Theory of Computation and Compiler Design Digital Assignment 1

Name: Om Ashish Mishra

Registration Number: 16BCE0789

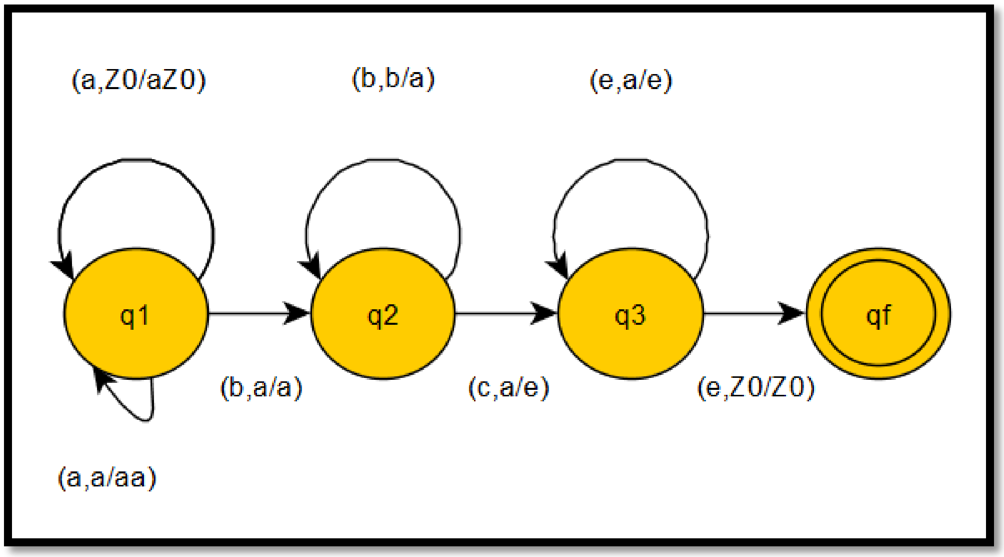
Slot: G1+TG1

1. A two-way pushdown automaton may move on its input tape in two directions. As usual for two-way automata we assume that the begin and end of the input tape is marked by special symbols. In this way the automaton can recognize those positions. Describe a two-way pda for each of the following languages. (a) { a n b n c n | n ∈ N } (easy) (b) { ww | w ∈ {a, b} ∗ } (nice puzzle) (c) { v c uvw | u, v, w ∈ {a, b} ∗ } (pattern matching; clever trick)

Ans:

<u>A two-way pushdown automaton:</u>

Unlike normal PDA where there is only one stack involved in the implementation of the language, and we have to check that if the stack is empty at the end to prove that it is possible to execute the string is present in the language. For example, $a^nb^nc^n$ is possible or not.



Therefore what we did is we took a stack and represented it this way and since we got an empty at end this id acceptable

But in a two way pushdown automation there is an involvement of 2 stacks and we will consider the language is acceptable if there s an empty second stack at the end. For example let's take the string, $a^nb^nc^n$ is possible or not.

Let L= aabbcc

On seeing 'a', we started the stack1 building:

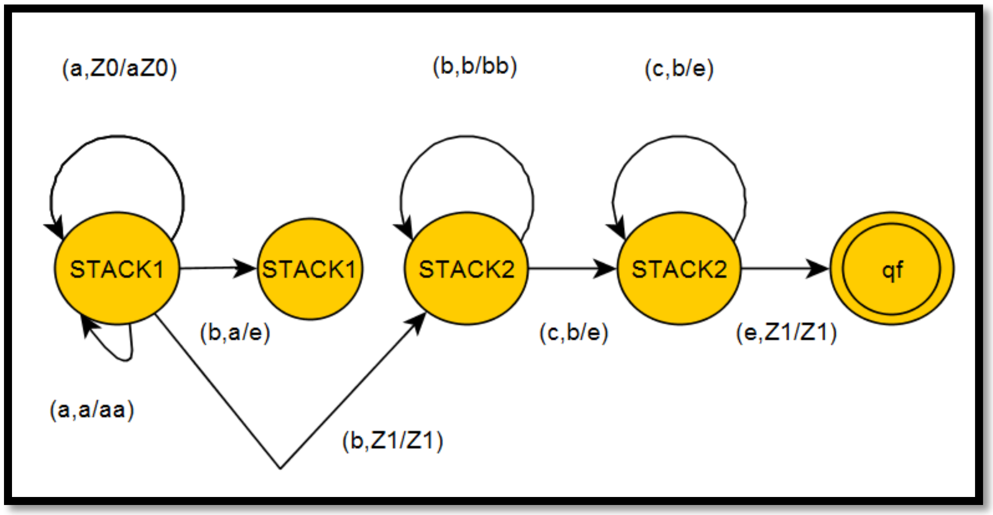
We inserted 2 'a's in stack 1.

