TOC-2019-20

1.a)

This argument is not acceptable. As we do not know the actual value of p, we cannot guarantee that p is a prime number. So we cannot guarantee that a^p is in L. So, the argument is unacceptable.

1.b)

In cryptography we prefer hard problems. Because we want the code hard to break without the key or password.

for example, RSA is a widely used scheme in encoding. In RSA,

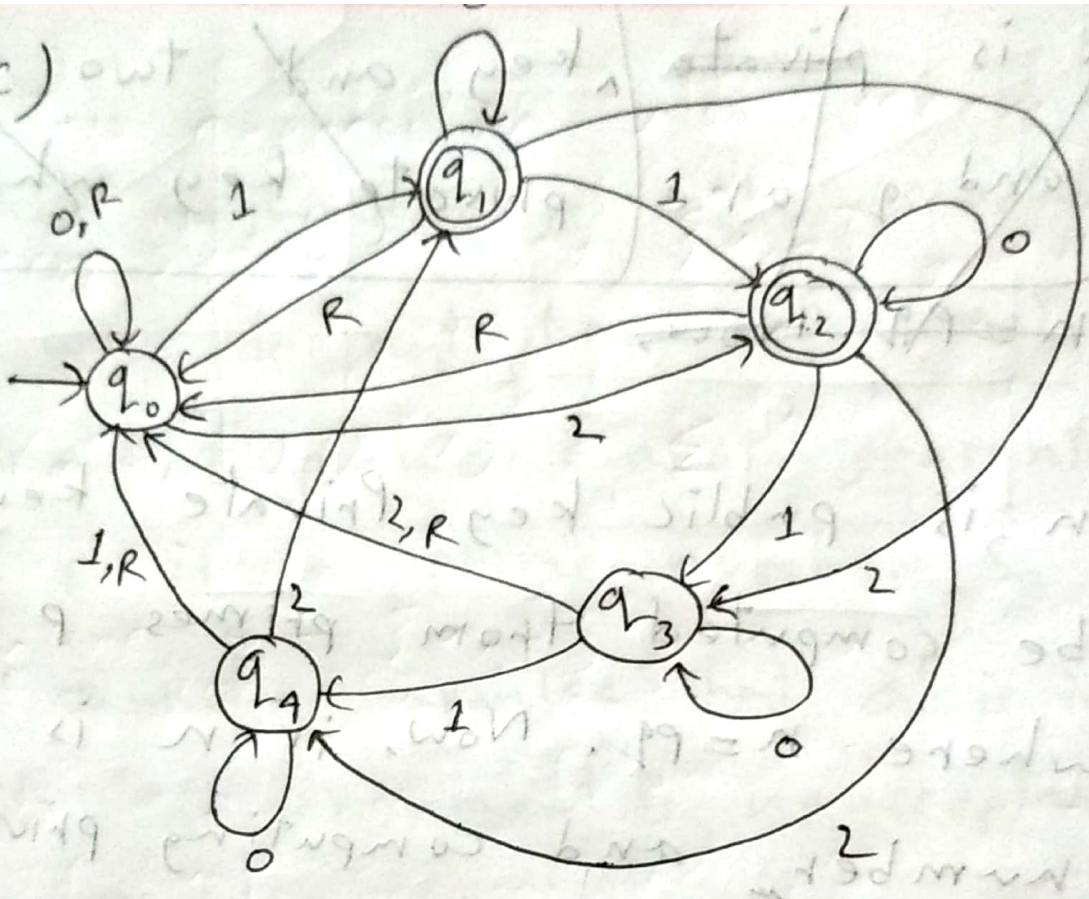
public

n is ~~private~~ key and two primes p and q are private key where

~~n = pq. Now,~~

n is public key. Private key can be computed from primes p and q where $n = pq$. Now, if n is a large number and computing private key is hard, then without p and q, it is very difficult to break the code. Thus, in cryptography hard problem is preferred.

1. c)



2. a)

A state in a finite automaton stores some information. To ~~recgo~~ recognize a language, we need to remember the characteristics of the string in the input. States help to store some information

2.b)

i) $\emptyset^* = \{\epsilon\} \subseteq (a \cup b)^*$

$a^* \subseteq (a \cup b)^*$

$b^* \subseteq (a \cup b)^*$

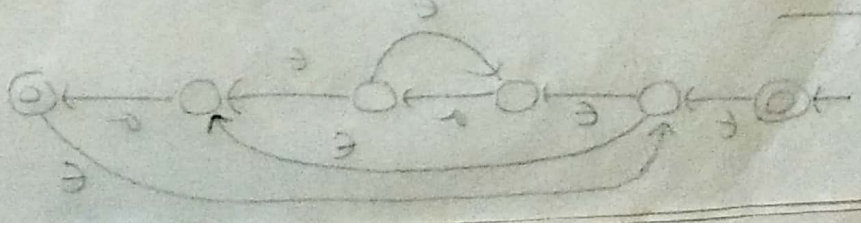
$\therefore \emptyset^* \cup a^* \cup b^* \cup (a \cup b)^* = (a \cup b)^*$

ii) ~~Given~~ Using given expression, we can express any string of alphabet $\in \{a, b\}$

