Unit - 4

Context 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 $G=\{V,T,S,P\}$ 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 $A \in V$.
>
> (α: closure of V union T; $\in$: belongs to)

Components of CFG:

There are four components in CFG.

1). There is a finite set of symbol 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 after represented by capital letters. The terminals are analogus 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:

$S \rightarrow \epsilon$

$S \rightarrow 0S1$

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,
$\epsilon, 0, 1$ are the terminals defining T,
S is a variable symbol defining V,
And S is start symbol from where production starts.

CGF vs RE:

The CGF 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 are capable to 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 CFGs:

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:

$S \rightarrow aMb$
$M \rightarrow A|B$
$A \rightarrow \varepsilon|aA$
$B \rightarrow \varepsilon|bB.$

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 $lw_1r \rightarrow lw_2r$ (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.

⊛. 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,

$S \to S+S$
$S \to S/S$
$S \to (S)$
$S \to S-S$
$S \to S*S$
$S \to a$ ..... Grammar(2)

or this can also be written as:
$S \to S+S \mid S/S \mid (S) \mid S-S \mid S*S \mid a$
in single line

nothing just named the grammer

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

first body solve गर्ने पछाडि अरु

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.	a	S	$S \to a$	a; String 1
2.	a*a	S	$S \to S*S$	a*a; String 2
3.	(a*a)	S	$S \to (S)$	String 1 & 2; String 3
4.	(a*a)/a	S	$S \to S/S$	String 1 & 3; String 4
5.	a+(a*a)/a	S	$S \to S+S$	String 1 & 4; String 5
6.	a+(a*a)/a-a	S	$S \to S-S$	String 1 & 5; String 6

(3) (2) (1) (4)

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) - 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:

a) **Leftmost Derivation:** Here leftmost symbol (variable) is replaced first. (LMD)

b) **Rightmost Derivation:** Here rightmost symbol is replaced first. (RMD)

For Example:- Consider the previous example of deriving string $a+(a*a)/a-a$. with the grammar (2). [ie, $S \to S+S \mid S/S \mid (S) \mid S-S \mid S*S \mid a$]

#Now the leftmost derivation for the given string is;

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

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

#Now the rightmost derivation for the given string is;

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

Start from right and move towards body and replace from right

Q. Consider a Grammar;

$S \to aAS \mid a$

$A \to SbA \mid SS \mid ba$

This | symbol is or.
i.e, $S \to aAS$ } S can be replaced by any of these two
or, $S \to a$

If ε symbol comes then it is empty symbol.

Given a string aaabaaa, give leftmost and rightmost derivation.

Solution:

Leftmost Derivation: (LMD or lm)

$S \to aAS$

$S \to aSSS$ (Rule $A \to SS$)

$S \to aaSS$ (Rule $S \to a$)

$S \to aaaASS$ (Rule $S \to aAS$)

$S \to aaabaSS$ (Rule $S \to ba$)

$S \to aaabaaS$ (Rule $S \to a$)

$S \to aaabaaa$ (Rule $S \to a$)

Rightmost Derivation: (RMD or rm)

$S \to aAS$

$S \to aAa$ (Rule $S \to a$)

$S \to aSSa$ (Rule $A \to SS$)

$S \to aSaASa$ (Rule $S \to aAS$)

$S \to aSaAaa$ (Rule $S \to a$)

$S \to aSabaaa$ (Rule $A \to ba$)

$S \to aaabaaa$ (Rule $S \to a$)

⊛ 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, then any string α is in $(V \cup T)^*$ such that $S \to^* \alpha$ is a sentential form. If $S \to^*_{lm} \alpha$, then α is a left sentential forms, and if $S \to^*_{rm} \alpha$, then α is a right sentential form.

⊛. 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.

