UNIT-III

- Formal Languages
- Grammars
- Classification of Grammars
- Chomsky Hierarchy Theorem
- Context free Grammar
- Left Most and Right-most Derivations
- Parse Trees
- Ambiguous Grammars
- Simplification of Context-Free Grammars
  - Elimination of Useless symbols
  - Elimination of ϵ-productions
  - Elimination of Unit productions
- Normal forms for Context Free Grammars
  - Chomsky Normal form
  - Greibach Normal form
- Pumping Lemma
- closure properties
- Applications of Context Free Grammars.

Grammars:

The grammar is basically defined as a set of 4-tuples (V, T, P, S). where

V is set of non-terminals (variables)
T is set of terminals
P is a set of productions (rules) that relate the non-terminals and terminals.
S is the start symbol with which strings in grammar are derived.

These productions define the strings belonging to the corresponding language. The motivation for these grammars was from the description of natural language using rules.

Ex:- Let us examine the rules used to define a sentence in English language.

<sentence> → <noun> <verd>
<noun> → <com-noun> | <prop-noun>
<verb> → ate | sat | ran
<com-noun> → Rama | Sita
<prop-noun> → She | He

Using these set of rules, many sentences can be derive by substituting for variables.

1. Rama ate
2. Sita sat
3. He ran
4. She sat.

Types of Grammars - Chomsky Hierarchy:

Linguist Noam Chomsky defined a hierarchy of langu in terms of complexity. This four level hierarchy, calle the Chomsky hierarchy, corresponds to four classes o machines.

The Chomsky hierarchy classifies grammars according to form of their productions into the following four levels.

(1) Type 0 grammars - Unrestricted grammars
(2) Type 1 grammars - Context sensitive grammars
(3) Type 2 grammars - Context free grammars
(4) Type 3 grammars - regular grammars.

(1) Type-0 grammars - Unrestricted Grammars: (URG)

These grammars include all formal grammars. In URGs, all the productions are of the form $\alpha \to \beta$, where α and β may have any number of terminals and non-terminals. ie, no restrictions on either side of productions.

Every grammar is included in it if it has atleast one non-terminal on the left hand side.

Ex:- $aA \to abCB$
$aA \to bAA$
$bA \to a$
$S \to aAb \mid \epsilon$

They generate exactly all languages that can be recognized by a turing machine. The language that is recognized by a Turing machine is defined as set of all the strings on which it halts. These Languages

are also known as the recursively enumerable languages.

(2) Type 1 grammar – Context Sensitive Grammars: (CSG)

These grammars define the context-sensitive languages. In Context Sensitive grammar, all the productions of the form $\alpha \to \beta$, where length of α is less than or equal to the length β. i.e $|\alpha| \leq |\beta|$. α and β may have any number of terminals and non-terminals. These languages are exactly all the languages that can be recognized by linear bound automata.

Ex:- $|\alpha| \leq |\beta|$ $\quad \alpha \to \beta$

$aAbcD \to abcDbcD$

(3) Type 2 Grammar - Context-free Grammar (CFG)

These grammars define the context-free languages. These are defined by rules of the form $\alpha \to \beta$ with $|\alpha| \leq |\beta|$ where $|\alpha| = 1$ and α is non-termind and β is a string of terminals and non-terminals. we can replace α by β regardless of where it appears. Hence the name context free grammar. These languages are exactly those languages that can be recognized by a pushdown automaton. CFG Context free languages defines the syntax of all programming languages.

Ex: (i) $S \to aS \mid Sa \mid a$

(ii) $S \to aAA \mid bBB \mid \epsilon$.

(4) Type 3 grammars - regular grammars:

These grammars generate the regular languages. Such a grammar restricts its rules to a single non-terminal on the LHS. The RHS consists of either a single terminal or string of terminals with single non-terminal on left or right end.

$\alpha \to \beta$, $\alpha = \{V\}$, $\beta = V\{T\}^*$ or $\{T\}^* V$ or T^*

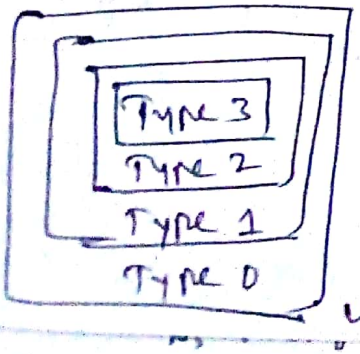
Eg: $A \to aA \mid a$ — right linear grammar

$A \to Aa \mid a$ — left linear grammar.

* Every regular language is context free, every context-free language is context-sensitive and every context-sensitive language is recursively enumerable.

Table: Chomsky's Hierarchy

Grammar	Language	Automaton	production rules
Type 0	Recursively enumerable	Turing machine	$\alpha \to \beta$ no restrictions on β, α α should have at least one non-terminals
Type 1	Context-sensitive	Linear bounded Automata	$\alpha \to \beta$ $\lvert\alpha\rvert \le \lvert\beta\rvert$
Type 2	Context-free	Push down automata	$\alpha \to \beta$ $\lvert\alpha\rvert = 1$
Type 3	Regular	Finite state automaton	$\alpha \to \beta$, $\alpha = \{V\}$ $\beta = \{V\}T^*$ or $= T^*\{V\}$ $= T^*$



Chomsky hierarchy of grammar

all the strings on which it runs.

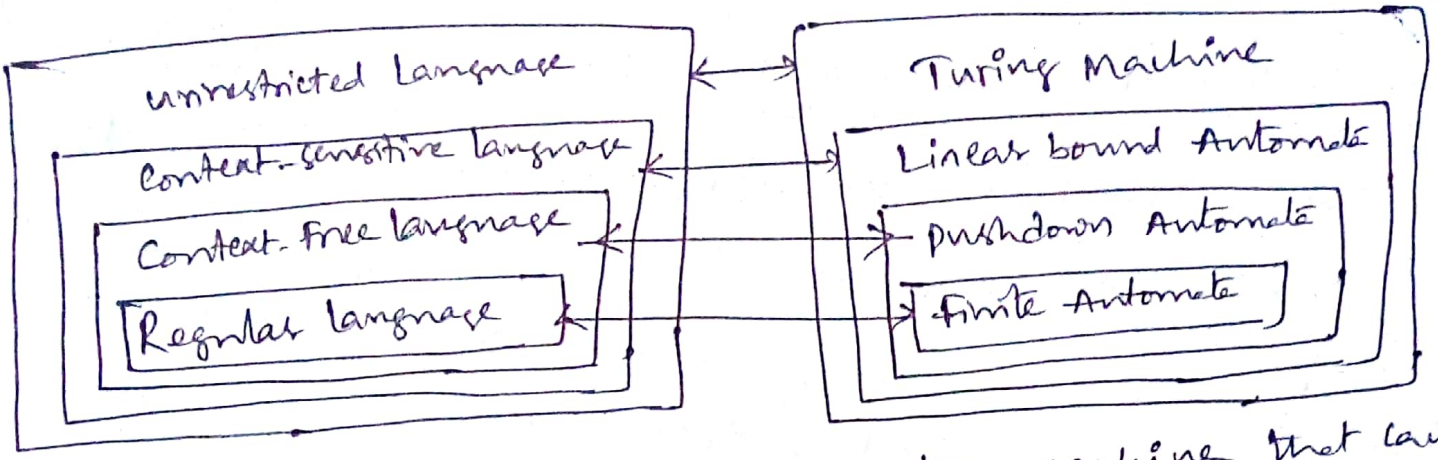


fig: The hierarchy of languages and the machine that can recognize the same as shown above fig.

. Every RG is context free, every CFG is context-sensitive and every CSG is unrestricted. So the family of regular language can be recognized by any machine.

