UNIT - III

GRAMMARS FORMALISM

Definition of Grammar:

A phrase-Structure grammar (or Simply a grammar) is (V, T, P, S), Where

(i) V is a finite nonempty set whose elements are called variables.

(ii) T is finite nonempty set, whose elements are called terminals.

(iii) S is a special variable (i.e an element of V) called the start symbol, and

(iv) P is a finite set whose elements are $\alpha \rightarrow \beta$,where α and β are strings on $V \cup T$, α has at least one symbol from V. Elements of P are called Productions or production rules or rewriting rules.

1. Regular Grammar:

A grammar G=(V,T,P,S) is said to be Regular grammar if the productions are in either right-linear grammar or left-linear grammar.

A→Bx|xB|x

1.1 Right –Linear Grammar:

A grammar G=(V,T,P,S) is said to be right-linear if all productions are of the form

A→xB or A→x.

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

1.2 Left –Linear Grammar:

A grammar G=(V,T,P,S) is said to be Left-linear grammar if all productions are of the form

A→Bx or A→x

Example:

1. Construct the regular grammar for regular expression r=0(10)*.

Sol: Right-Linear Grammar:

S→0A

A→10A|$\in$

Left-Linear Grammar:

S→S10|0

2. Equivalence of NFAs and Regular Grammar

2.1 Construction of $\in$-NFA from a right –linear grammar:

Let G=(V,T,P,S) be a right-linear grammar .we construct an NFA with $\in$-moves,

M=(Q,T, δ ,[S],[$\in$]) that simulates deviation in 'G'

Q consists of the symbols [ϕ] such that α is S or a(not necessarily proper)suffix of some

Right-hand side of a production in P.

We define α by :

1 .If A is a variable ,then α ([A], $\in$)={[α]|A→ α is a production}

2. If a is in T and α in $T^* \cup T^*V$, then δ ([a α],a)={[α]}

Example:

1. Construct an NFA for the right linear grammar S→0A , A→10A| $\in$

Sol:

From the above theorem, we know that

(1) IF A is a variable, then δ ([A], $\in$)={[α]A-> α is a production }

(2) If a is in T and α in $T^* \cup T^* \cup V$,then δ ([a α],a)={ [α]}.

The states Q ={[S]},[0A],[A],[10A],[E]}

These states are the suffix of right hand side of production P.

S-> 0A is a production gives a path from [S] to [0A] on reading input symbol E.

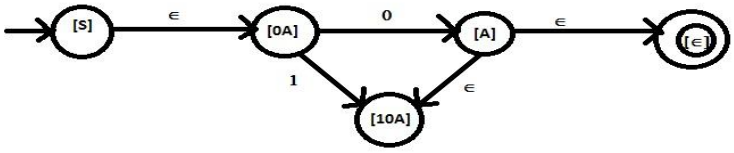
A-> 1-A production establish a path from [A] to [10A] on reading input symbol $\in$

According to rule 1

A-> $\in$ is also the direct production ,reading input $\in$,it gives the out put [$\in$]

δ ([0A],1)=[0A]

The Starting symbol is S and [$\in$] is the final state always.



2.2 Construction of ∈-NFA from a left-linear grammar

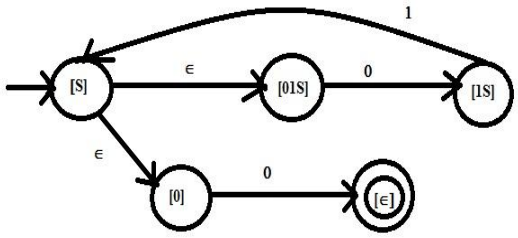
If 'G' is a left-linear grammar we will reverse all the right hand sider of the production then we will get right-linear grammar from which we will construct ∈-NFA of given left- linear we will exchange initial ,final states and reverse the direction of all the edges.

Example:

1. Construct an NFA for the grammar S→ S10|0

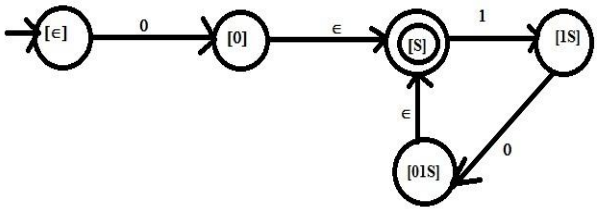
The grammar is LLG .But we can construct an NFA from the RLG Only To get the RLG from simply we will make reverse the right edges of the production

For this Grammar the NFA is



The δ is derived as in the previous example. $\delta([0,\in],0)= \in$

Now we wil reverse the edges of that NFA and exchange the initial state as final state then the ,NFA is



This is the for the given LLG

S→S10|0

3. Construction of right –linear and left –linear grammar from a given Finite Automata:

Right –linear grammar:

Let M=(Q,∑, δ ,q_0,F) be the given finite automata . First suppose that q_0 is not a final state. Then L=L(G) for a right –linear grammar G=(Q,∑, P,q_0) where P consists of production p-> aq whenever δ (p,a)=q and also p->q whenever δ (p,a) is a final state.

