Theory of Computation (CT-502)

Course Instructor
ANUJ GHIMIRE

DISCLAIMER

- *This document does not claim any originality and cannot be used as a substitute for prescribed textbooks.*
- *The information presented here is merely a collection from various sources as well as freely available material from internet and textbooks.*
- *The ownership of the information lies with the respective authors or institutions.*

Chapter 3

Context Free Language and Push Down Automata

Context Free Grammar

- We have introduced two different but equivalent, methods of describing languages i.e. ***Finite Automata*** and ***Regular Expressions***.
- Expressions used to represent regular languages are called regular expressions and the model that recognizes the regular expression is finite automata.
- However, there is a class of languages, i.e. Non Regular Languages, which are not processed by an NFA/DFA.
- In order to recognize these languages, there is a need to come up with a new kind of automata.

Context Free Grammar

- This is what brings us to the concept of ***Push-Down Automata (PDA).***
- A PDA is a more powerful version of automata which can recognize certain languages that are not recognized by an NFA/DFA.
- Languages recognized by PDA are generated by a set of rules called ***Context-Free Grammar(CFG)*** and are known as ***Context-Free Languages (CFL).***
- Context Free Languages are a larger class of languages that encompasses all regular languages and some non regular languages.

Context Free Grammar

- Context Free Grammar are the set of rules that are used to generate the string that belongs to Context Free Language
- Context Free Grammars are more expressive it is because if a language L is accepted by a finite automata then L can be generated by a context-free grammar as well. ***Beware: The converse is NOT true***
- CFG can describe certain features that have a recursive structure, which makes them useful in a variety of applications.

Context Free Grammar

- CFG are studied in fields of theoretical computer science compiler design, and linguistics.
- Designers of compilers and interpreters for programming languages often start by obtaining a grammar for the language.
- Most compilers and interpreters contain a component called a parser that extracts the meaning of a program prior to generating the compiled code or performing the interpreted execution.
- So construction of parser will be facilitated after we describe the CFG.

Context Free Grammar

- Formally Context Free Grammar ***G*** can be defined by quadruple:

$$G=(V, \sum, R, S)$$

Where, V= Set of variables / set of non-terminals (upper case letter) and set of terminals (lower case letter)

$\sum$=Set of terminals (lower case letter)

S= Start symbol

R =Set of production rules

R is of the form:

$P \rightarrow \alpha$, where $\alpha=\{V, \sum\}^*$ and $P \in V$ {***single non terminals***)

Context Free Grammar

- Let us consider a CFG as

$G=(V, \Sigma, R, S)$

where, V= {A,B,S,a,b}

$\Sigma = \{a,b\}$

R consists of:

$S \rightarrow AbB$

$A \rightarrow aA$

$A \rightarrow \varepsilon$

$B \rightarrow bB$

$B \rightarrow \varepsilon$

***S**= S is a start symbol*

Context Free Grammar

- Above CFG can be used to generate the string that belongs to the language L={ $\mathbf{a^n bb^m}$ for m,n>=o}
- *For example the string w=aaabbbb can be derived from the given grammer by using the production rule.*

Context Free Grammar

The process of string ***w=aaabbbb*** generation can be shown as:

$S \rightarrow AbB$ *using (i)*

$\rightarrow \boldsymbol{a}\boldsymbol{A}bA$ *using (ii)*

$\rightarrow a\boldsymbol{a}\boldsymbol{A}bA$ *using (ii)*

$\rightarrow aa\boldsymbol{a}\boldsymbol{A}bA$ *using (ii)*

$\rightarrow aaa\boldsymbol{\varepsilon}bA$ *using (iii)*

$\rightarrow aaab\boldsymbol{b}\boldsymbol{B}$ *using (iv)*

$\rightarrow aaabb\boldsymbol{b}\boldsymbol{B}$ *using (iv)*

$\rightarrow aaabbb\boldsymbol{b}\boldsymbol{B}$ *using (iv)*

$\rightarrow aaabbbb\boldsymbol{\varepsilon}$ *using (v)*

$\rightarrow aaabbbb$

which is the required string generated by the given grammar.

Context Free Grammar

- ***Rules of grammar for the regular expression are:***
  - ***a****

    $S \rightarrow aS$---------- *(i)*

    $S \rightarrow \varepsilon$ ----------*(ii)*
  - ***a+b***

    $S \rightarrow a$---------- *(i)*

    $S \rightarrow b$---------- *(ii)*
  - ***(a+b)****

    $S \rightarrow aS$---------- *(i)*

    $S \rightarrow bS$---------- *(ii)*

    $S \rightarrow \varepsilon$ ----------*(iii)*

Context Free Grammar

- $a^*.b^*$

  $S \rightarrow AB$---------- (i)

  $S \rightarrow \varepsilon$ ----------(ii)

  $A \rightarrow aA$---------- (iii)

  $A \rightarrow \varepsilon$ ----------(iv)

  $B \rightarrow bB$---------- (v)

  $B \rightarrow \varepsilon$ ----------(vi)

- a^*+b^*

  $S \rightarrow A$---------- (i)

  $S \rightarrow B$---------- (ii)

  $S \rightarrow \varepsilon$ ----------(iii)

  $A \rightarrow aA$---------- (iv)

  $A \rightarrow \varepsilon$ ----------(v)

  $B \rightarrow bB$---------- (vi)

  $B \rightarrow \varepsilon$ ----------(vii)

CFG_Example_1

Write the CFG for the language that generates the string of at least 2 length over ∑ {0,1}.

Solution,

The regular expression for the language that generates the string of at least 2 length is:

R.E= (0+1)(0+1)(0+1)*

Its rule for the grammar can be written as:

$S \rightarrow AAB$---------- *(i)*

$A \rightarrow 1$ ---------- *(ii)*

$A \rightarrow 0$ ----------*(iii)*

$B \rightarrow 1B$---------- *(iv)*

$B \rightarrow 0B$ ----------*(v)*

$B \rightarrow \varepsilon$ ----------*(vi)*

CFG_Example_1

Let G be the CFG for the given language which is given by quadruple as:

G=***(V, ∑, R, S)***

where, V= {A, B, S, 0, 1} are set of variables

∑= {0, 1} are set of terminals

R consists of:

$S \rightarrow AAB$---------- *(i)*

$A \rightarrow 1$ ---------- *(ii)*

$A \rightarrow 0$ ----------*(iii)*

$B \rightarrow 1B$---------- *(iv)*

$B \rightarrow 0B$ ----------*(v)*

$B \rightarrow \varepsilon$ ----------*(vi)*

S*= S is a start symbol*

CFG_Example_2

- Write the CFG for the language that generates the palindrome string over ∑ {a,b}.

Solution,

The set of palindrome string over {a,b} are:

{a, b, aa, bb,aba, baab, bbbaabbb, aaabbaaa, ababbaba.....}

The properties of palindrome string is that the first symbol must match with last symbol, second last symbol must match with second last symbol and so on.

CFG_Example_2

- Let us consider a CFG as

$G=(V, \Sigma, R, S)$

where, **V**= {A, a, b} are set of variables

$\Sigma = \{a,b\}$ set of terminals

R consists of:

$A \rightarrow aAa$

$A \rightarrow bAb$

$A \rightarrow a$

$A \rightarrow b$

$A \rightarrow \varepsilon$

S = A is the start symbol

CFG_Example_3

- Construct CFG for language: $\{a^m b^n : m>n, n\geq o\}$.

