AKTU

B. Tech II-Year

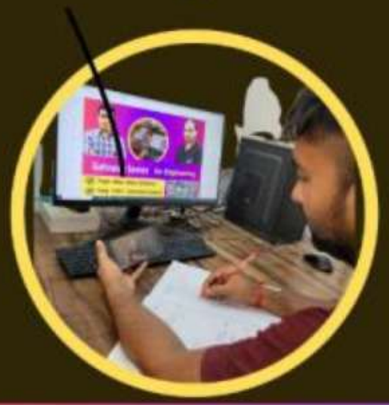
Gateway Classes

Semester -IV

CS IT & Allied Branches

BCS402 Theory of Automata and Formal Languages

UNIT-3 Regular and Non-Regular Grammars



Gateway Series for Engineering

- Topic Wise Entire Syllabus
- Long - Short Questions Covered
- AKTU PYQs Covered
- DPP
- Result Oriented Content

Download App

For Full Courses including Video Lectures

AKTU

B. Tech II-Year

Gateway Classes

GW GATEWAY CLASSES

BCS402 Theory of Automata and Formal Languages

Unit-3

Introduction to Regular and Non-Regular Grammars

Syallbus

(LINK IN DISCRIPTION)

Regular and Non-Regular Grammars: Context Free Grammar(CFG)-Definition, Derivations, Languages, Derivation Trees and Ambiguity, Regular Grammars- Right Linear and Left Linear grammars, Conversion of FA into CFG and Regular grammar into FA, Simplification of CFG, Normal Forms- Chomsky Normal Form(CNF), Greibach Normal Form (GNF), Chomsky Hierarchy, Programming problems based on the properties of CFGs.

For Full Courses including Video Lectures

Grammar

Grammar (Generator)

↓

Language

↓

FA

(finite automata) (Acceptor)

(NFA | DFA | ϵ-NFA)

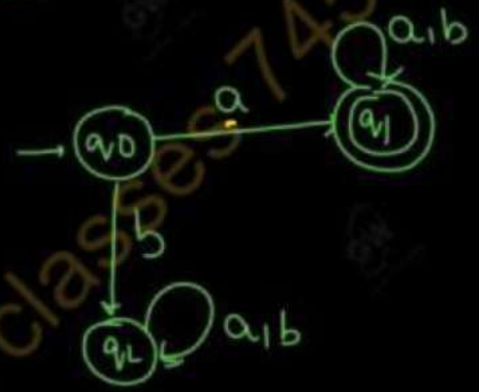
terminal a, b, c

Non-terminal A, B

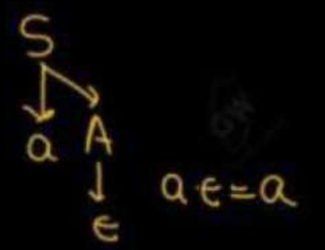
$\Sigma = \{a, b\}$

$L = \{a, ab, aaa, abab, \ldots\}$

start with a



$S \to aA$
$A \to aA \mid bA \mid \epsilon$



$a\epsilon = a$

aaaa

$S \to aA$
$\quad abA$
$\quad abaA$
$\quad ababA$
$\quad ababbA$
$\quad ababb\epsilon$

$S \to aA$
$\quad abA$
$\quad abbA$
$\quad abbbA$
$\quad abbb\epsilon$
$\quad abbb$

Grammar

A Grammar is a 4-tuple such that-

G = (V , T , P , S)

where-

V = Finite non-empty set of non-terminal symbols

T = Finite set of terminal symbols

P = Finite non-empty set of production rules

S = Start symbol

Example $L = \{w \mid \text{start with } a\}$

$S \rightarrow aA$

$A \rightarrow aA \mid bA \mid \epsilon$

S — Start Symbol — S

T — terminal — $\{a, b\}$

$V = \{S, A\}$

P

$S \rightarrow aA$

$A \rightarrow aA$

$A \rightarrow bA$

$A \rightarrow \epsilon$

$L = \{w \mid \text{start with } bb\}$

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

$B \rightarrow bbA$

$A \rightarrow aA \mid bA \mid \epsilon$

B

$\{a, b\}$

$\{B, A\}$

$B \rightarrow bbA$

$A \rightarrow aA$

$A \rightarrow bA$

$A \rightarrow \epsilon$

A Grammar is mainly composed of two basic elements

Terminal Symbols-

- Terminal symbols are those which are the constituents of the sentence generated using a grammar.
- Terminal symbols are denoted by using small case letters such as a, b, c etc.

Non-Terminal Symbols-

- Non-Terminal symbols are those which take part in the generation of the sentence but are not part of it.
- Non-Terminal symbols are also called as **auxiliary symbols** or **variables**.
- Non-Terminal symbols are denoted by using capital letters such as A, B, C etc.

QUESTIONS

Write a grammar having equal number of a and b

Context free Grammar

$S \rightarrow aSbS \mid bSaS \mid \epsilon$

$V = \{S\}$

$T = \{a, b\}$

$P = \{S \rightarrow aSbS, S \rightarrow bSaS, S \rightarrow \epsilon\}$

$S = \{S\}$

Write a grammar generate of string of balanced parenthesis

CFG

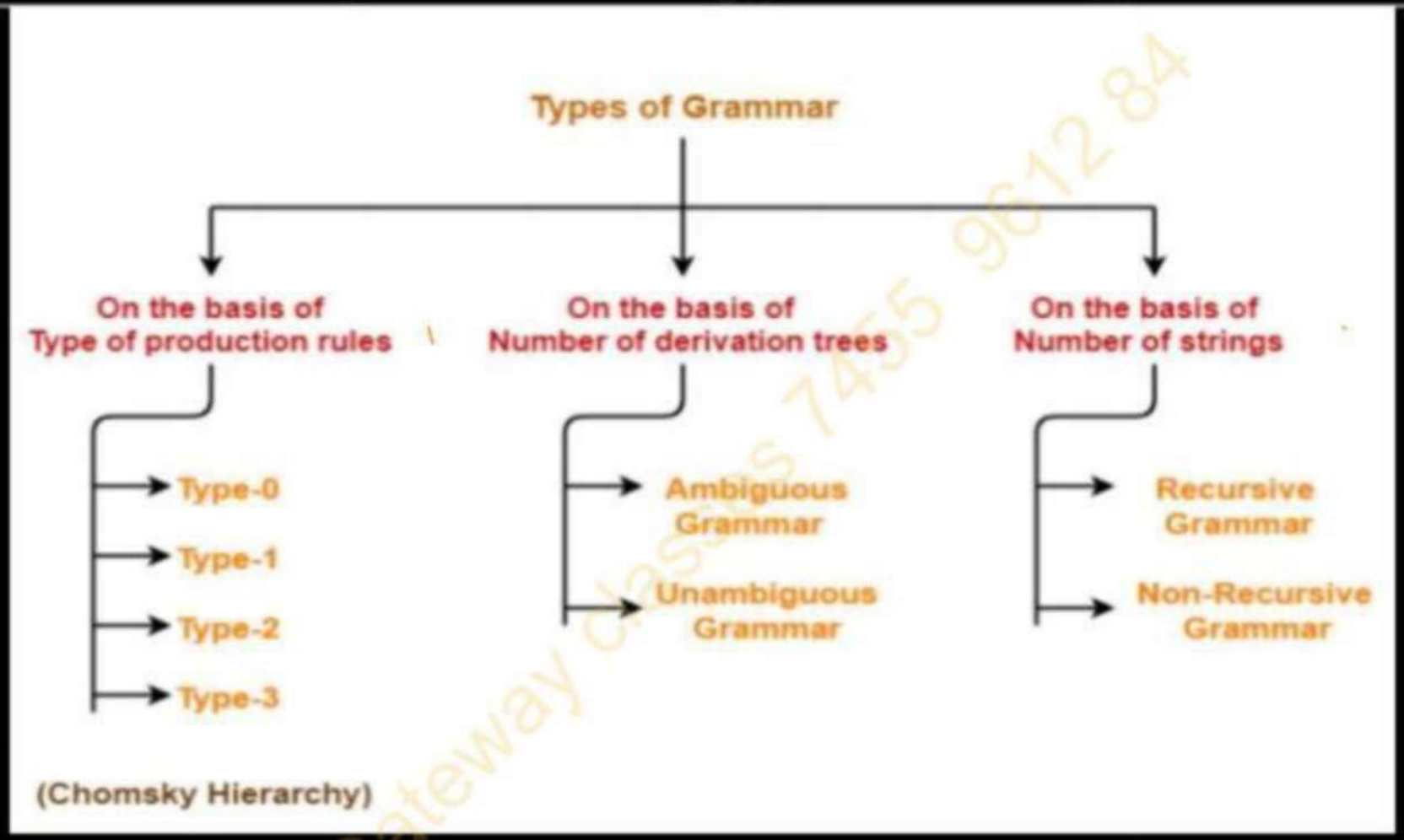
$S \rightarrow (S) \mid \epsilon \mid SS$

$V = \{S\}$

$T = \{(,)\}$

$P = \{S \rightarrow (S), S \rightarrow \epsilon, S \rightarrow SS\}$

$S = \{S\}$



Chomsky classification of grammar/ Chomsky hierarchy(NOAM CHOMSKY 1956)

GRAMMAR TYPE	GRAMMAR ACCEPTED	LANGAUGE ACCEPTED	AUTOMATION
Type-0	Unrestricted Grammar Recursive enumerable Grammar phase structure Grammar Non-contracting Grammar	Recursive enumerable language	Turing machine
Type-1	Context sensitive grammar (length increasing Grammar)	Context sensitive language	Linear bound automata
Type-2	Context free grammar	Context free language	Push down automata
Type-3	Regular grammar	Regular language	Finite automata

QUESTIONS

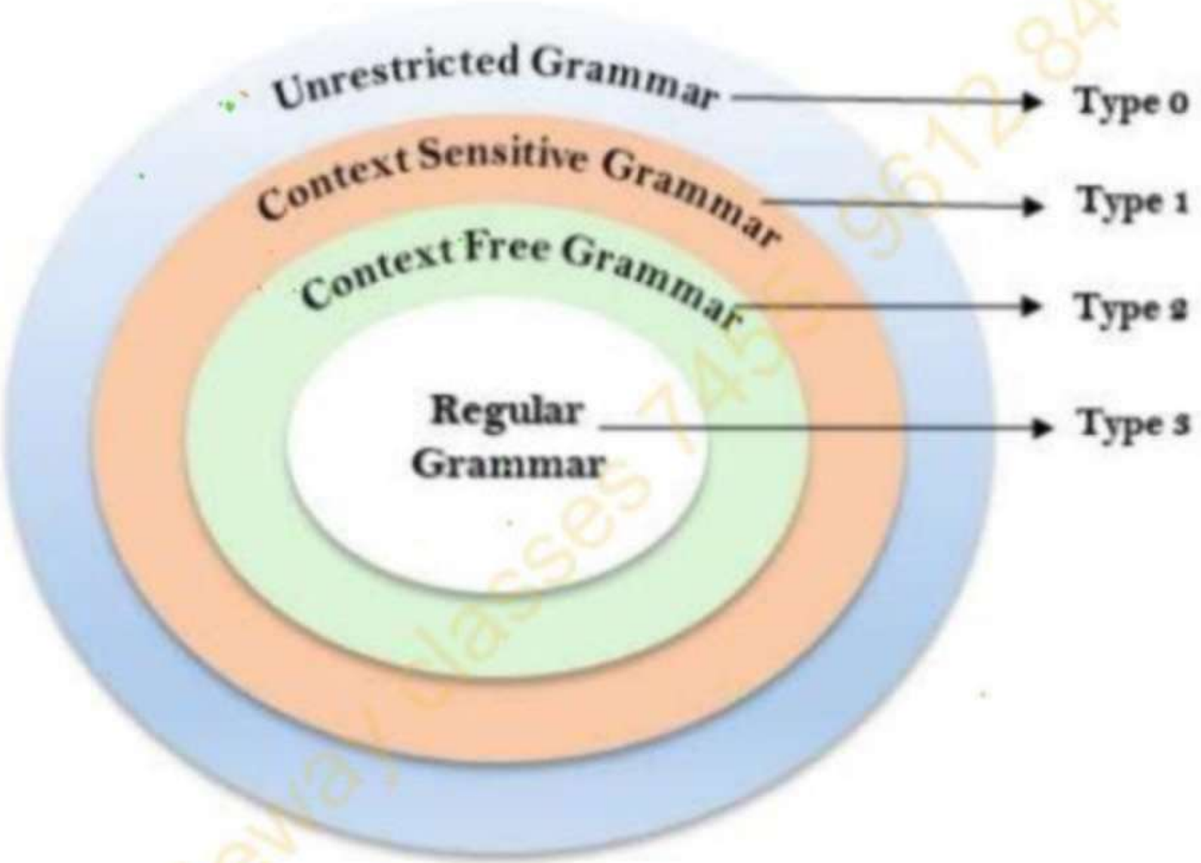


Fig: Chomsky Hierarchy

Production Rule

type-0

$\alpha \longrightarrow \beta$

$\alpha \in (VUT)^* V (VUT)^*$

V → Non-terminal
T → terminal

$\beta \in (VUT)^*$

$A \rightarrow aBa$ $\quad AB \rightarrow C$

$aAab \rightarrow aBaa$

$A \rightarrow \epsilon$

$ABB \rightarrow abb$

$Aaa \rightarrow abbC$

type-1

$\alpha \longrightarrow \beta$

$\alpha \in (VUT)^* V (VUT)^*$

$\beta \in (VUT)^+$

$|\alpha| \leq \beta$

$AB \rightarrow ABaa$ ✓ $\quad |AB| \leq |ABaa|$
$2 \leq 4$

$Aaa \rightarrow AaBaa$ -

$ABB \rightarrow aA$ X

$aAa \rightarrow ABBBB$ ✓

$Aa \rightarrow \epsilon$ X

S → asa S

type-2

$\alpha \rightarrow \beta$ $\quad |\alpha| = 1$

$\alpha \in V$

$\beta \in (V \cup T)^*$

$A \rightarrow aA$

$B \rightarrow BB$

$C \rightarrow aaa$

$XAA \rightarrow CAb$

type 3

Left linear Gramm

Right linear Gram

Left linear Grammar

$A \rightarrow a/Ba$

$A, B \in V \quad |A| = |B| = 1$

$a \in T^*$

$A \rightarrow a$

$A \rightarrow Ba$

$A \rightarrow a$

$A \rightarrow aB$

$A, B \in V$

$|A| = |B| = 1$

$a \in T^*$

Context free Grammar

A context free Grammar is a 4-tuple such that-

Z = (V , T , P , S)

where-

V = Finite non-empty set of non-terminal symbols

T = Finite set of terminal symbols

P = Finite non-empty set of production rules

S = Start symbol

G -> (VUT)*, where G ∈ V |G|=1

QUESTIONS

Construct a CFG for the language

1 $a^n b^n$, n>=0

$S \rightarrow asb \mid \wedge$

$S \rightarrow asb \mid \varepsilon$

V = {S}
T = {a, b}
P = {S → asb, S → ε}
S = {S}

2 $a^n b^n$, n>=1

$S \rightarrow asb \mid ab$

V = {S}
T = {a, b}
P = {S → asb, S → ab}
S = {S}

3 $a^n b^{n+2}$, n>=0

$S \rightarrow asb \mid bb$

V = {S}
T = {a, b}
P = {S → asb, S → bb}
S = {S}

4 $a^{2n} b^n$, n>=0

$S \rightarrow aasb \mid \varepsilon$

V = {S}
T = {a, b}
P = {S → aasbS, S → ε}
S = {S}

5 $a^{2n+3} b^n$, n>=0

$S \rightarrow aasb \mid aaa$

V = {S}
T = {a, b}
P = {S → aasb, S → aaa}
S = {S}

QUESTIONS

Construct a CFG for the language

6 $a^m b^n$, m, n>=0, m>n

$S \to AS_1$

$S_1 \to aS_1b \mid \varepsilon$] Equal No of a & b

$A \to aA \mid a$ ← Extra a add kiya

$V = \{S, S_1, A\}$

$T = \{a, b\}$

$P = \{S \to AS_1,\ S_1 \to aS_1b \mid \varepsilon,\ A \to aA \mid a\}$

$S = \{S\}$

7 $a^m b^n$, m, n>=0, m>=n

$S \to AS_1$

$S_1 \to aS_1b \mid \varepsilon$

$A \to aA \mid \varepsilon$

$V = \{S, A, S_1\}$

$T = \{a, b\}$

$P = \{S \to AS_1,\ S_1 \to aS_1b \mid \varepsilon,\ A \to aA \mid \varepsilon\}$

