UNIT-3

CONTEXT FREE GRAMMAR

To define a Syntactic Structure we use Grammar.

Grammar

A Grammar contains set of rules to define a formal language.

Definition

A grammar $G = (V, T, P, S)$ (or) (N, T, P, S) it is defined by 4 tuples.

- $V \longrightarrow$ set of non-terminals, (all non-terminals must be represented in uppercase)
- $T \longrightarrow$ set of terminals (input symbols (Σ)) all (lower case letters, digits, operators and keywords will be treated as terminals)
- $S \rightarrow$ Start Symbol (z_0)
- $P \rightarrow$ Productions are used to defines rules.

In a grammar every Production is in the form of

$$\alpha \longrightarrow \beta \ , \ \alpha, \beta \in (V \cup T)^*$$

Types of Grammars

Based on the productions grammars are classified into 4 types.

1. Type-0 grammar
2. Type-1 grammar
3. Type-2 grammar
4. Type-3 grammar.

Type-0 grammar

It is also called as un-restricted grammar.

All productions are in the form of

$\alpha \rightarrow \beta$ where $\alpha, \beta \in (V \cup T)^*$

Ex:- $V = \{A, B, C\}$

$T = \{0, 1\}$

$S = \{A\}$

$<0> \rightarrow A$

$A \rightarrow AB1OAB$

$AB \rightarrow 0$

$AB012 \rightarrow \epsilon$

Type-1 grammar

It is also referred as Context Sensitive grammar (CSG)

→ In CSG, every Production in the form of

$\alpha \rightarrow \beta$, where $\alpha, \beta \in (V \cup T)^*$

→ $|\alpha| \leq |\beta|$ (restriction)

Ex:- $AB \rightarrow A10$

$A \rightarrow B$

$A \rightarrow 0$

$B0 \rightarrow AX$ (because $\alpha > \beta$)

Type-2 Goramma

→ It is also referred as context free grammar. (CFG)

→ In CFG, every Production in the form like

$A \rightarrow \alpha$, $\alpha \in (V \cup T)^*$

Only one non-terminal on L.H.S.

Ex:- $A \rightarrow B$

$A \rightarrow \epsilon B$

$\epsilon \rightarrow 0X$

Type-3 Grammar

→ It is also referred as regular grammar.

→ The Production are in the form of

$A \rightarrow a$ (terminal)

$A \rightarrow aB$ (terminal followed by non-terminal)

Derivation

→ The process of deriving the string is called derivation.

⇒ Always string derivation starts from start symbol S.

⇒ During string derivation if any non-terminals replace with suitable production.

⇒ Based on the selection of non-terminals, there are 2 types of derivations.

1. Left Most Derivation (LMD)
2. Right Most Derivation (RMD)

Ex:- $A \rightarrow BC$

$B \rightarrow 0B \mid 1B \mid 1$

$C \rightarrow 10$

LMD

$A \Rightarrow BC$
$\Rightarrow 1BC$
$\Rightarrow 11C$
$\Rightarrow 1110$

RMD

$A \Rightarrow BC$
$\Rightarrow B10$
$\Rightarrow 1B10$
$\Rightarrow 1110$

① Consider the grammar $S \rightarrow asa \mid bSb \mid \epsilon$ and define the language.

1. $S \rightarrow \epsilon$
2. $S \rightarrow asa$
   $\Rightarrow aa$ {aa}
3. $S \rightarrow bSb$
   $\Rightarrow bb$ {bb}
4. $S \rightarrow aSa$
   $\Rightarrow abSba$
   $\Rightarrow ababa$
   $\rightarrow abaaba$
5. $S \rightarrow aSa$
   $\Rightarrow abSba$
   $\rightarrow abba$

$S \rightarrow bSb$
$\Rightarrow bbSbb$
$\Rightarrow bbaSabb$
$\Rightarrow bbaabb$

∴ The given string is accepting "even length palindrome"

② Consider the grammar and Find the language.

$S \to aSb$

$S \to \epsilon$

1. $S \to aSb$
   $\to ab$ {ab}

$S \to aSb$
$\Rightarrow aaSbb$
$\Rightarrow aabb$

∴ Equal no. of a's and b's.

$L = \{a^m b^n \mid m >= 0, n >= 0\}$

③ $S \to aSb$

$S \to ab$

$S \Rightarrow aSb$
$\to aabb$

$S \Rightarrow aSb$
$\Rightarrow aaSbb$
$\Rightarrow aaabbb$.

④ $S \to aB \mid bA$

$A \to a \mid aS \mid bAA$

$B \to b \mid bS \mid aBB$

1. $S \Rightarrow aB$
   $\Rightarrow ab$

$S \Rightarrow aB$
$\Rightarrow abS$
$\to abaB$
$\Rightarrow ababb$

2. $S \Rightarrow bA$
   $\Rightarrow ba$

$\to bA$
$\Rightarrow baS$
$\to baaB$
$\Rightarrow baab$

$\Rightarrow bA$
$\Rightarrow bbAA$
$\Rightarrow bbaA$
$\Rightarrow bbaa$

∴ Equal no. of a's & b's $L = \{w \mid n_a(w) = n_b(w)\}$

(5)

$S \to 0A$

$A \to 0AB \mid 0A \mid 1$

$B \to 1$

$S \Rightarrow 0A$
$\Rightarrow 00AB$
$\Rightarrow 001B$
$\Rightarrow 0011$

$S \Rightarrow 0A$
$\Rightarrow 00A$
$\Rightarrow 000AB$
$\Rightarrow 0001B$
$\Rightarrow 00011$

$L = \{01, 001, 0001, 00011, \ldots\}$

$L = \{0^m 1^n \mid m >= n\}$

(6) $S \to aCa$

$C \to aCa \mid b$

$S \to aCa$
$\Rightarrow aaCaa$
$\Rightarrow aabaa$

$\Rightarrow aCa$
aba

$S \Rightarrow aCa$
$\Rightarrow aaCaa$
$\Rightarrow aaaCaaa$
$\Rightarrow aaabaaa$

$L = \{a^n b a^n \mid n \geq 1\}$

(7) $S \to aAB$

$A \to bBb$

$B \to A \mid \epsilon$

$S \to aAB$
$\Rightarrow abBbB$
$\Rightarrow abAbB$
$\Rightarrow abbBbbB$
$\Rightarrow abbbbbB$
$\Rightarrow abbbbbb$

$S \Rightarrow aAB$
$\Rightarrow abBbB$
$\Rightarrow abbbB$
$\Rightarrow abb$

$\Rightarrow aAB$
$\Rightarrow abBbB$
$\Rightarrow abAbB$
$\Rightarrow abbBbbB$
$\Rightarrow abbbbb$

$\therefore L = \{a.b^{2n} \mid n \geq 1\}$

(8) $S \to 0A \mid \epsilon$

$A \to 1S$

$S \Rightarrow 0A$
$\Rightarrow 01S$
$\Rightarrow 010A$
$\Rightarrow 0101S \Rightarrow 0101$

$\therefore L = \{(01)^n \mid n >= 0\}$

Construction of Context Free Grammar

① Construct CFG for the language $L = \{0^n 1^n \mid n \geq 0\}$

$\Rightarrow T = \{0,1\}$, $L = \{\epsilon, 01, 0011, 000111, \ldots\}$

Sol:- $S \to \epsilon$

$S \to 0S1$ $\therefore S \to \epsilon \mid 0S1$

② Construct CFG, $L = \{a^n . b^n \mid n \geq 1\}$

$T = \{0,1\}$, $L = \{ab, aabb, aaabbb \ldots\}$

$S \to ab \mid asb$

③ Construct CFG for $L = \{0^n \mid n \geq 0\}$

$S \to \epsilon \mid 0S$ $L = \{\epsilon, 0, 00, 000, \ldots\}$

(or) $S \to 0 \mid 0S$ $(n \geq 1)$

④ Construct CFG for $(0+1)^*$ and derive the string 011010

$L = \{\epsilon, 0, 1, 01, 10, 11, 00, 101, \ldots\}$

$S \to \epsilon$

$S \to 0 \mid 1 \mid 0S1 \mid 1S0 \mid \epsilon$

$S \to 0S \mid 1S \mid \epsilon$

011010

$S \to 0S$
$\Rightarrow 01S$
$\Rightarrow 011S$
$\Rightarrow 0110S$
$\Rightarrow 01101S$
$\Rightarrow 011010$

⑤ Construct CFG, $L = \{a^m . b^n \mid m \geq 0, n \geq 0\}$

$L = \{\epsilon, a, ab, aab, aabb, \ldots\}$

Sol:-

$a^m, m \geq 0$ $\quad A \rightarrow \epsilon | aA$

$b^n, n \geq 0$ $\quad B \rightarrow bB | \epsilon$

$S \rightarrow AB$

$A \rightarrow aA | \epsilon$

$B \rightarrow bB | \epsilon$

aabbbb

$S \rightarrow AB$

$\Rightarrow aAB$

$\Rightarrow aaAB$

$\Rightarrow aaB$

$\Rightarrow aabB$

$\Rightarrow aabbB$

$\Rightarrow aabbbB$

$\Rightarrow aabbbbB$

$\Rightarrow aabbbb$

$\therefore$ The following string is derived/accepting.

⑥ Construct CFG, $L = \{a^m b^n \mid m \geq 0, n \geq 1\}$

$a^m, m \geq 0$, $A \rightarrow \epsilon | aA$

$B \rightarrow b | bB$

$S \rightarrow AB$

⑦ Construct CFG for language which accepts palindromes over the alphabet.

