Context Free Languages

A. Sud

(1) Free Grammar (CFG) is a grammar is
(2) every production is of the form $A \to \alpha$, where $A \in V$ and $\alpha \in (V \cup T)^*$.

Derivation Tree

A derivation in CFG can be represented using trees. Such trees are called derivation trees.

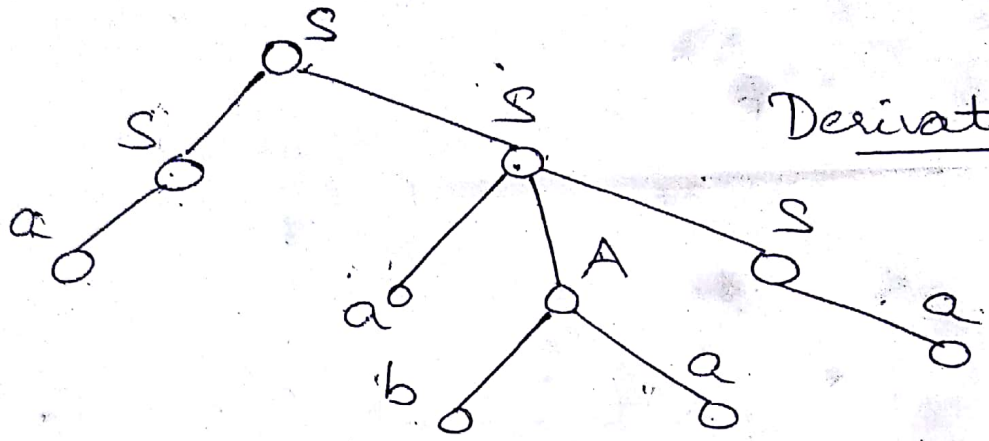
A derivation tree (or a parse tree) for a CFG $G = (V, T, S, P)$ is a tree satisfying the following conditions:-

1) Every vertex has a label which is a variable or symbol of A.
2) Root has label S.
3) Label of an internal vertex is a variable.
4) A vertex n is a leaf if its label is a symbol $a \in \Sigma$ or $\wedge$ (If leaf is $\wedge$, then the leaf is the only son of its father)
5) For the production like $A \to X_1 X_2 \cdots X_k$ in P, then we have a vertex n with label A which has sons with labels $X_1, X_2, \cdots X_k$.

eg. Let $G = (\{S, A\}, \{a, b\}, S, P)$ where P is :-

$S \rightarrow aAS \mid a \mid SS$

$A \rightarrow SbA \mid ba$



Derivation Tree

Ordering of Leaves from the Left

Yield of the derivation tree is the concatenation of the labels of the leaves without repetition in the left to right ordering.

eg. Yield of the above derivation tree is :-

aabaa

The derivation of the above derivation tree is

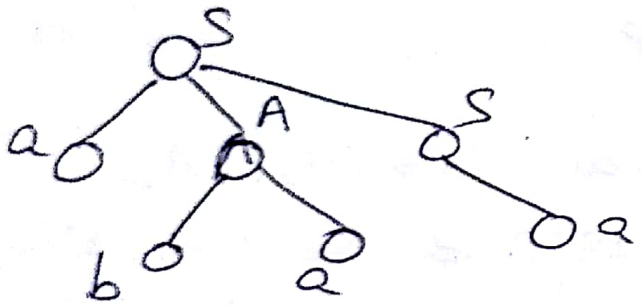
$S \Rightarrow SS \Rightarrow aS \Rightarrow aaAS \Rightarrow aabaS$

$\Downarrow$

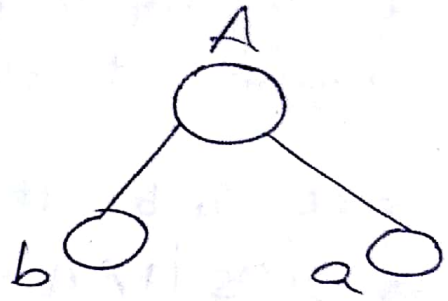
aabaa

A subtree of a derivation tree T is a tree:-
- whose root is some vertex v of T.
- whose vertices are the descendants of v together with their labels.
- whose edges are those connecting the descendants of v.

Eg: The subtrees of the above derivation tree are:



(a)



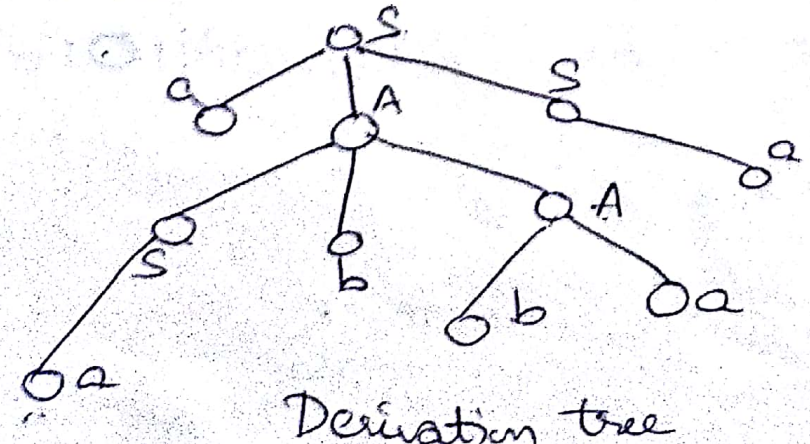
(b)