On seeing 'b', we started the stack2 building:

Then we inserted 2'b's into stack 2 and then

On seeing 'c', we started the stack2 poping:

Then we popped the 2'b's from stack 2 and as we got it empty the language is acceptable.



The b operation goes simultaneously like popping in stack1 and pushing in stack2, even if not the elements of b will be checked by c at the end. If equal end point is reached else no.

Therefore (a) is done.

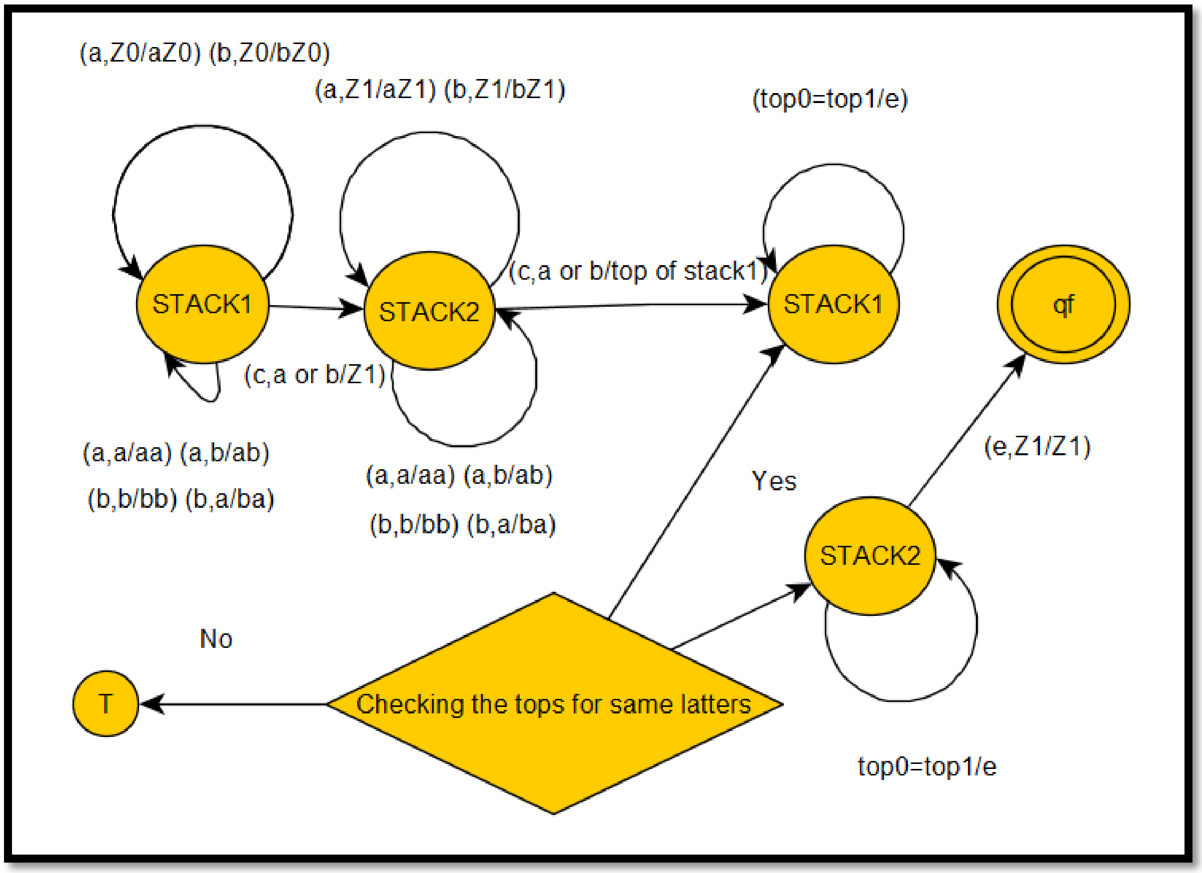
For (b) we see that (b) { ww | w ∈ {a, b} ∗ } Therefore we can check that :-

