

Unit - 4Context Free Grammar (CFG):

Grammars are used to generate the words of a language and to determine whether a word is in a language. Context free grammars are used to define syntax of almost all programming languages, in context of computer science. Thus an important application of context free grammars occurs in the specification and compilation of programming languages.

Formal Definition of Context Free Grammar:

Context free Grammar is defined by 4 tuples as  $G_1 = \{V, T, S, P\}$  where,

- $V$  = Set of variables or Non-terminal Symbols.
- $T$  = Set of Terminal Symbols
- $S$  = Start Symbol
- $P$  = Production Rule.

Context Free Grammar has production rule of the form  $A \rightarrow \alpha$ , where,  $\alpha = \{V \cup T\}^*$  and  $A \in V$ .

closure of  $V$  union  $T$

belongs to

Components of CFG:

There are four components in  $CFG_1$ .

- 1). There is a finite set of symbols that form the strings of language being defined. We call this alphabet the terminal symbol. In above tuples it is represented by  $T$ .
- 2). There is a finite set of variables also called non-terminal symbols. Each variable represents the language, i.e., a set of strings. It is represented by  $V$  in above tuples.
- 3). One of the variables represent the language being defined. It is called the start symbol. It is represented by  $S$  in above tuples.
- 4). There is a finite set of productions or rules that represent the recursive definition of a language. Each production consists of:

a) A variable that is being defined by the production. This is called head of production.

b) The production symbol →

c) A string of zero or more terminals and variables. This string is called the body of the production, represents one way to form the string in the language of the head.

Note: The variable symbols are often represented by capital letters.

The terminals are analogous to the input alphabet and are often represented by lower case letters. One of the variables is designed as a start variable. It usually occurs on the left hand side of the topmost rule.

Example:

$$S \rightarrow E$$

$$S \rightarrow 0S1$$

This is a CFG defining the grammar of all the strings with equal no. of 0's followed by equal no of 1's.

Here,

the two rules define the production P,

E, 0, 1 are the terminals defining T,

S is a variable symbol defining V,

And S is start symbol from where production starts.

CFG vs RE:

The CFG are more powerful than the regular expressions as they have more expressive power than the regular expression. Generally regular expressions are useful for describing the structure of lexical constructs as identical keywords, constants etc. But they do not have the capability to specify the recursive structure of the programming constructs. However, the CFG can define any of the recursive structure also. Thus, CFG can define the languages that are regular as well as those languages that are not regular.

## Use of CFG:

Context-free grammars are used in compilers and in particular for parsing, taking a string-based program and figuring out what it means. Typically, CFGs are used to define the high-level structure of a programming language. Figuring out how a particular string was derived tells us about its structure and meaning.

Meaning of context free: Consider an example:

$$\begin{aligned} S &\rightarrow aMb \\ M &\rightarrow A \mid B \\ A &\rightarrow \epsilon \mid aA \\ B &\rightarrow \epsilon \mid bB. \end{aligned}$$

Now, Consider a string aaAb, which is an intermediate stage in the generating of aaab. It is natural to call the strings "aa" and "b" that surround the symbol A, the "context" of A in this particular string. Now, the rule  $A \rightarrow aA$  says that we can replace A by the string aA no matter what the surrounding strings are; in other words, independently of the context of A.

When there is a production of form  $l w_1 r \rightarrow l w_2 r$  (but not of the form  $w_1 \rightarrow w_2$ ), the grammar is context sensitive since  $w_1$  can be replaced by  $w_2$  only when it is surrounded by the strings " $l$ " and " $r$ ".

## Context free language (CFL):

Context free language (CFL) is a language which is generated by a context-free grammar or type-2 grammar and gets accepted by a Pushdown Automata. The set of all context-free language is identical to the set of languages accepted by pushdown automata, and the set of regular languages is a subset of context-free languages. An inputed language is accepted by a computational model if it runs through the model and ends in an accepting final state. Context-free languages and context-free grammars have applications in computer science and linguistics such as natural language processing and computer language design.

## ④ Derivation using Grammar Rule:

The process of producing strings using the production rules of the grammar is called derivation. There are two possible approaches of derivation:

### 1) Body to head (Bottom Up) approach:-

Here, we take strings known to be in the language of each of the variables of the body, concatenate them, in the proper order, with any terminals appearing in the body, and infer that the resulting string is the language of the variables in the head.

Consider grammar,

$$\begin{aligned} S &\rightarrow S+S \\ S &\rightarrow S/S \\ S &\rightarrow (S) \\ S &\rightarrow S-S \\ S &\rightarrow S^*S \\ S &\rightarrow a \end{aligned}$$

Here given  $a + (a^*a)/a - a$ .

or this can also be written as:  
 $S \rightarrow S+S | S/S | (S) | S-S | S^*S | a$

in single line

..... Grammar(2)

nothing just  
named the  
grammar

first body solve it  
result String 1

Now, by this approach we start with any terminal appearing in the body and use the available rules from body to head.

S.N	String Inferred	Variable	Production	String Used.
1.	<u>a</u>	<u>S</u>	$S \rightarrow a$	a; String 1
2.	$a^*a$	<u>S</u>	$S \rightarrow S^*S$	$a^*a$ ; String 2
3.	$(a^*a)$	<u>S</u>	$S \rightarrow (S)$	String 1 & 2; String 3
4.	$(a^*a)/a$	<u>S</u>	$S \rightarrow S/S$	String 1 & 3; String 4
5.	$a + (a^*a)/a$	<u>S</u>	$S \rightarrow S+S$	String 1 & 4; String 5
6.	$a + (a^*a)/a - a$	<u>S</u>	$S \rightarrow S-S$	String 1 & 5; String 6

We write string here based on Production rule taken.

This may not apply for all so follow sequence, Body-left-Right

Looking at question we start with terminal (i.e., here for this small letter 'a'). Then we proceed as BODMAS rule as in math. First we complete bracket, Then divide, then multiply, then add, finally sub. Here, multiply is done earlier because bracket must be completed first OR Body-left(tail) OR Body-right(head)

i.e., String used ; New String named

for e.g. String 4 is formed by string 1 & 3 as follows;

$(a^*a)/a$   
String 3      String 1

## 2) Head to body (Top Down) approach:

Here, we use production from head to body. We expand the start symbol using a production, whose head is the start symbol. Here, we expand the resulting string until all strings of terminal are obtained. Here, we have two approaches:

- (LMD) a) leftmost Derivation: Here leftmost symbol (variable) is replaced first.
- (RMD) b) Rightmost Derivation: Here rightmost symbol is replaced first.