Now let q_0 be final state,so $\in$ is in L so introduce a new start symbol S with productions.

S->q_0/ $\in$

Example:

1. Derive the right linear grammar for the following DFA for $0(10)^*$

Solution:

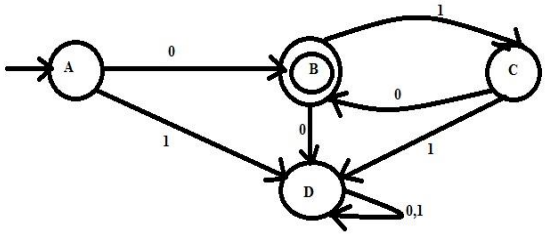
From the above Theorem

The transition fuction δ (p,a)= q gives a production p->aq.

If δ (p,a)= final state, then

p→ a is one more production

DFA:



The starting state can made as a starting symbol of the grammar.

δ (A,0)= B is represented by a production A->0B.

δ (A,0)=D =>A → 0D

δ (A,1)= D =>A →1D

We can write

B → OD from δ (B,0)=D

B → 1C from δ (B,1)=C

C → 0B from δ (C,0)=B

C → 1D from δ (C,1)=D

D → 0D from δ (D,0)=D

D → 1D from δ (D,1)=D

From the above DFA diagram

We can also write A→ 0and C → 0 as

δ (A,0)= Final state and

δ (C,0)= Final state

There are also to be added to the respective variables.

The RLG for The DFA is

A → 0B|1D|0

B → 0D|1C

C → 0B|1d|0

D → 0D|1D

Here D is useless symbol so thaht it is to be eliminated. For Applying the reduction technique the grammar can be reduced as

A → 0B|0
B → 1C
C → 0B|0

Left –linear grammar :

To get left-linear grammar reverse the right-hand sides of all the production of right –linear grammar.

Example:

1. Construct left linear grammar for the above regular expression $0(10)^*$

Solution:

To get left-linear grammar reverse the right-hand sides of all the production of right –linear grammar.

RLG: A → 0B|0

B → 1C

C → 0B|0

LLG:

A → B0|0

B → C1

C → B0|0

4. Context – Free Grammar:

A *context-free grammar* (CFG or just grammar) is defined formally as G=(V,T,P,S)

Where

V: a finite set of variables ("non-terminals"); e.g., A, B, C, …

T: a finite set of symbols ("terminals"), e.g., a, b, c, …

P: a set of production rules of the form $A \rightarrow \alpha$, where $A \in V$ and $\alpha \in (V \cup T)^*$

S: a start non-terminal; $S \in V$

A context-free grammar consists of a set of productions of the form $A \rightarrow \alpha$, where 'A' is a single non-terminal symbol and 'α' is a potentially mixed sequence of terminal and non-terminal symbols.

Eg:

E → E+E

E → E*E

E → (E)

E → id

In the above example, grammar tuples are defined as follows:

G=({E},{+,*,(,),id},{ E → E+E, E → E*E, E → (E), E → id},E).

In this chapter we use the following conventions regarding grammars.

1) The capital letters A,B,C,D,E and S denote variables; S is the start symbol unless otherwise stated.
2) The lowe-case letters a,b,c,d,e,digits,special symbols and boldface strings are terminals.
3) The capital letters X,Y and Z denote symbols that may be either terminals or varibales.
4) The lower-case letters u,v,w,x,y and z denote strings of terminals.
5) The lower-case Greek letters α,β and γ denote strings of varibles and terminals.

Generally we specify the grammar by listing the productions.

If A → α1, A → α2, A → α3, … A → αk are the prodcution then we may express then by

A → α1 | α2 | α3 | … | αk

Context Free Language:

- If G is a CFG, then L(G), the language of G, is $\{w \mid S \overset{*}{\Rightarrow} w\}$.