. CFL's are recognized by pushdown automata, Linear bounded automata and Turing machines.

. CSLs are recognized by Linear bounded automata and Turing machines

. Unrestricted languages are recognized by only Turing machine

Context Free Grammars: (CFG)

A Context Free Grammar G is defined as four tuple

$G = (V, T, P, S)$ where

V - is a finite set of variables or non-terminals

T - is a finite set of terminals

P - is a finite set of production rules

S - is a start symbol, $S \in V$.

each production is of the form $A \to \alpha$, where A is a variable, α may be either terminal or non-terminal

ie $A \in V$, and $\alpha \in (V \cup T)^*$.

Ex: Productions: $S \to aS$

$S \to \epsilon$

is a simple CFG that defines $L(G) = a^*$

where $G = (V, T, P, S)$

$V = \{S\}$, $T = \{a\}$, S.

Ex: The CFG for defining palindrome over {a or b}.

Sol: The productions 'P' are $S \to aSa$

$S \to bSb$

$S \to a | b | \epsilon$

aa, aaaa, abba, baab.

and the grammar $G = (\{S\}, \{a, b\}, P, S)$

Ex: The CFG for the set of strings with equal number of a's and b's.

abab aabb bbaa

Sol: productions P are $S \to aSbS$

$S \to bSaS$

$S \to \epsilon$.

Grammar $G = (\{S\}, \{a, b\}, P, S)$

Ex:- CFG for generating syntatically correct algebric expressions with the variables x, y and z.

Sol:- The Grammar $G = (\{S, T\}, \{+, -, *, /, (,), x, y, z\}, P, S)$

productions P are:

$S \rightarrow T+S \mid T-S \mid T$

$T \rightarrow T*T \mid T/T$

$T \rightarrow (S)$

$T \rightarrow x \mid y \mid z$

the grammar can generate the string $(x+y)*x-z*y/(x+x)$

*

Ex:- For generating a language that generates equal number of a's and b's in the form $a^n b^n$, the context free grammar will be defined as

ab
aabb
aaabbb
=.

$G = \{(S, A), (a, b), S,$

$S \rightarrow aAb,$
$A \rightarrow aAb$
$A \rightarrow \epsilon \}.$

Sol:- $S \rightarrow aAb$
$\rightarrow aaAbb$ (by $A \rightarrow aAb$)
$\rightarrow aaaAbbb$ (by $A \rightarrow aAb$)
$\rightarrow aaabbb$ (by $A \rightarrow \epsilon$)
$\Rightarrow a^3 b^3$

Derivation of CFGs:

It is a process of defining strings out of a grammar by application of the rules starting from the start symbol we can derive the terminal strings, beginning with the start symbol and repeatedly replacing a variable/non-terminal by the body of the production.

The language of CFG is the set of terminal symbols we can derive using these productions and is called a Context Free Language.

Ex:- Derive a^4 from by the grammar

Terminal : a
Non-terminal : S
productions are: $S \to aS$
$S \to \epsilon$

Sol:-

$S \to aS$
$\to aaS \quad [\because S \to aS]$
$\to aaaS \quad [\because S \to aS]$
$\to aaaaS \quad [\because S \to aS]$
$\to aaaa \quad [\because S \to \epsilon]$

$\therefore$ The language has the strings $\{\epsilon, a, aa, aaa, aaaa, \ldots\}$.

Ex:-
[The set of all the strings that can be derived from a Grammar is said to be the language generated from that grammar].

Ex:- Derive a^2 from by the grammar

Terminal : a,
Non-terminal : S
productions : $S \to SS$
$S \to a$
$S \to \epsilon$.

Sol:- Derivation of a^2 can be achieved in many ways as

$S \to SS$
$S \to Sa \quad [\because S \to a]$
$S \to aa \quad [\because S \to a]$

(or)

$S \to SS$
$\to SSS \quad [\because S \to SS]$
$\to SSa \quad [\because S \to a]$
$\to SSSa \quad [\because S \to SS]$
$\to SaSa \quad [\because S \to a]$
$\to \epsilon aSa \quad [\because S \to \epsilon]$
$\to \epsilon ab_2 = ag_2$

Ex:- Find L(G) and derive the string abbab for the following grammar?

Terminal : a, b
Non-terminal : S
productions : $S \to aS$
$S \to bS$
$S \to a$
$S \to b$

Sol:- the Given grammar is

$S \to aS \mid bS \mid a \mid b$

we can derive abbab as follows

$S \to aS$
$\to abS \quad [\because S \to bS]$
$\to abbS \quad [\because S \to bS]$
$\to abbaS \quad [\because S \to aS]$
$\to abbab \quad [\because S \to b]$

The context free grammar generated by the grammar Language is $(a+b)^+$.

Ex: Derive the string aaabbb from the grammar

$G = \{ \{S, A\}, \{a, b\}, S, \{ S \to aAb, aA \to aaAb, A \to \epsilon \} \}$

Sol:

$S \to aAb \quad \because S \to aAb$

$\to aaAbb \quad \because aA \to aaAb$

$\to aaaAbbb \quad \because aA \to aaAb$

$\to aaa\epsilon bbb$

$\to aaabbb \quad \because A \to \epsilon$

Ex: Derive string ab from the Grammar $G = \{ \{S, A, B\}, \{a, b\}, S, \{ S \to AB, A \to a, B \to b \} \}$

Sol:

$S \to AB$

$\to aB \quad \because A \to a$

$\to ab \quad \because B \to b$

$L(G) = \{ab\}$

Ex: Derive set of string accepted by the language from the grammar $G = \{ \{S, A, B\}, \{a, b\}, S, \{ S \to AB, A \to aA | a, B \to bB | b \} \}$.

Sol:

$S \to AB$

$\to aB \quad [\because A \to a]$

$\to ab \quad [\because B \to b]$

$S \to AB$

$\to aAB \quad [\because A \to aA]$

$\to aaB \quad [\because A \to a]$

$\to aabb \quad [\because B \to bB]$

$\to aabb \quad [\because B \to b]$

$S \to AB$

$\to aAB$

$\to aaB$

$\to aab$

$S \to AB$

$\to aB$

$\to abB$

$\to abb$

$L(G) = \{ab, a^2b, ab^2, a^2b^2, \ldots\}$

$= \{a^m b^n \mid m \geq 1 \text{ and } n \geq 1\}$

for the given string sets,
(a) what is the language accepted by CFG
(b) what grammar is required to derive these strings
(c) what is the regular expression?

(i) $\{\epsilon, a, aa, \ldots a^n, \ldots\}$

Sol: $L(CFG) = \{a^n \mid n \geq 0\}$

The CFG $G = \{V, T, P, S\}$
$V = \{S\}$, $T = \{a\}$, S is the start symbol

$P = \{$
$S \to \epsilon$
$S \to aS$
$\}$

Regular Expression $RE = a^*$.

