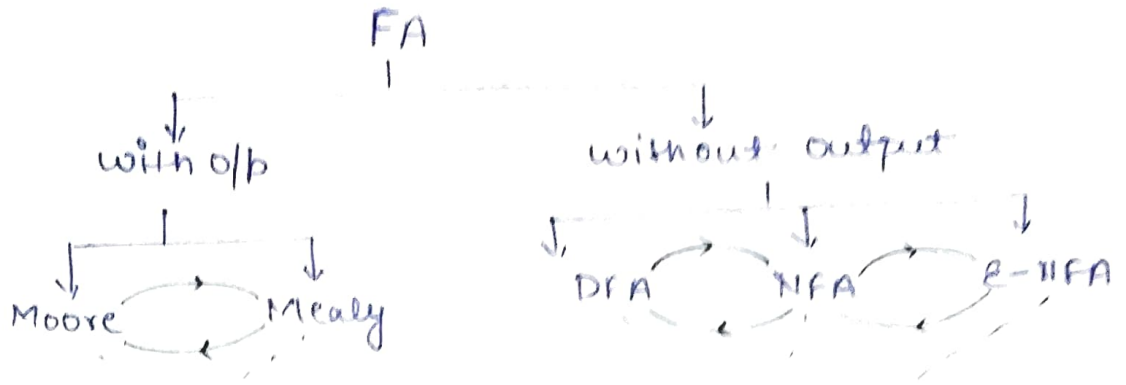
Note → All the finite Automata's (DFA, NFA, ε-NFA) are equivalent in power and inter-Convertible to one another.

Family of languages

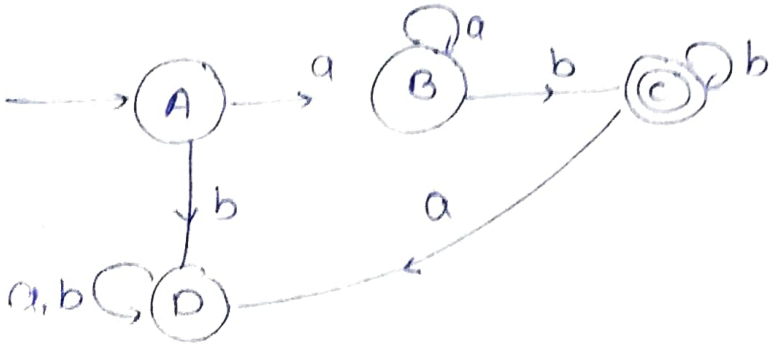


(Equivalent in Power)

(Equivalent in Power)

Ex $\{a^n b^n ; n \geq 1\}$

$L = \{ab, aabb, aaabbb, \ldots\}$

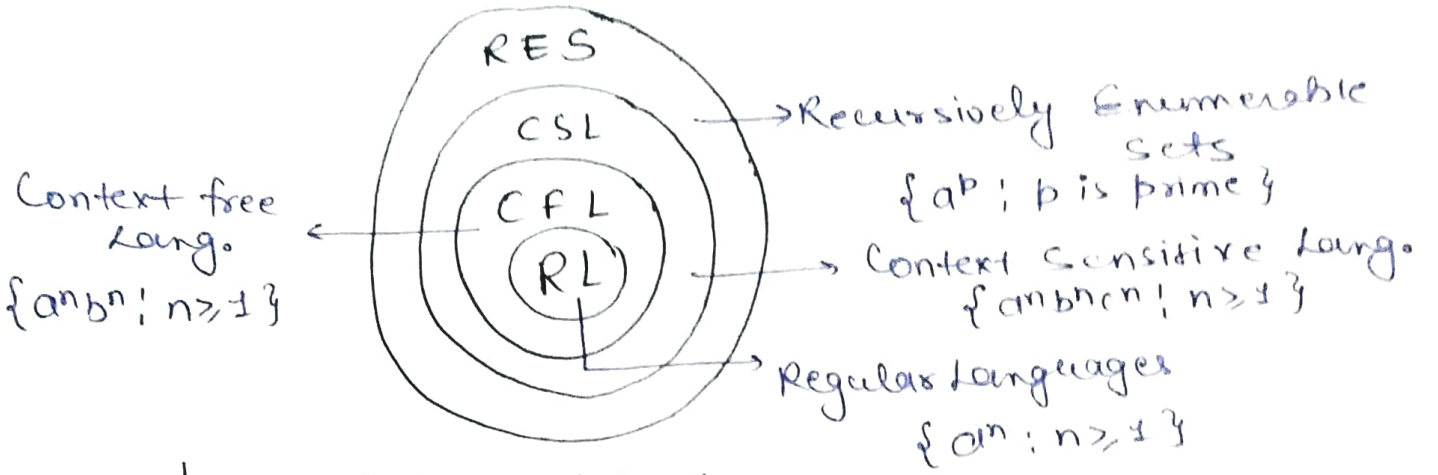


It is also accepted the language aab so it is not the suitable DFA

We can't count infinite no. of occurence using finite automata.

So, FA + memory ⇒ Push Down Automata (PDA)

So, Some languages may exist for which 'no' finite automata exist.