Solution,

The set of string over {a,b} are:

{a, aa, aaa, aaab, aaaabbb, aaaaabbbb, aaaabbb, aaaaab.....}

The properties of string generated by above CFG is that the occurrence of ***a*** is always greater than ***b,*** also the number of ***b*** can be zero i.e. no occurrence of ***b***.

CFG_Example_3

- Let us consider a CFG as

$G=(V, \Sigma, R, S)$

where, **V**= {S, A, a, b}

$\Sigma = \{a,b\}$

R consists of:

$S \rightarrow SA$

$S \rightarrow aS \mid a$

$A \rightarrow aAb \mid \varepsilon$

S = S is the start symbol

CFG_Example_4

- Construct CFG for language: $\{a^m b^n : m, n>0, m \geq n\}$.

Solution,

The set of string over {a,b} are:

{ab, aabb, aab, aaabb, aaaaabbbb, aaaabbb, aaaaab.....}

The properties of string generated by above CFG is that the occurrence of ***a*** and ***b*** can be same as well as the number of ***a*** can be more than number of ***b***.

CFG_Example_4

- Let us consider a CFG as

$G=(V, \Sigma, R, S)$

where, **V**= {S, a, b}

$\Sigma = \{a,b\}$

R consists of:

$S \rightarrow AB$

$A \rightarrow aA \mid a$

$B \rightarrow aBb \mid b$

S = S is the start symbol

Derivation of String from CFG

- Let G = (V, ∑, R, S) be a context free grammar. If w_1 w_2 w_3.....w_n are strings over variable V such that:

  $\boldsymbol{w_1} \rightarrow \boldsymbol{w_2} \rightarrow \boldsymbol{w_3}$ .....$\rightarrow \boldsymbol{w_n}$ then we can say w_n is derivable.

- The sequence of substitution to obtain a string is called derivation.
- Say w_i derives w_n i.e. $w_1 \rightarrow^* w_n$. Then, the sequence of steps to obtain w_n from w_i is called derivation.
- A derivation of a string ***w*** in a given grammar G is a sequence of substitutions starting with the start symbol and resulting in ***w***.

Derivation of String from CFG

- ***Language of CFG (L(G)):***
  - If G = (V, ∑, R, S) be a CFG, then the language of G denoted by L(G) is the set of terminal strings that have derivations from start symbol.

$$i.e.\ L(G) = \{ w \in \Sigma^* : S \rightarrow^* w \}$$

Derivation of String from CFG

- There are two types of derivation:
  - Left Most Derivation (LMD)
  - Right Most Derivation (RMD)
- In the leftmost derivation (LMD), at each step the production rule for the leftmost non-terminal is used whereas in rightmost derivation (RMD) the production rule for the rightmost non-terminal is used.

Derivation of String from CFG

Consider a grammar G as:

$G=(V, \Sigma, R, S)$

where, V= {S, A, B, a, b} are set of terminals

Σ= {a, b} are set of terminals

R consists of:

$S \rightarrow aAS$---------- *(i)*

$S \rightarrow aS$ ---------- *(ii)*

$S \rightarrow b$ ----------*(iii)*

$A \rightarrow bB$---------- *(iv)*

$A \rightarrow a$ ----------*(v)*

$B \rightarrow aA$ ----------*(vi)*

$B \rightarrow b$ ----------*(vii)*

S*= S is a start symbol*

Derivation of String from CFG

Let us derive the string ***w=abaaab*** using different derivation.

Left Most Derivation (LMD)

$S \rightarrow aAS$----------*rule (i)*

$\rightarrow a\mathbf{bB}S$----------*rule (iv)*

$\rightarrow ab\mathbf{aA}S$----------*rule (vi)*

$\rightarrow aba\mathbf{a}S$----------*rule (v)*

$\rightarrow abaa\mathbf{aS}$----------*rule (ii)*

$\rightarrow abaaa\mathbf{b}$----------*rule (iii)*

Derivation of String from CFG

Right Most Derivation (RMD)

$S \rightarrow aAS$----------*rule (i)*

$\rightarrow aA\mathbf{a}\mathbf{S}$----------*rule (ii)*

$\rightarrow aAa\mathbf{b}$----------*rule (iii)*

$\rightarrow a\mathbf{b}\mathbf{B}ab$----------*rule (iv)*

$\rightarrow ab\mathbf{a}\mathbf{A}ab$----------*rule (vi)*

$\rightarrow aba\mathbf{a}ab$----------*rule (v)*

Derivation Tree/ Parse Tree

- Derivation tree is the pictorial description of the derivation of the string using the production rule defined by the grammar.
- It is a hierarchical representation of derivation that shows how the start symbol of a grammar derives a strings in language.
- For a CFG, $G = (V, \sum, R, S)$, a parse tree has following properties:
  - The root is labeled by start symbol (S).
  - Each interior nodes of parse tree are variables (V).

Derivation Tree/ Parse Tree

- Each leaf node is labeled by terminal symbol ($\sum$) or ϵ .
- If an interior node is labeled with non-terminal ***A*** and its children are x_1 ,x_2 ,....x_n from left to right then there is a production P as:

$$A \rightarrow x_1 x_2 \ldots\ldots\ldots x_n \text{ for each } x_i \epsilon \sum$$

- Left hand side of the production rule is the root node at each level and the right hand side of the production rule is divided into multiple branches

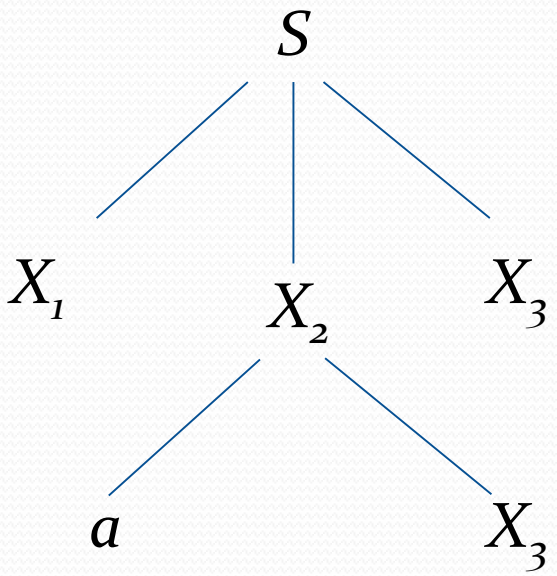
Derivation Tree/ Parse Tree

Consider a production rule

$$S \rightarrow X_1 X_2 X_3$$

$$X_2 \rightarrow aX_3$$

Its derivation tree can be drawn as:



Derivation Tree/ Parse Tree

Consider a grammar G as:

G=***(V, ∑, R, S)***

where, V= {S, A, B, a, b} are set of terminals

∑= {a, b} are set of terminals

R consists of:

$S \rightarrow aAS$---------- *(i)*

$S \rightarrow aS$ ---------- *(ii)*

$S \rightarrow b$ ----------*(iii)*

$A \rightarrow bB$---------- *(iv)*

$A \rightarrow a$ ----------*(v)*

$B \rightarrow aA$ ----------*(vi)*

$B \rightarrow b$ ----------*(vii)*

S*= S is a start symbol*

Derive a string w= abaaab using LMD draw parse tree

Derivation Tree/ Parse Tree

Left Most Derivation (LMD)