For Example:- Consider the previous example of deriving string

$a + (a * a) / a - a$ . with the grammar (2). [i.e,  $S \rightarrow S+S/S/S/(S)/S-S$ ]  
 $S^*S/a$ ]

# Now the leftmost derivation for the given string is;

- $S \rightarrow S+S$  (Rule  $S \rightarrow S+S$ )
- $S \rightarrow a+S$  (Rule  $S \rightarrow a$ )
- $S \rightarrow a+S-S$  (Rule  $S \rightarrow S-S$ )
- $S \rightarrow a+S/S-S$  (Rule  $S \rightarrow S/S$ )
- $S \rightarrow a+(S)/S-S$  (Rule  $S \rightarrow (S)$ )
- $S \rightarrow a+(S*S)/S-S$  (Rule  $S \rightarrow S*S$ )
- $S \rightarrow a+(a*S)/S-S$  (Rule  $S \rightarrow a$ )
- $S \rightarrow a+(a*a)/S-S$  (Rule  $S \rightarrow a$ )
- $S \rightarrow a+(a*a)/a-S$  (Rule  $S \rightarrow a$ )
- $S \rightarrow a+(a*a)/a-a$  (Rule  $S \rightarrow a$ )

start from left and Replace from left so we start with left part towards body

# Now the rightmost derivation for the given string is;

- $S \rightarrow S-S$  (Rule  $S \rightarrow S-S$ )
- $S \rightarrow S-a$  (Rule  $S \rightarrow a$ )
- $S \rightarrow S+S-a$  (Rule  $S \rightarrow S+S$ )
- $S \rightarrow S+S/S-a$  (Rule  $S \rightarrow S/S$ )
- $S \rightarrow S+S/a-a$  (Rule  $S \rightarrow a$ )
- $S \rightarrow S+(S)/a-a$  (Rule  $S \rightarrow (S)$ )
- $S \rightarrow S+(S*S)/a-a$  (Rule  $S \rightarrow S*S$ )
- $S \rightarrow S+(S*a)/a-a$  (Rule  $S \rightarrow a$ )
- $S \rightarrow S+(a*a)/a-a$  (Rule  $S \rightarrow a$ )
- $S \rightarrow a+(a*a)/a-a$  (Rule  $S \rightarrow a$ )

Start from right and more towards body and replace from right

Q. Consider a Grammar;

$$S \rightarrow aAS | a$$
$$A \rightarrow SbA | SS | ba$$

Given a string aaabaaa, give leftmost and rightmost derivation.

Solution:

leftmost Derivation: (LMD or lm)

$$\begin{aligned} S &\rightarrow aAS \\ S &\rightarrow aSSS \quad (\text{Rule } A \rightarrow SS) \\ S &\rightarrow aASS \quad (\text{Rule } S \rightarrow a) \\ S &\rightarrow aaaASS \quad (\text{Rule } S \rightarrow aAS) \\ S &\rightarrow aaabass \quad (\text{Rule } S \rightarrow ba) \\ S &\rightarrow aaabaas \quad (\text{Rule } S \rightarrow a) \\ S &\rightarrow aaabaaa \quad (\text{Rule } S \rightarrow a) \end{aligned}$$

Rightmost Derivation: (RMD or rm)

$$\begin{aligned} S &\rightarrow aAS \\ S &\rightarrow aAa \quad (\text{Rule } S \rightarrow a) \\ S &\rightarrow assa \quad (\text{Rule } A \rightarrow SS) \\ S &\rightarrow aSaASA \quad (\text{Rule } S \rightarrow aAS) \\ S &\rightarrow aSaAaa \quad (\text{Rule } S \rightarrow a) \\ S &\rightarrow aSabaaa \quad (\text{Rule } A \rightarrow ba) \\ S &\rightarrow aaabaaa \quad (\text{Rule } S \rightarrow a) \end{aligned}$$

This | symbol is on.  
i.e.,  $S \rightarrow aAS$  }  $S$  can be  
or,  $S \rightarrow a$  } replaced by  
any of these  
If  $E$  symbol comes then it is  
empty symbol.

④ Sentential Form :-

Derivations from the start symbol produce strings that have a special role. We call these "sentential forms". i.e., if  $G_1 = (V, T, P, S)$  is a CFG<sub>1</sub>, then any string  $\alpha$  is in  $(VUT)^*$  such that  $S \rightarrow^* \alpha$  is a sentential form. If  $S \rightarrow_{lm}^* \alpha$ , then  $\alpha$  is a left sentential form, and if  $S \rightarrow_{rm}^* \alpha$ , then  $\alpha$  is a right sentential form.

⑤ Language of Grammar (Context Free Grammar) :-

Let  $G_1 = (V, T, P, S)$  is a context free grammar. Then the language of  $G_1$  denoted by  $L(G_1)$  is the set of terminal strings that have derivation from the start symbol in  $G_1$ .

$$\text{i.e., } L(G_1) = \{x \in T^* \mid S \rightarrow^* x\}$$

The language generated by a CFG<sub>1</sub> is called the Context free language (CFL).

## Parse Tree / Derivation Tree:

Parse tree is a representation of strings of terminals using the productions defined by the grammar. A parse tree, pictorially shows how the start symbol of a grammar derives a string in the language. Parse tree may be viewed as a graphical representation for a derivation that filters out the choice regarding the replacement order.

Formally, given a Context Free Grammar  $G_1 = (V, T, P, S)$ , a parse tree is a n-ary tree having the following properties;

→ The root is labeled by the start symbol.

→ Each interior node of parse tree are variables.

→ Each leaf node of parse is labeled by a terminal symbol or  $\epsilon$ .

→ If an interior node is labeled with a non terminal A and its childrens are  $x_1, x_2, \dots, x_n$  from left to right then there is a production  $P$  as;

$$A \rightarrow x_1, x_2, \dots, x_n \text{ for each } x_i \in T.$$

Example: Consider the grammar  $S \rightarrow aSa | a | b | \epsilon$ .  
Now, for string  $S \rightarrow^* aabaa$ .

