UNIT-IV – CONTEXT FREE LANGUAGES

Context-free languages are applied in parser design i.e. in the design of compiler / interpreter that breaks data into smaller components.

They are also useful for describing block structures in programming languages. It is easy to visualize derivations in CFL as derivations can be represented using tree-structures.

(CFL Production : $A \longrightarrow \alpha$; $\alpha \in (V_N \cup \Sigma)^*$)
$A \in V_N$

(Regular languages - used in pattern recognition)

DERIVATION TREES

A derivation (also called a Parse tree) for a CFG $G = (V_N, \Sigma, P, S)$ is a tree satisfying the following cond^{n}s :-

(i) Every vertex has a label which is a variable or terminal or Λ.

(ii) The root has label S.

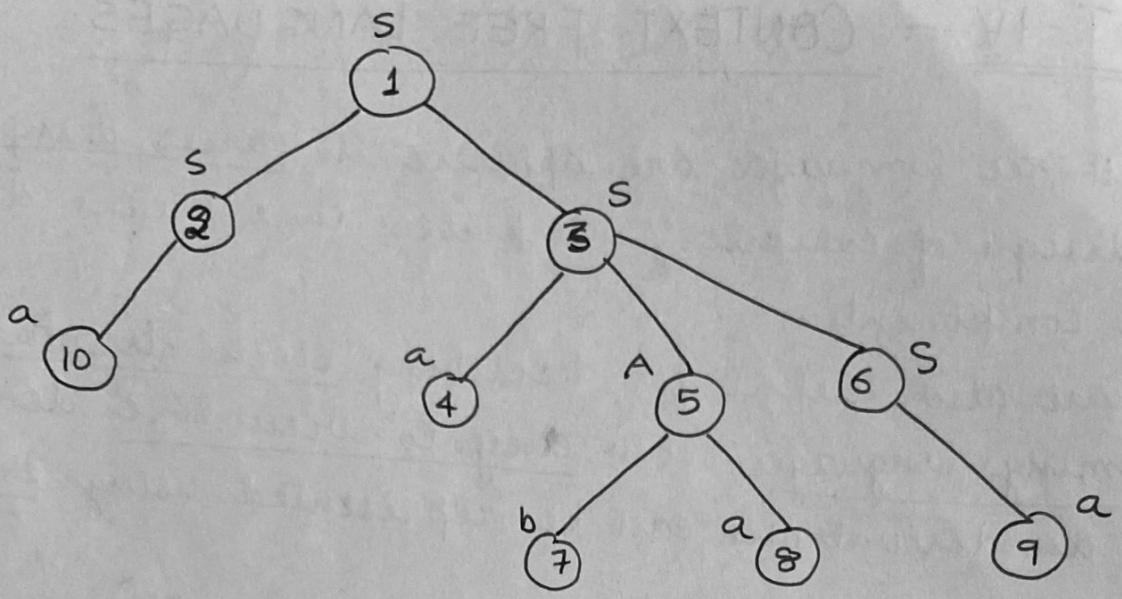
(iii) The label of an internal vertex is a variable.

(iv) If the vertices $n_1, n_2, \ldots, n_k$ written with labels $X_1, X_2, \ldots, X_k$ are the sons of vertex n with label A, then $A \longrightarrow X_1 X_2 \ldots X_k$ is a production in P.

(v) A vertex n is a leaf if its label is $a \in \Sigma$ or Λ; n is the only son of its father if its label is Λ.

example :- $G = (\{S, A\}, \{a, b\}, \{S \rightarrow aAS | a | SS; A \rightarrow SbA | ba\}, S)$

Derivation tree for G is given by :-



ORDERING OF LEAVES FROM LEFT TO RIGHT

We can order all the vertices of a tree in the following way. The successor of the root (i.e. sons of the root) are ordered from the left by the def^{n} of tree. So vertices at level 1 are ordered from left to right. If v_1 and v_2 are any 2 vertices at level 1 & v_1 is to the left of v_2, then we say that v_1 is to the left of any son of v_2. Also, any son of v_1 is to the left of v_2 & to the left of any son of v_2. Thus, we get a left to right ordering of vertices at level 2.

Repeating the process upto level k, where k = height of the tree, we have an ordering of all vertices from left

eg :- In tree above :-

Son of 2 namely 10 is to the left of sons of 3 namely 4, 5 and 6 (these are ordered from left 4-5-6). ∴ the vertices at level-2 in the left-to-right ordering are 10-4-5-6.

The order of leaves are 10-4-7-8-9.

YIELD OF A DERIVATION TREE

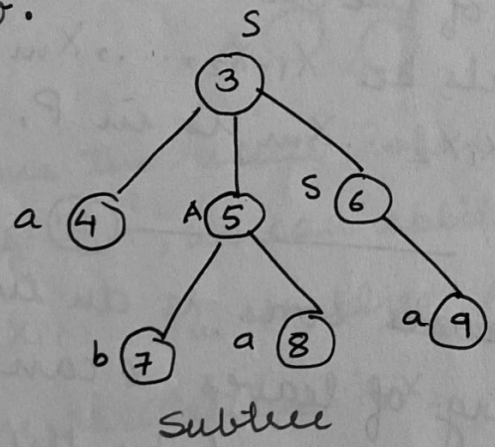
The yield of a derivation tree is the concatenation of the labels of the leaves without repetition in the left-to-right ordering.

eg :- Yield of tree in last figure is aabaa.