$S = \{S\}$

8 $\{w \mid n_a(w) = n_b(w)\}$

$S \to aSbS \mid bSaS \mid \varepsilon$

$V = \{S\}$ $P = \{S \to aSbS,\ S \to bSaS,\ S \to \varepsilon\}$

$T = \{a, b\}$ $S = \{S\}$

9 $ww^rUw(a+b)w^r$

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

$V = \{S\}$

$T = \{a\}$

$P = \{S \to aSa,\ S \to bSb,\ S \to \varepsilon,\ S \to a,\ S \to b\}$

$S = \{S\}$

10 $a^m b^m c^n$, n>=0 m>=0

$S \to S_1C$

$S_1 \to aS_1b \mid \varepsilon$

$C \to cC \mid \varepsilon$

$V = \{S, S_1, C\}$

$T = \{a, b\}$

$P = \{S \to S_1C,\ S_1 \to aS_1b,\ S_1 \to \varepsilon,\ C \to cC,\ C \to \varepsilon\}$

$S = \{S\}$

Q WW^r $T=\{a,b\}$

Even number of palindrome

$S \to asa \mid bsb \mid \varepsilon$

$V = \{S\}$
$T = \{a,b\}$
$P = \{S \to asa,\ S \to bsb,\ S \to \varepsilon\}$
$S = \{S\}$

Q $w(a+b)w^r$

Odd number palindrome

$S \to asa \mid bsb \mid a \mid b$

$V = \{S\}$
$T = \{a,b\}$
$P = \{S \to asa,\ S \to bsb,\ S \to a,\ S \to b\}$
$S = \{S\}$

QUESTIONS

Construct a CFG for the language

11 L={WCW^R | Where W belong (a , b)*}

$S_1 \rightarrow aS_1a \mid bS_1b \mid C$

V = { S_1 }
T = { a, b, c }
P = { $S_1 \rightarrow aS_1a$, $S_1 \rightarrow bS_1b$, $S \rightarrow C$ }
S = { S_1 }

12 Having any number of a over {a}

$S \rightarrow aS \mid \varepsilon$

V = {S}
T = {a}
P = $S \rightarrow aS \mid S \rightarrow \varepsilon$
S = {S}

13 L={$a^n b^m c^k$: n=m Or m<=k}

$S \rightarrow TC \mid AR$
$T \rightarrow aTb \mid \lambda$
$C \rightarrow Cc \mid \lambda$
$A \rightarrow Aa \mid \lambda$
$R \rightarrow bRC \mid C$

V = {S, T, C, A, R}
T = {a, b, c}

S = { S }

P = {$S \rightarrow TC$, $S \rightarrow AR$,
$T \rightarrow aTb$, $T \rightarrow \lambda$
$C \rightarrow Cc$; $A \rightarrow Aa$
$C \rightarrow \lambda$; $A \rightarrow \lambda$
$R \rightarrow bRC$
$R \rightarrow C$ }

QUESTIONS

14 $L = \{a^n b^m c^k \mid n=m \text{ or } m \; m \; != k\}$

$S \to TC \mid AR$
$T \to aTb \mid \varepsilon$
$C \to Cc \mid \varepsilon$
$A \to Aa \mid \varepsilon$
$R \to bRC \mid X \mid Y$
$X \to Xb \mid b$
$Y \to Yc \mid c$

$V = \{S, T, C, A, R\}$
$T = \{a, b, c\}$
$S = \{s\}$

$P = \{ S \to TC,\ S \to AR$
$T \to aTb,\ T \to \varepsilon$
$C \to Cc,\ C \to \varepsilon$
$A \to Aa,\ A \to \varepsilon$
$R \to bRC,\ R \to x$
$R \to Y$
$X \to Xb,\ X \to b$
$Y \to Yc$
$Y \to c \}$

15 $L = L = \{a^n b^m c^k \mid K = n+m\}$

$S \to asc \mid T$
$T \to bTC \mid \varepsilon$

$V = \{S, T\}$
$T = \{a, b, c\}$
$P = \{S \to asc,\ S \to T,\ T \to bTc,$
$T \to \varepsilon\}$
$S = \{s\}$

16 $L = \{a^n b^m c^k \mid n+2m=k$

$S \to asc \mid T$
$T \to bTcc \mid \varepsilon$

$V = \{S, T\}$
$T = \{a, b, c\}$
$P = \{S \to asc,\ S \to T,$
$T \to bTcc,\ T \to \varepsilon$
$S = \{S\}$

QUESTIONS

Construct a CFG for the language

1 $a^n b^m c^k$ $k=|n-m|$

$k+m=n$ or $k+n=m$

$S \to S_1 | S_2$

$S_1 \to aS_1c | T$

$T \to aTb | \varepsilon$

$S_2 \to XY$

$X \to aXb | \varepsilon$

$Y \to bYc | \varepsilon$

$V = \{ S, S_1, S_2, T, X, Y \}$

$T = \{a, b, c\}$

$S = \{S\}$

$P = \{ S \to S_1,\ S \to S_2$
$S_1 \to aS_1c \mid S_1 \to T$
$T \to aTb,\ T \to \varepsilon$
$S_2 \to XY$
$X \to aXb,\ X \to \varepsilon$
$Y \to bYc,\ Y \to \varepsilon \}$

2 $a^n b^n c^k$ $k>=3$

$S \to TC$

$T \to aTb | \varepsilon$

$C \to Cc | ccc$

$V = \{T, C, S\}$

$T = \{a, b, c\}$

$S = \{S\}$

$P = \{ S \to TC,\ T \to aTb,\ T \to \varepsilon,$
$C \to Cc,\ C \to ccc \}$

Derivation

- Derivation is a sequence of production rules. It is used to get the input string through these production rules. During parsing, we have to take two decisions.
- These are as follows:

1. We have to decide the non-terminal which is to be replaced.
2. We have to decide the production rule by which the non-terminal will be replaced

We have two options to decide which non-terminal to be placed with production rule.

1. Leftmost Derivation:

- In the leftmost derivation, the input is scanned and replaced with the production rule from left to right. So in leftmost derivation, we read the input string from left to right.

Example

E = E + E

E = E - E

E = a | b

Input

a - b + a

E	
E+E	E → E+E
E-E+E	E → E-E
a-E+E	E → a
a-b+E	E → b
a-b+a	E → a

E

E-E	E → E-E
a-E	E → a
a-E+E	E → E+E
a-b+E	E → b
a-b+a	E → a

2. Rightmost Derivation

- In rightmost derivation, the input is scanned and replaced with the production rule from right to left. So in rightmost derivation, we read the input string from right to left.

Example

E = E + E

E = E - E

E = a | b

Input

a - b + a

E	
E+E	E → E+E
E+a	E → a
E-E+a	E → E-E
E-b+a	E → b
a-b+a	E → a

Derive the string "abb" for leftmost derivation and rightmost derivation using a CFG given by

$S \rightarrow AB \mid \varepsilon$

$A \rightarrow aB$

$B \rightarrow Sb$ find leftmost and rightmost derivation

abb

(leftmost derivation)

S	
AB	$S \rightarrow AB$
aBB	$A \rightarrow aB$
aSbB	$B \rightarrow Sb$
aεbB	$S \rightarrow \varepsilon$
abB	
abSb	$B \rightarrow Sb$
abεb	$S \rightarrow \varepsilon$
abb	

Right most derivation

AB	$S \rightarrow AB$
ASb	$B \rightarrow Sb$
Aεb	$S \rightarrow \varepsilon$
Ab	
aBb	$A \rightarrow aB$
aSbb	$B \rightarrow Sb$
aεbb	$S \rightarrow \varepsilon$
abb	

Question2

Derive the string "aabbabba" for leftmost derivation and rightmost derivation using a CFG given by,

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

Right most derivation

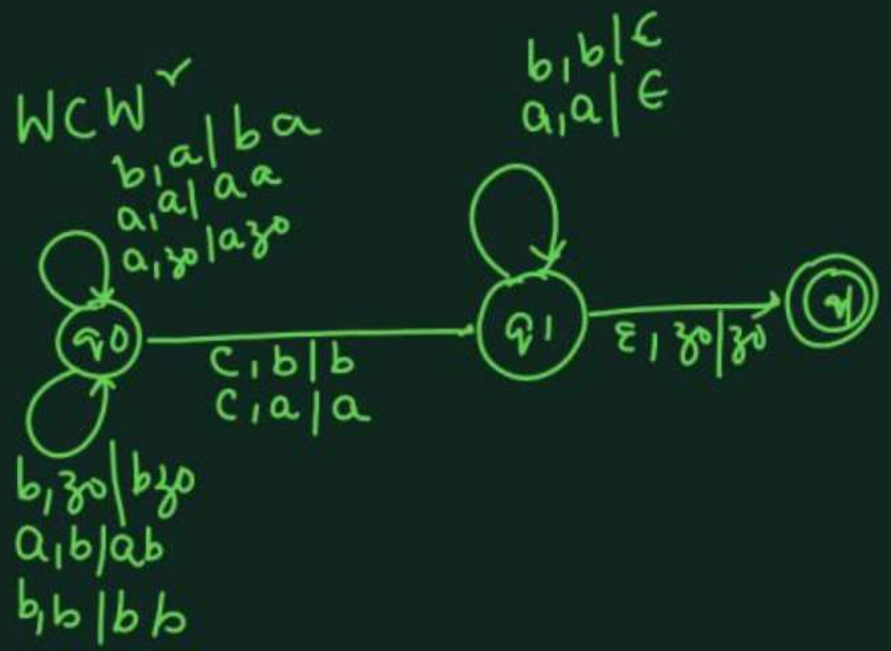
aB	$S \rightarrow aB$
aaBB	$B \rightarrow aBB$
aaBbS	$B \rightarrow bS$
aaBbbA	$S \rightarrow bA$
aaBbbba	$A \rightarrow a$
aabSbba	$B \rightarrow bS$
aabbAbba	$S \rightarrow bA$
aabbabba	$A \rightarrow a$

Leftmost derivation

aB	$S \rightarrow aB$
aaBB	$B \rightarrow aBB$
aabB	$B \rightarrow b$
aabbS	$B \rightarrow bS$
aabbaB	$S \rightarrow aB$
aabbabS	$B \rightarrow bS$
aabbabbA	$S \rightarrow bA$
aabbabba	$S \rightarrow a$

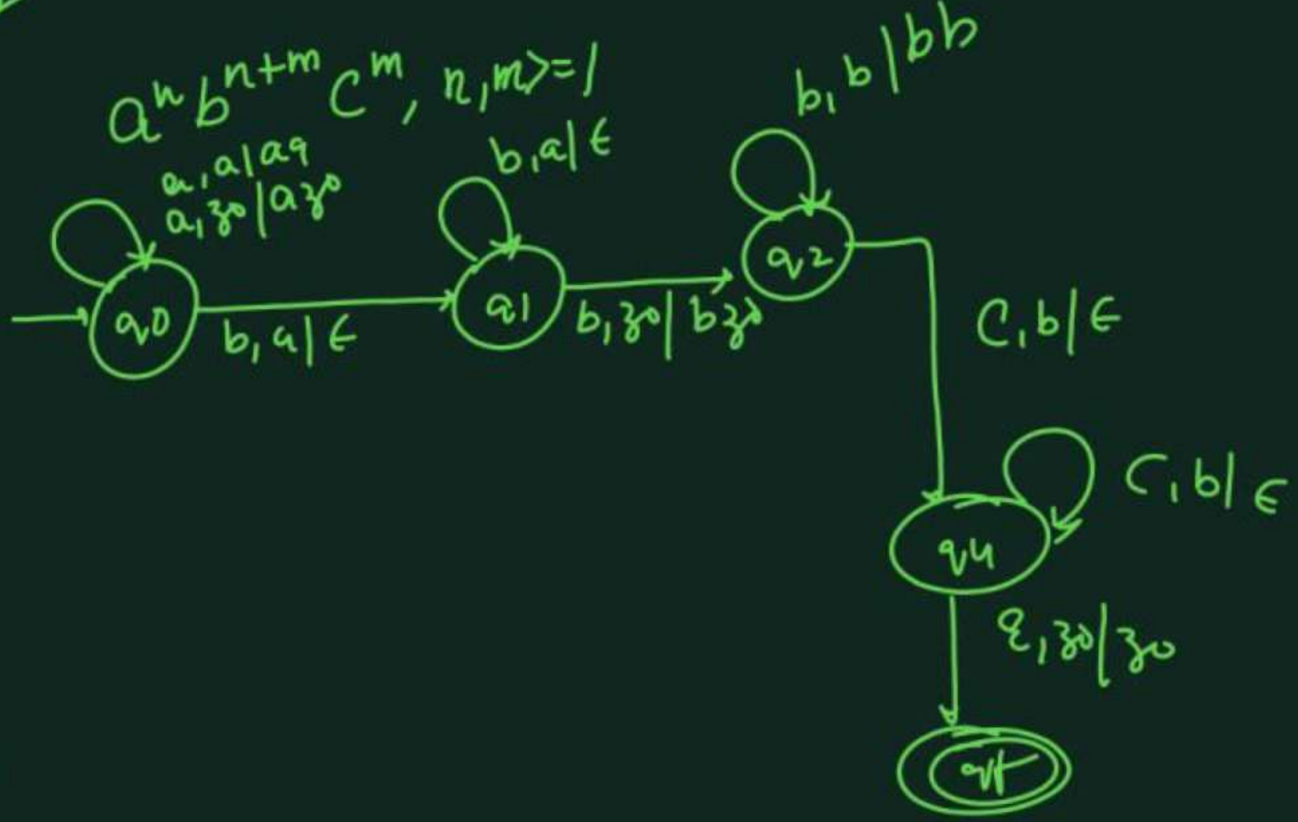
PDA

WCW^r



PDA

$a^n b^{n+m} c^m$; $n, m >= 1$



Derivation Tree(Parse tree)

- Derivation tree is a graphical representation for the derivation of the given production rules for a given CFG.
- It is the simple way to show how the derivation can be done to obtain some string from a given set of production rules.