  Note: 'w' must be a terminal string, S is the start symbol.

Examples:

1. Construct a CFG to generate set of palindromes over alphabet {a,b}.

Solution:

The productions of a grammar to generate palindromes over {a,b} are

$S \rightarrow aSb \mid bSb \mid \in$

Hence $S \Rightarrow aSa \Rightarrow abSba \Rightarrow ab \in ba \Rightarrow abba$

This is the even palindrome.

Productions to generate odd palindrome are

$S \rightarrow aSb \mid bSb \mid a \mid b$

Hence $S \Rightarrow aSa \Rightarrow abSba \Rightarrow ab\ a\ ba \Rightarrow ababa$

This is the odd palindrome.

2. Design CFG for a given language $L(G)=\{ a^ib^i \mid i \geq 0\}$

Solution: $L=\{ \in,ab,aabb,aaabbb,\ldots\}$

$S \rightarrow aSb \mid \in$

3. Design CFG for a given language $L(G)=\{ a^ib^i \mid i > 0\}$

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

$S \rightarrow aSb \mid ab$

4. Design CFG for a given language $L(G)=\{ ww^R \mid w \text{ is binary}\}$

Solution: $L=\{\in,00,11,0110,1001,010010,\ldots\}$

$S \rightarrow 0S0 \mid 1S1 \mid \in$

5. Design CFG for a given language $L(G)=\{ w\#w^R \mid w \text{ is binary}\}$

Solution: $L=\{\#,0\#0,1\#1,01\#10,10\#01,010\#010,\ldots\}$

$S \rightarrow 0S0 \mid 1S1 \mid \#$

6. Design CFG for a regular expression r=(a+b)*

Solution: L={∈,a,b,aa,ab,ba,bb,aaa,aab,bbb,bba,…}

S → aS | bS |∈

7. Give a CFG for the set of all well formed paranthesis.

Solution: S→SS | (S) | ( )

4.1 DERIVATION

- We derive strings in the language of a CFG by starting with the start symbol, and repeatedly replacing some variable A by the right side of one of its productions.
- That is, the "productions for A" are those that have A on the left side of the →.
- $\alpha A\beta \Rightarrow$ whenever there is a production A → γ
  - Subscript with name of grammar, e.g., $\overset{}{\underset{G}{\Rightarrow}}$, if necessary.

  Example: 011AS ⇒ 0110A1S
- $\alpha \overset{*}{\Rightarrow} \beta$ means string α can become β in zero or more derivation steps.

  Example: 011AS $\overset{*}{\Rightarrow}$ 011AS (zero steps);

  011AS $\overset{*}{\Rightarrow}$ 0110A1S (one step);

  011AS $\overset{*}{\Rightarrow}$ 0110011 (three steps);

Sentential Forms:

- Any string of variables and/or terminals derived from the start symbol is called a sentential form.
- Formally, α is a sentential form iff $S \overset{*}{\Rightarrow} \alpha$.

Types of Derivations:

→We have a choice of variable to replace at each step.

- Derivations may appear different only because we make the same replacements in a different order.
- To avoid such differences, we may restrict the choice.

1. **Left Most Derivation (LMD):** If at each step in a derivation a production is applied to the leftmost variable, then the derivation is called left most derivation.
2. **Right Most Derivation (RMD):** If at each step in a derivation a production is applied to the rightmost variable, then the derivation is called right most derivation.

→ $\overset{*}{\underset{lm}{\Rightarrow}}$, $\overset{*}{\underset{rm}{\Rightarrow}}$, etc., used to indicate derivations are leftmost and rightmost.

Derivation/Parse Trees:

Given a grammar with the usual representation G = (V, T, P, S) with variables V, terminal symbols T, set of productions P and the start symbol from V called S.

A derivation tree is constructed with

1) Each tree vertex is a variable or terminal or epsilon
2) The root vertex is S
3) Interior vertices are from V, leaf vertices are from T or epsilon
4) An interior vertex A has children, in order, left to right, X1, X2, ... , Xk when there is a production in P of the form A -> X1 X2 ... Xk
5) A leaf can be epsilon only when there is a production A -> epsilon and the leafs parent can have only this child.

Example 1: Construct parse tree for the following CFG and take input string is 0110011.

S → AS | ∈

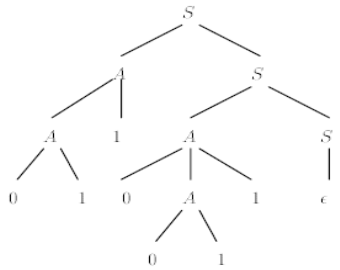
A → 0A1 | A1 | 01

Sol: Before constructing parse tree, first derive the given input string from the CFG.