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

A: $S \Rightarrow aAS \Rightarrow asbAS \Rightarrow aabAS \Rightarrow a^2bbaS$

$\Downarrow$

$a^2b^2a^2$



Derivation tree

Leftmost derivation

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

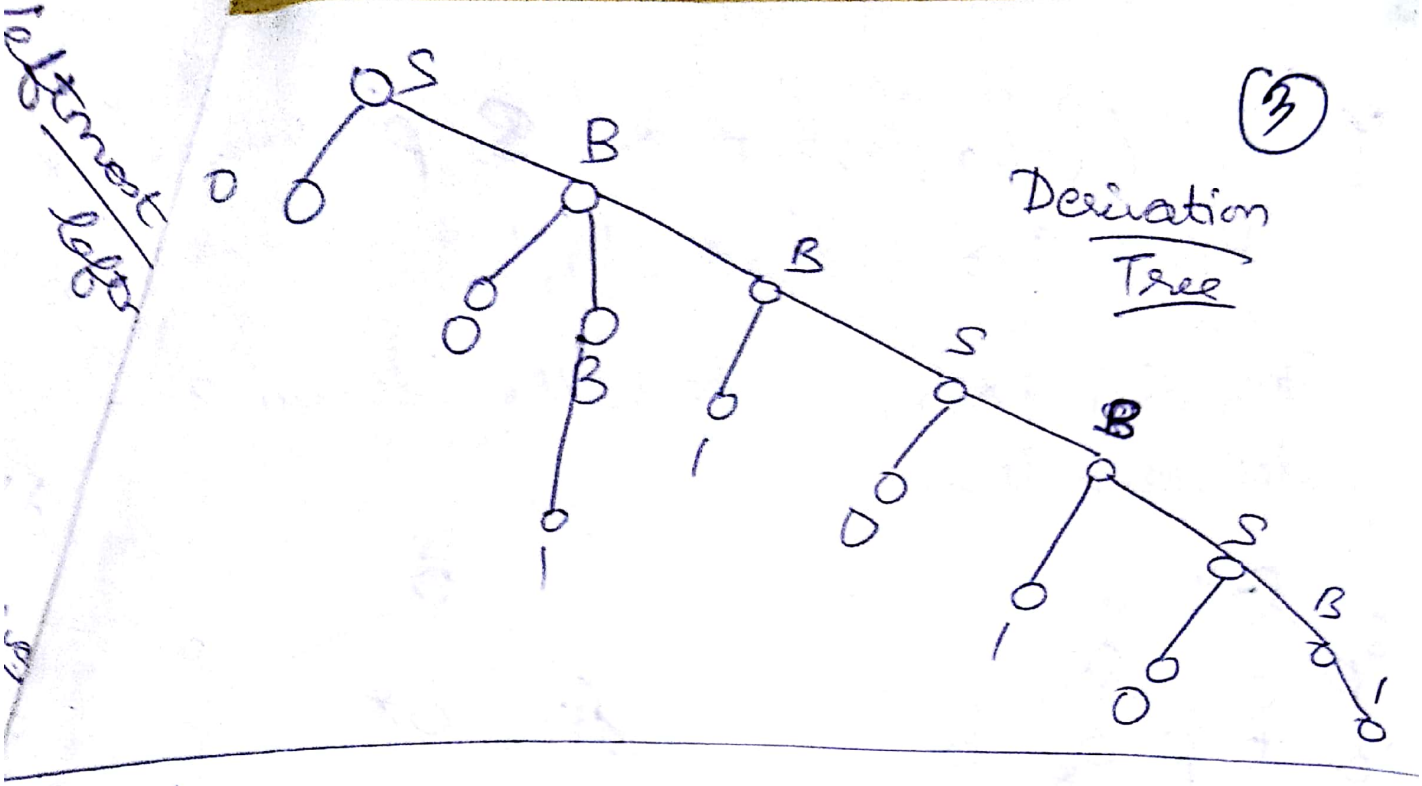
Rightmost derivation

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

Q: Let G be the grammar $S \to 0B \mid 1A$, $A \to 0 \mid 0S \mid 1AA$, $B \to 1 \mid 1S \mid 0BB$. For the string 00110101, find (a) leftmost derivation (b) the rightmost derivation and (c) the derivation tree.

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

(b) $S \Rightarrow 0B \Rightarrow 00BB \Rightarrow 00B1S \Rightarrow 00B10B$
$\Rightarrow 0^2 B101S \Rightarrow 0^2 B1010B \Rightarrow 0^2 B1010$
$\Rightarrow 0^2 11010$.



Derivation Tree

Ambiguity in Context Free Grammar

In some CFG, the same terminal string may be the yield of two derivation trees. So, there may be two different leftmost derivations (or 2 different rightmost derivation) of w.

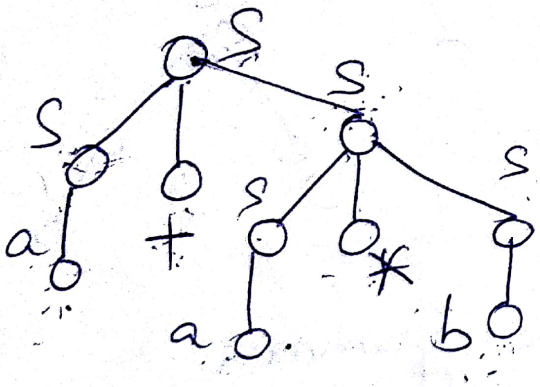
A terminal string $w \in L(G)$ is ambiguous if there exists two or more derivation trees for w (or there exist two or more leftmost or rightmost derivations) of w.