$S \rightarrow aSa | bSb | a | b | \epsilon$

for even length $S \rightarrow aSa | bSb | \epsilon$

odd length $S \rightarrow aSa | bSb | a | b$

⑧ Construct a CFG for $L = \{WCW^R \mid \text{where } W \in \{a, b\}\}$

$S \rightarrow aSa | bSb | c$

(9) Construct a CFG, for $L = \{ a^i . b^j . c^k \mid \text{where } j = i+k \}$

$A \rightarrow \epsilon \mid aA$

$a^i . b^j . c^k$

$\Rightarrow a^i b^{i+k} . c^k$

$\Rightarrow \underbrace{a^i b^i}_{A} \underbrace{b^k c^k}_{B}$

$S \rightarrow AB$

$A \rightarrow aAb \mid \epsilon$

$B \rightarrow bBc \mid \epsilon$

(10) Construct a CFG, generating all strings over $\{a, b\}$ consisting of equal no. of a's and b's.

$L = \{ \epsilon, ab, abab, baba, \ldots \}$

$S \rightarrow \epsilon \mid asbs \mid bsas$

$S \Rightarrow bsas$

$\Rightarrow baasbs$

$\Rightarrow baabs$

$\rightarrow baab$

(11) Constuct CFG, $L = \{ a^n b^{2n} \mid n \geq 0 \}$

$L = \{ \epsilon, abb, aabbbb, \ldots \}$

$S \rightarrow \epsilon$

$\therefore S \rightarrow asbb \mid \epsilon$

(12) Design CFG, $L = \{ a^{2n} b^m \mid n \geq 0, m \geq 0 \}$

$S \rightarrow AB$

$A \rightarrow aaA \mid \epsilon$

$B \rightarrow bB \mid \epsilon$

→ Construct CFG for a language of strings containing exactly one 'A' over an alphabet a, b.

$L = \{\epsilon, ab, bab, bba, \ldots\}$

$R.E = b^* a b^*$

$S \rightarrow BaB$

$B \rightarrow bB \mid \epsilon$

→ Design CFG for set of strings contains atleast three a's.

$R.E = (a+b)^* a (a+b)^* a (a+b)^* a (a+b)^*$

$S \rightarrow BaBaBaB$

$B \rightarrow aB \mid bB \mid \epsilon$

→ Design CFG for accepting 3 consecutive 0's over an alphabet.

$R.E = (0+1)^* 000 (0+1)^*$

$S \rightarrow B00B$

$B \rightarrow 0B \mid 1B \mid \epsilon$

→ Design CFG for $0^* 1 (0+1)^*$

~~$S \rightarrow X(A) \mid B$~~ $S \rightarrow A1B$

$A \rightarrow 0A \mid \epsilon$

$B \rightarrow 1B \mid \epsilon \mid 0B$

→ Construct a CFG, to obtaine balanced strings.

$L = \{\epsilon, (), (()), ()(), (()()), \ldots\}$

$S \rightarrow \epsilon$

$S \rightarrow (S) \mid SS$

$S \rightarrow (S) \mid SS \mid \epsilon$

→ Design CFG for $L = \{w \mid$ where length is odd, over i/p alphabet & always middle symbol in the every string is a.?

$L = \{a, aaa, bbabb, bab, ----\}$

$S \to a$

$S \to asb \mid bsb \mid bsa \mid asa$

$\therefore S \to asb \mid bsb \mid asa \mid bsa \mid a$

String acceptance. bababbb

→ $S \to bsb$

$\Rightarrow basbb$

$\Rightarrow babsbbb$

$\Rightarrow baba bbb$

→ Design the Regular expression for the context free grammar.

$S \to A \mid B$

$A \to 00A \mid \epsilon$

$B \to 000B \mid \epsilon$

Sol:- $S \to A \mid B$

$\Rightarrow 00A \mid B$

$\Rightarrow 0000A \mid B$

$\Rightarrow 00001B$

$\Rightarrow 00001000B$

$\Rightarrow 00001000000B$

$\Rightarrow 00001000000$

$\therefore (00)^* 1 (000)^*$ is the Regular Expression for given CFG.

→ Construct CFG to generate unequal number of a's & b's.

$S \rightarrow A$

$A \rightarrow CAa \mid CaC$

$C \rightarrow acbc \mid bcac \mid \epsilon$

Derivation Tree / Parse tree

* The graphical / Pictorial representation of a derivation is called derivation tree / Parse tree.
* In all derivation tree a Start Symbol will be act as root node.
* All the non-terminals will be acts as interior nodes.
* All the terminals will be act as leaf nodes.

Two types of Parse trees.

1. Left most Derivation tree
2. Right most Derivation tree.

① Consider the grammar & Construct LMD tree & RMD tree.

$S \rightarrow SbS/a$, for the string ababa.

$S \rightarrow SbS$

$\Rightarrow abS \quad (S \rightarrow a)$

$\Rightarrow abSbS \quad (S \rightarrow SbS)$

$\Rightarrow ababS$

$\Rightarrow ababa \quad (S \rightarrow a)$.

$S \rightarrow SbS$

$\Rightarrow Sba \quad (S \rightarrow a)$

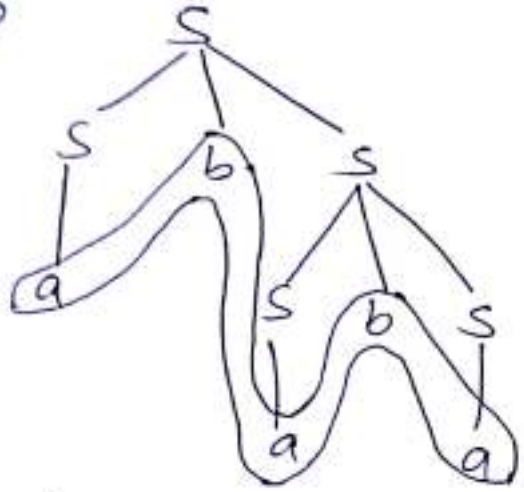
$\Rightarrow SbSba \quad (S \rightarrow SbS)$

$\Rightarrow Sbaba \quad (S \rightarrow a)$

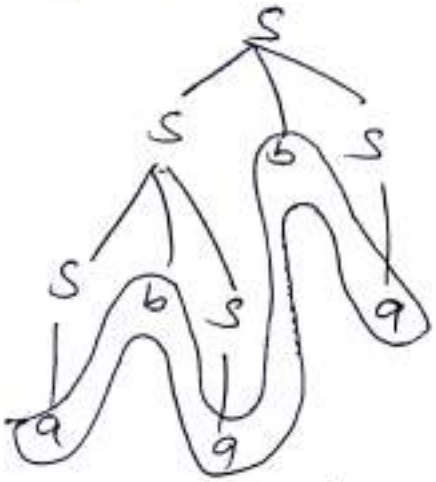
$\Rightarrow ababa \quad (S \rightarrow a)$

Yield of the tree T1 is obtained by concatenating leaf nodes from left to right in the derivation tree is called yield.

LMD



RMD



② Consider the CFG, $E \rightarrow E+E \mid E*E \mid id$ and derive the following string id+id*id & Construct LMDT & RMDT.

LMD.

$E \rightarrow E+E$

$\Rightarrow id+E$

$\Rightarrow id+E*E$

$\Rightarrow id+id*E$

$\Rightarrow id+id*id$

RMD

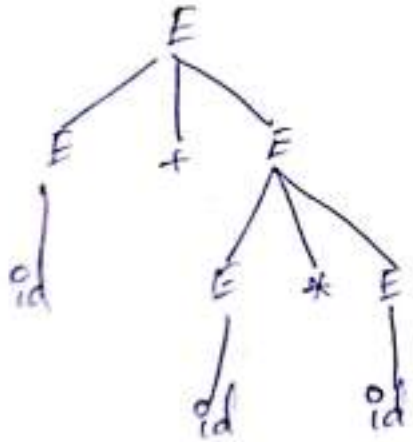
$E \rightarrow E+E$

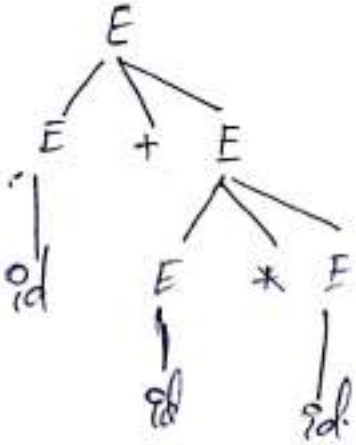
$\Rightarrow E+E*E$

$\Rightarrow E+E*id$

$\Rightarrow E+id*id$

$\Rightarrow id+id*id$





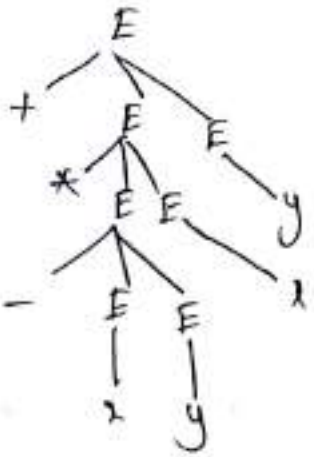
③ Consider the grammar, $E \rightarrow +EE/*EE/-EE/x/y$
Find the leftmost and rightmost derivation for the string $+*-xyxy$ and design Parse tree.

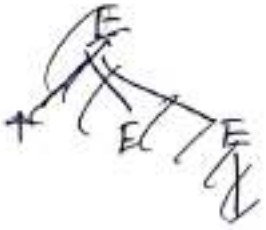
LMD.

$E \rightarrow +EE$
$\rightarrow +*EEE \quad (E \rightarrow *EE)$
$\Rightarrow +*-EEEE \quad (E \rightarrow -EE)$
$\Rightarrow +*-xEEE \quad (E \rightarrow x)$
$\Rightarrow +*-xyEE \quad (E \rightarrow y)$
$\Rightarrow +*-xyxE \quad (E \rightarrow x)$
$\rightarrow +*-xyxy \quad (E \rightarrow y)$

RMD

$E \rightarrow +EE$
$E \Rightarrow +Ey \quad (E \rightarrow y)$
$\Rightarrow +*EEy \quad (E \rightarrow *EE)$
$\Rightarrow +*Exy \quad (E \rightarrow x)$
$\Rightarrow +*-EExy \quad (E \rightarrow -EE)$
$\Rightarrow +*-Eyxy \quad (E \rightarrow y)$
$\Rightarrow +*-xyxy \quad (E \rightarrow x)$





⇒ Write a CFG for the language $L = \{wcw^R / w \in (a,b)^*\}$

Let G be CFG for language.

