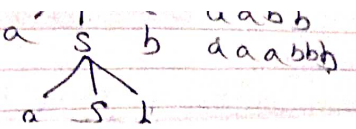
$a^n b^n, n \geq 0$

ch / e.



UNIT- 3

1. Context Free Grammer (CFG):

- A T.t is used to defined the rules for syntax off context free language.
- The construction of language with correct form is known as syntax of it and the correct meaning of the language is called schematic of language.

For checking the correctness of program syntactically use CFG.

The context free grammer is defined as four tuples (V, T, P, S)

V → Finite set of variables (Non-termining)

T → Finite set of Terminals ($V \cap T = \phi$)

P → Production rules ($\alpha \rightarrow \beta$)

S → start symbol.

- Parse Tree: (Explain More for 7 marks)

Parse Tree is a graphical representation of how the string is derived from grammer G.

→ Left Derivation:

At each step in a derivation, the left most non-termind is replaced, such derivation is called Left-Most derivation.

→ Right Derivation:

At each step in a derivation, the right-most non-terminal is replaced, such derivation is called Right-Most Derivtion

Q. Consider the Grammer with production.

$S \rightarrow aSS / b$

(aababbb) find L.M.D & R.M.D

⇒ L.M.D ⇒

S

aSS

$aaSSS$

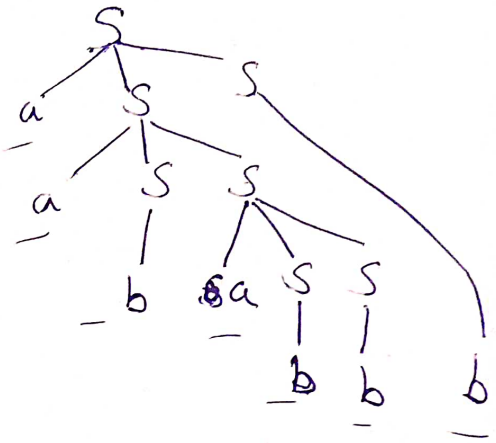
$aabSS$

$aabaSSS$

$aababSS$

$aababbS$

$aababbb$.



R.M.D ⇒

S

$aSbS$

aSb

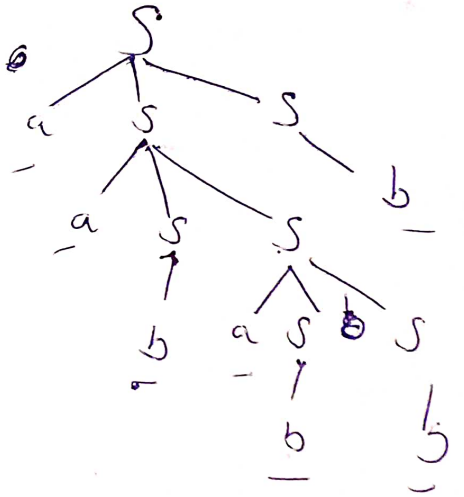
$aaSSb$

$aaSaSSb$

$aaSaSbb$

$aaSabbb$

$aababbb$



Ambiguous Gramer	Unambiguous Grammer
(i) ~~In ambiguous grammar, the~~ leftmost and ~~rightmost~~ derivation are not same.	(i) ~~In Amba~~ Unambiguas
(i) Exist more than 1 derivation for at least 1 string.	(i) Exist only one unique derivation tree for any string
(ii) May lead to multiple interpretations of same input	(ii) Ensure that the language's syntax is well-defined and unambiguous
(iii) Result in smaller parse tree.	(iii) Result in large parse tree.
(iv) Ambiguity can be problematic in programming languages and compiler.	(iv) Does not lead to any problem

Ambiguous grammer:

A Grammer is said to be Ambiguous. if there exists two or more derivation tree for a string w (that means two or more left derivation trees)

eg: $G = (\{S\}, \{a, b, +, *\}, P, S\}$ where P consists of

$S \to S+S \mid S*S \mid a+ \mid a$

The string a+a*b can be generated as:

$\Rightarrow$ $S \to S+S$ | $S \to S*S$

$\to a+S$ | $\to S+S*S$

$\to a+S*S$ | $\to a+S*S$

$\to a+a*S$ | $\to a+a*S$.