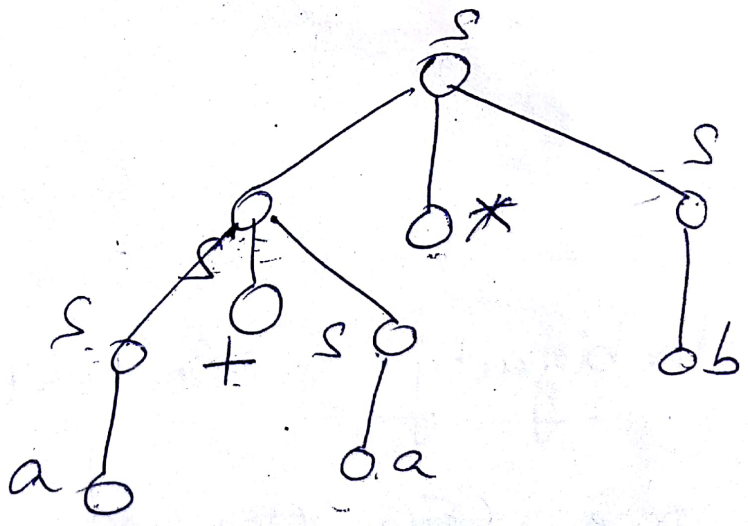
A context free grammar G is ambiguous if there exists some $w \in L(G)$ which is ambiguous.

eg. $G = (\{S\}, \{a, b, +, *\}, S, P)$ wh

Ps: $S \rightarrow S+S \mid S*S \mid a \mid b$

for $a+a*b$, we have the following 2 derivation trees:-





eless productions

em: Let $G = (V, T, S, P)$ be a CFG. , there exists an equivalent grammar $=(V, T, S, P)$ that does not contain any seless variables or productions.

Proof: In first part, we construct an intermediate grammar:-
$G_1 = (V_1, T_1, S, P_1)$ such that V_1 contains only variables A for which:-

(1) Set V_1 to ϕ.
(2) Repeat the following steps until no more variables are added to V_1.
For each $A \in V$ for which P has a production of the form
$A \to x_1 x_2 \cdots x_n$ with all x_i in $V_1 \cup T$, add A to V_1.
(3) Take P_1 as all productions in P whose symbols are in $V_1 \cup T$.

In 2nd part, we draw variable dependency graph for G. This graph has its vertices labelled with variables, with an edge between vertices C+D iff there is a production form $C \to xDy$.

From this graph, we find all variables that cannot be reached from S. These variables and productions which them are removed from the set.

Eg. $S \to a \mid aA \mid B \mid C$

$A \to aB \mid \wedge$

$B \to Aa$

$C \to cCD$

$D \to ddd$

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

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

$V_1 = \phi$

$V_1 = \{S\}$ $\because S \to a$ (and $a \in V_1 \cup T$)

$V_1 = \{S\}$ ~~$A \to aB$~~ ($aB \notin V_1 \cup T$)

$V_1 = \{S, A\}$ $A \to \wedge$ ($\wedge \in V_1 \cup T$)

$V_1 = \{S, A, B\}$ ~~$B \to Aa$~~ ($Aa \in V_1 \cup T$)

$V_1 = \{S, A, B\}$ $C \to cCD$ ($cCD \notin V_1 \cup T$)

$V_1 = \{S, A, B, D\}$ $D \to ddd$ ($ddd \in V_1 \cup T$)

So, remove C and all productions of C.

$V_1 = \{S, A, B, D\}$

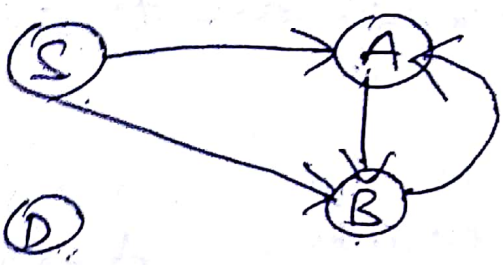
$T_1 = \{a, c, d\}$

P_1 is: $S \to a \mid aA \mid B$

$A \to aB \mid \wedge$

$B \to Aa$

$D \to ddd$



D is eliminated as no path from S to D.

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

$\hat{T} = \{a\}$

S – start variable

$\hat{P}$: $S \to a \mid aA \mid B$

$A \to aB \mid \wedge$

$B \to Aa$

Removing Λ-productions

Any production of a CFG of the form $A \to \Lambda$ is called a Λ-production.

Any variable A for which the derivation $A \overset{*}{\Rightarrow} \Lambda$ is possible is called nullable.

Theorem: Let G be any CFG with Λ not in L. Then, there exists an equivalent CFG $\hat{G}$ has no Λ-productions.

Proof: (1) For all productions, $A \to \Lambda$, put A into V_N. V_N is set of all nullable variables.

(2) Repeat the following step until no further variables are added to V_N.

For all production $B \to A_1, A_2 \dots A_N$ where $A_1, A_2 \dots A_N$ are in V_N. Put B into V_N.

For each such production of P, we put into $\hat{P}$, that production as well as all those generated by replacing nullable variables with Λ in all possible combinations. But if all x_i are nullable, the production $A \to \Lambda$ is not put into $\hat{P}$.

Eliminate all Λ-production

$S \rightarrow ABaC$

$A \rightarrow BC$

$B \rightarrow b \mid \Lambda$

$C \rightarrow D \mid \Lambda$

$D \rightarrow d$

$V_N = \{A, B, C\}$ (as $B \rightarrow \Lambda$, $C \rightarrow \Lambda$ and $A \rightarrow BC$)

Replace nullable variables A, B, C by Λ one by one.