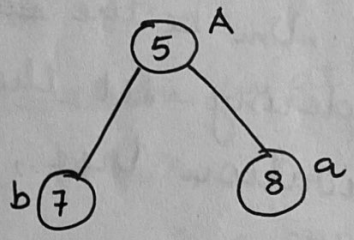
Defⁿ :- SUBTREE OF A DERIVATION TREE

A subtree of a derivation tree T is a tree
(i) whose root is some vertex v of T,
(ii) whose vertices are the descendants of v together with their labels, and
(iii) whose edges are those connecting the descendants of v.

eg :-



Subtree



Subtree

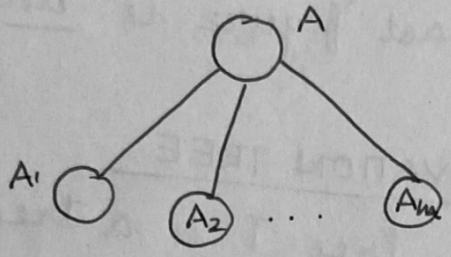
Note :- A subtree is a derivation tree except that the label of the root may not be S. It is called A-tree, if its root label is A.

THEOREM :- Let $G = (V_N, \Sigma, P, S)$ be a CFG. Then $S \overset{*}{\Rightarrow} \alpha$ iff there is a derivation tree for G with yield α.

PROOF :- We prove that $A \overset{*}{\Rightarrow} \alpha$ iff there is an A-tree with yield α. Once this is proved, the theorem follows by assuming $A = S$.

Let α be the yield of an A-tree T. We prove $A \overset{*}{\Rightarrow} \alpha$ by induction on the # of vertices in T.

When tree has only one internal vertex, the remaining vertices are leaves & are the sons of the root.



By def^n. of derivation tree $A \longrightarrow A_1 A_2 \ldots A_m = \alpha$ is a production in G i.e. $A \Rightarrow \alpha$. Thus, there is a basis for induction. Next, we assume the result holds $\forall$ trees with at most $(k-1)$ internal vertices. $(k \geq 1)$.

Let T be an A-tree with k internal vertices $(k \geq 2)$. Let $v_1, v_2, \ldots, v_m$ be the sons of the root in the left-to-right ordering. Let their labels be $X_1, X_2, \ldots, X_m$. By def^n of derivation tree, $A \longrightarrow X_1 X_2 \ldots X_m$ is in P, & so

$$A \Rightarrow X_1 X_2 \ldots X_m \qquad \text{---------- (1)}$$

As $k \geq 2$, at least one of the sons is an internal vertex. By left-to-right ordering of leaves, α can be written as $\alpha_1 \alpha_2 \ldots \alpha_m$, where α_i is obtained by the concatenation of the labels of the leaves which are descendants of vertex v_i. If v_i is an internal vertex, consider the subtree of T with v_i as its root. The # of internal vertices of this subtree $< k$ (as T has k internal vertices & atleast one of them, viz. its root, is not in the subtree). So by induction hypothesis applied to subtree, $X_i \overset{*}{\Rightarrow} \alpha_i$. If v_i is not an internal vertex i.e., a leaf, then $X_i = \alpha_i$.

using ① , we get

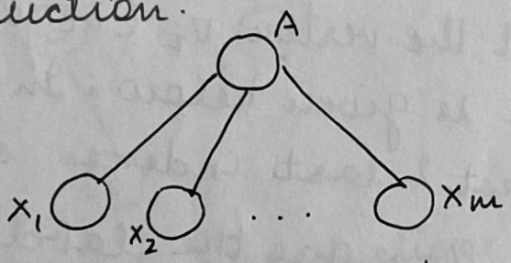
$$A \Rightarrow X_1 X_2 \dots X_m \overset{*}{\Rightarrow} \alpha_1 X_2 X_3 \dots X_m \dots \overset{*}{\Rightarrow} \alpha_1 \alpha_2 \dots \alpha_m = \alpha .$$

i.e. $A \overset{*}{\Rightarrow} \alpha$.

By the principle of induction $A \overset{*}{\Rightarrow} \alpha$ whenever α is the yield of an A-tree.

To prove the <u>only if</u> part, let us assume $A \overset{*}{\Rightarrow} \alpha$. We have to construct an A-tree whose yield is α. We do this by induction on # of steps in $A \overset{*}{\Rightarrow} \alpha$.

When $A \Rightarrow \alpha$, $A \to \alpha$ is a production in P. If $\alpha = X_1 X_2 \dots X_m$, the A-tree with yield α is given below. So, there is a basis for induction.



Assume the result holds for derivation in at most k-steps. Let $A \overset{k}{\Rightarrow} \alpha$, we can split this as $A \Rightarrow X_1 \dots X_m \overset{k-1}{\Rightarrow} \alpha$. Now, $A \Rightarrow X_1 X_2 \dots X_m$ implies $A \to X_1 X_2 \dots X_m$ is a production in P. In the derivation $X_1 X_2 \dots X_m \overset{k-1}{\Rightarrow} \alpha$, either

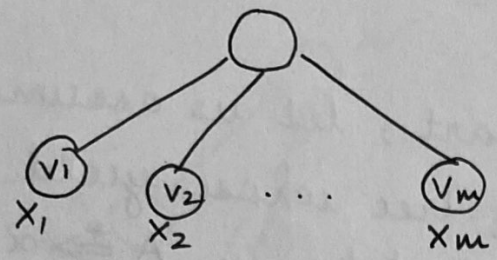
(i) X_i is not changed throughout the derivation, or

(ii) X_i is changed in some subsequent step.