We have,

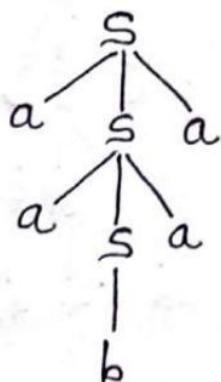
$$S \rightarrow aSa$$

$$S \rightarrow aasaa \text{ (Rule } S \rightarrow aSa\text{)}$$

$$S \rightarrow aabaa \text{ (Rule } S \rightarrow b\text{).}$$

each  $x_i$  belonging to T

So, the parse tree is → :



Q. Construct the parse tree for  $a^*(a+b00)$ , Considering  $S \rightarrow I \mid S + S$   
 $S \star S \mid (S)$ . And  $I \rightarrow a \mid b \mid Ia \mid Ib \mid IO \mid I1$ .

Solution:- The parse tree is;

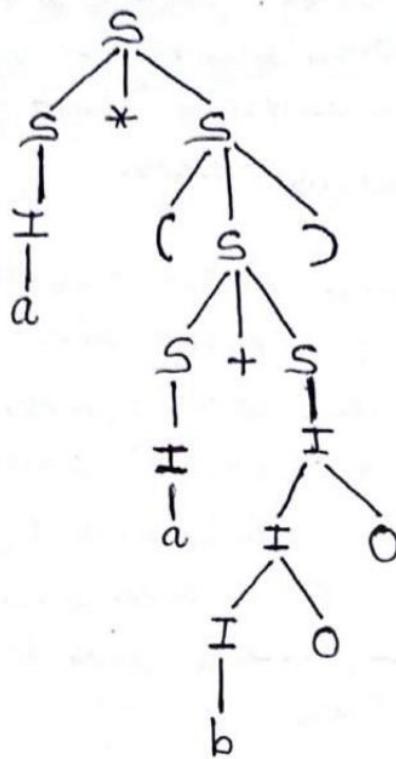


Fig. Parse tree for  $a^*(a+b00)$ .

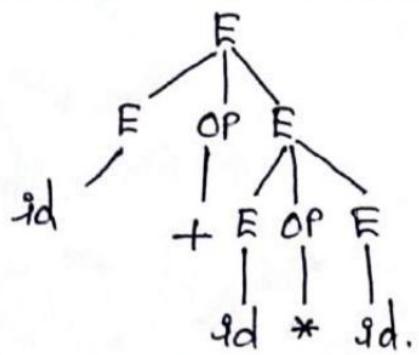
Q. Construct a grammar defining arithmetic expression and generate parse tree for  $id + id * id$  and  $(id + id)^* (id + id)$ .

$$E \rightarrow EOP E \mid (E) \mid id$$

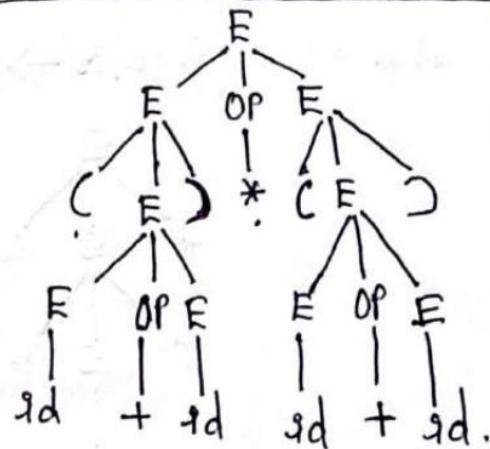
$$OP \rightarrow + \mid - \mid * \mid / \mid \uparrow$$

Solution:-

Parse tree for  $id + id * id$



Parse tree for  $(id + id)^* (id + id)$



## \* Ambiguity in Grammar:-

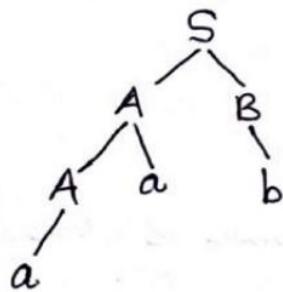
A Grammar  $G = (V, T, P, S)$  is said to be ambiguous if there is a string  $w \in L(G)$  for which we can derive two or more distinct derivation tree rooted at  $S$  and yielding  $w$ . In other words, a grammar is ambiguous if it can produce more than one leftmost or more than one rightmost derivation for the same string in the language of the grammar.

Example:  $S \rightarrow AB | aAb$   
 $A \rightarrow a | Aa$   
 $B \rightarrow b$

For any string  $aab$ ; we have two leftmost derivations as;

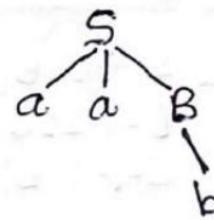
$S \rightarrow AB$   
 $S \rightarrow AaB$  (Rule  $A \rightarrow Aa$ )  
 $S \rightarrow aAB$  (Rule  $A \rightarrow a$ )  
 $S \rightarrow aab$  (Rule  $B \rightarrow b$ ).

The parse tree for this derivation is;



Also  
 $S \rightarrow aAB$   
 $S \rightarrow aab$  (Rule  $B \rightarrow b$ ).

The parse tree for this derivation is;



## \* Inherent Ambiguity:-

A context free language  $L$  is said to be inherently ambiguous if all its grammars are ambiguous.

E.g.  $L = \{0^i 1^j 2^k \mid i=j \text{ or } j=k\}$ .

## \* Regular Grammar:-

A regular grammar represents a language which is represented by regular expressions. The regular grammar is accepted by NFA and DFA and the language of grammar is called regular language. A regular grammar is a subset of CFG. The regular grammar may be either left or right linear.

## i) Right Linear Regular Grammar:

A regular grammar in which all of the productions are of the form  $A \rightarrow wB$  or  $A \rightarrow w$  where  $A, B \in V$  and  $w \in T$  is called right linear.

For Example:  $S \rightarrow 00B | 11C$

$\omega$ , i.e., any string

$B \rightarrow 11C | S | 1$

$C \rightarrow 00B | 00$

## ii) Left Linear Regular Grammar:

A grammar in which all of the productions are of the form  $A \rightarrow Bw$  or  $A \rightarrow w$ , where  $A, B \in V$  and  $w \in T$  is called left linear.