$L = \{wcw^R / w \in (a,b)^*\}$

$G = (V, T, P, S)$

where $V = \{S\}$, $T = \{a, b\}$

P is Productions given by

$S \rightarrow aSa$
$S \rightarrow bSb$
$S \rightarrow c$

for example abbcbba can be derived or not. here

$w = abb$ & $w^R = bba$.

$S \Rightarrow aSa \quad (S \to aSa)$
$\to abSba \quad (S \to bSb)$
$\to abbSbba \quad (S \to bSb)$
$\Rightarrow abbcbba \quad (S \to c)$

∴ The string abbcbba can be derived from given CFG.

⇒ Design a CFG for the language $L = \{a^n b^m : n \le m+3\}$

Let us assume $n = m+3$, then add more b's. Assume

CFG, $G_1 = (V, T, P, S)$

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

$T = \{a, b, \epsilon\}$

$P \Rightarrow S \to aaaA$
$A \to aAb \mid B$
$B \to Bb \mid \epsilon$

But, when we analyse this solution, then it is found incomplete since it creates atleast three a's. Let us take, $n = 1, 2,$

$S \to \epsilon \mid aA \mid aaA$

So, the solution is

$S \to \epsilon \mid aA \mid aaA$
$A \to aAb \mid B$
$B \to Bb \mid \epsilon$.

⇒ Design a CFG for the language $L\{0^n 1^n \mid n \ge 0) \cup (1^n 0^n \mid n > 0)\}$

Assume L as $L = L_1 \cup L_2$

where $L_1 = \{0^n 1^n \mid n \ge 0\}$
$L_2 = \{1^n 0^n \mid n \ge 0\}$

Let say G_1 be CFG for the language L_1

$G_1 = (V, T, P, S)$

$V = \{S_1\}$, $S = \{S_1\}$, $T = \{0, 1\}$

$P \Rightarrow S_1 \rightarrow 0S1 \mid \epsilon$.

Let say G_2 be CFG for the language L_2

$G_2 = (V, T, P, S)$

$V = \{S_2\}$, $S = \{S_2\}$, $T = \{0, 1\}$

$P \Rightarrow S_2 \rightarrow 1S_20 \mid \epsilon$.

$\therefore L = L_1 \cup L_2$

G be CFG for language L with starting non-terminal S.

Then the productions are:

$S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow 0S_11 \mid \epsilon$

$S_2 \rightarrow 1S_20 \mid \epsilon$.

$\Rightarrow$ Let G be CFG, $S \rightarrow bA \mid aA$

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

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

Find the string bbaababa find (i) Left most derivation (ii) Rightmost derivation (iii) Parse tree.

Sol: (i) Leftmost derivation for input $w = bbaababa$.

$S \Rightarrow bB$

$\Rightarrow bbBB$

$\Rightarrow bbaB$

$\Rightarrow bbaaS$

$\Rightarrow bbaabB$

$\Rightarrow bbaababS \Rightarrow bbaababB \Rightarrow bbaababa$

(ii) Right most derivation

$S \Rightarrow bB$

$S \Rightarrow bbBB$

$\Rightarrow bbBaS$

$\Rightarrow bbBabB$

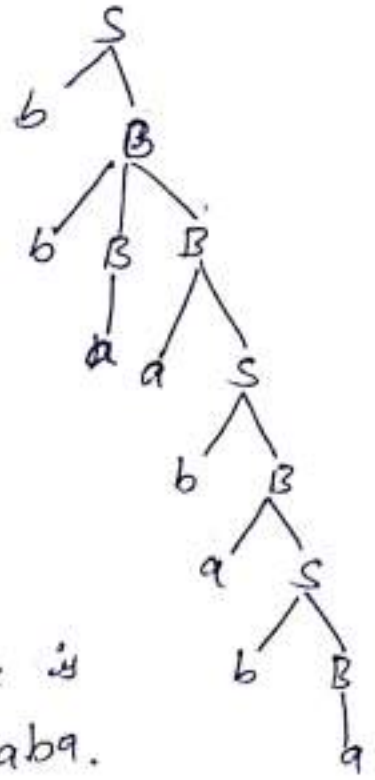
$\Rightarrow bbBabaS$

$\Rightarrow bbBababB$

$\Rightarrow bbBababa$

$\Rightarrow bbaababa.$

(iii) Parse tree.



(iv) Parse tree.

The Yield of Parse tree is bbaababa.

Parse tree Properties:

(a) Derivation tree is called Parse tree.

(1) The root node is always a node indicating start symbol.

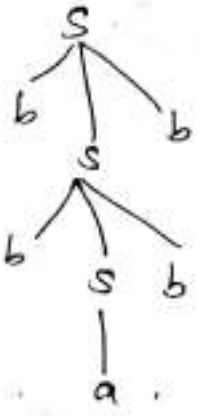
(2) The derivation is read from left to right.

(3) The leaf nodes are always terminal nodes.

(4) The interior nodes are always the non-terminal nodes.

For example:-

$S \rightarrow bSb \mid a \mid b$ is a Production rule. The S is a start symbol.



The above derivation tree drawn for deriving a string bbabb. By simply reading the leaf nodes we can obtain the derived string.

⇒ Construct the derivation tree for the string aabbabba from the CFG given by $S \to aB \mid bA$

$A \to a \mid aS \mid bAA$

$B \to b \mid bS \mid aBB$

To draw a tree, first we will obtain a derivation for the string aabbabba.

$S \Rightarrow aB$ $(S \to aB)$

$\Rightarrow aaBB$ $(S \to aBB)$

$\Rightarrow aabSB$ $(B \to bS)$

$\Rightarrow aabbAB$ $(S \to bA)$

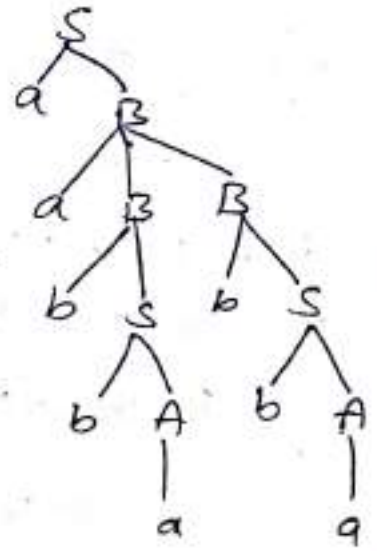
$\Rightarrow aabbaB$ $(A \to a)$

$\Rightarrow aabbabS$ $(B \to bS)$

$\Rightarrow aabbabbA$ $(S \to bA)$

$\Rightarrow aabbabba$ $(A \to a)$

derivation tree



Sentential Form

Let $G=(V,T,P,S)$ be the context free grammar. If $A \to \beta$ is a production of P and α and γ are any strings from non-terminals & terminals i.e. in $(VUT)^*$ then $\alpha A \gamma \to \alpha\beta\gamma$

Suppose $a_1, a_2, a_3, \ldots a_m$ are strings in $(VUT)^*$, $m \geq 1$

$$a_1 \Rightarrow a_2$$
$$a_2 \Rightarrow a_3$$
$$\vdots$$
$$a_{m-1} \Rightarrow a_m$$

Then we can say that $a_1 \overset{*}{\underset{G}{\Rightarrow}} a_m$ is called Sentential form of a language.

Ambiguity in Grammars and Languages

Ambiguous grammar is a grammar using which if it is possible to construct more than one parse trees for the same string.

For example: The CFG given by $G=(V,T,P,S)$

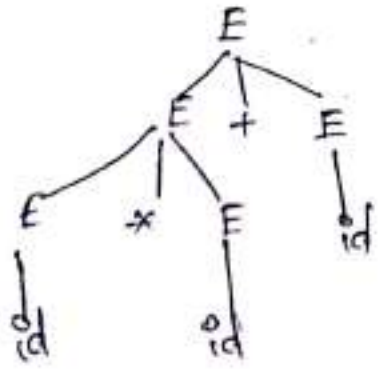
Where $V=\{E\}$

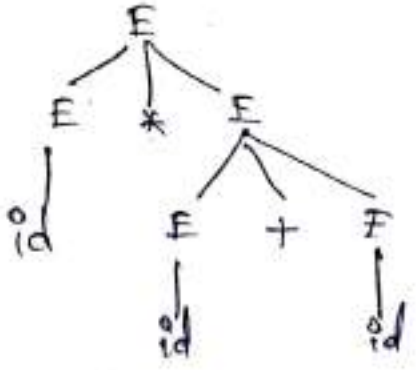
$T=\{id\}$

$P=\{ E \to E+E$
$E \to E*E$
$E \to id\}$

$S=\{E\}$

Now if the string is $id*id+id$ then we can draw the two different parse trees indicating that $id*id+id$.





