* Context-Free Grammar (CFG) :—

- Context-free grammar is defined by 4-tuples

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

V = Set of variables or non-terminal symbols.
T = Set of terminal symbols
S = Start symbol.
P = Production rule

- The context free grammar has production rule of the form

$A \longrightarrow \alpha$ where

$A \in V$ (Nonterminal symbol)

~~$\alpha \in (V \cup T)^*$~~

$\alpha \in (V \cup T)^*$ (Nonterminal & terminal symbol)

→ For example : For generating a language that generates equal no. of a's & b's in the form $a^n b^n$, the context free grammar will be defined as

$$G = (\{S, A\}, \{a, b\}, \{S \to aAb, A \to aAb/\epsilon\})$$

$S \Rightarrow aAb$
$\Rightarrow aaAbb$ [$\because A \to aAb$]
$\Rightarrow aaaAbbb$ [$\because A \to aAb$]
$\Rightarrow aaabbb$ [$\because A \to \epsilon$]

- The language generated by CFG is known as Context-Free Language.
- The machine which accepts CFL is known as Push-Down Automata (PDA).
- The context free grammar is also called as Type-2 grammar.
- Every regular grammar is context-free grammar.
- Examples of CFG are
1) $(\{A\}, \{a,b,c\}, P, A)$, $P: A \to aA, A \to abC$
2) $(\{S,F\}, \{0,1\}, P, S)$, $P: S \to 00S \mid 11F, F \to 00F \mid \epsilon$

* Derivation Using a Grammar :—

- Production rules are used to derive the strings.
- The generation of language using specific rule is called derivation.

Example 1 : Consider the grammar

$$G_1 = (\{S,A\}, \{a,b\}, \{S \to aAb,\ aA \to aaAb,\ A \to \epsilon\}, S)$$

$S \Rightarrow aAb$
$\Rightarrow aaAbb$ [$\because aA \to aaAb$]
$\Rightarrow aaaAbbb$ [$\because aA \to aaAb$]
$\Rightarrow aaabbb$ [$\because A \to \epsilon$]

$\therefore L = \{$ Equal no. of a's & b's $\}$

Example2: $G_2 = (\{S, A, B\}, \{a, b\}, \{S \to AB, A \to a, B \to b\}, S)$

$S \Rightarrow AB$
$\Rightarrow aB \quad [\because A \to a]$
$\Rightarrow ab \quad [\because B \to b]$

$L = \{ab\}$

Example3: $G_3 = (\{S, A, B\}, \{a, b\}, S, \{S \to AB, A \to aA \mid a, B \to bB \mid b\})$

① $S \Rightarrow AB$
$\Rightarrow ab \quad [\because A \to a, B \to b]$

② $S \Rightarrow AB$
$\Rightarrow aAB \quad [\because A \to aA]$
$\Rightarrow aAbB \quad [\because B \to bB]$
$\Rightarrow aabb \quad [\because A \to a, B \to b]$

③ $S \Rightarrow AB$
$\Rightarrow aAB \quad [\because A \to aA]$
$\Rightarrow aab \quad [\because A \to a, B \to b]$

④ $S \Rightarrow AB$
$\Rightarrow abB \quad [\because A \to a, B \to bB]$
$\Rightarrow abb \quad [\because B \to b]$

$\therefore L = \{ab, aabb, aaabbb, aab, abb, \cdots\}$

$= \{$ Strings which are starting & ending which different symbols $\}$

- There are two types of derivations
  - ↳ Leftmost derivation
  - ↳ Rightmost derivation

a) Left Most Derivation (LMD) :-

- A derivation $A \overset{*}{\Rightarrow} w$ is called a leftmost derivation if we apply a production only to the leftmost variable at every step.
- In another word, in LMD of a string, the leftmost non-terminal is replaced first at each step in a ~~der~~ derivation.
- The leftmost derivation of string abababa is over grammer of production $S \rightarrow SbS \mid a$

$$S \underset{lm}{\Rightarrow} \underline{S}bS$$

$$\underset{lm}{\Rightarrow} \underline{S}bSbS$$

$$\underset{lm}{\Rightarrow} \underline{S}bSbSbS$$

$$\underset{lm}{\Rightarrow} ab\underline{S}bSbS$$

$$\underset{lm}{\Rightarrow} abab\underline{S}bS$$

$$\underset{lm}{\Rightarrow} ababab\underline{S}$$

$$\underset{lm}{\Rightarrow} abababa$$

b) Rightmost Derivation (RMD) :-

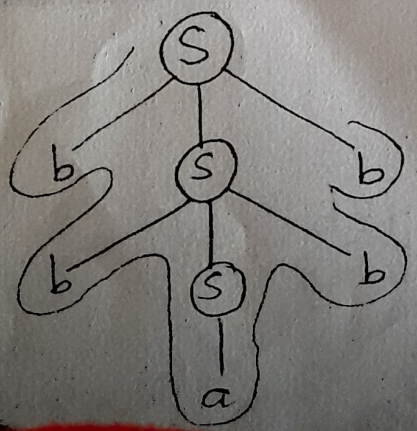
- A derivation $A \overset{*}{\Rightarrow} w$ is called rightmost derivation if we apply a production to rightmost variable at every step.
- In RMD of string, the rightmost nonterminal is replaced first at each step in a derivation.
- The rightmost derivation of string abababa over the production $S \rightarrow SbS \mid a$ is

$$S \underset{rm}{\Rightarrow} SbS$$
$$\underset{rm}{\Rightarrow} SbSbS$$
$$\underset{rm}{\Rightarrow} SbSbSbS$$
$$\underset{rm}{\Rightarrow} SbSbSba$$
$$\underset{rm}{\Rightarrow} SbSbaba$$
$$\underset{rm}{\Rightarrow} Sbababa$$
$$\underset{rm}{\Rightarrow} abababa$$

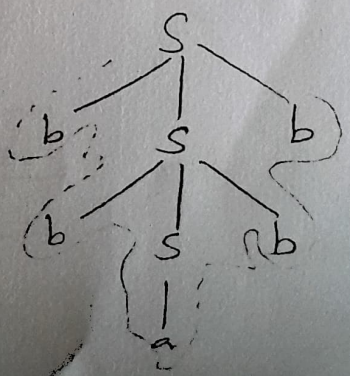
Page 1 Example

* Derivation tree / Parse tree / Syntax tree :-

- A derivation tree or parse tree is an ordered rooted tree that graphically represents the semantic information of strings derived from a context free grammar.
- The trees which represents derivations in a CFG are called derivation trees.
- It is graphical representation for derivation of given production rules for a given CFG.
- It is a simple way to show how derivation can be done to obtain some string from given set of production rule.
- Following are the propestes of any derivation tree
  - ↳ The root node is always a node indicating start symbol.
  - ↳ The derivation is read from left to right.
  - ↳ The leaf nodes are always termina/ nodes.
  - ↳ The interior nodes are always non-termina/ nodes.
- For example :
  Construct derivation tree for deriving a string 'bbabb' from grammar $S \longrightarrow bSb/a/b$