Language	Accepted by	Machine
GRAMMAR → (RG) → RL	→	Finite Automata
→ (CFG) → CFL	→	Pushdown Automata
→ (CSG) → CSL	→	Linear Bounded Automata
→ (UG) → REL	→	Turing Machine

⋮ → Unrestricted Grammars

Regular Expressions and Conversion

① Regular languages
- ③ acceptors → FA → with o/p, without o/p
- ② generators → Regular Grammars → Right Linear Grammar, Left Linear Grammar
- Representers → Regular Expressions

Regular Expression

Operations:
- i) + (Union)
- ii) . (Concatination)
- iii) * (Kleene closure)

Primitive RE's:
- a) ϕ → empty set, $\{\}$
- ε → Null string, $\{\varepsilon\}$
- Σ → $a \in \Sigma$, $\{a\}$

b) $r_1 + r_2$, $r_1 . r_2$, r_1^*] R. Expression

c) You can use a) and b) combiningly to form many Regular Expression as

$\phi = \{\}$, $\varepsilon = \{\varepsilon\}$, $a = \{a\}$, $a^* = \{\varepsilon, a, aa, aaa\}$

$a^+ = \{a, aa, \ldots\} \rightarrow \boxed{a \cdot a^*}$, $(a+b)^* = \{\varepsilon, a, b, aa, ab, ba, bb, \ldots\}$

a^+ ↓ {Additive closure}

$(a+b)^*$ ↓ set of all strings possible over a and b.

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

① $L_1 = \{$set of all string whose length $= 2\}$

$= \{aa, ab, ba, bb\}$

$\Rightarrow aa + ab + ba + bb \Rightarrow a(a+b) + b(a+b)$

$\Rightarrow (a+b)$ ② ← ②

$(a+b)$ ↓ set of all strings over a and b of length ②

② $L_2 = \{$ Set of all strings whose length $\geq 2\}$

$L_2 = \{aa, ab, ba, bb, aaa, \ldots\}$

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

$(a+b)(a+b)$ ↓ for two ; $(a+b)^*$ ↓ for zero or more

③ $L_3 = \{$ at most $2\} = \{\varepsilon, a, b, aa, ab, ba, bb\}$

$\varepsilon + a + b + (a+b)(a+b)$

$(a+b)(a+b+\varepsilon) + \varepsilon \rightarrow (a+b+\varepsilon)(a+b+\varepsilon)$

④ $L_4 = \{$ strings of even length.$\}$

$= \{\varepsilon, aa, ab, ba, bb, \ldots\}$

$((a+b)(a+b))^* \rightarrow (2^0, 2^1, 2^2 \ldots)$

↓ String of length 2

L_5 = { string of odd length }

$\boxed{((a+b)(a+b))^*(a+b)}$ = $(a+b)^{2*}(a+b)$

Ans

L_6 = { string of length divisible by 3 }

$L = 0, 3, 6, 9$

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

L_7 = { String length $\cong 2 \bmod 3$ }

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

L_8 = { no. of a's are exactly 2 }

$b^*ab^*ab^*$

L_9 = { a's are atleast 2 }

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

L_{10} = { a's are atmost 2 }

$b^*(\epsilon+a)b^*(\epsilon+a)b^*$

L_{11} = { no. of a's are even }

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

$\Downarrow \cong$

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

L_{12} = { string starts with a }

$a(a+b)^*$

L_{13} = { ends with a }

$(a+b)^*a$

L_{14} = { containing a }

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

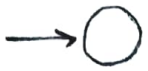
L_{15} = { starting and ending with different symbol }

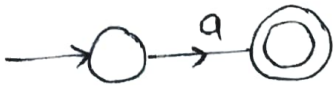
$\boxed{a(a+b)^*b + b(a+b)^*a}$

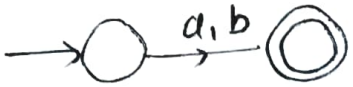
L_{16} = { starts and end with same symbol }

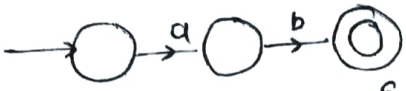
$$a(a+b)^*a + b(a+b)^*b + a + b + \varepsilon$$

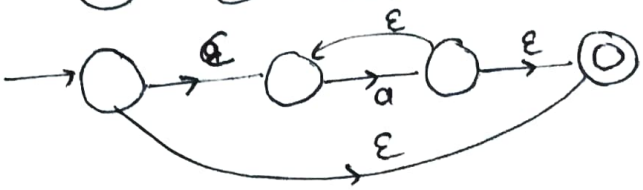
Regular Expression to finite Automata –

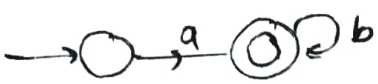
i) ϕ 

ii) a 

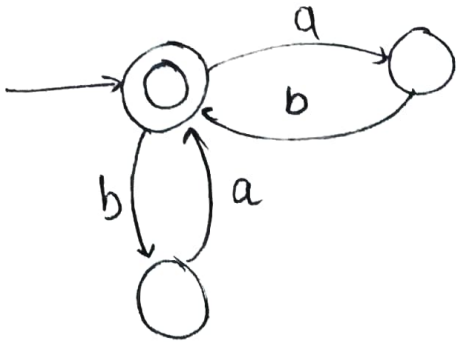
iii) (a+b) 

iv) (a.b) 

v) a^* 

vi) ab^* 

vii) $(ab)^*$ 

viii) $(ab+ba)^*$ 

→ Regular Expression and finite Automata both are equivalent in Power.

finite Automata to Regular Expression

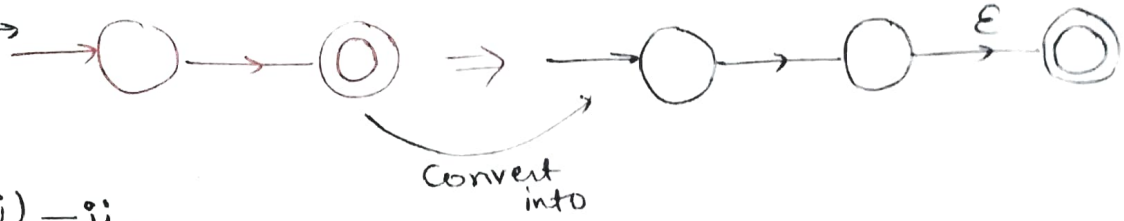
- Arden's Lemma (Not for Gate purpose)
- State Elimination Method → Not works for ε-NFA

Rules.