NOTE

- Parse tree follows the precedence of operators. The deepest sub-tree traversed first.
- So, the operator in the parent node has less precedence over the operator in the sub-tree.

A parse tree contains the following properties:

- The root node is always a node indicating start symbols.
- The derivation is read from left to right.
- The leaf node is always terminal nodes.
- The interior nodes are always the non-terminal nodes.

Derivation Tree(Parse tree)

➢Example 1

Production rules:

E = E + E

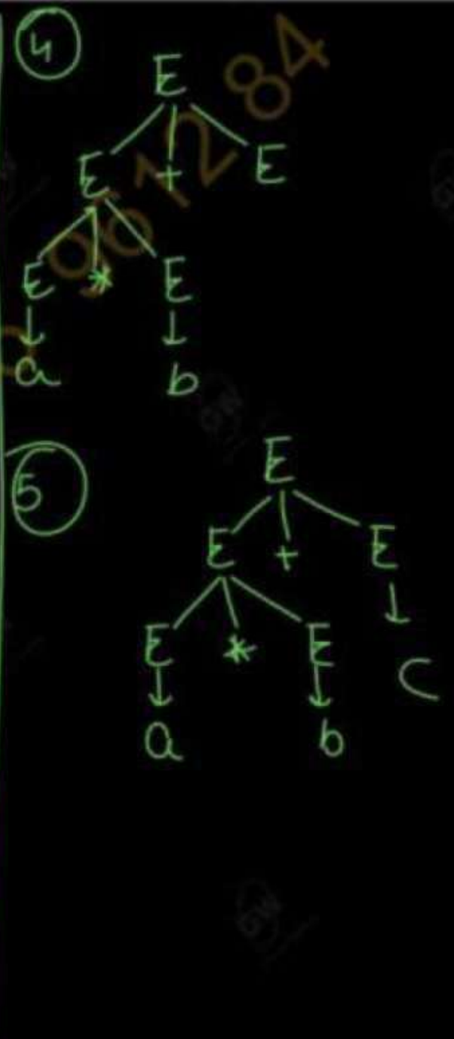
E = E * E

E = a | b | c

Input

a * b + c

* > + precedence



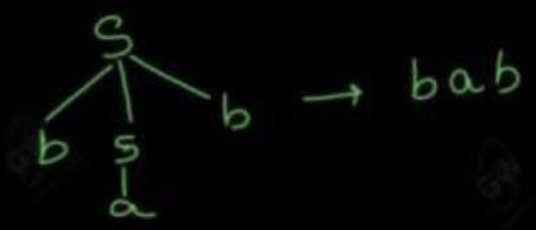
➢Example 2

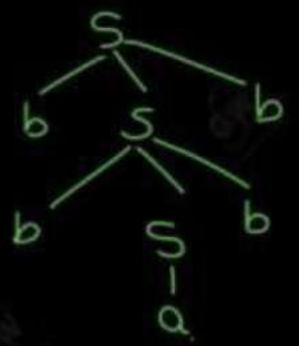
Production rules:

S → bSb | a | b

Draw a derivation tree for the string "bab" from the CFG given by

bbabb





bbabb

Parse (derivation tree)

➢ Example 3

Construct a derivation tree for the string aabbabba for the CFG given by,

S → aB | bA

A → a | aS | bAA

B → b | bS | aBB

S

aB S → aB

aaBB B → aBB

aabB B → b

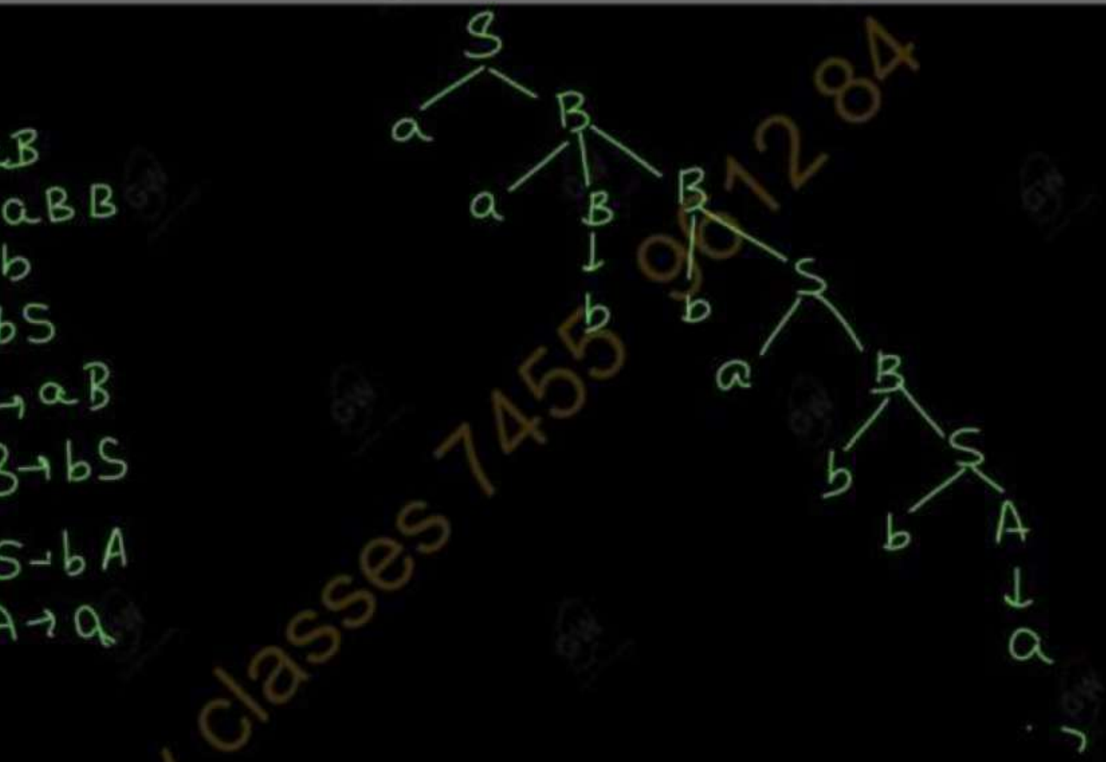
aabbS B → bS

aabbaB S → aB

aabbabS B → bS

aabbabbA S → bA

aabbabba A → a



Ambiguity Grammar / Ambigous Grammar

- A grammar is said to be ambiguous if there exists more than one leftmost derivation or more than one rightmost derivation or more than one parse tree for the given input string.
- If the grammar is not ambiguous, then it is called unambiguous.

↓ parse
Single left most derivation
Single Right most derivation

Example 1:

Let us consider a grammar G with the production rule

For the string "3 * 2 + 5", the above grammar can generate two parse trees by leftmost derivation:

$E \to I$
$E \to E + E$
$E \to E * E$
$E \to (E)$
$I \to \varepsilon \mid 0 \mid 1 \mid 2 \mid \ldots \mid 9$

E
E+E
E*E+E
I*E+E
3*E+E
3*I+E
3*2+E
3*2+I
3*2+5

left most derivation

(parse tree)

E E*E
I*E
3*E
3* E+E
3*I+E
3*2+E
3*2+I
3*2+5

left most derivation

(parse tree)

this shows that Grammar is Ambigious

Check whether the given grammar G is ambiguous or not.

$E \to E + E$

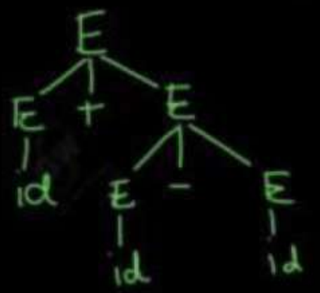
$E \to E - E$

$E \to id$

$id + id - id$

(left most derivation)

E	
E+E	E → E+E
id+E	E → id
id+E-E	E → E-E
id+id-E	E → id
id+id-id	E → id



Parse tree

(left most derivation) (Production Rule use)

E	
E-E	E → E-E
E+E-E	E → E+E
id+E-E	E → id
id+id-E	E → id
id+id-id	E → id

Parse tree

Ambiguous

Check whether the given grammar G is ambiguous or not.

$S \to aSb \mid SS$

$S \to \varepsilon$

input string
aabb

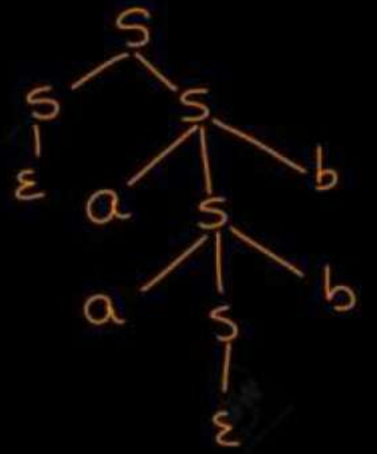
Right most derivation

S	
aSb	$S \to aSb$
aaSbb	$S \to aSb$
aaεbb	$S \to \varepsilon$
aabb	

Parse tree

Right most derivation

S	
SS	$S \to SS$
S aSb	$S \to aSb$
S aaSbb	$S \to aSb$
S aaεbb	$S \to \varepsilon$
S aabb	
ε aabb	$S \to \varepsilon$
aabb	



Ambiguous

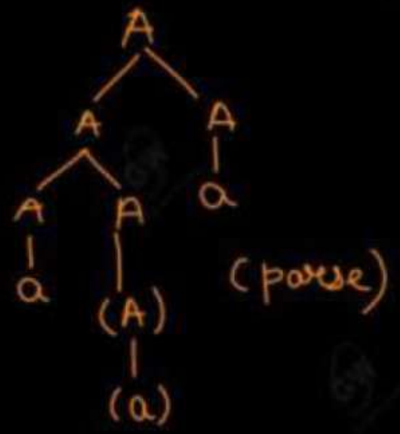
Check whether the given grammar G is ambiguous or not.

$A \to AA$
$A \to (A)$
$A \to a$

input a(a)a

Right most derivation

A	
AA	$A \to AA$
Aa	$A \to a$
AAa	$A \to AA$
A(A)a	$A \to (A)$
A(a)a	$A \to a$
a(a)a	$A \to a$



(parse)

A	
AA	$A \to AA$
AAA	$A \to AA$
AAa	$A \to a$
A(A)a	$A \to (A)$
A(a)a	$A \to a$
a(a)a	$A \to a$

Right most derivation

Ambiguous

Check whether the given grammar G is ambiguous or not.

$R \rightarrow R+R / R.R / R^* / a / b$

a.b+c

(Left most derivation)

R	
R+R	R → R+R
R.R+R	R → R.R
a.R+R	R → a
a.b+R	R → b
a.b+c	R → c

R	
R.R	R → R.R
a.R	R → a
a.R+R	R → R+R
a.b+R	R → b
a.b+c	R → c

2 Marks

transition function

$\delta(q, a) = p$

$\hat{\delta}(q, w) = p$

- Extended transition function

$w = 00111a$

- Last symbol
- Rest part

$\hat{\delta}(q, \varepsilon) = q$

$\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$

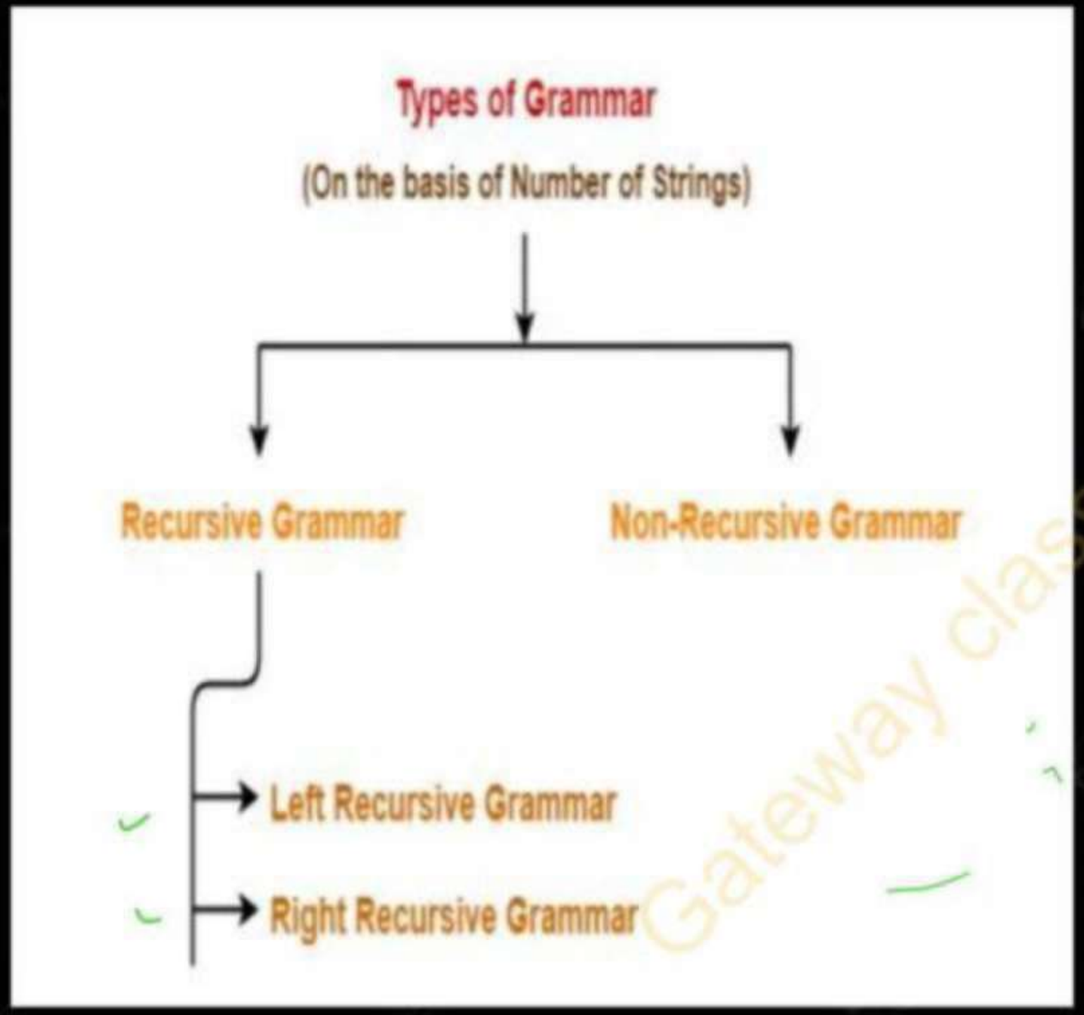
$\delta(z, a) = p$

$DFA = \{Q, \Sigma, F, q_0, \delta\}$

$L(A) = \{ w \mid \hat{\delta}(q_0, w) \in F \}$

Recursion-

Non-recursive grammar



- A grammar is said to be non-recursive if it contains no production that has the same variable at both its LHS and RHS.
- A grammar is said to be non-recursive if and only if it generates grammar generates finite number of strings, therefore it is a non-recursive grammar.
- A non-recursive grammar has neither left recursion nor right recursion

$S \rightarrow aA / bB$
$A \rightarrow a / b$
$B \rightarrow c / d$

- $L = \{ aa , ab , bc , bd \}$

Recursion-

- Recursion can be classified into following three types-

1. Left Recursion
2. Right Recursion
3. General Recursion

1. Left Recursion-

- A production of grammar is said to have **left recursion** if the leftmost variable of its RHS is same as variable of its LHS.
- A grammar containing a production having left recursion is called as Left Recursive Grammar.

