Unit 2 Dt-31/07/23

Regular Expression

- It is an important notation for specifying patterns.
- The language accepted by finite automata can be easily described by simple expressions called regular expressions.
- The languages accepted by regular expressions are known as regular languages.
- Each pattern matches a set of string, so regular expressions serve as names for a set of strings.

Ex:- $a(bb)^*ab$.

Regular set

- Any set that represents the values of regular expression is called a regular set.

Ex1: a^*

$L = \{\epsilon, a, aa, aaa, \ldots\}$

Ex2: a^+

$L = \{a, aa, aaa, \ldots\}$

operations on Regular expression

1. union:- If L_1 and L_2 are two regular languages then $L_1 \cup L_2$ is also regular

Ex: $L_1 = \{\epsilon, 0, 1\}$ $L_2 = \{10, 11, 01\}$

$L_1 \cup L_2 = \{\epsilon, 0, 1, 10, 11, 01\}$

2. concatenation:- If L_1 and L_2 are two regular languages then $L_1 \cdot L_2$ is also regular.

Ex: $L_1 = \{0, 1\}$ $L_2 = \{00, 11\}$

$L_1 \cdot L_2 = \{000, 011, 100, 111\}$

3. Kleene closure :- If L_1 is regular then L_1^* is also regular.

Ex: $L_1 = \{0, 1\}$

$L_1^* = \{ \epsilon, 00, 01, 11, 10, 000, \ldots \}$.

Algebraic Properties of Regular Expression

1. closure :- If R_1 and R_2 are regular expressions then R_1^* is also regular expression and similarly $R_1 + R_2$ is also regular expression and $R_1 \cdot R_2$ is also regular expression.

2. closure laws :- a. $(R^*)^* = R^*$

b. $(\phi)^* = \epsilon$. c. $R^+ = R(R^*)$

d. $R^* = R^* + \epsilon$.

3. Associativity :- If R_1, R_2 and R_3 are regular expression then

$R_1 + (R_2 + R_3) = (R_1 + R_2) + R_3$.

~~$R_1(R_2 + R_3) = R_1 \cdot R_2 + R_1$~~

$R_1(R_2 \cdot R_3) = (R_1 \cdot R_2) R_3$.

4. Identity :- In the case of union operation. if $R + x = R$ and x is epsilon (ϵ). so, the ϵ can be said as identity operator for union.

. if $R \cdot x = R$ and x is ϵ

so, then ϵ can be said as identity operator for concatenation

5. Annihilator :- if $R \cdot \phi = \phi$ then the ϕ can be said as Annihilator for concatenation operation.

6. commutative :- if R_1 and R_2 are two regular expression then $R_1 + R_2 = R_2 + R_1$ $\quad R_1 \cdot R_2 \neq R_2 \cdot R_1$

$a + b = b + a$ $\quad a \cdot b \neq b \cdot a$

7. Distributive property :- If R_1, R_2 and R_3 are regular expressions then.

- $(R_1 + R_2) R_3 = R_1 \cdot R_3 + R_2 \cdot R_3$. (Right distributed)
- $R_1 (R_2 + R_3) = R_1 R_2 + R_1 R_3$ (left distributed)
- $(R_1 \cdot R_2) + R_3 \neq (R_1 + R_3)(R_2 + R_3)$.

  $(ab) + c \neq (a+c)(b+c)$

8. Idempotent law :- $R_1 + R_1 = R_1$
$R_1 \cdot R_1 \neq R_1$

Dt- 01/08/23

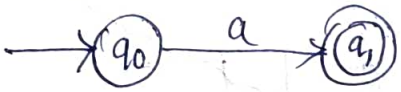
Construction of FA from RE

- There are two Methods for construction of finite automata from regular expression.

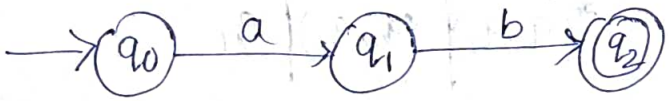
(1) Thompson's construction (2) subset method

(1) Thompson's construction :- we will reduce the regular expression into smallest regular expression and converting those to NFA and DFA (if required)

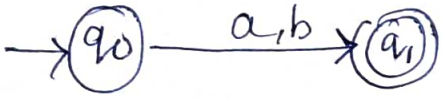
Case1: For a regular expression 'a' we can construct the following finite Automata.



Case2: If a regular expression 'ab' we can construct the following FA.



Case3: For a regular expression of the form '(a+b)' we can construct the following FA.



Case 4: For a regular expression of the form "$(a+b)^*$" we can construct the following FA.

Steps for Thompson's construction

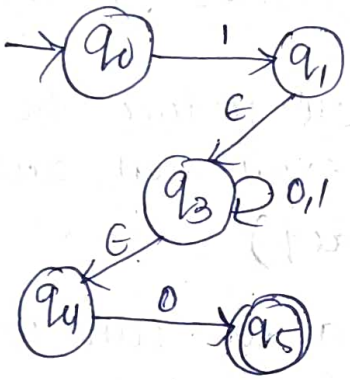
Step1: Construct a epsilon (ϵ) NFA from the regular expression.

Step2: Remove the epsilon moves and convert that to NFA.

Step3: Convert the NFA into equivalent DFA (if required).

Q) Convert the following regular expression into equivalent DFA. $1(0+1)^*0$.

Sol: Step1:



Step2:

NFA

Step3:

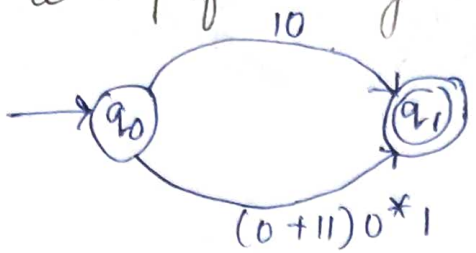
T	0	1
→ q_0	–	q_1
q_1	(q_1, q_2)	q_1
* q_2	–	–

T'	0	1
→ q_0	–	q_1
q_1	(q_1, q_2)	q_1
* (q_1, q_2)	(q_1, q_2)	q_1

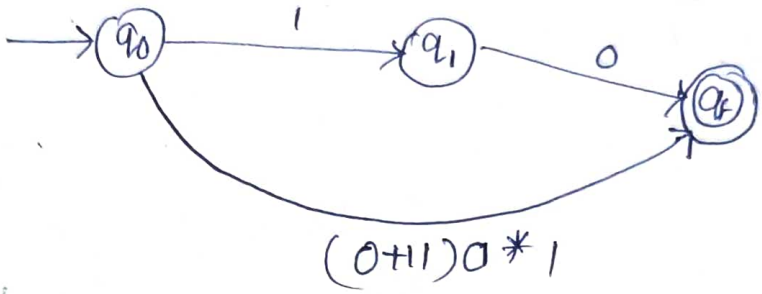
(2) Subset method

Q) construct a FA from regular expression. $10+(0+11)0^*1$

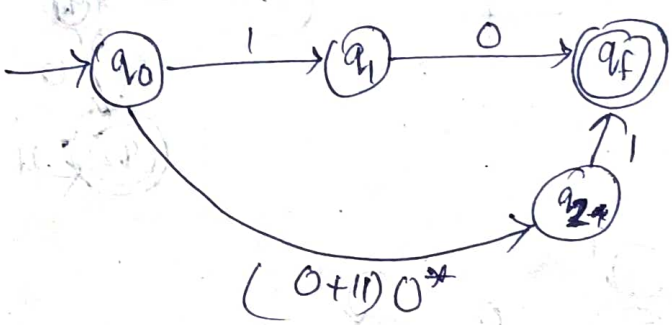
Sol^n: step 1:



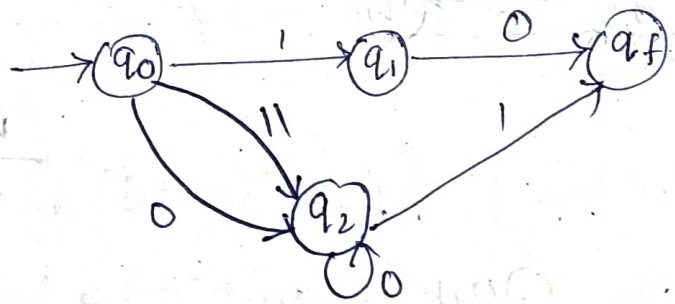
Step 2:



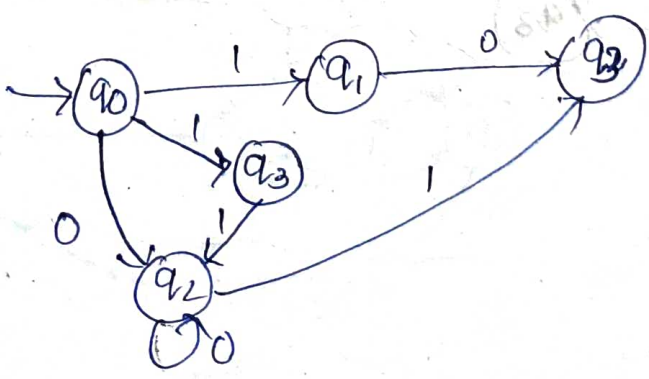
Step 3:



step 4:

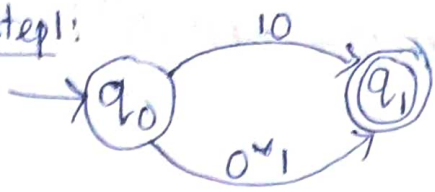


Step 5:

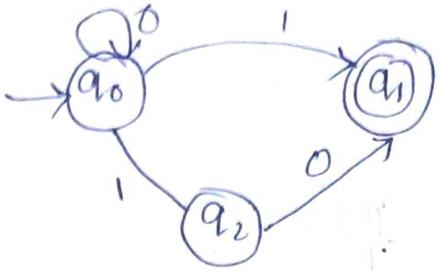


① $0^*1 + 10$

Step1:

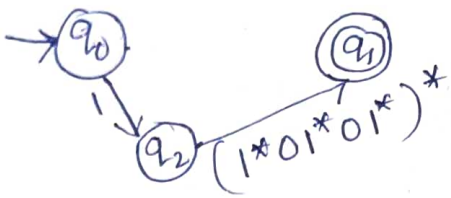


Step2:

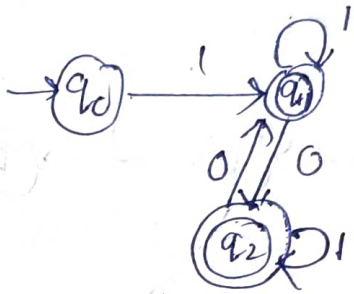


② $1(1^*01^*01^*)^*$

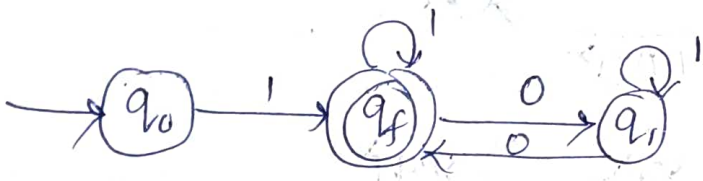
Step1:



Step2:



Step1:



Dt - 03/08/2

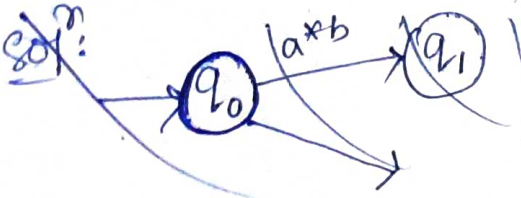
① write RE.

Soln: 0^*10^*

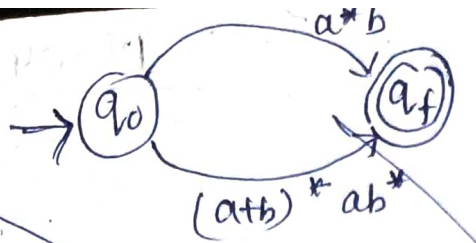
② convert RE to FA.

(i) $a^*b(a+b)^*\ ab^*$

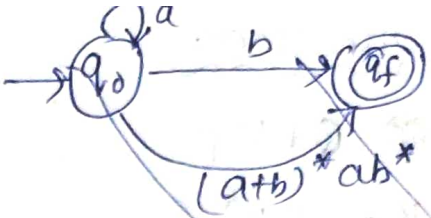
Soln:



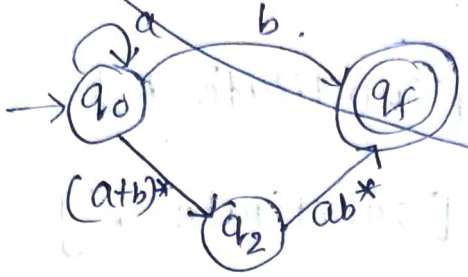
step1:



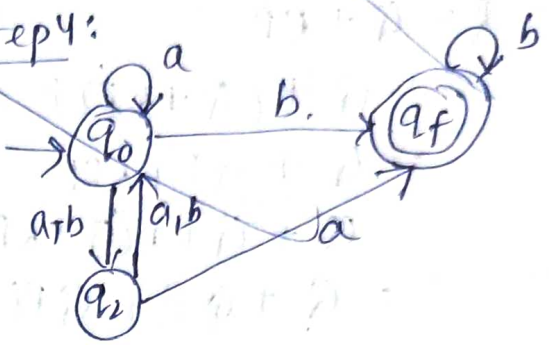
step2:



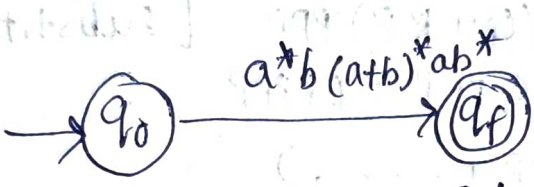
step3:



step4:

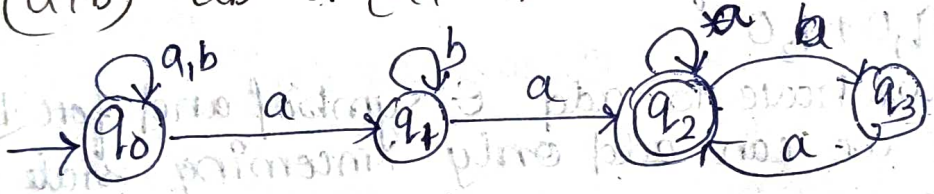


Step1:



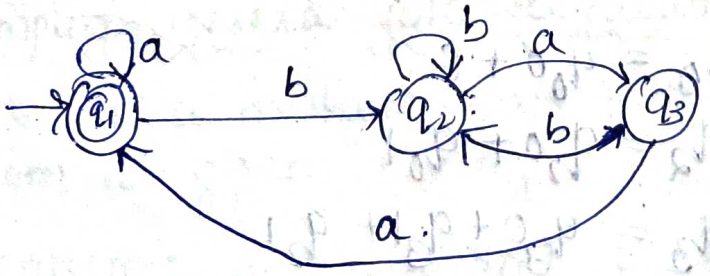
Step2:

(ii) $(a+b)^* ab^* a (a+ba)^*$



(iii) $a^* + (b(b+ab)^* aa)^*$ [$(a+b(b+ab)^* aa)^*$]

$a^* + (b(b^* + (ab)^*)aa)^*$



Arden's Theorem

$R = Q + RP$.

then, $R = QP^*$

Proof:

$R = Q + RP$

$= Q + (Q+RP)P$ [substitute R]

$= Q + QP + RPP$ [substitute R]

$= Q + QP + (Q+RP)PP$ [substitute R]

$= Q + QP + QPP + RPPP$

$= Q + QP + QPP + (Q+RP)PPP$ [substitute R]

$= Q + QP + QPP + QPPP + RPPPP$

$= Q(\epsilon + P + PP + PPP + \dots)$

$= QP^*$

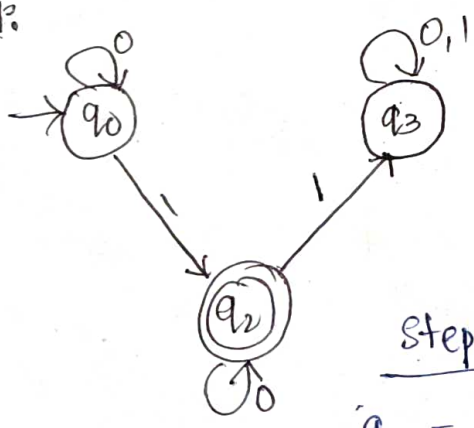
Step1: Create the equations for each and every state of the finite automata in the form

$$q_1 = q_1 0 + q_2 0$$

For initial state we have to add ϵ symbol and for the rest of the states we can add only incoming state and symbol and self loop.

Step2: Solve the equations for the final states of the finite automata.

Ex:



Step1:

$q_0 = q_0 0 + \epsilon$

$q_2 = q_2 0 + q_0 1$

$q_3 = q_3 0 + q_3 1 + q_2 1$

Step2: Now we will solve the equation.

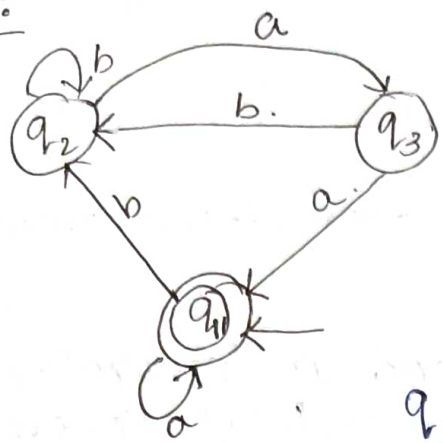
$q_0 = \epsilon 0^* = 0^*$

$q_2 = q_2 0 + q_0 1$

$= \underset{R}{q_2} \underset{P}{0} + \underset{Q}{0^* 1}$ $[R = Q + RP]$

$q_2 = 0^*10^* \quad [R = QP^*]$

Ex2:



Step1: $q_1 = \epsilon + q_3a + q_1a$

$q_2 = q_1b + q_3b + q_2b^*$

$q_3 = q_2a$

Step2: $q_3 = q_2a$

$q_2 = q_2b + q_1b + (q_2a)b$ [Replace q_3 by q_2a]

$= \underset{R}{q_2}\underset{P}{(b+ab)} + \underset{Q}{q_1b}$

$= q_1b(b+ab)^*$ [Arden's Theorem]

$q_1 = q_1a + q_3a + \epsilon$

$= q_1a + (q_2a)a + \epsilon$ [Replace q_3 by q_2a]

$= q_1a + (q_1b(b+ab)^*a)a + \epsilon$ [Replace q_2 by $q_1b(b+ab)^*$]

$= \underset{R}{q_1}\underset{P}{[a + (b(b+ab)^*aa]} + \underset{Q}{\epsilon}$

$= \epsilon(a + b(b+ab)^*aa)^*$

$= (a + b(b+ab)^*aa)^*$

Pumping lemma for regular languages

Dt - 08/08/23

- It is a method for pumping (generating) substrings from a given string.

Theorem: A language L is said to be regular if there exists an integer "p" such that $w \in L$.

- Divide the w into 3 parts namely x, y and z.

Condition: $w = xyz$

(i) $|xy| \leq p$.

(ii) $|y| \geq 1$ (or) $|y| > 0$

(iii) $\forall\ k \geq 0\ ;\ xy^k z \in L$.

Step1: let us assume the language we take as a regular language.

Step2: Divide w into x, y and z

Step3: According to our assumption the above 3 conditions hold for a regular language.

Step4: In order to prove the language is irregular we have to violate any one of the condition or every condition.

Step5: If any one of the condition does not hold then we can conclude that language as irregular language. (contradiction)

Application

1. Pumping lemma is used to show the languages that are irregular.
2. It is not used to show the languages that are regular.

Ex: prove that $L = \{ a^n b^n \mid n \geq 0 \}$ is irregular.

Sol: • Let us assume the language L is regular.

• So according to pumping lemma we can divide the string $w = xyz$.

If $n = 7$.

then $w = aaaaaaa\,bbbbbbb$.

$x = aaa$

$y = aaa$

$z = abbbbbbb$.

(i) $|xy| \leq n$ (ii) $|y| \geq 1$

$6 \leq 7$ ✓ $3 \geq 1$ ✓

(iii) $k = 2$

$xy^k z \Rightarrow xy^2 z \Rightarrow xyyz.$

$\underbrace{aaaaaaaaaa}_{10} \underbrace{bbbbbbb}_{7} \neq L$

The no. of a's is not equal to no. of b's. According to our assumption we reached (violated) condition (iii) so, the above language is irregular

Ex2: Prove that $L = \{a^n b^n \mid n \geq 0\}$ is irregular

Sol: • Let us assume the language L is regular

• By pumping lemma $w = xyz$

$x = a^f$

$y = a^g$ $[n = f+g+h]$.

$z = a^h b^n.$

(i) $|xy| \leq n$.

$|a^f . a^g| \leq n.$

$|a^{f+g}| \leq n$

$f+g \leq n$ ✓.

(ii) $|y| \geq 1$

$|a^g| \geq 1$

$g \geq 1$ ✓

(iii) if $k = 0$.

$xy^k z \Rightarrow xy^0 z \Rightarrow xz.$

$a^f a^h b^n \Rightarrow$

$a^{f+h} b^n \neq a^n b^n.$ ✗

• Hence it is irregular.

Ex: Prove the language $L = \{a^n b^{2n} \mid n \geq 0\}$ is irregular. Dt - 10/08/23

Sol: • Let us assume the language L is regular.

• By pumping lemma $w = xyz$

$x = a^f$

$y = a^g$

$z = a^h b^{2n}$

$[n = f+g+h]$.

(i) $|xy| \leq n$

$|a^f a^g| \leq n$

$f+g \leq n$. ✓

(ii) $|y| \geq 1$

$|a^g| \geq 1$

$g \geq 1$ ✓

(iii) $k = 0$

$xy^k z \Rightarrow xy^0 z \Rightarrow xz$

$a^f a^h b^{2n}$

$\Rightarrow a^{f+h} b^{2n} \neq a^n b^{2n}$ ✓

- Hence the language is irregular.

Closure properties of Regular Language

There are

1) union: If L_1 and L_2 are 2 regular languages then $L_1 \cup L_2$ will also be a regular language.

$L_1 = \{a^n / n > 0\}$

$L_2 = \{b^n / n > 0\}$

$L_1 \cup L_2 = \{a^n \cup b^n / n > 0\}$.

2) Intersection: If L_1 and L_2 are two regular languages then their intersection is also regular language.

$L_1 \cap L_2$

$L_1 = \{a^m b^n / n > 0 \ \& \ m > 0\}$

$L_2 = \{a^m b^n \cup b^n a^m / n > 0 \ \& \ m > 0\}$

$L_1 \cap L_2 = \{a^M b^n / m > 0 \ \& \ n > 0\}$.

3) concatenation: If L_1 and L_2 are regular language. their concatenation is also regular language.

$L_1 = \{a^n / n > 0\}$.

$L_2 = \{b^m / m > 0\}$.

$L_1 \cdot L_2 = \{a^n b^m / n > 0 \ \& \ m > 0\}$.