So,

$((a^* b^*)^* (b^* a^*)^*)^* = \Sigma^*$

where $\Sigma = \{a, b\}$

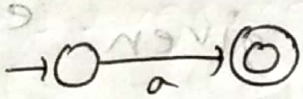


iii) This expression also generates every string using $\{a, b\}$

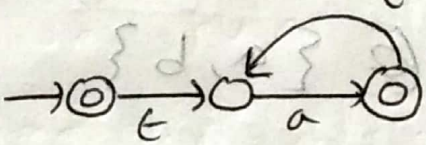
$\therefore (a^*b)^* \cup (b^*a)^* = \Sigma^*$

where, $\Sigma = \{a, b\}$

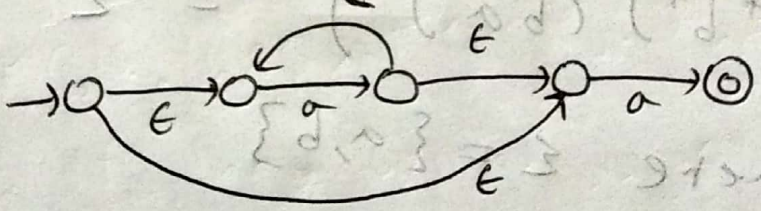
3. a) For a



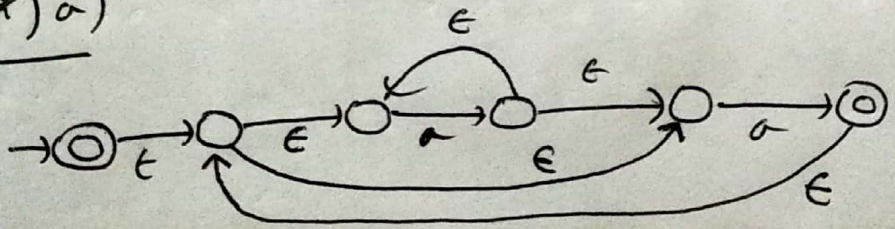
For a^*



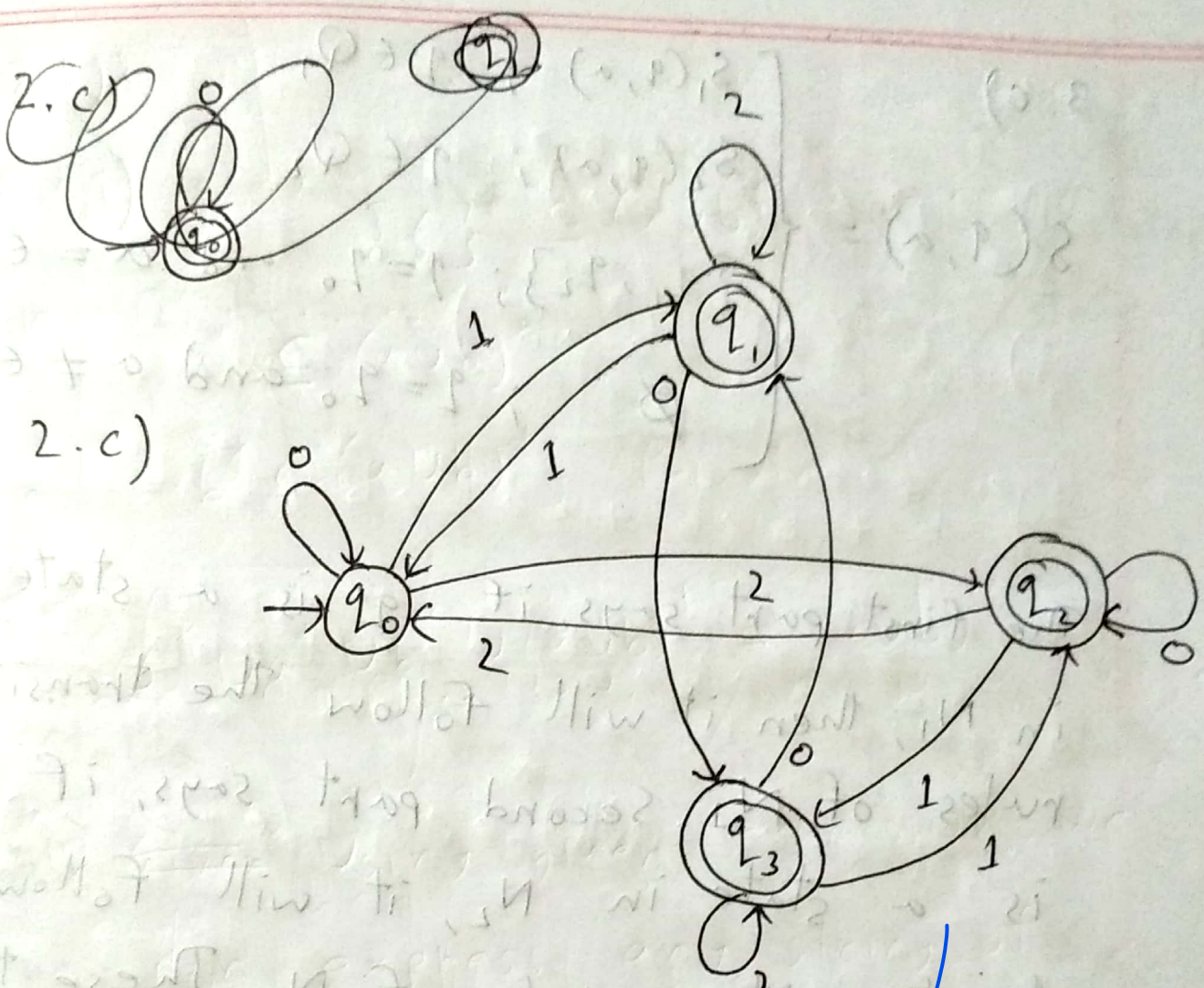
For $(a^*)a$



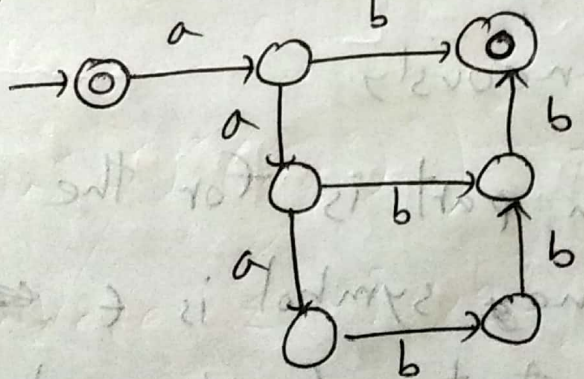
For $((a^*)a)^*$



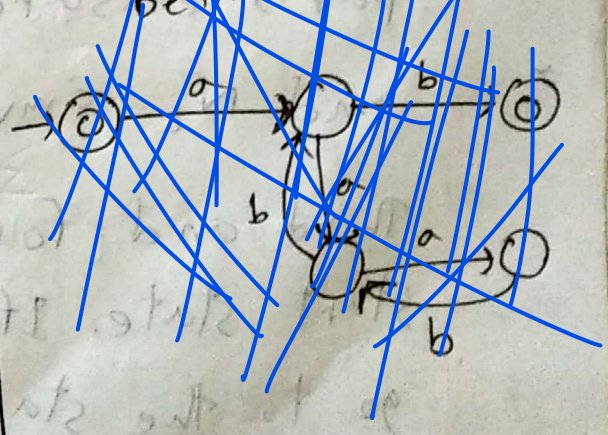
2.c)



3.b)



Better



3. c)

$$\delta(q,a) = \begin{cases} \delta_1(q,a) ; & q \in Q_1 \\ \delta_2(q,a) ; & q \in Q_2 \\ \{q_1, q_2\} ; & q = q_0 \text{ and } a = \epsilon \\ \emptyset ; & q = q_0 \text{ and } a \neq \epsilon \end{cases}$$

