18CSC301T FORMAL LANGUAGE AND AUTOMATA

UNIT II

Context free grammar and Language

Unit 2

- Context Free Grammar
  - Types
  - Derivations
  - Ambiguity
- Simplification of CFG
  - Elimination of useless symbols
  - Elimination of unit productions
  - Elimination of Null production
- Formal Form
  - CNF
  - GNF

Introduction to Grammar

Grammars: Introduction

- Grammars denote syntactical rules for conversation in natural languages.
- Noam Chomsky gave a mathematical model of grammar in 1956.
- A grammar is a set of production rules which are used to generate strings of a language.
- A grammar can be represented as 4 tuples (N, T, P, S)
- Where,
- N:- Set of Non terminals or variable list
- T:- Set of Terminals($T \in \Sigma$)
- S:- Special Non terminal called Starting symbol of grammar($S \in N$)
- P:- Production rule (of the form $\alpha \rightarrow \beta$, where α and β are strings on $N \cup \Sigma$)

Two basic elements of a Grammar

1. Terminal symbols
2. Non-terminal symbols

Terminal Symbols-

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

Non-Terminal Symbols-

- Non-Terminal symbols are denoted by using capital letters such as A, B, C etc.
- 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 **variables**.

Example

- Example: Grammar G1

  P1: $S \rightarrow AB$

  P2: $A \rightarrow a$

  P3: $B \rightarrow b$

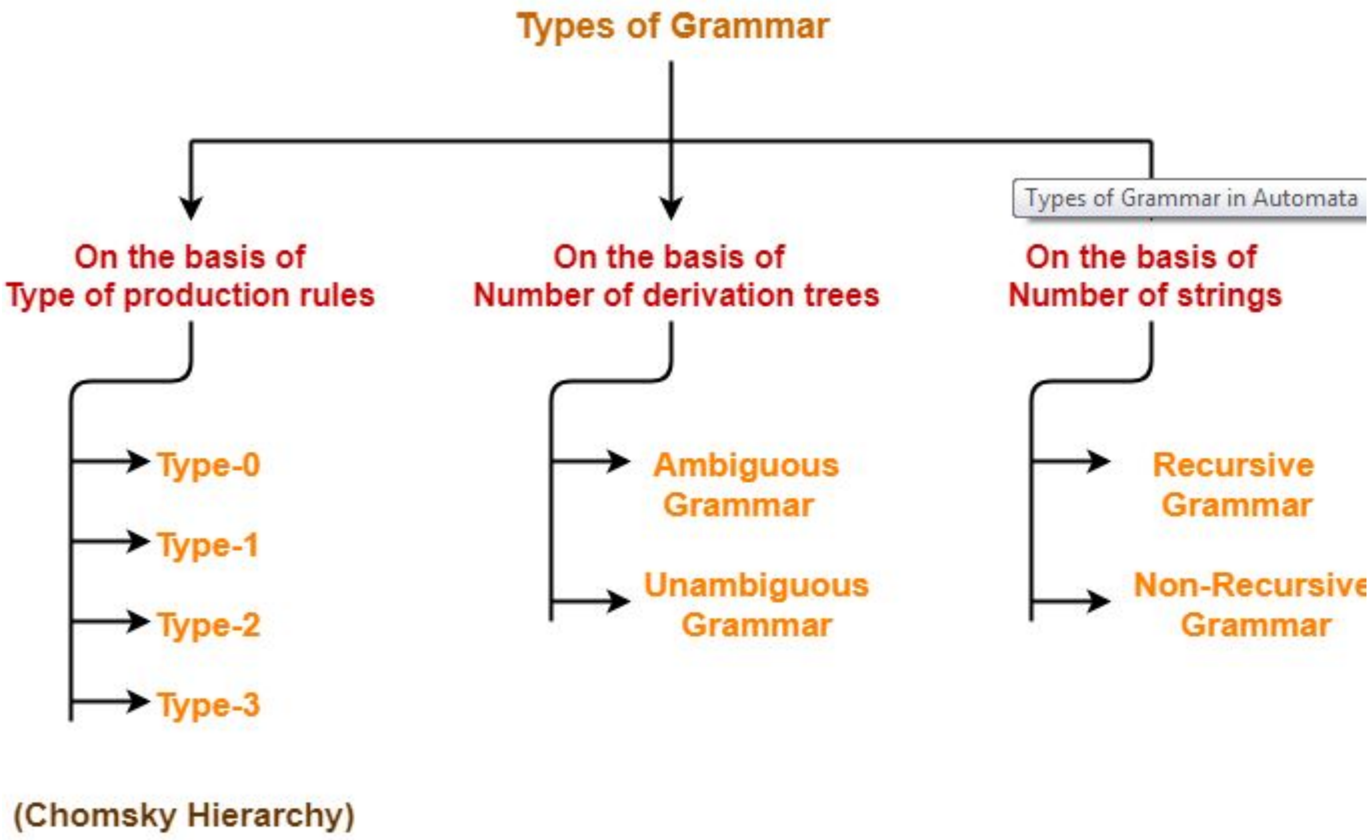
- G1= (N,T,P,S) = ({S, A, B}, {a, b}, {p1,p2,p3}, S)

Where,

- **S, A,** and **B** are Non-terminal symbols
- **a** and **b** are Terminal symbols
- **S** is the Start symbol, $S \in N$
- p1,p2,p3 – are Production rules

Types of Grammar

Types of Grammar



Chomosky Hierarchy

- According to Noam Chomosky, there are four types of grammars − Type 0, Type 1, Type 2, and Type 3.
- Type 0 known as unrestricted grammar.
- Type 1 known as context sensitive grammar.
- Type 2 known as context free grammar.
- Type 3 Regular Grammar.

Type 0: Unrestricted Grammar:

- Type-0 grammars include all formal grammars.
- Type 0 grammar languages are recognized by Turing Machine.
- These languages are also known as the Recursively Enumerable languages.
- Grammar Production in the form of $\alpha \to \beta$
- where

  α is $(V + T)^* V (V + T)^*$
  V : Variables/NT
  T : Terminals.

  β is $(V + T)^*$.
- In type 0 there must be at least one variable on Left side of production.

Example1 :

Sab –> ba
A –> S.

Here, Variables are S, A and Terminals a, b.

Example2 :

S → ACaB
Bc → acB
CB → DB
aD → Db

Type 1: Context Sensitive Grammar

- Type-1 grammars generate the context-sensitive languages.
- The language generated by the grammar are recognized by the Linear Bound Automata(LBA)

Rules:

1. First of all Type 1 grammar should be Type 0.
2. Grammar Production in the form of $\alpha \rightarrow \beta$

Where,

α , β is (V + T)+.

| α | <= | β |

i.e count of symbol in α is less than or equal to β

Example: 1

S –> AB
AB –> abc
B –> b

Example: 2

AB → AbBc
A → bcA
B → b

Type 2: Context Free Grammar:

- Type-2 grammars generate the context-free languages.
- The language generated by the grammar is recognized by a Pushdown automata (PDA)

Rules:

1. First of all it should be Type 1.
2. Left hand side of production can have only one variable.
3. Grammar Production in the form of $\alpha \rightarrow \beta$

Where,

α is Single NT

β is (V + T)*.