$\to a+a*b$ | $\to a+a*b$.

Thus, this Grammar is Ambiguous.

Q Check ambiguity for following grammar:

$S \to aSSb \mid bSSa \mid a$.

Let aaaabbbb

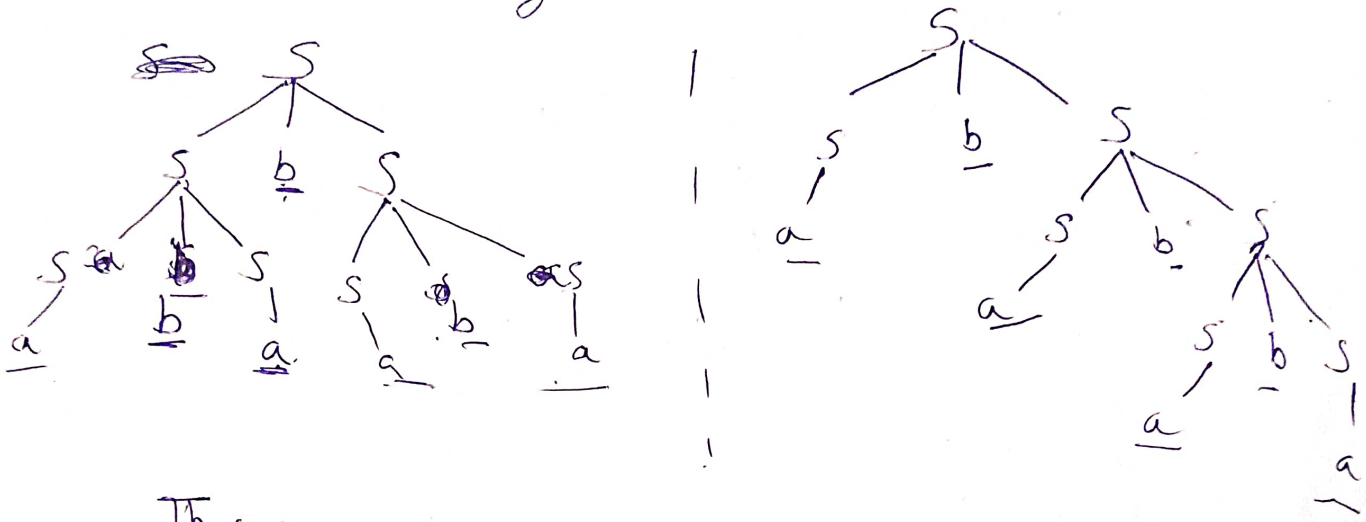
$\Rightarrow$

$S \to aSSb$

$aaSSb\ Sb$

(b) $S \Rightarrow sbs \mid a \mid \epsilon$ check ambiguity.

⇒ Let get the string abababa



This is ambigious

- Pushdown Automata (PDA):

A Pushdown Automata (PDA) is a type of formal automation in theoretical computer science. It is an extension of the finite automata with the addition of a stack, which provides it with more computational power and the ability to recognize context-free Languages.

PDAs are used to model and understand various aspects of computational systems and serve as the basis for parsing in programming languages and compilers.

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

- State / Variable
- Input
- Stack Alphabet
- Transition function
- Start State.
- Initial stack symbol
- Final State / Accepting State

↳ Instantaneous Description (ID):

Instantaneous Description (ID) is an informal notation of how a PDA "Computes" a input string and make a decision that string is accepted or rejected.