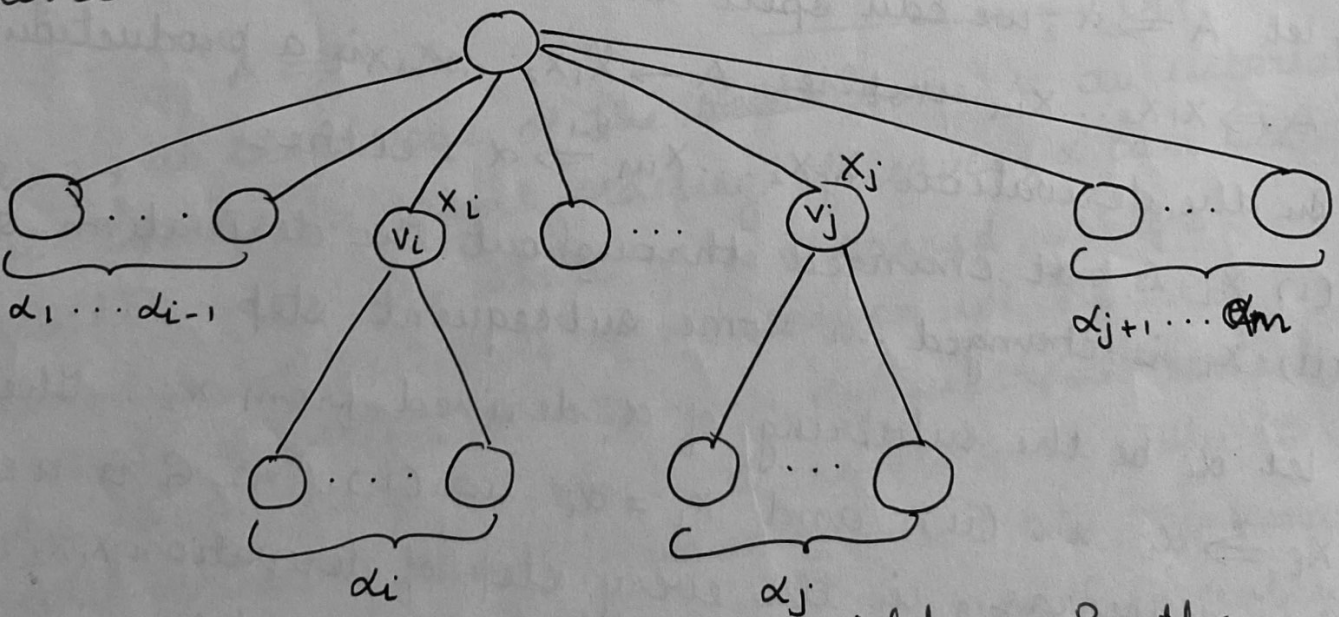
Let α_i be the substring of α derived from X_i. Then, $X_i \overset{*}{\Rightarrow} \alpha_i$ in (ii) and $X_i = \alpha_i$ in (i). As G is a content free grammar, in the every step of derivation $X_1 X_2 \dots X_m \overset{*}{\Rightarrow} \alpha$, we replace a single variable by a string. As $\alpha_1, \alpha_2, \dots, \alpha_m$, account for all the symbols in α, we have

$$\alpha = \alpha_1 \alpha_2 \dots \alpha_m .$$

We construct the derivation tree with yield α as follows. As $A \to X_1 X_2 \cdots X_m$ is in P, we construct a tree with m leaves whose labels are $X_1, X_2, \ldots, X_m$ in the left-to-right ordering. The tree is given below.



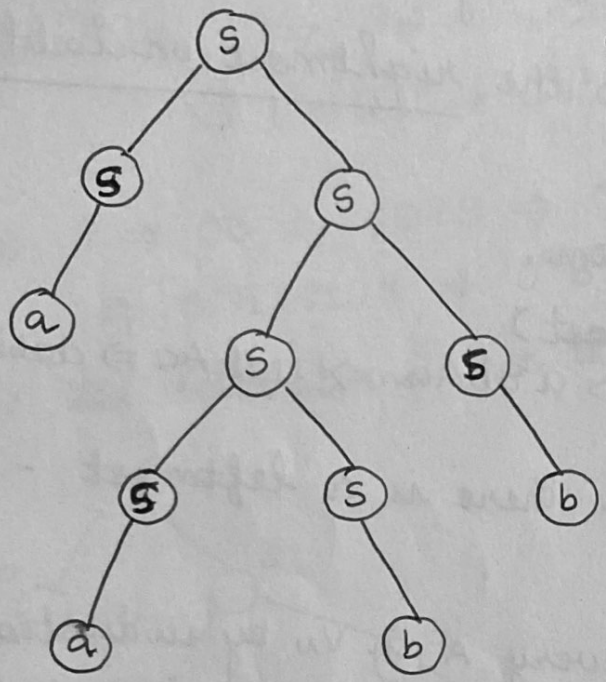
In (i) above, we leave the vertex v_i as it is. In (ii), $X_i \overset{*}{\Rightarrow} \alpha_i$ is less than k-steps (as $X_1 X_2 \cdots X_m \overset{k-1}{\Rightarrow} \alpha$). By induction hypothesis, $\exists$ an X_i-tree T_i with yield α_i. We attach the tree T_i at the vertex v_i (i.e. v_i is the root of T_i). The resulting tree is given below: In this figure, let i and j be the first & last indexes s.t. X_i & X_j satisfy (ii). So $\alpha_1 \cdots \alpha_{i-1}$ are the labels of leaves at level 1 in T. α_i is the yield of X_i-tree T_i, etc.



Thus, we get a derivation tree with yield α. By the principle of induction we get the result for any derivation.