$| \alpha | <= | \beta |$

i.e count of symbol in α is less than or equal to β

Example

S –> AB
A –> a/ε
B –> b

Type 3: Regular Grammar:

- Type-3 grammars generate regular languages.
- These languages can be accepted by a finite state automaton (FA)
- Type 3 is most restricted form of grammar.
- The productions must be in the form

$X \rightarrow Aa/a$

$X \rightarrow aA/a$

where,

X,A is Non Terminal

$a \in \sum *$

Example

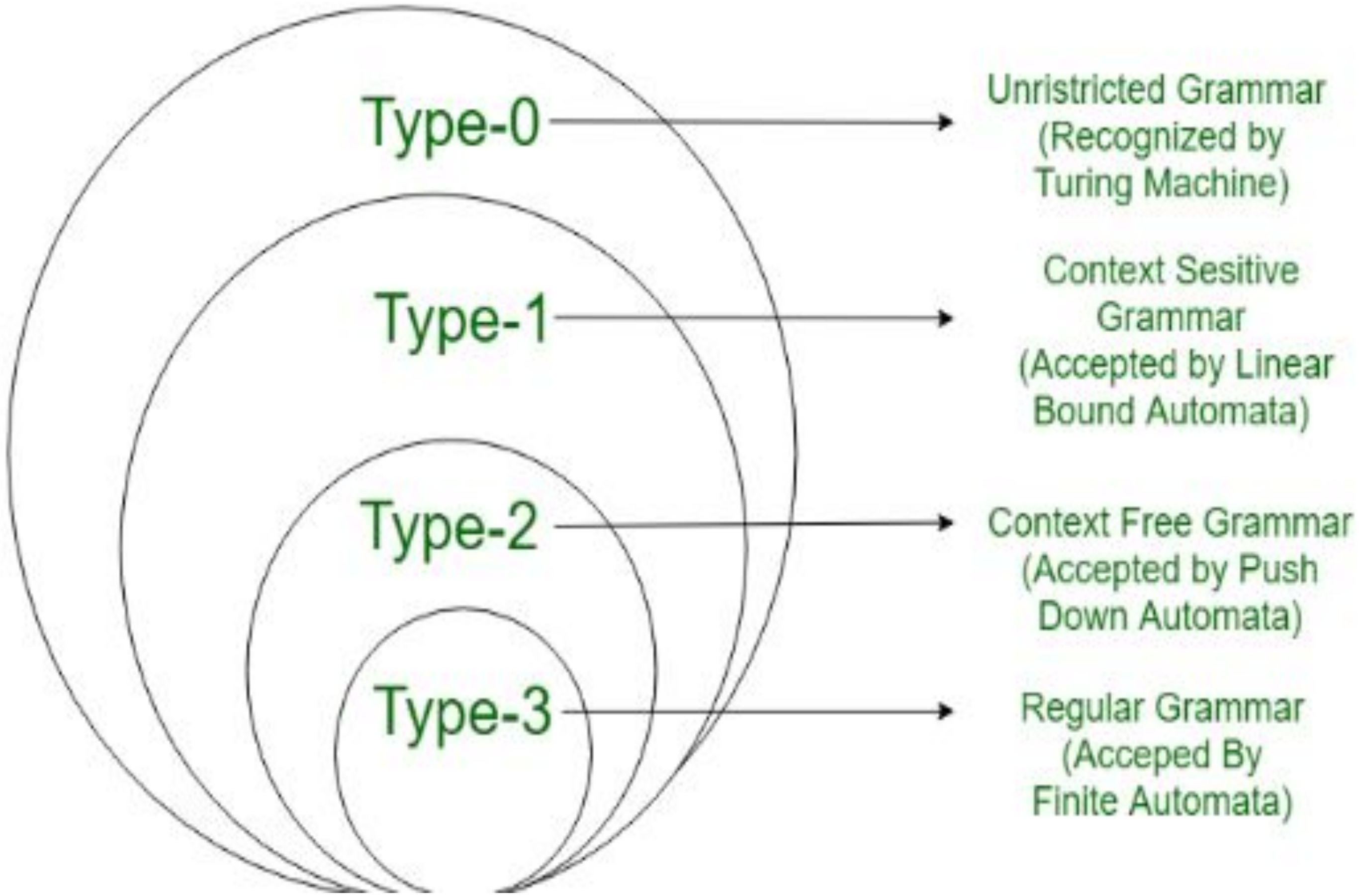
S->aS/b

S->aS/c

S->Sa/b

A->ba/ ε

Contd…


Type-0
Type-1
Type-2
Type-3
Unristricted Grammar (Recognized by Turing Machine)
Context Sesitive Grammar (Accepted by Linear Bound Automata)
Context Free Grammar (Accepted by Push Down Automata)
Regular Grammar (Acceped By Finite Automata)

CFG and its Languages

Context Free Grammars and Languages

- Context free grammar (CFG) is a formal grammar which is used to generate all possible strings in a given formal language.
- Context free grammar G can be defined by four tuples as:

  (N, T, P, S)
- Where,
- N:- Set of Non terminals or variable list
- T:- Set of Terminals($T \in \Sigma$)
- S:- Special Non terminal called Starting symbol of grammar($S \in N$)
- P:- Production rule (of the form $\alpha \rightarrow \beta$, where α and β are strings on $N \cup \Sigma$)
- In CFG, the start symbol is used to derive the string.
- We can derive the string by repeatedly replacing a non-terminal by the right hand side of the production, until all non-terminal have been replaced by terminal symbols.
- It is used to generate all possible patterns of strings in a given formal language.