$S \rightarrow ABaC \mid BaC \mid AaC \mid ABa \mid aC \mid Aa \mid Ba \mid a$

$A \rightarrow BC \mid B \mid C$

$B \rightarrow b$

$C \rightarrow D$

$D \rightarrow d$

Removing Unit production

Any production of a CFG of the form $A \rightarrow B$ where $A, B \in V$ is called a unit production.

Theorem: Let $G = (V, T, S, P)$ be any CFG without Λ productions. Then, there is a CFG, $\hat{G} = (\hat{V}, \hat{T}, S, \hat{P})$ that does not have any unit productions and that is equivalent to G.

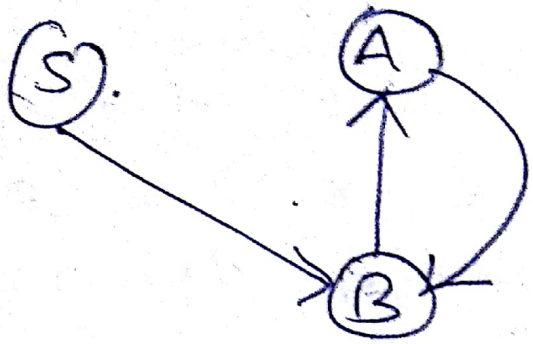
Proof: We can remove $A \to A$ without effect. Next, we draw a dependency graph with an edge (C, D) whenever the grammar has a unit production $C \to D$. If there is a walk between A & B, then $A \overset{*}{\Rightarrow} B$, the new grammar $\hat{G}$ is generated by 1st putting into $\hat{P}$, all non unit productions of P. Next for all A & B satisfying $A \overset{*}{\Rightarrow} B$, we add to $\hat{P}$ -

$$A \to y_1 | y_2 | \cdots | y_n$$

where $B \to y_1 | y_2 | \cdots | y_n$ is set of rules in $\hat{P}$ with B on left.

Q: Remove all unit productions.

$$S \to Aa | B$$
$$B \to A | bb$$
$$A \to a | bc | B$$



$S \overset{*}{\Rightarrow} B$, $B \overset{*}{\Rightarrow} A$, $A \overset{*}{\Rightarrow} B$

$S \overset{*}{\Rightarrow} A$

Add non unit productions

$S \rightarrow Aa$

$B \rightarrow bb$

$A \rightarrow a \mid bC \mid \cancel{B}$

In Unit productions:

$S \rightarrow bb$ (for $S \overset{*}{\Rightarrow} B$)

$S \rightarrow a \mid bc$ (for $S \overset{*}{\Rightarrow} A$)

$B \rightarrow a \mid bC$ (for $B \overset{*}{\Rightarrow} A$)

$A \rightarrow bb$ (for $A \overset{*}{\Rightarrow} B$)

So, we have P:-

$S \rightarrow Aa \mid bb \mid a \mid bC$

$B \rightarrow bb \mid a \mid bC$

$A \rightarrow a \mid bC \mid \cancel{B} \mid bb$

Simplification of Grammar

Thm: Let L be a CFL, that does not contain Λ. Then there exists a CFG, that generates L and that does not have any useless productions, Λ-productions or unit productions.

Sol:

1. Remove Λ-productions
2. Remove unit productions
3. Remove useless productions.

Q: Eliminate, Λ, unit and useless productions and simplify the grammar.

(a)
$$S \to aA$$
$$A \to aaA \mid \Lambda$$
$$B \to bc \mid bbC$$
$$C \to B$$

(i) Λ production

$V_N = \{A\}$

$S \to aA \mid a$

$A \to aaA \mid aa$

$B \to bC \mid bbC$

$C \to B$

* change in previous note

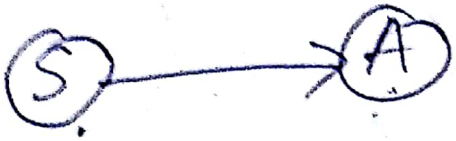
C ⇒ B

S → aA | a

A → aaA | aa

B → bC | bbC

C → bc | bbC



∴ B, C are useless

S → aA | a

A → aaA | aa

Chomsky Normal form

A CFG is in Chomsky Normal form if all production are of the form:

$A \to BC$ or $A \to a$

where A, B, C are in V,

& a is in T.

eg. S → AS | a

A → SA | b

Note: First simplify grammar (i.e remove ∧ and unit productions) and then convert into chomsky Normal form.

eg. Convert the grammar:-

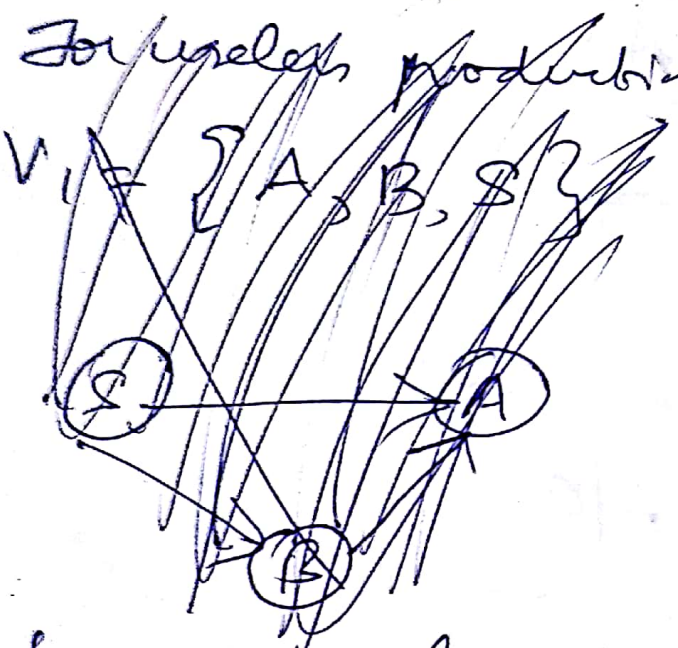
$S \rightarrow ABa$

$A \rightarrow aab$

$B \rightarrow Aa$

to Chomsky Normal form.