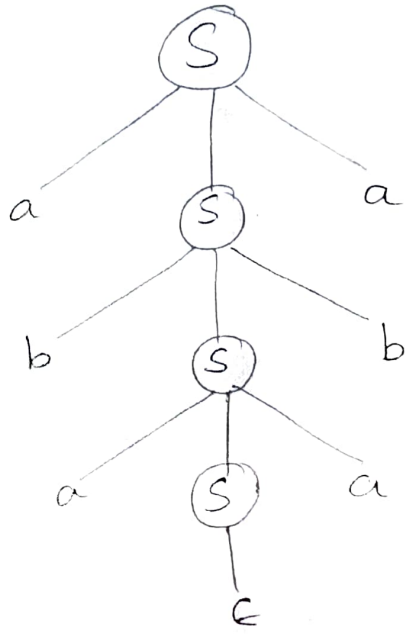
or



Example 2 : Draw a derivation tree for the string abaaba for CFG G where

$P = \{ S \rightarrow aSa, S \rightarrow bSb, S \rightarrow a|b|\epsilon \}$

Sol" :-



Example 3 : Consider the grammar $E \rightarrow +EE|*EE|-EE|x|y$ Find leftmost and rightmost derivation for the string '+ * − xyxy' and write parse tree

Solution : Given string is '+ * − xyxy'

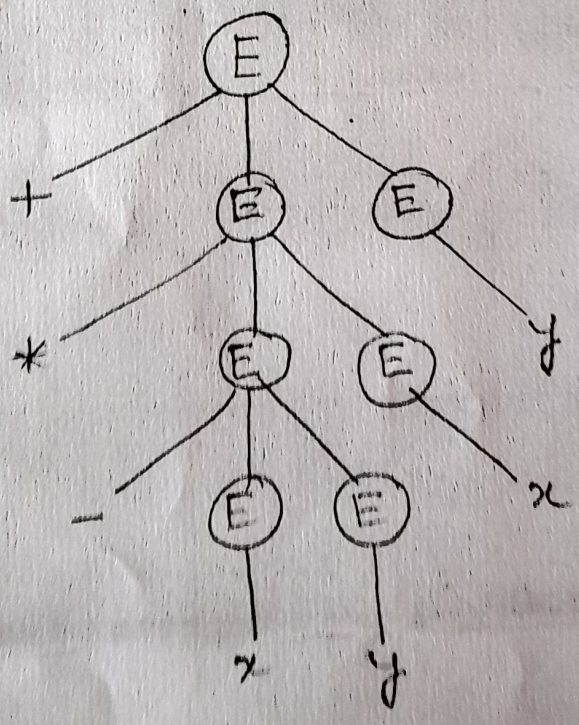
a) Leftmost derivation :-

$$
\begin{aligned}
E &\Rightarrow +\underline{E}E \\
&\Rightarrow +*\underline{E}EE \\
&\Rightarrow +*-\underline{E}EEE \\
&\Rightarrow +*-x\underline{E}EE \\
&\Rightarrow +*-xy\underline{E}E \\
&\Rightarrow +*-xyx\underline{E} \\
&\Rightarrow +*-xyxy
\end{aligned}
$$

b) Right-most derivation :—

$$
\begin{aligned}
E &\Rightarrow +EE \\
&\Rightarrow +Ey \\
&\Rightarrow +*EEy \\
&\Rightarrow +*Exy \\
&\Rightarrow +*-EExy \\
&\Rightarrow +*-Eyxy \\
&\Rightarrow +*-xyxy
\end{aligned}
$$

c) Parse tree :—



Example 4 : Given the grammar G1 as,

$S \rightarrow 0B \mid 1A$, $A \rightarrow 0 \mid 0S \mid 1AA$

$B \rightarrow 1 \mid 1S \mid 0BB$.

Give the LMD, RMD and derivation trees to derive the string 00110101 .

Solution :-

Given grammar is

$S \rightarrow 0B \mid 1A$

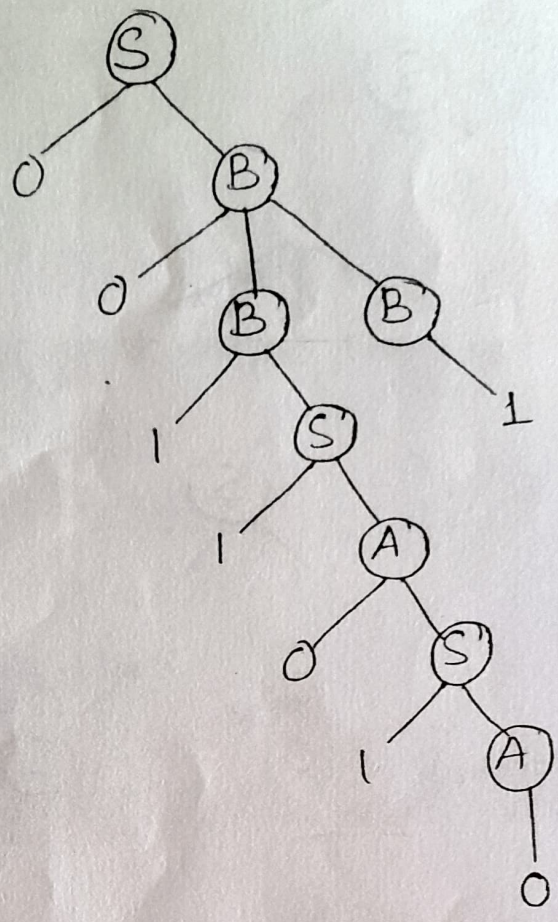
$A \rightarrow 0 \mid 0S \mid 1AA$

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

a) Leftmost derivation for string 00110101 is

$$S \underset{lm}{\Rightarrow} 0B$$
$$\underset{lm}{\Rightarrow} 00BB$$
$$\underset{lm}{\Rightarrow} 001SB$$
$$\underset{lm}{\Rightarrow} 0011AB$$
$$\underset{lm}{\Rightarrow} 00110SB$$
$$\underset{lm}{\Rightarrow} 001101AB$$
$$\underset{lm}{\Rightarrow} 00110101B$$
$$\Rightarrow 00110101$$

b) The leftmost derivation tree is shown below



c) The rightmost derivation for string ~~100~~ 00110101 is

$$
\begin{aligned}
S &\underset{rm}{\Longrightarrow} 0B \\
&\underset{rm}{\Longrightarrow} 00\underline{B}B \\
&\underset{rm}{\Longrightarrow} 00\underline{B}1 \\
&\underset{rm}{\Longrightarrow} 001\underline{S}1 \\
&\underset{rm}{\Longrightarrow} 0011\underline{A}1 \\
&\underset{rm}{\Longrightarrow} 00110\underline{S}1 \\
&\underset{rm}{\Longrightarrow} 001101\underline{A}1 \\
&\underset{rm}{\Longrightarrow} 00110101
\end{aligned}
$$

Ambiguous Grammar / Ambiguity in CFG :-

- A grammar is said to be ambiguous, if it produces more than one derivation tree for some i/p string generated by it.
- In other words, if there are more than one leftmost derivation or more than one rightmost derivation for a given string. Then such a grammar is said to be ambiguous.
- ~~Defn~~ Formal definition

$G = (V, T, P, S)$ is a CFG is said to ambiguous if and only if there exists a string in T^* that has more than one parse tree.

where

$V \rightarrow$ Finite set of variables.