$S \rightarrow aAS$----------*rule (i)*

$\rightarrow a\mathbf{bB}S$----------*rule (iv)*

$\rightarrow ab\mathbf{aA}S$----------*rule (vi)*

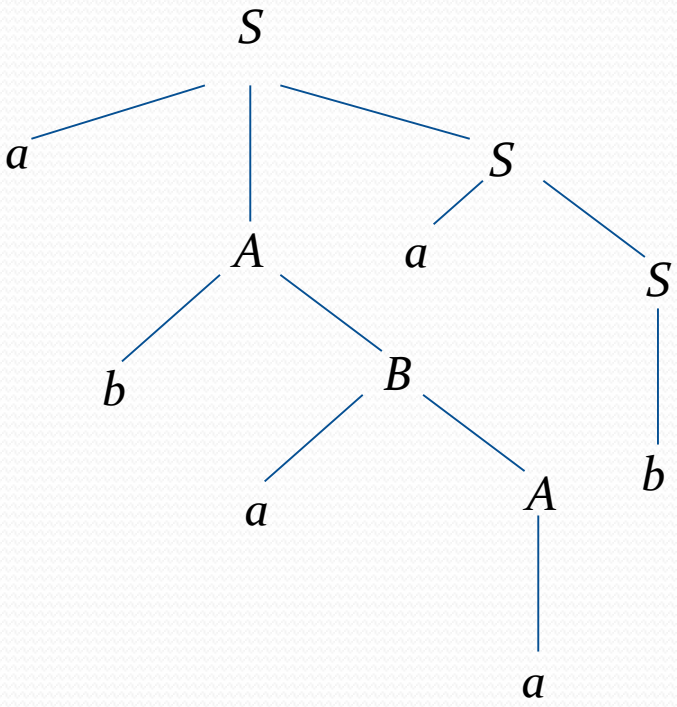
$\rightarrow aba\mathbf{a}S$----------*rule (v)*

$\rightarrow abaa\mathbf{aS}$----------*rule (ii)*

$\rightarrow abaaa\mathbf{b}$----------*rule (iii)*

Derivation Tree/ Parse Tree

Parse Tree for derivation



Derivation Tree/ Parse Tree

Consider a grammar G as:

G=***(V, ∑, R, S)***

where, V= {S, A, B, a, b} are set of terminals

∑= {a, b} are set of terminals

R consists of:

$S \rightarrow aA \mid bB$

$A \rightarrow b \mid bS \mid aAA$

$B \rightarrow a \mid aS \mid bBB$

S= *S is a start symbol*

For the string ***bbaababa*** find its LMD, RMD and draw parse tree.

Derivation Tree/ Parse Tree

Left Most Derivation (LMD)

$S \rightarrow bB$----------[using $S \rightarrow bB$]

$\rightarrow b\mathbf{bBB}$----------[using $B \rightarrow bBB$]

$\rightarrow bb\mathbf{a}B$----------[using $B \rightarrow a$]

$\rightarrow bba\mathbf{aS}$----------[using $B \rightarrow aS$]

$\rightarrow bbaa\mathbf{bB}$----------[using $S \rightarrow bB$]

$\rightarrow bbaab\mathbf{aS}$----------[using $B \rightarrow aS$]

$\rightarrow bbaaba\mathbf{bB}$----------[using $S \rightarrow bB$]

$\rightarrow bbaabab\mathbf{a}$----------[using $B \rightarrow a$]

Derivation Tree/ Parse Tree

Right Most Derivation (RMD)

$S \rightarrow bB$----------[using $S \rightarrow bB$]

$\rightarrow b\mathbf{bBB}$----------[using $B \rightarrow bBB$]

$\rightarrow bbB\mathbf{aS}$----------[using $B \rightarrow aS$]

$\rightarrow bbBa\mathbf{bB}$----------[using $S \rightarrow bB$]

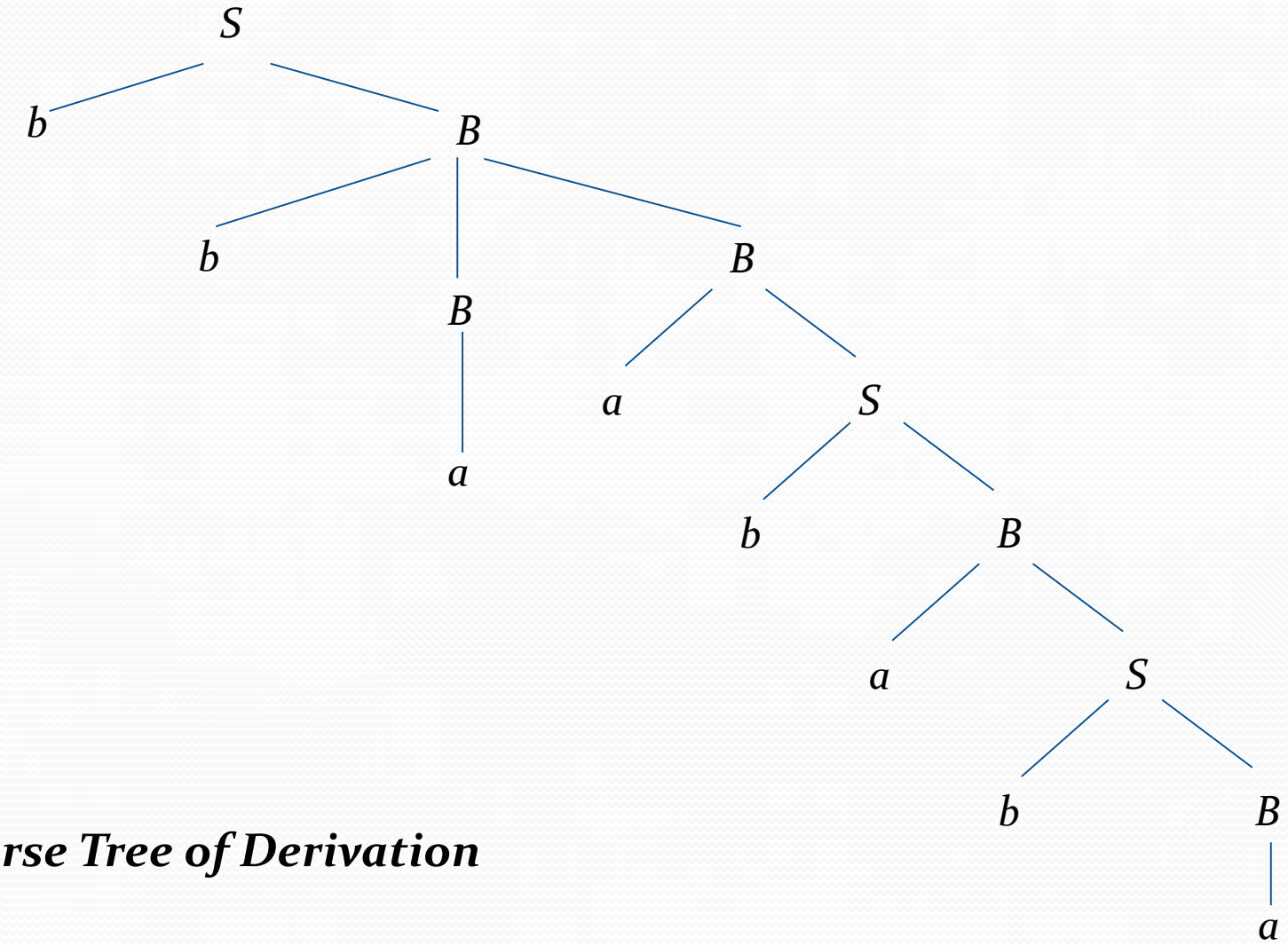
$\rightarrow bbBab\mathbf{aS}$----------[using $B \rightarrow aS$]