EXAMPLE

$S \rightarrow Sa / \in$

Left Recursion-

- Left recursion is considered to be a problematic situation for Top down parsers.
- Therefore, left recursion has to be eliminated from the grammar.

Elimination of Left Recursion

- Left recursion is eliminated by converting the grammar into a right recursive grammar.
- If we have the left-recursive pair of productions-
- $A \rightarrow A\alpha / \beta 1 / \beta 2$
- (Left Recursive Grammar)
- where β does not begin with an A

Then, we can eliminate left recursion by replacing the pair of productions with-

$A \rightarrow \beta 1A' / \beta 2A'$

$A' \rightarrow \alpha A' / \epsilon$

2. Right Recursion-

- A production of grammar is said to have right recursion if the rightmost variable of its RHS is same as variable of its LHS.
- A grammar containing a production having right recursion is called as Right Recursive Grammar.
- $S \rightarrow aS / \in$

Right recursion does not create any problem for the Top down parsers.

Therefore, there is no need of eliminating right recursion from the grammar.

3. General Recursion-

The recursion which is neither left recursion nor right recursion is called as general recursion.

$S \rightarrow aSb / \in$

NOTE

- Left recursive grammar is not suitable for Top down parsers.
- This is because it makes the parser enter into an infinite loop.
- To avoid this situation, it is converted into its equivalent right recursive grammar.
- This is done by eliminating left recursion from the left recursive grammar.

INDIRECT LEFT RECURSION

A CFG is called indirect recursive if it has the production rules of the form

A → B1a/Z

B-> AB/b

PRACTICE PROBLEMS BASED ON LEFT RECURSION ELIMINATION-

Consider the following grammar and eliminate left recursion-

$A \rightarrow ABd / Aa / a$

$B \rightarrow Be / b$

$A \rightarrow \alpha A'$

$A' \rightarrow B\alpha A' | \alpha A' | \varepsilon$

$B \rightarrow bB'$

$B' \rightarrow eB' | \varepsilon$

Consider the following grammar and eliminate left recursion-

$E \rightarrow E + E / E \times E / a$

$E \rightarrow \alpha E'$

$E' \rightarrow +EE' | *EE' | \varepsilon$

PRACTICE PROBLEMS BASED ON LEFT RECURSION ELIMINATION-

Consider the following grammar and eliminate left recursion-

$E \rightarrow E + T / T$

$T \rightarrow T \times F / F$

$F \rightarrow id$

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow id$

Consider the following grammar and eliminate left recursion-

$S \rightarrow (L) / a$

$L \rightarrow L, S / S$

$S \rightarrow (L) \mid a$

$L \rightarrow SL'$

$L' \rightarrow ,SL' \mid \varepsilon$

PRACTICE PROBLEMS BASED ON LEFT RECURSION ELIMINATION-

Consider the following grammar and eliminate left recursion-

$A \rightarrow Ba / Aa / c$

$B \rightarrow Bb / Ab / d$

left Recursion Remove

$A \rightarrow BaA' | cA'$

$A' \rightarrow aA' | \varepsilon$

$B \rightarrow Bb | Ab | d$

Substitute value of A in $B \rightarrow Ab$

$A \rightarrow BaA' | cA'$

$A' \rightarrow aA' | \varepsilon$

$B \rightarrow Bb | BaA'b | cA'b | d$

$A \rightarrow BaA' | cA'$

$A' \rightarrow aA' | \varepsilon$

$B \rightarrow cA'bB' | dB'$

$B' \rightarrow bB' | aA'bB' | \varepsilon$

PRACTICE PROBLEMS BASED ON LEFT RECURSION ELIMINATION-

Consider the following grammar and eliminate left recursion-

$X \rightarrow XSb / Sa / b$ — (1)

$S \rightarrow Sb / Xa / a$ — (2)

$X \rightarrow SaX' | bX'$
$X' \rightarrow SbX' | \varepsilon$
$S \rightarrow Sb | Xa | a$

Eliminating left Recursion from equation (1)

Step 2 : put value of X in $S \rightarrow Xa$

$X \rightarrow SaX' | bX'$
$X' \rightarrow SbX' | \varepsilon$
$S \rightarrow Sb | SaX'a | bX'a | a$

Remove the left Recursion

$X \rightarrow SaX' | bX'$
$X' \rightarrow SbX' | \varepsilon$
$S \rightarrow bX'aS' | aS'$
$S' \rightarrow bS' | aXaS' | \varepsilon$

---

(1) Apply

$A \rightarrow AA\alpha | \beta$

$A \rightarrow \beta Z$
$Z \rightarrow A\alpha Z | \varepsilon$

PRACTICE PROBLEMS BASED ON LEFT RECURSION ELIMINATION-

Consider the following grammar and eliminate left recursion-

$S \to Aa / b$

$A \to Ac / Sd / \epsilon$

No left Recursion

$S \to Aa | b$

$A \to Ac | Sd | \epsilon$

Substitute S in $S \to Sd$

$S \to Aa | b$

$A \to Ac | Aad | bd | \epsilon$

Remove left Recursion

$S \to Aa | b$

$A \to bdA' | A'$

$A' \to cA' | adA' | \epsilon$

Ques

$S \to Aa | b$

$A \to Ac | Z$

$Z \to Ab | b$

$S \to Aa | b$

$A \to ZA'$

$A' \to cA' | \epsilon$

$Z \to Ab | b$

Put A value in $Z \to Ab$

$S \to Aa | b$

$A \to ZA'$

$A' \to cA' | \epsilon$

$Z \to ZA'b | b$

Remove left Recursion

$S \to Aa | b$

$A \to ZA'$

$A' \to cA' | \epsilon$

$Z \to bZ'$

$Z' \to AbZ' | \epsilon$

Consider the ambiguous grammar

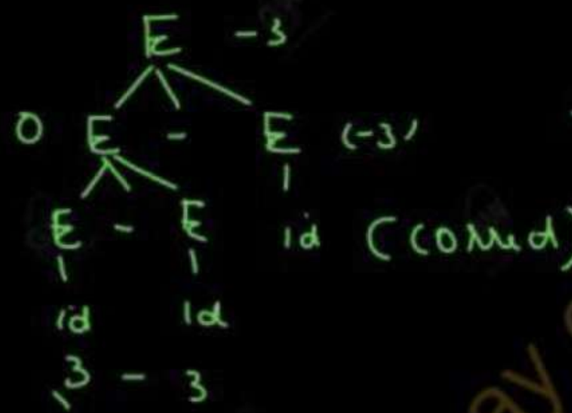
E -> E-E | id

id - id - id left to Right

id = 3

3 - 3 - 3

0 - 3 = -3



→ Ambiguity is a problem bcz it breaks the rules of Associativity & precedence & it confuse the parse.

Removal of Ambiguity :

We can remove ambiguity solely on the basis of the following two properties –

Precedence –

If different operators are used, we will consider the precedence of the operators. The three important characteristics are :

- The level at which the production is present denotes the priority of the operator used.
- The production at **higher levels** will have **operators with less priority**. In the parse tree, the nodes which are at top levels or close to the root node will contain the lower priority operators.
- The production at **lower levels** will have operators with **higher priority**. In the parse tree, the nodes which are at lower levels or close to the leaf nodes will contain the higher priority operators.

2. Associativity –

If the same precedence operators are in production, then we will have to consider the associativity.

- If the associativity is left to right, then we have to prompt a left recursion in the production. The parse tree will also be left recursive and grow on the left side.

  +, -, *, / are left associative operators.

- If the associativity is right to left, then we have to prompt the right recursion in the productions. The parse tree will also be right recursive and grow on the right side.

  ^ is a right associative operator.

Convert the ambiguous grammar to unambiguous Grammar

- **E -> E-E | id**

$E \rightarrow E-P \mid P$ (Non-Ambiguous)

$P \rightarrow id$

(id-id-id)

id-id-id-id

Convert the ambiguous grammar to unambiguous Grammar

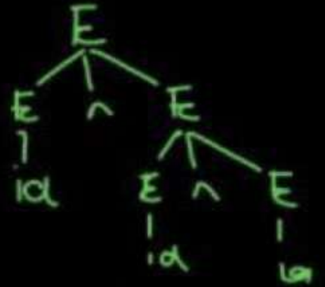
➢ E -> E^E | id

It must be Right Recursive

^ - Right Associative

id ^ id ^ id

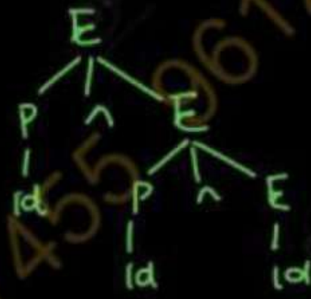
(Parse tree-1)
(left derivation)
(Wrong)



Parse tree-2
(left most derivation)
(Right)

(two parse tree Available So this is Ambiguous.

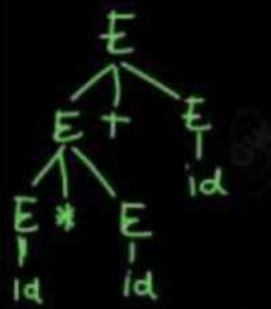
E → P^E | P
P → id



this become unambiguous Grammar

Convert the ambiguous grammar to unambiguous Grammar

➢ E -> E+E | E*E | id



(Parse tree-1)
(left most derivation)
(correct one)

(Parse tree-2)
(left most derivation)
(wrong)

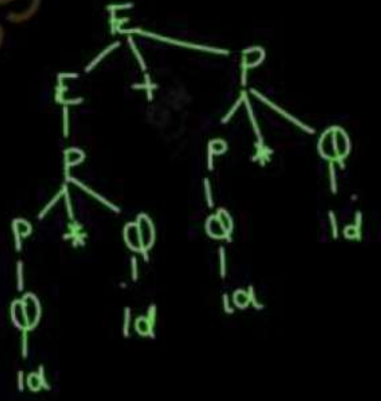
two parse tree possible using left most derivation

So this is Ambiguous Grammar.

+ * left to Right
So it Must be
left Recursive

E → E+P | P
P → P*Φ | Φ
Φ → id

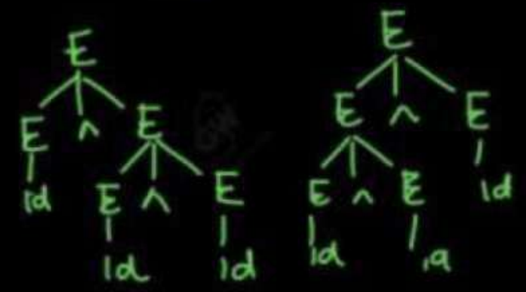
id*id+id*id



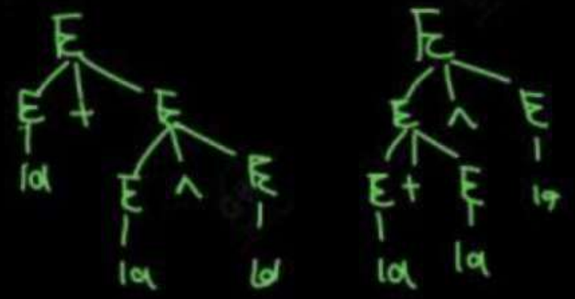
Convert the ambiguous grammar to unambiguous Grammar

➢ E -> E - E | E * E | E^E /id

id ∧ id ∧ id



id + id ∧ id



- < * < ∧ precedence

* - (left Associative) (left Recursive)

∧ (Right Associative) (Right Recursive)

(To parse the possible)

this is Ambiguous Grammar.

E → E - P / P
P → P * Q / Q
Q → Z ∧ Q / Z
Z → id

id - id * id ∧ id

Only one parse tree is possible so this is not a Ambiguous Grammar.

Convert the ambiguous grammar to unambiguous Grammar

➢ $R \rightarrow R+R / R.R / R^* / a / b$

+ . left Associative

$R \rightarrow R+T \mid T$
$T \rightarrow T.J \mid J$
$J \rightarrow J^* \mid Q$
$Q \rightarrow a \mid b$

(it is non-Ambigious Gram.

bexp → bexp or bexp / bexp and bexp / not bexp / T / F

where bexp represents Boolean expression, T represents True and F represents Fals

not > and > or (left Associative)

bexp → bexp or J | J
J → J and k | k
k → not k | M
M → T | F

Inherently Ambiguous Grammar

- An inherently ambiguous grammar is a context-free grammar (CFG) for which there exists at least one string that can have more than one distinct parse tree or derivation.
- This ambiguity means that no equivalent unambiguous grammar can generate the same language as the ambiguous grammar.

A classic example of an inherently ambiguous language is the language

$L=\{a^n b^m c^k | n=m \text{ or } m=k\}$.