String: aaa
$S \to aS$
$\to aaS$ [$\because S \to aS$]
$\to aaaS$ [$\because S \to aS$]
$\to aaa\epsilon$ [$S \to \epsilon$]
$\to aaa$

(ii) $\{\epsilon, ab, aabb, \ldots, a^n b^n, \ldots\}$

Sol: $L(CFG) = \{a^n b^n \mid n \geq 0\}$

CFG $G = \{V, T, P, S\}$
$V = \{S\}$, $T = \{a, b\}$, S is start symbol

$P = \{$ $S \to \epsilon$
$S \to aSb$
$\}$

String: aaabbb

$S \to aSb$ [$\because S \to aSb$]
$\to aaSbb$ [$\because S \to aSb$]
$\to aaaSbbb$ [$\because S \to \epsilon$]
$\to aaa\epsilon bbb$
$\to aaabbb$

(iii) $\{\epsilon, ab, abab, \ldots (ab)^n, \ldots\}$

Sol: $L(CFG) = \{(ab)^n \mid n \geq 0\}$

CFG $G = \{V, T, P, S\}$

$V = \{S\}$, $T = \{a, b\}$, S is start symbol

Productions are $P = \{ S \to \epsilon$
$S \to abS$
$\}$

String ababab

$S \to abS$
$\to ababS$ [$S \to abS$]
$\to abababS$ [$S \to \epsilon$]
$\to ababab\epsilon$ [$\because S \to \epsilon$]
$\to ababab$

(iv) $\{a, aab, aabab, \ldots\}$

Sol: $L(CFG) = \{a.(ab)^n \mid n \geq 0\}$

Grammar $G = \{V, T, P, S\}$

$V = \{S\}$, $T = \{a, b\}$, S is start symbol

Productions are $P = \{ S \to a$, $S \to Sab \}$

String: aabab

$S \to Sab$
$\to Sabab$ [$\because S \to Sab$]
$\to aabab$ [$\because S \to a$]

Operations on Production Rules:

Suppose M and N are languages, whose grammars have disjoint sets of non-terminals. Also, assume that start symbols, for the grammars of M and N are A and B respectively. Then, following are the rules to find new grammars generated from M and N:

(·) Union Rule: The language M∪N start with two productions,

$$S \to A \mid B$$

(·) Product Rule: The language MN start with the productions

$$S \to AB$$

(·) Closure Rule: The language M^* starts with the productions

$$S \to AS \mid \in$$

Ex: Obtain a CFG for the language of palindrome over the alphabet $\Sigma = \{a, b, c\}$.

abccba aabbaa aba

Sol:-

CFG $G = \{V, T, P, S\}$

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

$P = \{ S \to aSa$
$S \to bSb$
$S \to cSc$
$S \to a|b|c|\epsilon$
$\}$

S is start symbol.

Derivation for the string

abcba

$S \to aSa$
$\to abSba$ [$\because S \to bSb$]
$\to abcba$ [$\because S \to c$].

cbaabc

$S \to cSc$
$\to cbSbc$ [$\because S \to bSb$]
$\to cbabc$ [$\because S \to aSa$]
$\to cbaSabc$ [$\because S \to aSa$]
$\to cbaεabc$ [$\because S \to \epsilon$]
$\to cbaabc$

Ex:- Obtain a CFG for the language of even palindrome, over the alphabet $\Sigma = \{a, b\}$

Sol $G = (V, T, P, S)$

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

$P = \{ S \to aSa$
$S \to bSb$
$S \to \epsilon$
$\}$

S is the start symbol

String: abba

$S \to aSa$
$\to abSba$ [$\because S \to bSb$]
$\to abεba$ [$\because S \to \epsilon$]
$\to abba$

Ex:- Obtain a CFG for the language of odd palindrome over alphabet $\Sigma = \{a, b\}$ (8)

Sol:- CFG $G = \{V, T, P, S\}$

$V = \{S\}$

$T = \{a, b\}$

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

S is start symbol

String: ababa

$S \to aSa$
$\to abSba$
$\to ababa$
$=$

String: babab

$S \to bSb$
$\to baSab$
$\to babab$
$=$

Ex:- Construct the CFG for the regular expression $(0+1)^*$.

Sol:- CFG $G = \{V, T, P, S\}$

$L(CFG) = \{\epsilon, 0, 1, 01, 10, 11, 00, 110, \cdots\}$

$V = \{S\}$, $T = \{0, 1\}$

$P = \{ S \to \epsilon$
$S \to 0S$
$S \to 1S$
$\}$

Ex:- Construct CFG which consists of all the strings having atleast one occurance of 000..

Sol:- R.E : $(0+1)^* 000 (0+1)^*$

$G = (V, T, P, S)$

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

$T = \{0, 1\}$

Start 'S'

$P = \{ S \to ATA$
$A \to 0A \mid 1A \mid \epsilon$
$T \to 000$
$\}$

Ex- All strings having at least two a's, over $\Sigma = \{a, b\}$

Sol:- $r.e = (a+b)^* a (a+b)^* a (a+b)^*$

$G = (V, T, P, S)$

$V = \{S, T\}$

$T = \{a, b\}$

$P = \{ S \to TaTaT$

$T \to aT \mid bT \mid \in$

$\}$

Strings

$S \to TaTaT$

$\to aa$ [$\because T \to \in$]

$S \to TaTaT$

$\to bTaTaT$

$\to b\in aTaT$

$\to babTaT$

$\to bab\in aT$

$\to babaT$

$\to babab$

$\to baba$

Construct a CFG generating all integers with sign.

Sol:- $G = (V, T, P, S)$

$V = \{ S, \text{sign}, \text{digit}, \text{integer} \}$

$T = \{0, 1, 2, \cdots, 9, +, -\}$, S start symbol

$P = \{ S \to \langle \text{sign} \rangle \langle \text{integer} \rangle$

$\langle \text{integer} \rangle \to \langle \text{digit} \rangle \langle \text{integer} \rangle \mid \langle \text{digit} \rangle$

$\langle \text{digit} \rangle \to 0 \mid 1 \mid 2 \mid \cdots \mid 9$

$\langle \text{sign} \rangle \to + \mid -$

Ex: integer -17

$S \to \langle \text{sign} \rangle \langle \text{integer} \rangle$

$\to - \langle \text{integer} \rangle$

$\to - \langle \text{digit} \rangle \langle \text{integer} \rangle$

$\to - 1 \langle \text{integer} \rangle$

$\to - 1 \langle \text{digit} \rangle$

$\to - 17$

LeftMost Derivation and Right Most Derivation

If a word w is generated by a CFG by a certain derivation and at each step in the derivation, a rule of production is applied to the left most non-terminal in the working string, then this derivation is called a left most derivation (LMD).

ie replace leftmost variable first in a string, then the resulting derivation is called the leftmost derivation. Similarly, replacing the right most variable first at every step gives the right most derivation (RMD).

Ex: Consider CFG $S \to aB|bA$, $A \to a|aS|bAA$, $B \to b|bS|aBB$. Find LMD and RMD for the string w = aabbabba

LMD aabbabba

$S \to aB$ [∵ B → aBB]
$\to aaBB$ [∵ B → bS]
$\to aabSB$ [∵ S → bA]
$\to aabbAB$ [∵ A → a]
$\to aabbabS$ [∵ B → bS]
$\to aabbabbA$ [∵ S → bA]
$\to aabbabba$ [∵ A → a]

RMD aabbabba

$S \to aB$
$\to aaBB$ [∵ B → aBB]
$\to aaBbS$ [∵ B → bS]
$\to aaBbbA$ [∵ S → bA]
$\to aaBbba$ [∵ A → a]
$\to aabSbba$ [∵ B → bS]
$\to aabbAbba$ [∵ S → bA]
$\to aabbabba$ [∵ A → a]

Ex:- Derive the string baaaababaab for leftmost derivation and right most derivation using CFG given:

$S \to baXaS \mid ab$

$X \to Xab \mid aa$

Sol

LMD baaaababaab

$S \to baXaS$

$\to baXabaS$ [X→Xab]

$\to baXababaS$ [X→Xab]

$\to baaaababaS$ [X→aa]

$\to baaaababaab$ [S→ab]

RMD

$S \to baXaS$

$\to baXaaab$ [S→ab]

$\to baXabaaab$ [X→Xab]

$\to baXababaaab$ [X→ab]

$\to baaaababaab$ [X→aa]

Parse Tree or Derivation Tree

The derivation process can be shown pictorially as a tree to illustrate how a word is derived from a CFG. These trees are called as syntax trees, parse tree or derivation trees.