For Example:  $S \rightarrow B00 | C11$

$B \rightarrow C11 | S | 1$

$C \rightarrow B00 | 00$

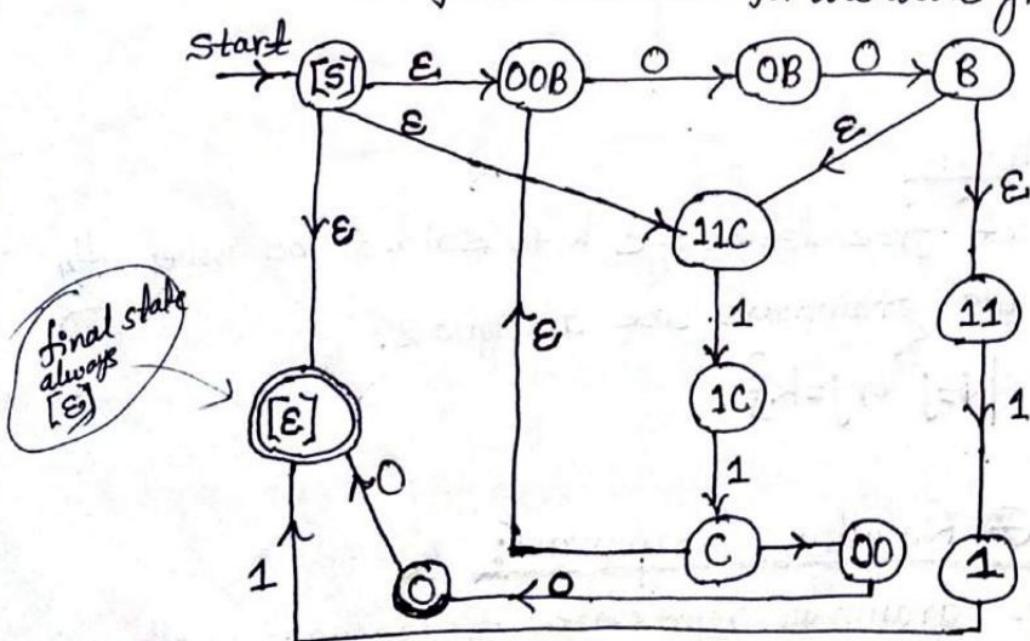
## ✳ Equivalence of Regular Grammar and Finite Automata:

Example 1:  $S \rightarrow 00B | 11C | \epsilon$

$B \rightarrow 11C | 11$

$C \rightarrow 00B | 00$

Solution: The finite automaton for the above grammar is given as;



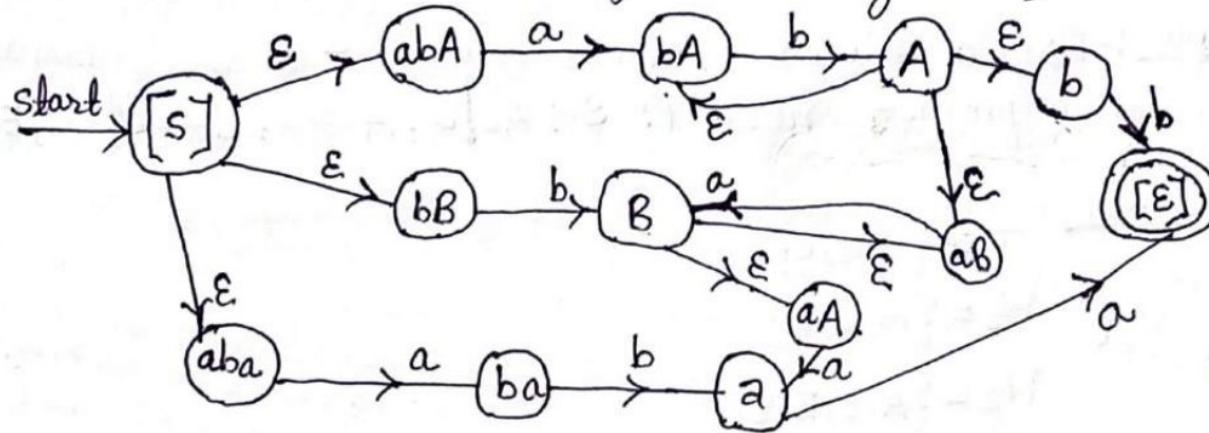
### Process

First we started with our start variable  $[S]$ . Which goes to  $00B$  and  $11C$  and  $[E]$ . On getting empty value  $\epsilon$ ,  $[E]$  is a final state. Now  $00B$  and  $11C$  are extended until when single  $B$  or  $C$  remains. Now,  $B$  goes to  $11C$  and  $11$  on  $\epsilon$ .  $C$  goes to  $00B$  and  $00$  on  $\epsilon$ . ~~Now~~,  $00B$  &  $11C$  are already extended so remaining  $11$  &  $00$  are extended and finally sent to final state.

Example 2:-  $S \rightarrow abA \mid bB \mid aba$   
 $A \rightarrow b \mid aB \mid bA$   
 $B \rightarrow aB \mid aA$

Solution:

Here,  $\epsilon$  does not appear on the body part of any productions, but we introduce state  $[\epsilon]$  and select it as accepting state. Thus the finite automaton for the above grammar is given as;



## \* Simplification of CFG<sub>1</sub>:

In CFG<sub>1</sub> sometimes all the production rules and symbols are not needed for the derivation of strings. Besides this, there may also be some NULL Productions and UNIT Productions. Elimination of these productions and symbols is called Simplification of CFG<sub>1</sub>. Simplification consists of: i) Reduction of CFG<sub>1</sub> ii) Removal of Unit Productions iii) Removal of Null Productions.

i) Reduction of CFG<sub>1</sub>: CFG<sub>1</sub> are reduced in two phases:

Phase 1: Derivation of an equivalent grammar G<sub>1'</sub> from the CFG<sub>1</sub>, G<sub>1</sub>, such that each variable derives some terminal string.

Derivation Procedure:

Step 1: Include all Symbols  $w_i$ , that derives some terminal and initialize  $i=1$ .

Step 2: Include symbols  $w_{i+1}$ , that derives  $w_i$ .

Step 3: Increment  $i$  and repeat step 2 until  $w_{i+1} = w_i$ .

Step 4: Include all production rules that have  $w_i$  in it.

① This is also called elimination of useless symbols.

No need to remember  
it will be clear  
on solving example  
and comparing e.g.  
with procedure  
Edu Neso Academy

Phase 2: Derivation of an equivalent grammar G<sub>1''</sub>, from the GFG<sub>1</sub>, G<sub>1'</sub>, such that each symbol appears in a sentential form.

### Derivation Procedure:

Step 1: Include the start symbol in  $Y_1$  and initialize  $i=1$ .

Step 2: Include all symbols  $Y_{i+1}$ , that can be derived from  $Y_i$ .

Step 3: Increment  $i$  and repeat Step 2 until  $Y_{i+1} = Y_i$ .

Step 4: Include all production rules that have  $Y_i$  in it.