LMD: S $\overset{*}{\underset{lm}{\Rightarrow}}$ AS $\overset{*}{\underset{lm}{\Rightarrow}}$ A1S $\overset{*}{\underset{lm}{\Rightarrow}}$ 011S $\overset{*}{\underset{lm}{\Rightarrow}}$ 011AS $\overset{*}{\underset{lm}{\Rightarrow}}$ 0110A1S $\overset{*}{\underset{lm}{\Rightarrow}}$ 0110011S $\overset{*}{\underset{lm}{\Rightarrow}}$ 0110011.

RMD: S $\overset{*}{\underset{rm}{\Rightarrow}}$ AS $\overset{*}{\underset{rm}{\Rightarrow}}$ AAS $\overset{*}{\underset{rm}{\Rightarrow}}$ AA $\overset{*}{\underset{rm}{\Rightarrow}}$ A0A1 $\overset{*}{\underset{rm}{\Rightarrow}}$ A0011 $\overset{*}{\underset{rm}{\Rightarrow}}$ A10011 $\overset{*}{\underset{rm}{\Rightarrow}}$ 0110011.

Parse tree:



4.2 Ambiguous Grammars:

A CFG is ambiguous if one or more terminal strings have multiple leftmost derivations or multiple rightmost derivations or multiple parse trees from the start symbol.

Example 1: Consider the following grammar:

S → AS | ∈

A → 0A1 | A1 | 01

The above CFG, the string 00111 has the following two leftmost derivations from S.

Sol:

LMD 1: S $\overset{lm}{\Rightarrow}$ AS $\overset{lm}{\Rightarrow}$ 0A1S $\overset{lm}{\Rightarrow}$ 0A11S $\overset{lm}{\Rightarrow}$ 00111S $\overset{lm}{\Rightarrow}$ 00111

LMD 2: S $\overset{lm}{\Rightarrow}$ AS $\overset{lm}{\Rightarrow}$ A1S $\overset{lm}{\Rightarrow}$ 0A11S $\overset{lm}{\Rightarrow}$ 00111S $\overset{lm}{\Rightarrow}$ 00111

Intuitively, we can use A → A1 first or second to generate the extra 1.

Example 2:

Consider the following grammar:

S → SS

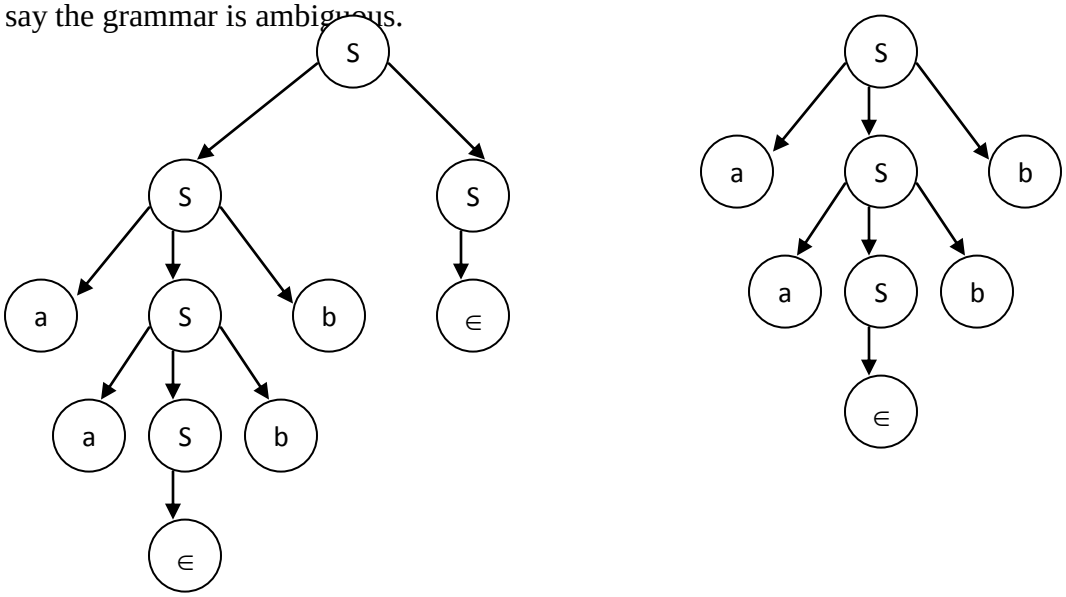
S → aSb

S → bSa

S → ∈

and the string w = aabb. We can draw the following 2 trees with the same string w = aabb, so we say the grammar is "ambiguous" in this case.