:- The derivation tree does not specify the order in which the production are applied for getting α. So, the same derivation tree can induce several derivations.

eg :- $G = (\{S\}, \{a, b\}, \{S \to aS \mid SS \mid a \mid b, S\})$



$S \Rightarrow SS \Rightarrow SSS \Rightarrow aSS \cdots$

$\Downarrow$

$SSSb$

$\vdots$

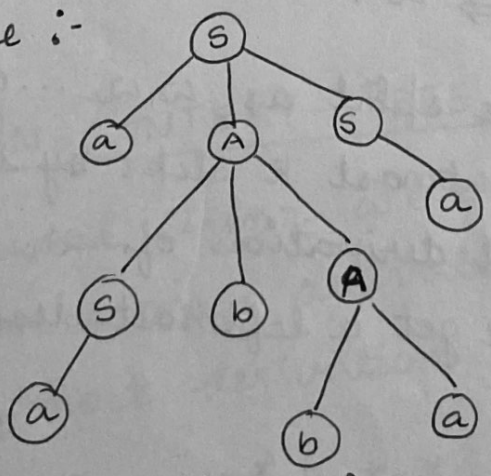
Ques :- Consider G whose productions are $S \to aAS \mid a$, $A \to SbA \mid SS \mid ba$ Show that $S \overset{*}{\Rightarrow} aabbaa$ & construct a derivation tree whose yield is aabbaa.

Soln :- $S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aabAS \Rightarrow aabbaa$

$= a^2b^2a^2$ ——— (☆)

Hence, $S \overset{*}{\Rightarrow} a^2b^2a^2$.

Derivation tree :-



Yield $= a^2b^2a^2$

∃ other derivations of $a^2b^2a^2$

ex :- $S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aSbAa \Rightarrow aabbaa$

LEFTMOST & RIGHTMOST DERIVATION

A derivation $A \overset{*}{\Rightarrow} w$ is called a leftmost derivation if we apply production only to the leftmost variable at every step.

A derivation $A \overset{*}{\Rightarrow} w$ is called a rightmost derivation if we apply production only to the rightmost variable at every step.

eg :- (☆) is a leftmost derivation.

(Neither leftmost nor rightmost)

$S \Rightarrow aAS \Rightarrow$ ~~aAaaaaa~~ $aSbAS \Rightarrow aSbAa \Rightarrow aabAa \Rightarrow aabbaa$

Theorem :- If $A \overset{*}{\Rightarrow} w$ in G, then there is a leftmost derivation of w.

Proof :- We prove the result for every A in V_N by induction on the # of steps in $A \overset{*}{\Rightarrow} w$. $A \Rightarrow w$ is a leftmost derivation as the LHS has only one variable. So there is a basis for induction. let us assume the result for derivations in atmost k-steps. Let $A \overset{k+1}{\Rightarrow} w$. The derivation can be split as :-

$$A \Rightarrow X_1 X_2 \dots X_m \overset{k}{\Rightarrow} w.$$

The string w can be split as $w_1 w_2 \dots w_n$ s.t. $X_i \overset{*}{\Rightarrow} w_i$

As $X_i \overset{*}{\Rightarrow} w_i$ involves atmost k steps by induction hypothesis we can find a leftmost derivation of w_i. Using these leftmost derivations, we get a leftmost derivation of w given by :-

$$A \Rightarrow X_1 X_2 \dots X_m \overset{*}{\Rightarrow} w_1 X_2 \dots X_m \overset{*}{\Rightarrow} w_1 w_2 X_3 \dots X_m \Rightarrow \dots \overset{*}{\Rightarrow} w_1 w_2 \dots w_n$$

Hence, by induction result is true.

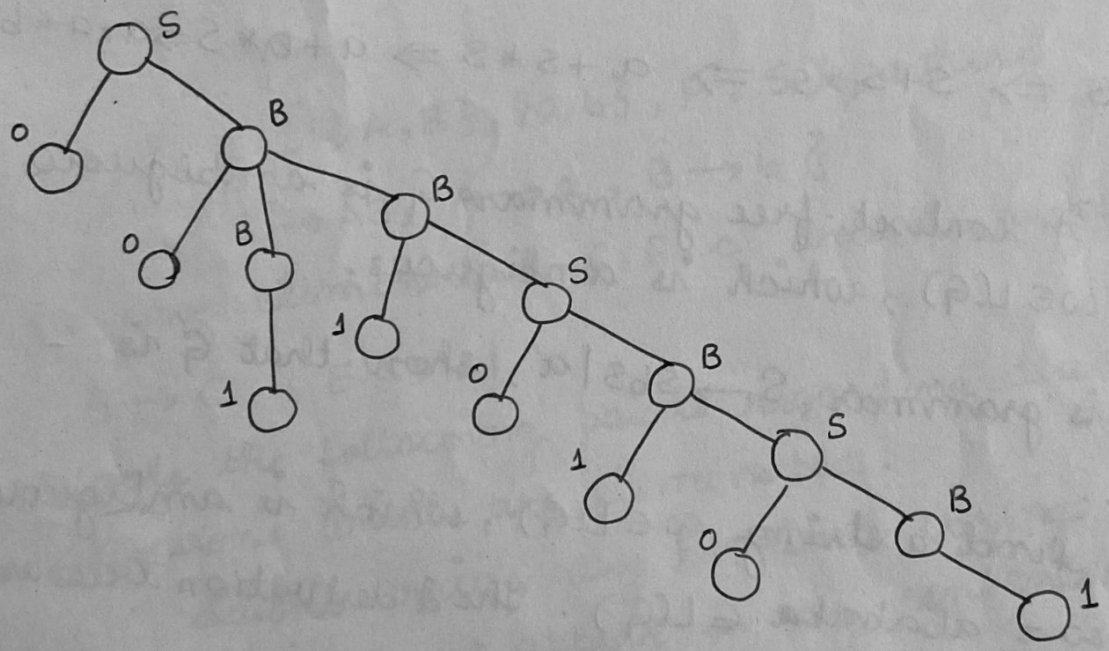
Ex :- Let G be the grammar $S \to 0B \mid 1A$, $A \to 0 \mid 0S \mid 1AA$, (5)
$B \to 1 \mid 1S \mid 0BB$. For the string 00110101, find

(a) the leftmost derivation.
(b) the rightmost derivation.
(c) the derivation tree.