Similar to Phase 1  
only step 2 differ  
in  $Y_2$  used instead  
of  $W_2$

Example :- Find a reduced grammar equivalent to the grammar

$G_1$ , having production rules  $P: S \rightarrow AC | B, A \rightarrow a, C \rightarrow c | BC, E \rightarrow aA | e$ .

Solution:

Phase 1:

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

Set of terminal symbols

all symbols  
that can derive  
terminal symbol

$$W_1 = \{A, C, E\}$$

all symbols that can  
derive symbols that are in  $W_1$ .  
Here  $S$  can derive  $AC$  which is  
also present in  $W_1$ . So,  $S$  is  
included.

$$W_2 = \{A, C, E, S\}$$

Now,  $W_3 = W_2$   
i.e.,  $W_{i+1} = W_i$   
So, we stop here

Now,

$$G'_1 = \{(A, C, E, S), \{a, c, e\}, P, (S)\}$$

All symbols that can be derive  
symbols that are in  $W_2$ .

recent non-terminal  
symbols that  
we get from  $W_3$ :

Since  $G_1 = \{V, T, S, P\}$

Now, Production Rule can be  
defined as;

$$P: S \rightarrow AC, A \rightarrow a, C \rightarrow c, E \rightarrow aA | e$$

Terminal symbols  
that can be derived from  
(A, C, E, S)

Start symbol

First look at  $P$ : given in  
question now copy it  
eliminating symbols that  
are not in  $G'_1$ .

Phase 2:

$$Y_1 = \{S\}$$

only start symbol  
included

$$Y_2 = \{S, A, C\}$$

All symbols that  $Y_1$  &  $e, S$   
can derive.

$$Y_3 = \{S, A, C, a, c\}$$

All symbols that  $Y_2$  can derive

$$Y_4 = \{S, A, C, a, c\}$$

Since  $Y_{i+1} = Y_i$  we stop here

Now,

$$G'_1 = \{(A, C, S), \{a, c\}, P, \{S\}\}$$

$$P: S \rightarrow AC, A \rightarrow a, C \rightarrow c$$

First look at  $P$  in phase 1  
then copy by reducing.  
No  $E$  symbol in above  
 $G'_1$  so reduced in  $P$   
also

Since, this  $P$  in phase 2 has lesser symbols than that of  $P$   
in question. So we have reduced the given grammar.

## ii) Removal of Unit Productions:

Any production rule of the form  $A \rightarrow B$  where  $A, B \in \text{Non Terminals}$  is called Unit Production.

### Procedure:

Step 1: To remove  $A \rightarrow B$ , add production  $A \rightarrow x$  to the grammar rule whenever  $B \rightarrow x$  occurs in the grammar. [ $x \in \text{Terminal}$ ,  $x$  can be Null].

Step 2: Delete  $A \rightarrow B$  from the grammar.

Step 3: Repeat from Step 1 until all Unit Productions are removed.

Example: Remove Unit Productions from the Grammar whose production rule is given by  $P: S \rightarrow XY, X \rightarrow a, Y \rightarrow Z|b, Z \rightarrow M, M \rightarrow N, N \rightarrow a$ .

### Solution:

Here,  $Y \rightarrow Z, Z \rightarrow M, M \rightarrow N$  are the Unit Productions given by  $P$  in the given question. Now we have to remove them as follows:

#### For $M \rightarrow N$

Since,  $N \rightarrow a$ , we add  $M \rightarrow a$

$P: S \rightarrow XY, X \rightarrow a, Y \rightarrow Z|b, Z \rightarrow M, M \rightarrow a, N \rightarrow a$

#### For $Z \rightarrow M$

Since  $M \rightarrow a$ , we add  $Z \rightarrow a$

$P: S \rightarrow XY, X \rightarrow a, Y \rightarrow Z|b, Z \rightarrow a, M \rightarrow a, N \rightarrow a$ .

#### For $Y \rightarrow Z$

Since  $Z \rightarrow a$ , we add  $Y \rightarrow a$ .

$P: S \rightarrow XY, X \rightarrow a, Y \rightarrow a|b, Z \rightarrow a, M \rightarrow a, N \rightarrow a$ .

Here,  $Y \rightarrow Z|b$   
since  $Z \rightarrow a$   
so,  $Y \rightarrow a|b$   
this  $b$  remains  
as it is

Hence, we removed all Unit Productions. But here if we look at  $P$  above the symbols  $Z, M$  and  $N$  are unreachable symbols since, all these are giving  $a$ . Here,  $S$  is giving  $XY$ ,  $X$  is giving  $a$ , where  $a$  is terminal symbol, also  $Y$  gives  $a|b$  which are also terminal symbols. There is no way from start symbol  $S$  through which we can reach  $Z, M$  and  $N$ . Hence, we have to remove the unreachable symbols and our production rule finally becomes as;

$P: S \rightarrow XY, X \rightarrow a, Y \rightarrow a|b$ .

## 11) Removal of Null Productions:

In a CFG, a Non-Terminal Symbol 'A' is a nullable variable if there is a production  $A \rightarrow \epsilon$  or there is a derivation that starts at 'A' and leads to  $\epsilon$ . (like  $A \rightarrow \dots \rightarrow \epsilon$ ).

### Procedure:

Step 1: To remove  $A \rightarrow \epsilon$ , look for all productions whose right side contains A.

Step 2: Replace each occurrences of 'A' in each of these productions with  $\epsilon$ .

Step 3: Add the resultant productions to the Grammar.

Example: Remove Null Productions from the following Grammar.

$$S \rightarrow ABAC, A \rightarrow aA|\epsilon, B \rightarrow bB|\epsilon, C \rightarrow C$$

### Solution:

Here, in the given question  $A \rightarrow \epsilon$  and  $B \rightarrow \epsilon$  are two null productions. So we remove them as follows;

To eliminate  $A \rightarrow \epsilon$

$$S \rightarrow ABAC$$

$$S \rightarrow ABC|BAC|BC$$

According to Step 1.

Different cases on replacing different A by  $\epsilon$  (i.e., empty).  
 Replacing 1st A with  $\epsilon = BAC$   
 $\Rightarrow$  2nd A with  $\epsilon = ABC$   
 replacing both = BC

Also,

$$A \rightarrow aA$$

$$A \rightarrow a$$

on replacing A by  $\epsilon$  (i.e., empty)

Since according to Step 1  $A \rightarrow aA|\epsilon$  also contains A on right side so this is also replaced by  $\epsilon$

New production:  $S \rightarrow ABAC|ABC|BAC|BC$

$$A \rightarrow aA/a$$

$$B \rightarrow bB/\epsilon$$

$$C \rightarrow C$$

we have new a so ~~changed by  $\epsilon$~~  no longer remaining as eliminated by a

Others remains same

To eliminate  $B \rightarrow \epsilon$

$$S \rightarrow ABAC|ABC|BAC|BC$$

$$S \rightarrow AAC|AC|C$$

for this we look at new production above

since it contains B on right side as step 1

ABAC if B is  $\epsilon = AAC$ , for ABC if B is  $\epsilon$  then AC, for BAC if B is  $\epsilon$  then AC but AC already there so no need to write. Similarly BC, B on getting  $\epsilon$  gave C.