4) Kleen closure: If L_1 is a regular language then L_1^* is also regular language.

$L_1 = (a \cup b)$

$L_1^* = (a \cup b)^*$

5) complement: If L is a regular language, then L' is regular language.

$L = \{a^n \mid n > 3\}$

$L' = \{a^n \mid n \leq 3\}$.

Dt - 11/08/23

write Regular Expression for the following

1) for accepting string a

Sol^n: $RE = a$

2) That accepts a or b.

Sol^n: $RE = a + b$.

3) That accepts 0 or More occurences of a.

Sol^n: $RE = a^*$

4) Accepts 1 or More occurences of a

Sol^n: $RE = a^+$

5) String that accepts ab.

Sol^n: $RE = a.b$

6) Accepts ab at the end.

Sol^n: $RE = (a+b)^*ab$

7) Starting with a

Sol^n: $RE = a(a+b)^*$

8) starting with a and ending with ab

Sol^n: $RE = a(a+b)^*ab$.

9) Containing a substring abb.

Sol^n: $RE = (a+b)^*abb(a+b)^*$

10) Containing any number of a's followed by any number of b's followed by any number of c's.

Soln: $RE = a^* b^* c^*$

11) Containing atleast one a followed by atleast one b followed by atleast one c.

Soln: $RE = a^+ b^+ c^+$

12) Ending with a or bb

Soln: $(a+b)^* (a+bb)$

13) Even number of a's followed by even number b's

Soln: $(aa)^* (bb)^*$

14) Even number of a's followed by odd number b's

Soln: $(aa)^* (bb)^* b$.

15) Representing strings of a's and b's having length 2.

Soln: $(a+b)(a+b)$

16) Having odd length

Soln: $((a+b)(a+b))^* (a+b)$

17) Having length less than or equal to 2.

Soln: $(\epsilon + a+b)\ (\epsilon + a+b)$

18) Accepts 10^{th} symbols from the end as a

Soln: $(a+b)^* a\ (a+b)(a+b)(a+b)(a+b)(a+b)(a+b)(a+b)(a+b)(a+b)(a+b)$.

19) Accepts 3 consecutive a's.

Soln: $b^* aaa b^*$

20) Accepts atleast 3 consecutive a's.

Soln: $(a+b)^* aaa (a+b)^*$.

Dt-14/08/23

Grammar

Terminal (a, b) ↑ → Production rule (P)

$G = \{V, T, P, S\}$ → start symbol (S)

↓

(Non-Terminal) A, B

Types of Grammar

i) on basis of types of production rules.

a) Type 0 (Recursive enumerable) b) Type 1 (context sensitive) c) Type 2 (context free) d) Type 3 (Regular)

ii) on basis of no. of derivation trees

a) Ambigious b) unAmbigious

iii) on basis of no. of strings

a) Recursive b) Non-recursive.

i) on basis of types of production rules:

a) Type3 (Regular Grammar):- A grammar is said to be regular if its production rules have a single non-terminal symbol on the left hand side, a single terminal on the right hand side or a single terminal followed by a non-terminal.

Ex: $A \longrightarrow Bx$. $(A, B \in V, x \in T^*)$

$A \longrightarrow xB$.

$A \longrightarrow x$.

Types of Regular Grammar

1) Left Linear Grammar:- A grammar is said to be left linear if the productions are in the form

$A \longrightarrow Bx$ (or) $A \longrightarrow x$.

$(A, B \in V, x \in T^*)$

that is the starting symbol of the right hand side is non-terminal.

2) Right linear Grammar: A grammar is said to right linear if all productions are of the form

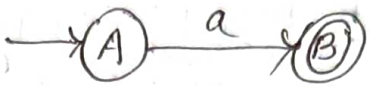
$A \to xB$ (or) $A \to x$.

$(A, B \in V \quad x \in T^*)$

Conversion of regular grammar into finite automata.

1. let us assume a production rule of the form

i) $A \to aB$.



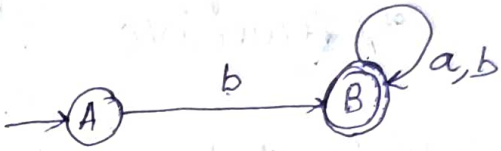
state symbol

$B \to \epsilon$.

$A \to aB \Rightarrow a \cdot \epsilon \Rightarrow a$

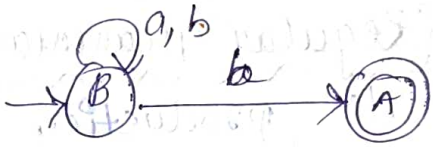
ii) $A \to bB$

$B \to \epsilon / aB / bB$.



iii) $B \to aB / bB / bA$.

$A \to \epsilon$

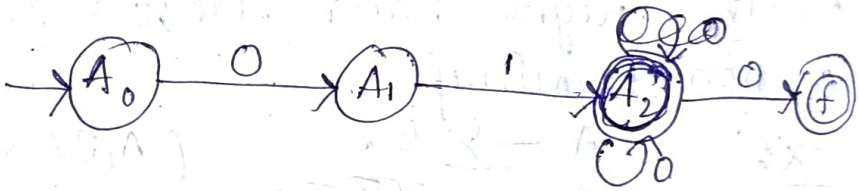


iv) $A_0 \to 0A_1$

$A_1 \to 1A_2$.

$A_2 \to 0A_2$

$A_2 \to 0$

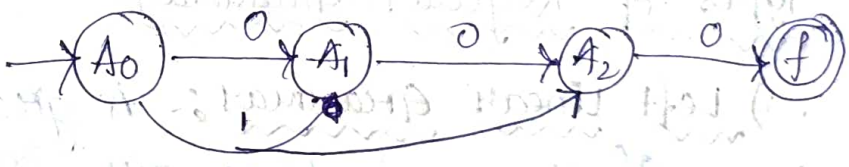


v) $A_0 \to 0A_1$

$A_0 \to 1A_2$

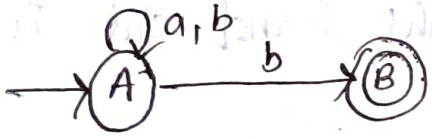
$A_1 \to 0A_2$

$A_2 \to 0$



Dt-25/08/2

Conversion of finite automata into regular grammar



$A \to aA$

$A \to bA$

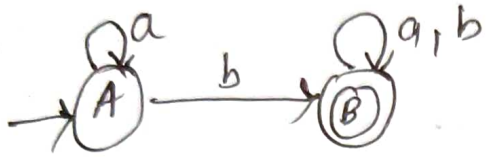
$A \to bB$

$B \to \epsilon$

$A \to aA / bA / bB$

$B \to \epsilon$.

(2)



$A \to aA$

$A \to bB$

$B \to aB$

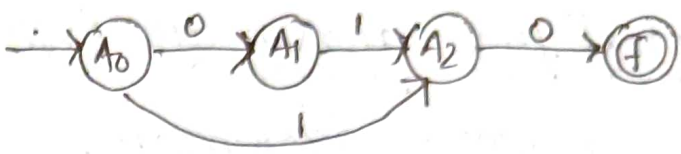
$B \to bB$

$B \to \epsilon$.

$A \to aA \mid bB$.

$B \to aB \mid bB \mid \epsilon$

(3)



$A_0 \to 0A_1$

$A_0 \to 1A_2$

$A_1 \to 1A_2$

$A_2 \to 0$

$A_0 \to 0A_1 \mid 1A_2$

$A_1 \to 1A_2$

$A_2 \to 0$.

Context free grammar (Type-2)

- It is type 2 grammar.
- It is a grammar from which all set of possible string that are generated from a formal languages.

Ex1:
$S \to aS$
$S \to bS$
$S \to \epsilon$.

(i) aa.
$S \Rightarrow aS \Rightarrow aaS \Rightarrow aa \cdot \epsilon \Rightarrow aa$

(ii) abb
$S \Rightarrow aS \Rightarrow abS \Rightarrow abbS \Rightarrow abb \cdot \epsilon$
$\Downarrow$
abb

(iii) aba.
$S \Rightarrow aS \Rightarrow abS \Rightarrow abaS \Rightarrow aba \cdot \epsilon \Rightarrow aba$

(iv) baa
$S \Rightarrow bS \Rightarrow baS \Rightarrow baaS \Rightarrow baa \cdot \epsilon \Rightarrow baa$

(v) bb
$S \Rightarrow bS \Rightarrow bbS \Rightarrow bb \cdot \epsilon \Rightarrow bb.$

(vi) abbab
$S \Rightarrow aS \Rightarrow abS \Rightarrow abbS \Rightarrow abbaS \Rightarrow abbabS \Rightarrow abbab \cdot \epsilon$
$\Downarrow$
abbab.

(vii) baaba
$S \Rightarrow bS \Rightarrow baS \Rightarrow baaS \Rightarrow baabS \Rightarrow baabaS \Rightarrow baaba \cdot \epsilon$
$\Downarrow$
baaba.

Ex2: $L = \{ W C W^R \mid \Sigma = \{a, b\}^* \}$

$P \Rightarrow \begin{cases} S \to aSa \\ S \to bSb \\ S \to C \end{cases}$

Sol:
(i) aca
$S \Rightarrow aSa \Rightarrow aca$.

(ii) abcba
$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abcba.$

(iii) bbcbb.
$S \Rightarrow bSb \Rightarrow bbSbb \Rightarrow bbcbb$

(iv) C
$S \Rightarrow C.$

(v) babacabab

$S \Rightarrow bSb \Rightarrow baSab \Rightarrow babSbab \Rightarrow babaSabab$

$\Downarrow$

babacabab.

Designing of context free grammar

Dt - 28/08/23

(1) a^*

$L = \{\epsilon, a, aa, \ldots\}$.

$S \to aS$ → self loop

$S \to \epsilon$

(2) $(a+b)^* = a^* + b^*$.

$S \to aS \mid bS$

$S \to \epsilon$

$L = \{\epsilon, a, b, aa, bb, \ldots\}$.

(3) Construct a CFG for language $L = \{ wcw^R \mid w \in (a,b)^* \}$

$L = \{ c, aca, bcb, aacaa, bbcbb, abcba, bacab, \ldots \}$

$S \to c$

$S \to aSa$

$S \to bSb$

(4) $L = \{ 0^n 1^n \mid n \geq 0 \}$.

if $n = 0$ → ϵ ~~ϕ~~

if $n = 1$ → 01

if $n = 2$ → 0011

$L = \{ \epsilon, 01, 0011, 000111, \ldots \}$

$S \to \epsilon$.

$S \to 0S1$

(5) $L = \{ 0^n 1^{2n} \mid n > 0 \}$

$L = \{ 011, 001111, 000111111, \ldots \}$.

$S \to 0S11$

$S \to 011$

$G = V, T, P, S \to S$

$S \quad 01$

(6) w is a palindrome over $(0,1)^*$

$L = \{ \epsilon, 00, 1, 00, 11, 1001, 0110, 101, 010, \ldots \}$

$S \to \epsilon$
$S \to 0S0$
$S \to 1$
$S \to 0$
$S \to 1S1$

(7) $L = \{$ any string $] w \in (0,1)^* \}$

$S \to \epsilon$
$S \to 0S$
$S \to 1S$

(8) L = Any string with even no. of 1's

$L = \{ \epsilon, 11, 1111, \ldots \}$

$S \to \epsilon$
$S \to 11S$

(9) contains 100 as substring?

$RE = (0+1)^* 100 (0+1)^*$.

if $(0+1)^* = A$.

$RE = A100A$.

$S \to A100A$
$A \to 0S \mid 1S \mid \epsilon$

(10) All strings that starts and ends with same symbol, $w \in (0,1)^*$

$L = \{ 00, 11, 010, 10 \ldots$

$S \to \epsilon$
$S \to 0S$
$P \to 1S$
$S \to 0P0$
$S \to 1P1$

$S \to \epsilon$
$S \to 0S$
$P \to 1S$

$S \to 1A1 \mid 0A0$
$A \to 0A \mid 1A \mid \epsilon$

(11) length of w is even.

$S \to \epsilon$
$S \to 0S0$
$S \to 1S1$
$S \to 0S1$
$S \to 1S0$.

Dt - 29/08/23

(12) The length of w is odd and middle symbol is 0.

$L = \{0, 000, 101, 100, 001, .\}$

$S \to 0$
$S \to 0S0$
$S \to 1S1$
$S \to 1S0$
$S \to 0S1$

(13) $L = \{0^* 1\}$

$L = \{1, 01, 001, 0001, \ldots\}$

$S \to 1$
$S \to 0S$

(14) $L = \{0^{2n} 1^{3n} \mid n \geq 0\}$

$L = \{\epsilon, 00111, 0000111111, \ldots\}$

$S \to \epsilon$
$S \to 00S111$

(15) $L = \{0^n 1^m \mid m = n, n, m \geq 0\}$

$L = \{0^n 1^n \mid n \geq 0\}$.

$S \to \epsilon$
$S \to 0S1$

(16) $L = \{1^n 0^* 1^n \mid n \geq 0\}$.

$n = 0$: 0^* $\quad$ $n = 1$: 10^*1 $\quad$ $n = 2$: 1100^*11

$P \to 0P \mid \epsilon$.
$S \to 1S1 \mid P$.

String Derivation.

- We can derive strings from a given set of production rules using 2 methods
  1) Left most Derivation (LMD) 2) Right most Derivation (RMD)
- We have to take two decision they are
  1) we have to decide the non-terminal which is to be replaced.
  2) we have to decide the production rules by which the non-terminal will be replaced.

1) Left Most Derivation: The input is scanned and replaced with the production rules from left to right.

Ex:
$E \to E + E$
$E \to E * E$.
$E \to 0 \mid 1$

$E \Rightarrow E + E \Rightarrow E * E + E$.
(0, 1, 0)

$0 * 1 + 0$.

Right Most Derivation:- The input is scanned and replaced with the production rule from right to left.

Ex: $E \to E+E$
$E \to E*E$
$E \to 0/1$

$E \to E+E \Rightarrow E+E*E$
$= 1+0*1$

LMD = RMD.

$E \to E+E \Rightarrow E+0 \Rightarrow E*E+0 \Rightarrow E*1+0$
$\Downarrow$
$0*1+0$

for derivation of any particular string we can use LMD or RMD. that gives or derives same string.

Parse Tree generation. (or) Derivation Tree Dt-31/08/

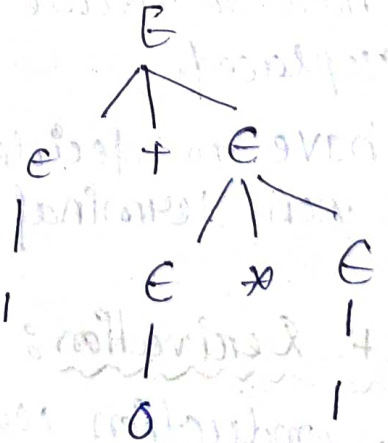
- It is a graphical representation for string derivation from a set of production rules.
- It is a tree structure.
  1) The root node represents the starting symbol of the grammar.
  2) The intermediate nodes represents non-terminals
  3) The child node represents terminals.

Ex: 1) $E \to E+E$
$E \to E*E$
$E \to 0/1$

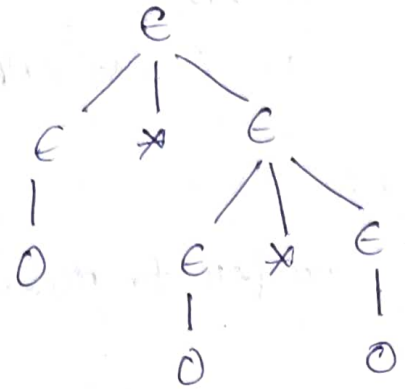
$T = \{0, 1, *, +\}$

$V = \{E\}$

String - $1+0*1$



2) $0 * 0 * 0$



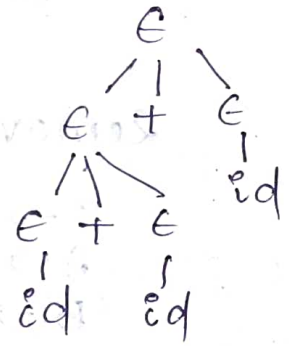
Ambiguous Grammar :- It depends upon parse tree.

- A grammar is said to ambiguous if it has more than one LMD. or more than one RMD. or more than one parse tree. for a string derivation.

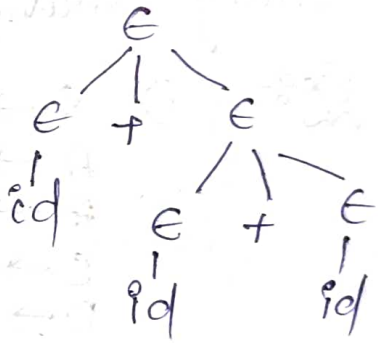
Ex:
$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow id$

$T = \{ +, *, id \}$
$V = \{ E \}$
input $\rightarrow id + id * id$

LMD



RMD.



Chomsky Normal form (CNF) :- It is a context-free grammar is said to be in CNF. if the set of production rules follows any one of the condition

1) A non-terminal deriving 2 non-terminals.
   Ex: $S \rightarrow AB$
2) A non-terminal generating a terminal.
   Ex: $A \rightarrow a$
3) The starting symbol of the grammar generates ϵ.
   Ex: $S \rightarrow \epsilon$.

Ex: 1) $S \to AB$ -
$A \to a$ -
$B \to b$ -
It is CNF

2) $A \to CD$ -
$C \to 0$ -
$D \to 0$ -
It is CNF

3) $S \to aAX$
$A \to b$ -
It is not CNF

Simplification of context free grammar

We have to check for 3 conditions in order to reduce any CFG.

1) Removing useless symbols
2) Removing null or ϵ productions.
3) Removing unique unit productions.

1) **Removing useless symbols:** A symbol is said to be useless if it does not occure on the right hand side of any production rule and also not involved in any string derivation.

Ex:- 1) $S \to ABC$
$A \to 0S \mid 1$
$B \to 0 \mid 1$
$C \to 1$
$D \to 0$

Remove $D \to 0$

$S \to ABC$
$A \to 0S \mid 1$
$B \to 0 \mid 1$
$C \to 1$

2) **Removing unit Productions:** A production rule is said to be unit production if it is in the form of a ~~non-terminal generating a terminal (or) a~~ non-terminal generating a non-terminal.

Ex: $S \to ABC$
$A \to 0S \mid B$.
$B \to 0 \mid 1$
$C \to 1$

$A \to B$ } unit production

$S \to ABC$
$A \to 0S \mid 0 \mid 1$
$B \to 0 \mid 1$
$C \to 1$

3) Removing null or ϵ productions

Ex: $S \to XYX$

$X \to 0X \mid \epsilon \Rightarrow X \to 0X \mid 0$

$Y \to 1Y \mid \epsilon \Rightarrow Y \to 1Y \mid 1$

$S \to YX \mid XY \mid XX \mid X \mid Y \mid XYX$

$X \to 0X \mid 0$

$Y \to 1Y \mid 1$

LH $X = \epsilon$

$S \to YX$

RH $X = \epsilon$

$S \to XY$

if $Y = \epsilon$

$S \to XX$

if $XY = \epsilon$ (or) $YX = \epsilon$

$S \to X$

if $XX = \epsilon$

$S \to Y$.

Q) $S \to abs \mid abA \mid abB$.

$A \to cd \mid$

$B \to aB$.

$C \to dC$.

Remove useless symbols.

Sol:

Remove $C \to dC$

$S \to abs \mid abA \mid abB$.

Remove $S \to abB$ (as it is not terminating)

$B \to aB$.

$S \to abs \mid abA$

$A \to cd$

Q) $T \to aaB \mid abA \mid aaT$

$A \to aA$

$B \to ab \mid b$

$C \to ab$.

Remove useless symbols

Remove $C \to ab$

$A \to aA$

$T \to abA$

$T \to aaB \mid aaT$

$B \to ab \mid b$

Q) $S \to ABCd$
$A \to BC$
$B \to bB \mid \lambda$
$C \to cC \mid \lambda$

Remove null moves.

$S \to ABCd$
$A \to BC, A \to B, A \to C$
$B \to bB \mid b$
$C \to cC \mid c$

if $A = \lambda$
$S \to BCd$
if $B = \lambda$
$S \to ACd$
if $C = \lambda$
$S \to ABd$.

if $AB = \lambda$
$S \to Cd$
if $BC = \lambda$
$S \to Ad$
if $AC = \lambda$
$S \to Bd$.

if A, B, C is λ.
$S \to d$.

$S \to ABCd \mid BCd \mid ACd \mid ABd \mid Cd \mid Ad \mid Bd \mid d$
$A \to BC \mid B \mid C$
$B \to bB \mid b$
$C \to cC \mid c$

Q) $S \to Aa \mid B$.
$A \to b \mid B$.
$B \to Aa$.

Remove unit production.

$S \to B$.
$A \to B$. } unit productions
$B \to A$

$A \to b \mid a$ $B \to b \mid a$

$S \to Aa \mid b \mid a$.

Q) $S \to 0A \mid 1B \mid C$
$A \to 0S \mid 00$
$B \to 1 \mid A$
$C \to 01$

$S \to C$ } unit productions
$B \to A$

$S \to 0A \mid 1B \mid 01$
$A \to 0S \mid 00$
$B \to 1 \mid 0S \mid 00$
$C \to 01$

Dt-04/09/22

Conversion of CFG into CNF.

Case 1: Eliminate start symbol S, if S exists on the right hand side any of the production rule:

- Replace the start symbol with $S_0 \to S$

Case 2: Remove null productions, useless symbols, unit productions.

Case 3: If a production rule is in the form of a non-terminal generating a terminal + non-terminal.

$A \to aB$

It can be decomposed as. $A \to XB$, if $X \to a$.

$A \to ab$ then $A \to XY$

if $X \to a$, $Y \to b$; $X \to a$, $Y \to b$.

Case 4: A non-terminal generating more than 2 non-terminals

1) $A \to XYZ$. then $A \to CZ$, $C \to XY$ if $C \to XY$

2) $B \to CDEF$. then $B \to XY$, $X \to CD$, $Y \to EF$ if $X \to CD$, $Y \to EF$

Ex 1:

$S \to a \mid aA \mid B$

$A \to aBB \mid \epsilon$.

$B \to Aa \mid b$.

Sol^n:

Case 1:

$S_0 \to S$

$S \to a \mid aA \mid B$

$A \to aBB \mid \epsilon$

$B \to Aa \mid b$

Case 2: (i) Remove null productions

$A \to \epsilon$.

$S_0 \to S$

$S \to a \mid aA \mid B$

$A \to aBB \mid \epsilon$

$B \to Aa \mid a \mid b$.

(ii) useless symbols

- no useless symbols are there

(iii) unit production.