Examples

Example 1:

Construct the CFG for the language having any number of a's over the set ∑= {a}. R.E= a*

Grammar :Production rule (P):

S → aS rule 1

S → ε rule 2

Derive a string "aaa

-> S

->aS

->aaS rule 1

->aaaS rule 1

->aaa ε rule 2

-> aaa (Required string)

Contd…

Example 2:

Construct a CFG for the regular expression (0 +1)*

Grammar :Production rule (P):

$S \rightarrow 0S \mid 1S$ rule 1

$S \rightarrow \varepsilon$ rule 2

Derive a string “1001”

->S

->1S rule 1

->10S rule 1

-> 100S rule 1

-> 1001S rule 1

-> 1001ε rule 2

-> 1001 (Required string)

Contd…

Example 3:

Construct a CFG for defining palindrome over ∑={a,b}, L = {wcwR}

Grammar :Production rule (P):

S → aSa   rule 1

S → bSb   rule 2

S → c   rule 3

Derive a string "abbcbba“

S → aSa

→ abSba   from rule 2

→ abbSbba   from rule 2

→ abbcbba   from rule 3   (Required string)

Contd…

Example 4:

Construct a CFG for defining palindrome over ∑={a,b}

Grammar :Production rule (P):

S → aSa rule 1

S → bSb rule 2

S → a/b/ε rule 3

Derive a string "abbabba“

S → aSa

→ abSba from rule 2

→ abbSbba from rule 2

→ abbabba from rule 3 (Required string)

Contd…

Example 5:

Construct a CFG for set of strings with equal no.of a's and equal no.of a's over ∑={a,b}

Grammar :Production rule (P):

S → SaSbS rule 1

S →SbSaS rule 2

S → ε rule 3

Derive a string " babaab "

S → SaSbS from rule 1

→ SbSaaSbS from rule 2

→SbSaS bSaaSbS from rule 2

→ babaab from rule 3 (Required string)

Contd…

Example 6:

Construct a CFG for the language L = $a^n b^{2n}$ where n>=1,over ∑={a,b}

Grammar :Production rule (P):

S → aSbb rule 1

S → abb rule 2

Derive a string " aabbbb “

S → aSbb from rule 1

→ aabbbb from rule 2 (Required string)

Contd…

Example 7:

Construct a CFG for the RE=(011+1)* (01)*

Grammar :Production rule (P):

$S \rightarrow AB$	rule 1
$A \rightarrow \varepsilon / CA$	rule 2
$C \rightarrow 011/1$	rule 3
$B \rightarrow \varepsilon / DB$	rule 4
$D \rightarrow 01$	rule 5

Derivation & Parse Tree

Derivations

- Starting with the start symbol, non-terminals are rewritten using productions until only terminals remain.
- Any terminal sequence that can be generated in this manner is syntactically valid.
- If a terminal sequence can’t be generated using the productions of the grammar it is invalid (has syntax errors).
- The set of strings derivable from the start symbol is the language of the grammar (sometimes denoted L(G)).
- Derivation is a sequence of production rules.
- It is used to get the input string through these production rules.

Contd…

- During parsing, we need to take the following two decisions.

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

- Based on the following 2 derivations, We have two options to decide which non-terminal to be placed with production rule .

1. Left most Derivation
2. Right most Derivation

- To illustrate a derivation, we can draw a derivation tree (also called a parse tree)

Left most 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.
- Leftmost non-terminal is always expanded.

Example:

E = E + E   Rule1

E = E - E   Rule2

E = a | b   Rule3

The leftmost derivation is:

W= a - b + a

E = E + E

E = E - E + E

E = a - E + E

E = a - b + E

E = a - b + a

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.
- Rightmost non-terminal is always expanded.

Example:

E = E + E Rule1

E = E - E Rule2

E = a | b Rule3

The rightmost derivation is:

W=a - b + a

E = E - E

E = E - E + E

E = E - E + a

E = E - b + a

E = a - b + a

Parse tree

- Parse tree is the graphical representation of symbol. The symbol can be terminal or non-terminal.
- In parsing, the string is derived using the start symbol.
- The root of the parse tree is that start symbol.
- All leaf nodes have to be terminals.
- All interior nodes have to be non-terminals.
- In-order traversal gives original input string.

Example:

Grammar G :

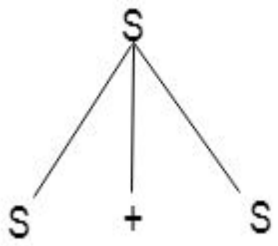
S → S + S | S * S

S → a|b|c

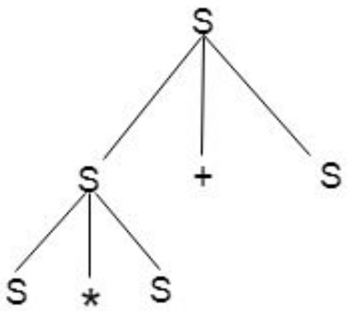
Input String : W=a * b + c

Parse Tree for Left most Derivation

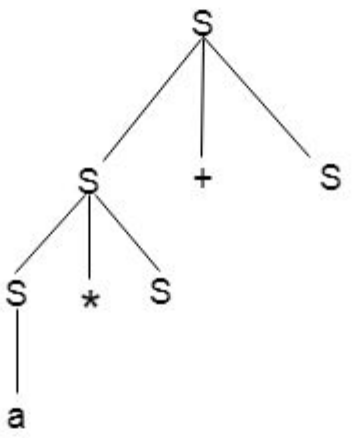
Step 1:



Step 2:

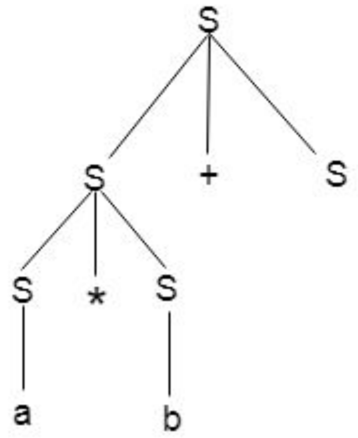


Step 3:

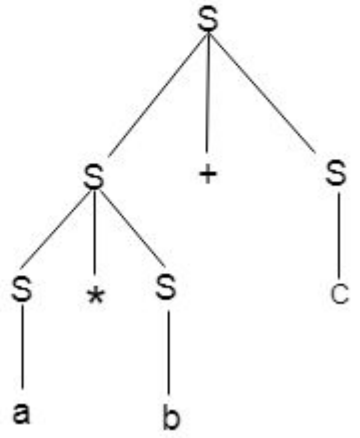


Contd…

Step 4:



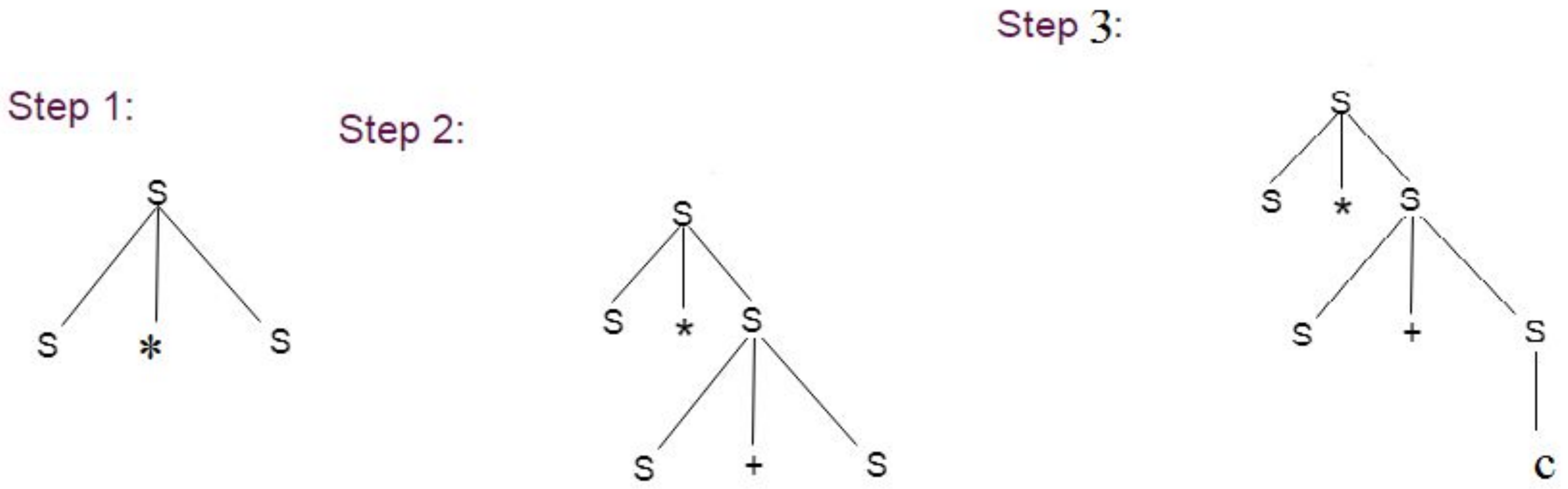
Step 5:



Contd...

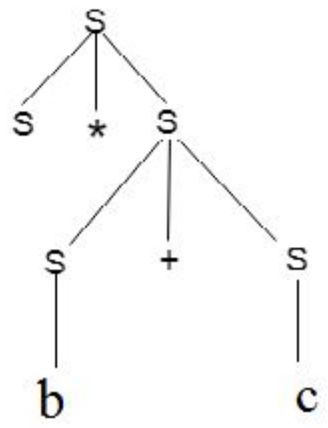
Input String : W=a * b + c

Parse Tree for Right most Derivation

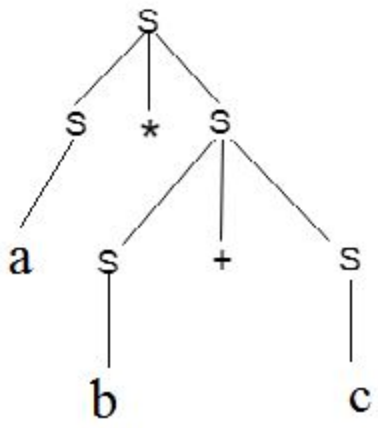


Contd…

Step 4:



Step 5:



What is the language defined by ‘G’

- G : $S \rightarrow aS/bS/a/b$

  $L(G) = (a+b)^+$
- G : $S \rightarrow XaaX$

  $X \rightarrow aX/bX/\varepsilon$

  $L(G) = (a+b)^* \, aa \, (a+b)^*$
- G : $S \rightarrow SS$

  $L(G) = \phi$

Contd...

- G : S →aca

  c →aca/b

  S → aca

  → aacaa

  → aaacaaa

  → aaabaaa

  $L(G) = a^n\ b\ a^n$
- G : S →0S1/ ε

  S → 0S1

  → 0 0S1 1

  → 0 00S11 1

  → 0 0011 1

  $L(G) = 0^n\ 1^n$ | for n>=0;

Ambiguous grammar

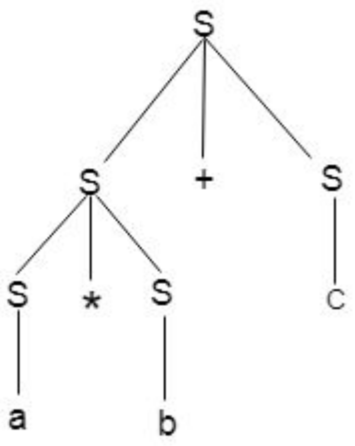
Ambiguity

- A grammar is said to be ambiguous if there exists more than one leftmost derivation or more than one rightmost derivative or more than one parse tree for the given input string.

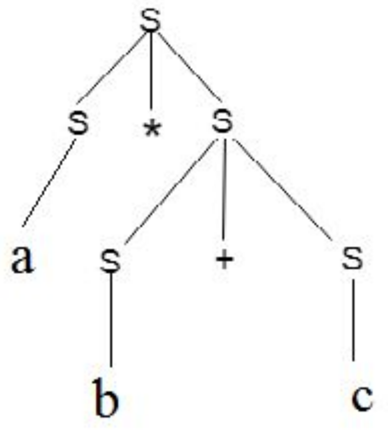
Example1: Input String : W=a * b + c

Parse Tree for Left most Derivation

[



Parse Tree for Right most



Contd...

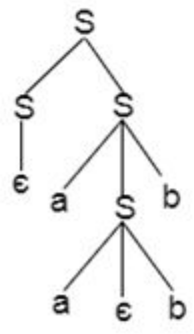
- Example 2 :

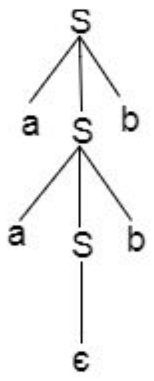
  S = aSb | SS

  S = ∈

Parse Tree I

Parse Tree II





Contd…

- If the grammar has ambiguity then it is not good for a compiler construction.
- No method can automatically detect and remove the ambiguity but you can remove ambiguity by re-writing the whole grammar without ambiguity.

Ambiguous grammar to unambiguous grammar

<u>Example1:</u>

- Show that the given Expression grammar is ambiguous. Also, find an equivalent unambiguous grammar.

Input Grammar:

$E \rightarrow E * E$

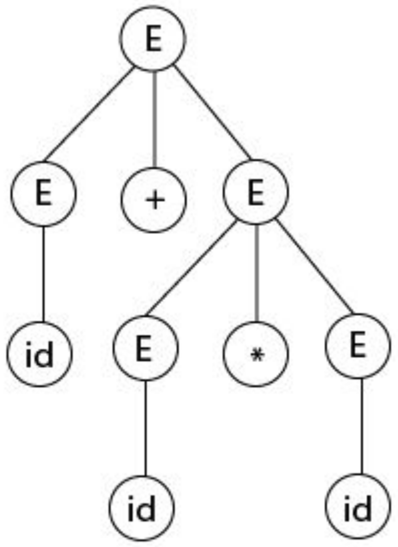
$E \rightarrow E + E$

$E \rightarrow id$

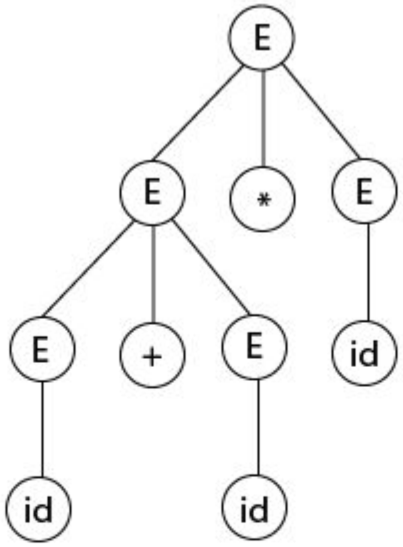
Solution:

- Let us derive the string "id + id * id"

Contd…



Parse tree 1



Parse tree 2

As there are two different parse tree for deriving the same string "id + id * id", the given grammar is ambiguous.