$\rightarrow bbBaba\mathbf{bB}$----------[using $S \rightarrow bB$]

$\rightarrow bbBabab\mathbf{a}$----------[using $B \rightarrow a$]

$\rightarrow bb\mathbf{a}ababa$----------[using $B \rightarrow a$]

Derivation Tree/ Parse Tree



Parse Tree of Derivation

Ambiguous Grammar

- A grammar G = (V, ∑, R, S) is said to be ambiguous if there is a string ***w Є L(G)*** for which we can derive two or more distinct derivation tree rooted at S and resulting the string ***w.***
- In other words, if there exist multiple leftmost derivation or multiple rightmost derivation for the same string for any grammar then it is ambiguous.

Ambiguous Grammar

- Let G = (V, ∑, R, S) be a CFG with the production rule of the form:

$$S \rightarrow AB \mid aaB$$

$$A \rightarrow a \mid Aa$$

$$B \rightarrow b$$

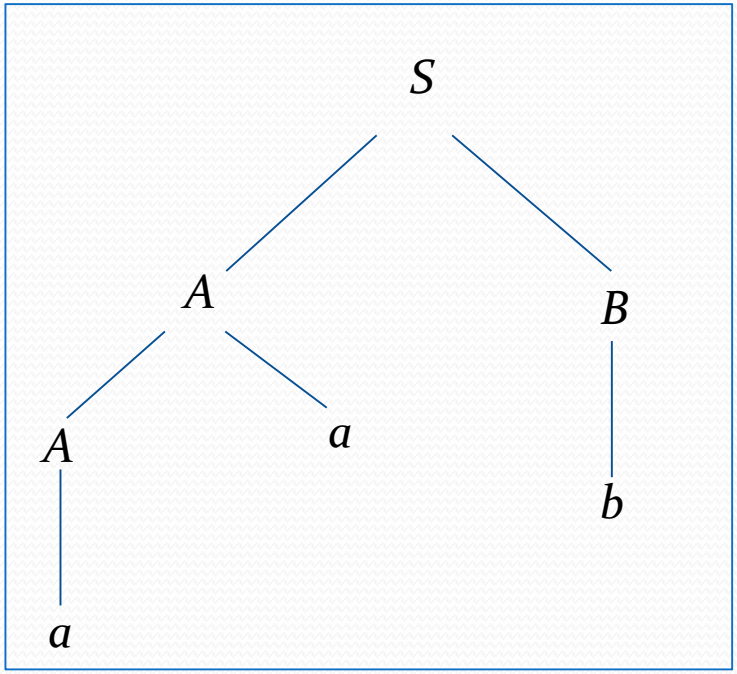
Now, for string ***w=aab*** , we have two left most derivation as:

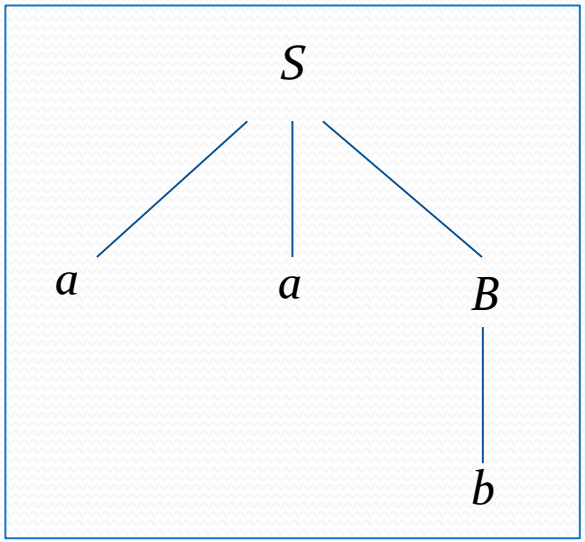
$$S \rightarrow AB \rightarrow AaB \rightarrow aaB \rightarrow aab$$

$$S \rightarrow aaB \rightarrow aab$$

Ambiguous Grammar

- The derivation tree can be:





- Since there are two parse tree for the same string ***w=aab***, this grammar is ambiguous.

Inherently Ambiguous

- Sometimes when we have an ambiguous grammar we can find an unambiguous grammar that generates the same language.
- Some context-free languages, however, can be generated only by ambiguous grammars.
- Such languages are called inherently ambiguous.
- Ambiguity is a property of a grammar, and it is usually, but not always possible to find an equivalent unambiguous grammar.
- An “inherently ambiguous language” is a language for which no unambiguous grammar exists.

Inherently Ambiguous

Assignment……

Explore about the example of Inherently Ambiguous grammar.

Simplification of CFG

- In CFG sometimes all the production rules and symbols used are not needed for the derivation of strings.
- Besides this, there may also be some NULL productions, UNIT productions and USELESS symbols.
- Elimination of these productions and symbols that are not used while derivation is called simplification of CFG.
- Simplification consists of:
  - Removal of Null Productions
  - Removal of Unit Productions
  - Removal of Useless symbol
    - Non generating symbol
    - Not reachable symbol

Removal of Null Production

- The production rule of the form

non-terminal → empty (ε) string

is called as null production.

- To eliminate ***ε*** -production from a CFG, we first find out null production rule of CFG and then remove then from the grammar.
- A non terminal is said to produce null, if it derives an epsilon(***ε***) in zero or more steps.

i.e if $A \rightarrow^* \varepsilon$ then A is null production.

Removal of Null Production

- Steps:
  - Find all the non terminal that contains null production rule in the grammar.
  - If such production rule is found then we replace that non terminal with the empty string and add the resulting production rule to the grammar.
  - After adding new production to the grammar, we discard the null production from the grammar.
  - The process is continued until all the null productions are eliminated.

Removal of Null Production_Example(1)

Consider a grammar:

$$S \rightarrow AB$$

$$A \rightarrow aAA \mid \varepsilon$$

$$B \rightarrow bBB \mid \varepsilon$$

Remove the null productions

Solution,

The null production in the grammar is:

$$A \rightarrow \varepsilon$$

$$B \rightarrow \varepsilon$$

Now we need to find all the non-terminal ***A and B*** that exists in the production rules.