$S \to B$ } unit production.
$S_0 \to S$ }

$S_0 \to a$

$S \to a \mid aA \mid Aa$

$S_0 \to a|aA|Aa|b.$
$S \to a|aA|Aa|ab$
$A \to aBB$
$B \to Aa|a|b$

Case3: $S_0 \to aA$ then $S_0 \to XA$ if $X \to a$
$X \to a.$

$S_0 \to aA|Aa.$
$A \to aBB$
$B \to Aa.$

So, $S_0 \to a|XA|Ax|b.$
$S \to a|XA|Ax|b.$
$A \to XBB$
$B \to Ax|a|b.$
$X \to a.$

Case 4: $S_0 \to XA|AX$
$S \to XA|AX$
$A \to XBB$
$B \to AX.$

if $C \to XB.$ then $A \to CB.$
$C \to XB.$

So, $S_0 \to a|XA|AX|b.$
$S \to a|XA|AX|b.$
$A \to CB$
$B \to Ax|a|b.$
$X \to a$
$C \to XB.$

Q1) $S \to ASB.$
$A \to aAs|a|\epsilon.$
$B \to SbS|A|bb.$

Soln: Case1: $S_0 \to S$
$S \to ASB$
$A \to aAS|a|\epsilon.$
$B \to SbS|A|bb.$

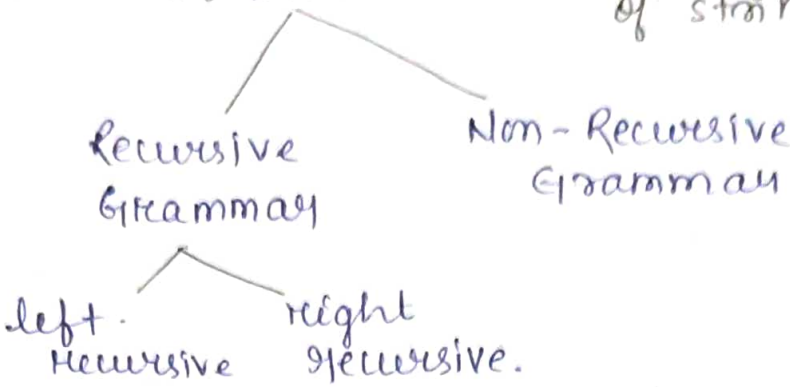
Case 2: Remove $A \to \epsilon$.

$S_0 \to S$

$S \to ASB \mid SB$

$A \to aAS$

Types of Grammar (on the basis of number of strings) 04-07/09/2



• Recursive Grammar :- A grammar is said to be recursive if it contains at least one production rule tha has same symbols on both LHS & RHS

Ex: $A \rightarrow SAB$.

Left Recursive Grammar :- A recursive grammar is said t be left recursive grammar. if the left most variable of the RHS is same as the variable in LHS.

Ex: $S \rightarrow Sa|b$.

Right Recursive Grammar :- If the rightmost most variable o the RHS is same as the variable. in LHS.

Ex: $S \rightarrow as|b$.

Non Recursive Grammar :- A grammar is said to be non-recursive if none of the production rule is havin same variable on both LHS & RHS.

Ex: $S \rightarrow AB \quad A \rightarrow a \quad B \rightarrow b$.

Elimination of left recursive.

we eliminate left recursion in oreder to avoid infinite loop formation while passing for a compiler design.

• A grammar in the form $A \rightarrow A\alpha|B$ can be written as

$A \rightarrow BA'$

$A' \rightarrow \alpha A'|\epsilon$.

Ex1: $E \to E+T \mid T$

A is E, α is +T, B is T.

$E \to TE'$

$E' \to +TE' \mid \epsilon$.

Ex2: $T \to T*F \mid F$

A is T, α is *F, B is F

$T \to FT'$

$T' \to *FT' \mid \epsilon$.

Ex3: $S \to a \mid \wedge \mid (T)$

$T \to T,S \mid S$

$T \to ST'$

$T' \to ,ST' \mid \epsilon$.

$S \to a \mid \wedge \mid (T)$

$T \to ST'$

$T \to ,ST' \mid \epsilon$.

Ex4: $A \to \underbrace{AAA}_{A\ \alpha} \mid \underbrace{a}_{B} \mid \underbrace{aB}_{B}$.

$A \to (a \mid aB)A' \to aA' \mid aBA'$

$A' \to AAA' \mid \epsilon$.

Greibach Normal form (GNF) :- A context free grammar is said to be in GNF if the production rule follows anyone of the below conditions:

1. starting symbol generating ϵ ($S \to \epsilon$)
2. A non-terminal generating a terminal ($A \to a$)
3. A non-terminal generating a terminal followed by any no. of non-terminals ($A \to aABC$)

	GNF	CNF.
(i)	$S \to \epsilon$	(i) $S \to \epsilon$.
(ii)	$A \to a$	(ii) $A \to a$
(iii)	$A \to aABC$	(iii) $A \to BC$.

Conversion of CFG to GNF.

Step1: convert the grammar into CNF. (if required)

Step2: Remove or eliminate left recursion (if required)

Step3: Convert the production rule into GNF.

Ex1: $S \to XB \mid AA$.

$A \to a \mid SA$

$B \to b$

$X \to a$.

Sol^{n}: Step1 :- The grammar is already in CNF.

Step2: There is no left recursion.

Step3: substituting the value of X in (1).

$S \rightarrow aB \mid AA$ - (1)

$A \rightarrow a \mid SA$ - (2)

$B \rightarrow b$ - (3)

$X \rightarrow a$. - (4).

Substitute the value of S in (2).

$S \rightarrow aB \mid AA$ - (1)

$A \rightarrow a \mid aBA \mid AAA$ - (2)

$B \rightarrow b$ - (3)

$X \rightarrow a$. - (4).

$\underset{A}{A} \rightarrow \underset{A}{AAA} \underset{\alpha}{A} \mid \underset{B}{a} \mid \underset{B}{aBA}$ - (Left recursion)

$A \rightarrow (a \mid aBA)\, A' \rightarrow aA' \mid aBAA'$

$A' \rightarrow AAA' \mid \epsilon$.

CYK Algorithm (Cocke - Younger - Kasami)

Dt - 21/01/2

- It is a parsing algorithm for context free grammar.
- To apply CYK the grammar needs to be in CNF.
- It is a method used to check whether the given string belongs to a particular set of production rules or grammar.

Q) $S \to AB$
$A \to BB \mid a$
$B \to AB \mid b$.
string abbb.

Sol^{n}:
- length of the string = 4
- Number the input string as

a	b	b	b
1	2	3	4

- construct a table with 4 rows 4 columns

	4	3	2	1
1	{S, B}	{B}	{S, B}	{A}
2	{S, B}	{A}	{B}	
3	{A}	{B}		
4	{B}			

- fill the table.

(1,1) → a ⇒ {A}
(2,2) → b ⇒ {B}
(3,3) → b ⇒ {B}
(4,4) → b ⇒ {B}

(1,2) → (1,1)(2,2) = (A)(B) = AB ⇒ {S, B}
(2,3) → (2,2)(3,3) = (B)(B) = BB ⇒ {A}
(3,4) → (3,3)(4,4) = (B)(B) = BB ⇒ {A}
(1,3) → (1,1)(2,3) (1,2)(3,3)
(A A) {A} ⇒ {ϕ} {S, B} {B} ⇒ {S, B} {B, B}
{ϕ} {A}
= {A}

Take union of {ϕ} and {A} = {A}.

(2,4) → (2,2)(3,4) (2,3)(4,4)
{B} {A} = BA = {ϕ} AB ⇒ {S, B}

{ϕ} ∪ {S, B} = {S, B}

(1,4) = (1,1)(2,4) | (1,2)(3,4) | (1,3)(4,4)

(A)(S,B) | (S,B)(A) | (A)(B)

= (AS)(AB). | (SA)(BA) | AB.

⇒ {S,B} | ϕ | {S,B}.

$\{S,B\} \cup \{\phi\} \cup \{S,B\} = \{S,B\}$.

- If Symbol S that is starting symbol is occuring at (1,n) where n is the length of the string, then we can say the string belongs to the grammar.

∴ The string belongs to the grammar because S is present at (1,4).

Q2) $S \to AB \mid BC$
$A \to BA \mid a$.
$B \to CC \mid b$
$C \to AB \mid a$.

String ababa.

- The length of string = 5
- Number the input string as

a	b	a	b	a
1	2	3	4	5

- construct table with 5 rows & column.

	5	4	3	2	1
1	{A,S}	{B}	{ϕ}	{S}	{A,C}
2	{B}	{S,C}	{A}	{B}	
3	{ϕ}	{S}	{A,C}		
4	{A,S}	{B}			
5	{A,C}				

- Fill the table

(1,1) = a ⇒ {A,C}
(2,2) = b ⇒ {B}
(3,3) = a ⇒ {A,C}
(4,4) = b ⇒ {B}
(5,5) = a ⇒ {A,C}

(1,2) = (1,1)(2,2) = {A,C}(B) = (AB)(CB) ⇒ {S}.

(2,3) = (2,2)(3,3) = {B}(A,C) = (BA)(BC) ⇒ {A},S

$(3,4) = (3,3)(4,4) = (A,C)(B) = (AB)(CB) \Rightarrow \{S\}$.

$(4,5) = (4,4)(5,5) = (B)(A,C) = (BA)(BC) \Rightarrow \{A,S\}$.

$(1,3) = (1,1)(2,3)$ $(1,2)(3,3)$

$(A,C)(A,S)$ $(S)(A,C)$

$(AA)(AS)(CA)(CS)$ $(SA)(SC)$

ϕ ϕ

$\{\phi\} \cup \{\phi\} = \{\phi\}$

$(2,4) = (2,2)(3,4)$ $(2,3)(4,4)$

$(B)(S)$ $(A,S)(B)$

BS. $(AB)(SB)$

ϕ $\{S, C\}$.

$\{\phi\} \cup \{S\} = \{S, C\}$

$(3,5) = (3,3)(4,5)$ $(3,4)(5,5)$

$(A,C)(A,S)$ $(S)(A,C)$

$(AA)(AS)(CA)(CS)$ $(SA)(SC)$

ϕ ϕ

$\{\phi\} \cup \{\phi\} = \{\phi\}$

$(1,4) = (1,1)(2,4)$ $(1,2)(3,4)$ $(1,3)(4,4)$.

$(A,C)(S,C)$ $(S)(S)$ $(\phi)(B)$

$(AS)(CS)(AC)(CC)$ S. B.

$\{\phi, B\}$ $\cup$ $\{\phi\}$ $\cup$ $\{\phi\} = \{\phi, B\}$.

$(2,5) = (2,2)(3,5)$ $(2,3)(4,5)$ $(2,4)(5,5)$

$(B)(\phi)$ $(A,S)(A,S)$ $(SC)(A,C)$

B (AA, AS, SA, SS) (SA, SC, CA, CC)

$\Rightarrow \{\phi\}$ $\cup$ $\{\phi\}$ $\cup$ $\{\phi, B\} = \{\phi, B\}$.

$(1,5) = (1,1)(2,5)$ $(1,2)(3,5)$ $(1,3)(4,5)$ $(1,4)(5,5)$

$(A,C)(\phi B)$ $(S)\phi$ $(\phi)(A,S)$ $\{B\}$. $\{A,C\}$

$(AB)(B)$ S $(A)(S)$ $(BA)(BC)$

ϕ ϕ ϕ $\{A,S\}$.

$\{S\}$ $\{S, C\}$.

$\{S\} \cup \{\phi\} \cup \{\phi\} \cup \{A, S\} = \{A, S, C\}$.

∴ It is accepted by grammar

D1 - 22/09

Closure properties of context free language

- context free languages can be generated by context free grammar.

1) **Union** :- If L_1 and L_2 are two context free languages their union $L_1 \cup L_2$ will also be context free.

Ex: $L_1 = \{a^n b^n c^m \mid m >= 0 \text{ \& } n >= 0\}$ &

$L_2 = \{a^n b^m c^m \mid n >= 0 \text{ \& } m >= 0\}$.

then

$L_3 = L_1 \cup L_2 = \{a^n b^n c^m \cup a^n b^m c^m \mid n >= 0, m >= 0\}$ is also context free.

2) **Concatenation** :- If L_1 & L_2 are two context free languages the $L_1 \cdot L_2$ will also be context free.

Ex: $L_1 = \{a^n b^n \mid n \geq 0\}$ & $L_2 = \{c^m d^m \mid m \geq 0\}$.

then,

$L_3 = L_1 \cdot L_2 = \{a^n b^n c^m d^m \mid n \geq 0 \text{ \& } m \geq 0\}$.

3) **Kleen Closure** :- If L is a context free language then L^* is also context free.

Ex:

4) **Pumping lemma for context free language** :- Pumping lemma is a proof for the language that is not context free.

- We break the string into 5 parts & pump the 2nd & 4th substrings.

Theorem:- If L is a context free language, there is a constant 'n' that depends exclusively on L, such that if $w \in L$ and $|w| >= n$, w can divided into 5 pieces, $w = uvxyz$, meeting the following requirements.

$\# = not$

- $|vxy| <= n$
- $|vy| \neq \epsilon$.
- for all $k >= 0$, the string $uv^k x y^k z \in L$.

Example: Find out whether $L = \{x^n y^n z^n \mid n \geq 1\}$ is context-free.

Sol^n: Let L be context free

$n = 4$

$S = xxxx\ yyyy\ zzzz$

$S = \underset{uvxyz}{abcde}$

case 1: $a = xx$, $b = xxy$, $c = yyy$, $d = z$, $e = zzz$.

$|bcd| = 7 \geq 4$.

$|bd| = 4 > 0$.

$ab^k c d^k e$ $\boxed{k=0}$

$ace = xxyyyzzz \neq xxxxyyyyzzzz$

Non-context free grammar

- Given an expression such that it is possible to obtain a center or midpoint in the strings so, that we can carryout left & right part's of the substrings using stack.

Ex1: $L = \{a^n b^n \mid n \geq 0\}$.

$n = 2$ aabb.

aa|bb.

α	b
α	b

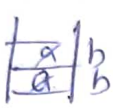
(CFG)

Ex2: $L = \{a^n b^m c^{(m+n)} \mid n \geq 1, m \geq 1\}$

$= a^n b^m c^m c^n$.

$= a^n b^m c^m c^n$. (CFG)

Ex3: $L = \{a^n b^{(2n)} \mid n \geq 0\}$.



(CFG)

Ex 4:- $L = \{a^n b^n c^n \mid n \geq 0\}$

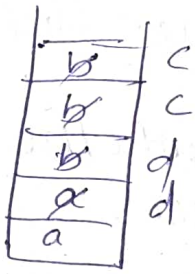
(non-CFG).

Combination of expression having mid point

$L = \{a^m b^m c^n d^n \mid n \geq 0 \quad m \geq 1\}$.

$m = 3, \; n = 2$

$a^3 b^3 c^2 d^2$

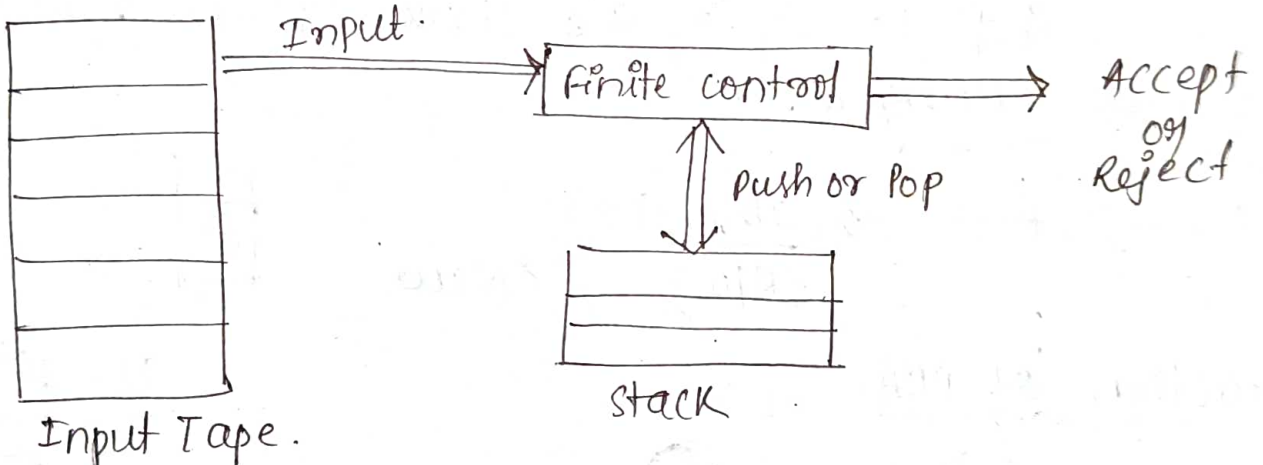


(Non-CFG)

UNIT 3

Push Down Automata :- It is a way to implement a CFG in the same way we design DFA for a regular grammar.

- A DFA can remember a finite amount of information, but a PDA can remember an infinite amount of information.
- A PDA is more powerful than FA. Any language which can be acceptable by FA can also be acceptable by PDA.
- PDA also accepts a class of language which even cannot be accepted by FA. Thus, PDA is much more superior to FA.
- There are two types of PDA :-
  (i) Deterministic PDA (ii) Non-deterministic PDA.



(Fig: Push Down Automata)

- The PDA can be defined as a collection of 7 components:-
  - Q = Finite set of states
  - Σ = Input set
  - Γ = a stack symbol which can be pushed & popped from the stack
  - q_0 = the initial state
  - Z = Start symbol which is in Γ.
  - F = a set of final states.
  - δ = mapping function which is used for moving from current state to next state.

Instantaneous Description (ID) :- ID is an informal notation of how a PDA computes an input string and makes a decision that string is accepted or rejected.

- An Instantaneous description is a triple (q, w, α) where :-

q : describes the current state
w : describes the remaining input.
α : describes the stack contents, top at the left.

Turnstile Notation :- $\vdash$ sign describes the turnstile notation and represents one move.

- $\vdash^*$ sign describes a sequence of moves.

Ex : $(p, b, T) \vdash (q, w, \alpha)$

Transition from state p to q. the input symbol 'b' is consumed and the top of the stack 'T' is represented by a new string α.

$(p, a, z) \vdash (q, abb, az)$

(R/w) → New

Construction of PDA.

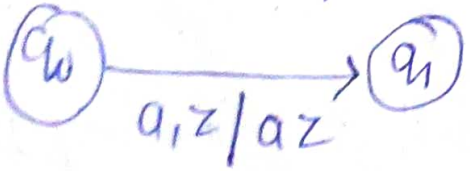
Dt - 26/09/23

1) Push :-

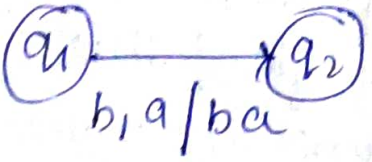
Input symbol , stack top / Input Symbol + stack top

- Let a be an input symbol and top of the stack is Z. Then the push movement can be represented in the form

input = a
top = Z.



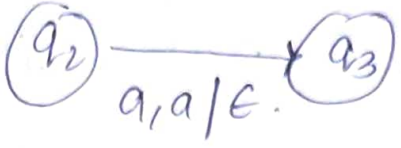
input = b
top = Z



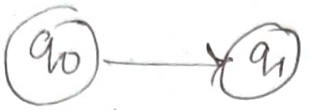
2) Pop :- 

Input symbol , stack top | ϵ.

- If top of the stack is a/b then we can represent the pop operation as

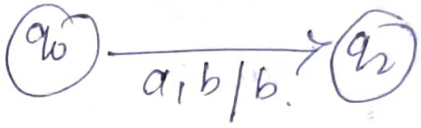


a = input symbol
a = top symbol

3) Ignore :- 

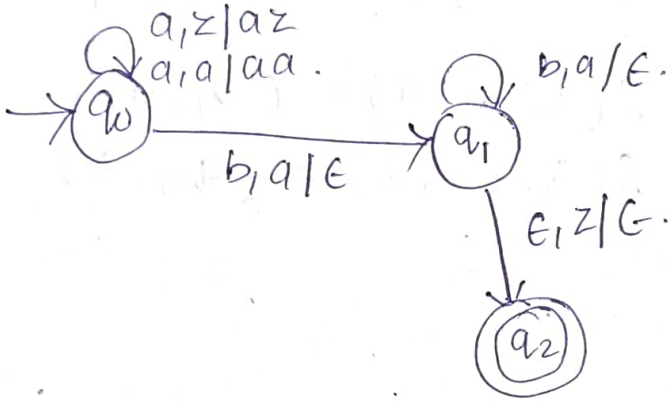
Input symbol , stack top | stack top.

- If top of stack is b and a is input symbol then we can represent the Ignore operation as

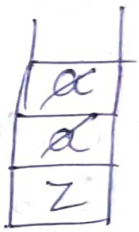


Ex: Construct a PDF for $L = \{a^n b^n \mid n \geq 1\}$.

Sol^n:



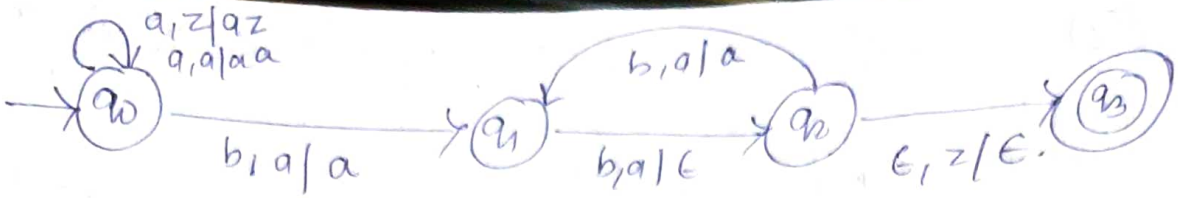
(1) a, z | az
(2) a, a | aa.
(3) a, a | aa.



$\delta(q_0, a, z) = (q_0, az)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, \epsilon)$
$\delta(q_1, b, a) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, z) = (q_2, \epsilon)$.

Ex: construct a PDA fore the language $L = \{a^n b^{2n} \mid n \geq 1\}$.

$n = 2$. $L = aabbbb$.

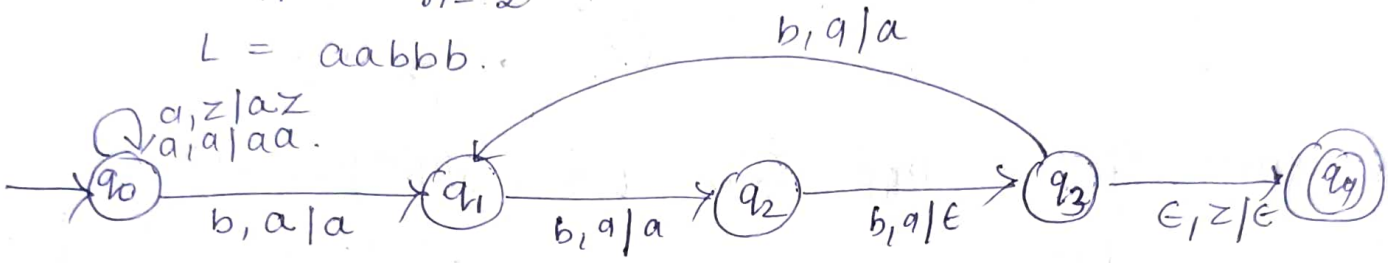


$\delta(q_0, a, z) = (q_0, az)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, a)$
$\delta(q_1, b, a) = (q_2, \epsilon)$
$\delta(q_2, b, a) = (q_1, a)$
$\delta(q_2, \epsilon, z) = (q_3, \epsilon)$

Ex: $L = \{a^n b^{3m} / \ m \geq 1, n \geq 1\}$.