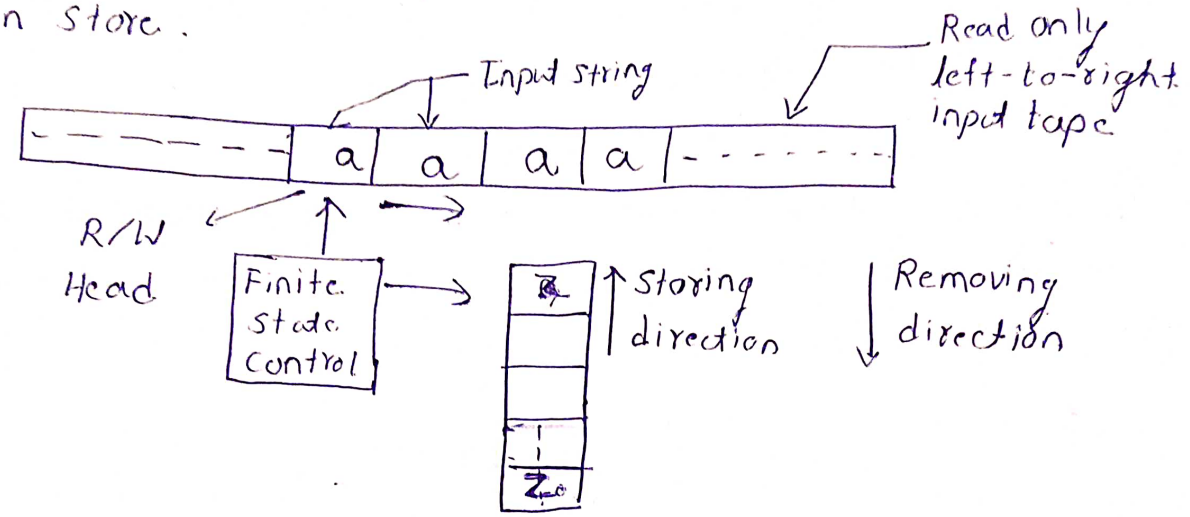
A ID is a triple (q, w, α), where:

1. q is the current state.
2. w is the remaining input.
3. α is the stack contents, top at the left.

(q, w, α)

Model of Pushdown Automata:

A push down automata is a finite state machine that is equipped with a memory device that functions as a push-down store.



Push down store (PDS)

PDA consist of four components:

(1) An input tape:
(2) A read/write head:
(3) A control unit
(4) A memory unit or stack.

} Explain this points futher. for 7/8 marks question

Q Define the pushdown automata for language.

$M = \{a^n b^n \mid n > 0\}$.

$\Rightarrow$ $M = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$ where $Q = \{q_0, q_1\}$ and $\Sigma = \{a, b\}$ and $\Gamma = \{A, Z\}$ and δ is given by

$\delta(q_0, a, Z) = \{(q_0, AZ)\}$

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

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

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

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

Note! Here,
- Length of the ~~th~~ string is fixed
- string start with [a] and end with [b]
- The no. of [a] = no. of [b]
- Push for [a] & ~~Pob~~ Pop for [b]

Let us see how this automata works for aaabbb

Row	State	Input	(Transition Function Used)	Stack	State after move.
1	q_0	aaabbb		Z	q_0
2	q_0	a aabbb	$\delta(q_0, a, Z) = \{(q_0, AZ)\}$	AZ	q_0
3	q_0	aa abbb	$\delta(q_0, a, A) = \{(q_0, AA)\}$	AAZ	q_0
4	q_0	aaa bbb	$\delta(q_0, a, A) = \{(q_0, AA)\}$	AAAZ	q_0
5	q_0	aaab bb	$\delta(q_0, b, A) = \{(q_1, \epsilon)\}$	AAZ	q_1
6	q_1	aaabb b	$\delta(q_1, b, A) = \{(q_1, \epsilon)\}$	AZ	q_1
7	q_1	aaabbb	$\delta(q_1, b, A) = \{(q_1, \epsilon)\}$	Z	q_1
8	q_1	ϵ	$\delta(q_1, \epsilon, Z) = \{(q_1, \epsilon)\}$	ϵ	q_1