Soln :- (a) $S \Rightarrow 0B \Rightarrow 00BB \Rightarrow 001B \Rightarrow 0011S \Rightarrow 00110B$
$\Rightarrow 0^2 1^2 01S \Rightarrow 0^2 1^2 010B \Rightarrow 0^2 1^2 0101$

(b) $S \Rightarrow 0B \Rightarrow 00BB \Rightarrow 00B1S \Rightarrow 00B10B \Rightarrow 0^2B101S$
$\Rightarrow 0^2B1010B \Rightarrow 0^2B10101 \Rightarrow 0^2110101.$

(c) The derivation tree is :-

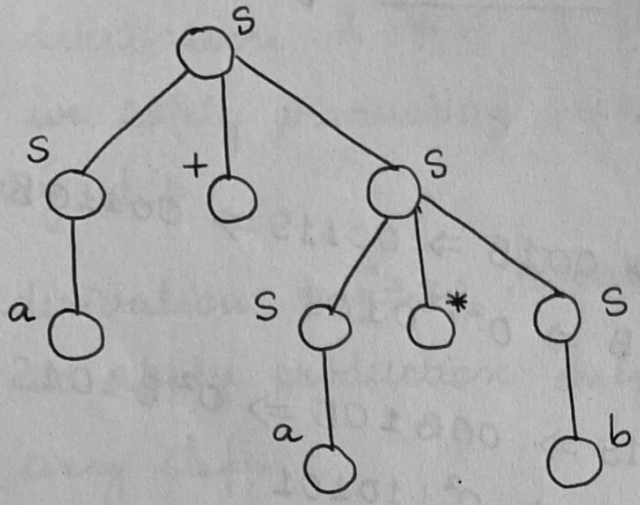


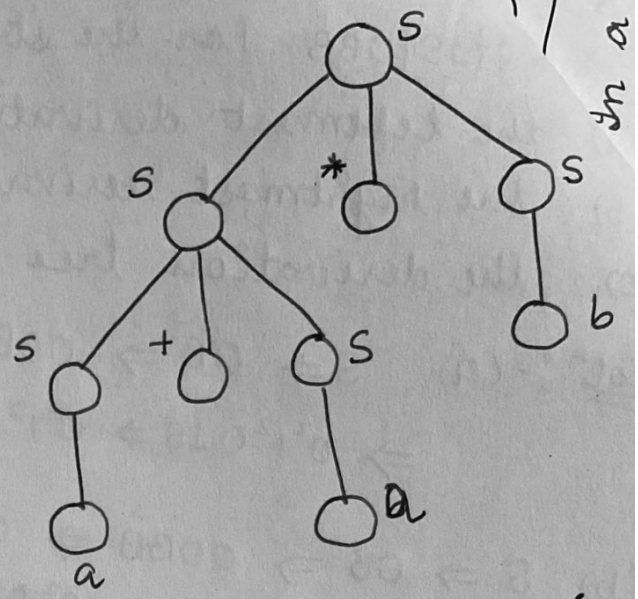
AMBIGUITY IN CONTEXT-FREE GRAMMARS

Defn :- A terminal string $w \in L(G)$ is ambiguous if there exist two or more derivation trees for w (or $\exists$ two or more leftmost derivations of w).

example : $G = (\{S\}, \{a, b, +, *\}, P, S)$ where
$P = \{ S \to S+S \mid S*S \mid a \mid b \}$

$\omega = a+a*b$





The leftmost derivations induced by 2 derivation trees:

i) $S \Rightarrow S+S \Rightarrow a+S \Rightarrow a+S*S \Rightarrow a+a*S \Rightarrow a+a*b$

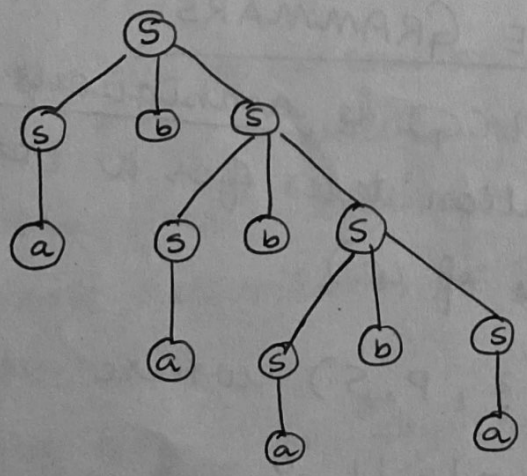
ii) $S \Rightarrow S*S \Rightarrow S+S*S \Rightarrow a+S*S \Rightarrow a+a*S \Rightarrow a+a*b.$

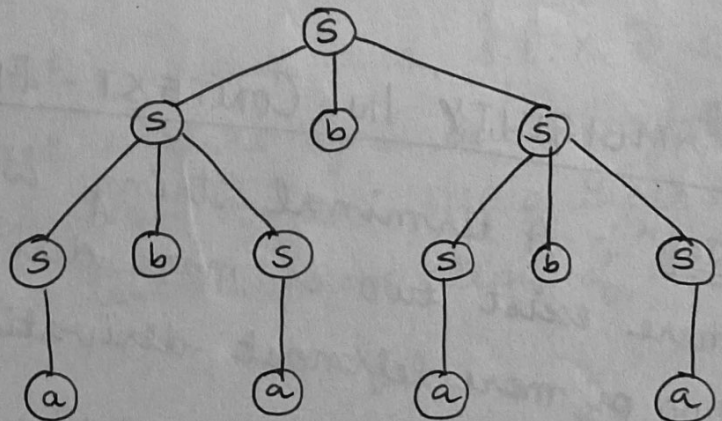
Definition:- A context-free grammar G is ambiguous if $\exists$ some $\omega \in L(G)$, which is ambiguous.