Removing ambiguity

Rewriting the grammar

For the Expression Grammar, use the following steps to get unambiguous grammar

1. Take care of precedence (Use a different non terminal for each precedence level and also start with the lowest precedence (PLUS)
2. Ensure associativity (define the rule as left recursive if the operator is left associative and as right recursive if the operator is right associative)

The equivalent unambiguous grammar

E → E + T

E → T

T → T * F

T → F

F → id

- It reflects the fact that ∗ has higher precedence than +.
- Also that, the operators + and ∗ are left-associative as these 2 are left recursive rules.

Contd…

Example2:

- Check that the given grammar is ambiguous or not. Also, find an equivalent unambiguous grammar.

S → S + S

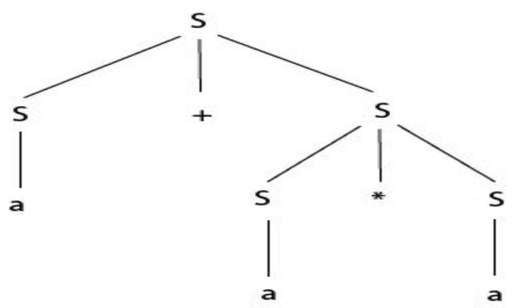
S → S * S

S → S ^ S

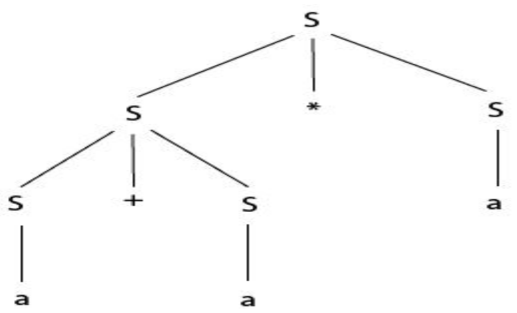
S → a

Solution:

Let us derive the string “a + a * a"



Parse tree 1



Parse tree 2

Contd…

The equivalent unambiguous grammar

$S \rightarrow S + A \mid A$

$A \rightarrow A * B \mid B$

$B \rightarrow C \wedge B \mid C$

$C \rightarrow a$

- It reflects the fact that ^ has higher precedence than * and +.
- The operators + and * are left-associative as these 2 are left recursive rules.
- The operators ^ is right associative as it is right recursive rule.

Elimination of Useless Symbols

Elimination of Useless Symbols

❖ Useful Symbols

❑ A symbol X in a CFG G = {V, T, P, S} is called useful

✔ if there exist a derivation of a terminal string from S where X appears somewhere,

✔ else it is called useless.

Elimination of Useless Symbols

- A CFG has no useless variables if and only if all its variables are reachable and generating.
- Therefore it is possible to eliminate useless variables from a grammar as follows:
  - ❑ Step 1: Find the non-generating variables and delete them, along with all productions involving non-generating variables.
  - ❑ Step 2: Find the non-reachable variables in the resulting grammar and delete them, along with all productions involving non-reachable variables.

Elimination of Useless Symbols

- Generating variables
  - A variable X is called as *generating*

  - if it derives a string of terminals.

  - Note that the language accepted by a context-free grammar is non-empty if and only if the start symbol is generating.

- Algorithm to find the non-generating variables in a CFG
  - Mark a variable X as "generating"
    - if it has a production X -> **w**, where w is a string of only terminals and/or variables previously marked "generating".
  - Repeat the above step until no further variables get marked "generating".
  - All variables not marked "generating" are non-generating

Elimination of Useless Symbols

- Reachable variables
  - A variable X is called as *reachable*
    - if the start symbol derives a string containing the variable X.
- Algorithm to find the non-reachable variables in a CFG
  - Mark the start variable as "reachable".
    - Mark a variable Y as "reachable" if there is a production X -> w, where **X** is a variable previously marked as "reachable" and **w** is a string containing Y.
  - Repeat the above step until no further variables get marked "reachable".
  - All variables not marked "reachable" are non-reachable

Elimination of Useless Symbols-Example

1. Remove the useless symbol from the given context free grammar

S -> abS | abA | abB
A ->cd
B->aB
C->dc

Solution:

❖ Step 1: Eliminate non-generating symbols i.e non-terminals which do not produce any terminal string

❖ In the given productions, B do not produce any terminal

❖ Eliminate all the productions in which B occurs.

- S -> abS | abA | ~~abB~~
- A ->cd
- ~~B->aB~~
- C->dc

❖ Resulting productions are: S -> abS | abA
A -> cd
C -> dc

Elimination of Useless Symbols-Example

- Step 2: Eliminate non-reachable symbols i.e non-terminals that can never be reached from the starting symbol

  - In the set of productions available after Step 2, 'C' is not reachable from starting symbol 'S'
  - Eliminate productions involving non-terminal 'C'

    S -> abS | abA

    A ->cd

    ~~C->dc~~
  - Final productions after eliminating useless symbols are:

    S -> abS | abA

    A ->cd

Elimination of Useless Symbols-Example

2. Remove the useless symbol from the given context free grammar

S -> aB / bX
A -> Bad / bSX / a
B -> aSB / bBX
X -> SBD / aBx / ad

❖ Step 1: Eliminate non-generating symbols i.e non-terminals which do not produce any terminal string

- A and X directly derive string of terminals a and ad, hence they are useful. Since X is a useful symbol so S is also a useful symbol as S -> bX.
- But B does not derive any terminals, so clearly B is a non-generating symbol.
- So eliminate the productions with B

S -> ~~aB~~ / bX
A -> ~~Bad~~ / bSX / a
~~B -> aSB / bBX~~
X -> ~~SBD~~ / ~~aBx~~ / ad

Elimination of Useless Symbols-Example

- The resulting productions are

  S -> bX
  A -> bSX / a
  X -> ad

❖ Step 2: Eliminate non-reachable symbols i.e non-terminals that can never be reached from the starting symbol

- In the reduced grammar A is a non-reachable symbol
- So remove the production involving A
- Final grammar after elimination of the useless symbols is

  S -> bX
  X -> ad

Elimination of Useless Symbols

- Elimination of useful symbols - Order of elimination
  - Always Eliminate non-generating symbol first and then eliminate non-reachable symbols
  - Reversing the order of elimination would not work

    S -> AB | a
    A -> aA
    B -> b

  - Here A is non-generating, and after deleting A (along with the production S -> AB) the variable B becomes unreachable. Hence, it is considered as useless variable
  - However, if we would first test for reachability, all variables would be reachable, and subsequently eliminating non-generating variables would leave us with B.