If we can find either 2 leftmost / rightmost derivations or 2 different derivation trees, then we can say the grammar is ambiguous.



4.3 Inherently Ambiguous context free language:

→ A CFL is said to be inherently ambiguous if every CFG that describes it is ambiguous every CFG that describes it is ambiguous

- A context free language for which we cannot construct an unambiguous grammar is inherently ambiguous CFL.

Example:

$L=\{a^n b^n c^m d^m \mid n\geq 1, m\geq 1\} \cup \{a^n b^m c^m d^n \mid n\geq 1, m\geq 1\}$

- An operator grammar is a CFG with no ∈-productions such that no consecutive symbols on the right sides of productions are variables.
- Every CFL without ∈ has an operator grammar.
- If all productions of a CFG are of the form A→xB or A→x, then L(G) is a regular set where x is a terminal string.

5. SIMPLIFICATION OF CFG

- In a CFG we may not use all the symbols for deriving a sentence. So, we eliminate symbols are productions in G, which are not useful.
- We can "simplify" grammars to a great extent. Some of the things we can do are:
  1. Elimination of useless symbols: those that do not participate in any derivation of a terminal string.
  2. Elimination of $\in$ - productions: those of the form variable $\rightarrow \in$.
  3. Elimination of Unit productions: those of the form variable $\rightarrow$ variable.

5.1 Eliminating Useless symbols:

- In order for a symbol X to be useful, it must:
  1. Derive some terminal string (possibly X is a terminal).
  2. Be reachable from the start symbol; i.e., $S \overset{*}{\Rightarrow} \alpha X \beta$
- Note that X wouldn't really be useful if α or β included a symbol that didn't satisfy (1), so it is important that (1) be tested first, and symbols that don't derive terminal strings be eliminated before testing (2).

Examples:

1. Eliminate useless symbols from the grammar

$S \rightarrow AB \mid a$
$A \rightarrow a$

Solution:

Here we find no terminal string is derivable from B. So that B is to be eliminated from productions $S \rightarrow AB$.

Remaking productions are

$S \rightarrow a$

$A \rightarrow a$

By rule 2, Here A is not useful to derive a string from starting symbol S. So we can eliminate A → a.

The final production is

S → a

2. Eliminate useless symbols from the grammar

S → aS | A | C

A → a

B → aa

C → aCb

Solution:

By rule 2, B is not useful to derive a string from starting symbol S. So we can eliminate

B → aa.

The Remaking productions are,

S → aS | A | C

A → a

C → aCb

By rule 1, C is not useful to derive some terminal string. So we can eliminate S → C and C→aCb productions.

The final productions are

S → aS | A

A → a

5.2 Eliminating ∈ - productions:

- A variable A is nullable if A $\overset{*}{\Rightarrow} \in$. Find them by a recursive algorithm.
  Basis: If A → ∈ is a production, then A is nullable.
  Induction: If A is the head of a production whose body consists of only nullable symbols, then A is nullable.
- Once we have the nullable symbols, we can add additional productions and then throw away the productions of the form A → ∈ for any A.
- If A → X1 X2 … Xk is a production, add all productions that can be formed by eliminating some or all of those Xi's that are nullable.

- But, don't eliminate all k if they are all nullable.

Examples:

1. Grammar G:

S → aS | bA

A → aA | ∈ , from this grammar eliminate ∈-productions.

Solution:

S → aS, S → bA gives S → bA and S → b

A → aA gives A → aA and A → a

After elimination of ∈-productions, the final grammar is

S → aS | bA | b

A → aA | a

2. Grammar G:

S → AaB | aaB

A → ∈

B → bbA | ∈, from this grammar eliminate ∈ - productions and then eliminate useless symbols.

Solution:

The given grammar is

S → AaB | aaB ……..……..(1)

A → ∈…… (2)

B → bbA | ∈ ……..…(3)

Step 1: Elimination of ∈ - productions

The given grammar contains two ∈-proctions. i.e A → ∈ and B → ∈

(1) ⇒ S → AaB | aaB | aB | Aa | a | aa {since A → ∈ and B → ∈ }

(2) ⇒ B → bbA | bb

The grammar is

S → AaB | aaB | aB | Aa | a | aa

B → bbA | bb

Step 2: Elimination of useless symbols from the above grmmar

S → AaB | aaB | aB | Aa | a | aa

B → bbA | bb

In this grammar Variable A is there, but it is not producing anything. So that it can eliminated.