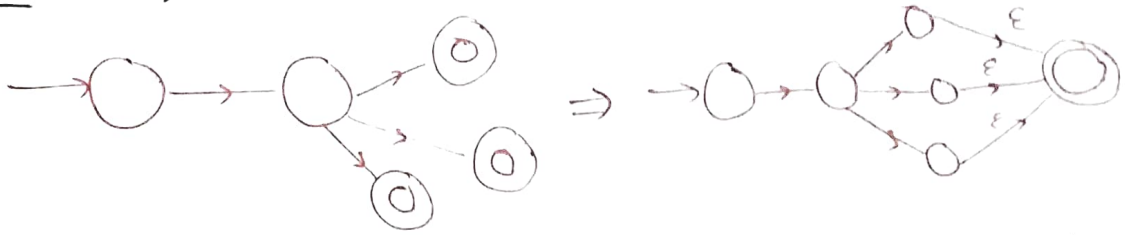
i) Initial state not suppose to have any incoming edge in initial state.

ii) No outgoing edge to the final state. It must have only 1 final state.

(iii) i →

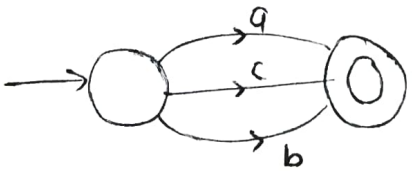


(ii) → ii →

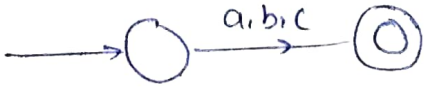


(iii) Eliminate the states other than initial and final states.

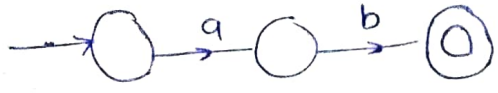
Ex



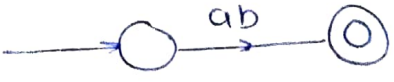
No incoming edge to initial state and No outgoing state to from final. So



→ (a+b+c)

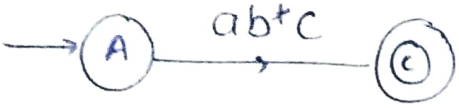
② 

↓



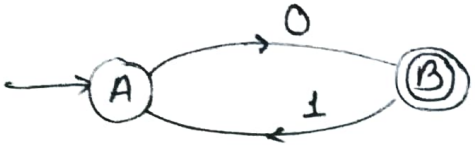
→ (ab)

(3)

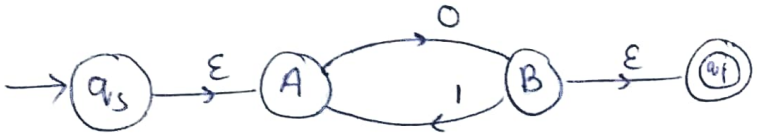


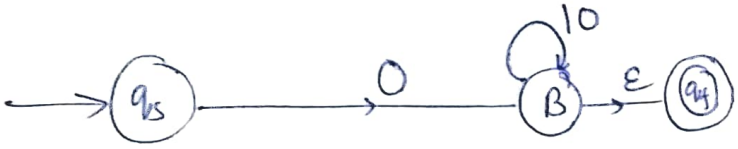
(Elimination of B)

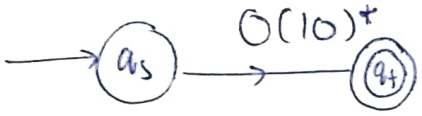
(4)



$\Rightarrow$

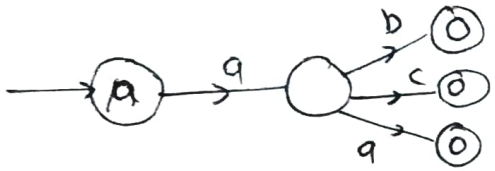


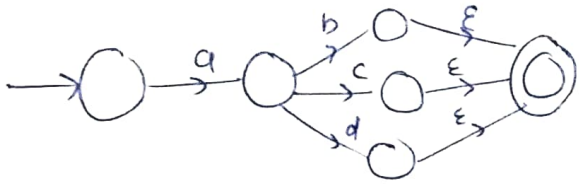


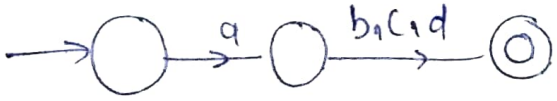


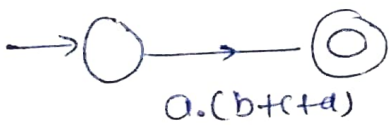
$(0(10)^*)$

(5)



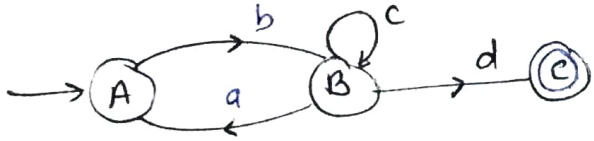


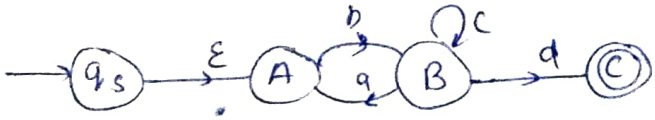




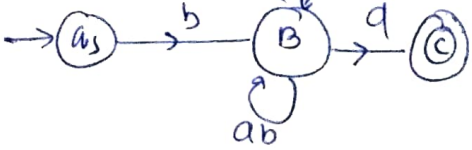
$\rightarrow$ $a.(b+c+d)$

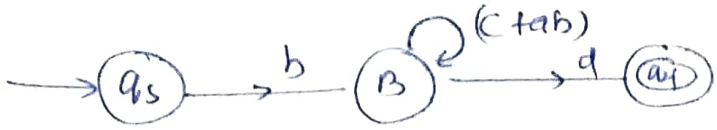
(6)