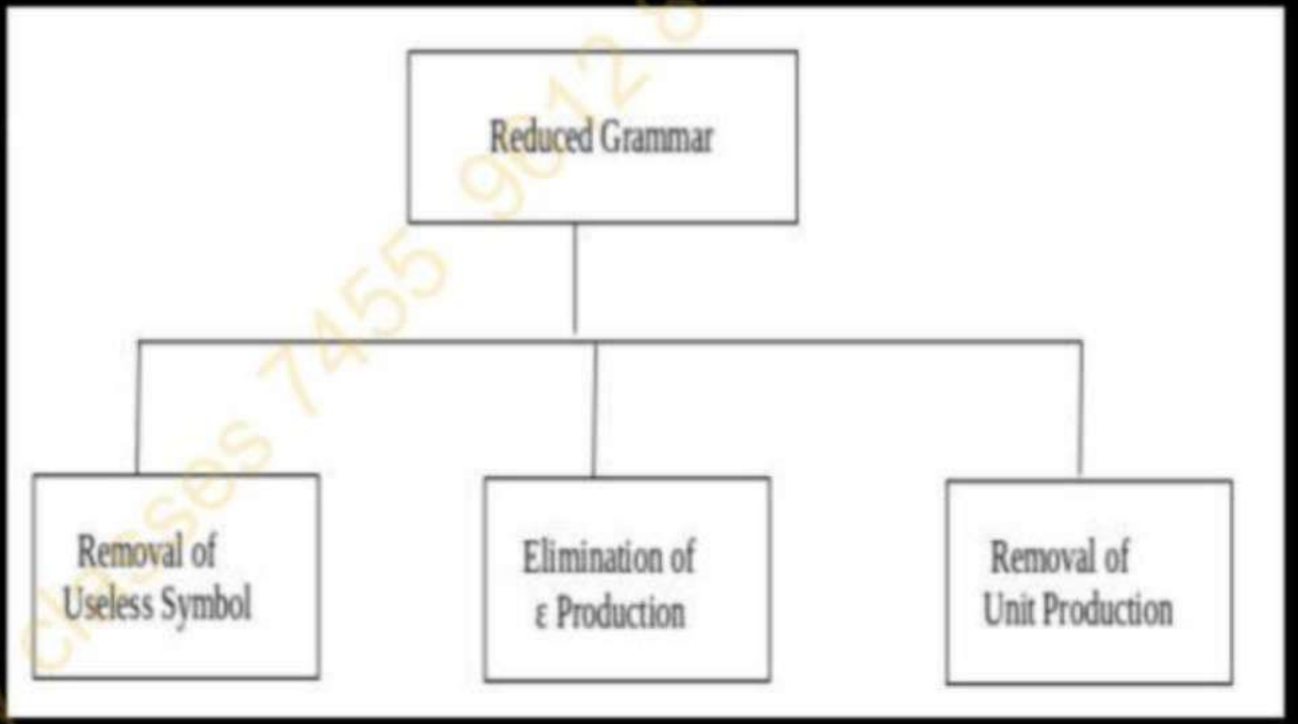
2 Marks Question

Simplification or minimization of CFG

- The process of deleting or eliminating of useless symbol, epsilon production and null production is called as simplification of CFG

NEED

- To make grammar more efficient and compiler friendly



Removal of useless symbol

- The variable which are not involved in the derivation of the string is known as useless symbol
- there are two ways of finding out and remove useless symbol-
- 1. Select the variable that can not be reached from the start symbol of the grammar and remove them along with their all production
- example1

S-> aAB

A->a

B->b

~~C->d~~

$\begin{cases} S \to aAB \\ A \to a \\ B \to b \end{cases}$

(Non-Reachable Symbol Remove)

2. Select the variable that are reachable from the start symbol but does not drive any terminal , remove this along with their production

Example 2 S->aA/aB

A->b

$S \to aA$
$A \to b$

(Non-Generating Symbol Removal)

Removal of epsilon production

The productions of type $S \rightarrow \varepsilon$ are called ε productions.

EXAMPLE 1

	After Removing $A-\varepsilon$	Removing $B \rightarrow \varepsilon$
S->AbB	$S \rightarrow AbB \mid bB$	$S \rightarrow AbB \mid bB \mid Ab \mid b$
A->a/ε	$A \rightarrow a$	$A \rightarrow a$
B-> b/ε	$B \rightarrow b \mid \varepsilon$	$B \rightarrow b$

Example 2:

	Remov $A \rightarrow \varepsilon$	Remove $B \rightarrow \varepsilon$	$S' \rightarrow S \mid \varepsilon$
S->AB	$S \rightarrow AB \mid B$	$S \rightarrow AB \mid B \mid A \mid \varepsilon$	$S \rightarrow AB \mid B \mid A$
A->a/ε	$A \rightarrow a$	$A \rightarrow a$	$A \rightarrow a$
B->b/ε	$B \rightarrow b \mid \varepsilon$	$B \rightarrow b$	$B \rightarrow b$

Example 3

S->aSb/ε Remove $S \rightarrow \varepsilon$

$S \rightarrow asb \mid ab$

Example 4:

	Remove $A \rightarrow \varepsilon$	Remove $B \rightarrow \varepsilon$
S->aAB	$S \rightarrow aAB \mid aB$	$S \rightarrow aAB \mid aB \mid aA \mid a$
A->aAA/ε	$A \rightarrow aAA \mid aA \mid a$	$A \rightarrow aAA \mid aA \mid a$
B->bBB/ε	$B \rightarrow bBB \mid \varepsilon$	$B \rightarrow bBB \mid bB \mid b$

Removal of unit production

The production of the form A->B Where A,B belong to V |A|=|B|= 1is known as unit production

EXAMPLE 1

S-Aa
A->a/B
B->d

$S \to Aa$
$A \to a|d$

Example 2

S->aAb
A->B/a
B->C/d
C->D/c
D->d

$S \to aAb$
$A \to B|a$
$B \to C|d$
$C \to d|C$

$S \to aAb$
$A \to a|d|a$

$S \to aAb$
$A \to a|d$
$B \to a|d$

EXAMPLE 3 $S \to 0A \mid 1B \mid C$

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

$S \to 0A|1B|01$
$A \to 0S|00$
$B \to 1/A$

$S \to 0A|1B|01$
$A \to 0S|00$
$B \to 1|0S|00$

consider the grammar for the simplification of production

S->aA/aBB
A->aaA/ε
B->bB/bbC
C->B

① Removal epsilon production

S→aA|a|aBB
A→aaA|aa
B→bB|bbC
C→B

② Useless production Removal

① Non-Generating symbol Removal

S→aA|a
A→aaA|aa

② Non-Reachable symbol Removal

S→aA|a
A→aaA|aa

Unit production Removal

→ No unit production

S→aA|a
A→aaA|aa

S->aC/SB
A->bSCa
B->aSB/bBC
C->aBC/ad

① ε-Removal

No epsilon production

S→aC|sB
A→bBCa
B→aSB|bBC
C→aBC|ad

② Useless production Removal

① Non-Generating Symbol Removal

S→aC
C→ad

② Non-Reachable Symbol Removal

S→aC
C→ad

No unit production

Simplified CFG

S→aC
C→ad

consider the grammar for the simplification of production

S->0A0/1B1/BB
A->C
B->S/A
C->S/ε Remove C→ε

S→0A0|1B1|BB
A→ε
B→S/A
C→S

Remove A→ε

S→0A0|00|1B1|BB
B→S|ε
C→S

Remov B→ε

S→0A0|00|1B1|11|B|ε
B→S
C→S

S→ε

S→0A0|00|1B1|11|B
B→ε
C→ε

Unit production Removal

S→0A0|00|1B1|11|ε
C→ε

Useless Symbol

① Non-Generating {0, 1, S}

S→00|11|ε
C→ε

Non Reachable

S→00|11|ε

S'→S|ε
S→00|11

consider the grammar for the simplification of production

S->ASB/ε

A->AaS/a

B->SbS/A/bb

Removal of epsilon

S→ ASB | AB

A→ AaS | Aa | a

B→ Sb | bS | SbS | b | A | bb

Removal of useless symbol

1. Non-Generating Symbol Removal

S→ AB

A→ Aa | a

B→ b | A | bb

2. Non-Reachable Symbol

S→ AB

A→ Aa | a

B→ b | A | bb

Removal of unit production

S→ AB

A→ Aa | a

B→ b | A | bb

$S \rightarrow AB$

$A \rightarrow Aa \mid a$

$B \rightarrow b \mid Aa \mid a \mid bb$

$B \rightarrow A$

(No-unit production)

CNF(Chomsky normal form)

- CNF stands for Chomsky normal form. A CFG(context free grammar) is in CNF(Chomsky normal form) if all production rules satisfy one of the following conditions:

1. A non-terminal generating two non-terminals. For example, S → AB.
2. A non-terminal generating a terminal. For example, S → a.
3. Start symbol generating ε. (e.g.; S-> ε)

G1 = {S → AB, S → c, A → a, B → b} CNF

G2 = {S → aA, A → a, B → c} XCNF

S → XA
A → a
B → c CNF
X → a

Steps for converting CFG into CNF

Step 1: Eliminate start symbol from the RHS. If the start symbol S is at the right-hand side of any production, create a new production as:

$S1 \rightarrow S$

Where S1 is the new start symbol

Step 2: In the grammar, remove the null, unit epsilon production

Useless Removal

Step 3: Eliminate RHS with more than two non-terminals.

For example, $S \rightarrow ASB$ can be decomposed as:

$S \rightarrow RS$

$R \rightarrow AS$

Eliminate terminals from the RHS of the production if they exist with other non-terminals or terminals. For example, production $S \rightarrow aA$ can be decomposed as:

$S \rightarrow RA$

$R \rightarrow a$

NOTE

- For a given grammar, there can be more than one CNF.
- CNF produces the same language as generated by CFG.
- Any Context free Grammar that do not have ε in it's language has an equivalent CNF

QUESTIONS

Convert the given CFG to CNF. Consider the given grammar G1:

S->ASA/aB

A_>B/S

B->b/ε

NOTE

ε, ^, ∧

New symbol introduce

S' → S
S → ASA | aB
A → B | S
B → b | ε

Remove epsilon symbol

B → ε

S' → S
S → ASA | aB | a
A → ε | S
B → b

S → AB
S → a

Remove A → ε

S' → S
S → ASA | SA | AS | S | aB | a
A → S
B → b

Remove Unit production

S' → S, S → S, A → S

Remove S → S

S' → S
S → ASA | SA | AS | aB | a
A → S
B → b

Remove S' → S

S' → ASA | SA | AS | aB | a
S → ASA | SA | AS | aB | a
A → S
B → b

Remove A → S

S' → ASA | SA | AS | aB | a
S → ASA | SA | AS | aB | a
A → AS | SA | ASA | aB | a
B → b

No useless symbol

Let X be SA

S' → AX | SA | AS | aB | a
S → AX | SA | AS | aB | a
A → AS | SA | AX | aB | a
B → b
X → SA

(let R → a)

S' → AX | SA | AS | RB | a
S → AX | SA | AS | RB | a
A → AS | SA | AX | RB | a
B → b
X → SA
R → a

(this Grammar is in CNF)

QUESTIONS

Convert the given CFG to CNF.
Consider the given grammar G1:

S → a | aA | B
A → aBB | ε
B → Aa | b

Starting symbol is not present at the Right Side of the production
So Step 1 is Skipped.

Remove epsilon/Null productions

Remove A → ε

S → a | aA | B
A → aBB
B → a | Aa | b

Remove unit production (S → B)

S → a | aA | b | Aa
A → aBB
B → a | Aa | b

Non-generate

No useless Symbol

S → a | XA | b | AX
A → XBB
B → a | AX | b
X → a

S → a | XA | b | AX
A → XZ
B → a | AX | b
X → a
Z → BB

I^st CNF

Both are correct

II^nd CNF

S → a | XA | b | AX
A → ZB
B → a | AX | b
X → a
Z → XB

Convert the given CFG to CNF.
Consider the given grammar G1:
S->aAbB
A->aA/a
B->bB/b

Starting Symbol is not on the Right Side of the production
So Step1 is skipped

Epsilon production Removal

No epsilon production

S→ aA bB
A→ aA|a
B→ bB|b

Null production Removal

→ No Null production

S→ aAbB
A→ aA|a
B→ bB|b

① Non generating Symbol
②

S→ XAYB
A→ XA|a
B→ YB|b
X→ a
Y→ b

S→ XC1
A→ XA|a
B→ YB|b
X→ a
Y→ b
C1→ AC2
C2→ YB

first way of writing CNF

Second way

S→ MN
A→ XA|a
B→ YB|b
X→ a
Y→ b
M→ XA
N→ YB

Thank You

Q $S \to aS|\varepsilon$
$A \to a$

① $S' \to S$
$S \to aS|\varepsilon$
$A \to a$

② Remove epsilon
$S' \to S$
$S \to aS|a$
$A \to a$

③ Remove unit production
$S' \to aS|a$
$S \to aS|a$
$A \to a$

④ Convert CNF
$S' \to ZS|a$
$S \to ZS|a$
~~$R \to a$~~
$Z \to a$

Q. $S \to aA|\varepsilon$
$A \to b$

$S \to zA|\varepsilon$
$A \to b$
$z \to a$

this is in CNF

Ques

$S \to aSa|B$
$B \to Aa|bb$
$A \to aAA|\varepsilon$

Convert it into CNF

H/W

Greibach Normal Form (GNF)

GNF stands for Greibach normal form. A CFG(context free grammar) is in GNF(Greibach normal form) if all the production rules satisfy one of the following conditions

- A start symbol generating ε. For example, S → ε.
- A non-terminal generating a terminal. For example, A → a.
- A non-terminal generating a terminal which is followed by any number of non-terminals. For example, S → aASB

G1 = {S → aAB | aB, A → aA| a, B → bB | b} GNF

G2 = {S → aAB | aB, A → aA | ε, B → bB | ε}

Remove A → ε

S → aAB|aB

A → aA|a

B → bB|ε

Remove B → ε

S → aAB|aB|aA|a GNF

A → aA|a

B → bB|b

Removal of Ambiguity :

Step 1. If the given grammar is not in CNF, convert it to CNF

Step 2. Change the names of non terminal symbols to A_1 till A_N in same sequence.

Step 3. Check for every production rule if RHS has first symbol as non terminal say A_j for the production of A_i, it is mandatory that i should be less than j. Not great and not even equal.

If i> j then replace the production rule of A_j at its place in A_i.

If i=j, it is the left recursion. Create a new state Z which has the symbols of the left recursive production, once followed by Z and once without Z, and change that production rule by removing that particular production and adding all other production once followed by Z.

Step 4. Replace very first non terminal symbol in any production rule with its production until production rule satisfies the above conditions.

