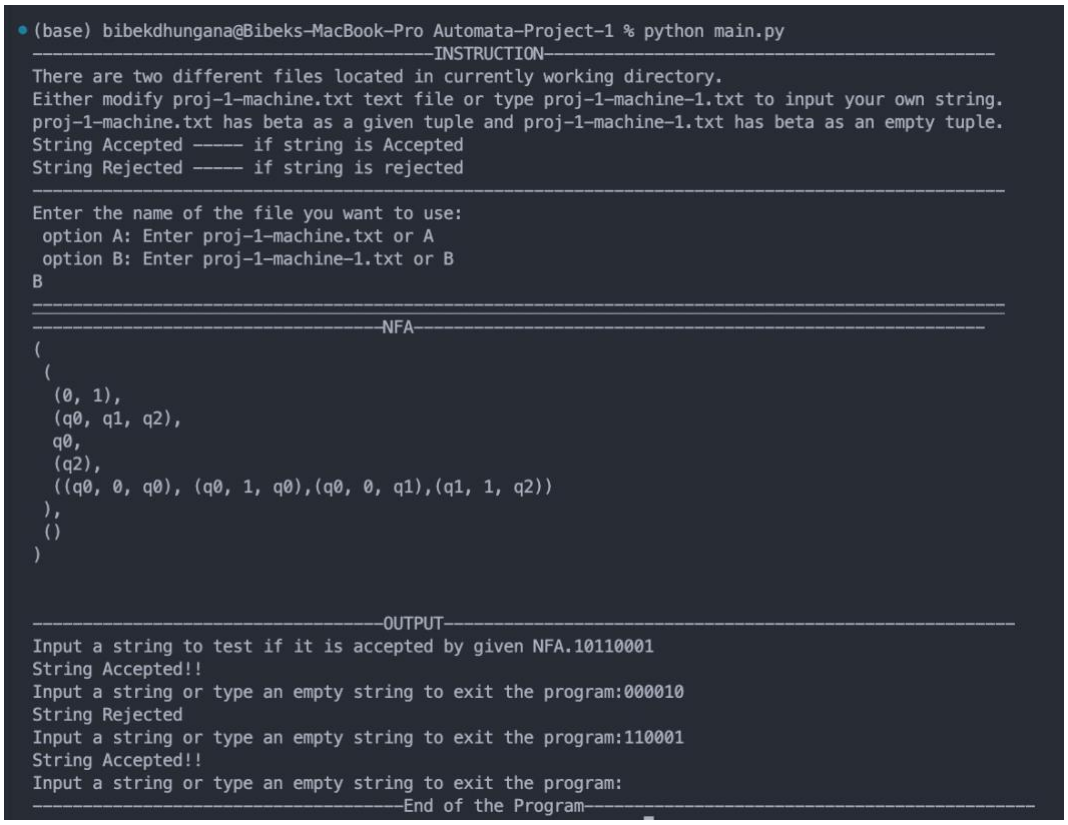

Fig 3. Output for our test cases

5. ACKNOWLEDGEMENT

Contribution by individual member is reported below:

Bibek Dhungana:

- Converted the pseudo code to Python code.
- Design and implementation of implicit data structure,
- Combined effort with team members in Outlining and explaining different sections of the report document and code documentation

Dhan Limbu:

- Documentation
- proofread of the report and testing multiple inputs.
- Combined effort with team members in Outlining and explaining different sections of the report document and code documentation

Santona Subedi:

- Generate pseudo code based on problem specification
- debugging, and refactoring the code
- Combined effort with team members in Outlining and explaining different sections of the report document and code documentation