UNIT 3 : CONTEXT FREE GRAMMAR AND CONTEXT FREE LANGUAGE

* BASIC Elements of grammer :

A grammar is a 4 tuple $G = \{V, T, P, S\}$ where V = finite set of non terminal symbol, T = finite set of terminal symbol, P = set of production rules and S = start symbol symbol.

The P is of the form $A \rightarrow B$ where A and B $\in (VUT)^*$

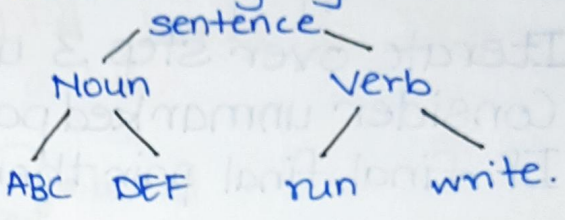
The grammer is normally represented using derivation tree.

The production rules of defining natural language are as follows :

sentence → Nouns, Verbs

Nouns → 'ABC', 'DEF'

Verbs → Run, write.



* FORMAL DEFINITION OF CONTEXT FREE GRAMMAR

The context free grammar can be formally defined as a set denoted by $G = \{V, T, P, S\}$ where V & T are set of non terminals and terminals respectively.

P is a set of production rules, where each production rule is in the form of

Non terminals ⟶ Non terminal

Non terminals ⟶ Terminals

S is a start symbol.

$P = \{ S \rightarrow S+S, S \rightarrow 5*S, S \rightarrow (S), S \rightarrow 4 \}$

If the language is 4+4*4 then we can use the production rules given by P. The start symbol is S The Number of non terminals in the rules P is one and the only non-terminal i.e S

The terminals are +, *, (,), and 4.

We are using following Conventions.

1) The capital letters and used to denote the non terminals.

2) The lower case letters are used to denote the terminals.

SENTENTIAL FORM

Consider $G=(V,T,P,S)$ be a context-free grammer then we can derive a string w from it. This w can be obtained $(V \cup T)^*$ where V denotes the set of non-terminal symbols

The derivation of w from start symbol s can be written as $S \overset{*}{\Rightarrow} w$ which is called as sentential form

If $S \overset{*}{\underset{lm}{\Rightarrow}} w$ then w is a left sentential form

If $S \overset{*}{\underset{rm}{\Rightarrow}} w$ then w is a right sentential form

Example : Consider the grammer $S \to S+S \mid S*S \mid 4$

Then to derive 4+4*4 we use above grammer as

$$S \underset{lm}{\Rightarrow} S*S \underset{lm}{\Rightarrow} S+S*S \underset{lm}{\Rightarrow} 4+S*S \underset{lm}{\Rightarrow} 4+4*S \underset{lm}{\Rightarrow} 4+4*S$$

Here S+S*S is called left sentential form

$$S \Rightarrow S+S \Rightarrow S+S*S \Rightarrow S+S*4 \Rightarrow S+4*4 \Rightarrow 4+4*4$$

Here S+S*S is called right sentential form.

Thus we can obtain desired language L by certain rules

~~Let us~~

Context free Language :

The language of grammer that can be defined by creating the production rules for the given condition

The language generated by context free grammer is called context free language. (CFL)

Let us understand the CFL with the help of various example.

1] Construct the CFG for the language having any number of a's over the set.

As we know RE = a^+

lets build production rules for same

$S \to aS$ rule 1

$S \to \varepsilon$ rule 1

Now if we want string 'aaaa' we can start with start symbols

S

aS

aaS

aaaS

aaaaS

aaaaε = aaaa.

$G = \{\{S\}, \{a\}, \{P\}, S\}$

we can also do r.e = a^* as $S \to \varepsilon$ is possible.

DERIVATION

The production rules are used for deriving certain strings. If $G = (V, T, P, S)$ be some context-free grammar then generation of some language using specific rules is called derivation.

As we know derivation of any string means replacement of non terminal by its appropriate definition. There may be a situation in which there are many non-terminals then which non terminal should be replaced by its definition is sometimes confusing to decide.

Here's hence we apply two methods of deriving.

The left most derivation is a derivation in which the leftmost non terminal is replaced ~~with~~ first to form the sentential form.

The right most derivation is a derivation in which the rightmost non terminal is replaced first to form the sentential form.