Ques:- If G is grammar $S \rightarrow SbS \mid a$, show that G is ambiguous.

We need to find a string $\omega \in L(G)$, which is ambiguous.

Consider, ω = abababa $\in L(G)$. The 2 derivation trees are:-





In a CFG G, it may not be necessary to use all the symbols in $V_N \cup \Sigma$, or all the productions in P for deriving sentences.

So, when we study CFL L(G), we try to eliminate those symbols & productions in G which are not useful for the derivation of sentences.

example:- $G = (\{S, A, B, C, E\}, \{a, b, c\}, P, S)$

where

$P = \{S \to AB, A \to a, B \to b, B \to C, E \to c \mid \wedge\}$

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

Let $G' = (\{S, A, B\}, \{a, b\}, P', S)$, where

$P' = \{S \to AB, A \to a, B \to b\}$

We have eliminated C, E, c and the productions;

$B \to C$, $E \to c \mid \wedge$

We note the following points regarding the symbols & productions which are eliminated.

(i) C doesn't arrive any terminal string.

(ii) E and c do not appear in any sentential form.

(iii) $E \to \wedge$ is a null production.

(iv) $B \to C$ simply replaces B by C.

CONSTRUCTION OF REDUCED GRAMMARS

Theorem: If G is a CFG s.t. $L(G) \neq \phi$, we can find an equivalent grammar G' s.t. each variable in G' derives some terminal string.

Proof: Construction of $G' = (V_N', \Sigma, P', S)$ from $G = (V_N, \Sigma, P, S)$ is as follows:

(a) **Construction of V_N'**

We define $W_i \subseteq V_N$ by recursion:

$W_1 = \{ A \in V_N \mid \exists$ a production $A \to \omega$ where $\omega \in \Sigma^* \}$

(If $W_1 = \phi$, some variable will remain after the application of any production, and so $L(G) = \phi$.)

$W_{i+1} = W_i \cup \{ A \in V_N \mid \exists$ some production $A \to \alpha$, $\alpha \in (\Sigma \cup W_i)^* \}$

By def^n, $W_i \subseteq W_{i+1}$ $\forall\, i$.

As V_N has only finite # of variables, $W_k = W_{k+1}$ for some $k \leq |V_N|$. $\therefore$ $W_k = W_{k+j}$ for $j \geq 1$.

(b) **Construction of P'**

$P' = \{ A \to \alpha \mid A, \alpha \in (V_N' \cup \Sigma)^* \}$

Ques:- Let $G = (V_N, \Sigma, P, S)$ be given by the productions $S \to AB$, $A \to a$, $B \to b$, $B \to C$, $E \to c$. Find G' s.t. every variable in G' derives some terminal string.

Sol^n **Construction of V_N'**

$W_1 = \{ A, B, E \}$; $\because$ $A \to a$, $B \to b$, $E \to c$ are the productions with terminal strings on RHS.

$W_2 = W_1 \cup \{ A_1 \in V_N \mid A_1 \to \alpha, \alpha \in (\Sigma \cup \{A, B, E\})^* \}$

$= W_1 \cup \{ S \} = \{A, B, E, S\}$

$W_3 = W_2 \cup \{ A_1 \in V_N \mid A_1 \to \alpha, \alpha \in (\Sigma \cup \{A, B, E, S\})^* \}$

$= W_2 \cup \{\phi\} = W_2$ $\therefore$ $V_N' = \{ S, A, B, E \}$

Construction of P'

$$P' = \{ A_1 \to \alpha \mid A_1, \alpha \in (V_N' \cup \Sigma)^* \}$$

$$= \{ S \to AB,\ A \to a,\ B \to b,\ E \to c \}$$

$$G' = (\{S, A, B, E\}, \{a, b, c\}, P', S)$$

Theorem: For every CFG $G = (V_N, \Sigma, P, S)$, we can construct an equivalent grammar $G' = (V_N', \Sigma', P', S)$ s.t. every symbol in $V_N' \cup \Sigma$ appears in some sentential form (i.e., for every X in $V_N' \cup \Sigma'$ $\exists$ α s.t. $S \overset{*}{\underset{G}{\Rightarrow}} \alpha$ & X is a symbol in the string α)

Proof:- Construction of W_i, $i \geq 1$

(i) $W_1 = \{S\}$

(ii) $W_{i+1} = W_i \cup \{ X \in V_N \cup \Sigma \mid \exists$ a production $A \to \alpha$ with $A \in W_i$ & α containing the symbol $X \}$

Note that $W_i \subseteq V_N \cup \Sigma$ & $W_i \subseteq W_{i+1}$. Again ∵ we have only finite # of elements in $V_N \cup \Sigma$, $W_k = W_{k+1}$ for some k. This means that $W_k = W_{k+j}$ for all $j \geq 0$.