$m = 1 \quad n = 2$

$L = aabbb..$



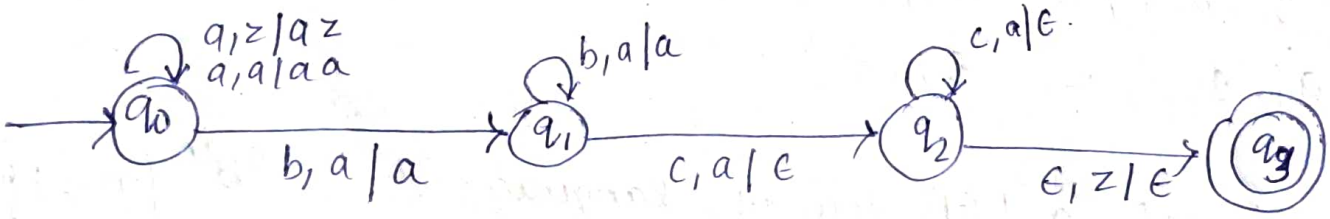
$\delta(q_0, a, z) = (q_0, az)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, a)$
$\delta(q_1, b, a) = (q_2, a)$
$\delta(q_2, b, a) = (q_3, \epsilon)$
$\delta(q_3, b, a) = (q_1, a)$
$\delta(q_3, \epsilon, z) = (q_4, \epsilon)$

Dt - 28/09/23

Ex: $L = \{a^n b^m c^n / \ n \geq 1, m \geq 1\}$.

$n = 2, \ m = 2.$

$L = aabbcc.$



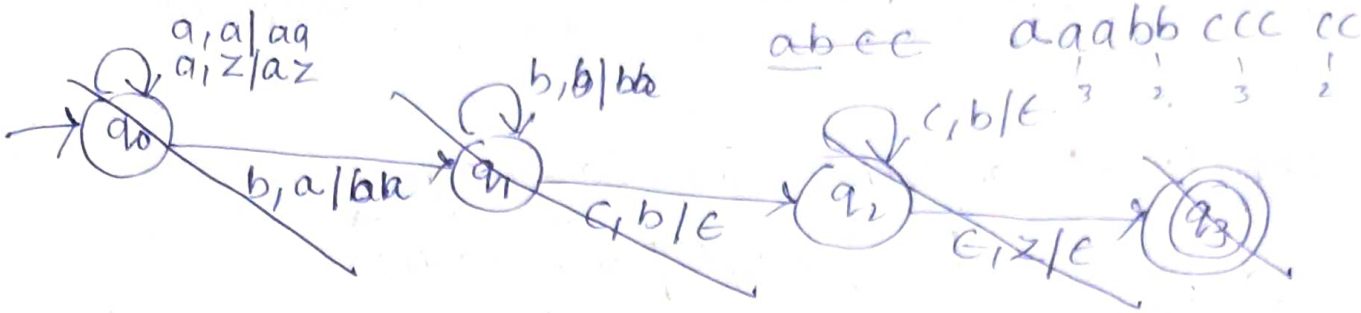
$\delta(q_0, a, z) = (q_0, az)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, a)$
$\delta(q_1, b, a) = (q_1, a)$
$\delta(q_1, c, a) = (q_2, \epsilon)$
$\delta(q_2, c, a) = (q_2, \epsilon)$
$\delta(q_2, \epsilon, z) = (q_3, \epsilon)$

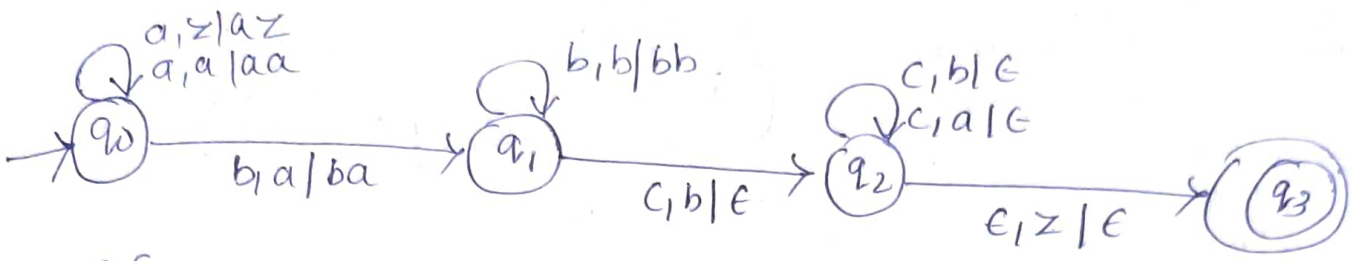
Ex: $L = \{ a^n b^m c^{(n+m)} \mid n \geq 1, m \geq 1 \}$

$a^n b^m c^n c^m$

$n = 1, m = 2$

ab cc aabb ccc cc

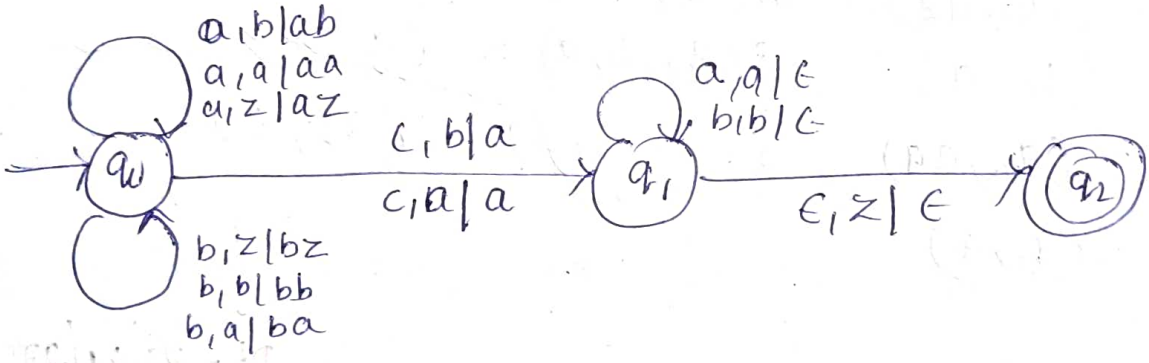




$\delta(q_0, a, z) = (q_0, az)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, ba)$
$\delta(q_1, b, b) = (q_1, bb)$
$\delta(q_1, c, b) = (q_2, \epsilon)$
$\delta(q_2, c, b) = (q_2, \epsilon)$
$\delta(q_2, c, a) = (q_2, \epsilon)$
$\delta(q_2, \epsilon, z) = (q_3, \epsilon)$

Q) Construct a PDA for the language.

$L = \{ w c w^R \mid w \in (a,b)^* \}$.



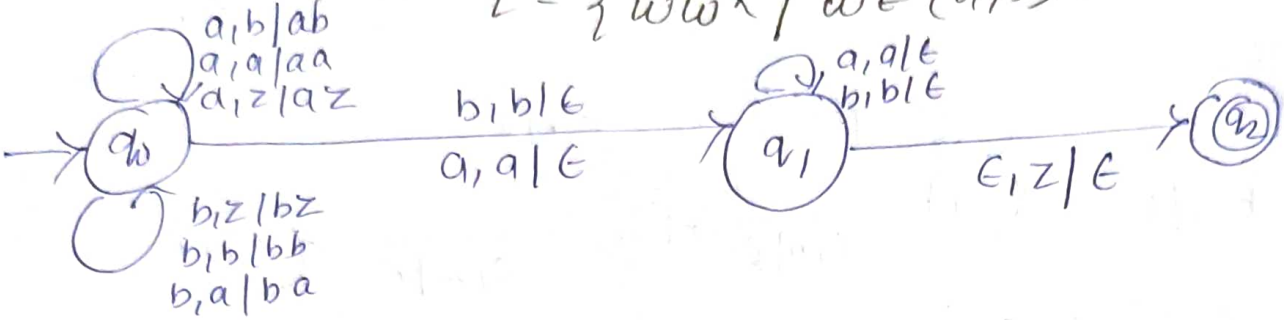
$\delta(q_0, a, z) = (q_0, az)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, a, b) = (q_0, ab)$
$\delta(q_0, b, z) = (q_0, bz)$
$\delta(q_0, b, b) = (q_0, bb)$
$\delta(q_0, b, a) = (q_0, ba)$
$\delta(q_0, c, a) = (q_1, a)$
$\delta(q_0, c, b) = (q_1, b)$
$\delta(q_1, a, a) = (q_1, \epsilon)$
$\delta(q_1, b, b) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, z) = (q_2, \epsilon)$

NPDA (Non-Deterministic Push Down Automata)

- It is similar to construction of NFA i.e. on taking a single symbol there can be multiple moments.

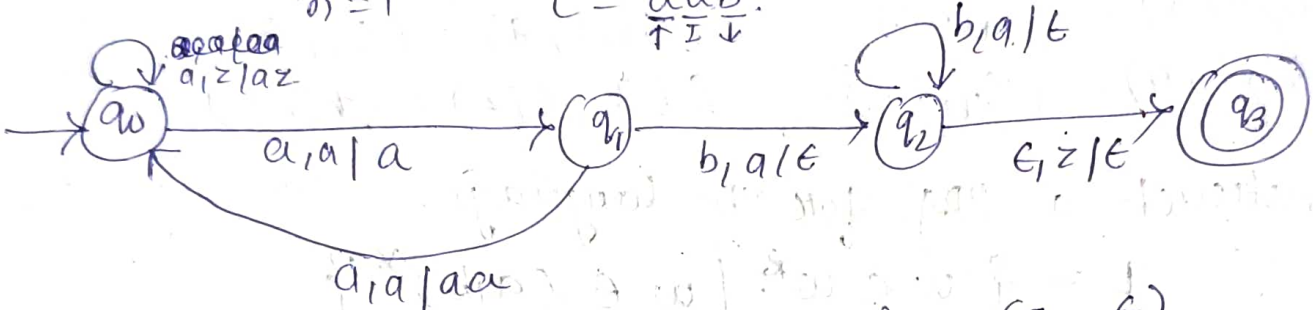
Ex: construct a NPDA for language

$$L = \{ ww^R \mid w \in (a,b)^* \}$$



Ex: $L = \{ a^{2n} b^n \mid n \geq 1 \}$

$n = 1$ $\quad L = aab$



$\delta(q_0, a, z) = (q_0, az)$
$\delta(q_0, a, a) = (q_1, a)$
$\delta(q_1, a, a) = (q_0, aa)$
$\delta(q_0, a, a) = (q_1, a)$
$\delta(q_1, b, a) = (q_2, \epsilon)$
$\delta(q_2, b, a) = (q_2, \epsilon)$
$\delta(q_2, \epsilon, z) = (q_3, \epsilon)$

CFG to PDA:

Dt - 29/09/23

- convert the CFG production rules into PDA
- There will be only one state 'q' for the PDA.
- The CFG's start symbols will be the PDA's initial symb
- Include the following rules for non-terminal system.

(i) If production rule is in the form: $A \to \alpha$, will.

represent that as $\delta(q, \epsilon, A) = (q, \alpha)$

(iii) Add the following rules for each terminal symbol, if a is a terminal symbol then we can write it as

$\delta(q, a, a) = (q, \epsilon)$

Ex: Given a CFG $S \rightarrow OBB$

$B \rightarrow OS \mid 1S \mid 0$.

Sol^n: The above production rules are in GNF, S, B are two non-terminals, so we will represent that as is'.

$\delta(q, \epsilon, S) = (q, OBB)$

$\delta(q, \epsilon, B) = \{(q, OS) \mid (q, 1S) \mid (q, 0)\}$.

- 0,1 are two terminal symbols, we will represent them as.

$\delta(q, 0, 0) = (q, \epsilon)$

$\delta(q, 1, 1) = (q, \epsilon)$.

Ex2: check whether the string 010000 is accepted by the PDA or not.

Sol^n: $\delta(q, 010000, S) \vdash \delta(q, 10000, 0BB) \vdash$

$\delta(q, 0000, BB) \vdash \delta(q, 000, SB) \vdash \delta(q, 00, BB) \vdash$

$\delta(q, 0, B) \vdash \delta(q, \epsilon)$ (Accepted)

Ex3: check whether the string 000000 is accepted by the PDA or not.

Sol^n: $\delta(q, 000000, S) \vdash \delta(q, 00000, BB) \vdash \delta(q, 0000, BBB) \vdash$

$\delta(q, 000, SB) \vdash \delta(q, 00, BB) \vdash \delta(q, 0, B) \vdash$

$\delta(q, \epsilon)$ (Accepted).

(Ex4 last page)

Turing Machine.

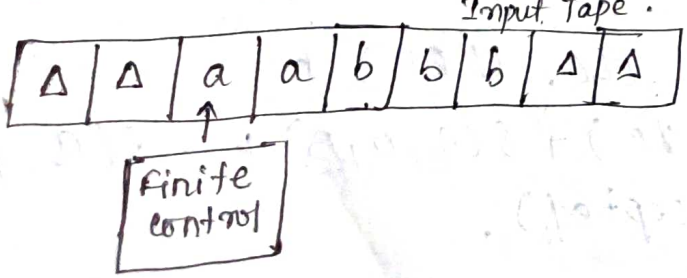
Dt- 02/10/

- It is invented by Alan Turing in 1936.
- It accepts Recursive Enumerable languages (Type - 0).
- It is a mathematical model of a simple abstract computer.
- It is used to study the properties of algorithms to determine the problems that can & cannot solved by a comput
- It helps in analysis of time and space complexity.

Functions of Turing Machine.

1) It accepts or rejects a string in the form of a long tape.
2) It can read, write and erase symbols from the long tap
3) The turing machine can only read one symbol at a time and it uses set of rules for the next state movement.
4) If it reaches the final state then the computation is successful or else unsuccessful.
5) The tape consists of some symbols known as blank sym that are used for the process of computation.
6) It consists of a header point that can move either side of the input tape.

Components of Turing Machine.



1) Input Tape :

2) finite control

In the above diagram Δ is the blank symbols that can be inserted at the starting and ending of the input tape (sometimes blank symbols can also be inserted only at the end of the input tape).

Formal Definition of Turing Machine

- It is a 7 tuple represented as :

$$TM = (Q, T, B, \Sigma, \delta, q_0, F)$$

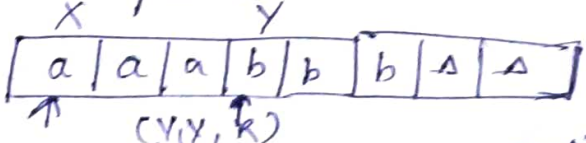
Q : set of finite states
T : Tape alphabet.
B : blank symbol
Σ :
δ : Transition function
q_0 : initial state
F : Final state.

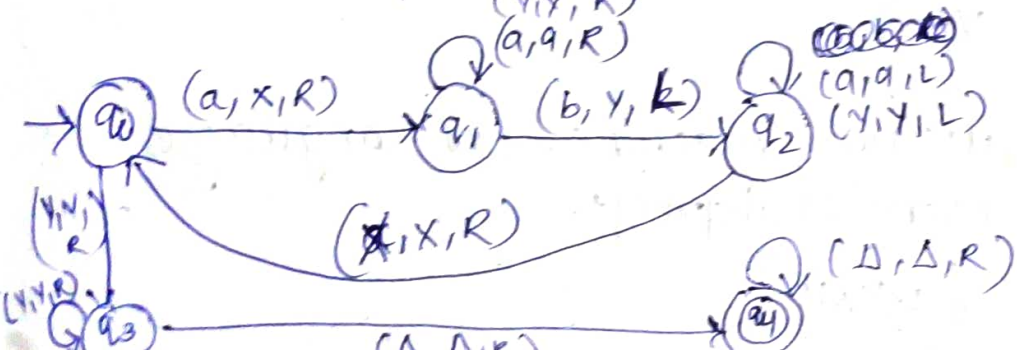
- Transition function is represented as :

$$Q \times T \longrightarrow Q \times T \times \{L, R\}$$

Q) construct a turing machine $L = \{a^n b^n \mid n \geq 0\}$

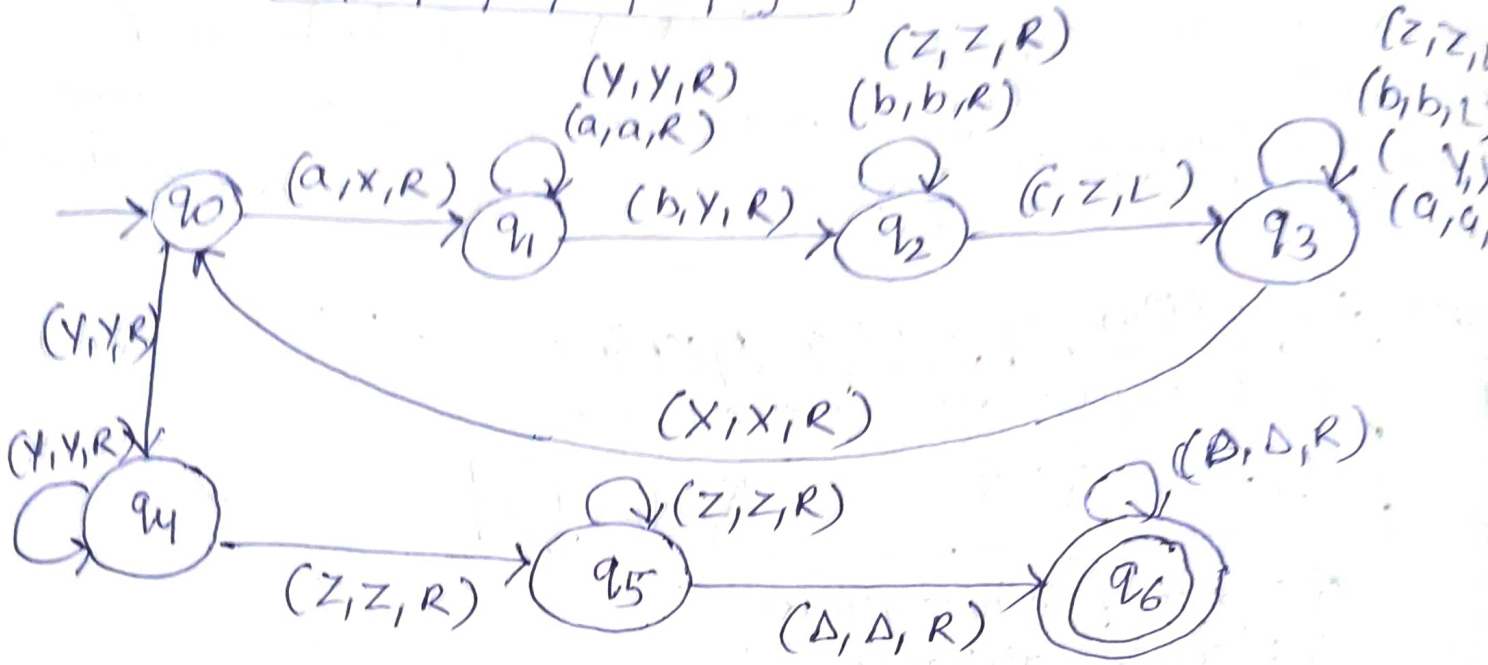
- let the input tape be





Exa: $L = \{a^n b^n c^n\}$.

a	a	b	b	c	c	Δ	Δ



$\delta(q_0, a) = (q_1, X, R)$
$\delta(q_1, a) = (q_1, a, R)$
$\delta(q_1, b) = (q_2, Y, R)$
$\delta(q_2, b) = (q_2, b, R)$
$\delta(q_2, c) = (q_2, Z, L)$
$\delta(q_3, b) = (q_3, b, L)$
$\delta(q_3, Y) = (q_3, Y, L)$
$\delta(q_3, a) = (q_3, a, L)$
$\delta(q_3, X) = (q_0, X, R)$
$\delta(q_0, a) = (q_0, X, R)$
$\delta(q_1, Y) = (q_1, Y, R)$

$\delta(q_2, Z) = (q_2, Z, R)$
$\delta(q_3, Z) = (q_3, Z, L)$
$\delta(q_0, Y) = (q_4, Y, R)$
$\delta(q_4, Y) = (q_4, Y, R)$
$\delta(q_4, Z) = (q_5, Z, R)$
$\delta(q_5, Z) = (q_5, Z, R)$
$\delta(q_5, \Delta) = (q_6, \Delta, R)$

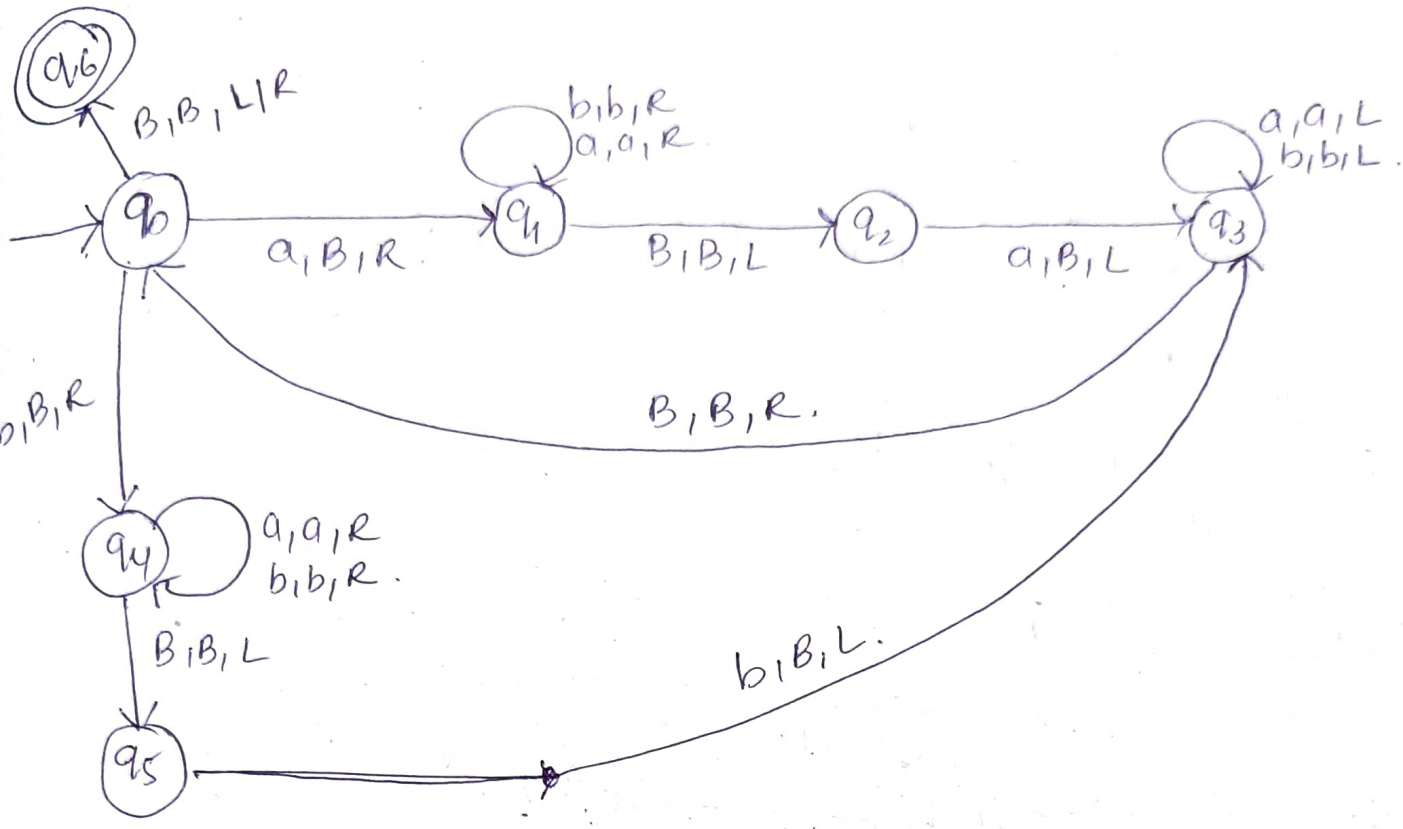
Q) Construct a Turing machine for language that accepts even palindrome.