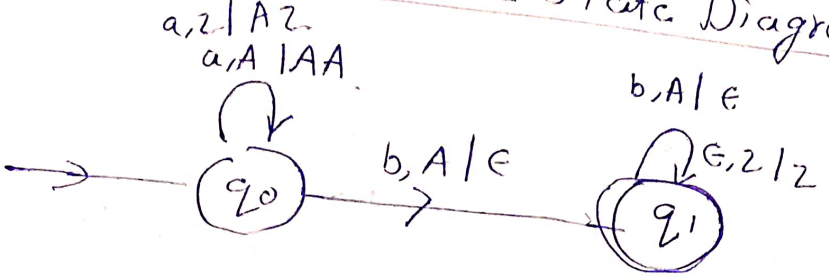
~~w~~ parsing and compiling programming language.

M

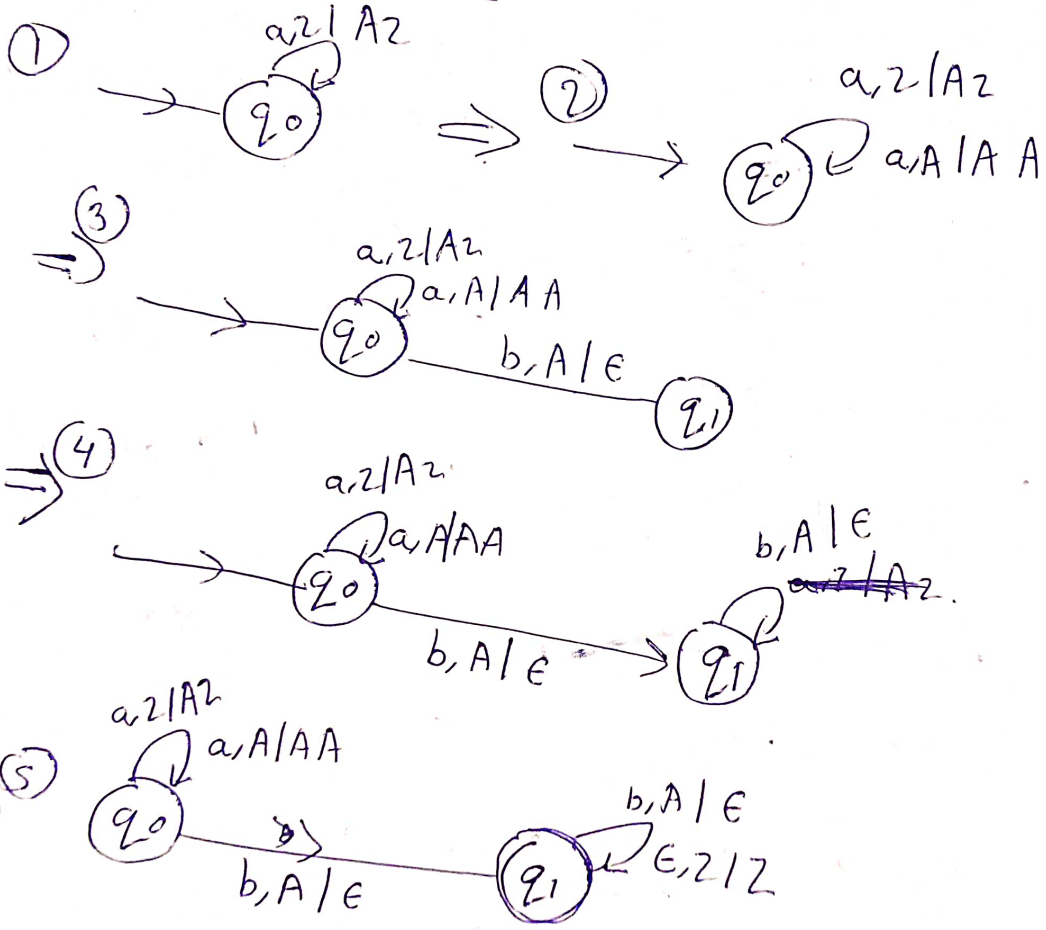
A

is eq
dow

Push Down Automata. State Diagram:

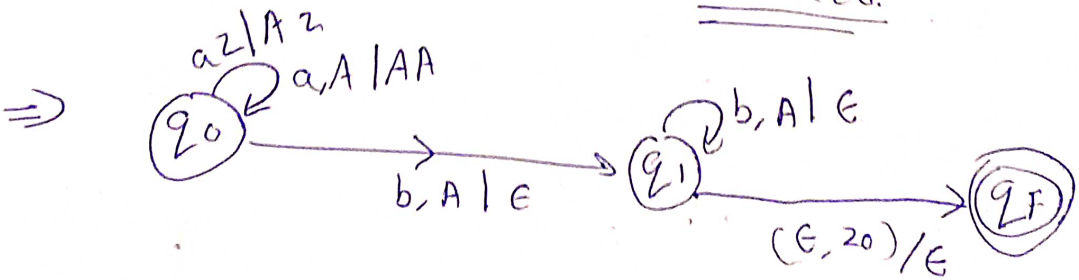


Stepwise below:



1
2
3
4

For futher extetinsion if need.



Transition (δ) will be:

$\delta(q_0, a, Z) = (q_0, 1Z_0)$

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

$\delta(q$

Q) Design PDA for $L = \{0^n 1^m 0^{m+n} \mid m, n \geq 1\}$

=> Here,

- The Language is fixed.
- All string start with 0 and end with 0.
- In between, string consist of 1.
- The number of 0's after 1 is equal to the starting 0's before one and number of 1's.

Valid strings: $L = \{0200, 001000, 00110000 \ldots\}$

~~Algorithm:~~ ~~Transitions~~

~~1. For 0~~ δ

Algorithm:

(i) For 0's before 1, push 0 to stack and for 1's push 1 to stack.

(2) For 0's after 1, if δ inp symbol is 0 stack top is 1 or 0, pop corresponding 1 or 0.

M

Transitions:

is eq
dow

$\delta(q_0, 0, z_0) \rightarrow (q_0, 0, z_0)$

$\delta(q_0, 0, 0) \rightarrow (q_0, 00)$

$\delta(q_0, 1, 0) \rightarrow (q_1, 10)$

$\delta(q_0, 1, 1) \rightarrow (q_1, 11)$

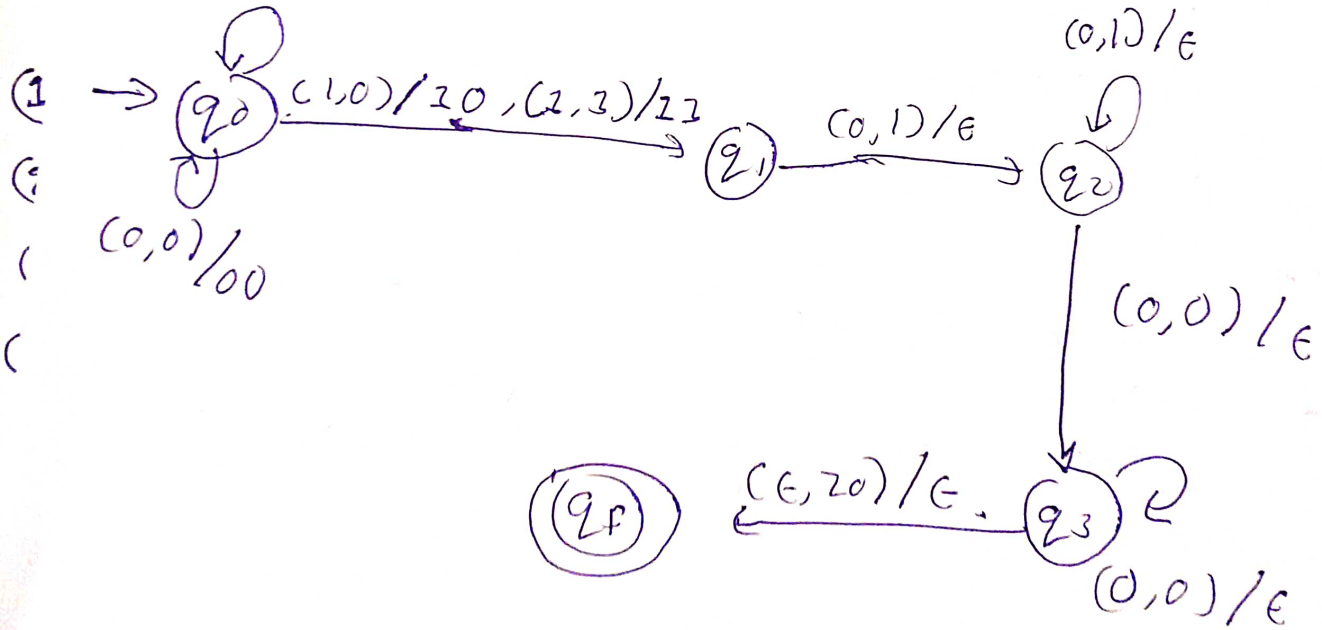
$\delta(q_1, 0, 1) \rightarrow (q_2, \epsilon)$