The first part says, if q is a state in N_1, then it will follow the transition rules of N_1. Second part says, if q is a state in N_2, it will follow the transition rules of N_2. These two parts ensure that N will simulate N_1 and N_2 simultaneously.

Third and fourth part is for the new start state. If the symbol is ϵ it will go to the start states of N_1 and N_2. If the symbol is not ϵ, this branch will die out.

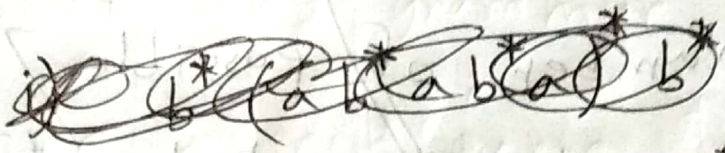
4. a)

minimum number of possible states is 1. Because only the start state maybe reachable from the

4. a) Minimum number of possible states is 1. Because no other states except the start state may be reachable. So, we just need the start state.

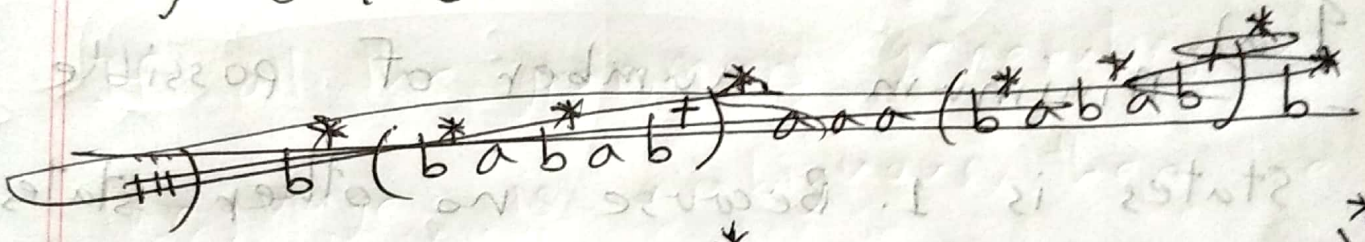
Maximum number of possible states is $2^5 = 32$. Because in NFA multiple states can be reached using one move. So, in DFA we need to take the subsets of Q of NFA.