A There are no $\wedge$ and unit productions.



~~So no useless production~~

For Chomsky Normal form:

$S \rightarrow AD$

$D \rightarrow BE$ → where $D \rightarrow Ba$, $a \in$

$E \rightarrow a$

$A \rightarrow EG$ → where $G \rightarrow \in F$

$F \rightarrow b$

$G \rightarrow \in F$

$B \rightarrow AE$

Griebach Normal form

CFG is in Griebach Normal form if all productions are of the form:-

$$A \rightarrow ax$$

where $a \in T$ and $x \in V^*$, $A \in V$.

For Griebach Normal form also remove λ, null & unit productions.

Eg. $S \rightarrow AB$

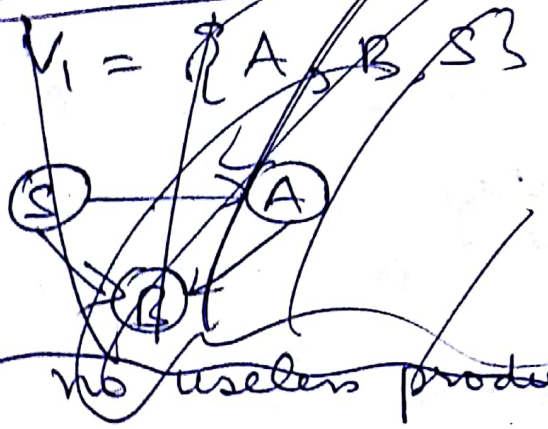
$A \rightarrow aA \mid bB \mid b$

$B \rightarrow b$

There are no λ and unit productions.

For useless productions

$V_1 = \{A, B, S\}$



So no useless productions also.

For Griebach Normal form:-

$S \rightarrow aAB \mid bBB \mid bB$

replace $A \rightarrow aA$ (A by bB) → A by b

$A \rightarrow aA \mid bB \mid b$

$B \rightarrow b$

Pumping Lemma for CFL

Let L be an infinite CFL, then there exist some +ve integer m such that any $w \in L$ with $|w| \geq m$ can be decomposed as:

$$w = uvxyz$$

with $|vxy| \leq m$ and $|vy| \geq 1$ such that $uv^i xy^i z \in L$ for $i = 0, 1, 2, \ldots$

If L is CFL, then it must hold pumping lemma. If pumping lemma is not satisfied for some L, then L is not CFL.

To prove a language to be non CFL:

1. For given +ve integer m, pick a string w in L of length equal or greater than m.
2. For all decomposition of w as $uvxyz$ such that $|vxy| \leq m$ and $|vy| \geq 1$, pick i for each decomposition in such a way that the pumped string w^i is not in L.

Q: Prove that $L = \{a^n b^n c^n : n \geq 0\}$ is not CFL.

y contains 'a' only.

$w = uvxyz$

$\underbrace{aa\text{---}a}_{u\ vxy}\ \underbrace{bb\text{---}b\ cc\text{---}c}_{z}$

$w_i = uv^ixy^iz$

$w_0 = \underbrace{a^{m-k}}_{u}\underbrace{a^k}_{v^ixy^i}\underbrace{b^m c^m}_{z}$

$i = 0$

$w_0 = a^k b^m c^m \notin L$

as $|vy| \geq 1$, vy contains atleast one a.
so w_0 has less no. of 'a' than 'b' or 'c'

) vxy contains a's + b's)

$\underbrace{aa\text{---}a}_{u}\underbrace{bb\text{---}b\ cc\text{---}c}_{vxy\ \ z}$

$w_i = uv^ixy^iz$

$w = \underbrace{a^{m-k}}_{u}\underbrace{a^k b^k}_{vxy}\underbrace{b^{m-k} c^m}_{z}$

As $|vy| \geq 1$, vy contains atleast one 'a'.
So, w_0 contains less no. of 'a' or 'b' than 'c'

Properties of CFL

The family of CFL is closed under union, concatenation & star closure operation.

Proof: Let L_1 and L_2 be 2 CFLs generated by $G_1 = (V_1, T_1, S_1, P_1)$ and $G_2 = (V_2, T_2, S_2, P_2)$ respectively.

$(V_1 \cap V_2) = \phi$

Let $G_3 = (V_3, T_3, S_3, P_3)$ be a CFG such that $L(G_3) = L(G_1) \cup L(G_2)$

$V_3 = V_1 \cup V_2 \cup \{S_3\}$

$T_3 = T_1 \cup T_2$

S_3 – start variable for G_3

$P_3 = P_1 \cup P_2 \cup \{S_3 \rightarrow S_1 \mid S_2\}$

Let $G_4 = (V_4, T_4, S_4, P_4)$ be CFG such that $L(G_4) = L(G_1)\,L(G_2)$

$V_4 = V_1 \cup V_2 \cup \{S_4\}$

$T_4 = T_1 \cup T_2$

S_4 = start variable for G_4.