Removal of Null Production_Example(1)

Here, the occurrence of A and B exists in the rule:

$$S \rightarrow AB$$

$$A \rightarrow aAA$$

$$B \rightarrow bBB$$

So, we replace ***A and B*** by ε by all possible ways and add the resulting production rule in the grammar

Let us take the production rule: $S \rightarrow AB$

$$S \rightarrow A \ [when\ B \rightarrow \varepsilon]$$

$$S \rightarrow B \ [when\ A \rightarrow \varepsilon]$$

Removal of Null Production_Example(1)

So the production rule for S can be:

$$S \rightarrow AB \mid A \mid B$$

Again, take the production rule: $A \rightarrow aAA$

$A \rightarrow aA$ *[when one A is replaced by* $A \rightarrow \varepsilon$*]*

$A \rightarrow a$ *[when both A is replaced by* $A \rightarrow \varepsilon$*]*

So the production rule for A can be:

$$A \rightarrow aAA \mid aA \mid a$$

Similarly, take the production rule: $B \rightarrow bBB$

$B \rightarrow bB$ *[when one B is replaced by* $B \rightarrow \varepsilon$*]*

$B \rightarrow b$ *[when both B is replaced by* $B \rightarrow \varepsilon$*]*

So the production rule for A can be:

$$B \rightarrow bBB \mid bB \mid b$$

Removal of Null Production_Example(1)

So the final production rule after removing the null productions are:

$$S \rightarrow AB \mid A \mid B$$

$$A \rightarrow aAA \mid aA \mid a$$

$$B \rightarrow bBB \mid bB \mid b$$

Removal of Null Production_Example(2)

- Consider a grammar:

$$S \rightarrow ABC$$
$$A \rightarrow BB \mid \varepsilon$$
$$B \rightarrow CC \mid a$$
$$C \rightarrow AA \mid b$$

Remove all the null productions.

Solution,

The null production in the grammar is:

$$A \rightarrow \varepsilon$$
$$C \rightarrow AA \ \textit{results in} \ C \rightarrow \varepsilon$$
$$B \rightarrow CC \ \textit{results in} \ B \rightarrow \varepsilon$$

Removal of Null Production_Example(2)

So, we replace ***A, B and C*** by ***ε*** by all possible ways and add the resulting production rule in the grammar

For $S \rightarrow ABC$

$S \rightarrow AB$	$[when\ C \rightarrow \varepsilon\ ]$
$S \rightarrow AC$	$[when\ B \rightarrow \varepsilon\ ]$
$S \rightarrow BC$	$[when\ A \rightarrow \varepsilon\ ]$
$S \rightarrow A$	$[when\ B \rightarrow \varepsilon\ and\ C \rightarrow \varepsilon\ ]$
$S \rightarrow B$	$[when\ A \rightarrow \varepsilon\ and\ C \rightarrow \varepsilon\ ]$
$S \rightarrow C$	$[when\ A \rightarrow \varepsilon\ and\ B \rightarrow \varepsilon\ ]$

After the removal of null production the resulting rules for S are:

$$S \rightarrow ABC \mid AB \mid AC \mid BC \mid A \mid B \mid C$$

Removal of Null Production_Example(2)

For $A \rightarrow BB$

$A \rightarrow B$ $\quad\quad$ $[when\ B \rightarrow \varepsilon\ ]$

So, after the removal of null production the resulting rules for A are:

$A \rightarrow BB \mid B$

For $B \rightarrow CC$

$B \rightarrow C$ $\quad\quad$ $[when\ C \rightarrow \varepsilon\ ]$

So, after the removal of null production the resulting rules for B are:

$B \rightarrow CC \mid C \mid a$

Removal of Null Production_Example(2)

For $C \rightarrow AA$

$C \rightarrow A$ $\quad\quad$ $[when\ A \rightarrow \varepsilon\ ]$

So, after the removal of null production the resulting rules for C are:

$C \rightarrow AA \mid A \mid b$

So the final production rule after removing the null productions are:

$S \rightarrow ABC \mid AB \mid AC \mid BC \mid A \mid B \mid C$

$A \rightarrow BB \mid B$

$B \rightarrow CC \mid C \mid a$

$C \rightarrow AA \mid A \mid b$

Removal of Unit Production

- A production rule of the form:

  Non Terminal → one Non-Terminal

  i.e A → B (where A and B, both are non-terminals) is called unit production.

- Procedure to remove unit production:
  - To remove the production rule of form $A \rightarrow B$, such that there exists a production $B \rightarrow \alpha$, where α is a terminal add $A \rightarrow \alpha$ to the grammar
  - Remove $A \rightarrow B$ from the grammar.
  - Repeat until all unit production are removed.

Removal of Unit Production_Example(1)

Consider a grammar

$S \rightarrow 0A \mid 1B \mid C$

$A \rightarrow 0S \mid 00$

$B \rightarrow 1 \mid A$

$C \rightarrow 01$

Remove the unit production

Solution,

The unit production in the grammar is:

$\boldsymbol{S \rightarrow C}$

$\boldsymbol{B \rightarrow A}$

Removal of Unit Production_Example(1)

To remove these unit production we have to search for a production rule for C and A and add these rule to the grammar.

Here, $C \rightarrow 01$

Now replace the production of $S \rightarrow C$ by $C \rightarrow 01$ and remove $S \rightarrow C$ from the grammar.

$$S \rightarrow 0A \mid 1B \mid 01$$

Again $A \rightarrow 0S \mid 00$

Replace the production rule $B \rightarrow A$ by all the rule of what A gives.

Removal of Unit Production_Example(1)

Replace the production rule $B \rightarrow A$ by all the rule of what A gives and remove $B \rightarrow A$ from grammar .

i.e. $B \rightarrow 1 \mid A$

$B \rightarrow 1 \mid 0S \mid 00$

Thus final CFG after removal of unit production is:

$S \rightarrow 0A \mid 1B \mid 01$

$A \rightarrow 0S \mid 00$

$B \rightarrow 1 \mid 0S \mid 00$

$C \rightarrow 01$

Removal of Unit Production_Example(2)

Consider a grammar

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow C \mid b$

$C \rightarrow D$

$D \rightarrow E$

$E \rightarrow a$

Remove the unit production

Removal of Unit Production_Example(2)

Solution,

The unit production in the grammar is:

$$B \rightarrow C$$

$$C \rightarrow D$$

$$D \rightarrow E$$

To remove these unit production we have to search for a production rule that contains the terminal symbol.

Here, such production rule is $E \rightarrow a$

Now replace the production of $D \rightarrow E$ by $E \rightarrow a$ and remove $D \rightarrow E$ from the grammar.

$$D \rightarrow a$$

$$E \rightarrow a$$

Removal of Unit Production_Example(2)

Again for $C \rightarrow D$

Replace the production of $C \rightarrow D$ by $D \rightarrow a$ and remove $C \rightarrow D$ from the grammar.

$$C \rightarrow a$$
$$D \rightarrow a$$
$$E \rightarrow a$$

Lastly for unit production: $B \rightarrow C$

Replace the production of $B \rightarrow C$ by $C \rightarrow a$ and remove $B \rightarrow C$ from the grammar.

$$B \rightarrow a \mid b$$
$$C \rightarrow a$$
$$D \rightarrow a$$
$$E \rightarrow a$$

Removal of Unit Production_Example(2)

Hence all the unit production has been removed so the final grammar is:

$$S \rightarrow AB$$

$$A \rightarrow a$$

$$B \rightarrow a \mid b$$

$$C \rightarrow a$$

$$D \rightarrow a$$

$$E \rightarrow a$$

Removal of Useless Symbol

- Useless symbols are those variables or terminals used in grammar that can never take part in derivation of any string from the start symbol.
- Formally, we say a symbol ***X*** is useful for a grammar G = (V, ∑, R, S) if there is some derivation of the form:

$$S \rightarrow^* \alpha X \beta \rightarrow^* w, \text{ where w is in } \sum^*.$$

- Here, ***X*** may be either variable or terminal and the sentential form ***α X β*** might be the first or last in the derivation.
- If ***X*** is not useful, we say it is useless.