$L = \{WW^R\}$.

Sol^n :

a	b	a	a	b	a	B

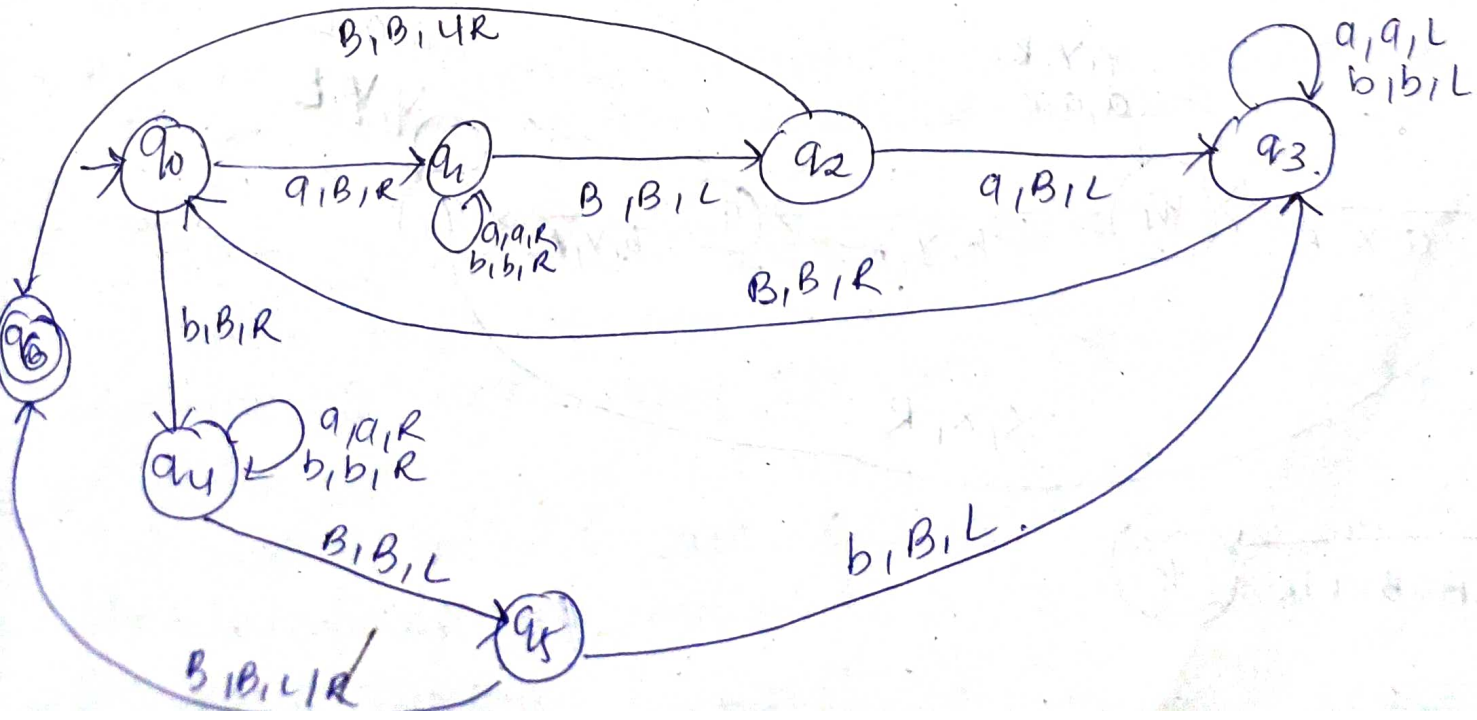
01-05/10/23



Q) Design a Turing machine for language having odd length palindrome.

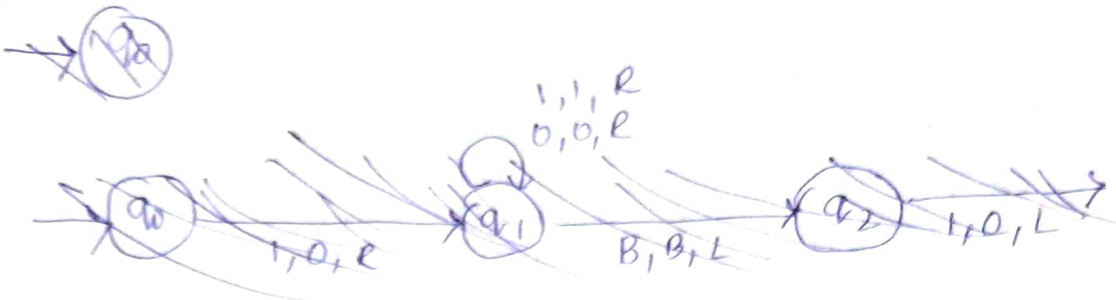
$$L = \{ wbw^R \}.$$

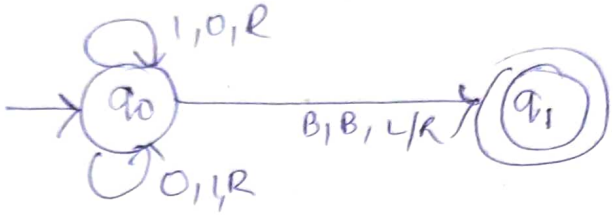
a	b	a	b	b	b	a	b	a	B.



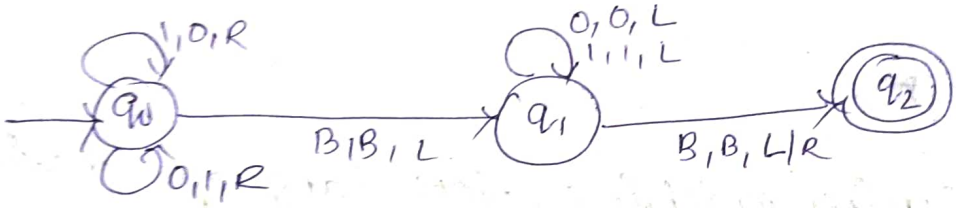
Q) construct a turing machine for is

1	0	1	B





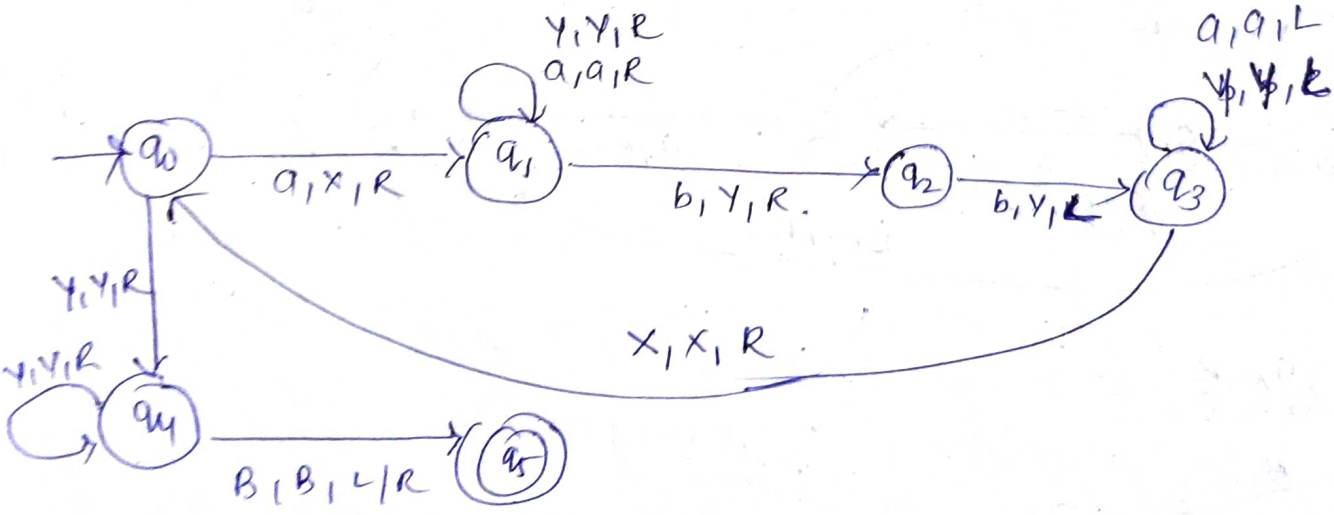
B	1	0	1	B



Q) construct a Turing machine for language.

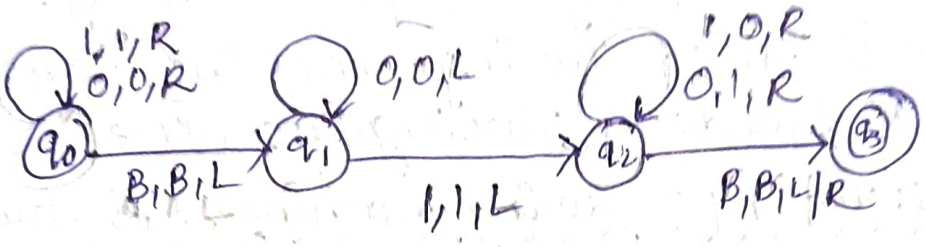
$L = \{a^M b^{2M}\}$.

X	a	a	a	Y b	Y b	b	b	b	b	B



Dt - 06/10/23

8) construct Turing machine for language 0's complement. $L = 0^*$



Dt - 09/10/23

Variants of TM

1. Multiple Track Turing Machine: A K-track Turing machine (for some $K>0$) has K-tracks and one R/W head that reads and writes all of them one by one.
   - A K-track Turing Machine can be simulated by a single track Turing machine.
2. Two-way infinite Tape Turing Machine:
   - Infinite tape of two-way infinite tape Turing machine is unbounded in both directions left and right.
   - Two-way infinite tape Turing machine can be simulated by one-way infinite Turing machine (Standard Turing machine).

tape Turing
and is controlled by a single head.
- The multi-tape Turing machine is different from K-track Turing machine but expressive power is th same.
- Multi-tape Turing machine can be simulated by sin tape Turing machine.

4. Multi-tape Multi-head Turing Machine: - The multi-tape Turing machine has multiple tapes an multiple heads.
- Each tape is controlled by a separate head.
- Multi-Tape Multi-head Turing machine can be simulated by a standard Turing machine.

5. Multi-dimensional Tape Turing machine: It has mult dimensional tape, where the head can move in any direction that is left, right, up or down.
- Multi dimensional tape Turing machine can be simulat by one-dimensional Turing machine.

6. Multi-head Turing Machine: A multi-head Turing machine contains two or more heads to read the symbols on the same tape.
- In one step all the heads sense the scanned symbols and move or write independently.
- Multi-head Turing machine can be simulated by a single head Turing machine.

7. Non-deterministic Turing machine: A non-determini Turing machine has a single, one-way infinite Tape
- for a given state and input symbol has at least one choice to move (finite number of choices for the next move), each choice has several choices for the path that it might follow for a given input string.

- A non-deterministic Turing machine is equivalent to the deterministic Turing machine.

Recursively enumerable language

- A recursively enumerable language is a formal language for which there exists a Turing machine (or other computable function) that will halt and accept when presented with any string in the language as input but may either halt and reject or loop forever when presented with a string not in the language. contrast this to recursive languages, which require that the Turing machine halts in all cases.

Dt – 10/10/23

Undecideable Problem

- A problem is said to undecideable if we can not an algorithm for getting the answers.
- In FLAT undecideable Problems means we can not construct a Turing machine. or the Turing machine will enter into infinite loop.

Post correspondance Problem as undecideable

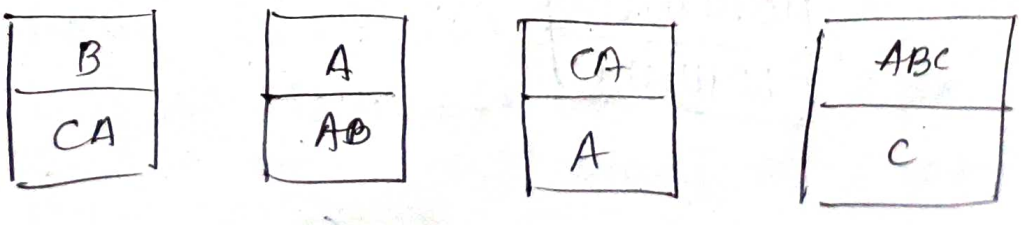
- According to Post correspondance problem they consist 2 strings A, B where

$$A = Y_1 Y_2 Y_3 \ldots Y_n$$
$$B = X_1 X_2 X_3 \ldots X_n.$$

such that $Y_1 Y_2 Y_3 \ldots Y_n \neq X_1 X_2 X_3 \ldots X_n$.

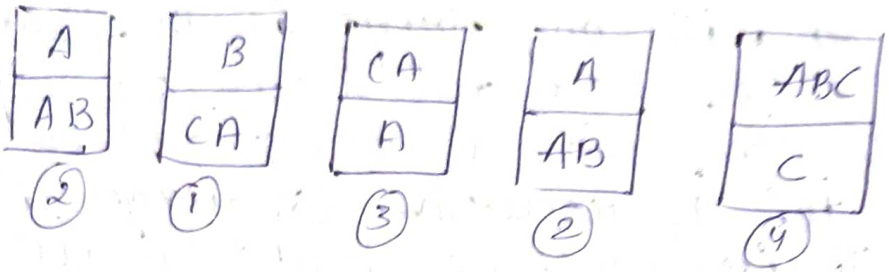
- There are n number of dominos (tiles), the task is to arrange the dominos in such a order that the numerators and denominators of all the tiles should be equal.

Case 1: Dominos



Problem: Arrange the Dominos in such a order that the string formed by the numerator is equal to the string formed by the denominator (Dominos can be repeated)

Soln: Start with a domino that is having a symbol same in both numerator and denominator at initial point.



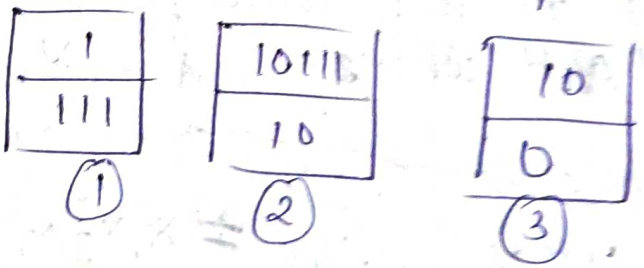
ABCAAABC
ABCAAABC

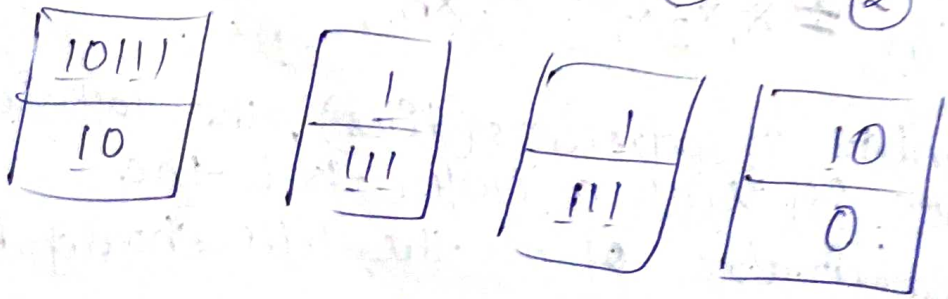
Note: The above problem is a case of decideability.

Case 2: Table Form.

	A	B
1	1	111
2	10111	10
3	10	0

- Consider the column A as numerator and B as denominator.
- The table can be represented as





10111110
10111110

This is decideable.

Q)

	A	B
1	10	101
2	011	11
3	101	011

check whether the given table is decideable or not?

10
101

011
11

101
011

10
101

(1)

101
011

(3)

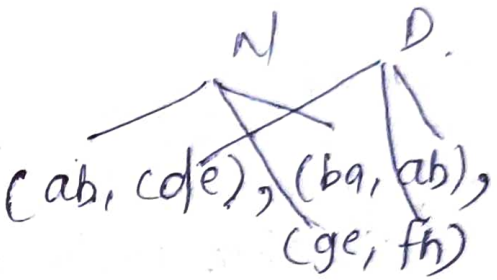
101
011

(3)

101
011

(3)

It is undecideable.



10 10 — 1010
101 101 — 101101 X

10 01 — 1001
101 11 — 10111 X

X = ab, ac, ca → N
Y = ac, b, ab → D.

Dt - 12|10|23

Linear Bounded Automata.

- It is similar to Turing machine. with the below properties:
  1) Turing machine with Non-deterministic logic
  2) Turing machine with Multi-Track.
  3) Turing machine with a bounded finite length of tape.

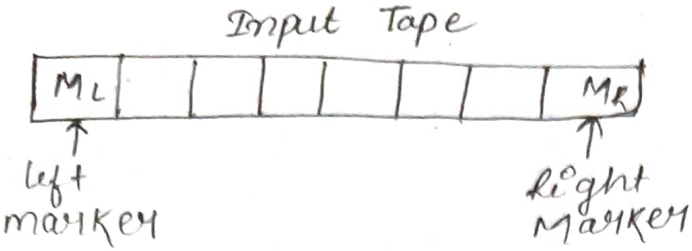
formal Definition of LBA

LBA can be defined as. 8 tuples.

$$M = \{Q, T, \epsilon, q_0, M_L, M_R, S, F\}.$$

Q: Finite set of states. q_0: Initial state.
T: Tape alphabet. M_L: Left bound of tape.
ϵ: Input alphabet. M_R: Right bound of tape.

δ: Transition function.
F: final state.



Ex: $L = \{a^{n!} \mid n \geq 0\}$.

$L = \{w^n \mid w \in (a,b)^+, n >= 1\}$.

$L = \{wwww^R \mid w \in (a,b)^+\}$.

Context Sensitive Grammar (Type 1)

- A grammar is said to be context sensitive if the production rules are in the form.

$\alpha \rightarrow \beta$. where $\alpha, \beta \in (V \cup T)^*$

Ex: $AA \rightarrow aB$

$aA \rightarrow bB$

$a \rightarrow b$.

- The language generated by context sensitive grammar is context sensitive language.

$S \rightarrow abc \mid aAbc$

$Ab \rightarrow bA$.

$Ac \rightarrow Bbcc$

$bB \rightarrow Bb$.

$aB \rightarrow aa \mid aaA$.

$L = \{a^n b^n c^n \mid n > 0\}$.

$S \Rightarrow aAbc \Rightarrow abAc \Rightarrow abBbcc \Rightarrow aBbbcc \Rightarrow aabbcc$

$aaAbbcc$.

$aabAbcc \Rightarrow aabbAcc$

$\Rightarrow aabbBbccc$

$\Rightarrow aabBbbccc$

$\Rightarrow aaBbbbccc$

$\Rightarrow aaabbbccc$

Unit 4: Primitive Recursive function

Dt - 7/11/23

Total recursive Function:- A recursive function is said to be total recursive if it is defined for its all arguments.

- Let 'f' be a function where if stores $a_1, a_2, \ldots, a_n$ is defined on a function 'g' $(x_1, x_2, \ldots, x_n)$. Every element of f is assigned to unique element of g.

Primitive Recursive function:- A function is said to be primitive recursive if and only if it is an initial function over n, or it is obtained by applying composition or recursion with finite number of the initial function over n.

μ- Recursive function:- It is partial function that takes finite number of tuples and returns a single natural number.

- They are the smallest class of partial functions that includes the initial function and is closed under composition, primitive recursion and the μ-operator.

Ackermann's Function:- It is a function with 2 arguments each of which can be assigned any non-integer value

- All primitive recursive functions are total and computable, but the Ackermann's function illustrates that not all total computable functions are primitive recursive.
- Ackermann's function is defined as

$$A(m,n) = \begin{cases} n+1 & \text{if } m=0 \text{ \& } n>0. \\ A(m-1,1) & \text{if } m>0 \text{ \& } n=0. \\ A(m-1, A(m,n-1)) & \text{if } m,n>0. \end{cases}$$

Q) Calculate A(1,1)

Soln: $A(1,1) = A(0, A(1,0)) = A(1,0)+1$

$= A(0,1)+1 = 1+1+1$

$= 3$

$A(1,2) = A(0, A(1,1)) = A(1,1)+1$

$= A(0, A(1,0))+1 = A(1,0)+1+1$

$= A(0,1)+1+1 = 1+1+1+1 = 4.$

Q) ~~$A(2,1)$~~ $= A(1, A(2,0)) = A(0, A(1, A(2,0)-1))$

$= A(1, A(2,0)-1)+1$

$= A(0, A(1, A(2,0)-1-1))+1$

$= A(1, A(2,0)-1-1))+1.$

Q) $A(2,1) = A(1, A(2,0))$

So, $A(2,0) = A(1,1) =$ ~~$A(0, A(1,0)) = A($~~ 3.

$A(1,3) = A(0, A(1,2)) = A(0,4) = 4+1 = 5.$

Q) $A(2,2) = A(2-1, A(2,2-1)) = A(1, A(2,1))$

$= A(1-0, A(1, A(2,1)-1)) = A(1, A(1,4))$

$= A(1, A(1,4)) = A(1,6).$

$= A(1-1, A(1,6-1)) = A(0, A(1,5)).$

$= A(1,5)+1 = A(0, A(1,4))$

$=$ ~~$A(1,4)+1$~~ $A(0,6) = 6+1 = 7$

Q) $A(1,4) = A(1-1, A(1,4-1)) = A(0, A(1,3)) = A(1,3)$

$= 6.$

Q) $A(1,3) = A(1-1, A(1,3-1)) = A(0, A(1,2))$

$= A(0,4) = 5.$

Cantor & Godel numbering

Dt - 09/11/23

Q) Define cantor & Godel numbering and find the value for the sequence 10110?