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

The language generated by a CFG 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 $G=(V,T,P,S)$, 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 ε. (epsilon)
→ If an interior node is labeled with a non terminal A and its childrens are $x1, x2, \ldots xn$ from left to right then there is a production P as;

$$A \rightarrow x1, x2, \ldots, xn \text{ for each } x_i \in T.$$

(each x_i belonging to T)

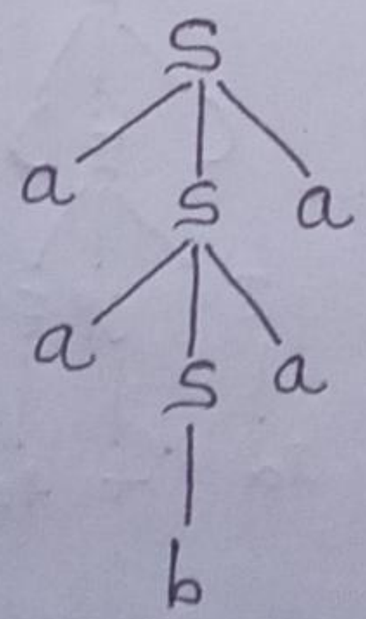
Example: Consider the grammar $S \rightarrow aSa|a|b|\varepsilon$.

Now, for string $S \rightarrow^* aabaa$.

We have,

$S \rightarrow aSa$
$S \rightarrow aaSaa$ (Rule $S \rightarrow aSa$)
$S \rightarrow aabaa$ (Rule $S \rightarrow b$).

So, the parse tree is →:



Q. Construct the parse tree for a*(a+b00), Considering S→I|S+S|S*S|(S). And I→a|b|Ia|Ib|I0|I1.

Solution: The parse tree is;

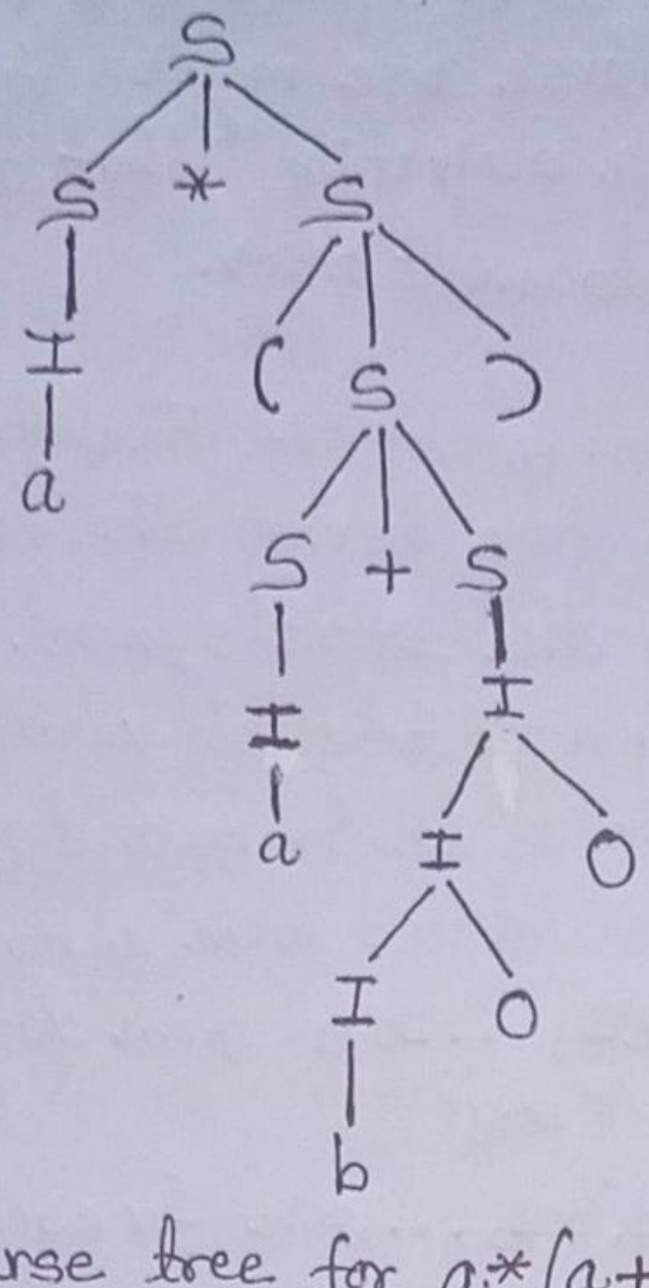


Fig. Parse tree for a*(a+b00).