Example.

leftmost derivation	rightmost derivation
$S \rightarrow xyx$	$S \rightarrow xyx$
$S \rightarrow ayx$	$S \rightarrow xya$
$S \rightarrow abx$	$S \rightarrow xba$
$S \rightarrow aba.$	$S \rightarrow aba.$

PARSE TREE:

Parse tree is a graphical representation for the derivation of the given production rules for a given CFG. It is simple way to show how the derivation can be done to obtain some strings from given set of production rules. The parse tree is also called derivation tree.

Properties of parse tree.

1) The root node is always node indicating start symbol
2) The derivation is read from left to right.
3) The leaf nodes are always terminal nodes.
4) Interior nodes are always non terminal nodes.

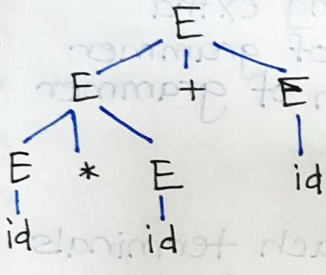
Ambiguous Grammer.

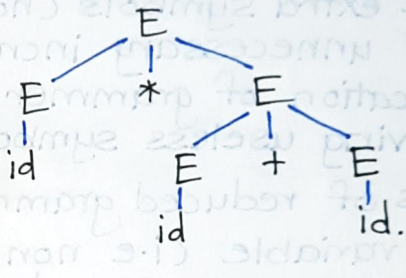
The grammer can be derived in either leftmost derivation or rightmost derivation one can draw a derivation tree called as parse tree or syntax tree based on these derivations. The parse tree has to be unique even if derivation is leftmost or rightmost.

If there exists more than one parse trees for a given grammer, that means there can be more than one leftmost and rightmost derivation possible and then that grammer is said to be ambiguous grammer

example : $G = (V, T, P, S)$ where $V = \{E\}$, $T = \{id\}$,
$P = \{E \to E+E, E \to E*E, E \to id\}$, $S = \{E\}$

Now, if the string is id * id + id then we can draw the two different Parse trees. indicating our id * id + id.

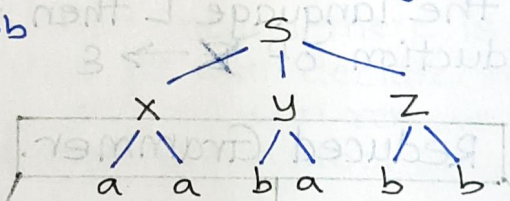


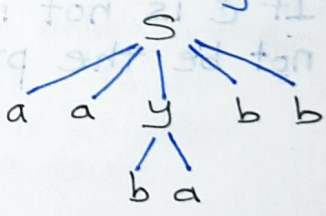


Inherent Ambiguity :

A context free grammer is called inherently ambiguously, if all the production roles in this grammer are ambiguous

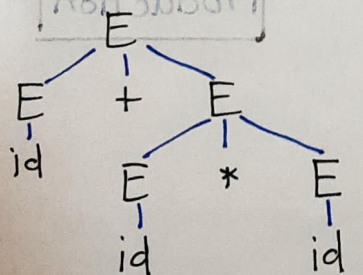
$S \to xyz \mid aaybb$
$x \to aay \mid aa$
$y \to baZ \mid ba$
$Z \to bZb \mid bb$

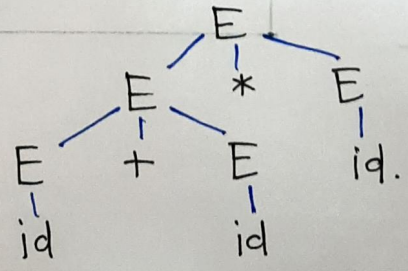




Removal of ambiguity.

Consider the ambiguous grammer as
$E \to E+E \mid E*E \mid id$





for removing the ambiguity we will apply one rule
If the grammer has left associative operator (such as +, −, *, /) then induce the left recursion and if the grammer has right associative operator (exponential operator) then induce right recursion.