These trees shows us clearly how the symbols of the terminal string are grouped into substrings, each of which belongs to the language of one of the variables of the grammar.

For constructing a parse tree for a grammar

$G = (V, T, P, S)$

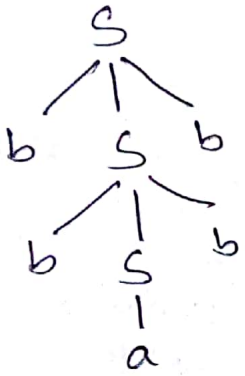
→ root is a start symbol

→ each node is labeled by either a variable, a terminal or ∈

- each interior node is marked by variable in V.
- If an interior node is labeled A, and its children are labeled $X_1, X_2, \cdots X_k$, from left, then $A \to X_1 X_2 \cdots X_k$ is a production in P.
- The leaf nodes are always the terminal symbols. no ter.

Ex:- Draw a derivation tree for 'bbabb' string for the CFG given by $S \to bSb \mid a \mid b$

Sol:

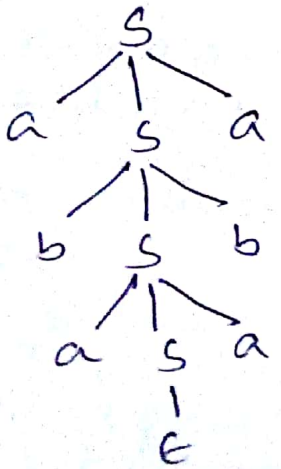


parse tree for bbabb
read the leaf nodes

Ex:- Draw a derivation tree for the string abaaba for the CFG given by G, where $P = \{ S \to aSa$
$S \to bSb$
$S \to a \mid b \mid \epsilon \}$

String abaaba

Sol



$\Rightarrow S \to aSa$

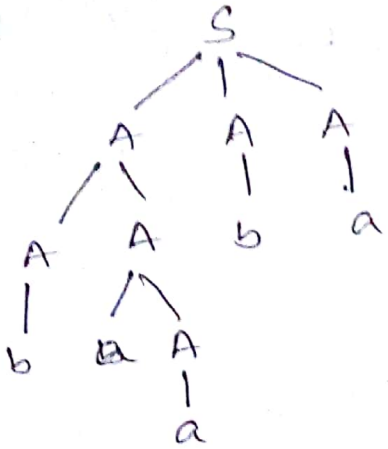
$\Rightarrow S \to bSb$

$\Rightarrow S \to aSa$

$S \to \epsilon$

Ex: Draw a derivation tree for the string baaba for the CFG given by G: S → AAA | aA
A → AA | aA | Ab | a | b }

Sol: String baaba

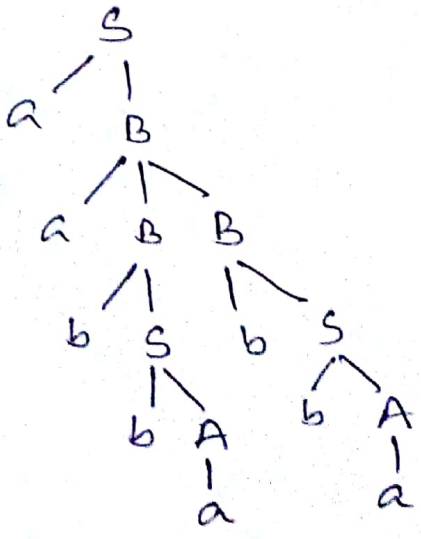


Derivation tree for baaba

S → AAA
→ AAba
→ bAba
→ bbAba
→ bbaba

Ex: Construct the derivation tree for the string aabbabba for the CFG given by S → aB | bA
A → a | aS | bAA
B → b | bS | aBB

Sol: String: aabbabba



Derivation tree

S → aB
→ aaBB
→ aabSB
→ aabbAB
→ aabbaBB
→ aabbabS
→ aabbabbA
→ aabbabba

Derivation

Ambiguous Grammar:

A CFG is ambiguous if there exist more than one parse tree, or equivalently, if there exist more than one leftmost derivation and thus there exist more than one right most derivation, for at least one word in its CFL.

ie. A grammar is ambiguous if there exist are more than one leftmost (or right most) derivation for given word w.

Ex:

Grammar

Ambiguous

- There exist more than one LMD or RMD for a string
- LMD & RMD represent different parse trees
- More than one parse tree for a string

unambiguous

- Unique LMD/RMD
- LMD & RMD represent same parse tree
- Unique parse tree.

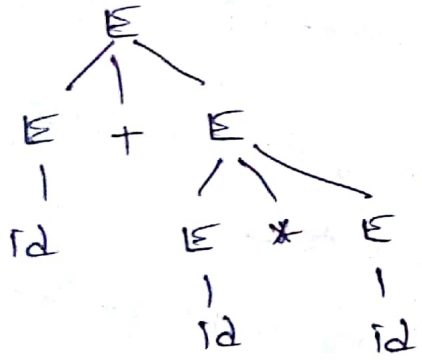
Ex:- Show that the following grammar is ambiguous

$E \to E+E$
$E \to E*E$
$E \to E-E$
$E \to id$ for string id+id*id.

Sol LMD id+id*id

$E \to E+E$
$\to id+E$
$\to id+E*E$
$\to id+id*E$
$\to id+id*id$

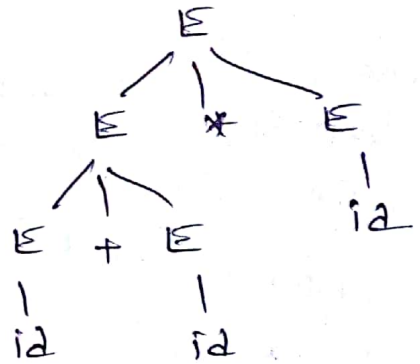
Parse tree



LMD id+id*id

$E \to E*E$
$\to E+E*E$
$\to id+E*E$
$\to id+id*E$
$\to id+id*id$

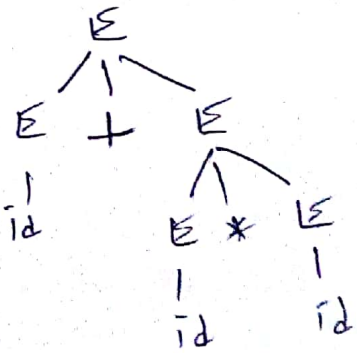
Parse tree



As there are more than one parse tree, the grammar is ambiguous.

RMD id+id*id

$E \to E+E$
$\to E+E*E$
$\to E+E*id$
$\to E+id*id$
$\to id+id*id$



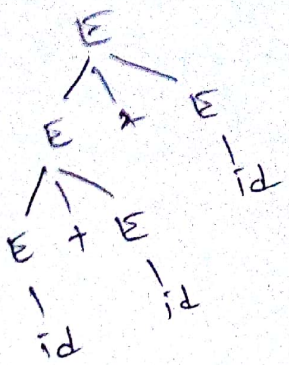
$E \rightarrow E * E$

$\rightarrow E * id$

$\rightarrow E + E * id$

$\rightarrow E + id * id$

$\rightarrow id + id * id$



Simplification of Grammars:-

The Grammar may contain extra/unnecessary symbols. These will increase the length of the grammar. Simplification of grammar involves of removing all these unnecessary symbols.

Simplification of CFG is done in 3 ways

(1) Elimination of useless symbols
(2) Elimination of ϵ - productions
(3) Elimination of Unit productions of the form $A \to B$.