$T \rightarrow$ Finite set of terminals.

$P \rightarrow$ Production rule of the form $A \rightarrow \alpha$ where $A \in V$ & $\alpha \in (VUT)^*$

$S \rightarrow$ Start symbol.

- Let us consider the grammar

$$E \rightarrow E+E / id$$

We can create two parse trees from this grammar to obtain the string id+id+id :

1st LMD

$E \Rightarrow \underline{E}+E$

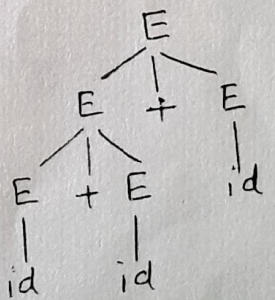
$\Rightarrow \underline{E}+E+E$

$\Rightarrow id+\underline{E}+E$

$\Rightarrow id+id+\underline{E}$

$\Rightarrow id+id+id$

Parse tree will be



2nd LMD : $E \Rightarrow \underline{E}+E$

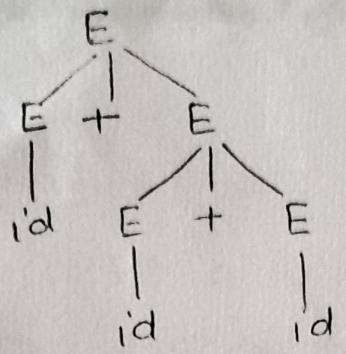
$\Rightarrow id+E$

$\Rightarrow id+\underline{E}+E$

$\Rightarrow id+id+E$

$\Rightarrow id+id+id$

Parse tree will be



Both above parse trees are derived from same grammar but both parse trees are different. Hence the grammar is ambiguous grammar.

- Let us consider another grammar:

$S \rightarrow SS \mid a \mid b$, Derive the string 'aba'

1st LMD: $S \Rightarrow \underline{S}S$

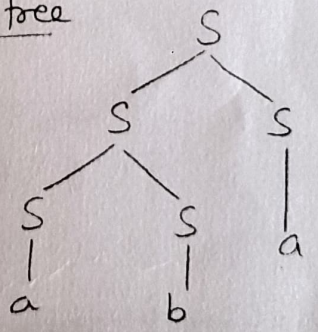
$\Rightarrow \underline{S}SS$

$\Rightarrow a\underline{S}S$

$\Rightarrow ab\underline{S}$

$\Rightarrow aba$

1st parse tree



2nd LMD: $S \Rightarrow \underline{S}S$

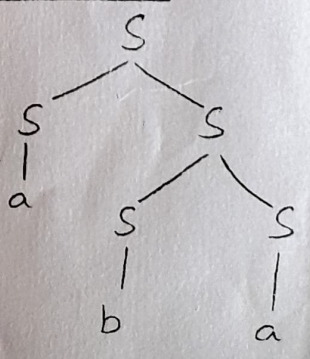
$\Rightarrow a\underline{S}$

$\Rightarrow a\underline{S}S$

$\Rightarrow ab\underline{S}$

$\Rightarrow aba$

2nd parse tree



Since there are two derivation trees for the string aba, the grammar is ambiguous.

Ambiguity of a grammar is undecidable i.e. there is no particular algorithm for removing ambiguity of grammar but we can remove ambiguity by:

"Rewriting the grammar such that there is only one derivation or parse tree possible for a string of language which the grammar represents".

Eliminating Ambiguity from the grammar:-

If grammar is of the form

$$A \longrightarrow \theta A \lambda A \beta \mid \theta_1 \mid \theta_2 \mid \theta_3 \mid \cdots \theta_n$$

Then the new unambiguous grammar can be created by introducing a new variable called A' and converting the grammar to form

$$A \longrightarrow \theta A \lambda A \beta \mid A'$$
$$A' \longrightarrow \theta_1 \mid \theta_2 \mid \theta_3 \mid \theta_4 \mid \cdots \theta_n$$

Here A' is a new variable which eliminate ambiguity from the grammar.

Example1 : Show that the following grammar is ambiguous: $E \rightarrow E+E \mid E-E \mid E*E \mid E/E \mid (E) \mid a$ where E is the start symbol.

solution :— Let us derive the string $(a+a*(a))$ is as follows

or

$$E \Rightarrow \underline{E}*E$$
$$\Rightarrow \underline{E}+E*E$$
$$\Rightarrow a+\underline{E}*E$$
$$\Rightarrow a+a*E$$
$$\Rightarrow a+a*(E)$$
$$\Rightarrow a+a*(a)$$

The derivation trees are as follows

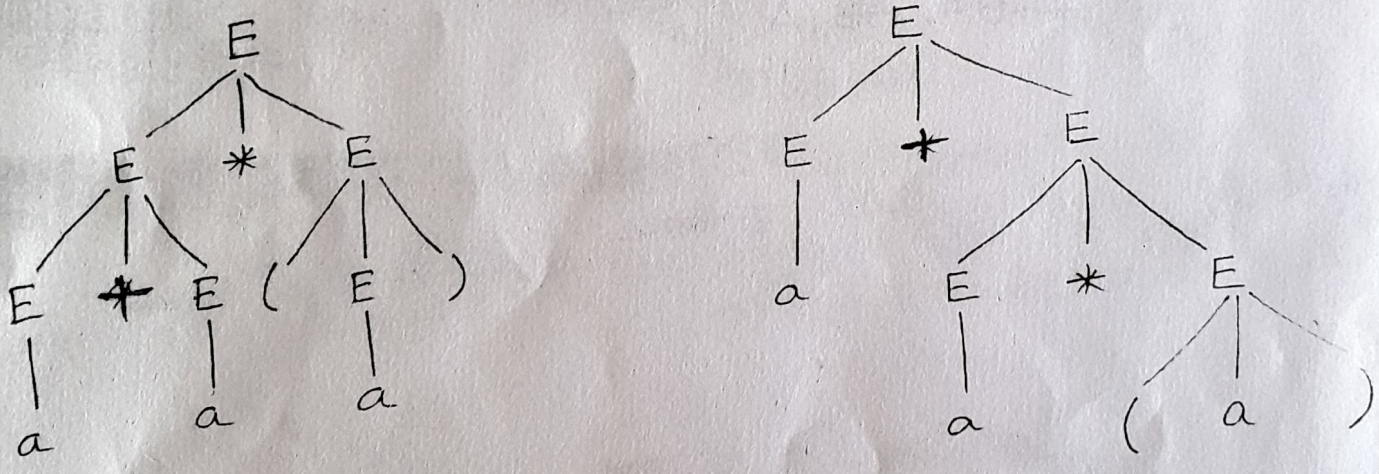


Fig:- Derivation trees for $a+a*(a)$

- Since two different derivation trees are possible for the same string $a+a*(a)$. So the given grammar is ambiguous.