Since we don't know the length of the w as it is not fixed we insert a latter 'c' in between like L=wcwc

Therefore what we are doing is we check for latters in w and insert them in stack 1 till we encounter c.

Then we go for stack 2 to filling till we encounter 'c' then we pop from stack2 and stack1 if the tops of stack1 and stack2 matches thus we prove the language is accepted or the Trap state is reached.

Here Z0 = No elements in stack 1, Z1=No elements in stack 2, e = Eplison, top0= top element of stack 1 and top1 = top element of stack 2

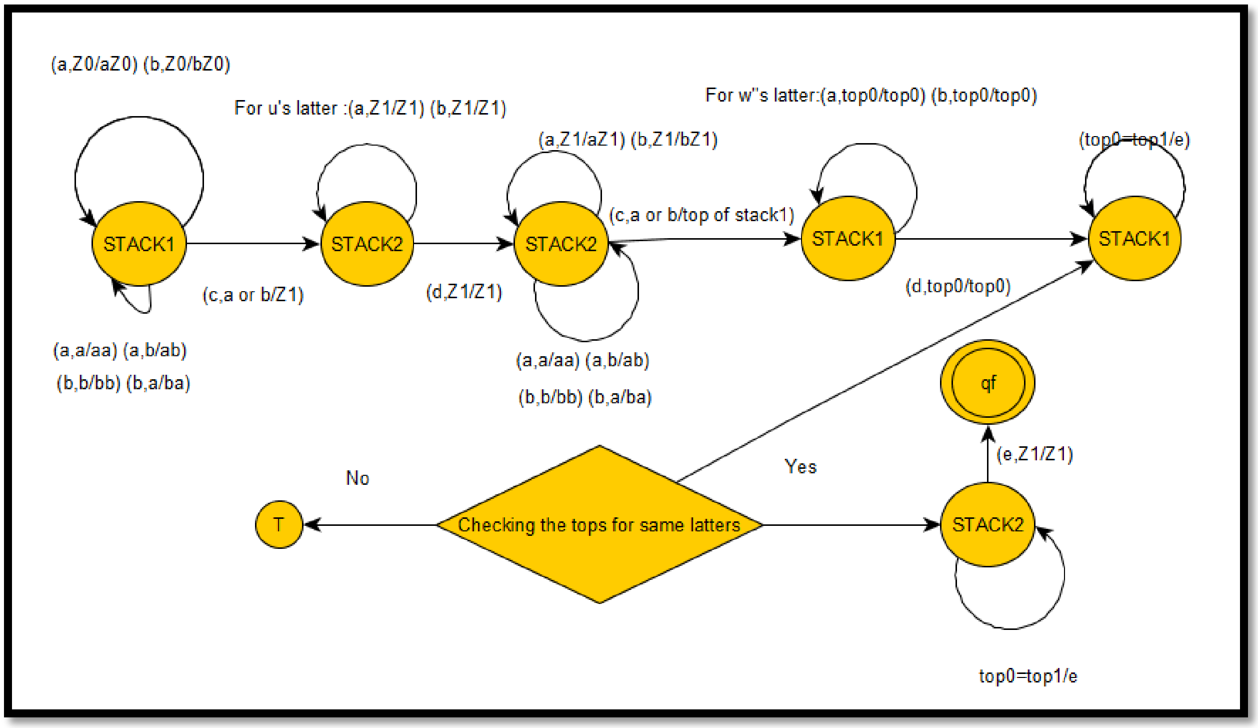


Therefore (b) is done. Here we pop till the stacks are empty by comparing to the length of w.