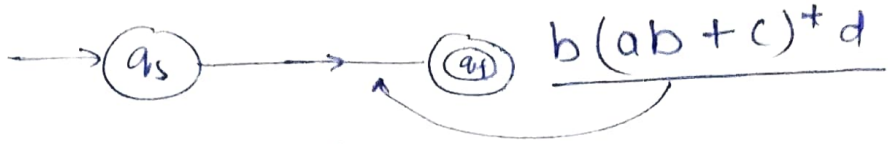




$\Rightarrow$

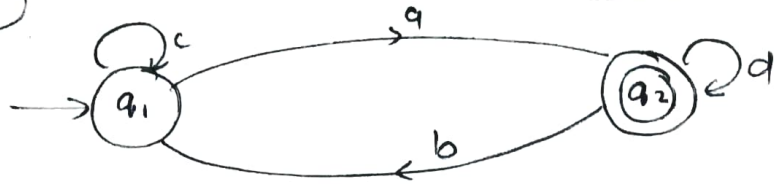




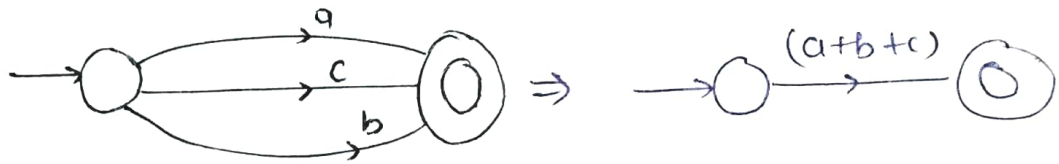


$b(ab+c)^* d$

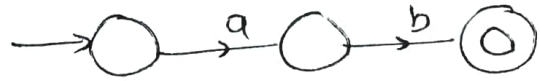
(7)



(8)

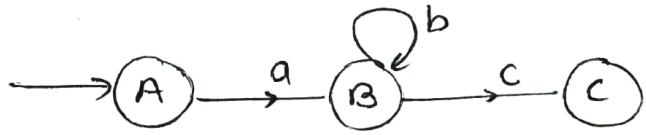


(9)



ab

(10)

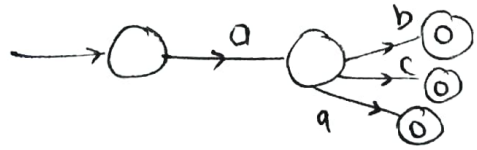


ab^*c

(11)

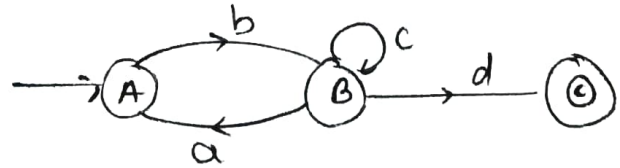
$0(10)^*$

(12)



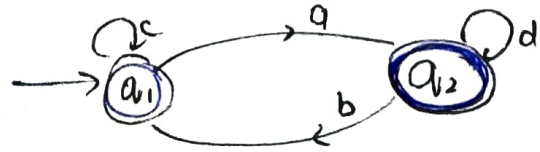
$a.(b+c+d)$

(13)

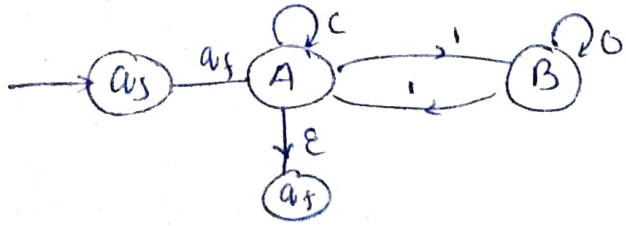


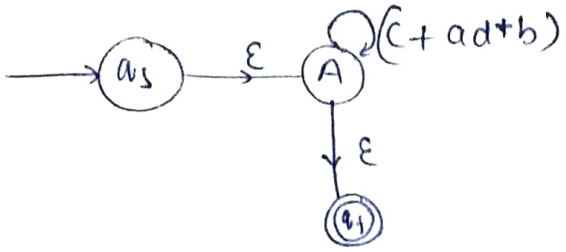
$b.(c+ab)^* d$

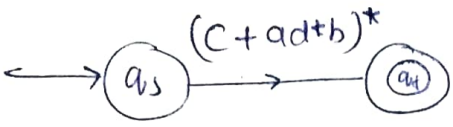
(14)



~~$c^*(a.(d+ab)^*$~~







① If both are independent then regular

② If both are dependent then limit should be bounded

③ If single variable in pump then powers should be in A.P

④ ① and ③ works combinedly

Testing whether a language is regular or not

If Language is finite then it must be regular.

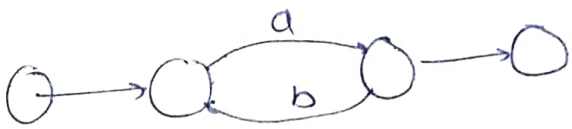
If Language is infinite then we need to check

Ex -

$L = \{ab, abab, ababab, \dots\}$

Infinite loop is accepted by the finite automata so it must contains loop in some pattern and if no such pattern then no finite automata will exist.