$E \rightarrow E+T$	$E \rightarrow E+T \mid T$
$E \rightarrow T$	$T \rightarrow T*F \mid F$
$T \rightarrow T*F$	$F \rightarrow id.$
$T \rightarrow F$	
$F \rightarrow id$	

Simplification of CFG

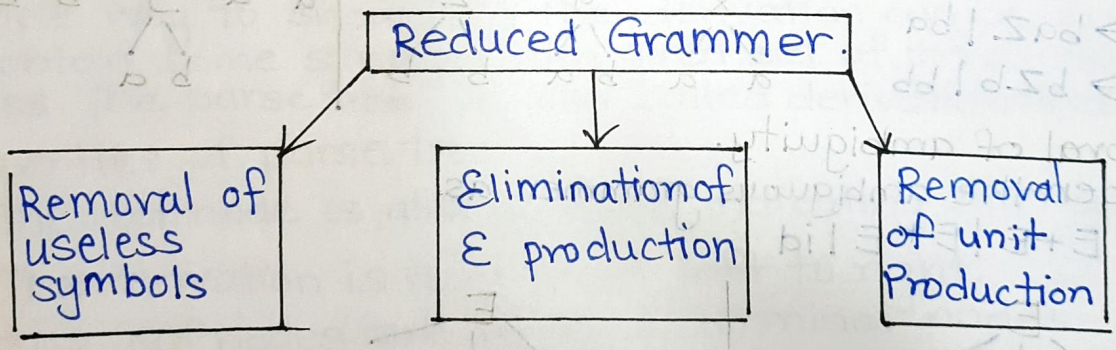
As we have seen various language can effectively be represented by Context free grammer. All the grammers are not always optimized. That means grammer may consist of some extra symbols (non terminals). Having extra symbols unnecessary increase the length of grammer Simplification of grammer means reduction of grammer by removing useless symbols.

Properties of reduced grammer are :

1> Each variable (i.e non terminal) and each terminals of G appears in the derivation of some word in L

2> There should not be only production as $X \rightarrow Y$ where X and Y are non terminals.

3> If ε is not in the language L then there need not be the production of $X \rightarrow \varepsilon$



REMOVAL OF USELESS SYMBOL

Any symbol is useful when it appears when it appears on the right hand side in the production rule and generates some terminal string. If no such derivation exists then it is supposed to be an useless symbol.

A symbol P is useful if there exists some derivation. In the following form.

$S \overset{*}{\Rightarrow} \alpha P \beta \quad \& \quad \alpha P \beta \overset{*}{\Rightarrow} w$

Then P is said to be useful symbol.

where α & β may be some terminal or non terminal symbol and will help us to derive certain string w in combination with P

example :

$G = (V, T, P, S)$ where $V = \{S, T, X\}$, $T = \{0, 1\}$

$S \rightarrow OT \mid 1T1X \mid 0 \mid 1$ rule 1

$T \rightarrow 00$ rule 2

In above CFG, nonterminals are S, T, X

To derive some string we have to start from symbols

S	
OT	$S \rightarrow OT$
000	$T \rightarrow 00$

Thus we can reach to certain string after following these rules.

But if $S \rightarrow X$ then there is no further rule as a defination to X. That means there is no point in the rule $S \rightarrow X$. Hence we can declare that X is a useless symbol. and we can remove this so after removal of this useless symbol CFG becomes.

$G = (V, T, P, S)$ where $V = \{S, T\}$ $T = \{0, 1\}$

and $P \{ S \rightarrow OT \mid 1T \mid 0 \mid$,

$T \rightarrow 00 \}$

S is a start symbol

Elimination of ε Productions from grammer.

As we have seen in finite Automata and regular expression that ε or a null string indicates a string with no value. we have also seen that even through some NFA contains ε moves we can convert that NFA without ε moves. Even in Context Free grammer, If at all there is ε production. we can remove it, without changing the ~~memory~~ meaning of the grammar.

Thus ε productions are not necessary in grammar

$S \to 0S \mid 1S \mid 0 \mid 1$ for $S \to 0S \mid 1S \mid \varepsilon$.

example:

$S \to XYX$, $X \to 0X \mid \varepsilon$, $Y \to 1Y \mid \varepsilon$

$\therefore S \to XYX$

if 1st X in rhs = ε $\therefore S \to XY$

if last X in Lhs = ε $\therefore S \to YX$

if Y = ε $\therefore S \to XX$

if X & Y are ε then $\therefore S \to X$

when both X replaced by ε $\therefore S \to Y$

$S \to XY \mid YX \mid XX \mid Y \mid X$

$\therefore X \to 0X$

if we place ε at rhs of X then

$X \to 0$

$X \to 0X \mid 0$

$Y \to 1Y \mid 1$

collectively we can write it as

$S \to XY \mid YX \mid XX \mid X \mid Y$

$X \to 0X \mid 0$

$Y \to 1Y \mid 1$

Removing Unit Productions

The unit productions are the productions in which one non-terminal gives another non terminals

example: if $X \to Y$, $Y \to Z$, $Z \to X$ then X, Y, Z are unit productions

Example: If $S \to 0A|1B|C$, $A \to 0S|00$, $B \to 1|A$

$C \to 01$

$C \to 01$ is a unit production so write a rule for what C gives.