Ex:-
$S \to AB$
$A \to a$
$B \to b|C$
$E \to c|\epsilon$

Here, C never defines a terminal and E is never reachable from S.

. E and C do not appear in any sentential form
. $E \to \epsilon$ is a null production
$B \to C$ simply replaces by B by C.

It is clear that removing these symbols and productions it would not effect the language generated, Hence the grammar can be simplified as

$S \to AB$
$A \to a$
$B \to b$

Elimination of useless symbols:

A symbol is useless if it cannot derive a terminal, it is not reachable from the start symbol.

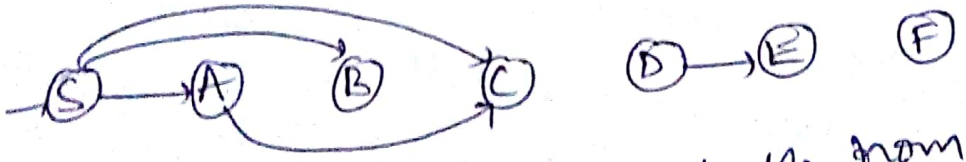
To check if the symbol is reachable from the start symbol, we can use dependency graph.

Ex: Eliminate useless symbols and productions from the following grammar.

$S \to ABa \mid BC$
$A \to aC \mid BCC$
$C \to a$
$B \to bCC$
$D \to E$
$E \to d$
$F \to e$

Sol Step 1: Eliminate useless symbols. All variables are found to be useful as each of them derive a terminal.

Step 2: Elimination of non-reachable variable.



So, D, E and F are non-reachable from S.

So after removing useless symbols, the grammar is

A $S \to ABa \mid BC$
$A \to aC \mid BCC$
$C \to a$
$B \to bCC$

$G = (V, T, P, S)$

useful

A grammar symbol X in CFG is useful, if and only if

- it derives a string of terminals
- it is used in derivation of at least one w in $L(CFG)$

i.e. (i) $X \xrightarrow{*} w$, where w is in T^*

(ii) $S \xrightarrow{*} \alpha X \beta \xrightarrow{*} w$, in $L(G)$

useless

A grammar symbol X in CFG is useless if,

- it does not derive a string of terminals,
- it does not occur in a derivation sequence of any w in $L(CFG)$.

Ex:- Eliminate useless symbols in the following grammar G.

$S \to BC \mid AB \mid CA$

$A \to a$

$C \to aB \mid b$

Sol:- Here B is not defined. Hence useless.

C and A are reachable and are deriving terminals.

Hence C and A useful.

The reduced grammar is

$S \to CA$

$A \to a$

$C \to b$

Eliminate useless symbols in the grammar G

$S \to aAa$
$A \to bBB$
$B \to ab$
$C \to ab$

Sol: Here C is useless, as it is not reachable from S. So, the reduced grammar is

$S \to aAa$
$A \to bBB$
$B \to ab$

Eliminate useless symbols in the following grammar G.

Sol: $S \to aS \mid A \mid C$
$A \to a$
$B \to aa$
$C \to aCb$

Sol:- Here C is useless, as it is not deriving any string. B is not reachable. So, the reduced grammar is

$S \to aS \mid A$
$A \to a$

$S \Rightarrow aS \Rightarrow aA \Rightarrow aa$.
$A \to a$
$B \to aa$
$C \to aCb \Rightarrow aa\underline{Cbb}$.

Eliminate the useless symbols in the following grammar.

$S \to AB \mid CA$, $B \to BC \mid AB$, $A \to a$, $C \to aB \mid b$.

Sol: Here B is not deriving any string. So B is useless symbol. Hence, the reduced grammar is

$S \to CA$
$A \to a$
$C \to b$

Elimination of ∈- productions:

A production of the form $X \to \epsilon$ is called an ∈-production. If X is a non-terminal $A \xrightarrow{*} \epsilon$, then A is called a 'Nullable non terminal'.

Procedure for eliminating ∈-productions:

(i) Construct V_n, the set of nullable variables
(ii) For each production $B \to A$, if A is nullable variable, replace nullable variable A by ∈, and add, all possible combinations of strings on the RHS of production.
(iii) Do not add the production $A \to \epsilon$.

Eg:- Eliminate the null (ϵ) production from the following grammar

$S \to ABaC$
$A \to BC$
$B \to b \mid \epsilon$
$C \to D \mid \epsilon$
$D \to d$.

Sol: Nullable variables are given by $V_n = \{B, C, A\}$

Hence equivalent grammar without null productions is

$S \to ABaC \mid BaC \mid AaC \mid ABa \mid aC \mid Aa \mid Ba \mid a$

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

$B \to b$

$C \to D$

$D \to d$

Ex:- Eliminate the null productions from the following grammar

$S \to aA$

$A \to BB$

$B \to aBb \mid \epsilon$

Sol: Nullable variables are $V_n = \{B, A\}$

Hence equivalent grammar without null productions is

$S \to aA \mid \epsilon$

$A \to BB \mid B$

$B \to aBb \mid ab$

Ex:- Eliminate ϵ-productions from the following grammar.

$S \to aS \mid AB$

$A \to \epsilon$

$B \to \epsilon$

$D \to b$

Sol Nullable variables are $V_n = \{A, B, S\}$

Hence, the equivalent grammar without nullable variables are is

$S \to aS \mid a \mid AB \mid A \mid B$

$D \to b$

Ex:- Eliminate null productions productions from the following grammar $S \to AaA$, $A \to Sb \mid bCC \mid \epsilon$, $C \to CC \mid abb$

Sol Nullable variable $V_n = \{A\}$

Hence, the equivalent grammar without Nullable variable is

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

$A \to Sb \mid bCC$

$C \to CC \mid abb$

Eliminating Unit Productions:

A unit product is a production of the form $A \to B$.

Here A and B are non terminals.

- presence of Unit productions in a grammar increases the cost of derivations.

procedure for eliminating Unit productions:

- For each pair of non-terminals A and B such that there is a production $A \to B$ and non-Unit productions from B are $B \to s_1 | s_2 | \ldots | s_n$, where $s_i \in (V+T)^*$ are strings of terminals and non-terminals, then create the new productions as

$$A \to s_1 | s_2 | \ldots | s_n.$$

- Do the same for all such pairs A and B simultaneously
- Remove all unit productions.

Ex: Eliminate Unit productions in the grammar

$S \to Aa | B$, $B \to A | bb$, $A \to a | bc | B$.

Sol:- Unit productions are $S \to B$, $B \to A$, $A \to B$

Eliminate $S \to B$,

Substitute the alternative of B to $S \to B$ i.e

$S \to Aa | B \Rightarrow S \to Aa | A | bb$

But $S \to A$ is a unit production.

Substitute alternative of A to $S \to A$, then

$S \to Aa | a | bc | bb$

Eliminate $B \to A$,

Substitute the alternative of A to $B \to A$, then

$B \to A|bb \Rightarrow B \to a|bc|bb$.

Eliminate $A \to B$,

Substitute the alternative of B to $A \to B$, then

$A \to a|bc|B \Rightarrow A \to a|bc|bb$.

The final set of productions after eliminating unit productions,

$S \to Aa|a|bc|bb$

$A \to a|bc|bb$

$B \to a|bc|bb$.

Simplify the following grammar

$S \to aA|aBB$

$A \to aAA|\epsilon$

$B \to bB|bbC$

$C \to B$

SA: Here it is better to eliminate null productions as this may introduce useless symbols and unit productions. Next eliminate unit productions and at the end eliminate useless symbols.