Example 2: Show that grammar $S \longrightarrow SS, S \longrightarrow a$ is ambiguous.

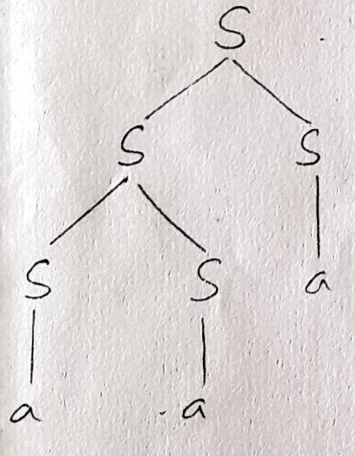
solution:- Given grammar is

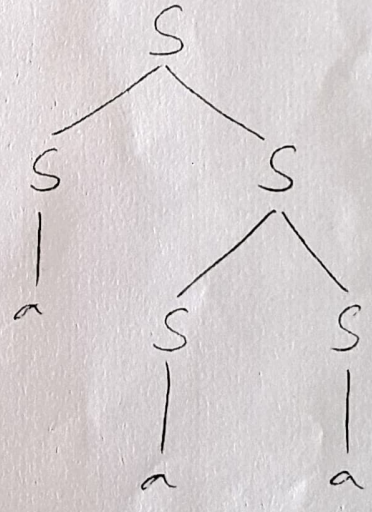
$$S \longrightarrow SS/a$$

let us derive the string from this grammar

$$S \Longrightarrow SS$$
$$\Longrightarrow SSS$$
$$\Longrightarrow aSS$$
$$\Longrightarrow aaS$$
$$\Longrightarrow aaa$$

Now we construct derivation trees for this string aaa.





Since, two different derivation trees are possible, for the string, so the grammar is ambiguous.

Example3: Show that the grammar is ambiguous

$$S \longrightarrow a \mid Sa \mid bSS \mid SSb \mid SbS$$

solution:

Given grammar G1 is

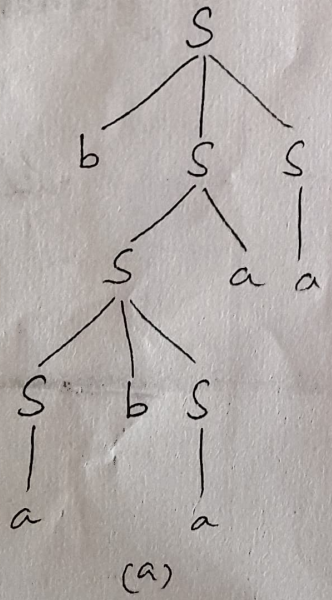
$$S \longrightarrow a \mid Sa \mid bSS \mid SSb \mid SbS$$

Consider the string w = babaaa which is generated by G.

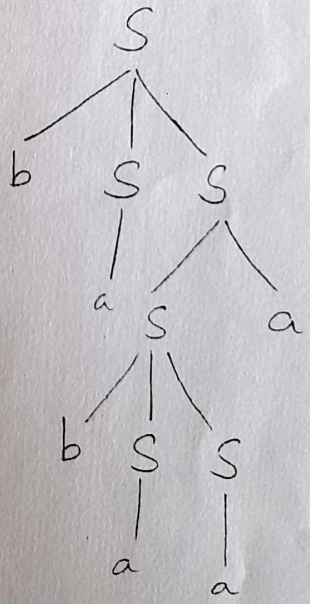
Let us first derive a string from this grammar

$$\begin{aligned} S &\Rightarrow b\underline{S}S \\ &\Rightarrow b\underline{S}aS \\ &\Rightarrow b\underline{S}bSaS \\ &\Rightarrow bab\underline{S}aS \\ &\Rightarrow babaa\underline{S} \\ &\Rightarrow babaaa \end{aligned}$$

Now construct derivation trees for the string babaaa



(a)



Since, there are two derivation trees, so grammar is ambiguous.

* Push Down Automata (PDA) :-

- A pushdown automata is a way to implement a context-free grammar in a similar way we design DFA for a regular language.
- A DFA can remember a finite amount of information but a PDA can remember an infinite amount of informatn.
- Basically finite automata with a stack memory can be viewed as PushDown Automata.

["Finite State Machine" + "stack"]

- A stack does two operations
  - ↳ Push: A new symbol is added at the top.
  - ↳ Pop: The top symbol is read & removed.
- The operation of stack is based on LIFO principle (i.e. Last symbol pushed on the stack will be popped first).
- The stack memory is also known as Pushdown Store.
- The stack head scans the top symbol of the stack.
- A PDA may or maynot read an i/p symbol, but it has to read the top of stack in every transition.
- A pushdown automata has three components
  - ↳ An i/p tape.
  - ↳ A Finite Control Unit
  - ↳ A stack with infinite size

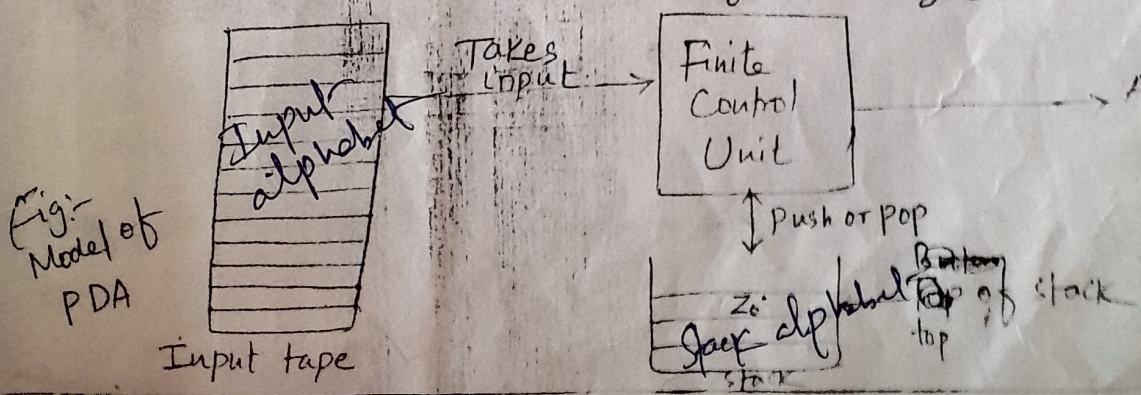


Fig:- Model of PDA

a) Input tape :-

- It is a sequential tape that consists of finite no. of cells where each cell holds single input symbol.

b) Finite Control Unit :-

- It performs read/write operation on cells of i/p tape and also take decision regarding which input symbol to read next.

c) A stack with infinite size :-

- It is used for storing temporary items and for performing PUSH and POP operations.

* Formal Definition of PDA :-

- Formally, PDA is defined as 7-tuple machine i.e.

$M = (Q, \Sigma, \delta, q_0, F, \tau, Z_0)$ where

Q = Finite set of states.