$\delta(q_2, 0, 1) \rightarrow (q_2, \epsilon)$

$\delta(q_2, 0, 0) \rightarrow (q_3, \epsilon)$

$\delta(q_3, 0, 0) \rightarrow (q_3, \epsilon)$

$\delta(q_3, \epsilon, z_0) \rightarrow (q_f, \epsilon)$



Q) Construct PDA for $L = \{a^m b^n c^p d^q \mid m+n = p+q\}$

⇒ Here,

- The language is fixed.
- string start from 'a' and end with d if $m, n, p, q \geq 1$
- Here $m+n = p+q$.

∴ According to formal defination

$Q = \{q_0, q_1, q_2, q_3, q_4\}$, $\Sigma = \{a, b, c, d\}$,

$\Gamma = \{1, 2\}$.

$L = \{abcd, aabcdd, \ldots\}$

Algorithm:

(i) If the input symbol 'm' or 'n' then push '1' to the stack.

(ii) If the input symbol is 'c' or 'd' then pop '1' from the stack

Transition:

$\delta(q_0, a, 2) = (q_0, 12)$

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

$\delta(q_0, b, 1) = (q_1, 11)$

$\delta(q_1, b, 1) = (q_1, 11)$

$\delta(q_1, c, 1) = (q_2, \epsilon)$

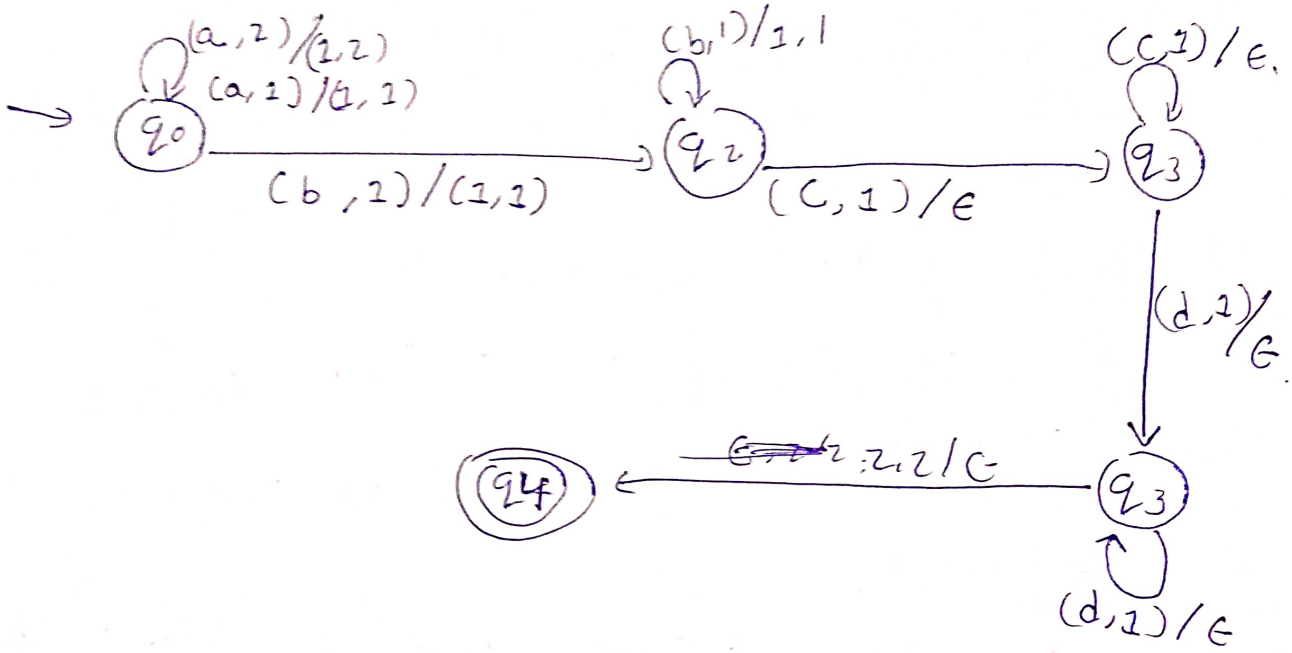
$\delta(q_2, c, 1) = (q_2, \epsilon)$

$\delta(q_2, d, 1) = (q_3, \epsilon)$

$\delta(q_3, d, 1) = (q_3, \epsilon)$