We define $V_N' = V_N \cap W_k$; $\Sigma' = \Sigma \cap W_k$

& $P' = \{ A \to \alpha \mid A \in W_k \}$

Ques: Consider $G = (\{S, A, B, E\}, \{a, b, c\}, P, S)$, where P consists of $S \to AB,\ A \to a,\ B \to b,\ E \to c$.

Sol":- $W_1 = \{S\}$

$W_2 = \{S\} \cup \{ X \in V_N \cup \Sigma \mid \exists$ a production $A \to \alpha$ with $A \in W_1$ & α containing $X \}$

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

$W_3 = \{S, A, B\} \cup \{a, b\}$

$W_4 = W_3$

$V_N' = \{S, A, B\}$; $\Sigma' = \{a, b\}$

$P' = \{ S \longrightarrow AB,\ A \longrightarrow a,\ B \longrightarrow b \}$

Definition: Let $G = (V_N, \Sigma, P, S)$ be a CFG. G is said to be **reduced or non-redundant** if every symbol in $V_N \cup \Sigma$ appears in the course of the derivation of some terminal string, i.e.- for every X in $V_N \cup \Sigma$, $\exists$ a derivation $S \overset{*}{\Rightarrow} \alpha X \beta \overset{*}{\Rightarrow} \omega \in L(G)$ (We can say X is useful in the derivation of terminal strings.)

THEOREM:- For every CFG G $\exists$ a reduced grammar G' which is equivalent to G.

The reduced grammar can be constructed in 2 steps :-

Step 1: We construct a grammar G_1 equivalent to the given grammar G so that every variable in G_1 derives some terminal string.

Step 2: We construct a grammar $G' = (V_N', \Sigma', P', S)$ equivalent to G_1 so that every symbol in G' appears in some sentential form of G' which is equivalent to G_1 & hence to G. G' is the required grammar.

Ques:- Find the reduced grammar equivalent to G whose productions are :-

$S \longrightarrow AB \mid CA$; $B \longrightarrow BC \mid AB$ $A \longrightarrow a$ $C \longrightarrow aB \mid b$.

Soln:- **STEP 1**:-

$W_1 = \{A, C\}$ $(\because A \to a, C \to b)$

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

$W_3 = W_2 \cup \emptyset \Rightarrow W_3 = W_2$

$V_N' = W_2 = \{A, C, S\}$

$P' = \{S \to CA, A \to a, C \to b\}$

Thus,

$G_1 = (\{S, A, C\}, \{a, b\}, \{S \to CA, A \to a, C \to b\}, S)$

Step 2 :-

$W_1 = \{S\}$

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

$W_3 = W_2 \cup \{a, b\}$

$\therefore G' = G_1$ is the reduced grammar.

Ques Construct the reduced grammar.

$S \to aAa$, $A \to Sb \mid bCC \mid DaA$, $C \to abb \mid DD$,

$E \to aC$, $D \to aDA$.

Sol" :- Step 1 :-

$W_1 = \{C\}$ $(\because C \to abb)$

$W_2 = W_1 \cup \{A, E\}$ $(\because A \to bCC, E \to aC)$

$W_3 = W_2 \cup \{S\}$

$W_4 = W_3 \cup \emptyset \Rightarrow W_4 = W_3$

$V_N' = \{S, A, E, C\}$; $P' = \{S \to aAa, A \to Sb \mid bCC, C \to abb, E \to aC\}$

Step 2 :

$W_1 = \{S\}$

$W_2 = \{S\} \cup \{a, A\} = \{S, A, a\}$

$W_3 = W_2 \cup \{b, C\} = \{S, A, C, a, b\}$

$W_4 = W_3 \cup \{a,b\} = W_3$

$\therefore \quad V_N'' = \{S, A, C\}$

$P'' = \{S \to aAa,\ A \to Sb \mid bCC,\ C \to abb\}$

ELIMINATION OF NULL PRODUCTIONS :-

Def^n :- The productions of the form $A \to \Lambda$ $(A \in V_N)$ is called a Null - Production.

EXAMPLE :- $G: S \to aS \mid aA \mid \Lambda$; $A \to \Lambda$.

$\exists$ 2 null-productions $S \to \Lambda$ & $A \to \Lambda$. We can delete $A \to \Lambda$ provided we erase A whenever it occurs in the course of a derivation of a terminal string. So, we can replace $S \to aA$ by $S \to a$.

If G_1 denotes the grammar whose productions are :- $S \to aS \mid a \mid \Lambda$, then $L(G_1) = L(G) = \{a^n \mid n \geq 0\}$.

$\therefore$ it is possible to eliminate null-productions, $A \to \Lambda$. If we eliminate $S \to \Lambda$ then we can't generate Λ in $L(G)$. But we can generate $L(G) - \{\Lambda\}$ even if we eliminate $S \to \Lambda$.

Def^n :- A variable A in a CFG is nullable if $A \overset{*}{\Rightarrow} \Lambda$.

Theorem :- If $G = (V_N, \Sigma, P, S)$ is a CFG, then we can find a CFG G_1 having no null productions s.t. $L(G_1) = L(G) - \Lambda$.

P.T.O.

Step 1: Construction of the set of nullable variables

We find nullable variables recursively:

(i) $W_1 = \{A \in V_N \mid A \to \Lambda \text{ is in } P\}$

(ii) $W_{i+1} = W_i \cup \{A \in V_N \mid \exists$ a production $A \to \alpha$ with $\alpha \in W_i^*\}$

By defn of W_i; $W_i \subseteq W_{i+1}$ $\forall i$. As V_N is finite, $W_{k+1} = W_k$ for some $k \leq |V_N|$. So, $W_{k+j} = W_k$ $\forall j$. let $W = W_k$. W is the set of all nullable variables.