The remaking productions are

S → aaB | aB | a | aa

B → bb

In this grammar no symbol is useless, then the final productions are,

S → aaB | aB | a | aa

B → bb

5.3 Eliminating Unit productions:

- The productions of the form A → B, where A,B ∈ V called unit production.
- Eliminate useless symbols and ∈ - productions.
- Discover those pairs of variables (A,B) such that $A \overset{*}{\Rightarrow} B$.
  - Because there are no ∈ - productions, this derivation can only use unit productions.
  - Thus, we can find the pairs by computing reachablity in a graph where nodes = variables, and arcs = unit productions.
- Replace each combination where $A \overset{*}{\Rightarrow} B \Rightarrow \alpha$ and α is other than a single variable by A → α.
  - i.e., “short circuit" sequences of unit productions, which must eventually be followed by some other kind of production.

Remove all unit productions.

Note: Consider the grammar G is S → A, A → B, B→ C, C → d.

Here A, B, C are the unit variables of length one. Then the resultant grammar is S → d. This is called the chain rule.

Example:

1. Eliminate unit productions from the following grammar.

S → A | bb

A → B |b

B → S | a

Solution:

In the given grammar, the unit productions are S → A, A → B and B → S.

S → A gives S → b.

S → A → B gives S → B gives S → a.

A → B gives A → a

A → B →S gives A → S gives A → bb.

B → S gives B → bb.

B → S → A gives B → A gives B → b.

The new productions are

S → bb | b | a

A → b | a | bb

B → a | bb | b

It has no unit productions. In order to get the reduced CFG, we have to eliminate the useless symbols. From the above grammar we can eliminate the A and B productions.

Then the resultant grammar is **S → bb | b | a**.

6. NORMAL FORMS

- In a Context Free Grammar, the right hand side of the production can be any string of variables and terminals. When productions in G satisfy certain restrictions, then G is said to be in a Normal Form.
- There are two widely useful Normal forms of CFG. They are

i. Chomsky Normal Form (CNF)

ii. Greibach Normal Form (GNF)

6.1 Chomsky Normal Form (CNF):

Definition: A context-free grammar G is in Chomsky normal form if any production is of the form:

$A \rightarrow BC$ or

$A \rightarrow a$

where 'a' is a terminal, A,B,C are non-terminals, and B,C may not be the start variable (the axiom)

Note:

1. In CNF number of symbols on right side of production strictly limited.
2. The rule $S \rightarrow \in$, where S is the start variable, is not excluded from a CFG in Chomsky normal form.

Conversion to Chomsky normal form:

Theorem: For every CFG, there is an equivalent grammar G in Chomsky Normal Form.

Proof:

Construction of grammar in CNF.

Step 1:

Eliminate null productions and unit productions.

Step 2:

Eliminate terminals on right hand side of productions as follows.

i. All the productions in P of the form $A \rightarrow a$ and $A \rightarrow BC$ are included.

ii. Consider $A \rightarrow w_1w_2 \ldots w_n$ will some terminal on right hand side. If w_i is a terminal say a_i, add a new variable c_{ai} and $c_{ai} \rightarrow P$. Repeat same for all terminals.

Step 3:

Restricting the number of variables on RHS as follows:

i. All the productions in P are added to P, if they are in the required form.

ii. Consider $A \rightarrow A_1A_2A_3 \ldots A_m$, then we introduce new productions are,

$A \rightarrow A_1C_1$

$C_1 \rightarrow A_2C_2$

$C_2 \rightarrow A_3C_3$

$C_{m-2} \rightarrow A_{m-1}C_{m-1}$

Example:

Convert the following CFG to Chomsky Normal Form (CNF):

$S \rightarrow aX \mid Yb$

$X \rightarrow S \mid \in$

$Y \rightarrow bY \mid b$

Solution:

Step 1 - Kill all $\in$ productions:

By inspection, the only nullable nonterminal is X.

Delete all $\in$ productions and add new productions, with all possible combinations of the nullable X removed.

The new CFG, without $\in$ productions, is:

$S \rightarrow aX \mid a \mid Yb$

$X \rightarrow S$

$Y \rightarrow bY \mid b$

Step 2 - Kill all unit productions:

The only unit production is $X \rightarrow S$, where the S can be replaced with all S's non-unit productions (i.e. aX, a, and Yb).