. Elimination of ϵ-productions. $A \to \epsilon$.

after elimination of ϵ-production, the grammar is

$S \to aA|a|aBB$

$A \to aAA|aA|a$

$B \to bB|bbC$

$C \to B$.

Here the Unit production is $C \to B$.

Eliminate Unit production, then the resulting grammar is

$S \to aA \mid a \mid aBB$

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

$B \to bB \mid bbC$

$C \to bB \mid bbC$.

Elimination of Useless symbols.

Here B and C are not deriving any terminals.

So, B and C are not useful symbols. After elimination of these useless symbols, the grammar is

$S \to aA \mid a$

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

Finally, the minimized grammar is

$S \to aA \mid a$

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

which defines any number of a's

Normal forms:

In CFG, on the right hand side of the production, there are any number of terminals and non-terminals, in any combination. So, we need to normalize such a grammar in a specific format. There should be a fixed number of terminals or non-terminals in context free grammar with some crieteria.

There are two important normal forms.

(1) Chomsky Normal Form (CNF)

(2) Greibach Normal Form (GNF)

(1) CHOMSKY Normal form: (CNF)

A CFG is in Chomsky Normal form, if each production has one of the two forms

(1) Non-Termind → string of exactly two Non-terminals ie $A \to BC$

(2) Non-termind → one termind, ie $A \to a$.

Procedure for Converting a CFG to CNF.

(1) Eliminate null productions and Unit productions.

(2) Exclude productions of the form $A \to BC \mid a$ as it is

(3) Eliminate strings of terminals on the right-hand side of productions if they exceed one as follows

↳ Suppose we have the production

$S \to a_1 a_2 a_3$ where $a_1 a_2 a_3$ are terminals

- then introduce non-terminal C_{a_i} for terminal a_i as

$C_{a_1} \to a_1$, $C_{a_2} \to a_2$, $C_{a_3} \to a_3$.

4. To restrict the number of variables on the right hand side, introduce new variables and sp separate them as follows

→ Suppose we have the production with n non-terminals as shown below with 5 non-terminals.

$Y \to X_1 X_2 X_3 X_4 X_5$.

add new N.T to the productions we get new and the modify the production as in the following

$Y \to X_1 R_1$
$R_1 \to X_2 R_2$
$R_2 \to X_3 R_3$
$R_3 \to X_4 X_5$.

where R_i are new non-terminals.

Ex:- Convert following CFG to CNF

$S \to bA \mid aB$
$A \to bAA \mid aS \mid a$
$B \to aBB \mid bS \mid b$

(a):- There are no null or unit productions.

$A \to a$, $B \to b$ are in the required format.

and other productions are not in CFG.

So, replace every terminal by the following variables :

$S \rightarrow C_bA \mid C_aB$

$A \rightarrow C_bAA \mid C_aS \mid a$

$B \rightarrow C_aBB \mid C_bS \mid b$

$C_a \rightarrow a$

$C_b \rightarrow b$

Here, C_bAA and C_aBB are not in the CNF. So add new non-terminals D and E

$S \rightarrow C_bA \mid C_aB$

$A \rightarrow C_bD \mid C_aS \mid a$

$D \rightarrow AA$

$B \rightarrow C_aE \mid C_bS \mid b$

$E \rightarrow BB$

$C_a \rightarrow a$

$C_b \rightarrow b$

This is in CNF.

Convert the following CFG to CNF

$S \to AB \mid aB$

$A \to aaB \mid \epsilon$

$B \to bbA$

Sol:- Eliminate ϵ productions

$S \to AB \mid aB \mid B$

$A \to aab$

$B \to bbA \mid bb$

Eliminate Unit productions

$S \to AB \mid aB \mid bbA \mid bb$

$A \to aab$

$B \to bbA \mid bb$

Replace terminals by a Variable

$S \to AB \mid C_aB \mid C_bC_bA \mid C_bC_b$

$C_a \to a$

$C_b \to b$

$A \to C_aC_aC_b$

$B \to C_bC_bA \mid C_bC_b$

Restrict to two Variables on the right hand side of each production

$S \to AB \mid C_aB \mid C_bD \mid C_bC_b$

$D \to C_bA$

$A \to C_aE$

$E \to C_aC_b$

$B \to C_bD \mid C_bC_b$

$C_a \to a, \quad C_b \to b$

Convert the following CFG to CNF.

$S \to ASB \mid \epsilon$
$A \to aAS \mid a$
$B \to SbS \mid A \mid bb$

Sol:- (1) Elimination ε production

$S \to ASB \mid AB$
$A \to aAS \mid aA \mid a$
$B \to SbS \mid Sb \mid bS \mid b \mid A \mid bb.$

(2) Elimination of Unit productions

$S \to ASB \mid AB$
$A \to aAS \mid aA \mid a$
$B \to SbS \mid Sb \mid bS \mid bb \mid b \mid aAS \mid aA \mid a$

(3) Elimination of useless symbols

Here all variables are generating string and are reachable from the start symbol.

So, the simplified grammar is

$S \to ASB \mid AB$
$A \to aAS \mid aA \mid a$
$B \to SbS \mid Sb \mid bS \mid bb \mid b \mid aAS \mid aA \mid a$

Now, Convert these productions into CNF.

(i) Replace ~~Eliminate~~ terminals from RHS by other prod.

$S \to ASB \mid AB$

$A \to C_aAS \mid a \mid C_aA$

$C_a \to a$

$B \to SC_bS \mid C_bC_b \mid SC_b \mid C_bS \mid b \mid C_aAS \mid a \mid C_aA$

$C_b \to b$

(ii) Restrict to two variables on the right-hand side of each production.

$S \to AD \mid AB$

$D \to SB$

$A \to C_aE \mid a \mid C_aA$

$E \to AS$

$B \to SF \mid C_bC_b \mid SC_b \mid C_bS \mid b \mid C_aE$

$F \to C_bS$

$C_b \to b$

The Grammar in CNF form $G'' = (V'', \{a, b\}, P', S)$

$V' = \{S, A, B, C_a, C_b, D, E, F\}$

P':

$S \to AD \mid AB$

$D \to SB$

$A \to C_aE \mid a \mid C_aA$

$E \to AS$

$B \to SF \mid C_bC_b \mid SC_b \mid C_bS \mid b \mid C_aE$

$F \to C_bS$

$C_b \to b$.

Convert following CFG to CNF.

P: $S \to ASA \mid aB$

$A \to B \mid S$

$B \to b \mid \in$

Sol:- Eliminate $\in$-productions

$S \to ASA \mid aB \mid a \mid SA \mid AS \mid S$ [$\because A \to \in$ $\because B \to \in$, $A \to B$]

$A \to B \mid S \mid \in$

$B \to b$

Eliminate unit productions [$\because S \to S$, $A \to B$, $A \to S$]

$S \to ASA \mid aB \mid a \mid SA \mid AS$

$A \to b \mid ASA \mid aB \mid a \mid SA \mid AS$

$B \to b$

Replace Terminals by non Terminals.

$S \to ASA \mid AaB \mid a \mid AS \mid SA$

$A \to b \mid ASA \mid AaB \mid a \mid SA \mid AS$

$B \to b$

$Aa \to a$

Restrict to two variables on RHS of each productions

$S \to AC \mid AaB \mid a \mid AS \mid SA$

$C \to SA$

$A \to b \mid AC \mid AaB \mid a \mid SA \mid AS$

$B \to b$ $\quad Aa \to a$

Greibach Normal Form (GNF)

A Context free grammar (CFG) is in GNF if the productions are in the following form

$A \to b$

$A \to bC_1C_2 \cdots C_n$

where $A, C_1, C_2, \ldots C_n$ are non terminals and b is terminal.