Step 2:- Construction of P':

(i) Any production whose R.H.S. doesn't have any nullable variables is included in P'.

(ii) If $A \to X_1 X_2 \ldots X_k$ is in P, the production of the form $A \to \alpha_1 \alpha_2 \ldots \alpha_k$ are included in P', where $\alpha_j = X_i$, if $X_i \notin W$. $\alpha_i = X_i$ or Λ if $X_i \in W$ and $\alpha_1 \alpha_2 \ldots \alpha_k \neq \Lambda$.

Actually (ii) gives several productions in P'. The productions are obtained either by not erasing any nullable variable on the R.H.S. of $A \to X_1 X_2 \ldots X_k$ or by erasing some or all nullable variables provided some symbol appears on the R.H.S. after erasing.

Then $G_1 = (V_N, \Sigma, P', S)$ is the required grammar with no null productions.

Ques:- Consider G with $P = \{S \to aS \mid AB, A \to \Lambda, B \to \Lambda, D \to b\}$. Construct a grammar G_1 without null productions generating $L(G) - \{\Lambda\}$.

Sol^{n} :- Step 1 :- Construction of set of nullable variables

$W_1 = \{ A \in V_N \mid A \to \Lambda \text{ is in } P \}$

$= \{ A, B \}$

$W_2 = \{A, B\} \cup \{S\}$ (as $S \to AB$; where $AB \in W_i^*$)

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

$W_3 = W_2 \cup \phi = W_2$

$\therefore W = W_2 = \{S, A, B\}$

Step 2: Construction of P'

(i) $D \to b$ is in P'

(ii) $S \to aS$ give rise to $S \to aS$ & $S \to a$

(iii) $S \to AB$ give rise to $S \to AB$; $S \to A$ and $S \to B$.
(we can't erase both nullable variable as in that case, we will get $S \to \Lambda$)

Result: There exists an algorithm to decide whether $\Lambda \in L(G)$ for a given CFG G.

☆ $\Lambda \in L(G)$ iff $S \in W$ (i.e. S is nullable)

ELIMINATION OF UNIT PRODUCTIONS

Def^{n} :- A Unit Production (or a chain rule) in a CFG G is a production of the form $A \to B$; $A, B \in V_N$.

Theorem :- If G is a CFG, we can find a CFG G_1 which has no null or unit productions s.t. $L(G_1) = L(G)$.

∴ We can apply corollary below of last theorem :-

COROLLARY :- If $G = (V_N, \Sigma, P, S)$ is a CFG we can find an equivalent CFG $G_1 = (V_N', \Sigma, P_1, S_1)$ without null-productions except $S_1 \to \Lambda$ when $\Lambda \in L(G)$. If $S_1 \to \Lambda$ is in P_1, S_1 does not appear on the R.H.S. of any production in P_1.

We apply above corollary to grammar G to get a gramma $G' = (V_N, \Sigma, P, S)$ without null productions s.t. $L(G') = L(G)$.

Let $A \in V_N$ (any variable)

Step 1 :- Construction of set of variables derivable from A

Define $W_i(A)$ recursively as follows :-

$W_0(A) = \{A\}$

$W_{i+1}(A) = W_i(A) \cup \{B \in V_N \mid C \to B \text{ is in } P \text{ with } C \in W_i(A)\}$

By def^n of $W_i(A)$, $W_i(A) \subseteq W_{i+1}(A)$. As V_N is finite, $W_{k+l}(A) = W_k(A)$ for some $k \leq |V_N|$. So, $W_{k+j}(A) = W_k(A)$ $\forall\ j \geq 0$. Let $W(A) = W_k(A)$. Then, $W(A)$ is the set of all variables derivable from A.

Step 2 :- Construction of A-productions in G_1:

The A-productions in G_1 are either

(i) the non-unit production in G' or

(ii) $A \to \alpha$ whenever $B \to \alpha$ is in G with $B \in W(A)$ & $\alpha \notin V_N$.

Actually (ii) covers (i) as $A \in W(A)$. Now we define $G_1 = (V_N, \Sigma, P_1, S)$, where P_1 is constructed using step 2 for every $A \in V_N$.

Ques Let G be $S \to AB, A \to a, B \to C \mid b, C \to D, D \to E$ & $E \to a$. Eliminate unit productions to get an equivalent grammar.

Soln :- Step 1 :- $W_0(S) = \{S\}$

$W_1(S) = W_0(S) \cup \phi = W_0(S)$

$\therefore W(S) = \{S\}$

Illy; $W(A) = \{A\}$; $W(E) = \{E\}$

$W_0(B) = \{B\}$, $W_1(B) = \{B, C\}$, $W_2(B) = \{B, C, D\}$,

$W_3(B) = \{B, C, D, E\}$; $W_4(B) = W_3(B)$

$\therefore W(B) = \{B, C, D, E\}$.

$W(C) = \{C, D, E\}$; $W(D) = \{D, E\}$

Step 2 :- The productions in G_1 are :-

$S \to AB$, $A \to a$, $E \to a$

$B \to b \mid a$, $C \to a$; $D \to a$

By construction, G_1 has no unit productions.

COROLLARY :- If G is a CFG, we can find an equivalent grammar G' which is reduced & has no null productions or unit productions.

Proof :- We can construct G' in the following way :-

STEP 1 :- Eliminate null productions to get G_1.

STEP 2 : Eliminate unit productions in G_1 to get G_2.

STEP 3 : Construct a reduced grammar G' equivalent to G_1.

G' is the required grammar equivalent to G.

★ Note that if we change the order in the above theorem, we may not get the most simplified grammar.