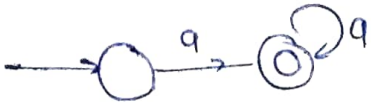
Here pattern is 'ab'.



According to Pumping Lemma if their is no pattern Language is not regular, but it is not necessarily said that Language is regular if pattern is found.

⇒ 'Pumping Lemma is a negativity test'

Ex → $L = \{a^n \mid n \geq 1\}$ → Regular bcoz finite automata exists



(2) $a^n b^m \mid n, m \geq 1$ → Regular Language both are independent.

(3) $a^n b^n \mid n \leq 10^{10^{10}}$ → n is bounded so language is finite and regular.

(4) $a^n b^n \mid n \geq 1$ → finite automata can do only finite counting not infinite counting. So Language is not regular.

'b' depends on 'a'

(5) $ww^R \mid |w| = 2, w \in \{a, b\}$ → finite language. so it is regular due to fixed length.

(6) $ww^R \mid w \in (a, b)^*$ → Language can't be regular bcoz $|w|$ is not finitely defined.

↓ Represents palindrome.

(7) $ww \mid w \in (a, b)^*$ → Language is not regular because $|w|$ is not finite.

(8) $a^n b^m c^k \mid m, n, k \geq 1$ → Language is regular all a, b, c have different power. No one is depend on other.

(9) $a^i b^{2j} \mid i, j \geq 1$ → Language is regular with Regular exp. as $aa^* bb(bb)^*$.

(10) $a^i b^{4j} \mid i, j \geq 1$ → It is also regular.

(11) $\Sigma = \{a\}$

i) $a^n \mid n$ is even → Regular with pattern of a^2.

ii) $a^n \mid n$ is odd → Regular with patter of a^2.

iii) $a^n \mid n$ is prime → Not regular because no loop

iv) $a^{n^2} \mid n \geq 1$ → Not regular.

iv) $a^{2^n} \mid n \geq 1$ → Not regular

Trick → 9:01

i) $a^n \mid n$ is even = $\{a^0, a^2, a^4, a^6 \ldots\}$

2-0=2, 6-4=2, 4-2=2

So all the powers in AP

If there is only one symbol in string and power alloted to the symbols are in AP as $\{0, 2, 4, 6, \ldots\}$ then Language is regular.

that's why (iii), (iv) and (v) is previous excemple are not regular.

$\Sigma = \{a, b\}$ → Two Symbols

i) $a^i b^{j^2} \mid i, j \geq 1$ → Here both a, b can generate independenty but b^{j^2} doesn't form any patten (AP) so due to b^{j^2} the Language is not regular.

ii) $a^i b^{2^n}$ → Not regular reason is same as above

iii) $a^i b^p \mid i \geq 1, p$ is prime → Not regular reason same.

iv) $w \mid n_a(w) \begin{matrix} \geq \\ = \\ \leq \end{matrix} n_b(w)$ → small 3 cases → w is of finite length and we want to save the legth of both strings contains 'a' and 'b' and also want to compare it that is not possible.

v) $w \mid n_a(w) \bmod 3 \leq n_b(w) \bmod 3$ → It is Regular.

vi) $ww^R \mid w \in (a,b)^*$ → Not a regular language.

vii) $a^n b^n c^n \mid n \geq 1$ ⟶ Non-regular

viii) $a^n b^{n+m} c^m$ ⟶ Non regular
$n, m \geq 1$

Note → All the examples discussed till now can be asked directly in GATE. So must remember them or mug them up.

Imp

$wxw^R \mid w, x \in (0,1)^*$

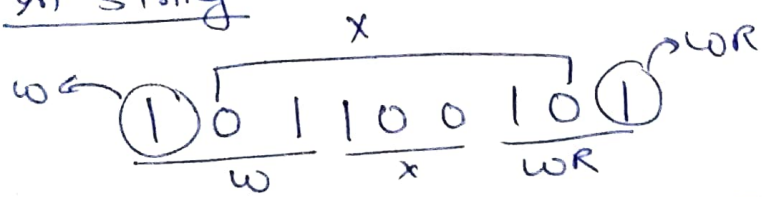
$w = 101$
$x = 100$
$w^R = 101$

$wxw^R \rightarrow 101100101$ (w | x | w^R)

Here we can say that Language is non Regular due ww^R and w is unbound but this language is 'Regular'.

Reason

On string



Here x के पास ability है कि वो w, w^R के bits को consume कर सकता है तो w और w^R चाहें तो केवल end symbol को Represent कर सकते हैं और finite length के बन सकते हैं। So Language is regular.

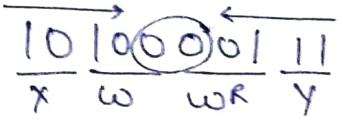
But if $wxw^R \mid w \in (0,1)^*, |x|=5$ — (2)

Here X can't expand beyond 5. So Language is not regular.

$\Rightarrow$ $xww^Ry \mid x,y,w \in (0,1)^+$

$x = 10 \quad w = 100$

$y = 11 \quad w^R = 001$



Here X can eat all symbol of w except last and Y can eat all symbol of w^R except R.

So, solution will contain pattern of 00 as substring or 11 as substring based on w and w^R

$(0+1)^*00(0+1)^*$ or $(0+1)^*11(0+1)^*$

So, Language is regular.

In above question if $|x|$ and $|y|$ is given as finite then Language xww^Ry is not regular.