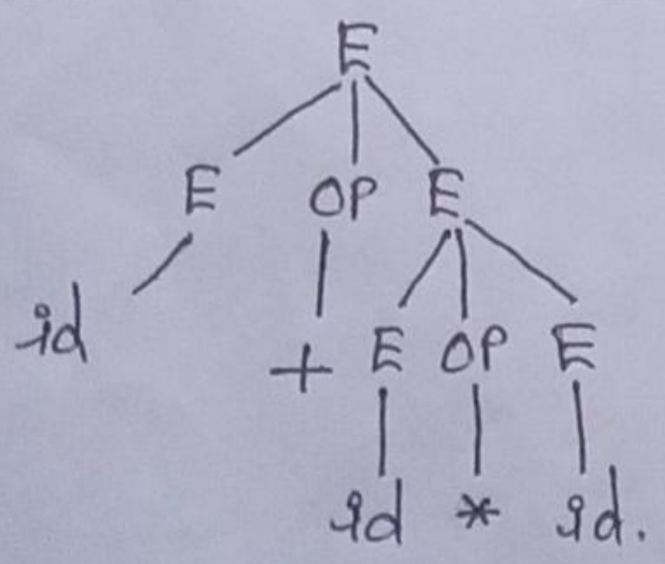
Q. Construct a grammar defining arithmetic expression and generate parse tree for id+id*idand (id+id)*(id+id).

E→EOPE|(E)|id

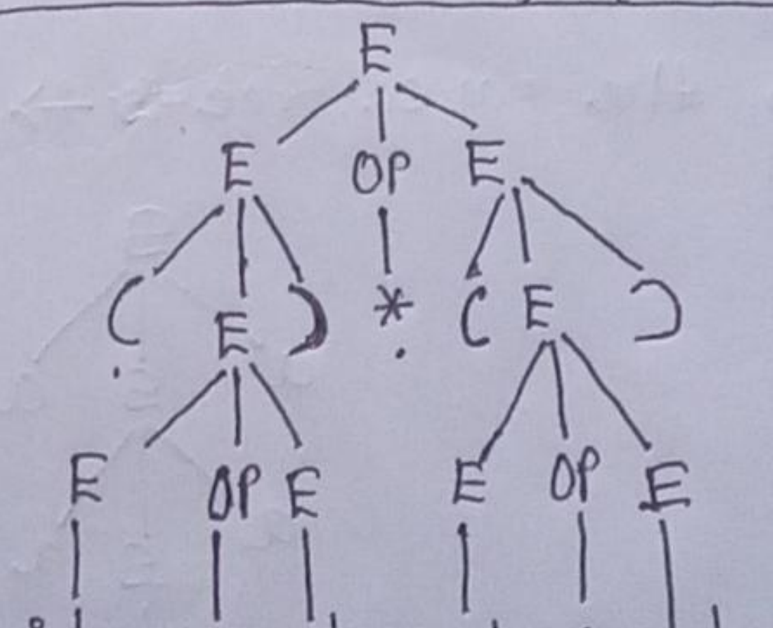
OP→+|−|*|/|↑.

Solution:-

Parse tree for id+id*id



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



⊛. Ambiguity in Grammar:-

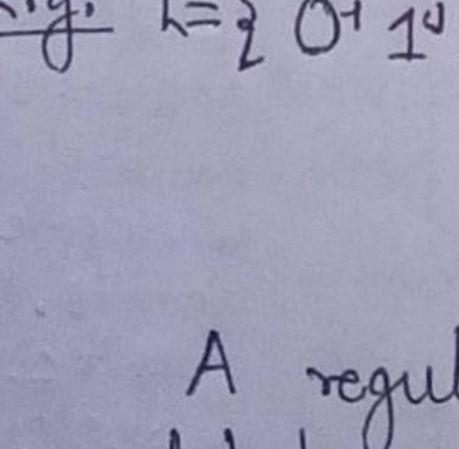
A Grammar $G_1 = (V, T, P, S)$ is said to be ambigious if there is a string $w \in L(G_1)$ for which we can derive two or more distinct derivation tree rooted at S and yielding w. In other words, a grammar is ambiguous if it can produce more than one leftmost or more than one rightmost derivation for the same string in the language of the grammar.