QUESTION

- S->CA/BB
- B->b/SB
- C->b
- A->a

You can Skip Step 1

No Useless symbol

No epsilon, No unit production

$A \to BC$ All the production

$A \to a$ Rule is in the form

this Grammer is CNF

let

S(A_1) C(A_2) A(A_3) B A_4

$A_1 \to A_2A_3 / A_4A_4$

$A_4 \to b / A_1A_4$

$A_2 \to b$

$A_3 \to a$

$A_i \to A_j$ $i<j$

$A_1 \to A_2A_3 / A_4A_4$

$A_4 \to b / A_2A_3A_4 / A_4A_4A_4$

$A_2 \to b$

$A_3 \to a$

$A_1 \to A_2A_3 / A_4A_4$

$A_4 \to b / bA_3A_4 / A_4A_4A_4$

$A_2 \to b$

$A_3 \to a$

Left Recursion $4 < 4$ $i = j$

$A_1 \to A_2A_3 / A_4A_4$

$A_4 \to b / bA_3A_4 / bZ / bA_3A_4Z$

$Z \to A_4A_4Z / A_4A_4$

$A_2 \to b$

$A_3 \to a$

$A_1 \to bA_3 / bA_4 / bA_3A_4A_4 / bZA_4 / bA_3A_4ZA_4$

$A_4 \to b / bA_3A_4 / bZ / bA_3A_4Z$

$Z \to bA_4Z / bA_3A_4A_4Z / bZA_4Z / bA_3A_4ZA_4Z$

$bA_4 / bA_3A_4A_4 / bZA_4 / bA_3A_4ZA_4$

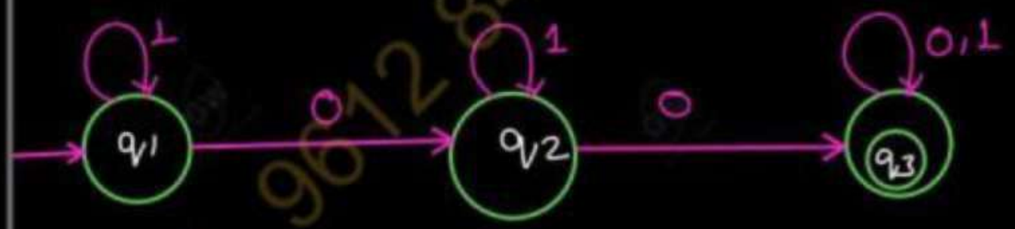
$A_2 \to b$

$A_3 \to a$

finite automata to CFG

$A \rightarrow aB$
$B \rightarrow aB \mid bB \mid \underline{\varepsilon}$

$V = \{A, B\}$
$T = \{a, b\}$
$P \{ A \rightarrow aB,\ B \rightarrow aB,\ B \rightarrow bB,\ B \rightarrow \varepsilon \}$
$S = \{A\}$



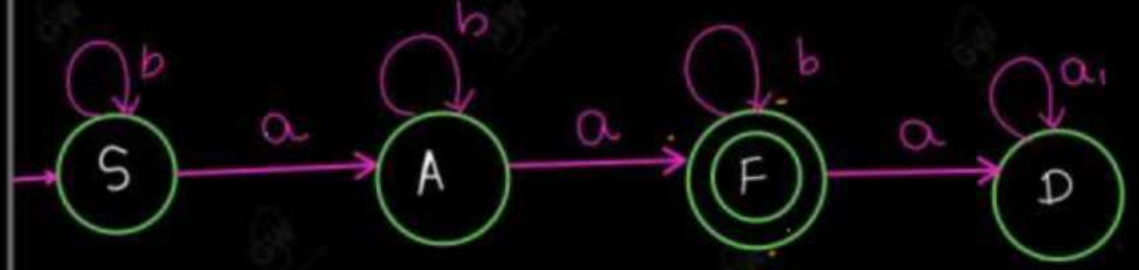
$q_1 \rightarrow 1q_1 \mid 0q_2$
$q_2 \rightarrow 1q_2 \mid 0q_3$
$q_3 \rightarrow 0q_3 \mid 1q_3 \mid \varepsilon$

$Q_1 \rightarrow 1Q_1 \mid 0Q_2$
$Q_2 \rightarrow 1Q_2 \mid 0Q_3$
$Q_3 \rightarrow 0Q_3 \mid 1Q_3 \mid \varepsilon$

$V = \{Q_1, Q_2, Q_3\}$
$T = \{1, 0\}$
$P = \{ Q_1 \rightarrow 1Q_1,\ Q_1 \rightarrow 0Q_2,\ Q_2 \rightarrow 1Q_2,\ Q_2 \rightarrow 0Q_3,$
$Q_3 \rightarrow 0Q_3,\ Q_3 \rightarrow 1Q_3,\ Q_3 \rightarrow \varepsilon \}$

$S = \{Q_1$

finite automata to CFG

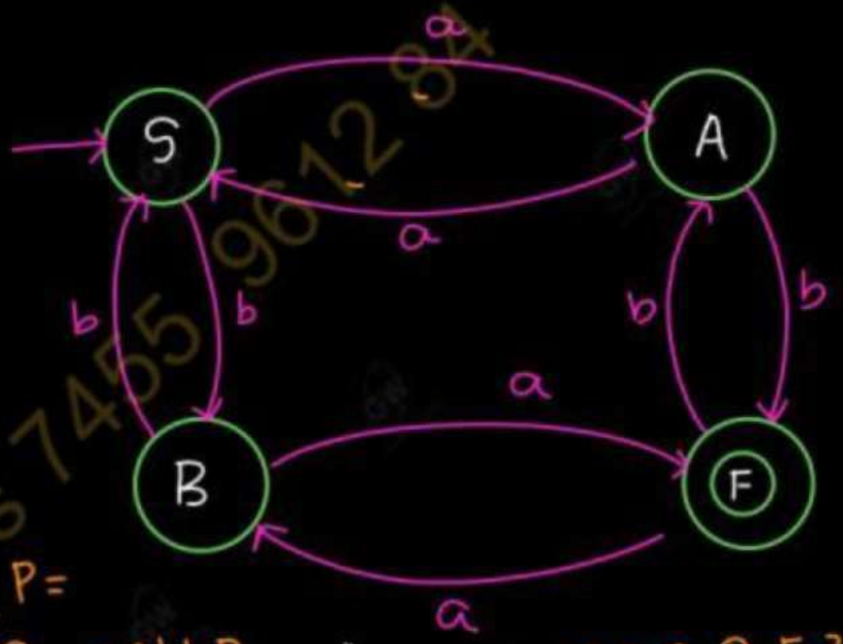


$S \to bS \mid aA$
$A \to bA \mid aF$
$F \to bF \mid \varepsilon$

$V = \{S, A, F\}$
$T = \{a, b\}$

$P = \{S \to bS,\ S \to aA,\ A \to bA,\ A \to aF,\ F \to bF,\ F \to \varepsilon\}$

$S = \{S\}$



$P =$
$\{S \to aA \mid bB$
$A \to aS \mid bF$
$B \to bS \mid aF$
$F \to aB \mid bA \mid \varepsilon\}$

$V = \{S, A, B, F\}$
$T = \{a, b\}$
$S = \{S\}$

Regular expression to CFG

R=(a+b)*aa(a+b)*

$S \rightarrow A\,aa\,A$

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

R=0*1(0+1)*

$S \rightarrow A1B$

$A \rightarrow \varepsilon \mid 0A$

$B \rightarrow \varepsilon \mid 0B \mid 1B$

R=a*

$\varepsilon, a, aa, aa \ldots$

$S \rightarrow \varepsilon \mid aS$

R=(a+b)*

$L: \{\varepsilon, a, b, aa, bb, ab, ba - - - - \}$

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

Regular expression to CFG

R=a*+a(a+b)*

$S \to X|Y$
$X \to \varepsilon|aX$
$Y \to aZ$
$Z \to \varepsilon|aZ|bZ$

R=a*b*

$S \to AB$
$A \to \varepsilon|aA$
$B \to \varepsilon|bB$

R=a*+b*

$S \to A|B$
$A \to aA|\varepsilon$
$B \to bB|\varepsilon$

RE=(a+b)*(a+b)+(ab+ba)*(a+b)*B(a+b)+a

$S \to X|Y|Z$
$Z \to a$
$X \to PQ$
$P \to aP|bP|\varepsilon$
$Q \to a|b$
$Y \to WPBQ$
$W \to abW|baW|\varepsilon$

$P = (a+b)^k$
$Q = a+b$

Decidability- Decision properties OF Regular language

- Approximately all the properties are decidable in case of finite automaton. Here we will use machine model to proof the decision properties
- Emptiness
- Non-Emptiness
- Finiteness
- Infiniteness
- Membership
- Equality

- Emptiness & Non-Emptiness

Step 1: select the state that can not be reached from the initial state and remove them (remove unreachable state)

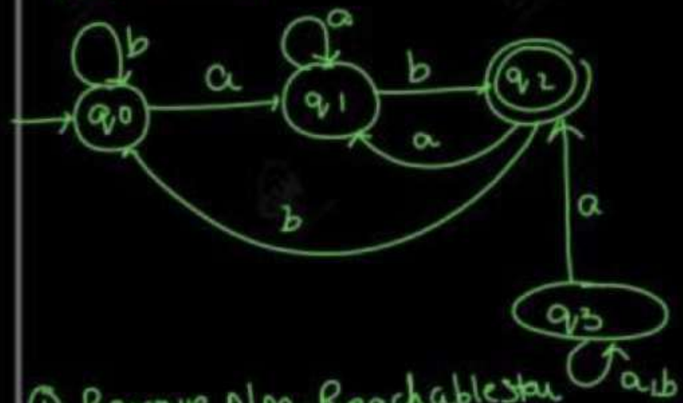
Step2: if the resulting machine contains at least one final state, so then the finite automata accept the non-empty language

Step 3: if t he resulting machine is free from final state, then automata accept empty language

Decidability- Decision properties OF Regular language

Example

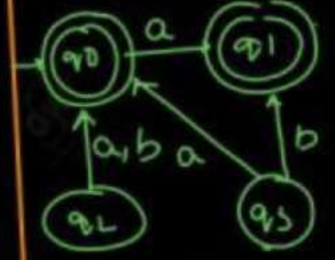
Non-Emptiness



① Remove Non-Reachable state

(Non-Empty final Automata

$L = \{w \mid \text{ending ab } \Sigma = \{a,b\}\}$



q_2, q_3 (Non-Reachable State

$L = \{\varepsilon, a\}$

Emptiness

ϕ

Decidability- Decision properties OF Regular language

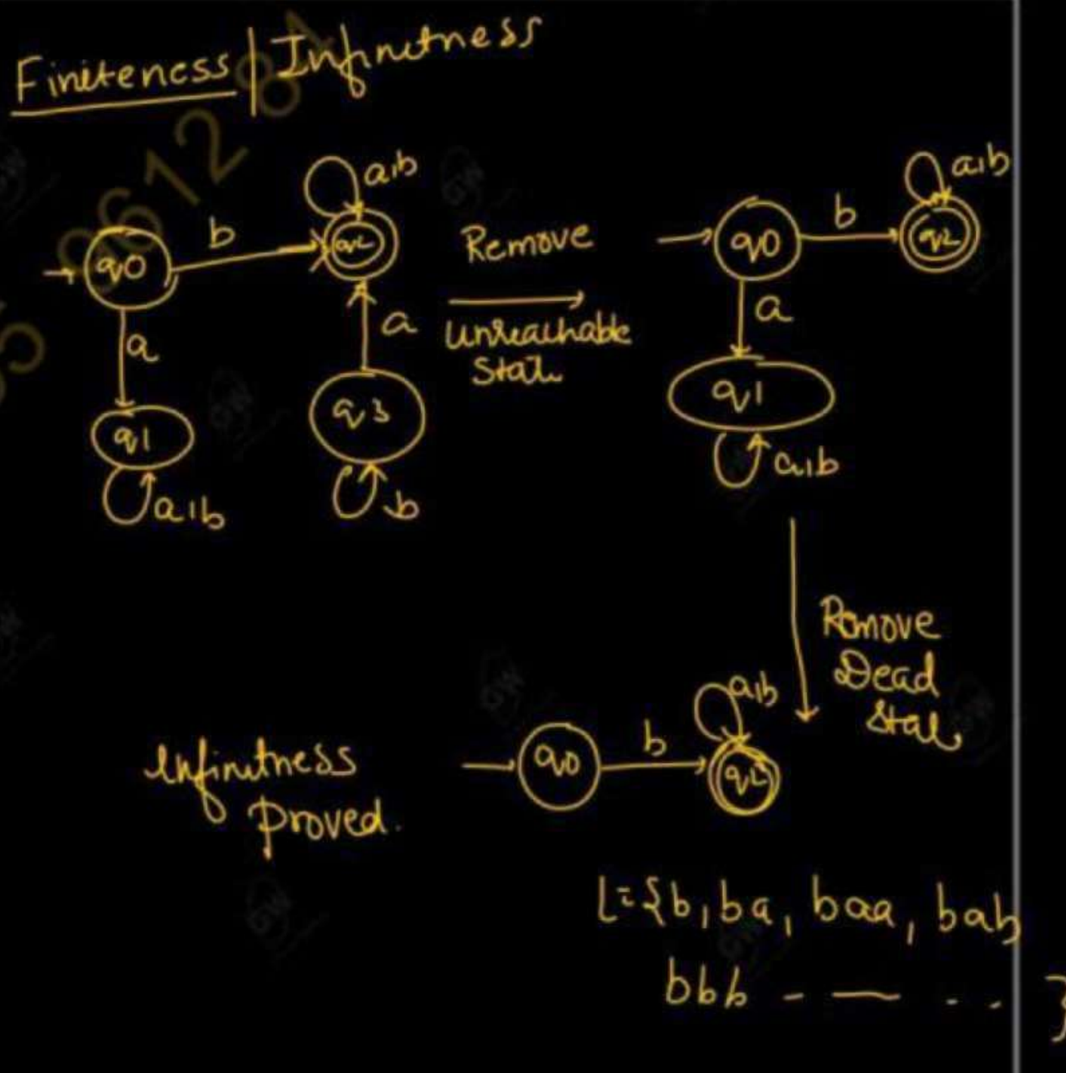
- Finiteness and infiniteness

Step 1: select the state that can not be reached from the initial state and remove them (remove unreachable state)

Step 2: select the state from which we can not reach final state and delete them (remove dead state)

Step 3: if the resulting machine contain loops or cycle then the f9inite automata accepts infinite language