① $S \rightarrow iCtS \mid iCtSeS \mid a$

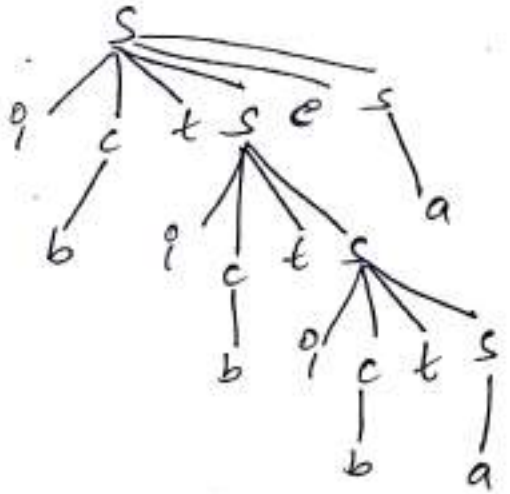
$C \rightarrow b$

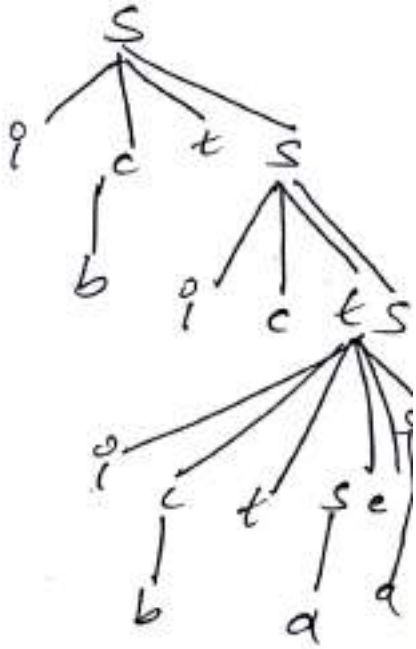
String is ibtibtibtaea. whether this grammar is ambigous or not..

Given string is ibtibtibtaea

$S \rightarrow iCtSeS$
$\Rightarrow ibtSeS \ (C \rightarrow b)$
$\Rightarrow ibtiCtSeS \ (S \rightarrow iCtS)$
$\Rightarrow ibtibtSeS \ (C \rightarrow b)$
$\Rightarrow ibtibtiCtSeS \ (S \rightarrow iCtS)$
$\Rightarrow ibtibtibtSeS \ (C \rightarrow b)$
$\Rightarrow ibtibtibtaeS \ (S \rightarrow a)$
$\Rightarrow ibtibtibtaea \ (S \rightarrow a)$

$S \Rightarrow iCtS$
$\Rightarrow ibtS \ (C \rightarrow b)$
$\Rightarrow ibtiCtS \ (S \rightarrow iCtS)$
$\Rightarrow ibtibtS \ (C \rightarrow b)$
$\Rightarrow ibtibtiCtSeS \ (S \rightarrow iCtS)$
$\Rightarrow ibtibtibtSeS \ (C \rightarrow b)$
$\Rightarrow ibtibtibtaeS \ (S \rightarrow a)$
$\Rightarrow ibtibtibtaea \ (S \rightarrow a)$





∴ It is an ambigous.

(2) $S \to a \mid aAb \mid abSb$

$A \to aAAb \mid bS$

String is aababbabb.

Sol:- $S \to aAb$

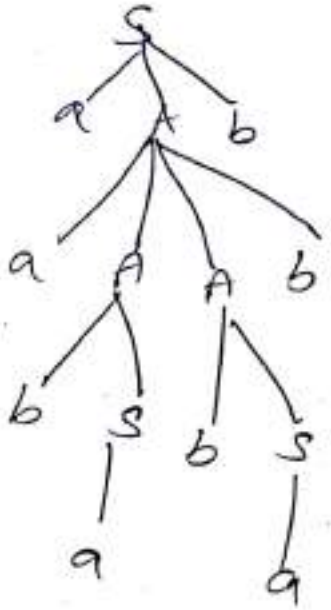
$S \Rightarrow aaAAbb$ ($A \to aAAb$)

$\Rightarrow aabSAbb$ ($A \to bS$)

$\Rightarrow aababAbb$ ($S \to a$)

$\Rightarrow aababSbb$ ($A \to bS$)

$\Rightarrow aababbabb$ ($S \to a$).



∴ It is unambiguous.

Applications of Context-Free Grammars

→ Context-free grammars were originally conceived by Noam Chomsky as a way to describe natural languages.

→ CFGs were used in Computer Science because of its capability of defining recursive rules.

→ This recursive way of defining rules and definitions has two major applications (among others) in CS:

1. Parsers:- A component of Compiler that discovers the structure of the source program and represents that structure by a parse.

2. DTD's in XML:- Describes the allowable tags and the way in which these tags may be nested.

Parsers:- Many of a programming language have a structure that may be described by regular expressions

Let us take an example of balanced paranthesis. In this, we match some left paranthesis against a right paranthesis that appears immediatly to its right remove both of them, and repeat if we eliminate all the parentheses then it is balanced. If we cannot match paranthesis then it is unbalanced.

Examples:- (), (()), (), ((())), ϵ are balanced paranthesis.

)(and (() are unbalanced paranthesis.

A grammar $G_{bal}(\{B\}, \{(,)\}, P, B\}$ generates all and only the strings of balanced paranthesis, where P consist of the productions.

$$B \rightarrow BB \mid (B) \mid \epsilon$$

Let us take the production $B \rightarrow BB$ if the concatenation of two strings of balanced paranthesis is balanced. Independently also we match the paranthesis.

The next production i.e. $B \rightarrow (B)$, if we place a pair of paranthesis around a balanced string, then the result is balanced. The third production $B \rightarrow \epsilon$ says that the empty string is balanced.

So, the Grammar G_{bal} generates all strings of balanced parenthesis. Programming languages consist of more than parenthesis, but parentheses are an essential part of arithmetic & Conditional expression.

There are numerous aspects of a typical programming language that be have like balanced parentheses. There will always parentheses themselves, in expressions of all types. Beginings and endings of code blocks, such as begin and end in pascal, or the curly braces {.....} of C, are examples. Curly braces appears in a C program must form a balanced sequences, with { in place of the left parentheses and } in place of the right parentheses.

There is a related pattern like parentheses i.e. if and else in C program. An if clause can appear unbalanced by any else clause, or it may be balanced by a matching else clause. A grammar that generates the possible sequence of if and else is:

$S \rightarrow \epsilon | SS | iS | iSe$

$i \Rightarrow$ if and $e \Rightarrow$ else

For the given grammar, the possible sequences are iei ieie, iei generated by the grammar. The sequences ei ieie, iei generated by the grammar. and ieeii are illegal & not generated by the grammar.

Let us take iecii, we derive this input string

$S \rightarrow iSe$

$\rightarrow iiSee$

But it is not possible to get that string becauses it is not derivable.

Let us take the string ieeeie. we will see it is the language or not. I^st e is matched with the i to its immediate left, if we remove this, we get;

ieeeie

ieeie

eie.

Since, all 'e's have not been exhausted. we look for the next. Since it has no i to its left, the test fails.

Let us take another example, iie matching the first e to i immediately to the left. So, remove ie then, i remains.

iie → i

The structure of 'C' program is -

```
if (condition)
{
-----
if (condition) statement;
else statement;
-----
}
```

Markup language

Markup language is a family of 'languages'. The Strings in the languages are documents with certain marks (called tags) in them. Tags is defined as semantics of various strings within the document HTML (Hyper text markup language) is one of markup language. This HTML has two functions

1. Creating links between documents
2. Describing the document.

Let us take an example of statements:

The things I hate:

1. Moldy bread
2. People who drive too slow in the fast lane.

The statements can be written in HTML. It consists of ordinary text with tags. matching tags are of the form <x> and </x> for some string. The text should be exphasized represented as <EM> and </EM>. Emphasized means put in italics or another appropriate font. The other matching tags <OL> and </OL> indicating an ordered list. i.e. an enumeration of list items.

The HTML source is:

```
<P>The things I <EM>hate </EM>
<OL>
<LI> Moldy bread
<LI> People who drive too slow in the fast lane
</OL>
```

There are two unmatched tags <p> and <LI> ie, Paragraph and List Items. Tags be matched by </P> & </LI> at the ends of paragraphs and list items but it does not require the matching. The parts of an HTML Document for a grammar are :

1. char is any string consisting of a single character that is leagal in HTML text. Note that blanks are included as characters.

Char → a | A | .....

2. Text is any string of characters that has no tags.

Text → ∈ | Char Text

Example : Text element as " Moldy bread "

3. Doc represents documents which are sequence of elements.

Doc → ∈ | element Doc.

4. Element is either a Text string, or a pair of matching tags and the document between them, or an unmatched tag followed by a document.

Element → Text |
<EM> Doc </EM> |
<p> Doc |
<OL> List </OL> | ...

5. List Item is the <LI> tag followed by a document, which is a single list item.

List Item → <LI> Doc.

6. List is a sequence of zero or more list items.

List → ∈ | List Item List.

XML and DTD

Another important markup language is XML (Extensible markup language). In this, CFG plays an important role as a part of the process of using that language.

The purpose of XML is not to describe the formatting of the document that is the job for HTML. XML is used to describe semantics of the text.

Example; <ADDR> Road No. 2 </ADDR> In XML, tags are represented as phrase which describes the address.

<ADDR> is the tag means address of a house.

The emerging XML standard for sharing information through web documents has a notation, called the DTD. (Document Type Definition, for describing the structure of such documents, through the nesting of semantic tags within the document. DTD is in Esence a CFG whose language is a class of related documents i.e., variables and Productions.

Form of a DTD is:

<! DOCTYPE Name-of-DTD [list of element definitions] >

An element definition has the form;

<! ELEMENT Element name (description of the element) >

Element descriptions are regular Expressions. The basis of these expressions are:

1. Elements of one type can appear within elements of another type, just as in HTML we might find emphasized text within a list.