Σ = Finite set of i/p symbol i.e. I/p alphabet

δ = Transition function which

$$\delta : Q \times (\underbrace{\Sigma \cup \{\epsilon\}}_{\text{I/p}}) \times \underbrace{\tau}_{\text{Top}} \longrightarrow Q \times \underbrace{\tau^*}_{\text{Top}}$$

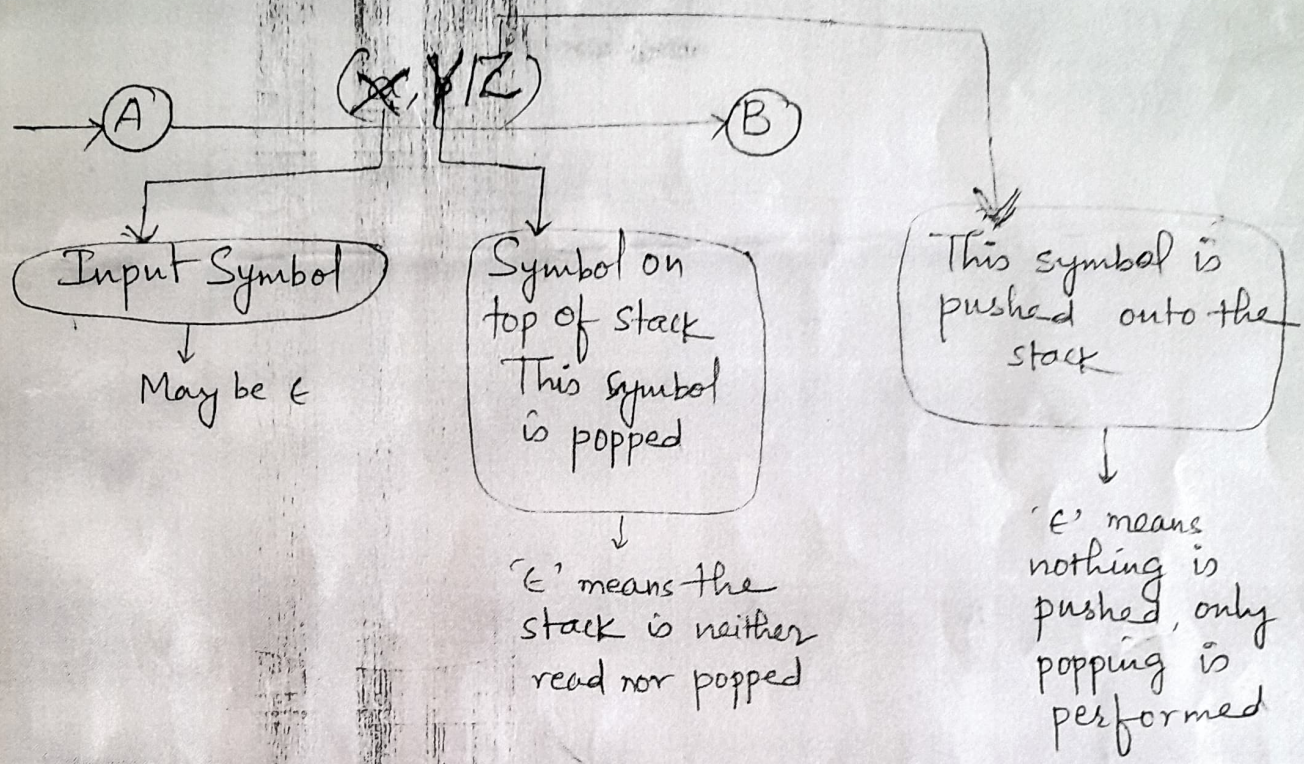
q_0 = Initial state

F = set of final states

τ = A finite stack alphabet i.e. set of symbols to push on the stack.

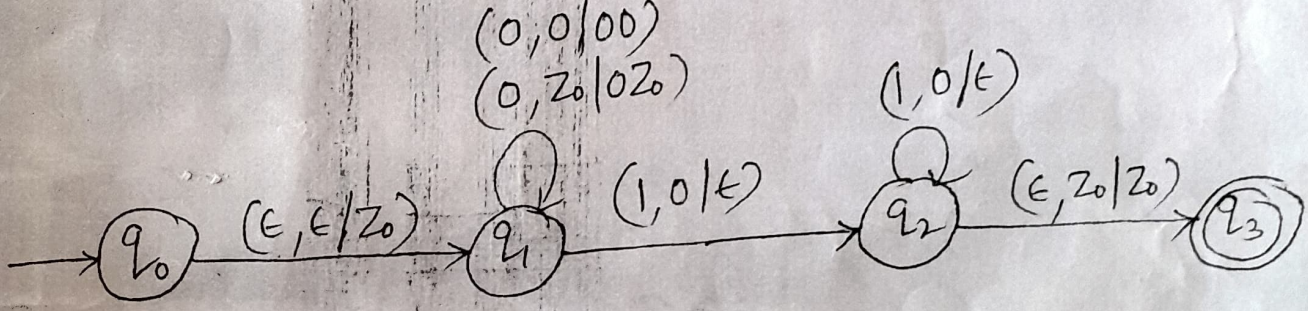
Z_0 = Bottom of stack or Start stack symbol.

⇒ Graphical Representation :—

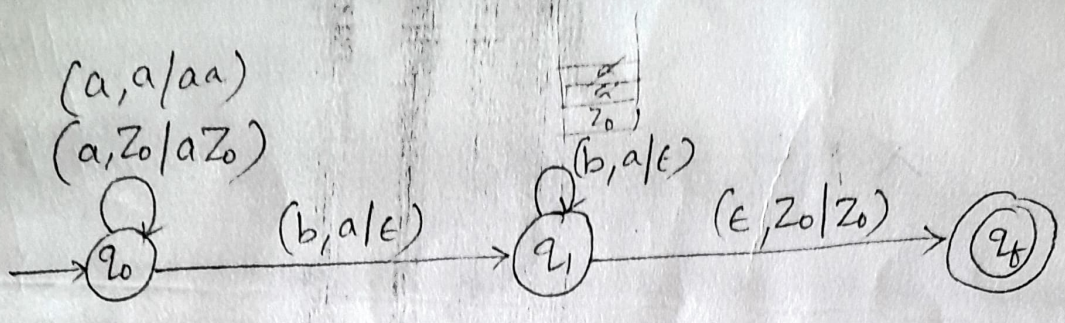


Example1 : Construct PDA that accepts

$$L = \{0^n 1^n \mid n \geq 0\}$$

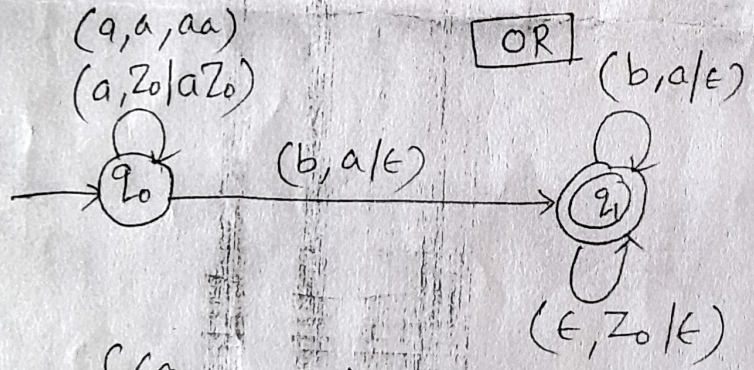


Example 2 : Construct PDA for $L = \{a^n b^n \mid n \geq 1\}$



$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, \epsilon)$
$\delta(q_1, b, b) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, Z_0) = (q_f, Z_0)$