Example: $S \to AB \mid aaB$
$A \to a \mid Aa$
$B \to b$

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

$S \to AB$
$S \to AaB$ (Rule $A \to Aa$)
$S \to aaB$ (Rule $A \to a$)
$S \to aab$ (Rule $B \to b$).

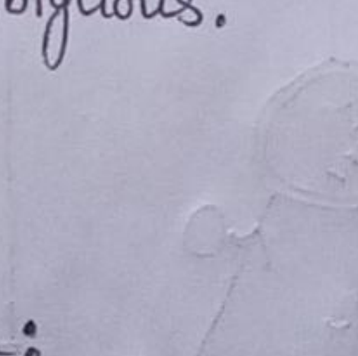
the parse tree for this derivation is;



Also,

$S \to aaB$
$S \to aab$ (Rule $B \to b$).

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 $A \to wB$ or $A \to w$ where $A, B \in V$ and $w \in T$ is called right linear.

(omega, i.e, any string — w)

For Example: $S \to 00B \mid 11C$

$B \to 11C \mid S \mid 1$

$C \to 00B \mid 00$

ii) Left Linear Regular Grammar:

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

For Example: $S \to B00 \mid C11$

$B \to C11 \mid S \mid 1$

$C \to B00 \mid 00$

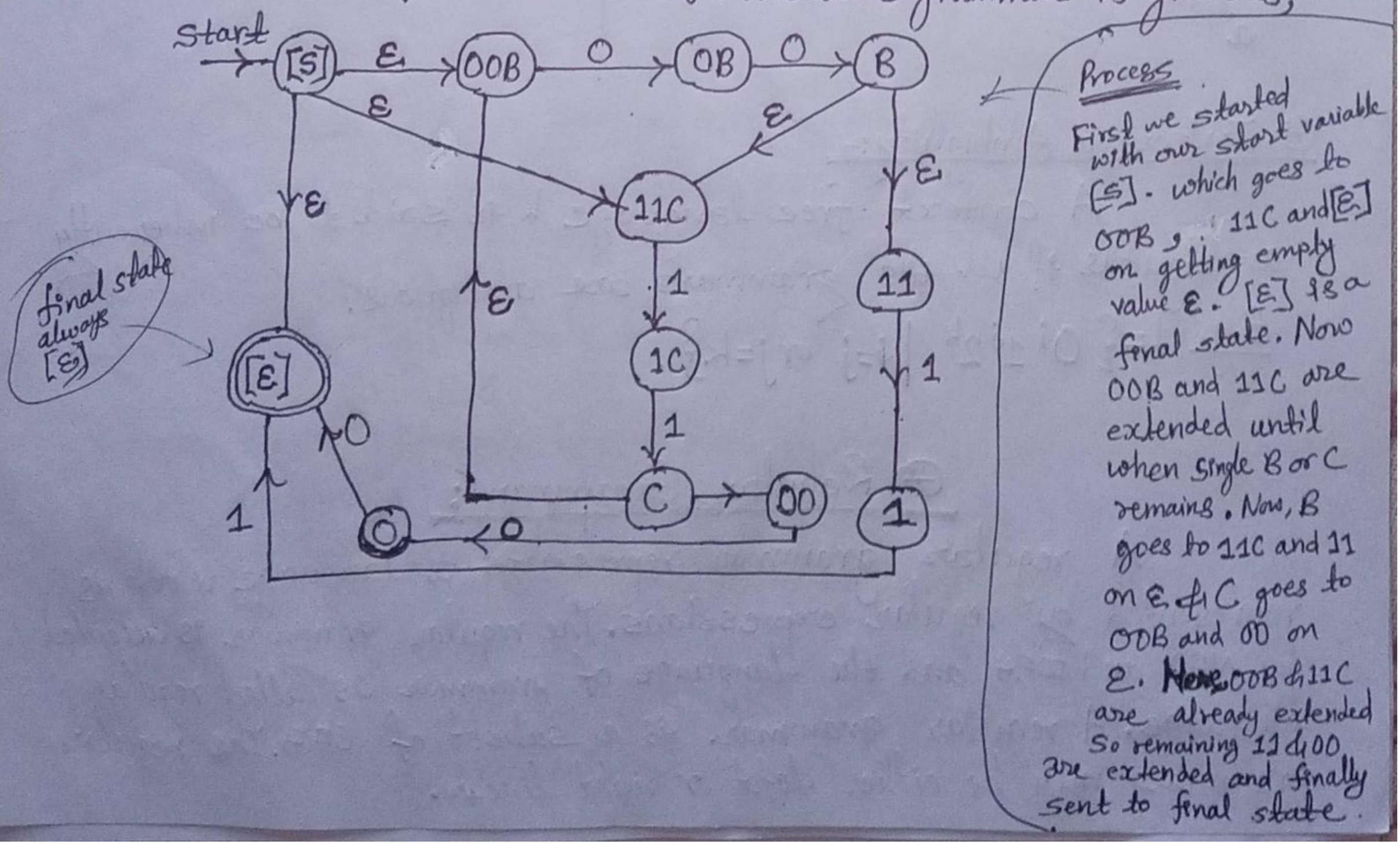
⊛. Equivalence of Regular Grammar and Finite Automata:

Example 1: $S \to 00B \mid 11C \mid \varepsilon$

$B \to 11C \mid 11$

$C \to 00B \mid 00$

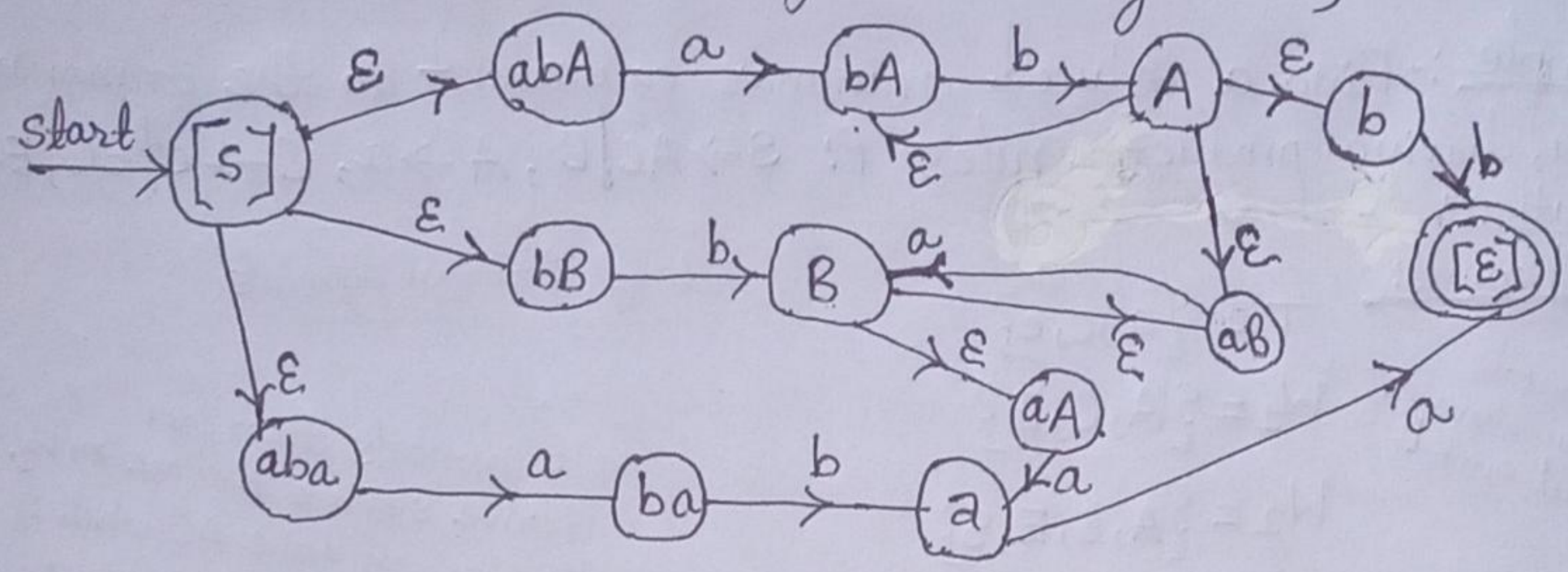
Solution: The finite automation for the above grammar is given as;



Example 2:- S → abA | bB | aba
A → b | aB | bA
B → aB | aA

Solution:

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



⊛. Simplification of CFGs:

In CFGs 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. Simplification consists of: i) Reduction of CFG ii) Removal of Unit Productions iii) Removal of Null Productions.

i) Reduction of CFG:

CFGs are reduced in two phases:

(1) This is also called elimination of useless symbols.