2. Special term #PCDATA standing for any text that does not involve XML tags. It is a term describes the Variable Text.

The operators which are used in DTD are:

1. | → Union
2. , → Concatenation.
3. * → Zero or more occurrences of
4. + → One or more occurrences of
5. ? → Zero or one occurrence of

→ Parenthesis many group operators to their arguments

→ Usual precedence of regular Expression operators applies

Let us take an example of Computer vendors get together to Create a DTD that they can use to publish, on the web, descriptions of a PC's that they currently sell. Each description of a PC will have a model number & details about the features of the model Ex:- The amount of RAM, number and size of disks and so on.

```
<!DOCTYPE PCSpecs[
<!ELEMENT PCS (PC*)>
<!ELEMENT PC (MODEL, PRICE, PROCESSOR, RAM,
                                    DISK+)>
```

```
<!ELEMENT MODEL (#PCDATA)>
<!ELEMENT PRICE (#PCDATA)>
<!ELEMENT PROCESSOR (MANF, MODEL, SPEED)>
<!ELEMENT MANF (#PCDATA)>
<!ELEMENT MODEL (#PCDATA)>
<!ELEMENT SPEED (#PCDATA)>
<!ELEMENT RAM (#PCDATA)>
<!ELEMENT DISK (HARDDISK | CD | DVD)>
<!ELEMENT HARDDISK (MANF, MODEL, SIZE)>
<!ELEMENT SIZE (#PCDATA)>
<!ELEMENT CD (#SPEED)>
<!ELEMENT DVD (#SPEED)>
]>
```

The above is a DTD of name PcSpecs. The first element is the start symbol of CFG, PCS (list of pc specifications)

XML document to the above DTD.

```
<PCS>
<PC>
<MODEL> 4560 </MODEL>
<PRICE> $2295 </PRICE>
<PROCESSOR>
<MANF> Intel </MANF>
<MODEL> Pentium </MODEL>
<SPEED> 800 MHz </SPEED>
</PROCESSOR>
```

<RAM> 256 </RAM>
<DISK> <HARDDISK>
<MANF> Maxtor </MANF>
<MODEL> Diamond </MODEL>
</HARDDISK> </DISK>
<SIZE> 30.5 Gb </SIZE>
<DISK> <CD>
<SPEED> 32X </SPEED>
</CD> </DISK> </PC>
<PC>
.....
</PC>
</PCS>.

The rules for the elements in DTD like productions of CFG.

<!ELEMENT PROCESSOR (MANF, MODEL, SPEED)>

Is analogous to the Production

Processor → Manf model speed

<!ELEMENT DISK (HARDDISK | CD | DVD)>

This can be written in CFG is

Disk → Harddisk | cd | Dvd

<!ELEMENT PC (MODEL, PRICE, PROCESSOR, RAM, DISK+)>

PC → Model Price Processor Ram Disks

Disks → Disk / Disk Disks.

Push Down Automata:-

Push Down Automaton is an NFA with ϵ-transitions permitted and one additional capability a stack on which it can store a string of 'stack symbols'.

The presence of a stack means the PDA can have an infinite amount of information. The PDA can access the information on its stack in a first-in first out way.

PDA recognize all and only the context free languages.

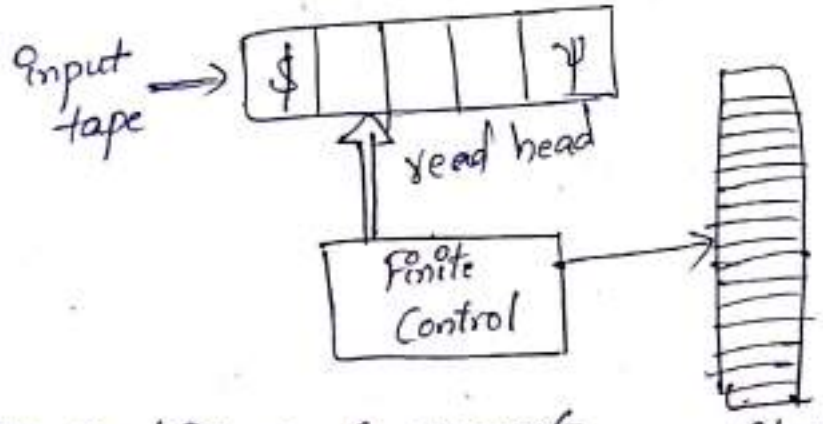
→ It will accept only regular language.

→ No auxilary memory available. ie, unable to remember previous Symbols.

→ It moves only in forward direction.

→ Write operation is not available.

Model of PDA



PDA Contains 4 Components

1) Input tape :- It contains set of cell, each cell contains 1 i/p symbol with 2 end markers.

2) Read Head :- Initially it points to start/first cell after reading, it moves in forward manner.

3) Stack :- It is a temporary storage area to store i/p symbols. Initially stack is empty.

4) Control unit :- It is going to define the behaviour of PDA by maintaining state & stack.
It points to both i/p tape & stack.

Definition :- A PDA contains 7 tuples.

$$M = (Q, \Sigma, S, \delta, q_0, z_0, F)$$

Q — set of finite states
Σ — i/p alphabet
S — set of stack symbols.
q_0 — initial state
z_0 — Initial stack symbol
F — set of Final states
δ — Transition function.

$$\delta : Q \times \{\Sigma \cup \{\epsilon\}\} \times S \rightarrow Q \times S^*$$

(i) $\delta(q_0, a, z_0) \rightarrow (q_0, a z_0)$ — pushing 'a' into stack no change in state

(ii) $\delta(q_1, b, a) \rightarrow (q_2, a)$ — Nothing, no change in the stack content

(iii) $\delta(q_1, a, b) \rightarrow (q_2, \epsilon)$ — pop 'b' from the stack, & state changes to q_2

(iv) $\delta(q_0, \epsilon, b) = (q_1, b)$ — without taking any input s/m moves from one state to another.

Instantaneous Description of PDA (ID) :-

* Consider a PDA, $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0 F)$ then the configuration of a particular stage can be defined by instantaneous description.

→ It contains three components (q, x, α)

q – is a state $\Rightarrow q \in Q$

→ ID's are useful for checking string acceptance.

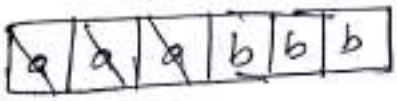
* Construct PDA for the language $L = \{a^n b^n / n \geq 1\}$, $\Sigma = \{a, b\}$

$\Gamma = \{Z_0\}$

→ Initially push all 'a's into the stack.

→ wherever b occurs change the state and pop 'a' from the stack.

→ Repeat above step untill stack is empty.



$\delta(q_0, a, Z_0) = (q_0, aZ_0)$

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

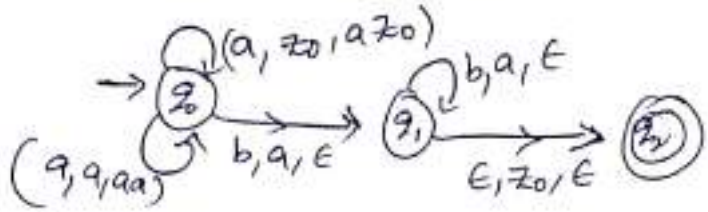
$\delta(q_0, b, a) = (q_1, \epsilon)$

$\delta(q_1, b, a) = (q_1, \epsilon)$

$\delta(q_1, \epsilon, Z_0) = (q_2, \epsilon)$

$\delta(q_1, \epsilon, Z_0) = (q_2, Z_0)$

∴ String accepted.



Consider String aaaabbb

$\delta(q_0, aaaabbbb, z_0)$

$\vdash (q_0, aaabbbb, az_0)$

$\vdash (q_0, aabbbb, aaz_0)$

$\vdash (q_0, abbbb, aaaz_0)$

$\vdash (q_0, bbbb, aaaaz_0)$

$\vdash (q_1, bbb, aaaz_0)$

$\vdash (q_1, bb, aaz_0)$

$\vdash (q_1, b, az_0)$

$\vdash (q_1, \epsilon, z_0)$

$\delta(q_2, \epsilon, \epsilon)$ ∴ String is accepted.

→ Construct a PDA that accepts the following languages $L = \{w/w$ is in $(a+b)^*$ and number of a's equal to number of b's$\}$. write the instantaneous description for the string 'aababb'. (or): $L = \{w/w \in (a+b)^*$ and $n_a(w) = n_b(w)\}$

Initially when Stack is empty then whatever we read either 'a' or 'b' we will simply push it onto the stack. Now, if we read 'a' and at the top of the stack if 'b' is Present then it will be erased by ε. Similarly if we read 'b' and top of the stack Contains 'a' then erase it by ε.

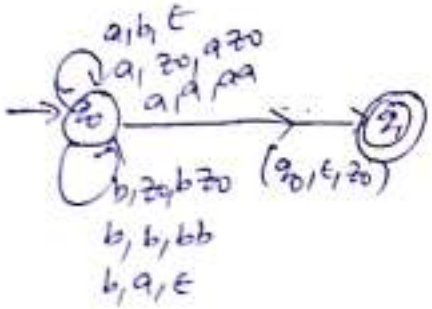
The Instantaneous description for this language as

$$\delta(q_0, a, z_0) = (q_0, az_0)$$
$$\delta(q_0, b, z_0) = (q_0, bz_0)$$
$$\delta(q_0, a, a) = (q_0, aa)$$
$$\delta(q_0, b, b) = (q_0, bb)$$
$$\delta(q_0, a, b) = (q_0, \epsilon)$$
$$\delta(q_0, b, a) = (q_0, \epsilon)$$
$$\delta(q_0, \epsilon, z_0) = (q_1, z_0)$$

Where q_0 is a start state and q_1 is a final state. when we reach to this state with ϵ input and having an empty stack. We move to accept / final state.

Given String is 'aababb'.

$$\delta(q_0, aababb, z_0) \vdash (q_0, ababb, az_0)$$
$$\vdash (q_0, babb, aaz_0)$$
$$\vdash (q_0, abb, az_0)$$
$$\vdash (q_0, bb, aaz_0)$$
$$\vdash (q_0, b, az_0)$$
$$\vdash (q_0, \epsilon, z_0)$$
$$\vdash (q_1, z_0)$$ Accept state.



→ Design a PDA that accepts a string of well formed parenthesis. The parenthesis is as (,), [,], {, }.

Initially i.e, in state q_0 either (, {, or [will be scanned and at that time the stack will be empty.