$P_4 = P_1 \cup P_2 \cup \{S_4 \rightarrow S_1 S_2\}$

Let $G_5 = (V_5, T_5, S_5, P_5)$ be CFG such that $L(G_5) = L(G_1)^*$

$V_5 = V_1 \cup \{S_5\}$

$T_5 = T_1$

S_5 : start variable for G_5

Proof for union

$L(G_3) = L_1 \cup L_2$

$\therefore w \in L_1 \Rightarrow S_3 \Rightarrow S_1 \overset{*}{\Rightarrow} w$

$w \in L_2 \Rightarrow S_3 \Rightarrow S_2 \overset{*}{\Rightarrow} w$

If $w \in L(G_3)$

(1) either $S_3 \Rightarrow S_1$

(2) or $S_3 \Rightarrow S_2$

i) If $S_3 \Rightarrow S_1$ has V_1 $(\because V_1 \cap V_2 = \phi)$
then derivation $S_1 \overset{*}{\Rightarrow} w$ involves productions in P_1 only

ii) If $S_3 \Rightarrow S_2$, has V_2
then derivation $S_2 \overset{*}{\Rightarrow} w$ involves productions in P_2 only.

$\therefore$ either (1) or (2)

---

(2) The family of CFLs is not closed under intersection and complementation operation.

Proof: Consider 2 languages :-

Let $L_1 = \{a^n b^n c^m : n, m \geq 0\}$

$L_2 = \{a^n b^m c^m ; n, m \geq 0\}$

L_1

$S \to S_1 S_2$

$S_1 \to a S_1 b \mid \wedge$

$S_2 \to c S_2 \mid \wedge$

L_2

$S \to S_1 S_2$

$S_1 \to a S_1 \mid \wedge$

$S_2 \to b S_2 c \mid \wedge$

$L_1 \cap L_2 = \{a^n b^n c^n : n \geq 0\}$ is not CFL (we have proved using pumping lemma

Complementation

Proof: using proof by contradiction technique

Let us assume that family of CFLs is closed under complementation.

ie L_1 and L_2 are CFLs

$\Rightarrow L_1 \cup L_2$ is also CFL (family of CFLs is closed under union operation)

$\Rightarrow \overline{L_1} \cup \overline{L_2}$ is also CFL (by assumption)

$\Rightarrow L_1 \cap L_2$ is CFL ($L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$)

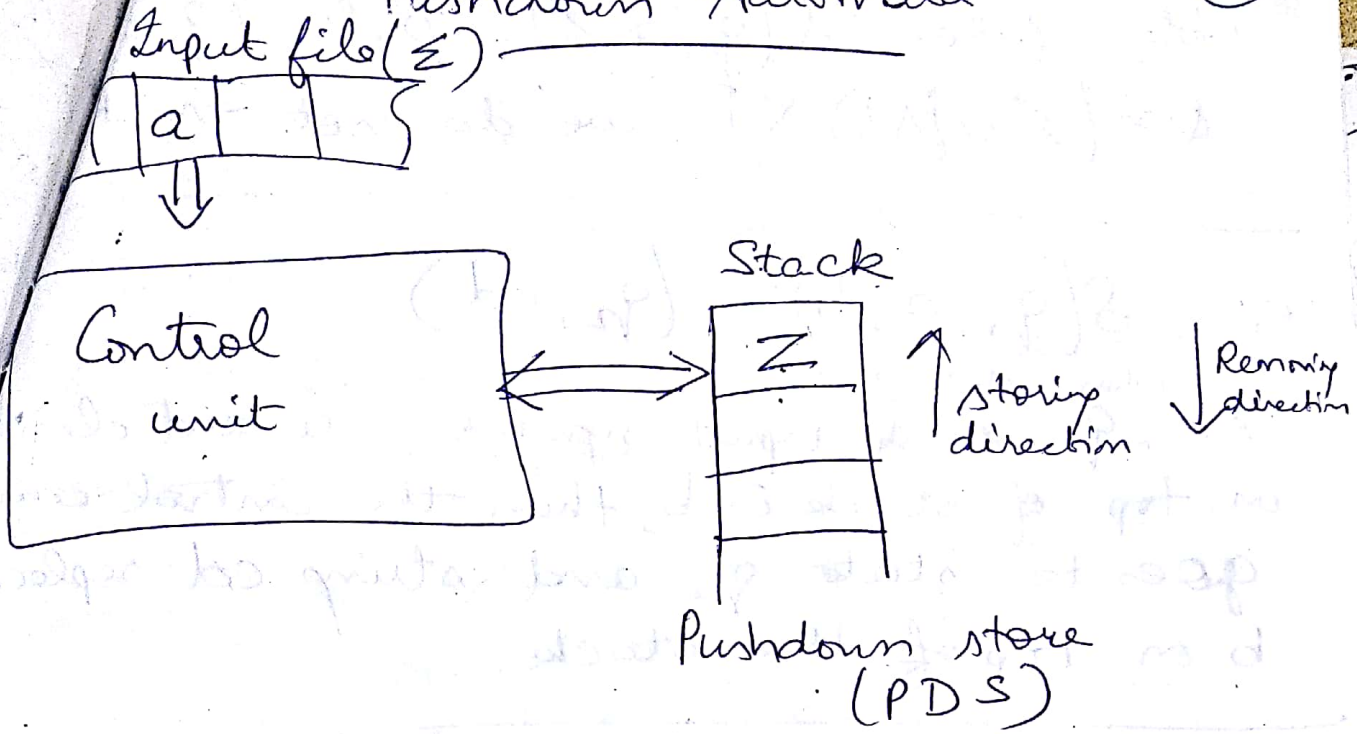
ie family of CFL is closed under intersection

But it is not true

So, our assumption is wrong.