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

Derivation Procedure:

Step 1: Include all Symbols W_1, 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.

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

Phase 2: Derivation of an equivalent grammar G'', from the CFG, G', such that each symbol appears in a sentential form.

Derivation Procedure:

Step1: Include the start Symbol in Y_1 and initialize $i=1$.

Step2: Include all symbols Y_{i+1}, that can be derived from Y_i.

Similar to phase 1. Only Step 1 differ as Y_i used instead of W_i

Step3: Increment i and repeat Step2 until $Y_{i+1}=Y_i$.

Step4: Include all production rules that have Y_i in it.

Example:- Find a reduced grammar equivalent to the grammar G_1, having production rules P: $S \to AC|B$, $A \to a$, $C \to c|BC$, $E \to aA|e$.

Solution:

Phase 1:

$T = \{a, c, e\}$ ← Set of terminal symbols

$W_1 = \{A, C, E\}$ ← all symbols that can derive terminal symbol

$W_2 = \{A, C, E, S\}$ ← 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.

$W_3 = \{A, C, E, S\}$ ← 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 $G = \{V, T, S, P\}$

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

Any production rule

Start symbol

Now, Production Rule can be defined as;

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

First look at P: given in question now copy it eliminating symbols that are not in G_1'

Phase 2:

In phase 2 we will look at grammer G_1' and Production P obtained ~~always~~ in phase 1

$Y_1 = \{S\}$ ← only start symbol included

$Y_2 = \{S, A, C\}$ ← All symbols that Y_1 i.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: $S \to AC$, $A \to a$, $C \to c$

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 $A \to B$ where $A, B \in$ Non Terminals is called Unit Production.

Procedure:

Step1: To remove $A \to B$, add production $A \to x$ to the grammar rule whenever $B \to x$ occurs in the grammar. [$x \in$ Terminal, x can be Null].

Step2: Delete $A \to B$ from the grammar.

Step3: Repeat from step1 until all Unit Productions are removed.

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

Solution:

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

For $M \to N$

Since, $N \to a$, we add $M \to a$ ← according to step 1

P: $S \to XY, X \to a, Y \to Z|b, Z \to M, M \to a, N \to a$ ← P now changed as this

For $Z \to M$

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

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

For $Y \to Z$

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

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

Here, $Y \to Z|b$ since $Z \to a$ so, $Y \to a|b$ 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: $S \to XY, X \to a, Y \to a|b$.

iii) Removal of Null Productions:

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

Procedure:

Step1 : To remove $A \to \varepsilon$, look for all productions whose right side contains A.

Step2 : Replace each occurences of 'A' in each of these productions with ε.

Step3 : Add the resultant productions to the Grammar.

Example: Remove Null Productions from the following Grammar.

$S \to ABAC,\ A \to aA|\varepsilon,\ B \to bB|\varepsilon,\ C \to C$

Solution:

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

To eliminate $A \to \varepsilon$

$S \to ABAC$

$S \to ABC|BAC|BC$ ← According to step 1.

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

Also,

$A \to aA$ ← Since according to step1 $A \to aA|\varepsilon$ also contains A on right side so this is also replaced by ε

$A \to a$ ← on replacing A by ε (i.e, empty)

New Production: $S \to ABAC|ABC|BAC|BC$

$A \to aA|a$ ← we have new a so ~~a changed by~~ ε no longer remains as eliminated by a

$B \to bB|\varepsilon$

$C \to c$

Others remains same

To eliminate $B \to \varepsilon$ ← for this we look at new production above

$S \to ABAC|ABC|BAC|BC$ ← Since it contains B on right side as step1

$S \to AAC|AC|C$ ← ABAC if B is ε = AAC, For ABC if B is ε then AC, for BAC if B is ε then AC but AC already there so no need to write. Similarly BC, B on getting ε give C.