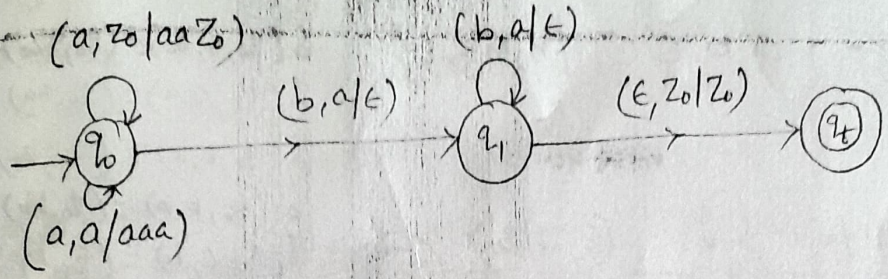
This is called acceptance of PDA by final state



$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, \epsilon)$
$\delta(q_1, b, a) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, Z_0) = (q_1, \epsilon)$

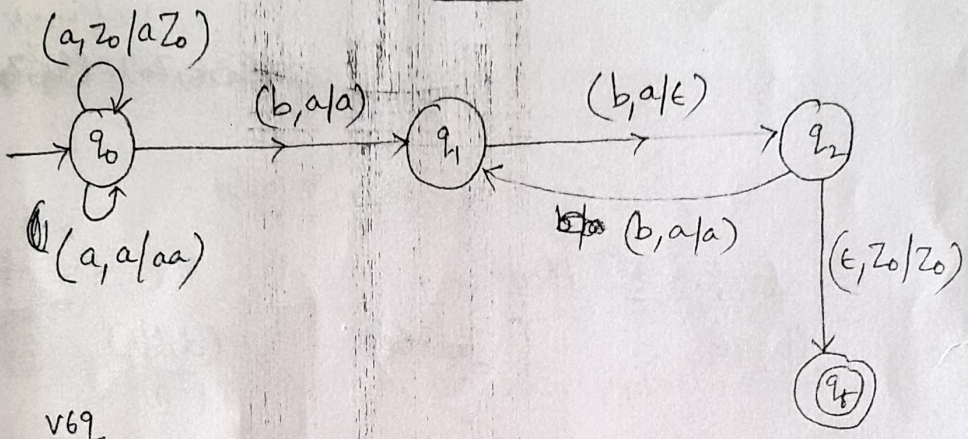
This is called acceptance of PDA by empty stack

V69
Example 11 :- Construct PDA for $L = \{ a^n b^{2n} \mid n \geq 1 \}$



$\delta(q_0, a, Z_0) = (q_0, aaZ_0)$
$\delta(q_0, a, a) = (q_0, aaa)$
$\delta(q_0, b, a) = (q_1, \epsilon)$
$\delta(q_1, b, a) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, Z_0) = (q_f, Z_0)$

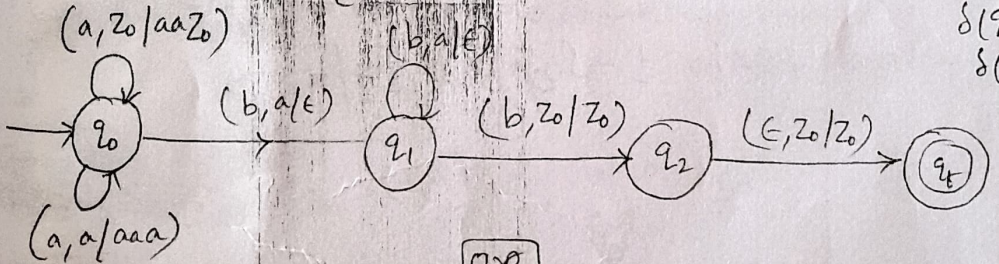
OR



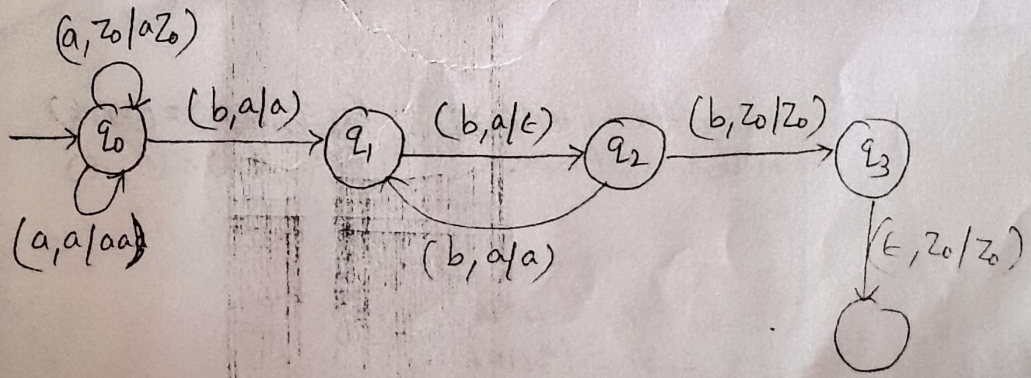
V69
Example 12 : Construct PDA for $L = \{ a^n b^{2n+1} \mid n \geq 1 \}$

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

$\delta(q_0, a, Z_0) = (q_0, aaZ_0)$
$\delta(q_0, a, a) = (q_0, aaa)$
$\delta(q_0, b, a) = (q_1, \epsilon)$
$\delta(q_1, b, a) = (q_1, \epsilon)$
$\delta(q_1, b, Z_0) = (q_2, Z_0)$
$\delta(q_2, \epsilon, Z_0) = (q_f, Z_0)$



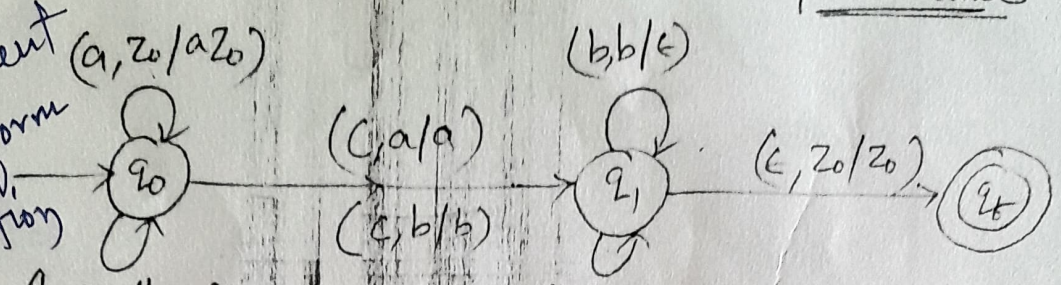
OR



V69

Example 13 : Construct PDA for $L = \{ wcw^R / w \in (a,b)^+ \}$ → Palindrome

Two different i/ps, perform pop operation after C



(b, Z0/bZ0)
(a, a/aa)
(b, a/ba)
(a, b/ab)
(b, b/bb)