- Godel numbering is represented as encoding in which a number is assigned to each and every symbol of a mathematical notation, after which a sequence of symbols are being represented with the help of natural numbers.
- To encode an entire formula which is a sequence of symbols, Godel uses the following system.
  - * If a sequence is given as ~~10110~~ $a_1, a_2, a_3 \ldots a_n$. that can be represented as power of prime numbers starting from 0.

$$a_1, a_2, a_3 \ldots a_n = 2^{a_1} \times 3^{a_2} \times 5^{a_3} \times \ldots p^{a_n}$$

$$10110 = 2^1 \times 3^0 \times 5^1 \times 7^1 \times 11^0$$

$$= 70.$$

Ex1: If x is represented as 5, = is represented as 6 and y is represented as 3. then the expression $x = y = x$ has the value ___.

Sol^n:

$$x = y = x$$

$$5\ 6\ 3\ 6\ 5 = 2^5 \times 3^6 \times 5^3 \times 7^6 \times 11^5$$

$$= 5.525 \times 10^{16}.$$

Church - Turing Thesis

Dt - 20/11/23

- Any mechanical computation can be performed by a turing machine.
- There is a TM corresponding to every computable problem.
- we can model any mechanical computer with a TM.
- The set of languages that can be decided by a TM is identical to the set of languages decided by a mechanical computer.

- If there is no TM that decides problem P, ... algorithm for problem P.

P class

- P stands for Polynomial time.
- It is the collection of decision problems (answers as yes or no) that can be solved by deterministic machine, in polynomial time.

Features :- 1) The solution of P problems is easy to find.
2) P is often a class of computational problems that are solvable and tractable.

Examples :- 1) Calculating the Greatest common Divisor
2) Finding a maximum matchup
3) Merge sort.

Unit-1

short:

1) NFA

- Non-DFA
- It accepts ϵ moves
- It requires less space
- It is not used in digital computers

DFA

- Deterministic FA
- It doesn't accept ϵ moves.
- It requires more space
- It is used in digital computers.

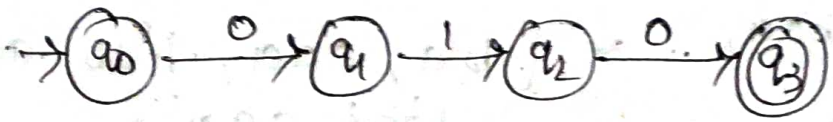
2) Regular expression :-

3) R^* is not equivalent to R^+.
R^* contains ϵ but R^+ not
R^* is Kleen closure R^+ is positive closure.

4) NFA can stimulate the characteristics of DFA. By restricting the transition in NFA such that at most one transition for each input.

5) FA (Finite Automata)

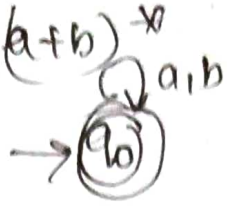
- These are used for discovering patterns
- It accepts set of symbols as i/p and transition from one state to another.
- It has 2 states accept and reject state.



6) Convert FA to RE :-

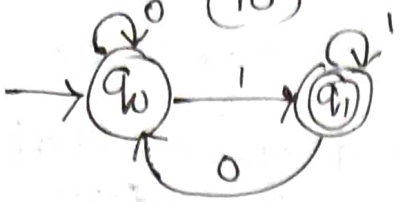
7)

$(a+b)^*$

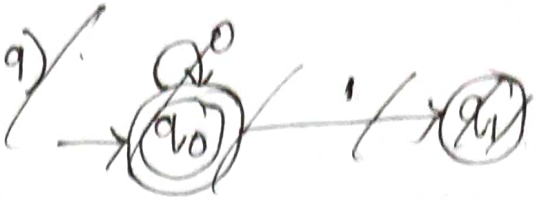


8) 10, 1100, 111000.

$(10)^*$



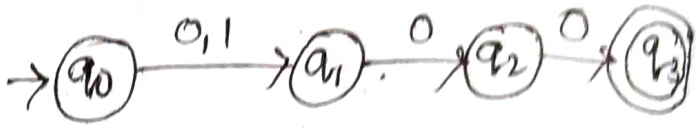
9)



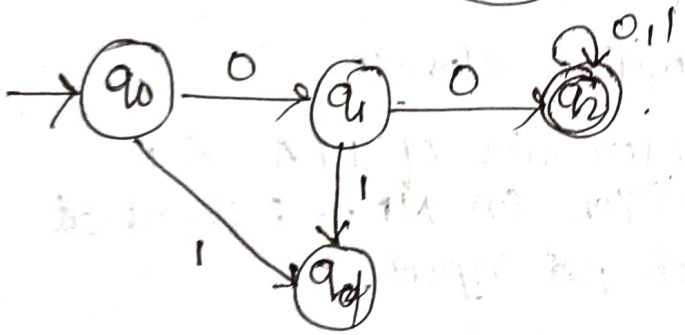
9)

	0	1
q_0	q_0	q_1
q_1	q_1	$\{q_0, q_1\}$

10) $(0+1)00$.



11) Start with 00 DFA



12) It can one or more than one final state

13) NFA with ε closure is similar to NFA.

14) $L_1 = \{a, ab, a^2\}$ $L_2 = \{b^2, aa\}$.

$L_1 \cdot L_2$

$= \{ab^2, aaa, abb^2, abaa, a^2b^2, a^2aa\}$

$L_2 \cdot L_1$

$= \{b^2a, aaa, b^2ab, aaab, b^2a^2, aaa^2\}$.

15) Difference

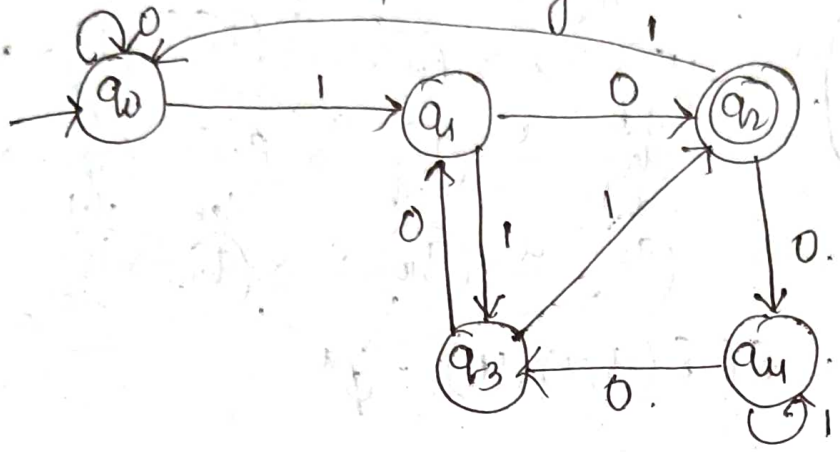
Kleen Closure	Positive.
1. It is represented as a^*	• It is represented as a^+
2. It includes empty string	• It doesn't include empty strings

3. It is concatenation of all possible of strings from L

3. It is includes all possible concatenations except one.

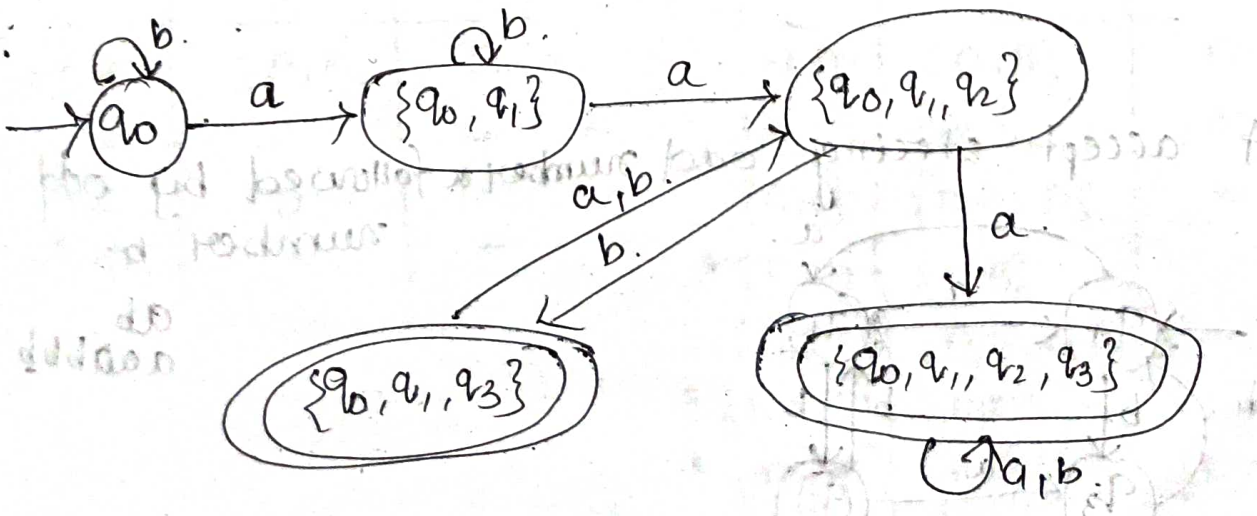
long

1) DFA accepts decimal string modulo 5 = 0.

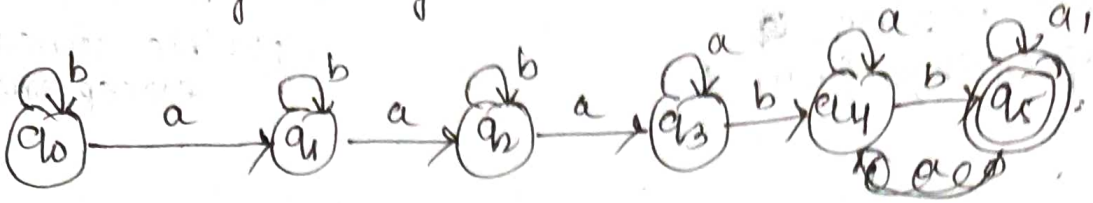


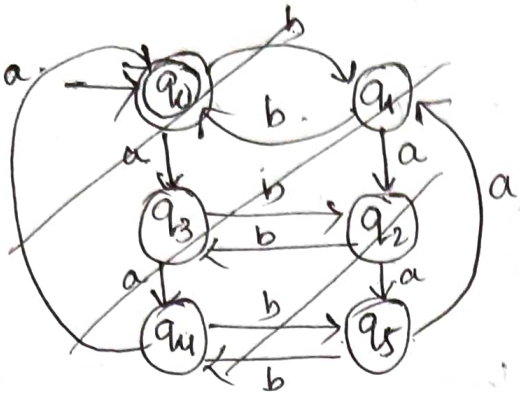
2)

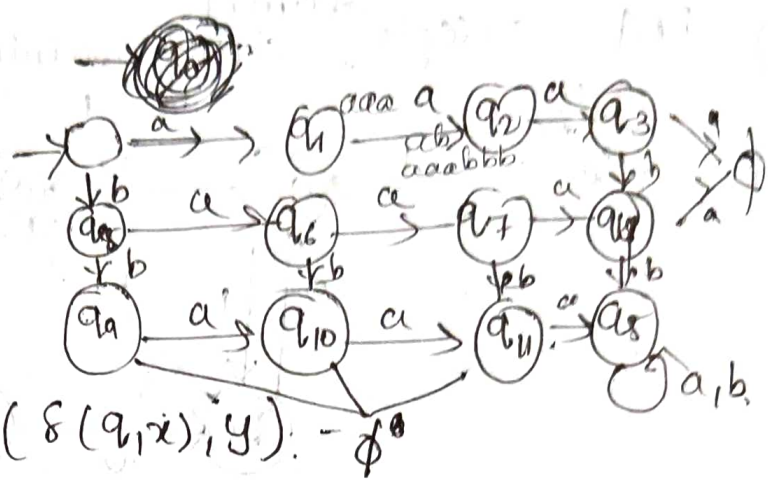
T'	a	b
→q0	{q0, q1}	q0
{q0, q1}	{q0, q1, q2}	{q0, q1}
{q0, q1, q2}	{q0, q1, q2, q3}	{q0, q1, q3}
{q0, q1, q3}	{q0, q1, q2}	{q0, q1, q2}
{q0, q1, q2, q3}	{q0, q1, q2, q3}	{q0, q1, q3, q2}



3) DFA string having at least 3a & 2b.







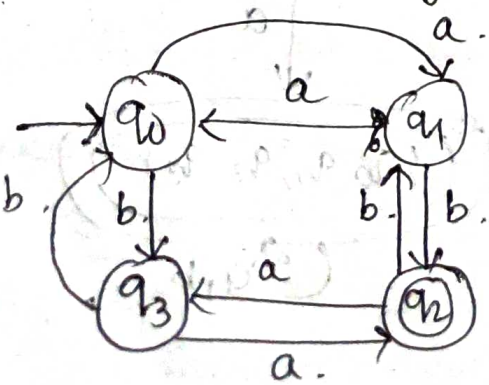
4) Prove $\delta(q_1, xy) = \delta(\delta(q_1, x), y)$

x - String
y - Symbol

Property of Transition function:

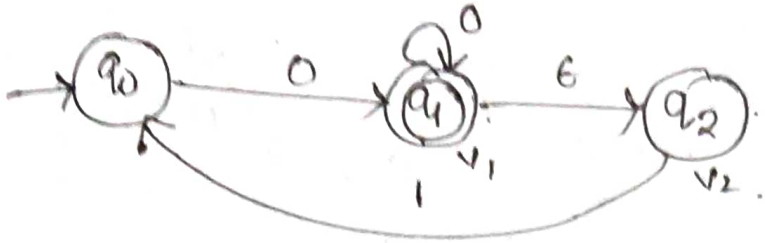
$$\delta(q_1, wa) = \delta(\delta(q_1, w), a)$$

5) DFA accept string odd number a followed by odd number b.



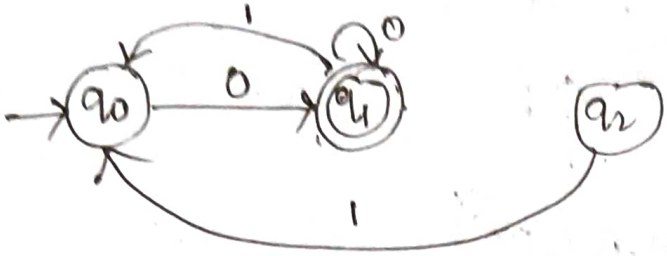
ab
aaabbb

6)

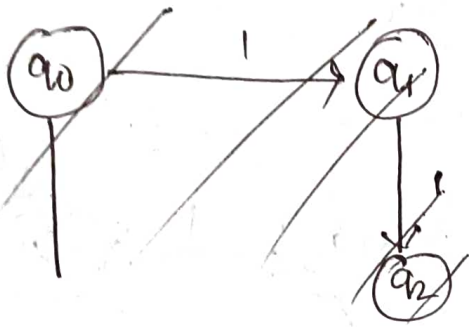


step 1: $\delta(q_2, 1) = q_0$.

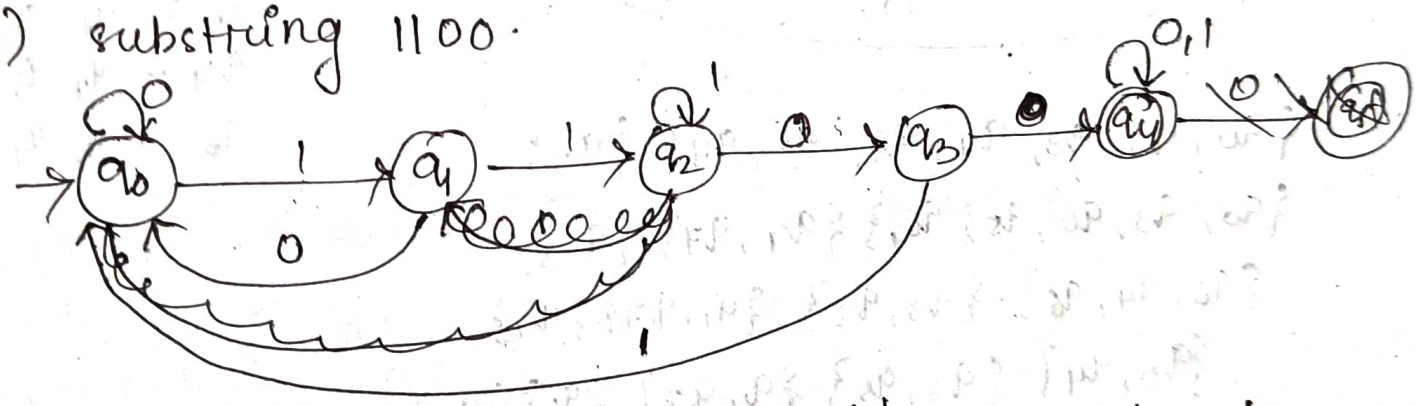
step 2:



7) ~~substring~~



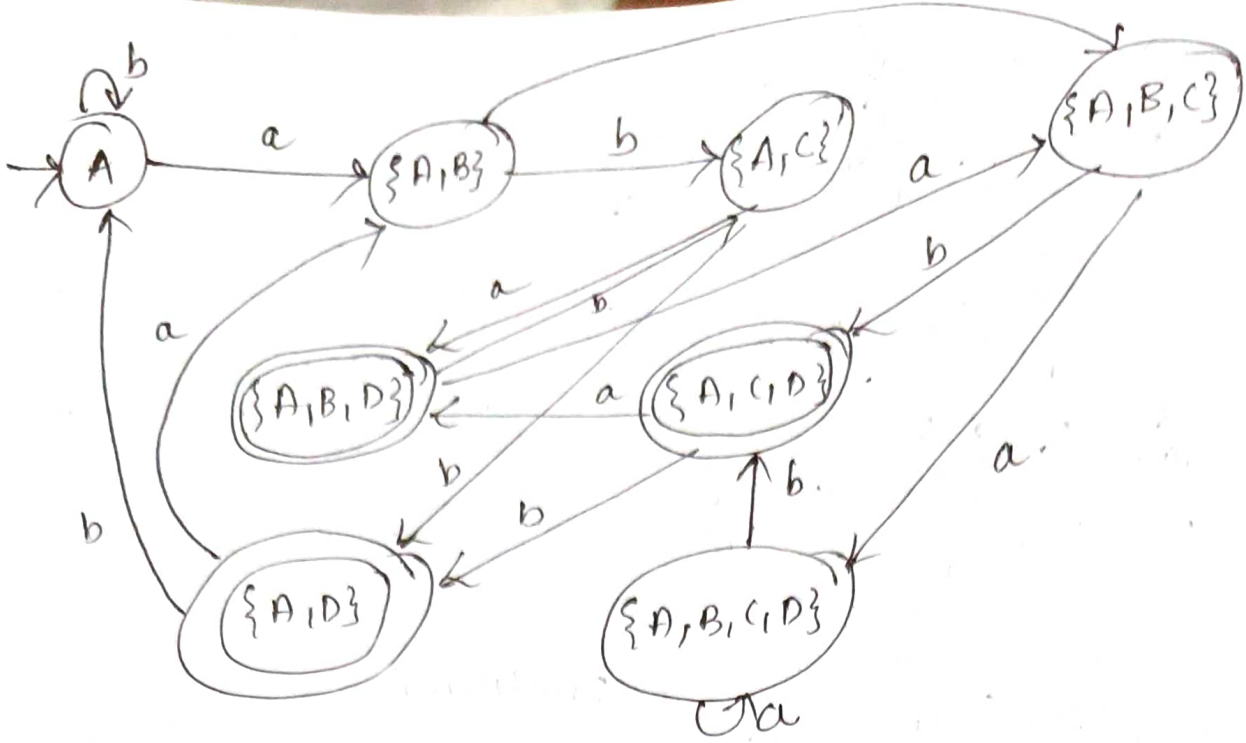
substring 1100.



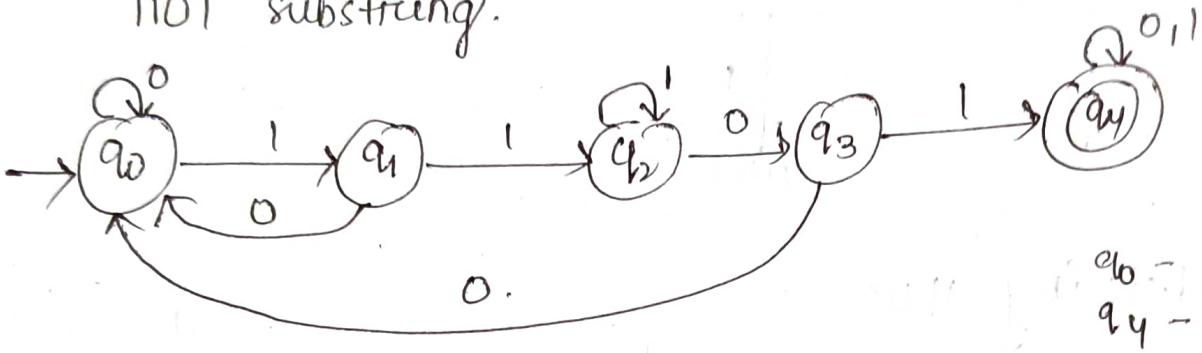
8)

T	a	b
→ A	{A, B}	A
B	C	C
C	D	D
* D	–	–

T'	a	b
→ A	{A, B}	A
{A, B}	{A, B, C}	{A, C}
{A, C}	{A, B, D}	{A, D}
* {A, D}	{A, B}	{A}
{A, B, C}	{A, B, C, D}	{A, E, D}
* {A, B, D}	{A, B, C}	{A, E}
* {A, C, D}	{A, B, D}	{A, D}
* {A, B, C, D}	{A, B, C, D}	{A, C, D}



9) 1101 substring.