4. b)



i) $b^*(ab^*ab^*ab^*)^*$

ii) $b^* + b^*ab^* + b^*ab^*ab^* + b^*ab^*ab^*ab^*$



iii) $((\epsilon + a + aa)b)^*aaa(b(\epsilon + a + aa))^*$

4. c) Let,

$$S = a^{2p}b^{3p}a^{p} \in L$$

where p is the pumping length.

Now, we will divide S into xyz.

As, $|xy| \leq p$, y cannot go to the 'b' region. y will only contain a.

Now, if we pump in y, for example $xyyz$, the first segment will contain more 'a' than 2p. So, $xyyz \notin L$ for any choice of y.

So, L is not regular.

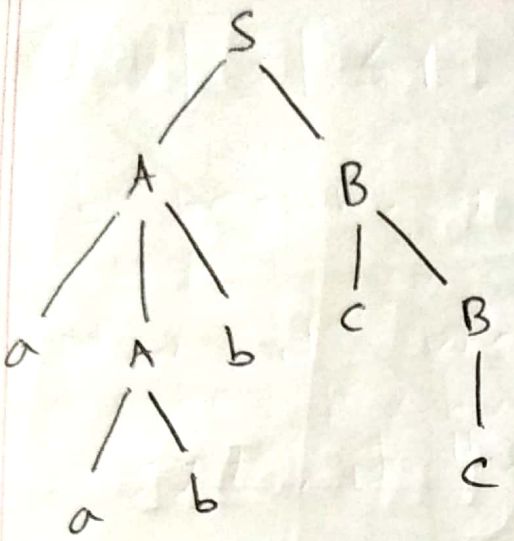
5. i) $S \rightarrow (S) \mid SS \mid \epsilon$

ii) $S \rightarrow 1SO \mid OS1 \mid SS \mid \epsilon$

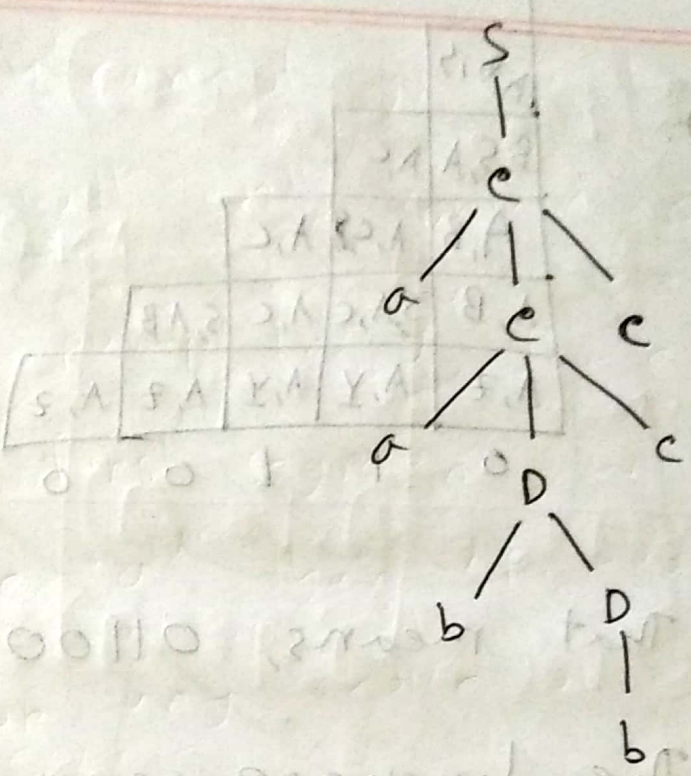
6. If any grammar has more than one distinct parse tree for same string, it is called ambiguous grammar.

If all the grammars generating a language is ambiguous, then the language is inherently ambiguous language.

For the given grammar if we consider the string aabbcc we can have two distinct parse trees:



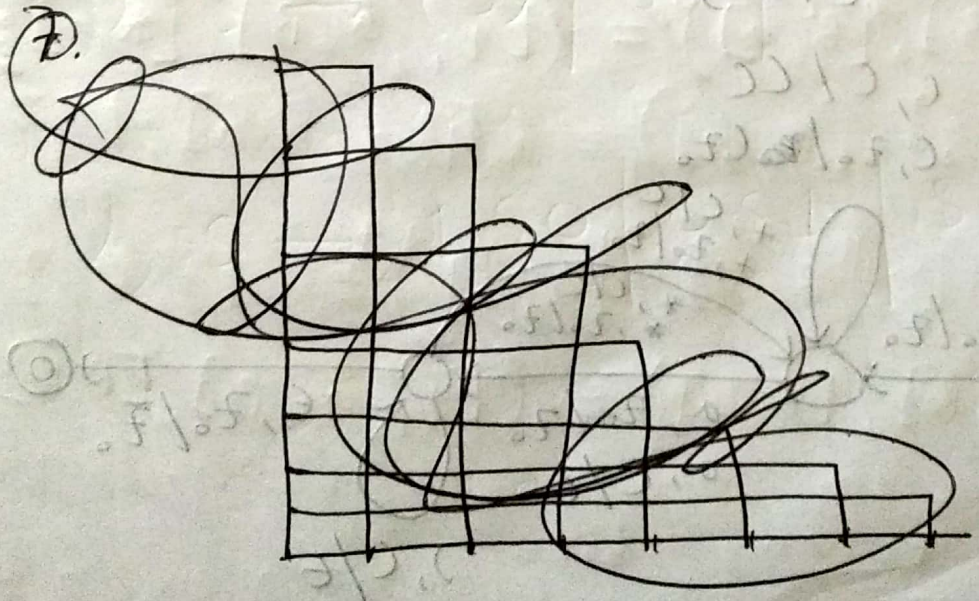
aabbcc



aabbcc

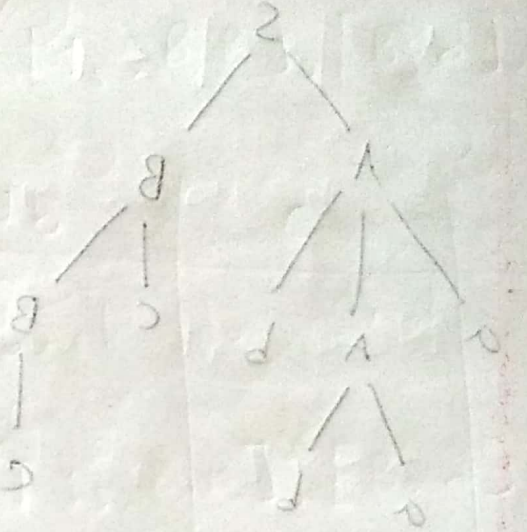
So, the ~~term~~ grammar is ambiguous.

2.



7.

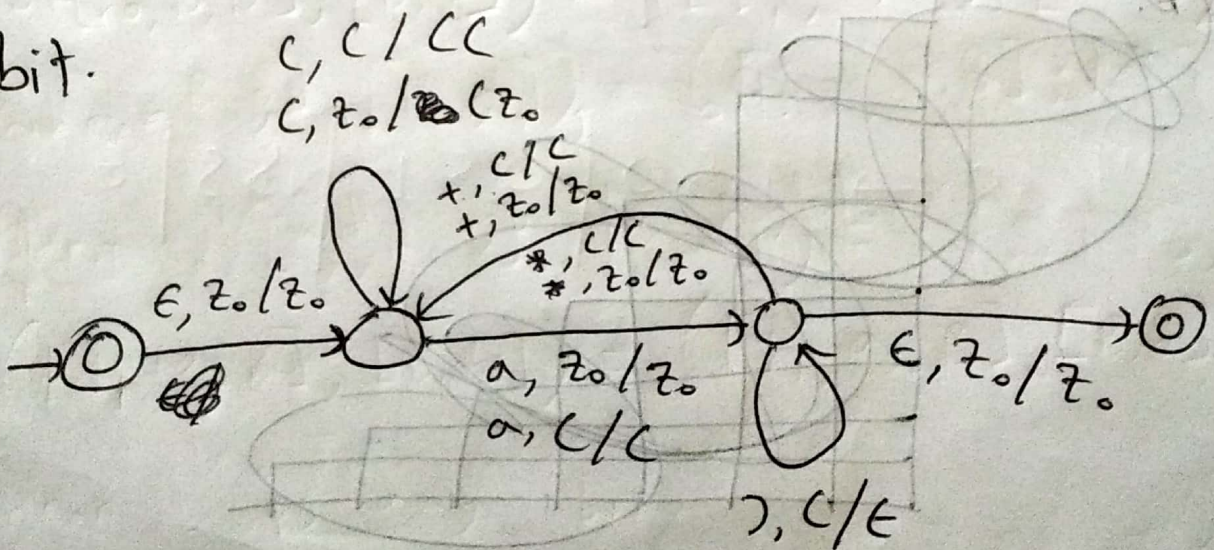
A,B,S				
B,S,A	A,C			
A,B	A,C	A,C		
A,B	S,A,C	A,C	S,A,B	
A,Z	A,Y	A,Y	A,Z	A,Z
0	1	1	0	0