Elimination of Useless Symbols

- If a symbol is useful then it is both generating and reachable
- Converse of above statement is not true.
  - For e.g. in CFG

    $S \rightarrow ABC$

    $B \rightarrow b$

B is both reachable and generating but still not useful

Elimination of Null Productions

Elimination of Null Productions

- Null Productions
  A production of type $A \rightarrow \epsilon$ is called as Null production
- In a given CFG, a non-terminal N is called as nullable
  - if there is a production $N \rightarrow \epsilon$ or
  - If there is a derivation that starts at N and leads to ϵ

- If $A \rightarrow \epsilon$ is a production to be eliminated
  - look for all productions, whose right side contains A, and
  - replace each occurrence of A in each of these productions to obtain the non ϵ-productions.
  - resultant non ϵ-productions must be added to the grammar to keep the language the same.

Elimination of Null Productions – Example

1. Remove the null productions from the following grammar

S -> aX / bX

X-> a / b / ϵ

<u>Solution:</u>

- There is one null production in the grammar X -> ϵ.
- To eliminate X -> ϵ, change the productions containing X in the right side.
- The productions with X in the right side are S -> aX and S -> bX
- So replacing each occurrence of X by ϵ, we get two new productions

  S-> a and S -> b
- Adding these productions to the grammar and eliminating X -> ϵ, we get

  S -> aX / bX / a / b

  X-> a / b

Elimination of Null Productions – Example

- 2. Remove the null productions from the following grammar

  S -> ABAC
  A -> aA / ϵ
  B -> bB / ϵ and
  C -> c

<u>Solution:</u>

- We have two null productions in the grammar A -> ϵ and and B -> ϵ
- To eliminate A -> ϵ we have to change the productions containing A in the right side.
- The productions with A in the right side are S -> ABAC and A -> aA.
- So replacing each occurrence of A by ϵ, we get four new productions

  S -> ABC / BAC / BC

  A -> a

- Add these productions to the grammar and eliminate A -> ϵ.

  S -> ABAC / ABC / BAC / BC
  A -> aA / a
  B -> bB / ϵ
  C -> c

Elimination of Null Productions – Example

- To eliminate B -> ϵ we have to change the productions containing B on the right side.
- The productions with B in the right side are S -> ABAC / ABC / BAC / BC and B -> bB
- Doing that we generate these new productions:

    S -> AAC / AC / C
    B -> b

  Add these productions to the grammar and remove the production B -> ϵ from the grammar. The new grammar after removal of ϵ – productions is:

    S -> ABAC / ABC / BAC / BC / AAC / AC / C
    A -> aA / a
    B -> bB / b
    C -> c

Elimination of Unit Productions

Elimination of Unit Productions

- Unit Production
  - A unit production is a production A -> B where both A and B are non-terminals.
  - Unit productions are redundant and hence should be removed.
- Follow the following steps to remove the unit production

1. Select a unit production A -> B, such that there exist a production B -> α, where α is a terminal
2. For every non-unit production, B -> α repeat the following step
   - Add production A -> α to the grammar
3. Eliminate A -> B from the grammar
4. Repeat the above steps , if there are more unit productions

Elimination of Unit Productions – Example

1. Eliminate Unit productions from the given grammar

S-> aX / bY / Y

X-> S

Y -> bY / b

<u>Solution:</u>

- There are two unit productions in the given grammar, S -> Y and X -> S
- Substituting the values of unit production S -> Y we get,

  S-> aX / bY / bY / b ----� S-> aX / bY / b

- Substituting the values of unit production X -> S we get,

  X-> aX / bY / Y

- Final set of productions would be,

  S-> aX / bY / b

  X-> aX / bY / Y

  Y -> bY / b

Elimination of Unit Productions – Example

2. Eliminate Unit productions from the given grammar

S -> AB
A -> a , B -> C , C -> D and D -> b

Solution:

- There are two unit productions in the given grammar, B -> C and C -> D
- Substituting the values of unit production B -> C in C -> D we get,

  B-> D
- Substituting the values of unit production B-> D in D -> b we get,

  B-> b
- Substituting the values of unit production C-> D in D -> b we get,

  C-> b
- C is a non-reachable symbol. Hence remove it
- Final set of productions after removing non-reachable symbol would be,

  S -> AB
  A -> a

  B-> b

Exercise Problems

1. Remove the useless symbols from the given grammar

   A -> xyz / Xyzz
   X -> Xz / xYz
   Y -> yYy / Xz
   Z -> Zy / z

2. Remove the useless symbols from the given grammar

   T → aaB | abA | aaT

   A → aA

   B → ab | b

   C → ad

Exercise Problems

3. Remove the ε production from the following CFG by preserving the meaning of it.

$S \rightarrow XYX$

$X \rightarrow 0X \mid \varepsilon$

$Y \rightarrow 1Y \mid \varepsilon$

4. Remove the ε production from the following CFG by preserving the meaning of it.

$S \rightarrow ASA \mid aB \mid b$

$A \rightarrow B$

$B \rightarrow b \mid \in$

Exercise Problems

5. **Identify and remove the unit productions from the following CFG**

S -> S + T/ T
T -> T * F/ F
F -> (S)/a

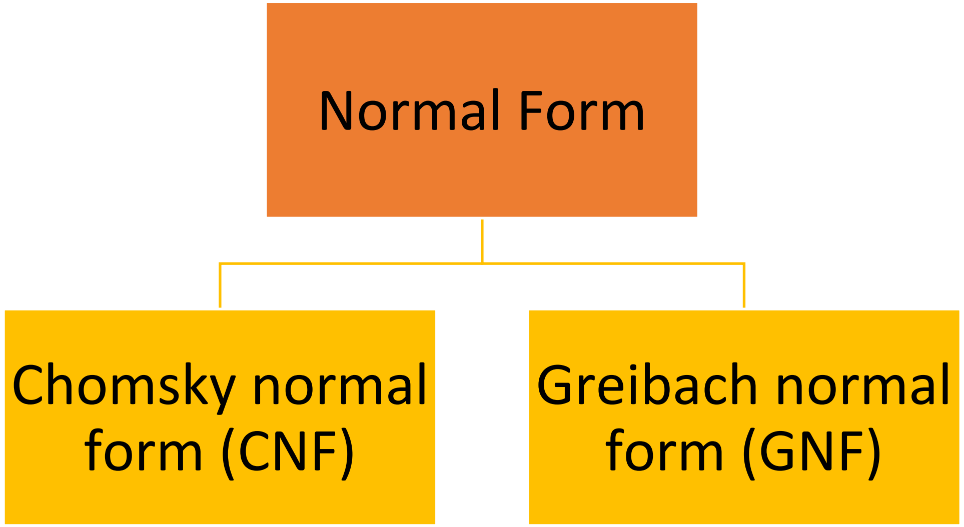
6. Remove the unit productions from the following grammar