The new CFG, without unit productions, is:

$S \rightarrow aX \mid a \mid Yb$

$X \rightarrow aX \mid a \mid Yb$

$Y \rightarrow bY \mid b$

Step 3 - Replace all mixed strings with solid nonterminals.

Create extra productions that produce one terminal, when doing the replacement.

The new CFG, with a RHS consisting of only solid nonterminals or one terminal is:

$S \rightarrow AX \mid YB \mid a$

$X \rightarrow AX \mid YB \mid a$

$Y \rightarrow BY \mid b$

$A \rightarrow a$

$B \rightarrow b$

Step 4 - Shorten the strings of nonterminals to length 2.

All nonterminal strings on the RHS in the above CFG are already the required length, so the CFG is in CNF.

6.2 Greibach Normal Form (GNF):

A CFG G = (V, T, P, S) is said to be in GNF if every production is of the form A → aα, where a € T and α € V*, i.e., α is a string of zero or more variables.

Definition: A production U € R is said to be in the form left recursion, if U : A → Aα for some A € V .

Left recursion in R can be eliminated by the following scheme:

• If A → Aα1|Aα2| . . . |Aαr|β1| β2| . . . | βs, then replace the above rules by

(i) A → βi| βiZ, 1≤ i ≤ s and

(ii) Z →αi|αiZ, 1≤ i≤ r

• If G = (V, T, P, S) is a CFG, then we can construct another CFG G1 = (V1, T, P1, S) in Greibach Normal Form (GNF) such that L(G1) = L(G) − {∈}.

The stepwise algorithm is as follows:

1. Eliminate null productions, unit productions and useless symbols from the grammar G and then construct a G' = (V', T, P', S) in Chomsky Normal Form (CNF) generating the language L(G') = L(G) − {∈}.
2. Rename the variables like A1,A2, . . .An starting with S = A1.
3. Modify the rules in R' so that if Ai → Aj γ € R' then j > i.
4. Starting with A1 and proceeding to An this is done as follows:
   (a) Assume that productions have been modified so that for 1≤ i≤ k, Ai → Aj γ € R' then j > i.
   (b) If Ak → Ajγ is a production with j < k, generate a new set of productions substituting for the Aj the body of each Aj production.
   (c) Repeating (b) at most k − 1 times we obtain rules of the form Ak → Apγ, p≥ k
   (d) Replace rules Ak → Akγ by removing left-recursion as stated above.

5. Modify the Ai → Ajγ for i = n−1, n−2, ., 1 in desired form at the same time change the Z production rules.

Example: Convert the following grammar G into Greibach Normal Form (GNF).

S → XA|BB

B → b|SB

X → b

A → a

Solution:

To write the above grammar G into GNF, we shall follow the following steps:

1. Rewrite G in Chomsky Normal Form (CNF)

   It is already in CNF.

2. Re-label the variables

   S with A1

   X with A2

   A with A3

   B with A4

   After re-labeling the grammar looks like:

   A1 → A2A3|A4A4

   A4 → b|A1A4

   A2 → b

   A3 → a

3. Identify all productions which do not conform to any of the types listed below:

   Ai → Ajxk such that j > i

   Zi → Ajxk such that j ≤ n

   Ai → axk such that xk € V* and a € T

4. A4 → A1A4 ..... identified

5. A4 → A1A4|b.

To eliminate A1 we will use the substitution rule A1 → A2A3|A4A4.

Therefore, we have A4 → A2A3A4|A4A4A4|b

The above two productions still do not conform to any of the types in step 3.

Substituting for A2 → b

A4 → bA3A4|A4A4A4|b

Now we have to remove left recursive production A4 → A4A4A4

A4 → bA3A4|b|bA3A4Z|bZ

Z → A4A4|A4A4Z

6. At this stage our grammar now looks like

A1 → A2A3|A4A4

A4 → bA3A4|b|bA3A4Z|bZ

Z → A4A4|A4A4Z

A2 → b

A3 → a

All rules now conform to one of the types in step 3.But the grammar is still not in Greibach Normal Form.