Then we will simply push it into stack.

$\therefore \delta(q_0, (, z_0) \vdash (q_1, (z_0)$

$\delta(q_0, [, z_0) \vdash (q_1, [z_0)$

$\delta(q_0, \{, z_0) \vdash (q_1, \{z_0)$

Further if we read more left parenthesis then we will simply push them onto the stack.

$\delta(q_1, (, () = (q_1, ((}$

$\delta(q_1, [, [) = (q_1, [[}$

$\delta(q_1, \{, \{) = (q_1, \{\{}$

$\delta(q_1, (, [) = (q_1, ([}$

$\delta(q_1, (, \{) = (q_1, (\{}$

$\delta(q_1, [, () = (q_1, [(}$

$\delta(q_1, \{, () = (q_1, \{(}$

$\delta(q_1, \{, [) = (q_1, \{[}$

$\delta(q_1, [, \{) = (q_1, [\{}$

Thus if we read any opening parenthesis we go on pushing it onto the stack. we read closing parenthesis or right parenthesis we start popping the stack contents.

$\therefore \delta(q_1,), () = (q_1, \epsilon)$

$\delta(q_1,], [) = (q_1, \epsilon)$

$\delta(q_1, \}, \{) = (q_1, \epsilon)$

-After reading the complete input string, will get ϵ to read but at the same time the contents of stack should be removed and it should simply contain z_0. PDA should enter the final state and that too in state q_0, so that all steps can be repeated if needed.

$\delta(q_0, (, z_0) = (q_1, (z_0)$.
$\delta(q_0, [, z_0) = (q_1, [z_0)$
$\delta(q_0, \{, z_0) = (q_1, \{z_0)$
$\delta(q_1, (, () = (q_1, (($
$\delta(q_1, [, [) = (q_1, [[$
$\delta(q_1, \{, \{) = (q_1, \{\{$
$\delta(q_1, (, [) = (q_1, ([$
$\delta(q_1, (, \{) = (q_1, (\{$
$\delta(q_1, [, () = (q_1, [($

$\delta(q_1, \}, () = (q_1, \{($
$\delta(q_1, \}, [) = (q_1, \{[$
$\delta(q_1,], [) = (q_1, [[$
$\delta(q_1,), () = (q_1, \epsilon)$
$\delta(q_1,], [) = (q_1, \epsilon)$
$\delta(q_1, \}, \{) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, z_0) = (q_0, z_0)$ //

The PDA $P = (\{q_0, q_1\}, \{(, [, \{,),], \}\}, \{[, (, \{, z_0\}, \delta, q_0, z_0, \{q_0\})$

Input string ({ } [ ]

$\delta(q_0, (\{ \} [ ], z_0) \vdash (q_1, \{ \} [ ], (z_0)$
$\vdash (q_1, \} [ ], \{ (z_0)$
$\vdash (q_1, [ ], (z_0)$
$\vdash (q_1,], [(z_0)$
$\vdash (q_1,), (z_0)$
$\vdash (q_1, \epsilon, z_0)$
$\therefore (q_0, z_0)$

Accept state

→ Construct PDA for the language $L = \{a^n b^{2n} \mid n \geq 1\}$

In this language 'n' number of a's should be followed by on number of b's. Hence we will apply a very simple logic and that is if we read single 'a' we will push two 'a's onto the stack. we read 'b' then for every single 'b' only one 'a' should get popped from the stack.

The ID can be as -

$$\delta(q_0, a, z_0) = (q_0, aaz_0)$$
$$\delta(q_0, a, a) = (q_0, aaa).$$

Now, when we read b we will change the state from q_0 to q_1 and start popping corresponding 'a'. Hence,

$$\delta(q_0, b, a) = (q_1, \epsilon)$$

This process of popping will be repeated unless all the symbols are read. Note that popping action occurs in state q_1 only.

$$\therefore \delta(q_1, b, a) = (q_1, \epsilon).$$

After reading all b's all the corresponding a's should get popped. Hence when we read ϵ as input symbol there should be nothing in the stack.

$$\delta(q_1, \epsilon, z_0) = (q_2, \epsilon)$$

PDA $P = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, \{q_2\})$

ID as, $\delta(q_0, a, z_0) = (q_0, aaz_0)$

$$\delta(q_0, a, a) = (q_0, aaa)$$
$$\delta(q_0, b, a) = (q_1, \epsilon)$$
$$\delta(q_1, b, a) = (q_1, \epsilon)$$
$$\delta(q_1, \epsilon, z_0) = (q_2, \epsilon)$$

PDA for some input string "aaabbbbbb"

$$\delta(q_0, aaabbbbbb, z_0) \vdash (q_0, aabbbbbb, aaz_0)$$
$$\vdash (q_0, abbbbbb, aaaaz_0)$$
$$\vdash (q_0, bbbbbb, aaaaaaz_0)$$
$$\vdash (q_1, bbbbb, aaaaaz_0)$$

$\vdash (q_1, bbbb, aaaaz_0)$

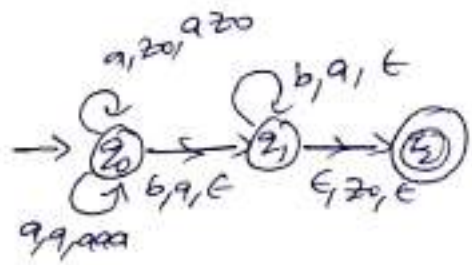
$\vdash (q_1, bbb, aaaz_0)$

$\vdash (q_1, bb, aaz_0)$

$\vdash (q_1, b, az_0)$

$\vdash (q_1, \epsilon, z_0)$

$\vdash (q_2, \epsilon)$



∴ Final state or Accept state.

→ Construct a PDA for the language $L = \{w / w \in (a+b)^* \text{ and } n_a(w) > n_b(w)\}$

$n_a(w)$ means total number of a's in the string and $n_b(w)$ means total number of b's in the string. The Problem is that total number of a's more than total number of b's in input string. The ID can be

$\delta(q_0, a, z_0) = (q_0, az_0)$

$\delta(q_0, b, z_0) = (q_0, b, z_0)$

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

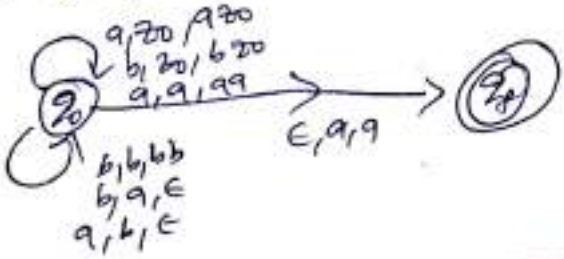
$\delta(q_0, b, b) = (q_0, bb)$

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

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

$\delta(q_0, \epsilon, a) = (q_f, a)$

where $P = (\{q_0, q_f\}, \{a, b\}, \{a, b, z_0\}, \delta, q_0, z_0, \{q_f\})$



consider the input string

$\delta(q_0, aababab, z_0) \vdash (q_0, ababab, az_0)$

$\vdash (q_0, babab, aaz_0)$

$\vdash (q_0, abab, az_0)$

$\vdash (q_0, bab, aaz_0)$

$\vdash (q_0, ab, az_0)$

$\vdash (q_0, b, aaz_0)$

$\vdash (q_0, \epsilon, az_0)$

$\vdash (q_f, a)$ ∴ Accept State

→ Design PDA for the language that accepts strings with $n_a(w) < n_b(w)$ where $w \in (a+b)^*$.

∴ The total number of a's less than total number of b's.

$\delta(q_0, a, z_0) = (q_0, az_0)$

$\delta(q_0, b, z_0) = (q_0, bz_0)$

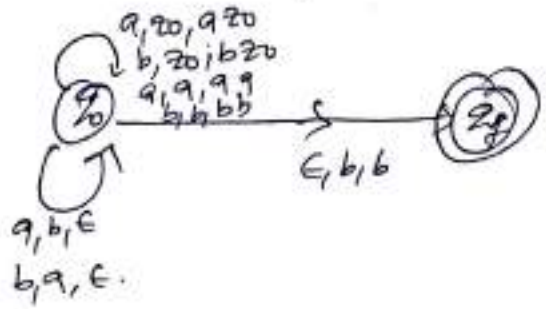
$\delta(q_0, a, a) = (q_0, aa)$

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

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

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

$\delta(q_0, \epsilon, b) = (q_f, b)$ ← The b's are more in number.



PDA for the input string "abbab".

$\delta(q_0, abbab, Z_0) \vdash (q_0, bbab, aZ_0)$

$\vdash (q_0, bab, Z_0)$

$\vdash (q_0, ab, bZ_0)$

$\vdash (q_0, b, Z_0)$

$\vdash (q_0, \epsilon, bZ_0)$

$\vdash (q_f, b)$

$\therefore$ Accept state.

Acceptance by PDA:-

PDA's are useful for checking string acceptance.

A string can be accepted in 2 ways

1. By final state
2. By empty state.

Acceptance by final state :—

Let 'w' be the given string then we can check the string acceptance by

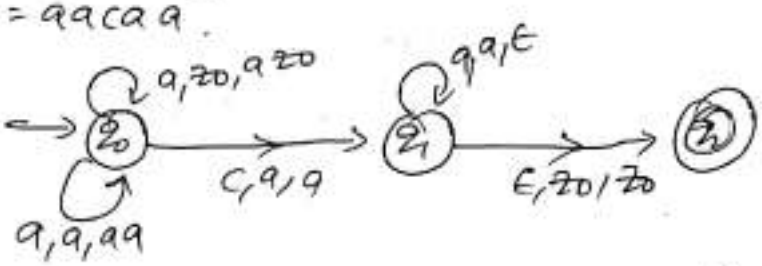
$(q_0, w, Z_0) \vdash^* (q_f, \epsilon, Z_0)$

Acceptance by empty state :—

Let 'w' be the given string & then we can check the string acceptance by empty stack.