For (c) we see that { v c uvw | u, v, w ∈ {a, b} ∗ } we can check that

This question is basically a replica of previous question just the different is that we have to remove the u and w string to take place and conduct the (b)'s operation like similar to the previous case we insert an extra term 'd' into the language like vcu'd'v'd'w.

Thus the PDA will go like this:



Therefore (c) is done. Here we pop till the stacks are empty by comparing to the length of w.

2. In the case of finite state automata, the two-way model is equivalent to the usual one-way automaton. Find a proof of this result.

Ans:

<u>2-way automation:</u>

It reads input string of the finite automata from left to right and vice versa.

The tuples of the 2 way automata

- Q is a finite set of states (denoted by capital letters like A, B... etc).
- Σ is a finite set of input terminals (Denoted by small letters like a, b... etc).
- δ is the **transition function**, that is, **δ: Q X (Σ U { ├, ┤ }) → Q X { L, R }** **R**=right pointer and **L**= left pointer
- q_0 is the initial state.
- F is a **set of** final state
- ├ is the left end maker
- ┤ is the right end maker

Transition Table :

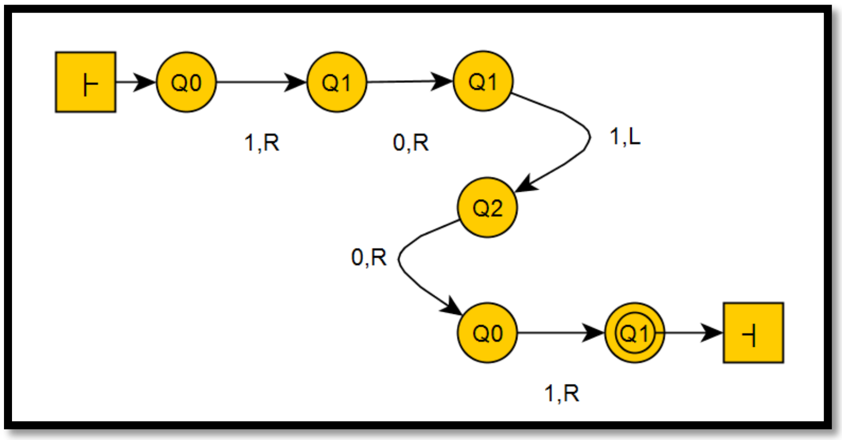
States	**0**	**1**
-> q0	(q0, R)	(q1, R)
q1 (Final State)	(q1, R)	(q2, L)
q2 (Dead State)	(q0, R)	(q2, L)

To understand the concept of two - way FA and, Let us take an example.

Suppose L = ‘10101’

- ➔ Initially : ├(q0) 10101 ┤
- ➔ (q0,1)=q1: ├ 1(q1) 0101 ┤ R
- ➔ (q1,0)=q1: ├ 10 (q1) 101┤ R
- ➔ (q1,1)=q2: ├ 101 (q2) 01┤ L
- ➔ (q2,0)=q0: ├ 1010 (q0) 1┤ R
- ➔ (q0,1)=q1: ├ 10101 (q1) ┤ R

Since the end of the string has been reached and we also have reached the marker thus the string of is acceptable.



Here we are going bit by bit and checking the acceptability of the language by 2-way FA and it is acceptable since we reached q1 state which is the final state.

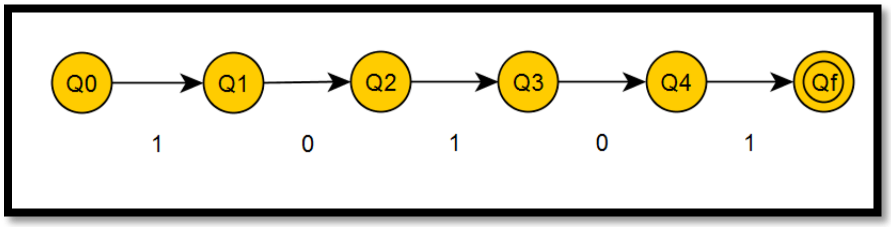
1-way automation:

It reads input string of the finite automata from left to right only.