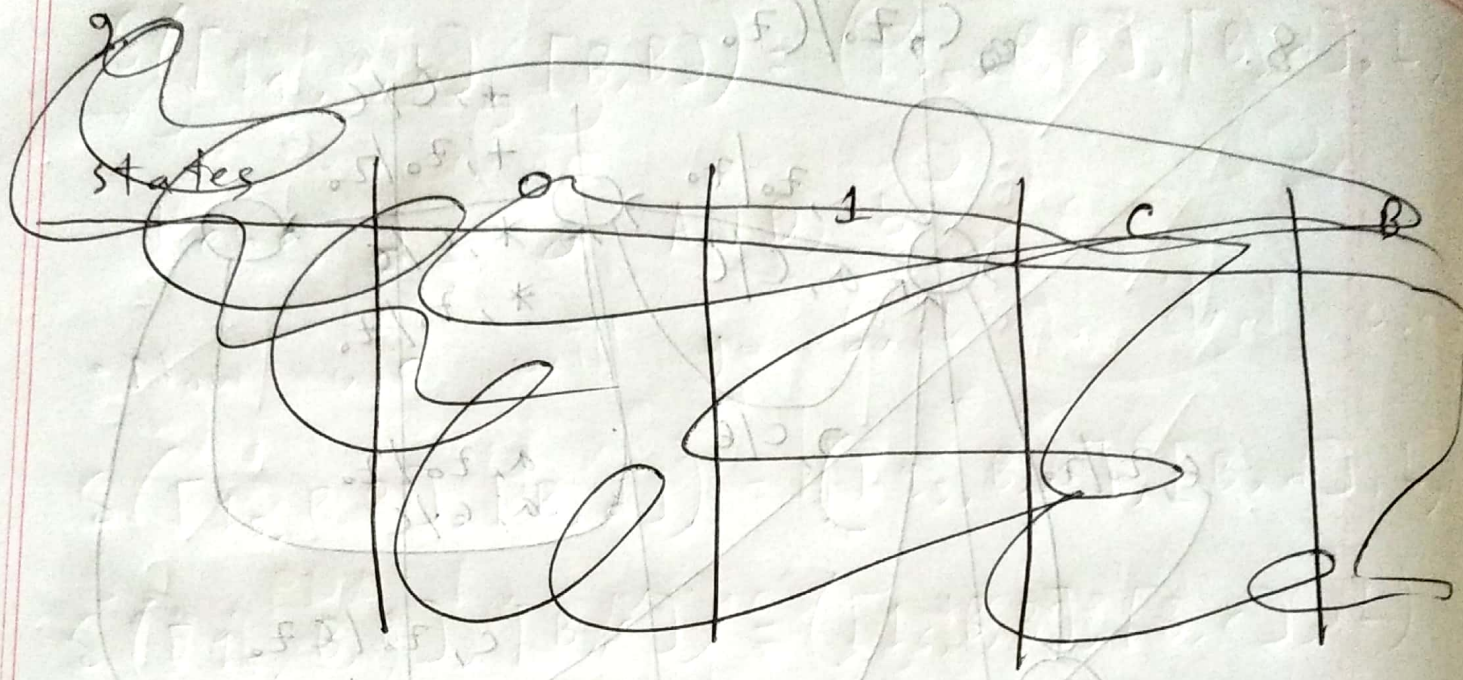


That means, 01100 is in L(G).

The language generated by the above grammar is the language containing all the binary strings that starts and ends with the same bit.

8.





9.

- TM will start in q_0 state and two blanks in it's storage:

$$\delta([q_0, B, B], [B, a]) = ([q_1, a, B], [*, a], R)$$

where a is any bit.

- Then it will go to right.

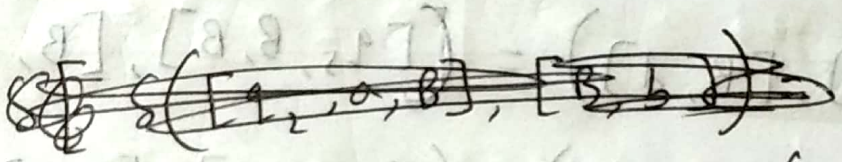
$$\delta([q_1, a, B], [B, b]) = ([q_1, a, B], [B, b], R)$$

where b is any bit.

- When we find the first c,

$$\delta([q_1, a, B], [B, c]) = \delta([q_2, a, B], [B, c], R)$$

- We will go through the marked cells and search for an unmarked one.



$$\delta([q_2, a, B], [*, b]) = ([q_2, a, B], [*, b], R)$$

$$\delta([q_2, a, B], [B, b]) = ([q_3, a, b], [*, b], R)$$

- Then we will go through to find the second c.

$$\delta([q_3, a, b], [B, d]) = ([q_3, a, b], [B, d], R)$$

$$\delta([q_3, a, b], [B, c]) = ([q_4, a, b], [B, c], R)$$

- Now we will find a blank cell and write:

$$\delta([q_4, a, b], [B, d]) = ([q_4, a, b], [B, d], R)$$

$\delta([q_4, a, b], [B, B]) = ([q_5, B, B], [B, d], L)$

Here, $d = a$ AND b

• Now we will go back to start over.