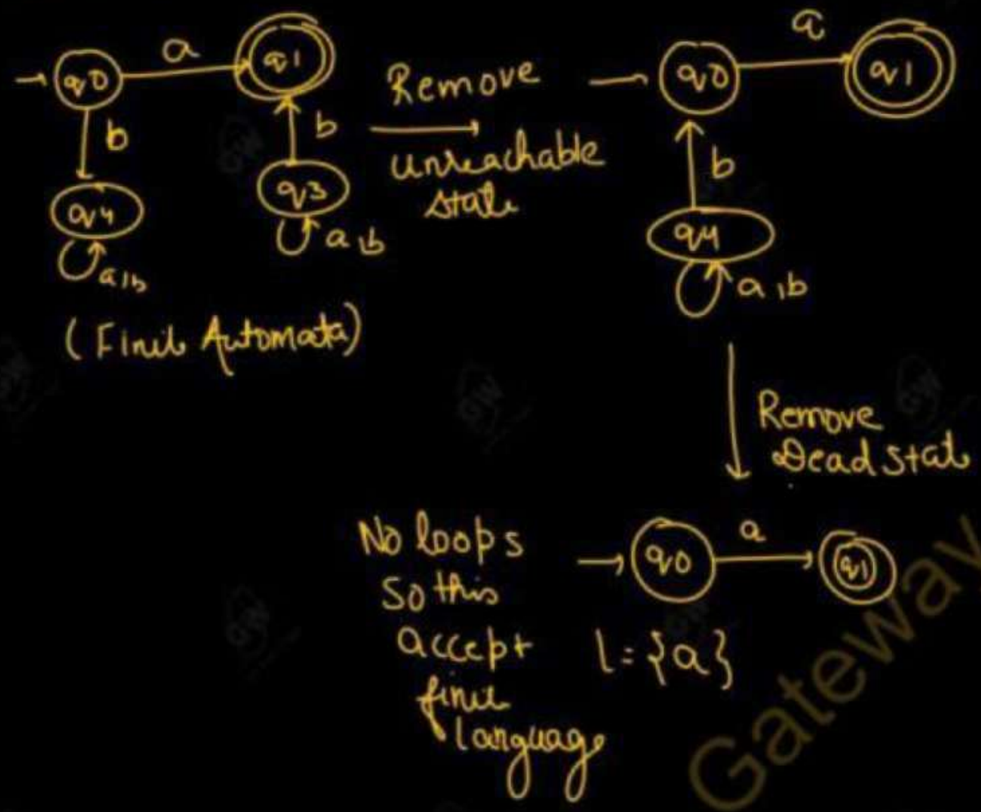
step 4:If the resulting machine do not contain loops or cycle then the finite automata accept finite language



Decidability- Decision properties OF Regular language

- example

Finiteness

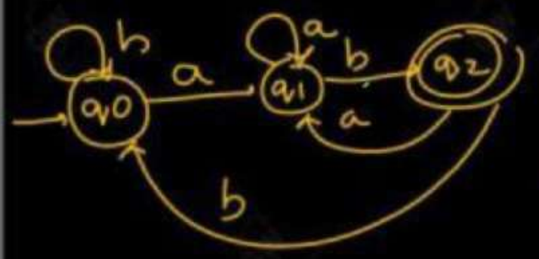


(Finite Automata)

- Membership
- Membership is a property to verify an arbitrary string is accepted by a finite automata or not i.e. it is a member of the language or not
- Let M is a finite automata that accept some string over the alphabet and let W be any string defined over the alphabet, if there exist a transition path in M, which start at initial state and end in anyone of the final state, then string W is member of M , otherwise W is not the member of M

Decidability- Decision properties OF Regular language

- Example



$L = \{ab, aaab, ababab, bbbab - - - \}$

abaaba ∉ L

$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2$

$q_2 \xrightarrow{a} q_1$

$q_1 \xrightarrow{a} q_2 \xrightarrow{b} q_1$

ab ∈ L

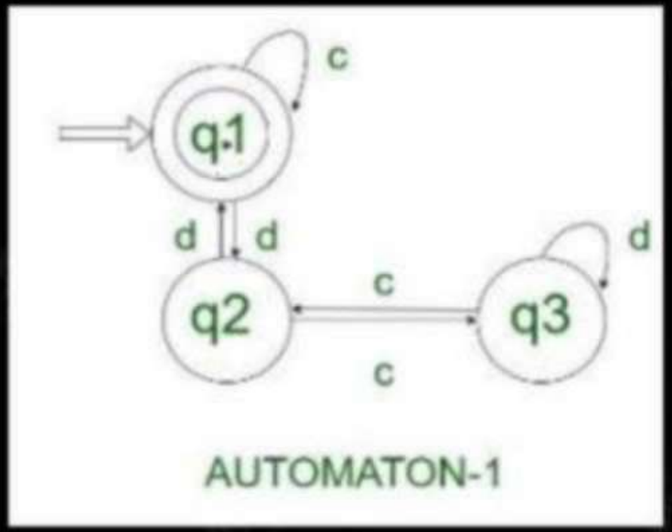
$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2$ – final state

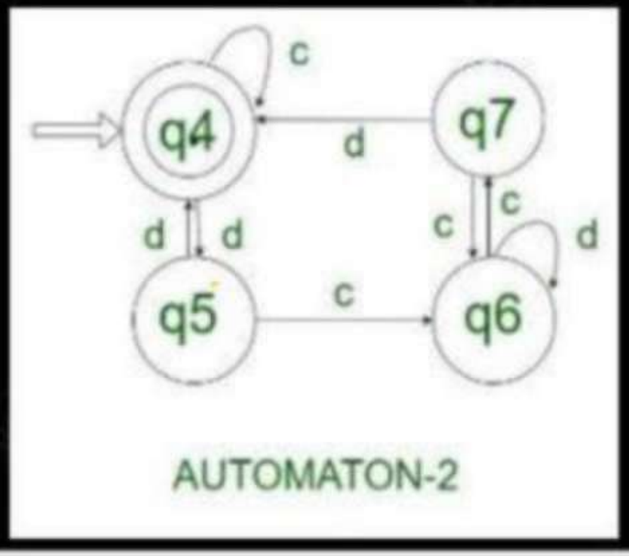
Equality

- If initial state is the final statue for one automaton then in second automaton also initial state must be the final state for them to be equivalent
- For any pair of state$\{q_i, q_j\}$ the transition for the input a belong to Σ is defined by $\{q_a, q_b\}$ where $\delta\{q_i, a\} = q_a$ and $\delta\{q_j, a\} = q_b$ the two automata are not equivalent if for a pair $\{q_a, q_b\}$ one is intermediate state and other is final state

Decidability- Decision properties OF Regular language

- Example





	c	d
→ {q1 q4}	{q1 q4} FS FS	{q2, q5} IS IS
{q2, q5}	{q3 q6} IS IS	{q1, q4} FS FS
{q3, q6}	{q2 q7} IS IS	{q3, q6} IS IS
{q2, q7}	{q3, q6} IS IS	{q1, q4} FS FS

Pumping lemma for context free language

- It is used to prove that language is not context free language
- let l is context free language let n be a integer constant select a string w from l such that |w|>=n
- divide the string w into 5 parts Uvwxy such that |vwx|<=n and |vx|>=1
- for i>=0 Uv^iwx^iy belong to l

show that $l=\{a^nb^nc^n \mid n>=1\}$ is not a CFL

Let L be a context free language

L = { abc, aabbcc, aaabbbccc — }

n=3 (let)

W = aaabbbccc

$|W| \geq n$

$9 \geq 3$ True

W = uvwxy

aaa b b b ccc

u v w x v

u = aa

v = a w = b

x = b y = bccc

$|vwx| <= n$

$|vx| \geq 1$

① $|vwx| <= n$

$|abb| <= 3$

$3 <= 3$ true

② $|vx| \geq 1$

$|ab| \geq 1$

$2 \geq 1$

true

③ $uv^iwx^iy \in L$ for $i \geq 0$

aa(a)^{1}b(b)^{1}bccc

i=0 aa(a)^{0}b(b)^{0}bccc

aa ε b ε bccc

aabbccc ∉ L

this prove that this not a CFL

Pumping lemma for context free language

show that $L=\{a^p \mid p \text{ is prime number}\}$ is not a CFL

Let L be a CFL

$L=\{a^2, a^3, a^5, a^7, a^{11}, \ldots\}$

Let $n=5$

$w = aaaaa$

$|w| \geq n$

$5 \geq 5$ true

$w = uvwxy$

a a a a a

u v w x y

① $|vwx| \leq n$

$|aaa| \leq 5$

$3 \leq 5$ true

② $|vx| \geq 1$

$|aa| \geq 1$

$2 \geq 1$ true

③

for $i \geq 0$

$uv^iwx^iy \in L$

$a(a)^ia(a)^ia$

$i=0$ $a(a)^0a(a)^1a = a^3 \in L$

$i=1$ $aaaaa = a^5 \in L$

$i=2$ $aa^2aa^2a = a^7 \in L$

$i=3$ $aa^3aa^3a = a^9 \notin L$

this is not a CFL

show that $L=\{a^{n^2} \mid n>=1\}$ is not a CFL

Let L is a CFL

$L=\{a, a^4, a^9, \ldots\}$

$n=2$ (let)

$w = aaaa$

$|w| \geq n$

$|aaaa| \geq 2$

$4 \geq 2$ true

$w = uvwxy$

a a ε a a

$|uvw| \leq n$

$2 \leq 2$

$|vx| \geq 1$

$u=a$

$v=a$

$w=\varepsilon$

$x=a$

$y=a$

① $|uvwxy| \leq n$

$|a \cdot \varepsilon \cdot a| \leq 2$

$|aa| \leq 2$

$2 \leq 2$ True

② $|vx| \geq 1$

$2 \geq 1$ true

③

for $i \geq 0$ uv^iwx^iy

$a(a)^i\varepsilon(a)^ia$

$a(a^i(a^ia$

$i=0$ $a(a^0(a^0a$

$a \cdot \varepsilon \cdot \varepsilon \cdot a = a^2 \notin L$

this proof that this is not a CFL.

Pumping lemma for context free language

show that $L=\{a^i b^j \mid j=i^2\}$ is not a CFL

Let L is a CFL

$L=\{ab, aabbbb, aaabbbbbbbbb - - - - -\}$

Let $n=4$

$w: aabbbb$

$|w| \geq n$

$|aabbbb| \geq 4$

$6 \geq 4$ true

$w = uvwxy$

aa bbbb

u v w x y

① $|vwx| \leq n$

$|bbb| \leq 4$

$3 \leq 4$

true

② $|vx| \geq 1$

$|bb| \geq 1$

$2 \geq 1$

true

③

for $i \geq 0$

$uv^i wx^i y \in L$

$aa(b)^1 b(b)^1 b$

$i:0$ $aa(b)^0 b(b)^0 b$

$aa\varepsilon b\varepsilon b$

$aabb \notin L$

this proof that this not a CFL

show that $L=\{a^n b^m a^n b^{n+m} \mid m,n>=0\}$ is not a CFL

Let L is CFL

$L=\{ababb, aabaabbb - \}$

Let $n=4$

$w = abaabb$

$|w| \geq n$

$|ababb| \geq 4$

$5 \geq 4$ true

$w = uvwxy$

ab ab b

$u = ab$

$v = b$

$w = \varepsilon$

$x = b$

$y = b$

③

for $i \geq 0$ $uv^i wx^i y$

$ab(a)^1 \varepsilon (b)^1 b$

$i:0$ $ab(a)^0 \varepsilon (b)^0 b$

$ab \varepsilon \varepsilon \varepsilon b$

$abb \notin L$

$i=1:$ $ababb \in L$

$i=2$ $aba^2 \varepsilon b^2 b$

$abaabbb \notin L$

this is not a CFL

Closure Properties of CFL

Union Property

$L_1 = \{a^x b^x \ x>0\}$

CFG

$S_1 \rightarrow aS_1b \mid ab$

So we can say S_1 is CFL

$L_2 = \{c^z a^z \ z \geq 0\}$

CFG

$S_2 \rightarrow cS_2a \mid \varepsilon$

$L_1 \cup L_2$ $\{S_1 \cup S_2\}$

$S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow aS_1b \mid ab$

$S_2 \rightarrow cS_2a \mid \varepsilon$

So, we can say CFL Closed under Union

Concatenation Property

$L_1 = \{a^x b^x, x>0\}$ L is CFL

CFG $S_1 \rightarrow aS_1b \mid ab$

$L_2 = \{c^z a^z, z>0$

CFG $S_2 \rightarrow cS_2a \mid \varepsilon$

$L_1 \cdot L_2 = a^x b^x c^z a^z \ x>0, z>0$

$L_2 \cdot L_1 = c^z a^z a^x b^x \ x>0, z>0$

$L = L_1 \cdot L_2$

$S \rightarrow S_1 \cdot S_2$

$S_1 \rightarrow aS_1b \mid ab$

$S_2 \rightarrow cS_2a \mid \varepsilon$

L is also CFL

So we can say CFL Closed under Concatination

$L' = L_2 \cdot L_1$

$S \rightarrow S_2 \cdot S_1$

$S_2 \rightarrow cS_2a \mid \varepsilon$

$S_1 \rightarrow aS_1b \mid ab$

L' is also CFL.

Closure Properties of CFL

Kleene Star Property

Let L is CFL then the Kleene star of L is Represented by L^*

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

$S \rightarrow aSb \mid \varepsilon$

$L^* = (a^x b^x)^*$

$S_1 \rightarrow SS_1 \mid \varepsilon$

CFL is closed under Kleene closure/ Kleene Star

However, CFLs are not closed under the following operations:

Intersection – If L1 and L2 are CFLs, then the intersection of these two, represented as L1 ∩ L2, may not be a CFL.

Intersection with a regular language – If L1 is a regular language and L2 is a CFL, then the intersection of these two, represented as L1 ∩ L2, will be a CFL.

Complement – If L1 is a CFL, the complement of L1, represented as L1', may not be a CFL.

Regular Grammar

Type-3 grammar/regular grammar:

Regular grammar generates regular language.

They have a single non-terminal on the left-hand side and a right-hand side consisting of a single terminal or single terminal followed by a non-terminal. | terminal more than 1 also possible

$A \rightarrow xB$ A→ aaaB, A→ bbbab, A→ Babab

$A \rightarrow x$

$A \rightarrow Bx$

where A, B ∈ Variable(V) and x ∈ T* i.e. string of terminals.