S -> AB
A -> a
B -> C / b
C -> D
D -> E
E -> a

Normal Form

Normal Form

- **Normalization** is the process of minimizing **redundancy** from a relation or set of relations.
- A grammar is said to be in normal form when every production of the grammar has some specific form
- In this course we are going to study 2 types of Normal form



Chomsky normal form (CNF)

Chomsky normal form (CNF)

- A context free grammar (CFG) is in Chomsky Normal Form (CNF) if all production rules satisfy one of the following conditions:

Let consider,
NT = Non terminal (Eg. A,S,E..)
T = Terminal (Eg. a,b,0,1--)

1. $S \rightarrow \varepsilon$
2. NT→ T (Eg. A → a)
3. NT → NT NT (Eg. A →SE)

Steps to convert a CFG to CNF

1. Eliminate null, unit and useless productions (Kindly refer previous slides).
2. Eliminate terminals from RHS if they exist with other terminals or non-terminals.

Example:
Consider A → aX
Then we can convert to CNF form such as
Let Z → a
A → ZX

CNF Normal form
NT → T
NT → NT NT

Steps to convert a CFG to CNF

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

Example:
Consider A → BDX
Then we can convert to CNF form such as
Let Z → BD
A → ZX

CNF Normal form
NT→ T
NT → NT NT

Solved problem

Convert the following into CNF

$S \rightarrow bA \,/\, aB$ ($\approx C_b$, $\approx C_a$)

$A \rightarrow bAA \,/\, aS \,/\, a$

$B \rightarrow aBB \,/\, bS \,/\, a$

Step 1

↳ No ϵ production ↳ No useless production

↳ No unit production

Let $C_a \rightarrow a$ $C_b \rightarrow b$

$S \rightarrow C_bA \,/\, C_aB$

$A \rightarrow bAA \,/\, C_aS \,/\, a$

$B \rightarrow aBB \,/\, C_bS \,/\, a$

Let

$X \rightarrow AA$

$Y \rightarrow BB$

$\therefore$

$S \rightarrow C_bA \,/\, C_aB$ $C_a \rightarrow a$

$A \rightarrow C_bX \,/\, C_aS \,/\, a$ $C_b \rightarrow b$ $Y \rightarrow BB$

$B \rightarrow C_aY \,/\, C_bS \,/\, a$ $X \rightarrow AA$

CNF Normal form

NT→ T

NT → NT NT

CNF Problem

- **Define the two normal forms that are to be converted from a context free grammar(CFG).**

  Convert the following CFG to Chomsky normal form:

  S→A/B/C

  A→aAa/B

  B→bB/bb

  C→baD/abD/aa

  D→ aCaa/D

- **Construct the following grammar in CNF:**

  S→ ABC/BaB

  A →aA/BaC/aaa

  B →bBb/a/D

  C →CA/AC

  D→ ε

CNF Problem

- **Convert the following grammar into CNF**

  $S \rightarrow cBA$
  $S \rightarrow A$
  $A \rightarrow cB \mid AbbS$
  $B \rightarrow aaa$

- **Construct a equivalent grammar G in CNF for the grammar G1 where**

G1=({S,A,B},{a,b},{S →ASB/ ε , A→ aAS/a, B→ SbS/A/bb},S)

Greibach Normal Form (GNF)

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:

Let consider,
NT = Non terminal (Eg. A,S,E..)
T = Terminal (Eg. a,b,0,1--)

1. S → ε
2. NT→ T (Eg. A → a)
3. NT → T (NT)* (Eg. A →aSBBA)

Steps to convert a CFG to GNF

1. Eliminate null, unit and useless productions (Kindly refer previous slides).
2. Convert the given grammar into CNF form (Kindly refer previous slides).
3. Rename the Non Terminal as (A1,A2,A3,....)
4. Check the production such that all production should be in the form $A_i \rightarrow A_j$ where($i \leq j$) .
5. If the production is not as per step 4, Replace the production as per Lemma I or Lemma II

Lemma I

If G = (V,T,P,S) is a CFG and, the set of 'A' production belong to P are

$$A \rightarrow A\alpha \qquad \text{------ (1)}$$

$$A \rightarrow \beta_1 \mid \beta_2 \mid \beta_3 \mid \beta_4 \text{ ----- } \mid \beta_n \text{ ------ (2)}$$

then Let G' = (V',T,P',S)

Where P' be

$$A \rightarrow \beta_1 \alpha \mid \beta_2 \alpha \mid \beta_3 \alpha \mid \beta_4 \alpha \text{ ----- } \mid \beta_n \alpha$$

By sub. (2) in (1)

Lemma II

If G = (V,T,P,S) is a CFG and, the set of ‘A’ production belong to P are

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid A\alpha_3 \text{ -----} \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \beta_1 \text{ ------ } \mid \beta_n$$

Then introduce a new non-terminal X

So,Let G’ = (V’,T,P’,S) , Where V’ = (V ∪ X)

Where P’ can be formed

$A \rightarrow \beta_i \quad (1 \le i \le n)$
$A \rightarrow \beta_i X$

1

$X \rightarrow \alpha_j \quad (1 \le j \le m)$
$X \rightarrow \alpha_j X$

2

Solved problem (1)

Convert the following to GNF

$S \rightarrow AB$

$A \rightarrow BS|b$

$B \rightarrow SA|a$

Solution:

Step 1 & 2 : The given grammar is in CNF form

Step 3: Renaming the production, Let $S = A_1$, $A = A_2$, $B = A_3$

$A_1 \rightarrow A_2 A_3$ ---- (1)

$A_2 \rightarrow A_3 A_1 | b$ ---- (2)

$A_3 \rightarrow A_1 A_2 | a$ ---- (3)

Step 4: While checking the condition $A_i \rightarrow A_j$ where$(i \leq j)$

Equation(3) is not in the format , so as per Lemma I let us Sub. The value of A_1 from (1) to (3), so

$A_3 \rightarrow A_2 A_3 A_2 | a$ ---- (4)

GNF form

1. $S \rightarrow \varepsilon$
2. NT→ T (Eg. $A \rightarrow a$)
3. NT → T (NT)* (Eg. $A \rightarrow aSBBA$)

CNF form

1. $S \rightarrow \varepsilon$
2. NT→ T (Eg. $A \rightarrow a$)
3. NT → NT NT (Eg. $A \rightarrow SE$)