The tuples of the 2 way automata

- Q is a finite set of states (denoted by capital letters like A, B... etc).
- Σ is a finite set of input terminals (Denoted by small letters like a, b... etc).
- δ is the **transition function**, that is, **δ: Q X Σ → Q(DFA) or 2^Q(NFA)**
- q_0 is the initial state.
- F is a **set of** final state

For NFA:-



The final state has been achieved therefore the string is acceptable.

Thus we see the string can we generated by using both 2 way FA and one way FA.

The 2-way model is equivalent to 1-way model.

Hence proved.

3. Stack automata are pda that may inspect their stack. The chapter states: "stack automata that do not read input during inspection of the stack are equivalent to pda's". Verify this fact.

Ans:

Stack Automata

The stack automata is a pda which reads the input and and checks it is to be inserted or popped from the stack.

Non reading Stack Automata

The stack does not read the input and goes on inserting till the last element is encountered. If the top is equal to the last latter or operation then the language os the string is acceptable.

Pushdown Automata

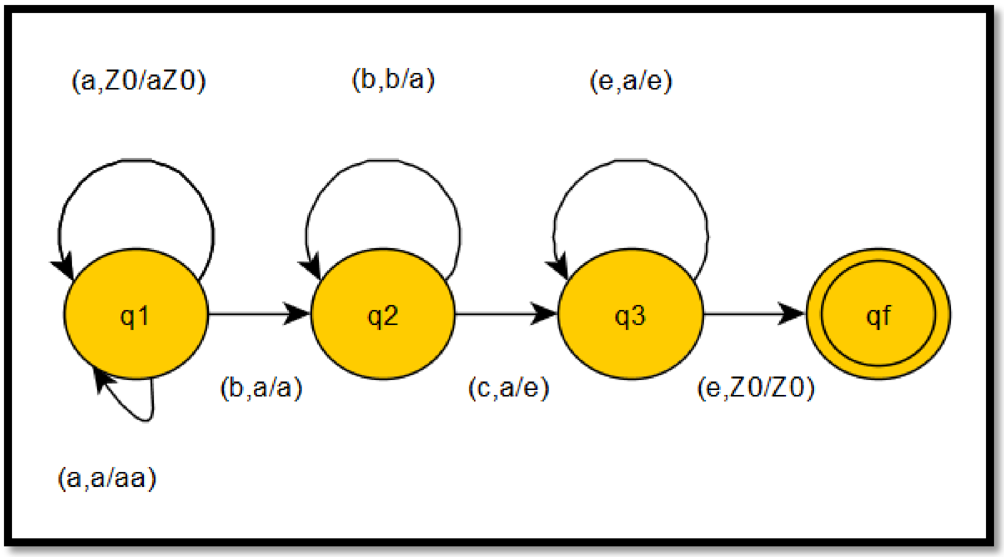
Unlike stack the PDA is also cannot read the elements in the stack.

The PDAs take the elements or latters from the string of language and do operations by comparing the top of the stack elements and not by reading and making the substring to the string provided and comparing it with the parts as stack automata does. Thus Non reading stack Automata is similar to the Pushdown Automata.

Verification:-

L= $a^n b^n c^n$

Let L=aabbcc



In this ‘a’ is pushed into the stack as many times it comes and when ‘b’ comes we ignore it and at last when ‘c’ is encountered we pop the elements.

How we check?

If a is seen we push it and top increases by 1 on Z0.

If a is seen again we push it and top increases by 1 on a.

If b is seen then we do no change to stack.

If b comes again, no change to take.

If c comes we pop a from the stack and top decreases by 1.

If c comes again we pop a from stack till we reach Z0 and top decreases by 1.

Thus the stack becomes empty and we get the language or the string accepted.

Thus we compare and read the data and thus they are same.

-------------------------------------XXXX---