Types of regular grammar:

Left Linear grammar(LLG)

Right linear grammar(RLG)

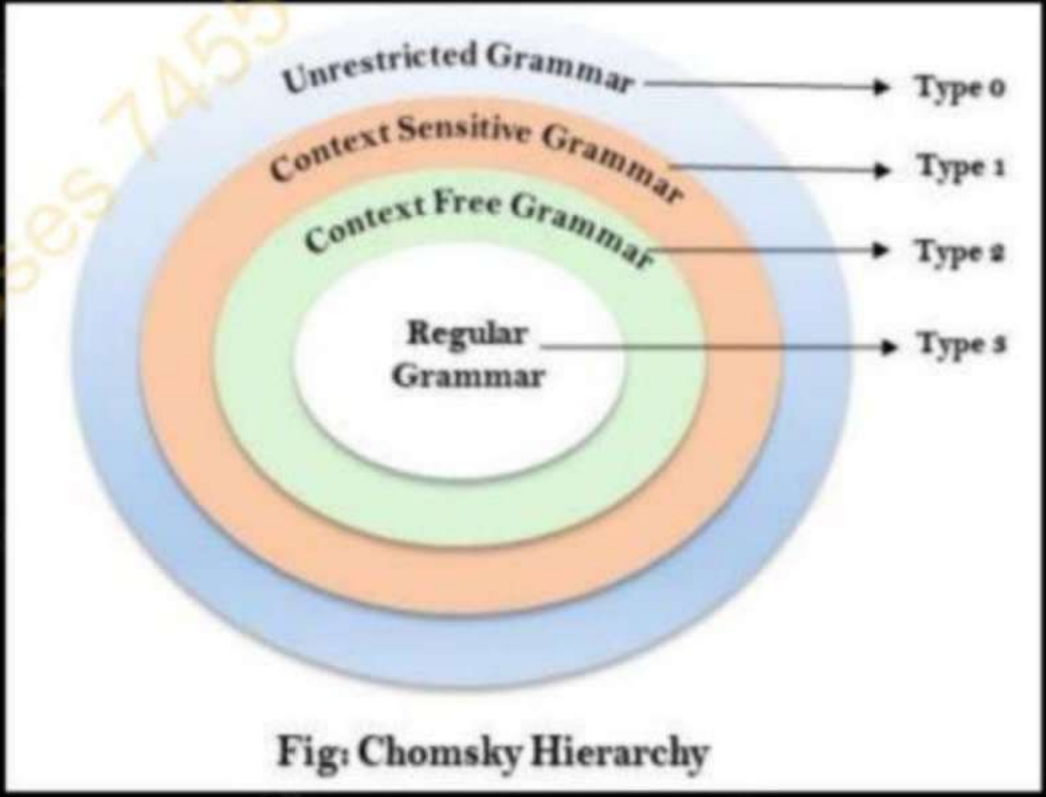


Fig: Chomsky Hierarchy

Regular Grammar

Left linear grammar(LLG):

$A \dashrightarrow Bx$

$A \dashrightarrow x$

where $A,B \in V$ and $x \in T^*$

Right linear grammar(RLG):

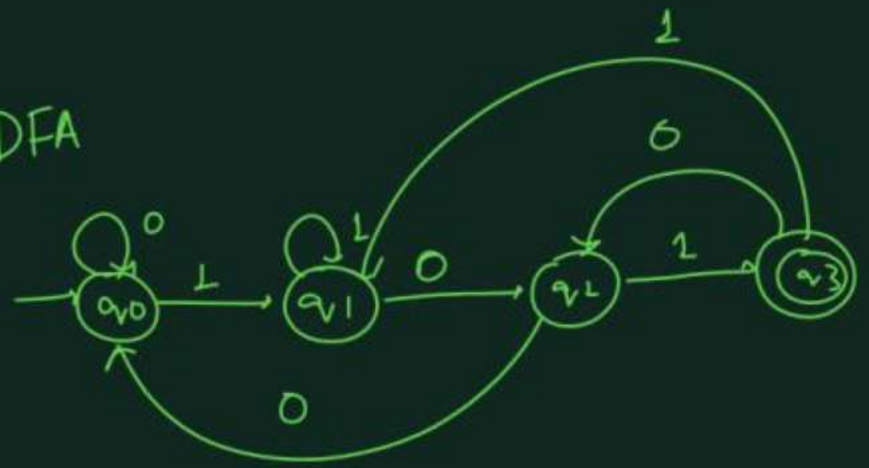
$A \dashrightarrow xB$

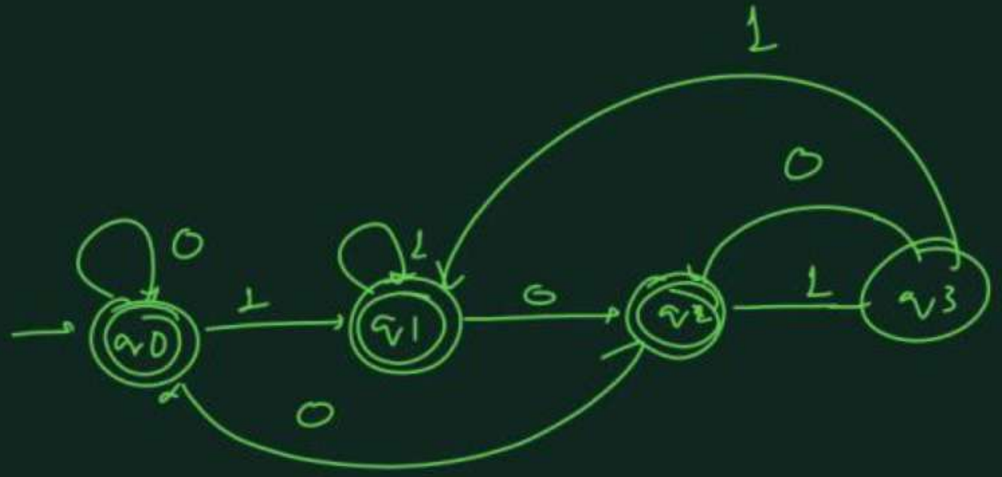
$A \dashrightarrow x$

where $A,B \in V$ and $x \in T^*$

DOES not accept 101

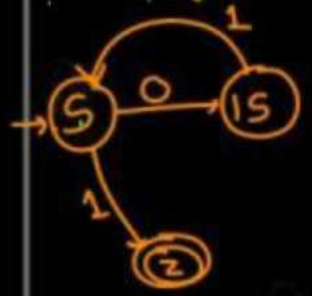
DFA

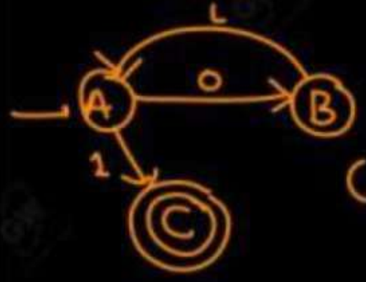




Convert regular grammar to finite automata

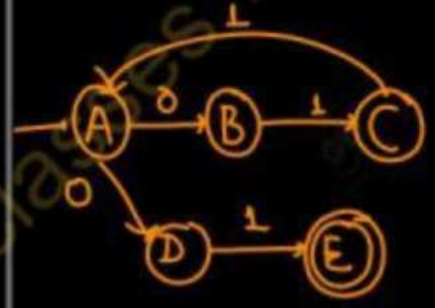
S->01S/1 (Right linear Grammar)





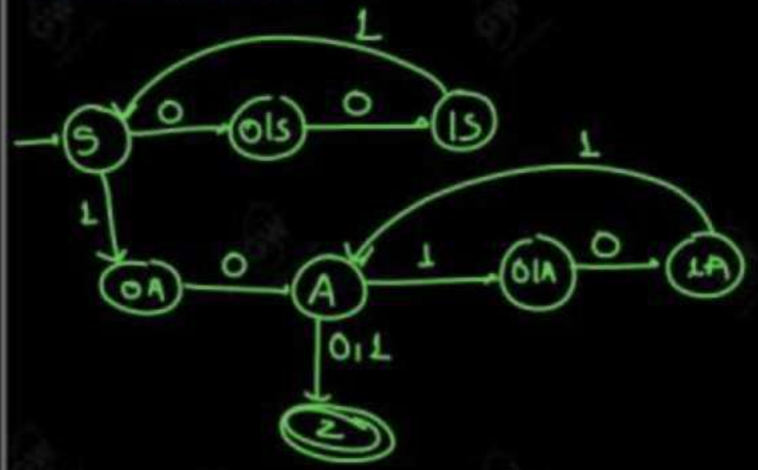
(NFA) ⟶ NFA to DFA Refer Unit 1

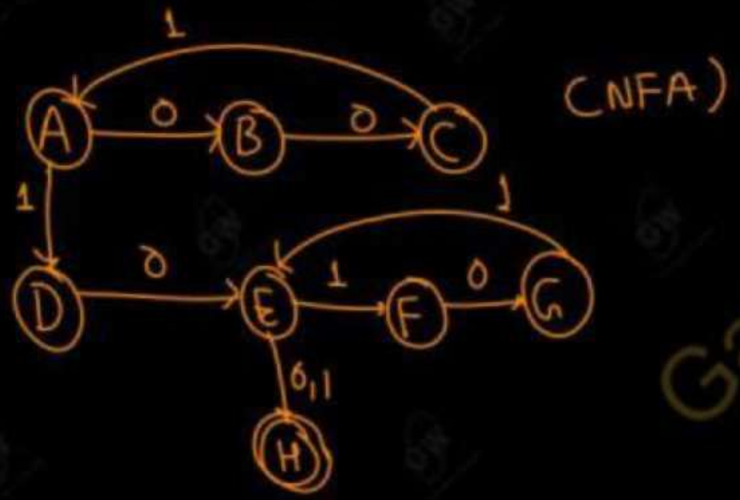
S->011S/01 (Right linear Grammar)



(NFA)

S->001S/10A
A->101A/0/1





(NFA)

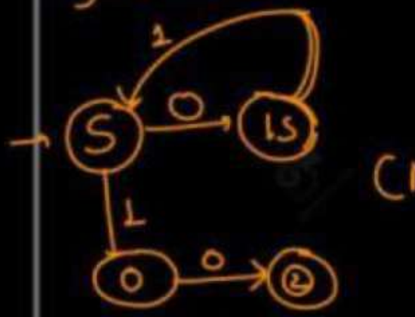
How to write left regular grammar

- Reverse the right-hand side of the production
- construct the NFA
- Interchange the initial state and final state
- Change the direction of the edges

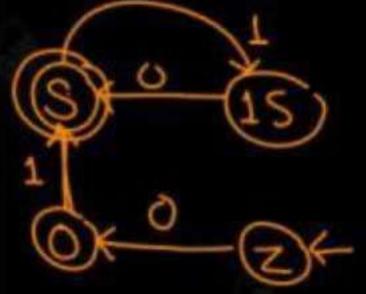
RHS

$S \to S10 \mid 01$

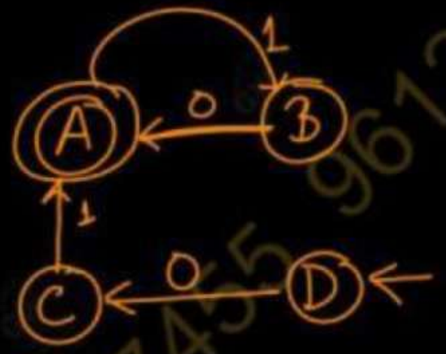
$S \to 01S \mid 10$ (Reverse the Right hand side of production)



(NFA)



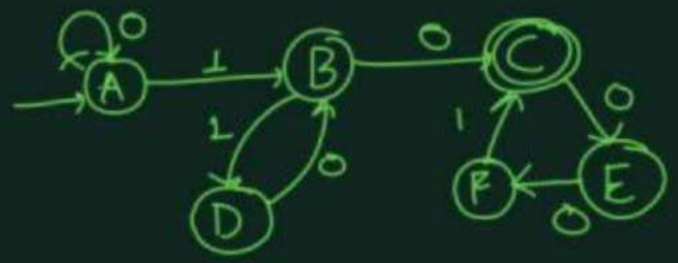
Interchange initial & final state & Reverse direction of edges.



(NFA)

NFA to DFA Conversion Already done in Unit-1

Q1

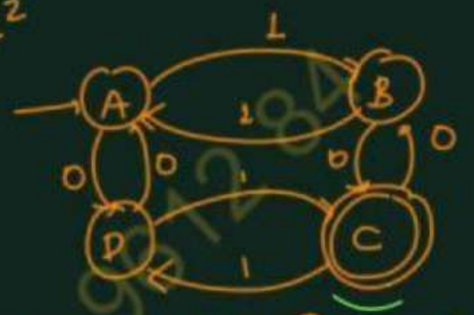


(Finite Automata Convert to Regular Grammar

A → 0A | 1B
B → 0C | 1D
C → 0E | ε
D → 0B
E → 0F
F → 1C

Regular Grammar (Right linear Grammar)

Q2



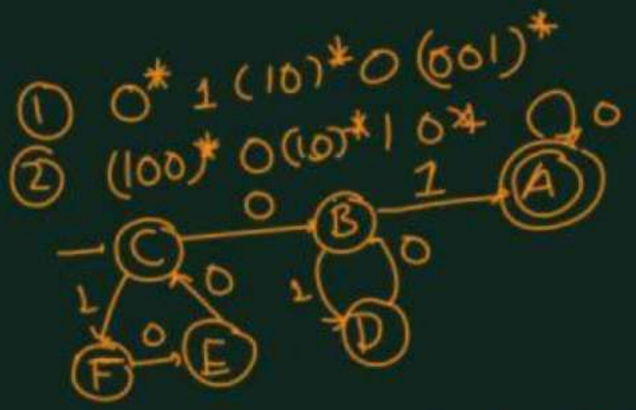
Write Regular Grammar

A → 0D | 1B
B → 0C | 1A
C → 0B | 1D | ε
D → 0A | 1C

(Right linear Grammar)

① $0^* 1 (10)^* 0 (001)^*$

② $(100)^* 0 (10)^* | 0^*$



$C \to 1F \mid 0B$	$C \to F1 \mid B0$
$F \to 0E$	$F \to E0$
$E \to 0C$	$E \to C0$
$B \to 1A \mid 1D$	$B \to A1 \mid D1$
$D \to 0D$	$D \to D$
$A \to 0A \mid \varepsilon$	$A \to A0 \mid \varepsilon$

Epsilon production

Unit production

Useless

$S \to AB$

$A \to C \mid a$

$C \to b$

$B \to Ba \mid d$

$S \to aABC$ ×

$S \to a$

$S \to AB$

$S \to a$

How to write left regular grammar from finite automata

1 obtain the regular expression

2 reverse the regular expression

3 construct the finite automata

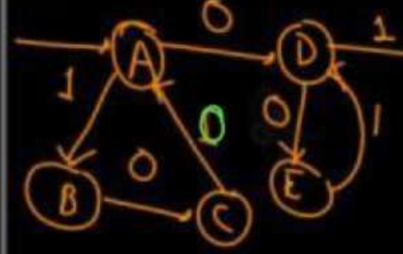
4 construct the right regular grammar

5 reverse the right hand of every production

Q1 left linear Grammar

① $0^*1(10)^*0(001)^*$ (prefix)

② Reverse $(100)^*0(01)^*|0^*$



(Finite Automata)

$A \to 0D|1B$
$B \to 0C$
$C \to 0A$
$D \to 0E|1F$
$E \to 1D$
$F \to 0F|\varepsilon$

Right linear Grammar

$A \to D0|B1$
$B \to C0$
$C \to A0$
$D \to E0|F|$
$E \to D1$
$F \to F0|\varepsilon$

(left linear Grammar)