$xww^R \mid x,w \in (0,1)^+$ → Not a regular Language

$w = 101$

$w^R = 101$ → 11 101 101

$x = 11$

x w w^R

Last two symbols doesn't guaranteed 00, 11 always

$ww^Ry \mid w,y \in (0,1)^+$ → Not a regular language

③ Pumping Lemma (Negative Test)

If L be an infinite regular language and $w \in L$ such that $|w| \geq n$, for some positive integer n, then w can be decomposed into three strings $w = xyz$ such that

i) $y \neq \varepsilon$

ii) $|xy| \leq n$

iii) for every $k \geq 0$ the string xy^kz is also in L.

↳ It shows loop in string.

Note → Every regular lang. satisfies the pumping lemma.

② If a Language 'L' doesn't satisfy the pumping lemma then 'L' is not regular.

③ If a Language 'L' satisfies the pumping lemma it need not be always regular.

④ Pumping lemma is used to prove that some of the languages are not regular.

Ex $\{a^n b^n : n \geq 1\}$ → Non regular

$\underset{x}{a}\ \underset{y}{aaa}\ \underset{z}{bbb}$ $\quad n = 4$

If we repeat y then

$y = aa$ ← $a\ aaaa\ abbbb$ → string not belongs to L.

$y = bb$ ← $a\ aaabbbbbb$ → string not belong to L

$y = ab$ ← $a\ aaba\ bbb$ → "

Grammars

Introduction to Grammar :-

$G = (V, T, P, S)$ — P: Productions; S: Starting Symbol

V: Vertices; T: Terminals

Ex →

$S \to aSB$
$S \to aB$
$B \to b$

$V = \{S, B\}$ → LHS

$T = \{a, b\}$

→ P

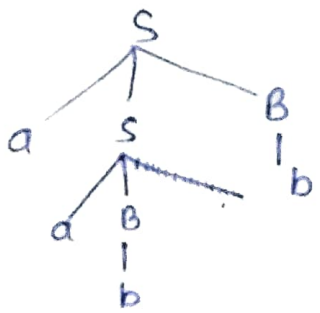
$S = \{S\}$

1. **Derivation of aabb**

→ ① $S \Rightarrow aSB$
$S \Rightarrow aaBB$
$S \Rightarrow aabB$
$S \Rightarrow aabb$

Intermediate results are called sentential or sequential form. It may be Left Sentential or Right Sentential.

→ ② Derivation Tree



This Grammar represents $a^n b^n \mid n \geq 1$

Ex → ① $L = \{aa, ba, ab, bb\}$ $(a+b)(a+b)$

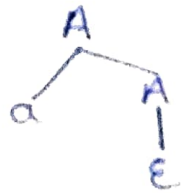
$S \to AA$
$A \to a$
$A \to b$

Set of all strings of length 2.

② $a^n \mid n \geq 0$

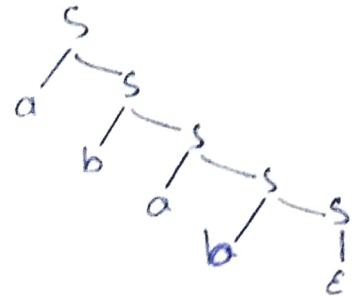
$\{a, aa, aaa, \ldots\}$

$A \to Aa \mid \varepsilon$
$A \to aA \mid \varepsilon$
$A \to \{\varepsilon, a, aa, \ldots\}$



(3) $L = (a+b)^*$ for abab

$S \rightarrow aS \mid bS \mid \varepsilon$



(4) String of length atleast 2

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

Consisting of two blocks so basic building block of $(a+b)^*$ and $(a+b)^2$ is $S \rightarrow AAB$

$A \rightarrow a \mid b$

$B \rightarrow aB \mid bB \mid \varepsilon$

(5) String of length atmost 2.

$(a+b+\varepsilon)(a+b+\varepsilon)$

$S \rightarrow AA$

$A \rightarrow a \mid b \mid \varepsilon$

(6) Strings starts with 'a' and ends with 'b'.

$a(a+b)^* b$

$S \rightarrow aAb$

$A \rightarrow aA \mid bA \mid \varepsilon$

(7) Set of strings starts and ends with different symbols.

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

$S \rightarrow aAb \mid bAa$

$A \rightarrow aA \mid bA \mid \varepsilon$

(8) Starts and ends with same symbol

$S \rightarrow aAa \mid bAb \mid a \mid b \mid \varepsilon$

$A \rightarrow aA \mid bA \mid \varepsilon$

(9) $a^n b^n \mid n \geq 1$

This language is not regular so we can't write its grammar using Regular Expression we have to choose another approach.

$S \rightarrow asb \mid ab$

(10) $wwR \cup wawR \cup wbwR \mid w \in (a,b)^*$

wwR → even length palindrome

$wawR \cup wbwR$ → odd length palindrome

$S \rightarrow aSa \mid bSb \mid a \mid b \mid \varepsilon$

$a \mid b$ → without it all even length palindrome

ε → without this all odd length palindrome

(11) Even length strings

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

$S \rightarrow BS \mid \varepsilon$

$B \rightarrow AA$

$A \rightarrow a \mid b$

(12) $a^n b^m \mid n,m \geq 1$

$S \rightarrow AB$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$

(13) $a^n b^n c^m \mid n,m \geq 1$

Generate both separately and club them.