Also,

$$B \rightarrow bB$$

$$B \rightarrow b$$

bB, B on getting  $\epsilon$

New Production:  $S \rightarrow ABAC|ABC|BAC|BC|AAC|AC|C$

$$A \rightarrow aA/a$$

$$B \rightarrow bB/b$$

$$C \rightarrow C$$

Hence, all Null productions are removed.

## Chomsky Normal Form (CNF):-

In Chomsky Normal Form (CNF) we have a restriction on the length of RHS, which is: elements in RHS should either be two variables or a Terminal. A CFG is in Chomsky Normal Form if the productions are in the following forms:

$$A \rightarrow a$$

$$A \rightarrow BC$$

where, A, B and C are non-terminals  
& a is a terminal.

### Steps to convert a given CFG to Chomsky Normal Form:-

Step 1: If the start symbol S occurs on some right side, create a new start symbol  $S'$  and a new production  $S' \rightarrow S$ .

Step 2: Remove Null Productions. (By Method that we discussed earlier).

Step 3: Remove Unit Productions. (By Method that we discussed earlier).

Step 4: Replace each Production  $A \rightarrow B_1 \dots B_n$  where  $n > 2$ , with  $A \rightarrow B_1 C$  where  $C \rightarrow B_2 \dots B_n$ . Repeat this step for all productions having two or more symbols on the right side.

Step 5: If the right side of any production is in the form  $A \rightarrow aB$  where 'a' is a terminal and A and B are non-terminals, then the production is replaced by  $A \rightarrow XB$  and  $X \rightarrow a$ . Repeat this step for every production which is of the form  $A \rightarrow aB$ .

Example: Convert the following CFG to CNF:

$$P: S \rightarrow ASA | aB, A \rightarrow B|S, B \rightarrow b|\epsilon.$$

Solution:

Step 1: Since S appears in RHS, we add a new state  $S'$  and  $S' \rightarrow S$  is added to the production.

$$P: S' \rightarrow S, S \rightarrow ASA | aB, A \rightarrow B|S, B \rightarrow b|\epsilon.$$

Step 2: Remove the Null Productions:  $B \rightarrow \epsilon$  and  $A \rightarrow \epsilon$ .

After Removing  $B \rightarrow \epsilon$ :  $P: S' \rightarrow S, S \rightarrow ASA | aB | a, A \rightarrow B | S | \epsilon, B \rightarrow b$

After Removing  $A \rightarrow \epsilon$ :  $P: S' \rightarrow S, S \rightarrow ASA | aB | a | AS | SA | S, A \rightarrow B | S, B \rightarrow b$

Step 3: Remove the Unit Productions:  $S \rightarrow S$ ,  $S' \rightarrow S$ ,  $A \rightarrow B$  and  $A \rightarrow S$ :

After Removing  $S \rightarrow S$ :  $P: S' \rightarrow S, S \rightarrow ASA | aB | a | AS | SA, A \rightarrow B | S, B \rightarrow b$

After Removing  $S' \rightarrow S$ :  $P: S' \rightarrow ASA | aB | a | AS | SA, S \rightarrow ASA | aB | a | AS | SA, A \rightarrow b | S, B \rightarrow b$

Here S is not removed directly from  $S'$ , instead replaced by value of S becoz LHS  $\neq$  RHS

Here S is directly removed because it is  $S \rightarrow S$ , LHS and RHS same so.

After Removing A  $\rightarrow$  B:

$P: S' \rightarrow ASA|aB|a|AS|SA, S \rightarrow ASA|aB|a|AS|SA,$   
 $A \rightarrow b|S, B \rightarrow b.$

STEPS B can't replace AS

After Removing A  $\rightarrow$  S:

$P: S' \rightarrow ASA|aB|a|AS|SA, S \rightarrow ASA|aB|a|AS|SA,$   
 $A \rightarrow b|ASA|aB|a|AS|SA, B \rightarrow b.$

Step 4: Now find out the productions that has more than two variables in RHS.

$S' \rightarrow ASA, S \rightarrow ASA$  and  $A \rightarrow ASA.$

After removing these, we get:

$P: S' \rightarrow AX|aB|a|AS|SA, S \rightarrow AX|aB|a|AS|SA,$   
 $A \rightarrow b|AX|aB|a|AS|SA, B \rightarrow b, X \rightarrow SA.$

Step 5: Now change the productions:  $S' \rightarrow aB, S \rightarrow aB$  and  $A \rightarrow aB$

Finally we get:  $P: S' \rightarrow AX|YB|a|AS|SA,$   
 $S \rightarrow AX|YB|a|AS|SA,$   
 $A \rightarrow b|AX|YB|a|AS|SA,$   
 $B \rightarrow b,$   
 $X \rightarrow SA,$   
 $Y \rightarrow a.$

Since X was already used  
so we took next variable  
Y

### \* Greibach Normal Form (GNF):-

A CFG is in Greibach Normal Form if the productions are in following forms:

$A \rightarrow b$   
 $A \rightarrow bC_1C_2 \dots C_n.$

where  $A, C_1, \dots, C_n$  are Non-Terminals  
and  $b$  is a 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 (using Unit & Null Productions removal methods discussed before).

Step 2: Check whether the CFG is already in Chomsky Normal Form (CNF) and convert it to CNF if it is not.

Step3: Change the names of the Non-Terminal Symbols into some As in ascending order of i.

Step4: Alter the rules so that the Non-Terminals are in ascending order, such that, If the production is of the form  $A_j \rightarrow A_i x$ , then,  $i < j$  and should never be  $i \geq j$ .

Example: Convert the given CFG to GNF:

$$S \rightarrow CA | BB, B \rightarrow b | SB, C \rightarrow b, A \rightarrow a$$

Solution:

Last step 4 is hard to understand better to study with video by neso academy

Step1: Since there are no any Unit Productions or Null Productions in the given question, so we are done with step 1.