(a, a/ϵ)

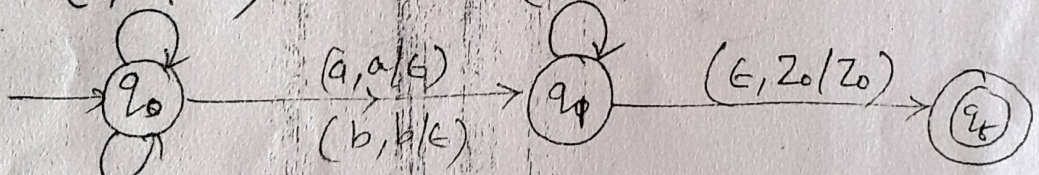
$\delta(q_0, c, a) = (q_1, a)$
$\delta(q_0, c, b) = (q_1, b)$
$\delta(q_1, b, b) = (q_1, \epsilon)$
$\delta(q_1, a, a) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, Z_0) = (q_f, Z_0)$

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, b, Z_0) = (q_0, bZ_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_0, ba)$
$\delta(q_0, a, b) = (q_0, ab)$
$\delta(q_0, b, b) = (q_0, bb)$

V70

Example 14 : — Construct NPDA for $L = \{ ww^R / w \in (a,b)^+ \}$

It is Difficult to identify when to push and ~~pop~~ when to pop. So this problem is solved by NPDA.



(a, b/ab)
(b, a/ba)
(a, a/aa)
(b, b/bb)

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, b, Z_0) = (q_0, bZ_0)$
$\delta(q_0, a, b) = (q_0, ab)$
$\delta(q_0, b, a) = (q_0, ba)$
$\delta(q_0, a, a) = (q_0, ab)$
$\delta(q_0, b, b) = (q_0, bb)$

$\delta(q_0, a, a) = (q_1, \epsilon)$
$\delta(q_0, b, b) = (q_1, \epsilon)$
$\delta(q_f, a, a) = (q_1, \epsilon)$
$\delta(q_1, b, b) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, Z_0) = (q_f, Z_0)$

VTU

Example 15 : Construct NPDA for

$L = \{a^n b^n / n \geq 1\} \cup \{a^n b^{2n} / n \geq 1\}$

$\delta(q_s, a, Z_0) = (A, aZ_0)$

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

$\delta(A, b, a) = (B, \epsilon)$

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

$\delta(B, \epsilon, Z_0) = (C, Z_0)$

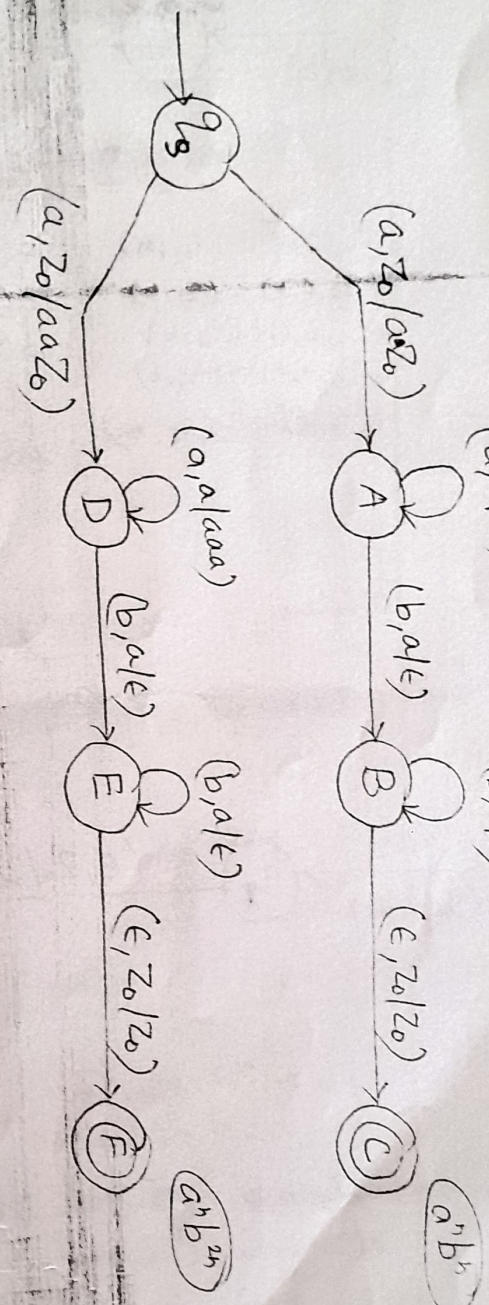
$\delta(q_s, a, Z_0) = (D, aaZ_0)$

$\delta(D, a, a) = (D, aaa)$

$\delta(D, b, a) = (E, \epsilon)$

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

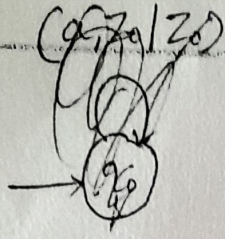
$\delta(E, \epsilon, Z_0) = (F, Z_0)$

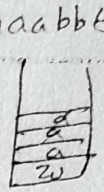


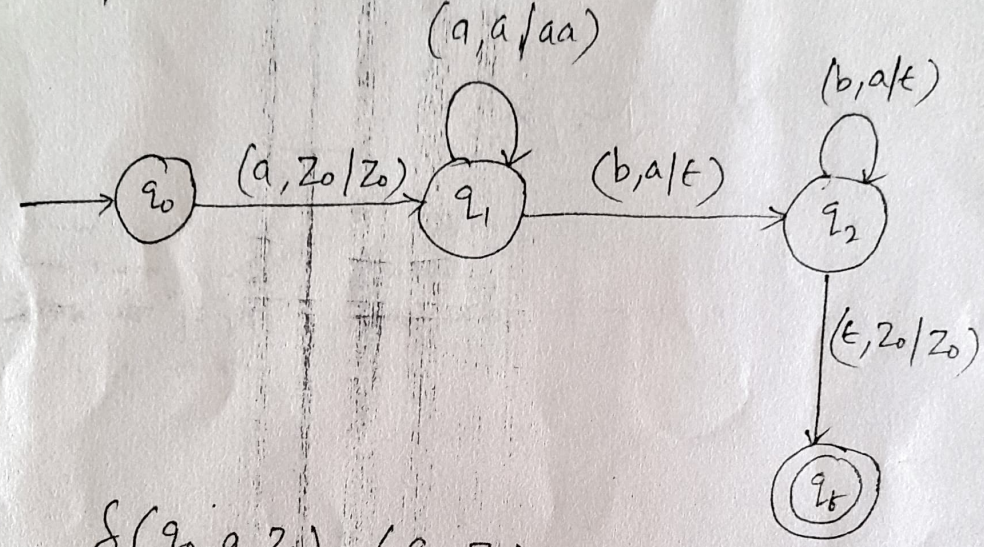
Example 16: Construct PDA that accepts language