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

$B \to A$ is also unit production so write a rule for what A gives

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

NORMAL FORMS :

There are two important normal forms

1. Chamsky's Normal form
2. Griebach Normal form

1. Chamsky's Normal form:

It can be defined as :-

Non terminal $\to$ Non terminal . Non terminal
Non terminal $\to$ terminal

The given CFG must be converted into above format then we can say that the grammar is in ~~CFG~~ CNF

Before converting the grammar into CNF it must be in reduced form.

That means remove all the useless symbols, ε-productions and unit productions from it

Thus reduced grammar can be then converted to CNF

Example: Find CNF equivalent of.

$S \to aA\ bB$

$A \to aA|a$

$B \to bB|b$

Step 1: Simplification of grammar

The grammar is already in a simple form without

1) ε-productions
2) Unit productions
3) Useless symbols

Step 2: Every symbol in α, in a production of the form $A \to \alpha$ where $|\alpha| \geq 2$ should be a variable.

This can be done by adding two productions

$Ca \to a$ and $Cb \to b$

The set of productions after the above changes is

$P_2 \to \{ S \to CaA \mid CbB,\ A \to CaA \mid a,\ B \to CbB \mid b,\ Ca \to a,\ Cb \to b \}$

Step 3: Convert to CNF

$S \to CaC_1$
$C_1 \to AC_2$
$C_2 \to CbB$
$A \to CaA \mid a$

$B \to CbB \mid b$
$Ca \to a$
$Cb \to b.$

Corresponding to $S \to CaACbB$

$P_3 \to \{ S \to CaC_1, C_1 \to AC_2, C_2 \to CbB,\ A \to CaA \mid a,\ B \to CbB \mid b,\ Ca \to a,\ Cb \to b \}$

GREIBACH NORMAL FORM

A context free grammar $G = (V, T, P, S)$ is said to be in GNF if every production is of the form

$A \to a\alpha.$

where $a \in T$ is a terminal and α is a string of zero or more variables.

Non Terminal $\to$ one terminal, any number of non terminal.

The Language should be without ε
Right hand side of production should start with a terminal followed by a string of non terminals of length zero or more.

Algorithm for conversion from CFG to GNF

1. Eliminate ε-productions, unit productions and useless symbols from the grammar
2. In production of the form $A \rightarrow x_1 x_2 \dots x_i \dots x_n$ other than x_i, every other symbol should be a variable x_i could be a terminal.
3. Rename variables as $A_1, A_2, A_3 \dots A_n$ to be a production of A
4. Modify the productions to ensure that if there is a production $A_i > A$ then i should be $\leq j$
   If there is a production $A_i > A_j \alpha$ with $i > j$ then we ~~have~~ must generate productions substituting for A_j
5. Repeating step 4 : several times will guarentee that for every production $A_i \rightarrow A_j \alpha$ & $i \leq j$
6. Removing left recursion from every production of the form $A_K \rightarrow A_K \alpha$. B-productions should be added to remove left recursion.
7. Modify A_i - productions to the form $A_i \rightarrow a\alpha$ where a is a terminal and α is a string of non terminals.
8. Modify B_i - productions to the form $B_i \rightarrow a\alpha$ where a is a terminal and α is a string of non terminals.

CONSTRUCT A GRAMMAR IN CFG GNF in which is equivalent to the grammar

$S \rightarrow AA \mid a \qquad A \rightarrow SS \mid b.$

step 1: grammar is already in simple form.

We can proceed to renaming of variables

Variable S and A are renamed as A_1 and A_2 respectively

The set of productions after renaming becomes.