Step2: The given CFG is already in Chomsky Normal Form so, we are also done with step 2.

Step3: Here we have S, C, A and B as our Non-Terminal Symbols. Now, we have to change these as follows:

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

But remember we should rename these in ascending order as the production given in question. Order should not be changed.

Now, we get equivalent production as;

$$A_1 \rightarrow A_2 A_3 | A_4 A_4$$

$$A_4 \rightarrow b | A_1 A_4$$

$$A_2 \rightarrow b$$

$$A_3 \rightarrow a$$

Step4:

$$A_1 \rightarrow A_2 A_3 | A_4 A_4$$

$$A_4 \rightarrow b | A_2 A_3 A_4 | A_4 A_4 A_4$$

$$A_4 \rightarrow b | b A_3 A_4 | A_4 A_4 A_4$$

If it is of form  $b C_1, \dots, C_n$  then it is in GNF so, now this part is okay.

Let me take  $A_1 \rightarrow A_2 A_3$  we have form  $A_j \rightarrow A_i x$ . Now check if  $i < j$ . Here  $1 < 2$ , so no need to alter. but if  $i \geq j$  then we replace  $A_j$  with value of  $A_i$  in any other part of production

$A_1$  replaced by  $A_2 A_3 | A_4 A_4$  and others copied as they are but this again not following the rule.

If  $i = j$  then it is left recursion, so it must be removed by rules as ..

Since we encounter  $i = j$ , so, now

we remove it by left recursion by introducing a new variable. let new variable be Z. Now we make new production for Z by taking variables following the problematic one and once write it with new variable and once without it.

$Z \rightarrow A_4 A_4 Z | A_4 A_4$ Now production for  $A_4$  becomes; $A_4 \rightarrow b | bA_3 A_4 | bz | bA_3 A_4 Z$ 

$A_4 A_4 A_4$  was our problem  
 Last  $A_4$  is problematic one  
 followed by  $A_4 A_4$ . So, Once  
 write  $A_4 A_4 Z$  and once  $A_4 A_4$

Since we had  $A_4 \rightarrow b | bA_3 A_4 |$ Now,  $b | bA_3 A_4 |$  is written

~~as it is~~ as it is. Then the term following  $b$  is changed by  $Z$  once and once written by adding  $Z$  to  $bA_3 A_4$

Now this is in GNF

### Now the Grammar is:

 $A_1 \rightarrow A_2 A_3 | A_4 A_4$  $A_4 \rightarrow b | bA_3 A_4 | bz | bA_3 A_4 Z$  $Z \rightarrow A_4 A_4 | A_4 A_4 Z$  $A_2 \rightarrow b$  $A_3 \rightarrow a$ Here,

$A_4$  in  $A_3$  replaced  
 by  $b$  in  $A_4$  and others  
 copied as it is

Here  $A_2, A_4$  at start  
 violates the form of  
 GNF so they are replaced  
 by corresponding values.

 $A_1 \rightarrow bA_3 | bA_4 | bA_3 A_4 A_4 | bzA_4 | bA_3 A_4 Z A_4$  $A_4 \rightarrow b | bA_3 A_4 | bz | bA_3 A_4 Z$ . $Z \rightarrow bA_4 | bA_3 A_4 A_4 | bzA_4 | bA_3 A_4 Z A_4 | bA_4 Z | bA_3 A_4 Z$   
 $bZA_4 Z | bA_3 A_4 Z A_4 Z$  $A_2 \rightarrow b$  $A_3 \rightarrow a$ 

### ⊗. Backus-Naur Form (BNF):

Now we have everywhere  
 terminal symbols at  
 beginning so we are done  
 and this is required GNF

This is another notation used to specify the CFG. It is named so after John Backus, who invented it, and Peter Naur, who refined it. Here, concept is similar to CFG, only the difference is instead of using symbol " $\rightarrow$ " in production, we use symbol " $::=$ ". We enclose all non-terminals in angle brackets,  $\langle \rangle$ .

For Example: The BNF for identifiers is as; $\langle \text{Identifier} \rangle ::= \langle \text{letter or underscore} \rangle | \langle \text{Identifier} \rangle | \langle \text{symbol} \rangle$  $\langle \text{letter or underscore} \rangle ::= \langle \text{letter} \rangle | \langle \_ \rangle$  $\langle \text{symbol} \rangle ::= \langle \text{letter or underscore} \rangle | \langle \text{digit} \rangle$  $\langle \text{letter} \rangle ::= a | b | \dots | z$  $\langle \text{digit} \rangle ::= 0 | 1 | 2 | \dots | 9$

## \*. Context Sensitive Grammar:

A context sensitive grammar (CSG) is a formal grammar in which left hand sides and right hand sides of any production may be surrounded by a context of terminal and non-terminal symbols.

Formally CSG can be defined as;

A formal grammar,  $G_1 = (V, T, P, S)$  is the context sensitive if all the production 'P' are of the form;

$$\alpha A \beta \rightarrow \alpha \gamma \beta, \text{ where } A \in V, \alpha \beta \in (V \cup T)^* \text{ and } \gamma \in (V \cup T)^+.$$

The name context sensitive is explained by  $\alpha$  and  $\beta$  the form the context of  $A$  and determine whether  $A$  can be replaced with  $\gamma$  or not. Thus the production  $A \rightarrow \gamma$  can only be applied in contexts where  $\alpha$  occurs to left of  $A$  and  $\beta$  occurs to right of  $A$ .

Example:  $\{a^n b^n c^n | n \geq 1\}$  is a context sensitive defined as;

$$S \rightarrow aSBC | aBC$$

$$CB \rightarrow BC$$

$$aB \rightarrow ab$$

$$bB \rightarrow bb$$

$$bc \rightarrow bc$$

$$cC \rightarrow cc$$

## ⊗. Chomsky Hierarchy:

Grammar Type	Grammar Accepted	Language Accepted	Automation
Type-0	Unrestricted Grammar	Recursively Enumerable Language.	Turning Machine
Type-1	Context Sensitive Grammar	Context Sensitive Language	Linear Bounded Automation
Type-2	Regular Grammar	Context Free Language	Pushdown Automata
Type-3	Regular Grammar	Regular Language	Finite State Automation.

## \* Pumping lemma for CFL:

The pumping lemma for context free language is used to prove that a language is Not Context Free.