Removal of Useless Symbol

- Eliminating a useless symbol includes identifying whether or not the symbol is "generating" and "reachable".
- Removal of useless symbol from CFG includes:
  - Removal of non generating symbol
  - Removal of not reachable symbol
- A symbol X is generating if $X \rightarrow^* w$ for some terminal string w.
- A symbol y is reachable if there is derivation $S \rightarrow^* XyZ$ for some X and Z.
- Thus we eliminate non-generating and then non reachable symbols.

Removal of Useless Symbol

- For eliminating ***non generating symbol***, identify the non-generating symbols in CFG and remove those from productions rules.
- For eliminating ***non-reachable symbols*** from CFG, first identify non-reachable symbols and eliminate symbols non-reachable symbols and production rules.

Removal of Useless Symbol_Example(1)

Consider the grammar G

$S \rightarrow aB \mid bX$

$A \rightarrow Bad \mid bSX \mid a$

$B \rightarrow aSB \mid bBX$

$X \rightarrow SBd \mid aBX \mid ad$

Let, S is start symbol.

Remove all useless symbol

Removal of Useless Symbol_Example(1)

Solution,

First identify all the non-generating symbol in the grammar.

Here ***S, A*** and ***X*** are generating symbols but ***B is non-generating*** because there is no way from ***B*** you can reach to any terminal symbol.

So, remove all the rules with ***B***. The new grammar is then:

$$S \rightarrow bX$$

$$A \rightarrow bSx \mid a$$

$$X \rightarrow ad$$

Removal of Useless Symbol_Example(1)

Now remove non-reachable symbol if any from the grammar

Here clearly you can see that we cannot reach to ***A*** from start symbol ***S*** so, ***A is non-reachable.***

So, remove ***A*** from the grammar and the simplified grammar will be:

$$S \rightarrow bX$$

$$X \rightarrow ad$$

Removal of Useless Symbol_Example(2)

Consider the grammar G

$S \rightarrow AB \mid CA$

$A \rightarrow a$

$B \rightarrow BC \mid AB$

$C \rightarrow aB \mid b$

Let, S is start symbol.

Remove all useless symbol

Removal of Useless Symbol_Example(2)

Solution,

First identify all the non-generating symbol in the grammar.

Here ***A***, ***C*** are generating symbols as they can generate terminals. S can also generate terminal since S→ AC but ***B is non-generating*** because there is no way from ***B*** you can reach to any terminal symbol.

So, remove all the rules with ***B***. The new grammar is then:

$$S \rightarrow CA$$

$$A \rightarrow a$$

$$C \rightarrow b$$

Removal of Useless Symbol_Example(2)

Now remove non-reachable symbol if any from the grammar

Here clearly you can see that we can reach to ***A and C*** from start symbol ***S*** so, ***there is no any non-reachable.***

So, the required simplified grammar is:

$$S \rightarrow CA$$

$$A \rightarrow a$$

$$C \rightarrow b$$

Chomsky Normal Form (CNF)

- A context free grammar G (V, ∑, R, S) is said to be in Chomsky's Normal Form (CNF) if every production in the grammar are in one of the two forms:

$$A \rightarrow BC \ \&$$

$$A \rightarrow a$$

where ***A, B, and C*** are the non-terminals and ***a*** is terminal

- A grammar in CNF is one which should not have:
  - ε-production
  - Unit production
  - Useless symbols

Chomsky Normal Form (CNF)

- Procedure to find Equivalent Grammar in CNF
  - ***Step 1***: If the start symbol S occurs on same right side then create a new start symbol ***S′*** and a new production ***S′→ S***.
  - ***Step 2***: Eliminate all the ε-productions, unit productions and useless symbols from grammar if any.
  - ***Step 3:*** If the right side of any production is in of the form ***A→ aB*** where ***'a'*** is a terminal and ***A and B*** are non-terminal then the production is replaced by ***A→XB*** and ***X → a***. Repeat this step for every Production which is of the form ***A→ aB***.

Chomsky Normal Form (CNF)

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

Chomsky Normal Form(CNF)_Example(1)

Convert the following CFG into CNF G = (V, ∑, R, S) where

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

$\sum = \{a, b\}$

R consists of:

$S \rightarrow aAB \mid AaB \mid B$

$A \rightarrow aA \mid \varepsilon$

$B \rightarrow ab \mid bA$

Solution,

To convert CFG into CNF , we have to simplify the given CFG.

Chomsky Normal Form(CNF)_Example(1)

In given grammar there are ε -productions so we remove those productions

Here, **A → ε** is the null production so we replace ***A*** with ***ε*** by all possible ways and add the resulting production rule in the grammar

S → aAB | AaB | aB | B

A → aA | a

B → ab | bA | b

Chomsky Normal Form(CNF)_Example(1)

In given grammar there are unit productions also so we remove those productions from grammar

Here, **S → B** is the unit production.

To remove these unit production we have to search for a production rule for ***B*** and add these rule to the grammar.

Here, ***B → ab | bA | b***

Now replace the production of ***S → B*** by production rules of ***B*** and remove ***S → B*** from the grammar.

S → aAB | AaB | aB | ab | bA | b

A → aA | a

B → ab | bA | b

Chomsky Normal Form(CNF)_Example(1)

$S \rightarrow aAB \mid AaB \mid aB \mid ab \mid bA \mid b$

$A \rightarrow aA \mid a$

$B \rightarrow ab \mid bA \mid b$

There is no any useless symbol so this is the final simplified grammar

In this simplified grammar all the production rules are not in the form of:

$$\boldsymbol{A \rightarrow BC} \ \& \ \boldsymbol{A \rightarrow a}$$

Now, Converting this simplified form to get CNF:

$S \rightarrow CAB \mid ACB \mid CB \mid CD \mid DA \mid b$

$C \rightarrow a$

$D \rightarrow b$

$A \rightarrow CA \mid a$

$B \rightarrow CD \mid DA \mid b$

*We also convert **ab** to **CD** as **ab** is not of the form for CNF*

Chomsky Normal Form(CNF)_Example(1)

Here , again the production rules $\boldsymbol{S \rightarrow CAB}$ and $\boldsymbol{S \rightarrow ACB}$ are not in required form of CNF

we can perform like:

$$S \rightarrow EB \mid FB \mid CB \mid CD \mid DA \mid b$$

$$E \rightarrow CA$$

$$F \rightarrow AC$$

$$C \rightarrow a$$

$$D \rightarrow b$$

$$A \rightarrow CA \mid a$$

$$B \rightarrow CD \mid DA \mid b$$

This the required CNF of the given CFG

Chomsky Normal Form(CNF)_Example(2)

Convert the following CFG into CNF G = (V, ∑, R, S) where

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

$\sum = \{a, b\}$

R consists of:

$S \rightarrow AAC$

$A \rightarrow aAb \mid \varepsilon$

$C \rightarrow aC \mid a$

Solution,

To convert CFG into CNF , we have to simplify the given CFG.

In given grammar there are ε -productions so we remove those productions

Chomsky Normal Form(CNF)_Example(2)

Here, $A \rightarrow \varepsilon$ is the null production so we replace A with ε by all possible ways and add the resulting production rule in the grammar

$$S \rightarrow AAC \mid AC \mid C$$
$$A \rightarrow aAb \mid ab$$
$$C \rightarrow aC \mid a$$

In given grammar there are unit productions also so we remove those productions from grammar

Here, $S \rightarrow C$ is the unit production.