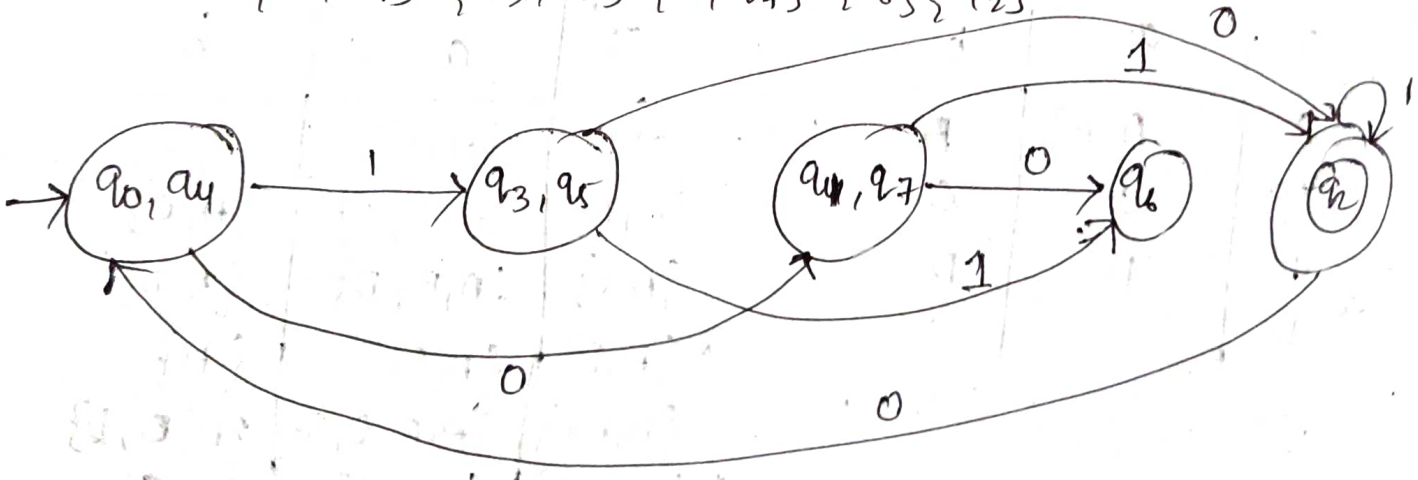
$q_0 - q_1\ q_5$
$q_4 - q_7\ q_5$
$q_6 - q_6\ q_4$

10) $\{q_0, q_1, q_3, q_4, q_5, q_6, q_7\}\ \{q_2\}$

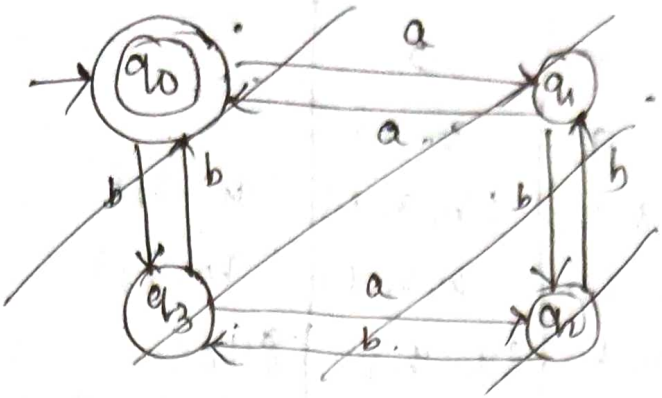
$\{q_0, q_3, q_4, q_5, q_6\}\ \{q_1, q_7\}\ \{q_2\}$

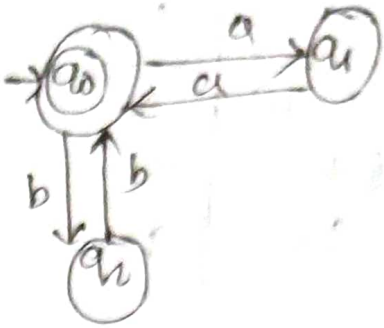
$\{q_0, q_4, q_6\}\ \{q_3, q_5\}\ \{q_1, q_7\}\ \{q_2\}$

$\{q_0, q_4\}\ \{q_3, q_5\}\ \{q_1, q_7\}\ \{q_6\}\ \{q_2\}$



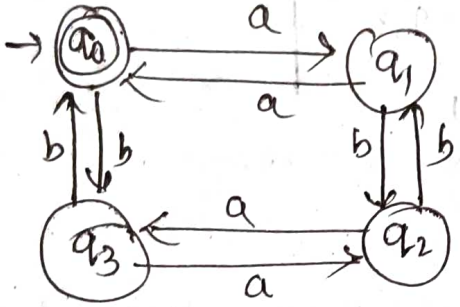
11) DFA - string even a followed even b.





12) $\{q_0, q_1, q_2, q_3, q_4, q_5\}$ $\{q_6\}$.

13)



14)

T	C	d
(q_1, q_4)	(q_1, q_4)	(q_2, q_5)
(q_2, q_5)	(q_3, q_6)	(q_1, q_4)
(q_3, q_6)	(q_2, q_7)	(q_3, q_6)
(q_2, q_7)	(q_3, q_6)	(q_1, q_4)

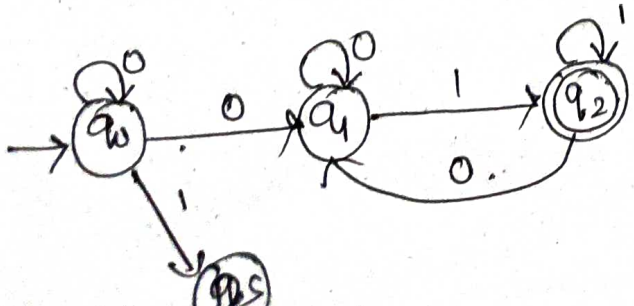
Equivalent.

15) - same 13.

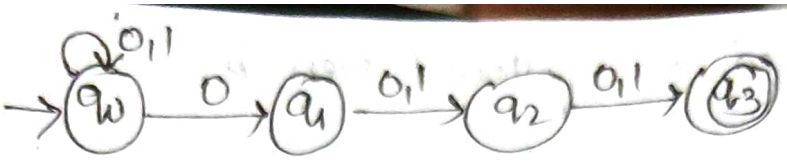
16) Steps to minimize FA.

Step1: Remove inaccessible states.

17)



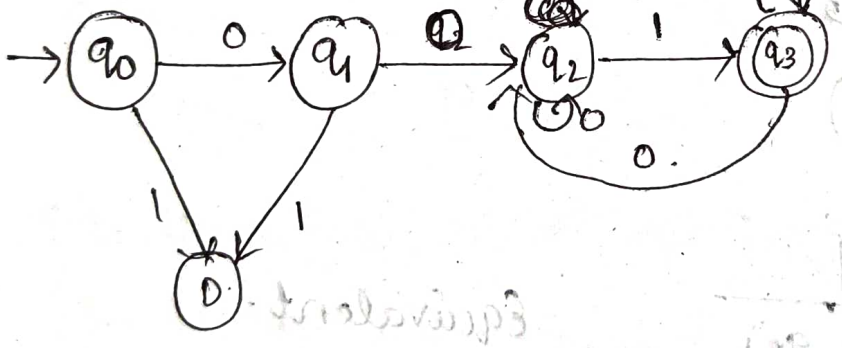
18)



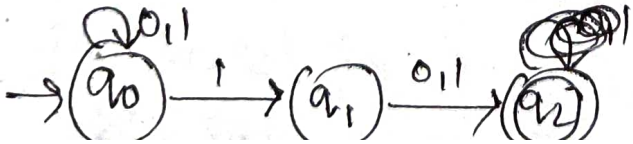
T	0	1
→ q_0	$\{q_0, q_1\}$	q_0
q_1	q_2	q_2
q_2	q_3	q_3
* q_3	–	–

T'	0	1
→ q_0	$\{q_0, q_1\}$	q_0
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0, q_3\}$
$\{q_0, q_3\}$	$\{q_0, q_1\}$	q_0
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_2, q_3\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$
$\{q_0, q_2, q_3\}$	$\{q_0, q_1, q_3\}$	$\{q_0, q_2\}$
$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_2, q_3\}$

19)



20)



Unit - 2

short

1) CFG :- It is a grammar used to derive set of strings from given rules formal language.
Type - 2.

a^N $S \to \epsilon$
$S \to aS$.

2) Derivation - It is process of deriving or generating string based on production rule.

Derivation Tree - It is graphical representation of string derivation from set of production rules.

Sentential form - It is string of symbols can be derived from start symbol by P.R.

Right Most Derivation :- Input is scanned and replacing production rule from right to left

LMD.

3) Ambiguous Grammar :- A grammar is ambiguous if it has more than RMD or LMD or one parse tree.

4) RE = $(a+b)^* aabb (a+b)^*$.

5) RE = $(a+b)^* bb$.

6) Pumping lemma - to prove it is regular grammar

7) Regular language

context Free language

8)

9) FA for $(a+b)^*$.

10) Regular language. ex

11) context sensitive language. ex

12) $RE = 0\,(0+1)^*1$

13) $RE = (0+1)^*11$

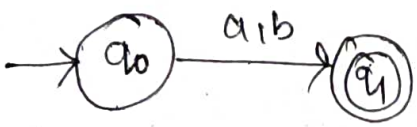
14) Left recursion grammar – A grammar where the left most symbol in right side is same as left symbol

2e

$$A \rightarrow \underbrace{AA}_{A\,\alpha} \mid \underbrace{aB}_{\beta}.$$

$$A \rightarrow BA'$$
$$A \rightarrow \alpha A' \mid \epsilon.$$

15) FA for (a+b)



long

1) CFG to GNF.

$S \to AY \mid XX$
$X \to x \mid SX$
$Y \to y$
$A \to x$.

In CNF.
No Left recursive.

Replace S in S.

$S \to AY \mid XX$
$X \to x \mid AYX \mid XXX$.
$Y \to y$
$A \to x$.

$X \to \underbrace{X}_{A}\underbrace{XX}_{\alpha} \mid \underbrace{AYX}_{B} \mid x$

$X \to (AYX \mid x)X'$
$X' \to XXX'$

1) $S \to AY \mid XX$
$X \to x \mid SX$
$Y \to y$
$A \to x$

A in S.

$S \to xY \mid XX$
$X \to x \mid SX$
$Y \to y$
$A \to x$

S in X

$S \to xY \mid XX$
$X \to x \mid xXX \mid xYX$
$Y \to y$
$A \to x$

$X \to \underbrace{X}_{A}\underbrace{XX}_{\alpha} \mid \underbrace{x \mid xYX}_{B}$.

$X \to (X \mid xYX)X'$
$X' \to XXX' \mid \epsilon$.

$S \to xY \mid XX$
$X \to xX' \mid xYXX' \mid$
$X' \to XXX' \mid \epsilon$.
$Y \to y$
$A \to x$.

Remove ϵ.

$S \to xY \mid XX$.
$X \to xX' \mid x \mid xYXX' \mid xYX$
$X' \to XXX' \mid XX$
$Y \to y$
$A \to x$.

X in S ⊢ x'

$S \to xy \mid xx'x \mid xx \mid xyxx'x \mid xyxx$

$X \to xx' \mid x \mid xyxx' \mid xyx$

$X' \to xx'xx' \mid xxx' \mid xyxx'xx' \mid xyxx' \mid xx'x \mid xx \mid xyxx'x \mid xyxx$

$Y \to y$

$A \to x$.

2) baaba (5)

Cocke - Younger Kasis.

It needs to be in CNF.

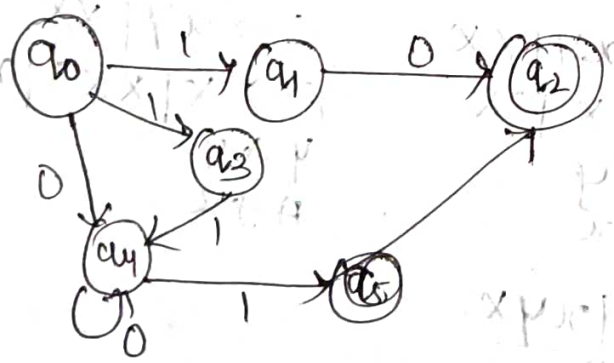
3) Arden's Theorem

State -

$R = Q + RP$

$R = QP^*$.

4) $10 + (0+11)0^*1$



5)

$q_1 = q_10 + \epsilon \cdot = q_{00} = \epsilon^*$.

$q = q_11 + q_0$

$q_3 \to q_1 + q_30 + q_31$.

$q = q_11 + q0$

$= \underset{Q}{0^*1} \cdot + \underset{R}{q_0} \underset{P}{0}$.

0^*10^*.

6) $S \to AAA|B$
$A \to aA|B$
$B \to \epsilon$.

Step1: $S_0 \to S$

Step2: null.

$S_0 \to S$
$S \to AAA|BAA|A|B$.
$A \to aA|B|a$.

useless symbol.
$S \to AAA|AA$.

$S_0 \to S$
$S \to AAA|B|\epsilon$.
$A \to aA|B|\epsilon$.
~~$B \to \epsilon$~~.

useless symbol.
B is useless.

$S \to AAA|AA|A$
$A \to aA|a$.

$A \to \epsilon$.
$S_0 \to S$
$S \to AAA|B|AA|A$.
$A \to aA|a|B$.

unit production.
$S \to A$.
$S \to AAA|AA|aA|a$.
$A \to aA|a$.

Step3:
$S_0 \to AAA|AA|XA|a$.
$S \to AAA|AA|XA|a$
$A \to XA|a$.
$X \to a$

Step4:
$S_0 \to YA|AA|XA|a$
$S \to YA|AA|XA|a$
$A \to XA|a$
$X \to a$
$Y \to AA$

f) Pumping lemma :- If L is a regular language then there exist a number 'n' such that $w \in L$.
- w is divided into 3 parts x, y, z.

$|xy| \leq n$. $k \geq 0$. $xy^kz \in L$.
$|y| \geq 1$

8) Chomsky hierarchy. with example.

9) $S \to SS \mid (S) \mid a$.

Left Recursive

$S \to \underset{S}{SS} \mid \underset{\beta}{(SY} \mid \underset{\beta}{a}$.

$S \to (SY \mid a)\, S' \to (SYS' \mid$

$S' \to SS' \mid \epsilon$.

- If is not in CNF.

$S \to SS \mid XSY \mid a$.

$X \to ($

$Y \to)$

- not in CNF

$S \to SS \mid (SY \mid a$.

$X \to ($

$Y \to)$.

- $S \to (SYS' \mid aS'$

$S' \to SS' \mid \epsilon$.

$X \to ($

$Y \to)$.

Replace S' & $O' \to \epsilon$

$S \to (SYS' \mid aS'$

$S' \to (SYS'S' \mid$

- $S \to (SYS' \mid \epsilon SY \mid aS' \mid a$.

$S' \to SS' \mid S$

$X \to ($

$Y \to)$

- $S \to (SYS' \mid (SY \mid aS' \mid a$.

$S' \to (SYS'S' \mid (SYS' \mid aS'S' \mid aS' \mid (SYS' \mid (SY \mid aS' \mid a$.

$X \to ($

$Y \to)$

10) Properties of CFL.

1) union.

2) concatenation

3) Kleen closure

4) Pumping lemma.

11) $q_1 = q_1a + q_3a + \epsilon$

$q_2 = q_1b + q_2b + q_3b$.

$q_3 = q_2a$.

$q_2 = q_1b + q_2b + q_2a$.

$q_2 = \underset{Q}{q_1b} + \underset{R}{q_2}\underset{P}{(b+a)}$

$q_2 = q_1b(b+a)^*$.

$q_1 = q_1a + q_2aa$

$q_1 = q_1a + q_1b(b+a)^*aa$.

$q_1 = \underset{R}{q_1}\underset{P}{(a + b(b+a)^*aa)}$

$q_1 = (a+b(b+a)^*aa)^*$.

12) GNF same 9

13) $L = x^n y^n z^n \mid n >= 1$

Let $n = 4$.

$L = xxxxyyyyzzzz$ $w = abcde$.

$a = xxx$ $|bcd| >= n$.

$b = xy$ $6 >= 4$.

$c = y$

$d = yyz$ $|bd| >= 1$

$e = zzz$ $5 > 1$

$k = 0$.

$w = ab^k cd^k e = ace = xxxyzzz \notin L$.

14) Prove union of 2 regular languages is regular.

15) $E \to E(T) \mid T$. $T \to T(F) \mid F$. $F \to id$

$E \to TE'$

$E' \to (T)E' \mid \epsilon$.

$T \to FT'$

$T' \to (F)T' \mid \epsilon$.

16) a b b c
1 2 3 4 4 rows & 4 columns.

17) $A = \{a^n b^n \mid n \geq 0\}$.

$w = aaabbb$ $n = 3$

$x = aa$

$y = ab$

$z = bbb$

$|xy| \leq n$. $3 \leq n$.

$|y| \geq 1$ $1 \geq 1$

$k = 0$.

$w = xy^k z = xz = aabbb \notin L$.

18) union of two CFL is also context free.

$L_1 = \{a^n b^n c^m \mid n \geq 0, m \geq 0\}$

$L_2 = \{a^n b^m c^m \mid n \geq 0, m \geq 0\}$

$L_1 \cup L_2 = \{a^n b^n c^m \cup a^n b^m c^m \mid n \geq 0, m \geq 0\}$.

19) Steps for CFG to GNF.

Step1: It is in CNF

Step2: Remove LG

Step3: convert production rule to GNF.

unit-3

1) **Push Down Automata** – It is to represent the CFG like DFA for regular language.

- It is more superior ↳ better than DFA.
- It represents language which were not represented by DFA.

2) components of PDA.

$$L = \{Q, q_0, F, \Sigma, \delta, \Gamma, Z\}$$

Q: finite set of states
q_0: initial state
F: finites set of final states
Σ = set of input.
δ = Mapping function
Γ = stack symbol pushed or popped from stack.
Z = first symbol in Γ

3) **Deterministic PDA** — **NDPDA**

4) context sensitive language :- these are the language where terminal with non-terminal deriving terminal with non terminal

as → aBB
aB → S.

5) **Turing machine** :- It is to represent the recursive enumerable language (Type 0)

- It has input tape & finite control
- It reads, writes & erase in input tape.

6) components of TM

$(Q, f, q_0, \Sigma, \delta, T, B)$

T: Turing alphabet

B: Blank symbol

7) Recursive Enumerable language :- It is a formal language by TM where it halts and accepts if string is in language or else halt and reject or infinite loop

8) Variants of TM :- Multi-Tape TM :- multiple tapes one head handled by single Tape

Multi-head multi-Tape - handled by standard turing machine.

9) Linear Bound Automat with example :- It is same as TM but with non-deterministic, multi tape, bounded length of tape finite

$L = \{a^{n!} \mid n >= 0\}$

10) Define ID of TM

11) Instantenous ~~Defination~~ Description :- It is used to represent the PDA where it takes input and decides to accept or reject.

represented by (q, w, α)

q: represent present state

w: represent next input set

α: describe stack content

12) How stack is used for deciding acceptance of string in pda.

push, pop, Ignore.

13) undecidable Problem :- It is problem for which no algorithm is there even T.M cannot.

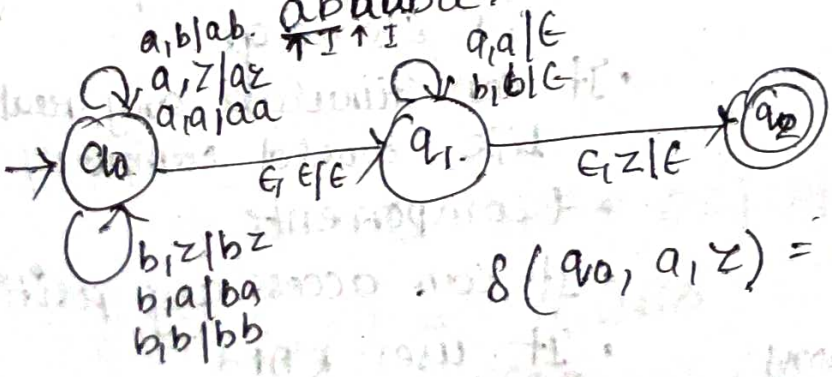
14) The finite control moves on both direction for any input tape.

15)

PDA	Turing machine
• It is for CFG	• It is recursively enumerable lange
• It has 6 components	• It has 7 components
• It use stack	• It uses RAM
• It is less powerful	• It is more powerful

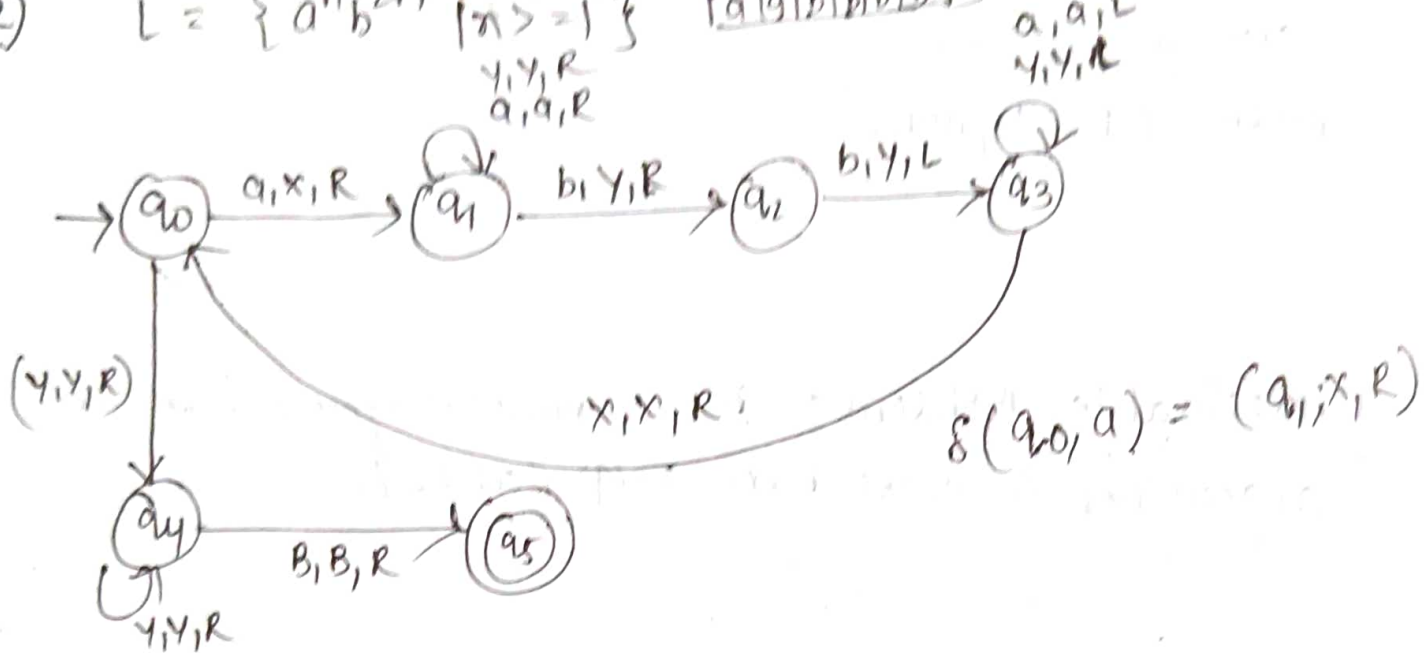
long?