$S \rightarrow AB$

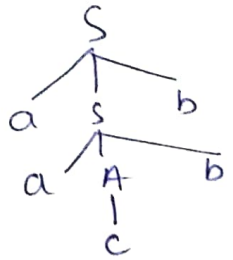
$A \rightarrow aAb \mid ab$

$B \rightarrow cB \mid c$

(14) $a^n c^m b^n \mid n,m \geq 1$

$S \rightarrow asb \mid aAb$

$A \rightarrow cA \mid c$



(15) $a^n b^n c^m d^m \mid n,m \geq 1$

If power occurs in pair then stay together

$S \rightarrow AB$

$A \rightarrow aAb \mid ab$

$B \rightarrow cBd \mid cd$

(16) $a^n b^n c^n \mid n \geq 1$

$S \rightarrow asbc \mid abc$

Grammar Not possible for this pattern

(17) $a^n b^{2n} \mid n \geq 1$

$S \rightarrow asbb \mid abb$

(17) $a^n b^m c^m d^n \mid n,m \geq 1$

$S \rightarrow aSd \mid aAd$

$A \rightarrow bAC \mid bC$

(18) $a^n b^m c^n d^m \mid n,m \geq 1$ → No CFG can be possible

(19) $a^{m+n} b^m c^n \mid n,m \geq 1$

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

$S \rightarrow aSc \mid aAC$

$A \rightarrow aAb \mid ab$

(20) $a^n b^{n+m} c^n \mid n,m \geq 1$

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

$S \rightarrow AB$

$A \rightarrow aAb \mid ab$

$B \rightarrow bBC \mid bc$

(21) $a^n b^m c^{n+m} \mid n,m \geq 1$

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

$S \rightarrow aSc \mid aAC$

$A \rightarrow bAC \mid bc$

Types of Grammar and problems on Linear Grammar and Context free Grammar.

According to chomsky Grammar have 4 types

i) Type 3 ii) Type 2 3) Type 1 4) Type 0

Type 3 → Right Linear Grammar or Left Linear Grammar

$A \rightarrow \alpha B / \beta$ $\quad$ $A \rightarrow B\alpha \mid \beta$

$A, B \in V$, $\alpha, \beta \in T^*$ $\quad$ $A, B \in V$, $\alpha, \beta \in T^*$

Ex → $A \rightarrow aB$, $B \rightarrow aB \mid bB \mid a \mid b$ $\quad$ $A \rightarrow Ba$, $B \rightarrow Ba \mid Bb \mid a \mid b$

Ex → $A \rightarrow Ba \mid a$, $B \rightarrow aB \mid a$ → Combination of both so not a type-3.

Type-3 grammar generates always regular languages so called as regular grammar.

Finite Automata to Regular Grammar

(1)

FA

$A \rightarrow aB$
$B \rightarrow aB \mid bB \mid \varepsilon$] Right-linear Grammar (Regular)

Language → Strings starts with 'a'.

On reversing the Grammar becomes Left-Linear Grammar as

$A \rightarrow Ba$
$B \rightarrow Ba \mid Bb \mid \varepsilon$

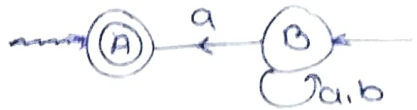
Language → Strings end with 'a'.

FA → RLG —Reverse→ LLG (But this procedure to convert FA → LLG is not appropriate)
(L) (L) (L^R)

Appropriate procedure is-

FA —Reverse→ FA → RLG —Reverse→ LLG
(L) (L^R) (L^R) $(L^R)^R = L$

$(FA)^R$:



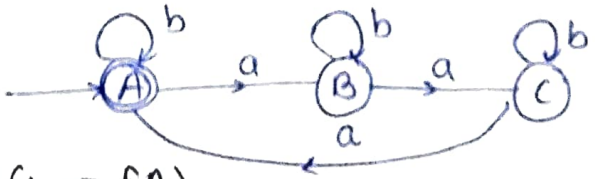
$\Rightarrow B \rightarrow aB \mid bB \mid aA$
$A \rightarrow \varepsilon$] RLG for Finite Automata

Reverse the production

$\Rightarrow B \rightarrow Ba \mid Bb \mid Aa$
$A \rightarrow \varepsilon$] Left Linear Grammar for Finite Automata

$B \rightarrow aA$ → It means B on transaction a going to state A.

(i) $A \rightarrow aB \mid bA \mid b$
$B \rightarrow aC \mid bB$
$C \rightarrow aA \mid bC \mid a$



(RLG → FA)

LLG ⟶ FA

LLG (L) $\xrightarrow{R}$ RLG (LR) ⟶ FA (LR) $\xrightarrow{R}$ FA (L)

Note

→ Regular Grammar and Finite Automata both are equivalent in power.

Q → What is the Language represented by given Grammar?

Soln On this type of questions create a Finite automata and find the Language.

Type-2 Grammar (Context free Grammar)

Productions are of form :-

$A \longrightarrow \alpha$

$A \in V, \ \alpha \in (V \cup T)^*$ ← Either V or T / combination of both V and T

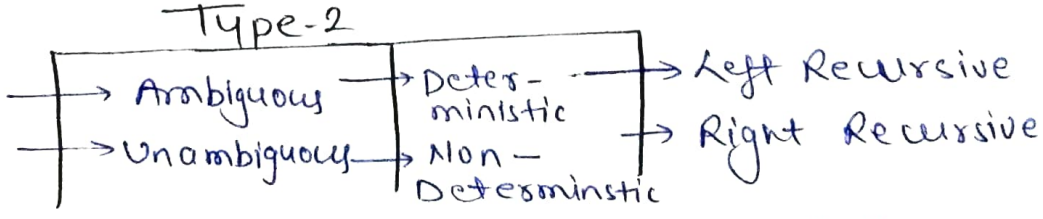
Note

→ If a grammar is regular then it must be context free but the converse is not always true.

Ex → $A \longrightarrow aAb \mid ab$

→ Machine that accept CFG is called PDA's.

Elimination of epsilon, unit production and useless symbols from CFG



→ Everything is discussed in Compiler Design Lectures.