To remove these unit production we have to search for a production rule for C and add these rule to the grammar.

Here, $C \rightarrow aC \mid a$

Chomsky Normal Form(CNF)_Example(2)

Now replace the production of $S \rightarrow C$ by production rules of C and remove $S \rightarrow C$ from the grammar.

$$S \rightarrow AAC \mid AC \mid aC \mid a$$

$$A \rightarrow aAb \mid ab$$

$$C \rightarrow aC \mid a$$

There is no any useless symbol so this is the final simplified grammar

In this simplified grammar all the production rules are not in the form of:

$$A \rightarrow BC \ \& \ A \rightarrow a$$

Chomsky Normal Form(CNF)_Example(2)

Now, Converting this simplified form to get CNF:

$$S \rightarrow AAC \mid AC \mid aC \mid a$$

$$A \rightarrow aAb \mid ab$$

$$C \rightarrow aC \mid a$$

These can be written as:

$$S \rightarrow AAC \mid AC \mid DC \mid a$$

$$D \rightarrow a$$

$$A \rightarrow DAE \mid DE$$

$$E \rightarrow b$$

$$C \rightarrow DC \mid a$$

Chomsky Normal Form(CNF)_Example(2)

Here , again the production rules $S \rightarrow AAC$ and $A \rightarrow DAE$ are not in required form of CNF

We can perform like:

These can be written as:

$$S \rightarrow AF \mid AC \mid DC \mid a$$

$$F \rightarrow AC$$

$$D \rightarrow a$$

$$A \rightarrow DG \mid DE$$

$$G \rightarrow AE$$

$$E \rightarrow b$$

$$C \rightarrow DC \mid a$$

Greibach Normal Form (GNF)

- A context free grammar G=(V, ∑, R, S) is said to be in Greibach Normal Form (GNF) if every production in the grammar is in the form:

$$\alpha \rightarrow a\beta^*$$

where α ***and*** $\beta \in V$ *and* $a \in \sum$.

i.e. ***Non-Terminal →exactly one Terminal***

OR

Non-Terminal → exactly one Terminal followed by any number of Non-Terminal

Regular Grammar

- A grammar is said to be regular if it has the production rule of the form:

$$A \rightarrow aB$$

$$A \rightarrow a$$

$$A \rightarrow \varepsilon$$

where $A, B \in V$ *and* $a \in \sum$.

i.e. ***Non-Terminal →exactly one Terminal***

OR

Non-Terminal → exactly one Terminal followed by exactly one Non-Terminal

- A regular grammar may be left linear or right linear.

Linear Grammar

- A linear grammar is also a context-free grammar that has at most one non-terminal symbol on the right hand side of each grammar rule.
- A rule may have just one terminal symbols on the right hand side (zero non-terminals) .

E.g.

$$S \rightarrow aA$$

$$A \rightarrow aBb \mid a \mid \varepsilon$$

$$B \rightarrow Bb$$

Left and Right Linear Grammar

- A left linear grammar is a linear grammar in which the non-terminal symbol always occurs on the left side.

E.g.

$$S \rightarrow Aa$$
$$A \rightarrow ab$$

- A right linear grammar is a linear grammar in which the non-terminal symbol always occurs on the right side.

E.g.

$$S \rightarrow abaA$$
$$A \rightarrow \varepsilon$$

Meaning of Context Free

- Let us consider a CFG as

$G=(V, \Sigma, R, S)$

where, $V = \{P, M, A, B, a, b\}$

$\Sigma = \{a, b\}$

R consists of:

$P \rightarrow aMb$

$M \rightarrow A$

$M \rightarrow B$

$A \rightarrow aA \mid \varepsilon$

$B \rightarrow bB \mid \varepsilon$

S = P is the start symbol

Meaning of Context Free

- 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 surrounds 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***.

Meaning of Context Free

- When there is a production of form:

  $$\boldsymbol{aaAb} \rightarrow \boldsymbol{aaBb}$$ (but not of the form A→ B),

- This grammar is context sensitive since A can be replaced by B only when it is surrounded by the strings “aa" and “b“.
- Such grammar i.e. Context Sensitive and is even powerful than the CFG in term of generating wide variety of string.

Pumping Lemma for CFG

- ***Theorem:*** Let A be a context free language. Then there exists an integer n, called the pumping length, such that any string w in A, with $|w| \geq n$, can be decomposed into five parts as $w = uvxyz$, such that:
  - $for\ all\ i \geq 0,\ uv^ixy^iz \in A$
  - $|vy| \geq 1,\ y \notin \varepsilon$
  - $|vxy| \leq n$
- In words, the pumping lemma states that by replacing the portion v ***and*** y in w by zero or more copies of it, the resulting string is still in the language A.

Pumping Lemma for CFG_Example(1)

- Show that $L = \{a^n b^n c^n : n>=1\}$ is not CFL using pumping lemma.

Solution,

Let L be a CFL and w be any string in L, where $w \in L$ and p be pumping constant with $|w| >= p$.

Now w can be written as $w = uvxyz = a^p b^p c^p$ such that:

- $for\ all\ i \geq 0,\ uv^i xy^i z \in L$
- $|vxy| <= p$ and
- $|vy| > 0$

Pumping Lemma for CFG_Example(1)

We prove this by contradiction,

Let us assume that given language L is context free.

If it is context free then we will have pumping length (say p). Then our string ***s*** $=a^pb^pc^p$.

Let we take p=4 then, $\boldsymbol{s=a^4b^4c^4}$. Now we divide ***s*** into five parts ***uvxyz***.

Dividing the string into ***uvxyz*** can be done in two ways:

- *v and y contains only one type of symbol.*
- *either v or y have more than one type of symbol*

Pumping Lemma for CFG_Example(1)

Case-I : v and y contains only one type of symbol.

i.e. s=**aaaabbbbcccc** [u=**a**, v=**b**, x=**bb**, y=**b**, z=**cc**]

For this let us check the condition :

for all $i \geq 0$, $uv^ixy^iz \in L$, *with* $i=2$

s=aaaabbbbbbbbcccc

We get s=$\mathbf{a^4b^8c^4} \notin L$.

Case-II : either v or y have more than one type of symbol

i.e. s=**aaaabbbbbcccc** [u=**aa**, v=**aa**, x=**b**, y=**b**, z=**bbcccc**]

Pumping Lemma for CFG_Example(1)

for all $i \geq 0$, $uv^ixy^iz \in L$, *with* $i=2$

s=aaaaaabbbbbbcccc

We get s=$a^6b^5c^4 \notin$ L.

*So, s=***aaaabbaabbbbcccc** $\notin$ L

This concludes that ***s*** cannot be pumped which leads to the contradiction.

Hence the given language is not CFL.

Pumping Lemma for CFG_Example(2)

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

Solution,

Let L be a CFL and w be any string in L, where $w \in L$ and p be pumping constant with $|w| >= p$.

Now w can be written as $w = uvxyz = a^p b^p c^p$ such that:

- $for\ all\ i \geq 0,\ uv^i xy^i z \in L$
- $|vxy| <= p$ and
- $|vy| > 0$

Pumping Lemma for CFG_Example(2)

We prove this by contradiction,

Let us assume that given language L is context free.