Also,

$B \to bB$

$B \to b$ ← bB, B on getting ε

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

$A \to aA|a$

$B \to bB|b$

$C \to c.$

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:

$A \to a$
$A \to BC$ where A, B and C are non-terminals & a is a terminal.

Steps to convert a given CFG to Chomsky Normal Form:

Step1: If the Start symbol S occurs on some right side, create a new Start symbol S' and a new production $S' \to S$.

Step2: Remove Null Productions. (By Method that we discussed earlier).

Step3: Remove Unit Productions. (By Method that we discussed earlier).

Step4: Replace each Production $A \to B_1 \dots B_n$ where $n > 2$, with $A \to B_1 C$ where $C \to B_2 \dots B_n$. Repeat this step for all productions having two or more symbols on the right side.

Step5: If the right side of any Production is in the form $A \to aB$ where 'a' is a terminal and A and B are non-terminals, then the production is replaced by $A \to XB$ and $X \to a$. Repeat this step for every Production which is of the form $A \to aB$.

Example: Convert the following CFG to CNF:

P: $S \to ASA \mid aB$, $A \to B \mid S$, $B \to b \mid \varepsilon$.

Solution:

Step1: Since S appears in RHS, we add a new state S' and $S' \to S$ is added to the production.

P: $S' \to S$, $S \to ASA \mid aB$, $A \to B \mid S$, $B \to b \mid \varepsilon$.

Step2: Remove the Null Productions: $B \to \varepsilon$ and $A \to \varepsilon$.

After Removing $B \to \varepsilon$: P: $S' \to S$, $S \to ASA \mid aB \mid a$, $A \to B \mid S \mid \varepsilon$, $B \to b$

After Removing $A \to \varepsilon$: P: $S' \to S$, $S \to ASA \mid aB \mid a \mid AS \mid SA \mid S$, $A \to B \mid S$, $B \to b$

always look at latest production ☞

Step3: Remove the Unit Productions: $S \to S$, $S' \to S$, $A \to B$ and $A \to S$:

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

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

i.e $S' \neq S$

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

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

(यसमा B लाई replace गरिन्छ)

After Removing A→B:

P: S'→ASA|aB|a|AS|SA, S→ASA|aB|a|AS|SA,
A→b|S, B→b.

After Removing A→S:

P: S'→ASA|aB|a|AS|SA, S→ASA|aB|a|AS|SA,
A→b|ASA|aB|a|AS|SA, B→b.

Step4: Now find out the productions that has more than two variables on RHS.

S'→ASA, S→ASA and A→ASA.

After removing these, we get:

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

Step5: Now change the productions: S'→aB, S→aB and A→aB

Finally we get: P: S'→AX|YB|a|AS|SA,
S→AX|YB|a|AS|SA,
A→b|AX|YB|a|AS|SA,
B→b,
X→SA,
Y→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→b
A→bC1C2.....Cn.

where A, C1,....,Cn are Non-Terminals and b is a Terminal.

Steps to Convert a given CFG to GNF:-

Step1: 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).

Step2: 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 A_i 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 $A_i \to A_j x$, then, $i < j$ and should never be $i \geq j$.

Example: Convert the given CFG to GNF:
$S \to CA \mid BB$, $B \to b \mid SB$, $C \to b$, $A \to a$

Last step4 is hard to understand better to study with video by neso academy

Solution:-

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 CGF is already in Chomsky Normal Form so, we are also done with step2.

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;

$A_1 \to A_2A_3 \mid A_4A_4$
$A_4 \to b \mid A_1A_4$
$A_2 \to b$
$A_3 \to a$

Step4:

$A_1 \to A_2A_3 \mid A_4A_4$
$A_4 \to b \mid A_2A_3A_4 \mid A_4A_4A_4$
$A_4 \to b \mid bA_3A_4 \mid A_4A_4A_4$

Let we take for. $A_1 \to A_2A_3$ we have form $A_i \to A_j x$. 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_j that is in any other part of production

A_1 repraced by $A_2A_3 \mid A_4A_4$ and others copied as they are but this again not following the rule.