Family of CFLs is not closed under complementation.

Pushdown Automata



A pushdown automaton consists of:-

sep-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$$

where

Q is finite set of internal states of control unit

Σ is input alphabet

Γ is finite set of symbols called stack alphabet.

$\delta : Q \times (\Sigma \cup \{\Lambda\}) \times \Gamma \rightarrow$ finite subsets of $Q \times \Gamma^*$ is the transition function

$q_0 \in Q$ is intial state of control unit.

$Z \in \Gamma$ is stack start symbol

$F \subseteq Q$ is set of final states

Note: When $\delta(q, a, Z) = \phi$ for $(q, a, Z) \in Q \times (\Sigma \cup \{\Lambda\}) \times \Gamma$, we do not mention it.

---

eg. $\delta(q_1, a, b) = (q_2, cd)$

when state is

Here, q_1, input symbol is a and element on top of stack is b, then the control unit goes to state q_2 and strip cd replaces b on top of the stack.

Remarks: (1) $\delta(q, a, Z)$ is finite subset of $Q \times \Gamma^*$. The elements of $\delta(q, a, Z)$ are of the form (q', α), where $q' \in Q$, $\alpha \in \Gamma^*$. $\delta(q, a, Z)$ may be empty set.

(2) At any time, the pda is in some state q and the PDS has some symbols from Γ. The pda reads an input symbol a and topmost symbol Z in PDS. Using δ, pda makes a transition to state q' and writes strip α after removing Z. When $\alpha = \Lambda$, topmost symbol Z is erased.

* δ is defined on $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$ means may make transition without reading input symbol ie when $\delta(q, \lambda, z)$ defined, it is called λ-moves.

Pda cannot take a transition when PDS is empty, here the pda halts.

When we write $\alpha = z_1 z_2 \ldots z_m$ in PDS, z_1 is topmost element, z_2 is below z_1 etc. and z_m is below z_{m-1}.

Instantaneous description (ID)

Let $A = (Q, \Sigma, \Gamma, \delta, p_0, z, F)$ be pda, an instantaneous description (ID) is (q, x, α) where $q \in Q$, $x \in \Sigma^*$ and $\alpha \in \Gamma^*$

eg. $(q, a_1 a_2 \ldots a_n, z_1 z_2 \ldots z_m)$ is an ID

where:-

q: current state

$a_1 a_2 \ldots a_n$: input string to be processed in same order.

$z_1, z_2 \ldots z_m$: PDS has z_1 at top, z_2 at second position from top and z_m is lowest element in PDS.

The triplet (q, w, u) is an ID where

q: state of control unit

w: unread part of input strip

u: stack contents with leftmost symbol indicating the top of the stack.

A move from one ID to another will be denoted by symbol $\vdash$

Q:- Design PDA for

$$L = \{a^n b^n : n \geq 0\}$$

We have equal no. of a's & b's, we push '1' to stack on reading a and pop '1' on reading b as a and b are equal.

$$(\{q_0, q_1\}, \{a, b\}, \{z, 1\}, \delta, q_0, z, \{q_1\})$$

δ is:-

$\delta(q_0, a, z) = (q_0, 1z)$

$\delta(q_0, a, 1) = (q_0, 11)$

$\delta(q_0, b, 1) = (q_0, \wedge)$

$\delta(q_0, \wedge, z) = (q_1, \wedge)$

(Here $\delta(q_0, b, z) = \phi$ as 'b' cannot be 1st input symbol initially as $L = a^n b^n$)

$w = aaabbb \in L$ should be accepted.

$(q_0 aaabbb, z) \vdash (q_0, aabbb, 1z)$

$\vdash (q_0, abbb, 11z)$

$\vdash (q_0, bbb, 111z)$

$\vdash (q_0, bb, 11z)$

$\vdash (q_0, b, 1z)$

$\vdash (q_0, \wedge, z)$

~~$\vdash (q_2, \wedge, \wedge)$~~

$\vdash (q_1, \wedge, \wedge)$

So, it is accepted

$\omega = abab \notin L$, should not be accepted

$(q_0, \underline{a}bab, \underline{z}) \vdash (q_0, \underline{b}ab, \underline{1}z)$

$\vdash (q_0, \underline{a}b, \underline{z})$

$\vdash (q_0, \underline{b}, \underline{1}z)$

$\vdash (q_0, \wedge, z)$

$\vdash (q_1, \wedge, \wedge)$

It is accepted, so our pda is wrong.

We redefine ~~PDA~~ PDA as:-

$(\{q_0, q_1, q_2\}, \{a, b\}, \{z, 1\}, \delta, q_0, z, \{q_2\})$

Change here is that on reading b, the state is changed to q_1 and q_2 is final state.