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

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







$\delta(q_0, a, Z_0) = (q_1, Z_0)$

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

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

$\delta(q_2, b, a) = (q_2, \epsilon)$

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

Example · Consider the language $L = \{a^n b^n / n \geq 1\}$

In DPDA, δ is given as

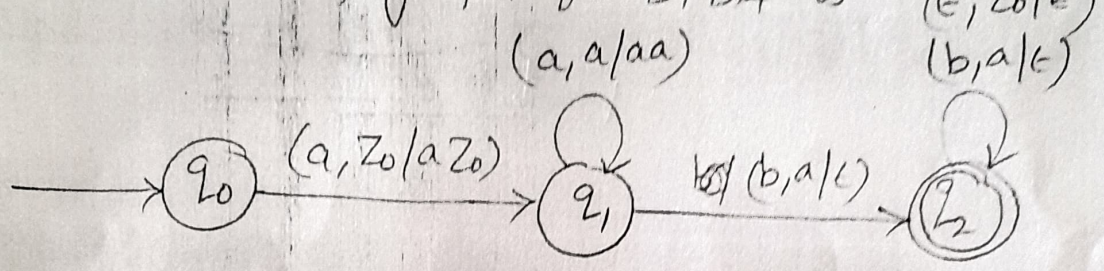
$\delta(q_0, a, Z_0) = (q_1, aZ_0)$

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

$\delta(q_2, b, a) = (q_2, \epsilon)$

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

The transition graph for DPDA is



Let us see the acceptance of string aabb

$\delta(q_0, aabb, Z_0) \Rightarrow (q_1, abb, aZ_0)$

$\Rightarrow (q_1, bb, aaZ_0)$

$\Rightarrow (q_2, b, aZ_0)$

$\Rightarrow (q_2, \epsilon, Z_0)$

$\Rightarrow (q_2, \epsilon)$

The above DPDA accepts the given language. Hence it is deterministic.

* Equivalence of CFG & PDA :-
* Conversion of CFG to PDA :-

- While designing a PDA, we will do first PUSH and POP operations i.e. We will push symbols into the stack and then do POP the pushed symbol using another symbols.
- But while converting CFG to PDA, first we will simply write production rules and then we POP

Example 1: Convert the following grammar to a PDA that accepts the language by empty stack.

$S \longrightarrow aSa$
$S \longrightarrow bSb$
$S \longrightarrow c$

or Let G_1 be a CFG that generates the set of palindromes given by $S \longrightarrow aSa/bSb/c$

Solution :- Given grammar G_1 is

$S \longrightarrow aSa$
$S \longrightarrow bSb$
$S \longrightarrow c$

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

Production rules are as follows.

$R_1 : \delta(q_0, \epsilon, \epsilon) = (q_0, S)$
$R_2 : \delta(q_0, \epsilon, S) = (q_0, aSa)$
$R_3 : \delta(q_0, \epsilon, S) = (q_0, bSb)$
$R_4 : \delta(q_0, \epsilon, S) = (q_0, c)$

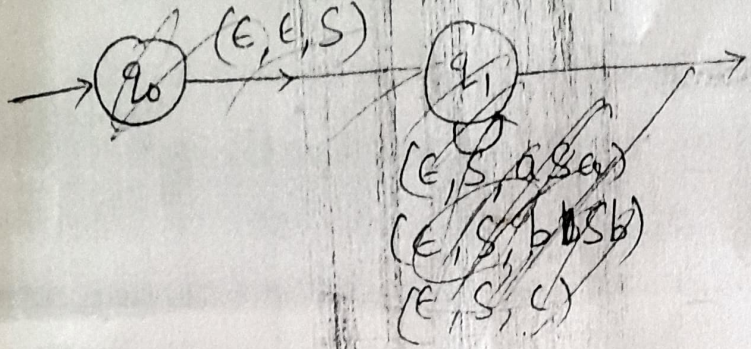
} Production rules with no i/p symbols

$R_5 : \delta(q_0, a, a) = (q_1, \epsilon)$
$R_6 : \delta(q_1, b, b) = (q_2, \epsilon)$
$R_7 : \delta(q_2, c, c) = (q_3, \epsilon)$

} Pop operation

Note :- Whenever there is change of i/p then there will be state change.

Graphical representation



$S \Rightarrow aSa$
$\Rightarrow abSba$
$\Rightarrow abbSbba$
$\Rightarrow abbcbba$

Sr No	State	Unread input	stack	Rules used
1	q_0	abbcbba	S	R_1
2.	q_0	abbcbba	~~bSb~~ aSa	R_2
3.	q_1	bbcbba	Sa	R_5
4.	q_0	bbCbba	bSba	R_3
5.	q_2	bcbba	Sba	R_6
6.	q_0	bcbba	bSbba	R_3
7.	q_2	cbba	Sbba	R_6
8.	q_0	cbba	cbba	R_4
9.	q_3	bba	bba	R_7
10.	q_2	~~bb~~ ba	ba	R_6
11.	q_2	a	a	R_5
12.	q_1	ϵ	ϵ	

Example2 : Convert the following grammar to PDA that accepts the same language by empty stack.

$S \rightarrow 0S1/A$

$A \rightarrow 1A0/S/\epsilon$

Solution :- Given grammar G is

$S \longrightarrow 0S1/A$

$A \longrightarrow 1A0/S/\epsilon$

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

Production rules are as follows

$R_1 : \delta(q_0, \epsilon, \epsilon) = (q_0, S)$

$R_2 : \delta(q_0, \epsilon, S) = (q_0, 0S1)$

$R_3 : \delta(q_0, \epsilon, S) = (q_0, A)$ } Production rule

$R_4 : \delta(q_0, \epsilon, A) = (q_0, 1A0)$

$R_5 : \delta(q_0, \epsilon, A) = (q_0, S)$

$R_6 : \delta(q_0, \epsilon, A) = (q_0, \epsilon)$

$R_7 : \delta(q_0, 0, 0) = (q_1, \epsilon)$

$R_8 : \delta(q_1, 1, 1) = (q_2, \epsilon)$ } Pop operation

let us derive the string from grammar & check for that using empty stack

$S \Rightarrow 0S1$

$\Rightarrow 00S11$

$\Rightarrow 00A11$

$\Rightarrow 001A011$

$\Rightarrow 001011$

$S \Rightarrow 0S1$

$\Rightarrow 0A1$

$\Rightarrow 01A01$

$\Rightarrow 01S01$

$\Rightarrow 010S101$

$\Rightarrow 010A101$

$\Rightarrow 010\epsilon101 \Rightarrow 010101$

Sr NO	State	Unread i/p	Stack	Rules used
1.	q_0	010101	S	R_1
2.	q_0	010101	0S1	R_2
3.	q_1	10101	S1	R_7
4.	q_0	10101	1A01	R_3
5.	q_2	0101	A01	R8
6.	q_0	0101	S01	R_5
7.	q_0	0101	0S101	R_2
8.	q_1	101	S101	R_7
9.	q_0	101	A101	R_3
10.	q_0	101	101	R_6
11.	q_2	01	01	R8
12.	q_1	1	1	R_7
13.	q_2	ϵ	ϵ (Accepted)	R8