$(q_0, w, Z_0) \vdash^* (q_1, \epsilon, \epsilon)$

Eg:- Consider the given PDA and check the string acceptance for $w = aacaa$



$(q_0, aacaa, Z_0) \vdash (q_0, acaa, aZ_0)$

$\vdash (q_0, caa, aaZ_0)$

$\vdash (q_1, aa, aaZ_0)$

$\vdash (q_1, a, aaZ_0)$

$\vdash (q_1, \epsilon, Z_0)$

$\vdash (q_2, \epsilon, Z_0)$ $\in F$

$\therefore$ String is accepted.

Deterministic Push down Automata

A DPDA contains 7 tuples

$M = (Q, \Sigma, S, \delta, q_0, z_0, F)$ it must satisfy following properties:

1. For every $q \in Q$, $a \in \Sigma$, $x \in S$ and $\delta(q, a, x)$ contains only one transitions.
2. For every $q \in Q$, $x \in S$

   $\delta(q, \epsilon, x) \neq \phi$ then I

   $\delta(q, a, x) = \phi$ for every a ~~on the~~ Σ

→ All CFL's are not accepted by DPDA it accepts only Deterministic CFL.

① Construct PDA $L = \{ w w^R / \text{where } w \in (a+b)^* \}$ & check it is DPDA or not.

$\delta(q_0, \epsilon, z_0) = (q_f, z_0)$

$\delta(q_0, a, z_0) = (q_0, a z_0)$

$\delta(q_0, b, z_0) = (q_0, b z_0)$

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

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

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

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

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

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

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

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

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

$\delta(q_1, \epsilon, z_0) = (q_f, \epsilon)$

∴ It is not a DPDA, because it contains multiple transition.

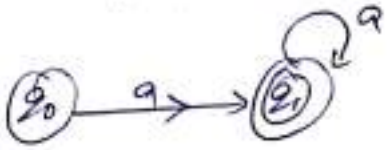
DPDA & Regular languages.

→ Regular languages are accepted by FA

→ Every regular language is a context free language but vice versa is not true.

→ Every regular language is accepted by PDA.

① $L = \{a^n / n \geq 1\}$



$L = \{a, aa, aaa, \ldots\}$

$\delta(q_0, a, z_0) = (q_1, z_0)$

$\delta(q_1, a, z_0) = (q_1, z_0)$

In case of regular language we need to accept the string by reaching final state.

② Design DPDA for accepting all the strings starts abb$(a+b)^*$

$\delta(q_0, a, z_0) = (q_1, z_0)$

$\delta(q_1, b, z_0) = (q_2, z_0)$

$\delta(q_2, b, z_0) = (q_3, z_0)$

$\delta(q_3, a, z_0) = (q_3, z_0)$

$\delta(q_3, b, z_0) = (q_3, z_0)$

Conversion of PDA to CFG

Let PDA, $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0)$

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

Variables V

1. Special Symbol S
2. $[pXq]$ where p, q are states in Q & X is in Γ

Productions R

1. For all States p,
   $S \rightarrow [q_0 z_0 p]$
2. Let $\delta(q, a, X) = (r, Y_1 Y_2 \cdots Y_k)$
   $[qXr_k] \rightarrow a[rY_1r_1][r_1Y_2r_2] \cdots [r_{k-1}Y_kr_k]$
   For all states $r_1, r_2, \cdots r_k$

⇒ Consider the following PDA to CFG

$P = (\{q, p\}, \{0, 1\}, \{X, z\}, \delta, q, z)$

$\delta(q, 1, z) = (q, Xz)$

$\delta(q, 1, X) = (q, XX)$

$\delta(q, \epsilon, X) = (q, \epsilon)$

$\delta(q, 0, X) = (p, X)$

$\delta(p, 1, X) = (p, \epsilon)$

$\delta(p, 0, z) = (q, z)$

6): $V = \{S, [qXq], [qXp], [pXq], [pXp], [qZq], [qZp], [pZq], [pZp]\}$

(i) $S \rightarrow [qZq]$

$S \rightarrow [qZp]$

(ii) $\delta(q,1,Z) = (q,XZ)$

$[qZq] \rightarrow 1[qXq][qZq]$

$[qZq] \rightarrow 1[qXp][pZq]$

$[qZp] \rightarrow 1[qXq][qZp]$

$[qZp] \rightarrow 1[qXp][pZp]$

(iii) $\delta(q,1,X) = (q,XX)$

$[qXq] \rightarrow 1[qXq][qXq]$

$[qXq] \rightarrow 1[qXp][pXq]$

$[qXp] \rightarrow 1[qXq][qXp]$

$[qXp] \rightarrow 1[qXp][pXp]$

(iv) $\delta(q,\epsilon,X) = (q,\epsilon)$

$[qXq] \rightarrow \epsilon$

(v) $\delta(q,0,X) = (p,X)$

$[qXq] \rightarrow 0[pXq]$

$[qXp] \rightarrow 0[pXp]$

(vi) $\delta(p,1,X) = (p,\epsilon)$

$[pXp] \rightarrow 1$

(vii) $\delta(p,0,Z) = (q,Z)$

$[pZq] \rightarrow 0[qZq]$

$[pZp] \rightarrow 0[qZp]$

→ The PDA is as given below

$A = (\{q_0, q_1\}, \{0,1\}, \{S, A\}, \delta, q_0, S, \phi)$ where δ is as given below

$\delta(q_0, 1, S) = \{(q_0, AS)\}$ $\quad$ $\delta(q_1, 1, A) = \{(q_1, \epsilon)\}$

$\delta(q_0, \epsilon, S) = \{(q_0, \epsilon)\}$ $\quad$ $\delta(q_1, 0, S) = \{(q_0, S)\}$

$\delta(q_0, 1, A) = \{(q_0, AA)\}$

$\delta(q_0, 0, A) = \{(q_1, A)\}$

Construct the CFG equivalent to this PDA.

Sol:- Let, we will construct a CFG

$G_1 = (V, T, P, S)$

Here, $T = \{0, 1\}$

$V = \{S \cup [q_0 A q_0], [q_0 A q_1], [q_0 S q_0] [q_0 S q_1] [q_1 S q_1] [q_1 S q_0]$
$[q_1 A q_1] [q_1 A q_0]\}$

Now let us build the production rules as —

(i) $S \rightarrow [q_0 S q_0]$

$S \rightarrow [q_0 S q_1]$

(ii) $\delta(q_0, 1, S) = \{(q_0, AS)\}$

$[q_0 S q_0] \rightarrow 1 [q_0 A q_0] [q_0 S q_0]$

$[q_0 S q_0] \rightarrow 1 [q_0 A q_1] [q_1 S q_0]$

$[q_0 S q_1] \rightarrow 1 [q_0 A q_0] [q_0 S q_1]$

$[q_0 S q_1] \rightarrow 1 [q_0 A q_1] [q_1 S q_1]$

(iii) $\delta(q_0, \epsilon, S) = \{(q_0, \epsilon)\}$

$[q_0 S q_0] \rightarrow \epsilon$

(iv) $\delta(q_0, 1, A) = \{(q_0, AA)\}$

$[q_0 A q_0] \rightarrow 1 [q_0 A q_0] [q_0 A q_0]$

$[q_0 A q_0] \rightarrow 1 [q_0 A q_1] [q_1 A q_0]$

$[q_0 A q_1] \rightarrow 1 [q_0 A q_0] [q_0 A q_1]$

$[q_0 A q_1] \rightarrow 1 [q_0 A q_1] [q_1 A q_1]$

(v) $\delta(q_0, 0, A) = \{(q_1, A)\}$

$[q_0 A q_0] \rightarrow 0 [q_1 A q_0]$

$[q_0 A q_1] \rightarrow 0 [q_1 A q_1]$

(vi) $\delta(q_1, 1, A) = (q_1, \epsilon)$

$[q_1 A q_1] \rightarrow 1$

(vii) $\delta(q_1, 0, S) = \{(q_0, S)\}$

$[q_1 S q_0] \rightarrow 0 [q_0 S, q_0]$

$[q_1 S q_1] \rightarrow 0 [q_0 S q_1]$

→ Convert the following push down automata to Context free grammar.

$M = (\{q_0, q_1\}, \{a, b\}, \{z_0, z_a\}, \delta, q_0, z_0, \phi)$ where

δ is given by

$\delta(q_0, a, z_0) = (q_0, z_a, z_0)$

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

$\delta(q_0, b, z_a) = (q_1, \epsilon)$

$\delta(q_1, b, z_a) = (q_1, \epsilon)$

$\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$

Sol :-

$G = (N, T, P, S)$ where

(a) N = set of non-terminals symbols

$N = \{S \cup [q_0, z_0, q_0], [q_0, z_0, q_1], [q_1, z_0, q_0], [q_1, z_0, q_1]$
$[q_0, z_a, q_0], [q_0, z_a, q_1], [q_1, z_a, q_0], [q_1, z_a, q_1]\}$