For converting a given grammar to GNF, we have two lemmas.

(1) Lemma 1. (Substitution rule)

Let $G = (V, T, P, S)$ be a CFG. Let $A \to B\alpha$ be a production in P if there is a production $B \to B_1 | B_2 | B_3 | B_4 \ldots$. Then the equivalent grammar can be obtained by substituting B in A.

The resulting grammar is

$A \to B_1\alpha | B_2\alpha | B_3\alpha | B_4\alpha$.

(2) Lemma 2 (Elimination of left recursion)

Grammar of the form $A \to A\alpha | B$ is called left-recursive grammar. To eliminate left recursion, rewrite grammar as

$A \rightarrow \beta A'$

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

If we eliminate ϵ productions, the grammar is

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \mid \alpha$

We can generalize this grammar.

For any CFG given as

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid A\alpha_3 \cdots \mid \beta_1 \mid \beta_2 \mid \beta_3 \cdots$$

The equivalent grammar after eliminating left recursion is

$$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \beta_3 A' \cdots \mid \beta_1 \mid \beta_2 \mid \beta_3$$

$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \alpha_3 A' \cdots \mid \alpha_1 \mid \alpha_2 \mid \alpha_3 \cdots$$

Procedure to Convert the CFG to GNF

Step 1: Eliminate null productions and unit productions and Construct CNF.

Step 2: Rename variables as $A_1, A_2, A_3, \ldots$ Starting with S as A_1 (Variables names in ascending order A_i)

Step 3: For each production of the form $A_i \rightarrow A_j\alpha$, apply the following

(a) if $j > i$ leave the productions as they are
(b) if $j = i$ apply lemma 2 [∵ Left Recursion]
(c) if $j < i$ apply lemma 1 [∵ Substitution]

Step 4: For each production of the form $A_i \to A_j\alpha$, where $j > i$, apply substitution lemma. If A_j is in GNF to bring A_i to GNF.

Ex: Convert the CFG to GNF.

$S \to AA \mid a$

$A \to SS \mid b$.

Sol: ~~Step 1:~~ Here, there is no null productions and unit productions. the given grammar is in the form of CNF.

$S \to AA \mid a$

$A \to SS \mid b$

Step 2: Rename the variables.. S and A as A_1 and A_2 respectively.

$S = A_1$

$A = A_2$

then, the grammar is

$A_1 \to A_2A_2 \mid a$

$A_2 \to A_1A_1 \mid b$.

Step 3: Consider $A_1 \to A_2$ and $A_2 \to b$ are in GNF.

Apply Lemma 1 for $A_2 \to A_1A_1$, then the productions becomes

$A_1 \to A_2A_2 \mid a$

$A_2 \to A_2A_2A_1 \mid aA_1 \mid b$

Step 4: Elimination of Left Recursion (Lemma 2)

$A_2 \to A_2 A_2 A_1 \mid aA_1 \mid b$

$A_2 \to b \mid aA_1 \mid A_2 A_2 A_1$.

The production $A_2 \to A_2 A_2 A_1$ is in left recursion.

According to Lemma 2, introduce new variable Z,

$Z \to A_2 A_1 Z \mid A_2 A_1$

Now $A_2 \to b \mid aA_1 \mid bZ \mid aA_1 Z$

$Z \to A_2 A_1 Z \mid A_2 A_1$

Step 5: Convert the grammar into proper form by using Lemma 1 (substitution).

$A_1 \to A_2 A_2 \mid a$

$A_2 \to b \mid aA_1 \mid bZ \mid aA_1 Z$

$Z \to A_2 A_1 Z \mid A_2 A_1$ after substitution,

these grammar is

$A_1 \to bA_2 \mid aA_1 A_2 \mid bZA_2 \mid aA_1 ZA_2 \mid a$

$A_2 \to b \mid aA_1 \mid bZ \mid aA_1 Z$

$Z \to bA_1 Z \mid aA_1 A_1 Z \mid bZA_1 Z \mid aA_1 ZA_1 Z \mid$
$bA_1 \mid aA_1 A_1 \mid bZA_1 \mid aA_1 ZA_1$.

The above grammar is in GNF.

Ex: Convert the following CFG to GNF

$S \to XA \mid BB$

$B \to b \mid SB$

$X \to b$

$A \to a$

Sol:- Step1: The given grammar is in CNF.

i.e There is no unit productions

There is no null productions

Step2: Rename the variables.

S with A_1

X with A_2

A with A_3

B with A_4

then the productions are

$A_1 \to A_2A_3 \mid A_4A_4$

$A_4 \to b \mid A_1A_4$

$A_2 \to b$

$A_3 \to a$

Here $A_2 \to b$ and $A_3 \to b$ and $A_4 \to b$ are in GNF. Remaining productions are not in GNF.

Step3: Identify the productions that do not in the GNF. $A_4 \to A_1A_4$ is not in GNF. [$A_i \to A_j\alpha$ if $j < i$, then apply Lemma 1].

Lam

then apply lemma (1), for A_4.

$A_4 \to A_1 A_4 | b$ becomes substitute A_1 prod

$A_4 \to A_2 A_3 A_4 | A_4 A_4 A_4 | b$

Substitute $A_2 \to b$, then

$A_4 \to b A_3 A_4 | A_4 A_4 A_4 | b$.

Now the production $A_4 \to A_4 A_4 A_4$ is in left recu

then apply Lemma(2), then the productions are

$Z \to A_4 A_4 Z | A_4 A_4$

$A_4 \to b A_3 A_4 | b | b A_3 A_4 Z | bZ$.

Now the resulting grammar is

$A_1 \to A_2 A_3 | A_4 A_4$

$A_4 \to b A_3 A_4 | b | b A_3 A_4 Z | bZ$

$Z \to A_4 A_4 Z | A_4 A_4$.

$A_2 \to b$

$A_3 \to a$.

Still the grammar is not in GNF.

Step 5: A_4, A_2 and A_3 productions are in GNF, and A_1 a

Z are not in GNF.

$A_1 \to A_2 A_3 | A_4 A_4$

Substitute for A_2 and A_4 to convert into GNF (Lemma).

$A_1 \rightarrow bA_3 \mid bA_3A_4A_4 \mid bA_4 \mid bA_3A_4ZA_4 \mid bZA_4$

for $Z \rightarrow A_4A_4Z \mid A_4A_4$

Substitute A_4 to convert into GNF

$Z \rightarrow bA_3A_4A_4Z \mid bA_4Z \mid bA_3A_4ZA_4Z \mid bZA_4Z \mid$

$bA_3A_4A_4 \mid bA_4 \mid bA_3A_4ZA_4 \mid bZA_4.$

Step 6: The final grammar is

$A_1 \rightarrow bA_3 \mid bA_3A_4A_4 \mid bA_4 \mid bA_3A_4ZA_4 \mid bZA_4$

$A_4 \rightarrow bA_3A_4 \mid b \mid bA_3A_4Z \mid bZ$

$Z \rightarrow bA_3A_4A_4Z \mid bA_4Z \mid bA_3A_4ZA_4Z \mid bZA_4Z \mid$

$bA_3A_4A_4 \mid bA_4 \mid bA_3A_4ZA_4 \mid bZA_4$

$A_2 \rightarrow b$

$A_3 \rightarrow a.$

Greibach Normal form (71)

A CFG is in GNF if the productions are in the following forms.

$A \to b$

$A \to bC_1C_2 \ldots C_n$.

where $A, C_1, C_2, \ldots C_n$ are Non-Terminals and b is terminal.

Steps to Convert a given CFG to GNF

Step 1: check if the given CFG has any Unit productions or Null productions and Remove if there are any.