If it is context free then we will have pumping length (say p). Then our string ***s*** $=a^pb^pa^pb^p$.

Let we take p=5 then, ***s*** $=a^5b^5a^5b^5$. Now we divide ***s*** into five parts ***uvxyz***.

Dividing the string into ***uvxyz*** can be done in two ways:

- *vxy does not straddle a boundary.*
- *vxy straddles the first boundary*

Straddle means it does not lie on the either side of boundary but lies in one part only without crossing boundary

Pumping Lemma for CFG_Example(2)

Case-I : vxy does not straddle a boundary.

i.e. s=**aaaaabbbbbaaaaabbbbb** [*i.e.* **uvxyz**]

For this let us check the condition :

for all $i \geq 0$, $uv^ixy^iz \in L$, *with* $i=2$

s=**aaaaabbbbbbbaaaaabbbbb**

We get s=$\mathbf{a^5b^7a^5b^5} \notin L$.

Pumping Lemma for CFG_Example(2)

Case-II : vxy straddles the first boundary

i.e. s=**aaaaabbbbbbaaaaaabbbbb** [*i.e.* **uvxyz**]

For this let us check the condition :

for all $i \geq 0$, $uv^ixy^iz \in L$, *with* $i=2$

i.e. s=**aaaaaaaabbbbbbbaaaaaabbbbb** [*i.e.* **uvxyz**]

We get s=$\mathbf{a^7b^7a^5b^5} \notin L$.

Pumping Lemma for CFG_Example(2)

Case-III : vxy straddles the third boundary

i.e. s=**aaaaabbbbbbaaaaabbbbbb** [*i.e.* **uvxyz**]

For this let us check the condition :

for all $i \geq 0$, $uv^ixy^iz \in L$, *with* $i=2$

i.e. s= s=**aaaaabbbbbbaaaaaaaabbbbbbbb** [*i.e.* **uvxyz**]

We get s=$\mathbf{a^5b^5a^7b^7} \notin L$.

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

Closure Properties of CFG

- Context Free Languages are closed under
  - Union
  - Concatenation
  - Kleene Star
- That is we can establish a new rule in the existing grammar which can generate the string similar to the existing grammar.
- To prove the closure properties we need to redefine the four tuples of the grammar

Closure Properties of CFG (Union)

- Let $G_1 = (V_1, \sum_1, R_1, S_1)$ and $G_2 = (V_2, \sum_2, R_2, S_2)$ be two context free grammars.
- Let us consider that they have disjoint set of non terminals.
- To show the union of two CFG is also a CFG we need to define the new start symbol say S which is not in G_1 and G_2
- Then construct a new grammar $G = (V, \sum, R, S)$ where ,

$$\boldsymbol{V = V_1 \cup V_2 \cup \{S\}}$$

$$\sum = \sum_1 \cup \sum_2$$

$$\boldsymbol{R = R_1 \cup R_2 \cup \{S \rightarrow S_1 \mid S_2\}}$$

- Here G is clearly a context free grammar because two new rules added are also of correct form.

Closure Properties of CFG (Union)

- Now , we can claim that $L(G) = L(G1) \cup L(G2)$
- It is because of the rule $S \rightarrow S1 \mid S2$.
- If we want to generate the string as per the rule of G1 we can use the rule S→S1 and using the production rule of R1 it can generate the string that belongs to L(G1).
- Again if we want to generate the string as per the rule of G2 we can use the rule S→S2 and using the production rule of R2 it can generate the string that belongs to L(G2).
- So, context free languages are closed under union.

Closure Properties of CFG (Concatenation)

- Let $G_1 = (V_1, \sum_1, R_1, S_1)$ and $G_2 = (V_2, \sum_2, R_2, S_2)$ be two context free grammars.
- Let us consider that they have disjoint set of non terminals.
- To show the concatenation of two CFG is also a CFG we need to define the new start symbol say S which is not in G_1 and G_2
- Then construct a new grammar $G = (V, \sum, R, S)$ where ,

$$V = V_1 \cup V_2 \cup \{S\}$$

$$\sum = \sum_1 \cup \sum_2$$

$$R = R_1 \cup R_2 \cup \{S \rightarrow S_1 S_2\}$$

- Here G is clearly a context free grammar because new rule added is also of correct form.

Closure Properties of CFG (Concatenation)

- Here, if $S_1 \rightarrow^* w_1$ and $S_2 \rightarrow^* w_2$ then we can claim $S \rightarrow^* w_1w_2$ as $S \rightarrow S_1S_2$
- Now , we can claim that $L(G) = L(G_1).L(G_2)$
- If we want to generate the string w_1w_2 it is possible because of rule $S \rightarrow S_1S_2$ which can generate the string that belongs to L(G1) and L(G2).
- The symbol S_1 generates the string that belongs to $L(G_1)$ and S_2 generates the string that belongs to $L(G_2)$.
- So, context free languages are closed under concatenation.

Closure Properties of CFG (Kleene Star)

- Let $G_1 = (V_1, \sum_1, R_1, S_1)$ be a context free grammar.
- To show the kleen star of CFG is also a CFG we need to define the new start symbol say S which is not in G_1.
- Then construct a new grammar $G = (V, \sum, R, S)$ where,

$$\boldsymbol{V = V_1 \cup \{S\}}$$

$$\sum = \sum_1$$

$$\boldsymbol{R = R_1 \cup \{S \rightarrow SS_1 \text{ and } S \rightarrow \varepsilon\}}$$

- Here G is clearly a context free grammar because new rule added is also of correct form.

Closure Properties of CFG (Kleene Star)

- Here, if $S_1 \rightarrow^* w_1$ and we can claim $S \rightarrow^* w_1$ as $S \rightarrow SS_1$ and $S \rightarrow \varepsilon$
- Now , we can claim that $L(G) = L(G_1)$
- If we want to generate the string w_1 we can repeatedly use $S \rightarrow SS_1$ as per our need and after that we can use the rule R1 to generate w_1.
- The symbol S_1 generates the string that belongs to $L(G_1)$.
- So, context free languages are closed under kleen star.

Closure Properties of CFG

- The Context Free Language are not closed under ***Intersection*** and ***Complement*.**
- This property of CFL can be shown by using the Pumping Lemma.
- We know that L1 = $\{0^n1^n2^n \mid n >= 1\}$ is not a CFL (use the Pumping Lemma).
- However, L2 = $\{0^n1^n2^i \mid n >= 1, i >= 1\}$ is a CFL, and its CFG is:

$S \rightarrow AB$

$A \rightarrow 0A1 \mid 01$

$B \rightarrow 2B \mid 2$

Closure Properties of CFG

- So is $L_3 = \{0^i 1^n 2^n \mid n >= 1, i >= 1\}$ is a CFL, and its CFG is:

$S \rightarrow AB$

$A \rightarrow 0A \mid 0$

$B \rightarrow 1B2 \mid 12$

But $L_1 = L_2 \cap L_3$ is NOT a CFL.

So, context free languages are not closed under intersection.

Closure Properties of CFG

- ***CFLs are NOT closed under Complement.***
- If L is a CFL, its complement $\overline{L}$

We know $L1 \cap L2 = \overline{\overline{L1} + \overline{L2}}$

Also CFLs are closed under union, it would follow that the CFLs are closed under intersection.

However we know CFL are not closed under intersection.

Hence CFL are not closed under complement.