$\delta([q_5, B, B], [B, d]) = ([q_5, B, B], [B, d], L)$

$\delta([q_5, B, B], [B, c]) = ([q_6, B, B], [B, c], L)$

$\delta([q_6, B, B], [B, d]) = ([q_6, B, B], [B, d], L)$

$\delta([q_6, B, B], [*, d]) = ([q_6, B, B], [*, d], L)$

$\delta([q_6, B, B], [B, c]) = ([q_7, B, B], [B, c], L)$

$\delta([q_7, B, B], [B, d]) = ([q_7, B, B], [B, d], L)$

$\delta([q_7, B, B], [*, d]) = ([q_0, B, B], [*, d], R)$

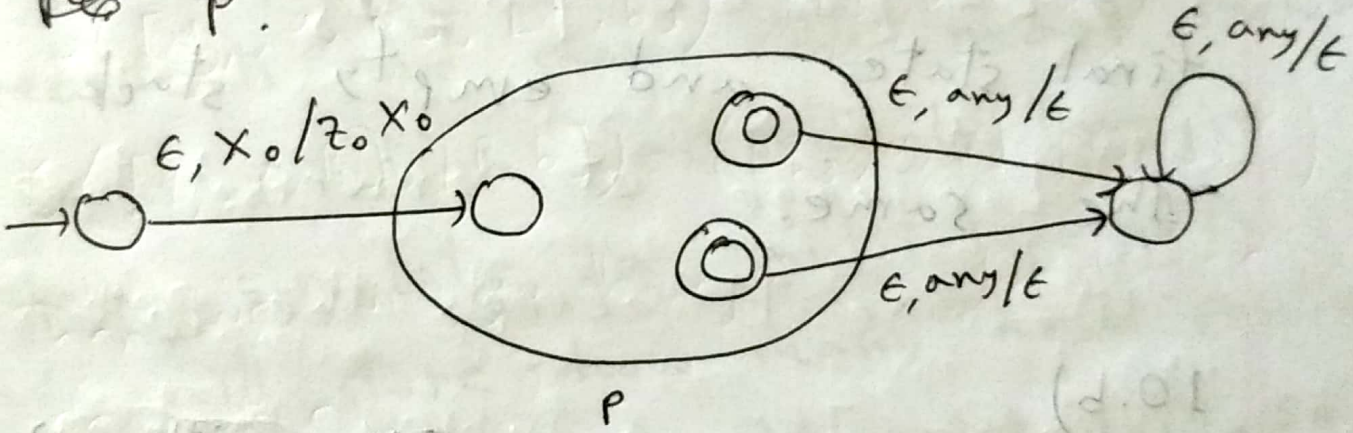
Now, if we encounter c, we will go to a stop state. TM will halt.

$\delta([q_0, B, B], [B, c]) = ([q_8, B, B], [B, c], R)$

10. a)

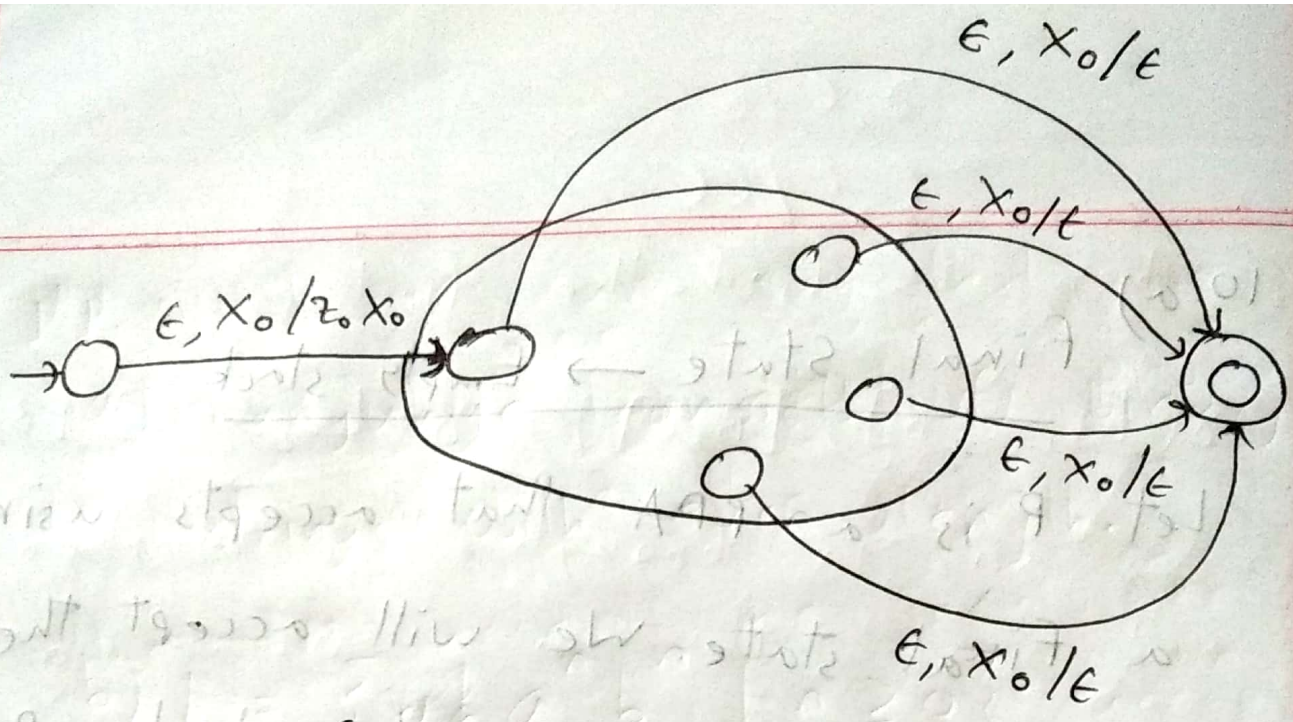
Final State → Empty stack

Let P is a PDA that accepts using a final state. We will accept the language using empty stack PDA P'.