Step 2: Check whether the CFG is already in CNF and Convert it to CNF if it is not.

Step 3: Change the names of the Non-terminal symbols into some A_i in ascending order of i.

Step 4: After the order so that the non-terminals are in ascending order, check that, if the productions is of the form $A_i \to A_j X$, then, $i < j$ and should never be $i \geq j$.

Step 5: Remove Left Recursive

Ex:- $S \to CA \mid BB$
$B \to b \mid SB$
$C \to b$
$A \to a$
} These are in CNF - Not req

Replace: S with A_1
C with A_2
A with A_3
B with A_4

we get- $A_1 \to A_2A_3 \mid A_4A_4$

$A_4 \to b \mid A_1A_4$

$A_2 \to b$

$A_3 \to a$.

Step: $A_i \to A_jX$ Then $i<j$ and should never be $i \geq j$.

Then $A_1 \to A_2A_3 \mid A_4A_4$ ✓

$A_4 \to b \mid A_1A_4$ problem replace $A_1 \to A_2A_3 \mid A_4A_4$.

replace $A_4 \to b \mid A_2A_3A_4 \mid A_4A_4A_4$ ✗

replace $A_2 \to b$

$A_4 \to b \mid bA_3A_4 \mid A_4A_4A_4$ ✗ $i=j$, $A_4 \to A_4A_4$
Left recursion.

Step 5: Remove left Recursion.

Step 5 : Remove Left Recursion

$A_4 \to A_4 A_4 A_4$ — is in left Recursion.

Introduce a new variable to remove the left recursion.

$A_4 \to b \mid bA_3A_4 \mid A_4A_4A_4$

(so) write a new variable that contains without [illegible] variable and new variable and [illegible] without

Introduce a variable 'Z'

$Z \to A_4 A_4 Z \mid A_4 A_4$

then $A_4 \to b \mid bA_3A_4 \mid \underline{bZ} \mid \underline{bA_3A_4Z}$

write along with new variable

$Z \to A_4A_4 \mid A_4A_4Z.$

$A_2 \to b$

$A_3 \to a$

Now the grammar is

$A_1 \to A_2A_3 \mid A_4A_4$

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

$Z \to A_4A_4 \mid A_4A_4Z$

$A_2 \to b$

$A_3 \to a$

Step 6, then ~~$A_1 \to b A_3$~~ $\mid b A_4A_4 \mid$ ~~$bA_3A_4A_4A_4$~~ $\mid$

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

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

$Z \to bA_4 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4ZA_4 \mid$
$bA_4Z \mid bA_3A_4A_4A_4Z \mid bZA_4Z \mid bA_3A_4ZA_4Z$

(13) Pumping Lemma (for Context Free Languages) (1)

- Pumping Lemma (for CFL) is used to prove that a language is NOT Context Free Language.

⇒ Context Free Language:

- In formal language theory, a context free language is a language generated by some context free Grammar
- The set of all CFL is identical to the set of language accepted by Push down Automata.
- Context Free Grammar defined by 4 tuples as

  $G = \{V, \Sigma, S, P\}$ where

  V = set of variables or Non-terminals

  Σ = set of Terminal Symbols

  S = Start Symbol

  P = production Rule

- Context Free Grammar has production rule of the form $A \to \alpha$, where $\alpha \in \{V \cup \Sigma\}^*$ and

  $A \in V$.

=

If A is a context free Language, then, A has a pumping length 'p' such that any string 's' where $|s| \geq p$ may be divided into 5 pieces $s = uvwxyz$ such that the following conditions must be true

(i) $uv^i x y^i z$ is in A for every $i \geq 0$

ii) $|vy| > 0$

iii) $|vxy| \leq p$

⇒ To prove that a Language is not Context free using Pumping Lemma (for CFL) follow the steps given below: (we prove using Contradiction)

→ Assume that A is context free

→ It has to have a pumping Length (say P)

→ All strings longer than P can be pumped $|s| \geq p$

→ Now find a string 's' in A such that $|s| \geq p$

→ Divide S into uvwxyz

→ Show that $uv^i wxy^i z \notin A$ for some i

→ Then consider the ways that S can be divided into uvwxyz

→ Show that none of these can satisfy all 3 pumping conditions at the same time.

→ S cannot be pumped == Contradiction.

(4) Pumping Lemma (for Context Free Languages)-

Ex:- Show that $L = \{a^N b^N c^N \mid N \geq 0\}$ is not context free.

Sol:
→ Assume that L is context free.
→ L must have a pumping length (say P)
→ Now we take a string S such that $S = a^P b^P c^P$
→ we divide S into parts UVWXYZ.

Eg: P = 4, bb, $S = a^4 b^4 c^4$

Case I: V and Y each contains only one type of Symbol.

aaaabbbbccccc
U V X Y Z

$UV^iXY^iZ \quad (i=2)$

UV^2XY^2Z

- UV^iXY^iZ

Put i=2, i.e. UV^2XY^2Z

⇒ aaaaaabbbbcccc

⇒ $a^6b^4c^5 \notin L$

Case II: Either V or Y has more than one kind of symbols.

$a^4b^4c^4$

⇒ aaaabbbbcccc
V X Y Z

- UV^iXY^iZ

Put i=2, then UV^2XY^2Z

⇒ aaaabbaabbbbcccc

⇒ $a^4b^2a^2b^5c^4 \notin L$

Ex:- Show that $L = \{ww \mid w \in \{0,1\}^*\}$ is not context free.

Sol:- Assume that L is context free.

- L must have a pumping length (say P)
- Now we take a string S such that $S = 0^P 1^P 0^P 1^P$.
- we divide S into parts UVXYZ

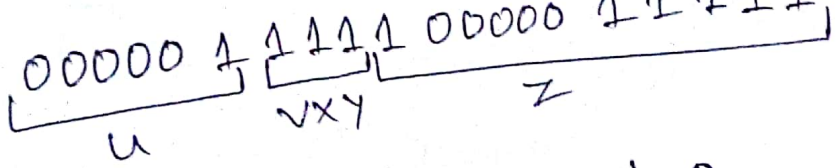
such that, i) UV^iXY^iZ is in A for every $i \geq 0$

ii) $|VY| > 0$

iii) $|VXY| \leq P$

Case 1: pumping length $P = 5$, So, $S = 0^5 1^5 0^5 1^5$.

Case 1: VXY does not straddle a boundary.
(cross the line)



UV^iXY^iZ

for $i = 2$, UV^2XY^2Z

$|VY| > 0$

$|VXY| \leq P$. $\therefore |VXY| \leq 5$

then 000001 11 1 11 1 00000 11111

$= 0^5 1^7 0^5 1^5 \notin L$

Case 2a: vxy straddles the first boundary

$0^p 1^p 0^p 1^p$

U V X Y Z

$uv^i xy^i z$

for $i=2$, uv^2xy^2z

then $000\,0000\,1\,1111\,11\,00000\,11111$

ie $0^7 1^7 0^5 1^5 \notin L$.

Case 2b: vxy straddles the third boundary. $p=5$

$0^p 1^p 0^p 1^p$

U V X Y Z

uv^ixy^iz

for $i=2$ uv^2xy^2z

then $00000\,11111\,00000\,11111$

$0^5 1^5 0^7 1^7 \notin L$.

Case 3: vxy straddles the midpoint, for $p=5$.

$0^p 1^p 0^p 1^p$

U V X Y Z

$|vy| \geq 0$, $|vxy| \leq p$.

uv^ixy^iz etc.

for $i=2$, then uv^2xy^2z.

then $0000011\,1111\,1\,0000\,000\,11111$

ie $0^5 1^7 0^7 0^5 \notin L$

So L is not Context Free.