7. All productions for A2,A3 and A4 are in GNF

for A1 → A2A3|A4A4

Substitute for A2 and A4 to convert it to GNF

A1 → bA3|bA3A4A4|bA4|bA3A4ZA4|bZA4

for Z → A4A4|A4A4Z

Substitute for A4 to convert it to GNF

Z → bA3A4A4|bA4|bA3A4ZA4|bZA4|bA3A4A4Z|bA4Z|bA3A4ZA4Z|bZA4Z

8. Finally the grammar in GNF is

A1 → bA3|bA3A4A4|bA4|bA3A4ZA4|bZA4

A4 → bA3A4|b|bA3A4Z|bZ

Z → bA3A4A4|bA4|bA3A4ZA4|bZA4|bA3A4A4Z|bA4Z|bA3A4ZA4Z|bZA4Z

A2 → b

A3 → a

7. Closure Properties of CFL's:

- The context-free languages are closed under
  - substitution
    - Let Σ be an alphabet and let L_a be a language for each symbol a in Σ. These languages define a substitution s on Σ.
    - If $w = a_1a_2 \ldots a_n$ is a string in Σ*, then $s(w) = \{ x_1x_2 \ldots x_n \mid x_i$ is a string in $s(a_i)$ for $1 \leq i \leq n \}$.
    - If L is a language, $s(L) = \{ s(w) \mid w$ is in L $\}$.
    - If L is a CFL over Σ and $s(a)$ is a CFL for each a in Σ, then s(L) is a CFL.
  - union
  - concatenation
  - Kleene star
  - homomorphism
  - reversal
  - intersection with a regular set
  - inverse homomorphism

8. Non-closure Properties of CFL's:

- The context-free languages are not closed under
  - intersection
    - $L_1 = \{ a^nb^nc^i \mid n, i \geq 0 \}$ and $L_2 = \{ a^ib^nc^n \mid n, i \geq 0 \}$ are CFL's. But $L = L_1 \cap L_2 = \{ a^nb^nc^n \mid n \geq 0 \}$ is not a CFL.
  - complement
    - Suppose comp(L) is context free if L is context free. Since $L_1 \cap L_2 = \text{comp}(\text{comp}(L_1) \cup \text{comp}(L_2))$, this would imply the CFL's are closed under intersection.
  - difference
    - Suppose $L_1 - L_2$ is a context free if L_1 and L_2 are context free. If L is a CFL over Σ, then comp(L) = Σ* - L would be context free.

9. Pumping Lemma for CFL's:

Pumping Lemma for CFL's is used to show that certain languages are non context free. There are three forms of pumping lemma.

1. Standard form of pumping lemma: For every non finite context-free language L, there exists a constant n that depends on L such that for all z in L with $|z| \geq n$, we can write z as $uvwxy$ where

1. $vx \neq \varepsilon$,
2. $|vwx| \leq n$, and

3. for all $i \geq 0$, the string uv^iwx^iy is in L.

One important use of the pumping lemma is to prove certain languages are not context free.

2. Strong form of pumping lemma (Ogden's Lemma): Let L is an infinite CFL. Then there is a constant n such that if z is any word in L, and we mark any n or more positions of z "distinguished", then we can write z=uvwxy such that

i. v and x together have at least one distinguished positions,

ii. vwx has atmost n distinguished positions, and

iii. for all i≥0, uv^iwx^iy is in L.

3. Weak form of pumping lemma: Let L is an infinite CFL. When we pump the length of strings are

$|uvwxy|=|uwy|+|vx|$

$|uv^2wx^2y|=|uwy|+2|vx|$

.....

$|uv^iwx^iy|=|uwy|+i|vx|$.

When we pump the lengths are in arithmetic progression.

Example:

1. The language L = { $a^nb^nc^n \mid n \geq 0$ } is not context free.

Solution:

The proof will be by contradiction. Assume L is context free. Then by the pumping lemma there is a constant *n* associated with L such that for all *z* in L with $|z| \geq n$, *z* can be written as *uvwxy* such that

1. $vx \neq \varepsilon$,
2. $|vwx| \leq n$, and
3. for all $i \geq 0$, the string uv^iwx^iy is in L.

Consider the string $z = a^nb^nc^n$.

From condition (2), *vwx* cannot contain both *a*'s and *c*'s.

- Two cases arise:
  1. *vwx* has no *c*'s. But then *uwy* cannot be in L since at least one of *v* or *x* is nonempty.
  2. *vwx* has no *a*'s. Again, *uwy* cannot be in L.

- In both cases we have a contradiction, so we must conclude L cannot be context free.

Applications of Grammars:

1. Specifying Syntax of programming languages.
2. Representing syntactic structures in natural languages.
3. Models of computation.