T = set of terminal symbols

$T = \{a, b\}$

S is the start symbol

(i) $S \rightarrow [q_0 z_0 q_0] | [q_0, z_0, q_1]$

(ii) $\delta(q_0, a, z_0) = (q_0, z_a, z_0)$

$[q_0 z_0 q_0] \rightarrow a [q_0, z_a, q_0] [q_0, z_0, q_0]$

$[q_0 z_0 q_0] \rightarrow a [q_0 z_a q_1] [q_1 z_0 q_0]$

$[q_0 z_0 q_1] \rightarrow a [q_0 z_a q_0] [q_0 z_0 q_1]$

$[q_0 z_0 q_1] \rightarrow a [q_0 z_a q_1] [q_1 z_0 q_1]$

(iii) $\delta(q_0, a, z_a) = (q_0, z_a, z_a)$

$[q_0, z_a, q_0] \rightarrow a [q_0, z_a, q_0] [q_0 z_a q_0]$

$[q_0 z_a q_0] \rightarrow a [q_0 z_a q_1] [q_1 z_a q_0]$

$[q_0 z_a q_1] \rightarrow a [q_0 z_a q_0] [q_0 z_a q_1]$

$[q_0 z_a q_1] \rightarrow a [q_0 z_a q_1] [q_1 z_a q_1]$

(iv) $\delta(q_0, b, z_a) = (q_1, \epsilon)$

$[q_0 z_a q_1] \rightarrow b$

(v) $\delta(q_1, b, Z_a) = (q_1, \epsilon)$

$[q_1 Z_a q_1] \to b$

(vi) $\delta(q_1, \epsilon, Z_0) = (q_1, \epsilon)$

$[q_1, Z_0, q_1] \to \epsilon$

⇒ Construct CFG for the PDA, $M = (\{q_0, q_1\}, \{0, 1\}, \{R, Z_0\}$, $\delta, q_0, Z_0, \phi)$ and δ is given by

$\delta(q_0, 1, Z_0) = (q_0, RZ_0)$ $\quad$ $\delta(q_1, 0, Z_0) = (q_0, Z_0)$

$\delta(q_0, 1, R) = (q_0, RR)$ $\quad$ $\delta(q_0, \epsilon, Z_0) = (q_0, \epsilon)$

$\delta(q_0, 0, R) = (q_1, R)$ $\quad$ $\delta(q_0, 1, R) = (q_1, \epsilon)$

Sol:

$G = (V, T, P, S)$

Here $T = \{0, 1\}$

$V = \{ S \cup [q_0, S, q_0], [q_0, S, q_1], [q_1, S, q_1], [q_0, Z_0, q_0], [q_0, Z_0, q_1]$, $[q_1, Z_0, q_1], [q_0, R, q_0], [q_0, R, q_1], [q_1, R, q_1] \}$

(i) $S \to [q_0, Z_0, q_0]$

$S \to [q_0, Z_0, q_1]$

(ii) $\delta(q_0, 1, Z_0) = (q_0, RZ_0)$

$[q_0, Z_0, q_0] \to 1 [q_0, R, q_0] [q_0, Z_0, q_0]$

$[q_0, Z_0, q_0] \to 1 [q_0, R, q_1] [q_1, Z_0, q_0]$

$[q_0, Z_0, q_1] \to 1 [q_0, R, q_0] [q_0, Z_0, q_1]$

$[q_0, Z_0, q_1] \to 1 [q_0, R, q_1] [q_1, Z_0, q_1]$

(iii) $\delta(q_0, 1, R) = (q_0, RR)$

$[q_0, R, q_0] \to 1 [q_0, R, q_0] [q_0, R, q_0]$

$[q_0, R, q_0] \to 1 [q_0, R, q_1] [q_1, R, q_0]$

$[q_0, R, q_1] \to 1 [q_0, R, q_0] [q_0, R, q_1]$

$[q_0, R, q_1] \to 1 [q_0, R, q_1] [q_1, R, q_1]$

(iv) $\delta(q_0, 0, R) = (q_1, R)$

$[q_0, R, q_0] \to 0 [q_1, R, q_0]$

$[q_0, R, q_1] \to 0 [q_1, R, q_1]$

(v) $\delta(q_1, 0, z_0) = (q_0, z_0)$

$[q_1, z_0, q_0] \to 0 [q_0, z_0, q_0]$

$[q_1, z_0, q_1] \to 0 [q_0, z_0, q_1]$

(vi) $\delta(q_0, \epsilon, z_0) = (q_0, \epsilon)$

$[q_0, z_0, q_0] \to \epsilon$

(vii) $\delta(q_1, 1, R) = (q_1, \epsilon)$

$[q_1, R, q_1] \to 1$

Conversion of CFG to PDA

Let $G = (V, T, P, S)$ be a CFG

Construct PDA, accept by empty stack.

PDA, $P = (\{q\}, T, V \cup T, \delta, q, S)$

Transition Function δ

1. For each variable A,

$$\delta(q, \epsilon, A) = \{(q, \beta) \mid A \to \beta \text{ is in } P\}$$

2. For each terminal a,

$$\delta(q, a, a) = (q, \epsilon)$$

Q:- Construct PDA for the following CFG & test whether "abbabb" is in N(P)

$$G = (\{S, A\}, \{a, b\}, P, S)$$

$$P = \{ S \to AA \mid a$$
$$A \to SA \mid b \}$$

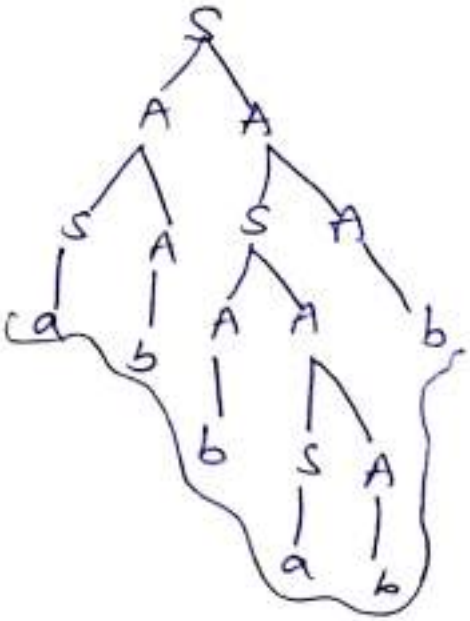
Sol:- PDA, $P = (\{q\}, \{a, b\}, \{S, A, a, b\}, \delta, q, S)$

$$\delta(q, \epsilon, S) = \{(q, AA), (q, a)\}$$

$$\delta(q, \epsilon, A) = \{(q, SA), (q, b)\}$$

$$\delta(q, a, a) = (q, \epsilon)$$

$$\delta(q, b, b) = (q, \epsilon)$$



$(q, abbabb, S) \vdash (q, abbabb, AA)$

$\vdash (q, abbabb, SAA)$

$\vdash (q, abbabb, aAA)$

$\vdash (q, bbabb, AA)$

$\vdash (q, bbabb, bA)$

$\vdash (q, babb, A)$

$\vdash (q, babb, SA)$

$\vdash (q, babb, AAA)$

$\vdash (q, babb, bAA)$

$\vdash (q, abb, AA)$

$\vdash (q, abb, SAA)$

$\vdash (q, abb, aAA)$

$\vdash (q, bb, AA)$

$\vdash (q, bb, bA)$

$\vdash (q, b, A)$

$\vdash (q, b, b)$

$\vdash (q, \epsilon, \epsilon)$

"abbabb" is accepted.

→ Construct PDA for the given CFG

$S \rightarrow 0AA$

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

Test whether 0104 is acceptable by this PDA.

Sol:- PDA, $P = \{\{q\}, \{0,1\}, \{S, A, 0, 1\}, \delta, q, S, \phi\}$

$\delta(q, \epsilon, S) = \{(q, 0AA)\}$

$\delta(q, \epsilon, A) = \{(q, 0S), (q, 1S), (q, 0)\}$

$\delta(q, 0, 0) = \{(q, \epsilon)\}$

$\delta(q, 1, 1) = \{(q, \epsilon)\}$

$\delta(q, 010000, S) \vdash \delta(q, 010000, 0AA)$
$\vdash \delta(q, 10000, AA)$
$\vdash \delta(q, 10000, 1SA)$
$\vdash \delta(q, 0000, SA)$
$\vdash \delta(q, 0000, 0AAA)$
$\vdash \delta(q, 000, AAA)$
$\vdash \delta(q, 000, 0AA)$
$\vdash \delta(q, 00, AA)$
$\vdash \delta(q, 00, 0A)$
$\vdash \delta(q, 0, A)$
$\vdash \delta(q, 0, 0)$
$\vdash \delta(q, \epsilon)$

$\therefore$ Accepted by the PDA.

⇒ Construct the PDA from the given grammar

$S \rightarrow 0S1 \mid A$
$A \rightarrow 1A0 \mid S \mid \epsilon$

$\therefore$ PDA $P = \{\{q\}, \{0,1\}, \{S, A, 0, 1\}, \delta, q, S, \phi\}$

$\delta(q, \epsilon, S) = \{(q, 0S1), (q, A)\}$

$\delta(q, \epsilon, A) = \{(q, 1A0), (q, S), (q, \epsilon)\}$

$\delta(q, 0, 0) = \{(q, \epsilon)\}$

$\delta(q, 1, 1) = \{(q, \epsilon)\}$

⇒ Construct the PDA for the following grammar.

$S \rightarrow AA/a$

$A \rightarrow SA/b$

PDA, $P = (\{q_0, q_1, q_f\}, \{a, b\}, \{S, A, a, b\}, \delta, \{q_0\}, \{q_f\})$

$\delta(q_1, \epsilon, S) = \{(q_1, AA), (q_1, a)\}$

$\delta(q_1, \epsilon, A) = \{(q_1, SA), (q_1, b)\}$

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

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

$\delta(q_1, \epsilon, \$) = (q_f, \epsilon)$ Accept State

$\delta(q_0, \epsilon, \epsilon) = (q_1, S\$)$

⇒ Construct PDA for the following grammar

$S \rightarrow aABB/aAA$

$A \rightarrow aBB/a$

$B \rightarrow bBB/A$

Sol:- PDA, $P = (\{q_0, q_f\}, \{a, b\}, \{Z_0, S, A, B, a, b\}, \delta, q_0, Z_0, q_f)$

$\delta(q_0, \epsilon, S) = \{(q_0, aABB), (q_0, aAA)\}$

$\delta(q_0, \epsilon, A) = \{(q_0, aBB), (q_0, a)\}$

$\delta(q_0, \epsilon, B) = \{(q_0, bBB), (q_0, aBB), (q_0, a)\}$

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

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

$\delta(q_0, \epsilon, Z_0) = (q_0, \epsilon)$ Accept.