If it is of form $bG_1, \ldots G_n$ then it is in GNF so, now this part is okay.

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.

$Z \rightarrow A_4A_4Z \mid A_4A_4$

Now production for A_4 becomes;

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

$A_4A_4A_4$ was our problem last A_4 is problematic one followed by A_4A_4. So, once write A_4A_4Z and once A_4A_4

Since we had $A_4 \rightarrow b \mid bA_3A_4 \mid A_4A_4A_4$.
Now, $b \mid bA_3A_4 \mid$ is written as it is. Then the term following b is changed by Z once and once written by adding Z to bA_3A_4

Now this is in GNF

Now the Grammar is:

$A_1 \rightarrow A_2A_3 \mid A_4A_4$

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

$Z \rightarrow A_4A_4 \mid A_4A_4Z$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Here,

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

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

$Z \rightarrow bA_4 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4ZA_4 \mid bA_4Z \mid bA_3A_4Z \mid bZA_4Z \mid bA_3A_4ZA_4Z$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

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.

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

⊛. Bakus-Naur Form (BNF):

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

For Example: The BNF for identifiers is as;

<identifier> ::= <letter or underscore> | <identifier> | <symbol>

<letter or underscore> ::= <letter> | <_>

<symbol> ::= <letter or underscore> | <digit>

<letter> ::= a | b | .... | z.

<digit> ::= 0 | 1 | 2 | .... | 3.

⊛. 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, $G=(V,T,p,S)$ is the context sensitive if all the production 'P' are of the form;

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

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

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

$S \rightarrow aSBC \mid aBC$
$CB \rightarrow BC$
$aB \rightarrow ab$
$bB \rightarrow bb$
$bc \rightarrow bc$
$cC \rightarrow cc$

⊛. Chomsky Hierarchy:

Grammar Type	Grammar Accepted	Language Accepted	Automation
Type-0	Unrestriced 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:

(application of pumping lemma)

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^ixy^iz is in A for every $i \geq 0$
ii) $|vy| > 0$
iii) $|vxy| \leq p$

(everything similar to pumping lemma for Regular Expression discussed before only difference is condition ① i.e, instead of xy^iz here we have uv^ixy^iz)

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 $uv^ixy^iz \in A$ 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, a a a a b b b b c c c c
u v x y z

i.e, $a^4b^4c^4$

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

$|vy|>0$ & $|xvy|\geq p$ | since, $p=4$, $x=5, v=2, y=1$, $xvy=8$, $\therefore |xvy|\geq p$, i.e, $|8|\geq 4$

यो बुझ्नलाई मात्र हो directly 1 line मा लेख्दा $a^6b^4c^5$ ले L as mege

$\Rightarrow$ aaaaaabbbbccccc

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.

first condition नै match गरेन so no need to check second & third conditions. If it matches then we check

does not satisfy first condition

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

i.e, a a a a b b b b c c c c
u v x y z

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

$\Rightarrow$ aa aabb aabbbbbcccc

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 xv^ixy^iz 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 $S=0^P1^P0^P1^P$. If we take $P=5$ then, $S=0^51^50^51^5$.

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

Case1: vxy does not straddle a boundry.

0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1
u vxy z

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

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

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

(W any string
W=W in question,
but Here
i.e, $W \neq W$ according
to as our
question.

$\Rightarrow$ 000001111111 0000011111

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

Case II: vxy straddles the first boundry.

we can take in any way
↓
Here, V is 00
x is 1
& y is 11

i.e, 00000111111000011111
u v x y z

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

$\Rightarrow$ 000000011111110000011111.

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 boundry.

i.e, 000001111100001111 1
u v x y z

taken
V as 00
x as 1
y as 11

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

$\Rightarrow$ 000001111100000011111

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, 00000111 11100 0011111
u v x y 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^2xy^2z.

$\Rightarrow$ 000001111111000000011111

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 L1 and L2, their union L1 ∪ L2 is also context free language.

ii) The context free language are closed under concatenation:

Given any two context free languages L1 and L2, their concatenation L1.L2 is also context free language.

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

Given any two context free languages L1 and L2, their intersection L1 ∩ L2 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.