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  $S = uvxyz$  such that the following conditions must be true:

- i)  $uv^iz \in A$  for every  $i \geq 0$
- ii)  $|vz| > 0$
- iii)  $|vxy| \leq P$

everything similar to pumping lemma for Regular Expression discussed before  
only difference is condition i)  
i.e, instead of  $xyz$  here we have  $uvxyz$

# To prove that a language is Not Context Free, using Pumping lemma (for CFL) we 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  $uvxyz$ .

→ Show that  $uv^iz \in A$  for some i.

→ Then consider the ways that S can be divided into  $uvxyz$ .

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

→ S cannot be pumped == CONTRADICTION.

Example:- Show that  $L = \{a^N b^N c^N \mid N \geq 0\}$  is Not Context Free (CFL).

Solution:-

We prove this by contradiction. Let we assume that given language L is context free. If it is context free then we will have pumping length (say P). Now we take a string S such that  $S = a^P b^P c^P$ . Let we take  $P=4$  then,  $S = a^4 b^4 c^4$ . Now we divide S into parts  $uvxyz$ . So, there are two cases as follows:

Case-I:  $v$  and  $y$  each contain only one type of symbol.

i.e.,  $\underline{\underline{aaaa}} \underline{\underline{bbbb}} \underline{\underline{cccc}}$

Now for this case let we check condition  $uv^2xy^2z$  taking  $i=2$  i.e.,  $uv^2xy^2z$ .

$\Rightarrow \underline{\underline{aaaa}} \underline{\underline{abb}} \underline{\underline{ccc}}$

Here, we get  $a^6b^4c^5$

Since  $a^6b^4c^5$  does not follow the pattern  $a^Nb^Nc^N$  given in the question which means it does not belongs to given language  $L$ .

We assumed earlier that  $L$  is context free which means it must satisfy all three conditions, but it does not satisfy which lead to contradiction.

Case-II: Either  $v$  or  $y$  has more than one kind of symbols.

i.e.,  $\underline{\underline{aa}} \underline{\underline{abbb}} \underline{\underline{bbcccc}}$

Now, for this case let we check condition first i.e.,  $uv^2xy^2z$  taking  $i=2$  i.e.,  $uv^2xy^2z$ .

$\Rightarrow aaabbbaabbbbbbcccc$

Here the pattern of language  $a^Nb^Nc^N$  is not followed. So this string also does not belong to  $L$ .

So, we proved that  $uv^2xy^2z$  taking  $i=2$  all cases does not lie in our language. This means  $S$  cannot be pumped which leads to contradiction. Hence, the language is not CFL.

Example 2: Show that  $L = \{ww \mid w \in \{0,1\}^*\}$  is Not Context Free.

Solution:-

Assume that  $L$  is Context free then,  $L$  must have a Pumping length (say  $P$ ). Now we take a string  $S$  such that  $S = 0^P 1^P 0^P 1^P$ . If we take  $P=5$  then,  $S = 0^5 1^5 0^5 1^5$ .

Now we divide  $S$  into parts  $uvxyz$ . So, we have cases as follows:

Case 1:  $vxy$  does not straddle a boundary.

$\underline{\underline{00000}} \underline{\underline{11111}} \underline{\underline{00000}} \underline{\underline{11111}}$

$u \quad vxy \quad z$

$v$  is 1st  
 $x$  is 2nd  
 $y$  is 3rd  
respectively

straddle means it does not lie on either sides of boundary but it lies in one part only without crossing boundary.

Now we check our first condition  $uv^i xy^j z$  taking  $i=2$ .  
i.e.,  $uv^2 xy^2 z$ .

$$\Rightarrow 000001111110000011111$$

Here we get,  $0^5 1^7 0^5 1^5$

Since first half  $0^5 1^7 \neq$  second half  $0^5 1^5$  so, this does not belong to L. i.e.,  $0^5 1^7 0^5 1^5 \notin L$ .

(W<sub>any string</sub>  
W=W in question,  
but Here  
i.e., W ≠ W according  
to as per  
question.)

Case II: vxy straddles the first boundary.

i.e.,  $\underbrace{0000}_{u} \underbrace{111}_{v \text{ } x \text{ } y} \underbrace{110000011111}_{z}$

we can take in  
any way  
Here, V is 00  
x is 11  
y is 11

Now we check for our first condition  $uv^i xy^j z$  taking  $i=2$ .  
i.e.,  $uv^2 xy^2 z$ .

$$\Rightarrow 000000011111100000011111$$

Here we get,  $0^7 1^7 0^5 1^5$ .

Since  $0^7 1^7 \neq 0^5 1^5$ . So,  $0^7 1^7 0^5 1^5 \notin L$ .

Case III: vxy straddles the third boundary.

i.e.,  $\underbrace{0000}_{u} \underbrace{11111}_{v \text{ } x \text{ } y} \underbrace{0000011111}_{z}$

taken  
V as 00  
x as 1  
y as 11

Now again we check for our first condition  $uv^i xy^j z$  taking  $i=2$ .

i.e.,  $uv^2 xy^2 z$ .

$$\Rightarrow 00000111110000000111111$$

Here we get,  $0^5 1^5 0^7 1^7$

Since  $0^5 1^5 \neq 0^7 1^7$ . So,  $0^5 1^5 0^7 1^7 \notin L$ .

Case IV: vxy straddles the midpoint.

i.e.,  $\underbrace{0000}_{u} \underbrace{11111}_{v \text{ } x \text{ } y} \underbrace{000011111}_{z}$

taken  
V as 11  
x as 1  
y as 00

Now again we check for our first condition taking  $i=2$ . i.e.,  $uv^2 xy^2 z$ .

$$\Rightarrow 00000111111000000011111$$

Here we get,  $0^5 1^7 0^7 1^5$ .

Since,  $0^5 1^7 \neq 0^7 1^5$ . So,  $0^5 1^7 0^7 1^5 \notin L$ .

Since any of the cases does not satisfy our conditions, it leads to contradiction for our assumption. Hence, the given language is not CFL.

## ⊗ Closure Properties of Context Free Languages (CFL):

Following are some of the principal closure properties of context free languages:

i) The context free language are closed under union:

Given any two context free languages  $L_1$  and  $L_2$ , their union  $L_1 \cup L_2$  is also context free language.

ii) The context free language are closed under concatenation:

Given any two context free languages  $L_1$  and  $L_2$ , their concatenation  $L_1 \cdot L_2$  is also context free language.

iii) The context free language are not closed under intersection:

Given any two context free languages  $L_1$  and  $L_2$ , their intersection  $L_1 \cap L_2$  is not context free language.

iv) The context free language are closed under Kleen closure:

v) The context free language are not closed under complement.