δ is:-

$\delta(q_0, a, z) = (q_0, 1z)$

$\delta(q_0, a, 1) = (q_0, 11)$

$\delta(q_0, b, 1) = (q_1, \wedge)$

$\delta(q_1, b, 1) = (q_1, \wedge)$

$\delta(q_1, \wedge, z) = (q_2, \wedge)$

$\delta(q_0, \wedge, z) = (q_2, \wedge)$ (for $\omega = \wedge$)

eg. $w = abab$

$(q_0, abab, z) \vdash$ ~~$(q_0, b, 1z)$~~

$\vdash (q_0, bab, 1z)$

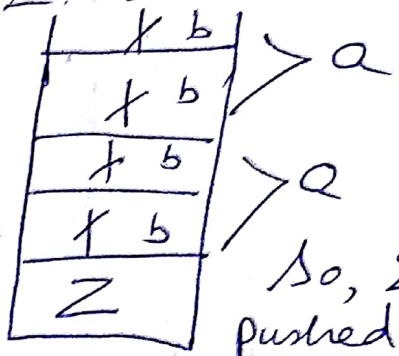
$\vdash (q_1, ab, z) = \phi$

~~$\vdash (q_0, b, 1z)$~~

So, not accepted.

Q: $L = \{a^n b^{2n} : n \geq 0\}$

eg. aabbbb



So, 2 '1's are pushed for every a and '1' is poped for every 'b'.

$(\{q_0, q_1, q_2\}, \{a, b\}, \{z, 1\}, \delta, q_0, z, \{q_2\})$

δ is:

$\delta(q_0, a, z) = (q_0, 11z)$

$\delta(q_0, a, 1) = (q_0, 111)$

$\delta(q_0, b, 1) = (q_1, 1)$ (Here again $\delta(q_0, b, z) = \phi$)

$\delta(q_1, b, 1) = (q_1, 1)$

$\delta(q_1, 1, z) = (q_2, 1)$

$(q_0, \wedge, z) = (q_2, \wedge)$ (for $\omega = \wedge$) (4)

aabbbb

$(q_0, aabbbb, z) \vdash (q_0, abbbb, 1z)$

$\vdash (q_0, bbbb, 11z)$

$\vdash (q_1, bbb, 111z)$

$\vdash (q_1, bb, 11z)$

$\vdash (q_1, b, 1z)$

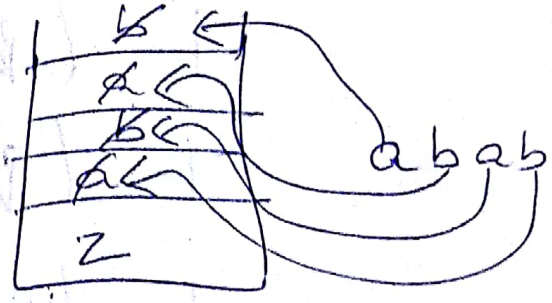
$\vdash (q_1, \wedge, z)$

$\vdash (q_2, \wedge, \wedge)$

So it is accepted.

Q: Construct a pda accepting the set of all strings over $\{a, b\}$ with equal no. of a's and b's.

A: eg. abab



So, whenever input string is 'a', and top of stack is 'b', then 'b' is erased, and when input string is 'b' and top of stack is 'a', then 'a' is erased.

So, pda is :-

$A = (\{q_0, q_f\}, \{a, b\}, \{Z_0, a, b\}, \delta, q_0, Z_0, \{q_f\})$

where δ is :-

$\delta(q_0, a, Z) = (q_0, aZ)$

$\delta(q_0, b, Z) = (q_0, bZ)$

$\delta(q_0, a, a) = (q_0, aa)$

$\delta(q_0, b, b) = (q_0, bb)$

$\delta(q_0, a, b) = (q_0, \Lambda)$

$\delta(q_0, b, a) = (q_0, \Lambda)$

$\delta(q_0, \Lambda, Z) = (q_f, \Lambda)$

$q, w = abab$

$(q_0, abab, Z) \vdash (q_0, bab, aZ)$

$\vdash (q_0, ab, Z)$

$\vdash (q_0, b, aZ)$

$\vdash (q_0, \Lambda, Z)$

$\vdash (q_f, \Lambda, \Lambda)$

Q3. Construct a pda for accepting the language:-

$$L = \{ w w^R : w \in \{a, b\}^+ \}$$

eg. $ab \mid ba$

(1) Add a for 'a' and b for 'b' Initially

$ab \wedge ba$

(2) when a null is encountered ie middle is reached, change state to q_1 and then:-

(3) 

if bb, then ∧

(4) δ_{01} 

if aa, then ∧ again

$Q = \{q_0, q_1, q_2\}$

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

$\Gamma = \{a, b, z\}$

$F = \{q_2\}$

$\delta(q_0, a, a) = \{q_0, aa\}$

$\delta(q_0, b, a) = (q_0, ba)$

$\delta(q_0, a, b) = (q_0, ab)$

$\delta(q_0, b, b) = (q_0, bb)$

$\delta(q_0, a, z) = (q_0, az)$

$\delta(q_0, b, z) = (q_0, bz)$

Next we guess the middle of the string, where the pda switches from state q_0 to q_1,

$\delta(q_0, \wedge, a) = (q_1, a)$

$\delta(q_0, \wedge, b) = (q_1, b)$

a set to match w^R against contents of stack,

$\delta(q_1, a, a) = (q_1, \wedge)$

$\delta(q_1, b, b) = (q_1, \wedge)$

} To match a b | b a

and finally

$\delta(q_1, \wedge, z) = (q_2, \wedge)$

eg: $(q_0, abba, z)$

$\vdash (q_0, bba, az)$

$\vdash (q_0, \Delta ba, baz)$

- Middle of string has come, so we apply move + move to new state.

$\vdash (q_1, ba, baz)$

$\vdash (q_1, a, az)$

$\vdash (q_1, \Lambda, z)$

$\vdash (q_2, \Lambda, \Lambda)$