Empty stack → Final state

Let P is a PDA that accepts using an empty stack. We will accept the language using final state PDA P'.



set of
So, ~~PDA~~ languages accepted by final state and empty stack are the same.

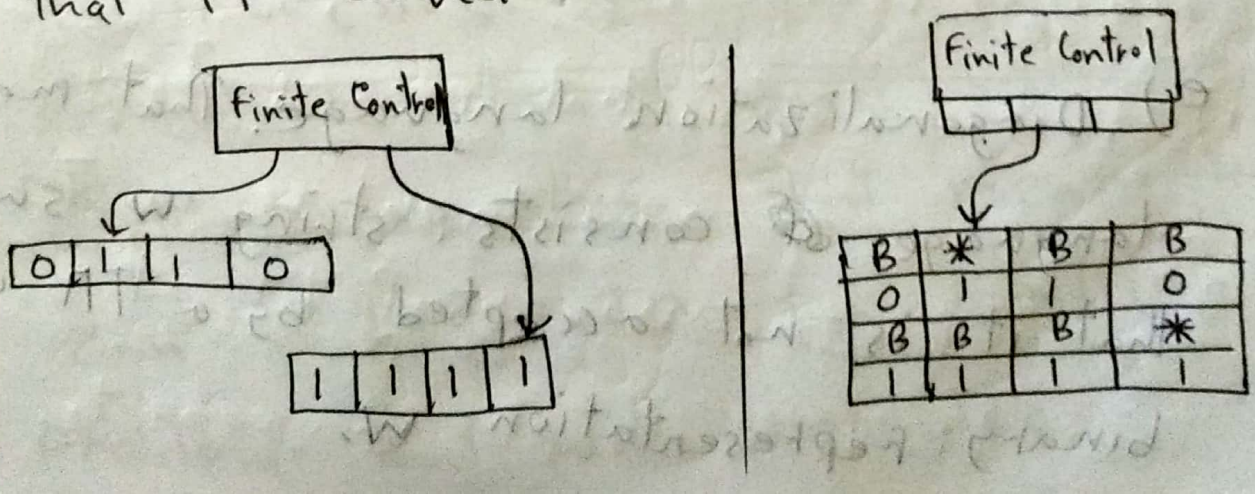
10.b)

We will use a TM with 2n tracks to simulate an n-tape TM. ~~The even two consecu~~ Between two consecutive tracks, first track will show the position of the i-th head and second track will contain the

contents of the i-th tape.

Now, TM will go to the position of the heads and read the tape symbols. Then it will change state. Then it will again go to the head positions, update the tape contents and move the mark showing head position accordingly. Thus, it can simulate a multi tape TM.

We will store how many head positions the head have to its ~~right~~ left, so that it ~~do not get~~ does not get lost.



11.

a) Language consists of binary strings with even length.

b) $L = \{a^n b^n \mid n \geq 0\}$

c) $L = \{a^n b^n c^n \mid n \geq 0\}$

d) Language of universal Turing machine. It accepts the pair (T, w) where T is a turing machine that accepts the string w.

e) Diagonalization language. That means language consists string w such that it is not accepted by a TM with binary representation w.

13.

Halting problem is the problem that determines

12. Given a TM and a string, we have to determine if the TM halts if given the string in the input with or without accepting. This is the halting problem.

At first, we assume halting problem is recursive. That means it will guarantee to halt with or without accepting. That means we can make a TM for halting problem that will guarantee to halt.

At first, we can say that we can make a TM for halting problem. It will simulate the given TM and if the TM accepts the input, our TM for halting problem will accept and halt. So, halting problem is RE.

Now, if halting problem is recursive, universal language will recursive too. At first we will simulate the input pair (T, w) in the TM for haulting Problem. Now:

i) If it accepts, that means T will halt. So, we can now input w in universal TM and see if it accepts or rejects.

ii) If a TM for halting problem rejects, that means T will not halt for w. So, we can say T will not accept w and we can reject (T, w)

Thus, we can also make universal language recursive reducing it to halting problem. But we know, universal language is not recursive. So, there is a contradiction.

So, halting problem is RE but not recursive.

13. a) It is true. Because SAT is an NP-complete problem. That means, every problem in NP can be reduced to SAT in polynomial time. If SAT can be solved in polynomial time, all problem in NP can be solved in polynomial time, resulting P=NP.

b) False. The set of languages decided by polynomial time non-deterministic TM is the NP class. NP class is a subset of PSPACE (languages decided by polynomial space deterministic TM).

c) True. The first one is PSPACE and the latter one is NPSPACE.

PSPACE = NPSPACE