Lemma 1

$A \rightarrow A\alpha$ ------ (1)

$A \rightarrow \beta_1 | \beta_2 | \beta_3 | \beta_4$ ----- $| \beta_n$ ------ (2)

$A \rightarrow \beta_1 \alpha | \beta_2 \alpha | \beta_3 \alpha | \beta_4 \alpha$ ----- $| \beta_n \alpha$

Solved problem (1)

Lemma 2

$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid A\alpha_3 \text{-----} \mid A\alpha_m \mid \beta_1 \mid \beta_1 \text{------} \mid \beta_n$

$A \rightarrow \beta_i \quad (1 \le i \le n)$
$A \rightarrow \beta_i X$ 1

$X \rightarrow \alpha_j \quad (1 \le j \le m)$
$X \rightarrow \alpha_j X$ 2

Again as per Lemma I sub. The value of A_2 from equ. (2) in (4), we may get

$A_3 \rightarrow A_3 A_1 A_3 A_2 \mid b A_3 A_2 \mid a$ ---- (5)

So, Now let solve by Lemma 2,

$\underbrace{A_3}_{A} \rightarrow \underbrace{A_3}_{A} \underbrace{A_1 A_3 A_2}_{\alpha} \mid \underbrace{b A_3 A_2 \mid a}_{\beta}$ ---- (5)

$A_3 \rightarrow b A_3 A_2 \mid a$ ---- (6) (GNF)
$A_3 \rightarrow b A_3 A_2 X \mid aX$ ---- (7) (GNF)
$X \rightarrow A_1 A_3 A_2$ ---- (8)
$X \rightarrow A_1 A_3 A_2 X$ ---- (9)

Now sub (6) & (7) in (2)
$A_2 \rightarrow b A_3 A_2 A_1 \mid aA_1 \mid b A_3 A_2 X A_1 \mid aX A_1 \mid b$ ---- (10)(GNF)
Now Sub (10) in (1)
$A_1 \rightarrow b A_3 A_2 A_1 A_3 \mid aA_1 A_3 \mid b A_3 A_2 X A_1 A_3 \mid aX A_1 A_3 \mid bA_3$ ---- (11)(GNF)
Now sub (11) in (8)&(9)
$X \rightarrow b A_3 A_2 A_1 A_3 A_3 A_2 \mid aA_1 A_3 A_3 A_2 \mid b A_3 A_2 X A_1 A_3 A_3 A_2 \mid aX A_1 A_3 A_3 A_2 \mid bA_3 A_3 A_2$ ---- (12) (GNF)
$X \rightarrow b A_3 A_2 A_1 A_3 A_3 A_2 X \mid aA_1 A_3 A_3 A_2 X \mid b A_3 A_2 X A_1 A_3 A_3 A_2 X \mid aX A_1 A_3 A_3 A_2 X \mid bA_3 A_3 A_2 X$ ---- (13) (GNF)

Answer:

$A_1 \rightarrow b A_3 A_2 A_1 A_3 \mid aA_1 A_3 \mid b A_3 A_2 X A_1 A_3 \mid aX A_1 A_3 \mid bA_3$
$A_2 \rightarrow b A_3 A_2 A_1 \mid aA_1 \mid b A_3 A_2 X A_1 \mid aX A_1 \mid b$
$A_3 \rightarrow b A_3 A_2 \mid a$
$A_3 \rightarrow b A_3 A_2 X \mid aX$
$X \rightarrow b A_3 A_2 A_1 A_3 A_3 A_2 \mid aA_1 A_3 A_3 A_2 \mid b A_3 A_2 X A_1 A_3 A_3 A_2 \mid aX A_1 A_3 A_3 A_2 \mid bA_3 A_3 A_2$
$X \rightarrow b A_3 A_2 A_1 A_3 A_3 A_2 X \mid aA_1 A_3 A_3 A_2 X \mid b A_3 A_2 X A_1 A_3 A_3 A_2 X \mid aX A_1 A_3 A_3 A_2 X \mid bA_3 A_3 A_2 X$

Solved problem (2)

Convert the following Grammar into GNF

$S \to AA / a$

$A \to SS / b$

Step 1 :→ There is no ε, unit or null production

Step 2 :- Rename let $S = A_1$, $A = A_2$

∴

$A_1 \to A_2 A_2 / a$ —— ①

$A_2 \to A_1 A_1 / b$ —— ②

Step 3 :- As ② is not in the format of $A_i \to A_j$ where $i < j$

we replace leftmost A_1 as.

$A_2 \to \underbrace{A_2}_{A}\underbrace{A_2 A_1}_{\alpha} / \underbrace{a A_1 / b}_{\beta}$

If $A \to A\alpha_1 \cdots / \beta_1 / \beta_2 \cdots$

$A \to \beta_i$

$A \to \beta_i X$

$X \to \alpha_j$

$X \to \alpha_j X$

∴ $A_2 \to aA_1 / b$
$A_2 \to aA_1 X$
$A_2 \to bX$
} GNF → ③

$X \to A_2 A_1$
$X \to A_2 A_1 X$
} → ④

As ① is not in the format, replace the leftmost A_2 by (

$A_1 \to aA_1A_2 / bA_2 / aA_1XA_2 / bXA_2 / a$

Solved problem (2)

Ily Sub in (4)

$X \rightarrow aA_1A_1 / bA_1 / aA_1XA_1 / bXA_1$

$\cancel{X \rightarrow aA_1A_1X / bX / aA_1B_1X / bXX}$

$X \rightarrow aA_1A_1X / bA_1X / aA_1XA_1X / bXA_1X$

∴ The CNF form are

$$\boxed{\begin{array}{l} A_1 \rightarrow aA_1A_2 / bA_2 / aA_1XA_2 / bXA_2 \\ A_2 \rightarrow aA_1 / b / aA_1X / bX \\ X \rightarrow aA_1A_1 / bA_1 / aA_1XA_1 / bXA_1 \\ X \rightarrow aA_1A_1X / bA_1X / aA_1XA_1X / bXA_1X \end{array}}$$

where,
assumed. $A_1 \Leftrightarrow S$ and $A_2 \Leftrightarrow A$

Exercise problems

1. Convert the following CFG G to Greibach normal form generating the same language
   $S \rightarrow ABA$
   $A \rightarrow aA/\lambda$
   $B \rightarrow bB/\lambda$

2. What is the purpose of normalization? Construct the CNF and GNF for the following grammar and explain the steps.
   $S \rightarrow aAa/bBb/\epsilon$
   $A \rightarrow C/a$
   $B \rightarrow C/b$
   $C \rightarrow CDE/\epsilon$
   $D \rightarrow A/B/ab$

3. Convert the following grammar into GNF
   $S \rightarrow XY1 \mid 0$
   $X \rightarrow 00X \mid Y$
   $Y \rightarrow 1X1$