1) PDA for ww^R

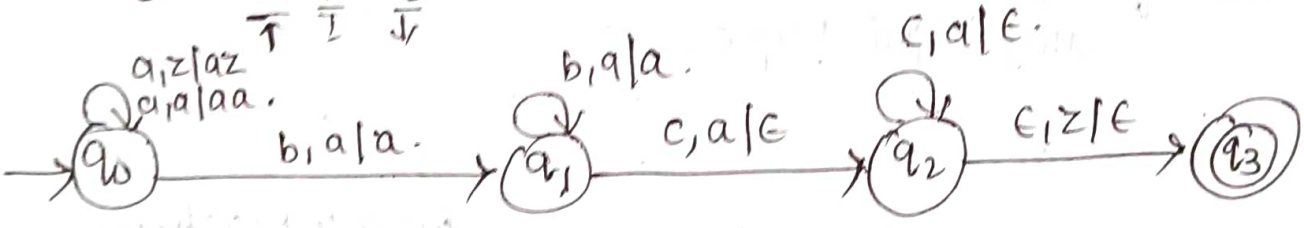


$\delta(q_0, a, Z) = (q_0, aZ)$

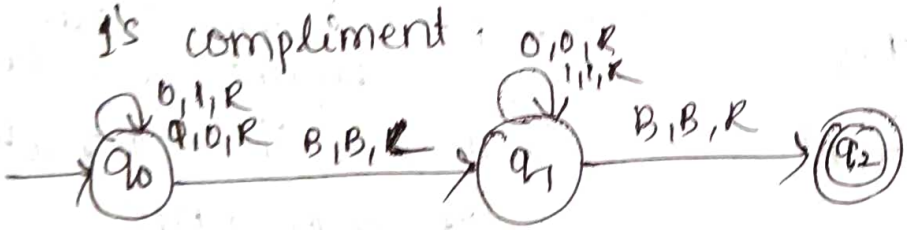
2) $L = \{ a^n b^{2n} \mid n >= 1 \}$



3) $L = a^m b^m c^n$



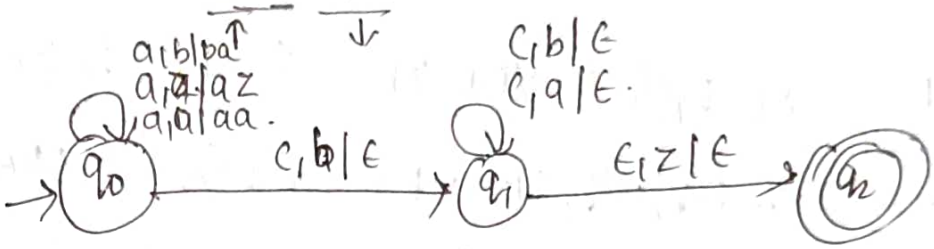
4) 1's compliment.



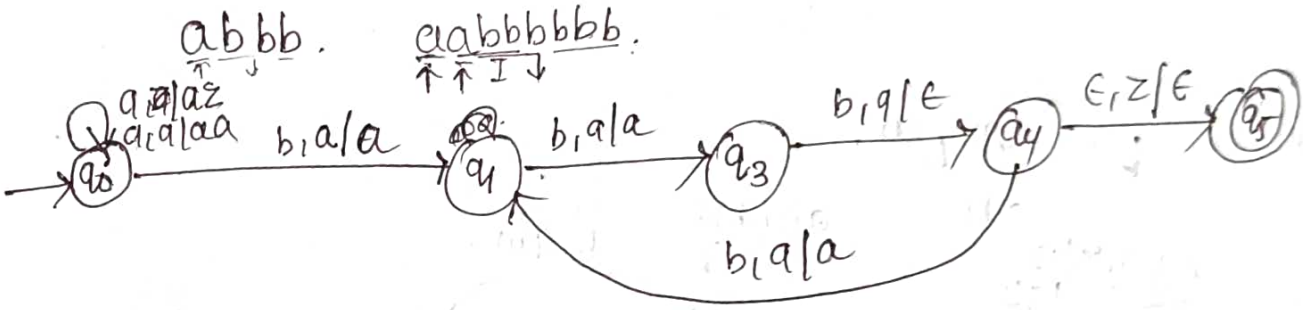
5)

PDA	TM
• It is defined for CFL	• It is for recursive enumerable language.
• It is less powerful	• It is more powerful
• It has limited	• It has infinite tape length storage.
• It cannot.	• It can stimulate any real-life world computer
• 7 components	• 7 components
• It can access only top of stack	• It can access any position
• It uses stack as memory	• It uses RAM

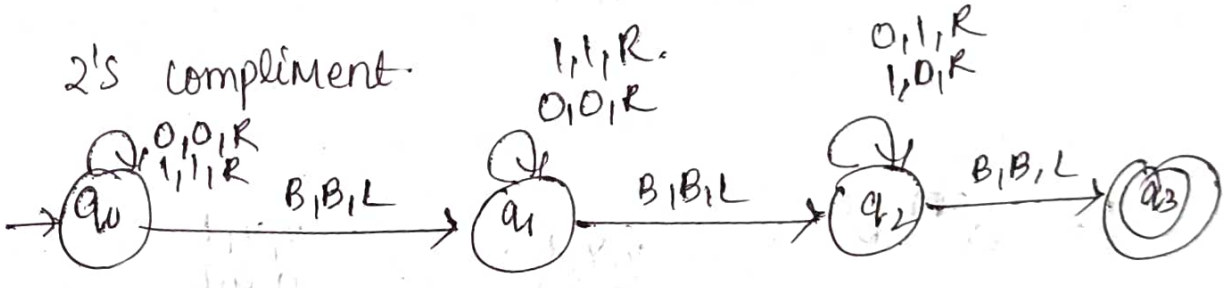
6) $L = a^n b^m c^n c^m$.



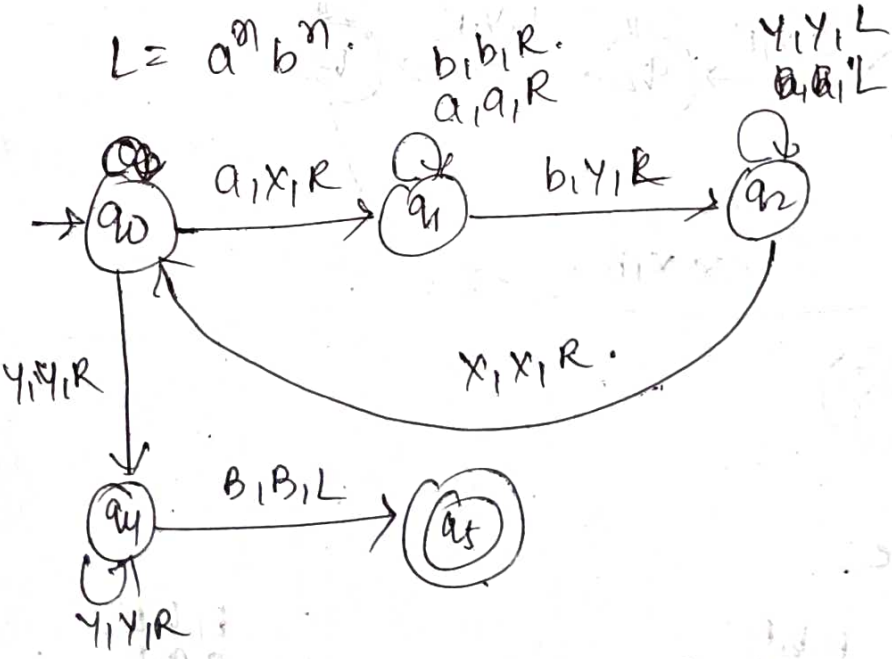
7) $L = a^n b^{3n}$.



8) 2's compliment.



9) $L = a^n b^n$.



10) $S \rightarrow OBB \quad B \rightarrow 0S|1S|0$.

$\delta(q, \epsilon, S) = (q, OBB)$

$\delta(q, \epsilon, B) = \{(q, 0S), (q, 1S), (q, 0)\}$

$\delta(q, 0, 0) = (q, \epsilon)$

$\delta(q, 1, 1) = (q, \epsilon)$

0104.

$\delta(q_1, 0104, s) \vdash \delta(q_1, 104, \$BB) \vdash \delta(q_1, 104, BB)$

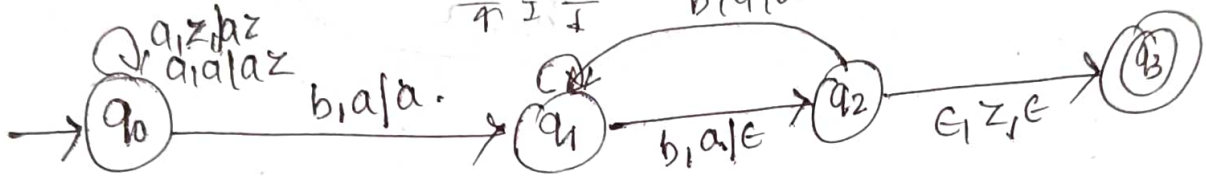
$\delta(q_1, 104, sBB) \vdash \delta(q_1, 04, sBBB) \vdash \delta(q_1, 4, BBBB)$

$\delta(q_1, 4, BBB)$.

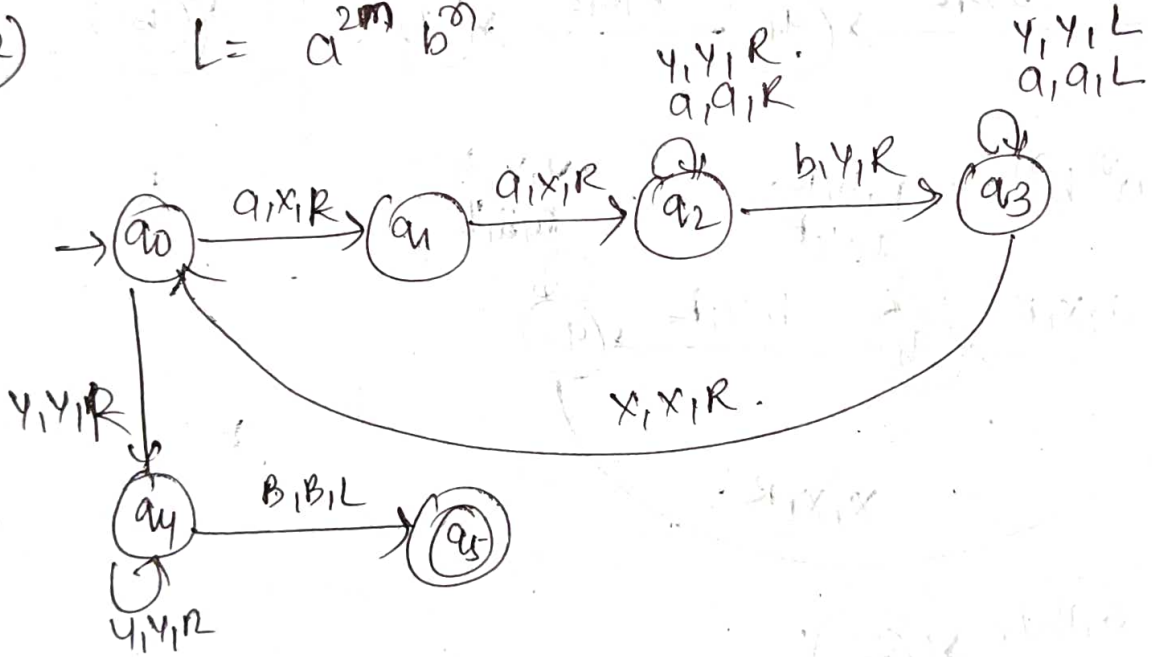
11) $L = a^n b^{2n}$.

abb

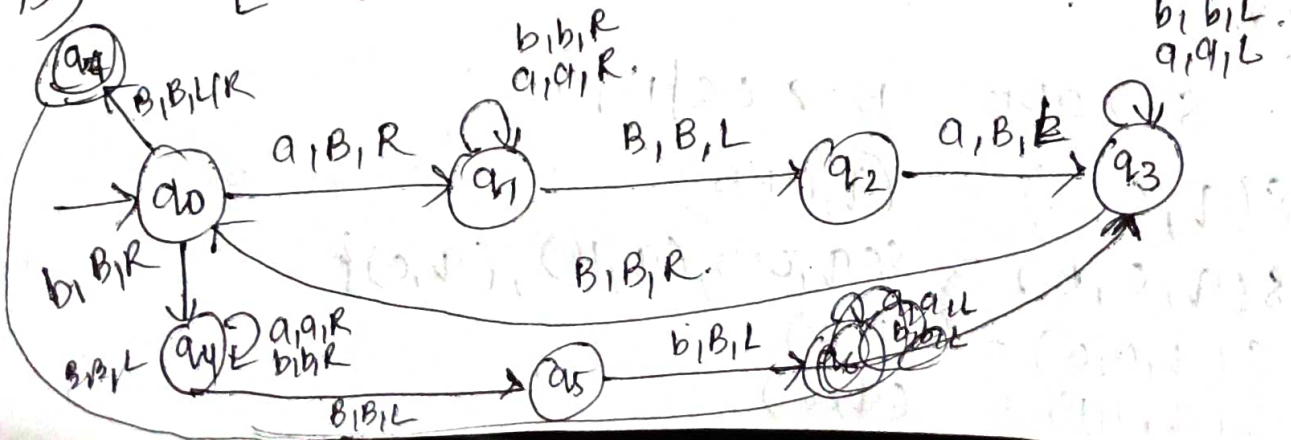
aa bb bb



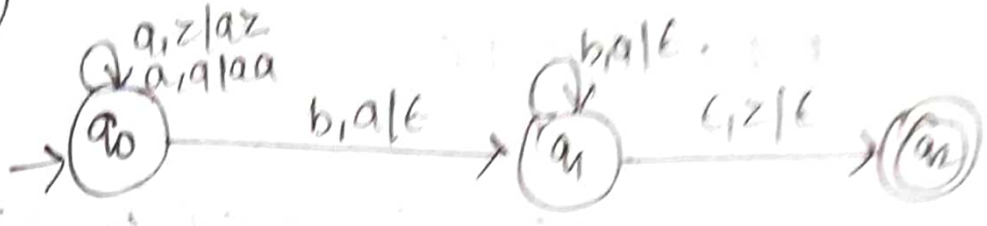
12) $L = a^{2n} b^n$.



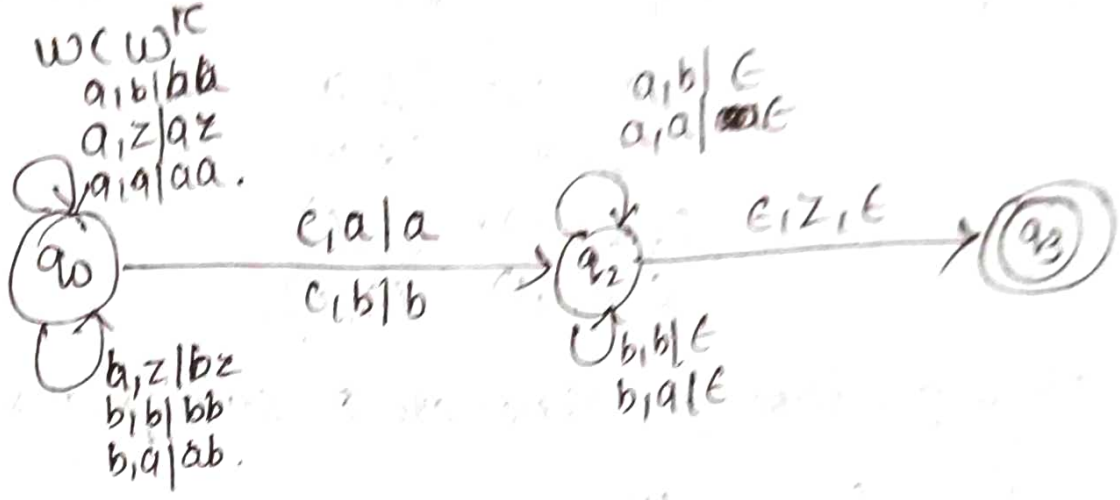
13) $L = ww^R$.



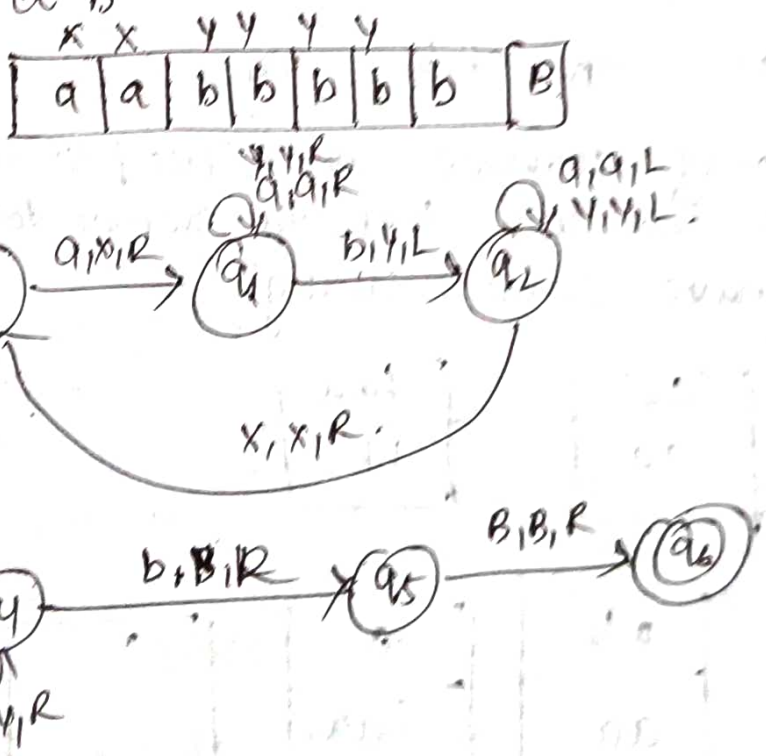
14) $a^n b^n$



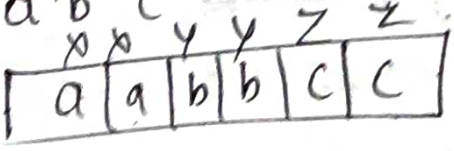
15) $w c (w)^{rc}$



16) $a^n b^{2n+1}$



17) $a^n b^n c^n$



18) $\delta(q, \epsilon, S) = (q, 0S1), (q, A)$

$\delta(q, \epsilon, A) = (q, 1A0), (q, S)$

$\delta(q, 0, 0) = (q, \epsilon)$

$\delta(q, 1, 1) = (q, \epsilon)$.

18) $S \to 0S1 \mid A$
$A \to 1A0 \mid S \mid \epsilon$

$S \to 0S1 \mid 1A0$

~~$S \to 0S1 \mid 1S0 \mid \epsilon$~~

$S \to 0S1 \mid 1S0 \mid \epsilon$.

GNF $S \to 0SX \mid 1SY \mid \epsilon$.
$X \to 0$
$Y \to 0$

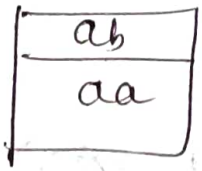
$\delta(q_1, \epsilon, S) = \{(q_1, 0SX), (q_1, 1SY), (q_1, \epsilon)\}$
$\delta(q_1, \epsilon, X) = (q_1, 0)$
$\delta(q_1, \epsilon, Y) = (q_1, 0)$
$\delta(q_1, 0, 0) = (q_1, \epsilon)$
$\delta(q_1, 1, 1) = (q_1, \epsilon)$.

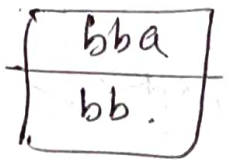
19) Post correspondance :- These are 2 strings A & B

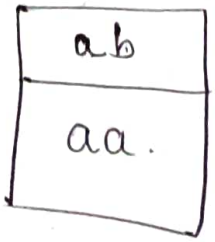
$A = a_1 \ldots a_n$
$B = b_1 \ldots b_n$.

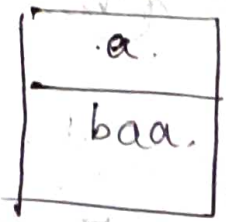
such that $a_1 a_2 \ldots a_n = b_1 b_2 \ldots b_n$.

- There are n no. of dominos which need to be arranged in such way that the numerator and denominator will have same string.









unit-4.

1) Recursive function :- It is a function that calls itself during execution.
- It break down a problem into smaller problems and solve each problem by invoking same function.

2) Primitive Recursive Function :- It is primitive recursive function as it contains initial functions or it is found out by composition or recursive of initial function over n.
finite number of.

3) µ-recursive function :- It is partial function take finite number of tuples and gives single natural number.
- They are the smallest class of primitive recursive function having initial functions closed under the composition, primitive recursion and µ-operator.

4) A godel numbering is an encoding process where some values are assigned to the each symbol of mathematical notation.
Here natural number represents the symbol of mathematical notation.
If a sequence defined $a_1, a_2, \ldots a_n$ then it is represented by power of prime number from 2.

$$a_1, a_2, \ldots, a_n = 2^{a_1} \times 3^{a_2} \times \ldots \times 3^{a_n}.$$

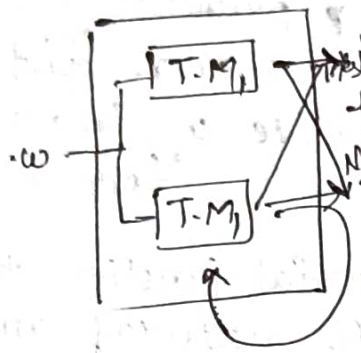
5) Church turing hypothesis
- Each and every problem is solved with TM.
- There is TM-n corresponding to every computable problem.
- we can model any mechanical computer with a TM
- The set of languages that can be decided by TM is same as set of languages decided by any mechanical computing machine.

If there is no TM that decides problem P, there is no

algorithm that solves problem P.

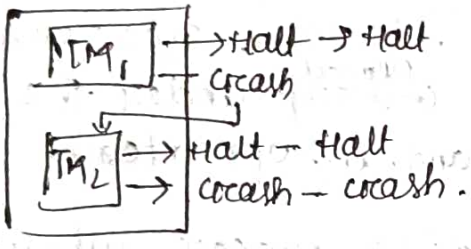
b) Recursive language - If there exist a TM that can decide whether any given string belongs to language in finite amount of time.

union of Recursively enumerable language.

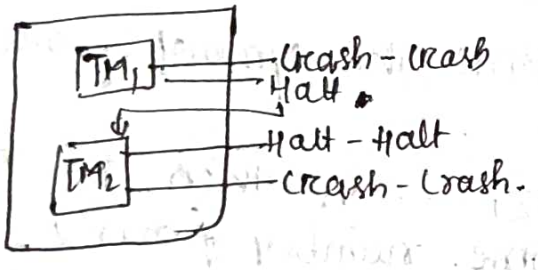


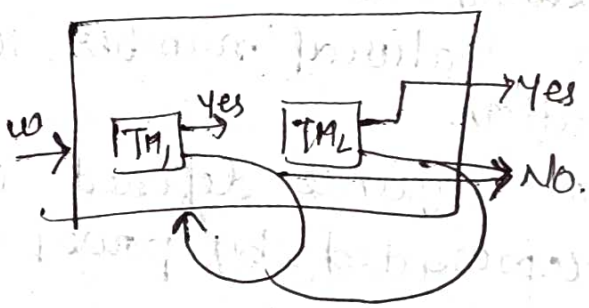
If L_1 halts then system halts

If L_1 does not halt and L_2 halts then system halts.



Intersection





Recursive languages recursive under intersection.

$L_1 = \{a^n b^n c^n d^m \mid n>=0 \text{ \& } m>=0\}$

$L_2 = \{a^n b^n c^n d^n \mid n>=0 \text{ \& } m>=0\}$

L_1 says n no. of a's followed ....

L_2 says n no. of a's followed ....

Their intersection says. n no. of a's followed ....
n no. of d is recursive.