$A_1 \rightarrow A_2A_2 \qquad A_2 \rightarrow A_1A_1$

$A_1 \rightarrow a \qquad A_2 \rightarrow b.$

step 2: Energy production of the form $A_i \rightarrow A_j\alpha$ with $i>j$ must be modified to make $i \leq j$

A_2-production $A_2 \rightarrow A_1A_1$ should modified to make $i<j$ modified.

we must substitute $A_2A_2 \mid a$ for A_1

we shouldnt touch the 2nd A_1 of A_1A_1

$A_2 \rightarrow A_1A_1 \Longrightarrow A_2 \rightarrow A_2A_2A_1$

$A_2 \rightarrow aA_1$

The resulting set of productions is:

$A_1 \rightarrow A_2A_2 \mid a \qquad A_2 \rightarrow A_2A_2A_1 \mid aA_1 \mid b$

Step 3: renaming left recursions.

The A_2 productions $A_2 \rightarrow A_2A_2A_1 \mid aA_1 \mid b$ contains left recursions from A_2 productions can be removed through introduction of B_2-production

$A_2 \rightarrow aA_1B_2 \mid bB_2$

$B_2 \rightarrow A_2A_1B_2 \mid A_2A_1.$

The resulting set of productions is.

$A_1 \rightarrow A_2A_2 \mid a.$

$A_2 \rightarrow aA_1B_2 \mid aB_2 \mid aA_1 \mid b.$

$B_2 \rightarrow A_2A_1B_2 \mid A_2A_1$

Step 4: A_2 productions are in GNF

A_1 & B_2 productions can be converted to GNF with the help of A_2 - productions

$A_2 \rightarrow aA_1B_2 \mid bB_2 \mid aA_1 \mid b$ ... in GNF

$A_1 \rightarrow A_2A_2$

Substitute

$A_1 \rightarrow aA_1B_2A_2 \mid bB_2 \mid aA_1 \mid b$ - ①

$A_1 \rightarrow aA_1B_2A_2 \mid bB_2A_2 \mid aA_1A_2 \mid bA_2$

$A_1 \rightarrow a$ .... in GNF

Now for B_2 productions

$B_2 \rightarrow A_2A_1B_2 \Rightarrow$ substitute ① for 1st A_2

$B_2 \rightarrow aA_1B_2A_1B_2 \mid bB_2A_1B_2 \mid aA_1A_1B_2 \mid bA_1B_2$

$B_2 \rightarrow A_2A_1 \Rightarrow$ substitute ① for 1st A_2

$B_2 \rightarrow aA_1B_2A_1 \mid bB_2A_1 \mid aA_1B_1 \mid bA_1$.

The final set of productions is:

$A \rightarrow aA_1B_2 \mid bB_2 \mid aA_1 \mid b$.

$A_1 \rightarrow aA_1B_2A_2 \mid bB_2A_2 \mid aA_1A_2 \mid bA_2 \mid a$.

$B_2 \rightarrow aA_1B_2A_1B_2 \mid bB_2A_1B_2 \mid aA_1A_1B_2 \mid bA_1B_2 \mid aA_1B_2A_1 \mid bB_2A_1 \mid aA_1A_1 \mid bA_1$

$V = (A_1, A_2, B_2)$

$T = (a, b)$

$S = A_1$

PUMPING LEMMA OF CFG

Break down a CFG into $uvxyz$. $|w| \geq n$
such that $v, y \neq \epsilon$, $|vxy| \leq n$
now pump up x, v & y. $i \geq 0$

$u v^i x y^i z$

check if pumped string is a CFG or not
v & y can be pumped multiple times.

$\underbrace{00}_{u}\underbrace{00}_{v}\underbrace{0}_{x}\underbrace{111}_{y}\underbrace{1}_{z}$.

$|w| = 10$ $\quad n = 10$ $\quad$ CFL: $0^n 1^n$

$\therefore |vxy| = 6. \leq n.$

$u = 00$
$v = 00$
$x = 01$
$y = 11$
$z = 11$

$\therefore$ pumping v & y.

$i = 2.$

$u v^2 x y^2 z.$

00 0000 01 1111 11

new string : 00000001111111. is in CFL.

$\therefore$ 00000111111 is in CFL.

CHOMSKY HIERARCHY

The chomsky hierarchy represents the class of language that ~~the~~ are accepted by different machine. The category of languages in Chomsky Hierarchy is as given below

language class	language	grammar	Machine	example
Type 3	Regular	Regular grammar	FSM i.e NFA to DFA	a^*b^*
Type 2	context free	context free	PDA	a^nb^n
Type 1	Decidable language	Context sensitive language	linear bounded automata.	$a^nb^nc^n$
Type 0	Computable language	unrestricted grammar	Turing machine	n!

Applications of CFG:

1) parsing technique
2) Markup Language
3) XML and Document type definitions.