$w \in L(G)$ → String generated by Language generated from Grammar.

If $w \in L(G)$ then you will get the answer of set from of productions in G in finite step

1 step → $S \to \alpha$ if α belongs to w then stop either go for 2 step productions

2 steps → $S \to aAb$
$\Rightarrow abb$] if you don't found the string here then go for 3 step Grammar

3 Step Grammar $S \to aAbb$
$\to aaBb$
$\to aabb$

Similary if $w \in L(G)$ then you may get the match in k steps but k can maximize upto unknown no so this Algorithm is a brute force.

Assume if $w \notin L(G)$ and we are checking steps discuss above then we will get stuck into an infinite loop.

To avoid this we can give length of string as $|w|$.
But assume that we reach upto string size n such than $n > |w|$ but still a production like $A \to \varepsilon$ is inside the set of productions and it may reduce the size at the end. So we have to modify the grammar as it doesn't contain and production like $A \to \varepsilon$ or $A \to B$ then every production will have atleast two symbols on its RHS it may leads to 'number of variables by 1' or 'string length by 1' increases.

Algo that doesn't contain $A \to \varepsilon$ and $A \to B$ is of Order $O(P^{2|w|+1})$ exponential.

P → no of productions $|w|$ → length of string

CYK Algorithm → Polynomial $O(|w|^3)$

Elimination of ε-productions

We can't eliminate ε-production in a case if $\frac{\varepsilon \in L(G)}{(S \to \varepsilon)}$ otherwise we can eliminate.

Ex $S \to asb \mid aAb$

$A \to \varepsilon$

Step-1 finout all null production and nullable variable.

Nullable variable is a variable that can generate ε.

Step-2 After finding Nullable variable on RHS of production then write it without or with nullable productions. as —

Here nullable variable is A then

$S \to asb \mid aAb \mid ab$

Replace by ε to Nullable variable

Ex-② $S \to AB$

$A \to aAA \mid \varepsilon$ Nullable = {A, B, S}

$B \to bBB \mid \varepsilon$

Here $\varepsilon \in L(G)$ so we can't remove all null productions.

$S \to AB \mid B \mid A \mid \varepsilon$ → Null (ε) can be present only is starting production in set of productions.

$A \to aAA \mid aA \mid a$

$B \to bBB \mid bB \mid b$

③ $S \to AbaC$

$A \to BC$

$B \to b \mid \varepsilon$

$C \to d \mid \varepsilon$

$D \to d$

Nullable Symbols {A, B, C}

$S \to AbaC \mid bac \mid Aba \mid ba$

$A \to BC \mid B \mid C$

$B \to b$

$C \to D$

$D \to d$

Sometimes eliminating ε-production may leads to unit production and both are useless.
So, don't consider unit production in valid CFG.

Elimination of Unit Production

Ex → S → Aa | B
B → A | bb
A → a | bc | B

Grammar without Unit Productions
S → Aa | bb | a | bc
B → bb | a | bc
A → a | bc | bb

If we eliminate S → B then we may loose chance to generate bb as S → B → bb then add bb.

Then
S → B → A → a, bc
B → A → a, bc
B → A → B No need to add anything
A → B → bb

add 'bb'

So, we recover all the productions in the eliminated unit production sets.

(2) S → AB
A → a
B → c | b
C → D
D → E
E → a

B → ~~C~~ → D → E → a
C → ~~D~~ → E → a
D → ~~E~~ → a

Remove unit productions and must include the symbol that (unit reachable)

S → AB
A → a
B → b | a
D → a
C → a
E → a

Ans//

Elimination of useless symbol

Symbol is useful if it is reachable from start state and it is deriving the string with terminals, means reachability & derivability.

Ex → $S \rightarrow AB \mid a$, $A \rightarrow BC \mid b$, $B \rightarrow aB \mid C$, $C \rightarrow aC \mid B$

$T = \{a, b\}$

$V = \{A, B, C, S\}$

Note imp

Useful symbols = { a, b, S, A }

Contains →
- terminals
- LHS of production contains terminals on RHS
- If any other production RHS is derivable from above two points then its LHS is to be consider.

But here other two are

$B \rightarrow aB \mid C$
$C \rightarrow aC \mid B$

] RHS doesn't derivable from above two so both are useless symbol

So delete all the production contain B and C on LHS and remove B and C from RHS in useful symbol

so, production becomes

$S \rightarrow a$ [AB removed due to B]

$A \rightarrow b$ [BC removed due to B & C]

And since A is not reachable so it will get delete too. So final set of production will contain only a single production as,

$S \rightarrow a$

Ex. $S \rightarrow AB \mid Ae$

$A \rightarrow aAb \mid bAa \mid a \mid b$

$B \rightarrow bbA \mid aaB \mid AB$

$C \rightarrow abCA \mid aDb$

$D \rightarrow bD \mid ac$

$T = \{a, b\}$

$V = \{S, A, B, C, D\}$

Useful symbols = { a, b, A, B, S }

B → due to B → bbA

S → due to S → AB

C and D are not useful. So final set of production will be

S → AB

Both are reachable from starting state.

A → aAb | bAa | a

B → bbA | aaB | AB

Ex → S → ABC | BaB

A → aA | BaC | aaa

B → bBb | a

C → CA | AC

T = { a, b }

V = { S, A, B, C }

B → a

S → BaB

A → aaa

Useful symbol = { a, b, B, S, A }

C → unuseful symbol so remove it

S → BaB

A → aA | Baa → not reachable from starting symbol so remove it

B → bBb | a

So, final prodn are –

S → BaB

B → bBb | a .

CNF Introduction

CFG

- CNF (Chomsky NF)
- GNF (Greibach NF)