$\delta(q_3, \epsilon, 2) = (q_4, \epsilon)$.

Transition diagram :



Q Design PDA for $L = \{WW^R / W \in \{a,b\}^*\}$

(May be out of syllabus)

⇒ Here,
- Length of the ~~sto~~ string is variable.
- Starting and ending of the string can be a or b
- String is palindrome.

Note, soln from VBD, Usc.
Soln from next page for easyness

∴ Valid string:

$L = \{abba, baab, aabbbbaa, baaaaab, \ldots\}$

∴ According to Formal defination,

$Q = \{q_0, q_1, q_2\}$ and $\Sigma = \{a, b\}$ and $\Gamma = \{1, 2\}$

Case 1: ~~Starting~~ String start with 'a'
- Push '1' for input symbol 'a' & Pop '1' for input sumbol 'b'.
- When the stack came to initial state i.e '2' ~~then~~ and current input symbol is 'b' then push '1' for 'b' ~~in~~ and pop '1' for input symbol 'a'.
- Replace 'Zo' by 'ϵ'.

Case 2: String start with 'b'.
- Push '1' for input symbol 'b' & pop '1' for input Symbol 'a'.
- When the stack come to initial state i.e '2' and current input ~~is~~ symbol is 'a' then push '1' for 'a' and pop '1' for input symbol 'b'.
- Replace 'Zo' by 'ϵ'.

Transition:

$\delta(q_0, a, Z_0) \rightarrow (q_0, 1Z_0)$

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

$\delta(q_0, a, 1) \rightarrow (q_1, \epsilon)$

$\delta(q_0, a, 0) \rightarrow (q_0, a0)$

$\delta(q_0, b, Z_0) \rightarrow (q_0, 0, Z_0)$

$\delta(q_0, b, 0) \rightarrow (q_0, b0)$

$\delta(q_0, b, 0) \rightarrow (q_1, \epsilon)$

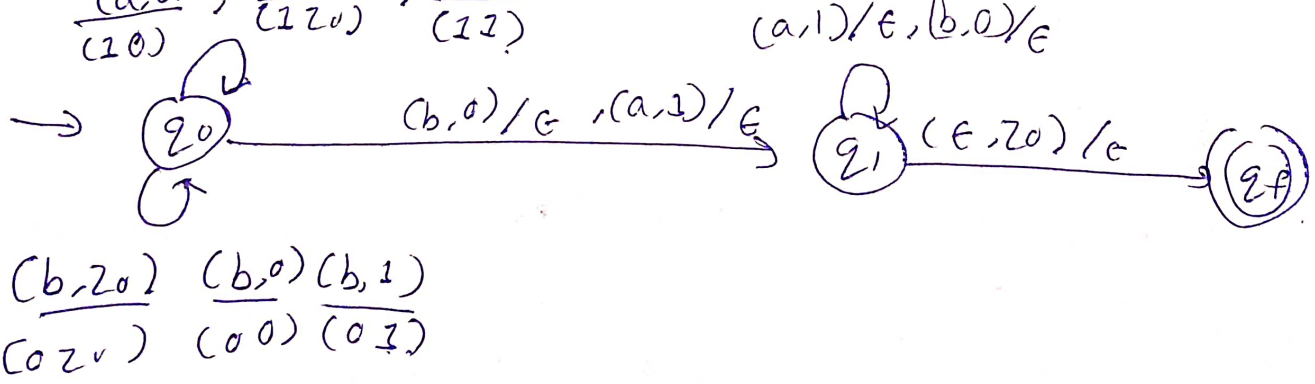
$\delta(q_0, b, 1) \rightarrow (q_0, 01)$

$\delta(q_1, a, 1) \rightarrow (q_1, \epsilon)$

$\delta(q_1, b, 0) \rightarrow (q_1, \epsilon)$

$\delta(q_1, \epsilon, Z_0) \rightarrow (q_f, \epsilon)$

Transition Diagram:



Q) Design PDA for $L = \{ WW^R / W \in \{a,b\}^* \}$.

Soln According to college lecture.

$\Rightarrow$

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

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

$F = \{q_0\}$

Valid string:

$L = \{abba, baab, aabbbbaa, baaaaab, \}$

∴ Transition:

$\delta(q_0, a, Z_0) = (q_0, a, Z_0)$
$\delta(q_0, b, a) = (q_0, ba)$] — string start with a.

$\delta(q_0, b, b) = (q_2, \epsilon)$

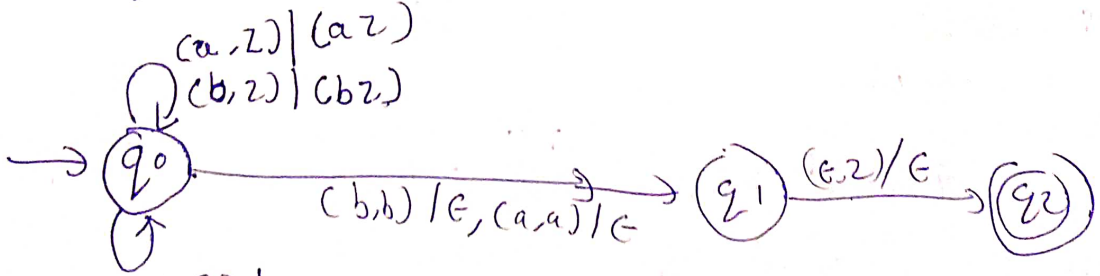
$\delta(q_1, a, a) = (q_2, \epsilon)$

$\delta(q_0, b, Z_0) = (q_0, b, Z_0)$
$\delta(q_0, a, b) = (q_0, ab)$] — string start with b

~~$\delta(q_0, a, a)$~~

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

- Diagram:



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

$\delta(a, b) | (ab)$

Q. PDA for $L = \{wcw^R \mid w \in \{a,b\}^*\}$

⇒

Valid string: $L = \{abcba, abbcbba, aabcbaa, \ldots\}$

Transition:

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

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

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

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

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

~~$\delta(q_0, c) = q_1$~~,

$\delta(q_0, c, a) = (q_1, az_0)$

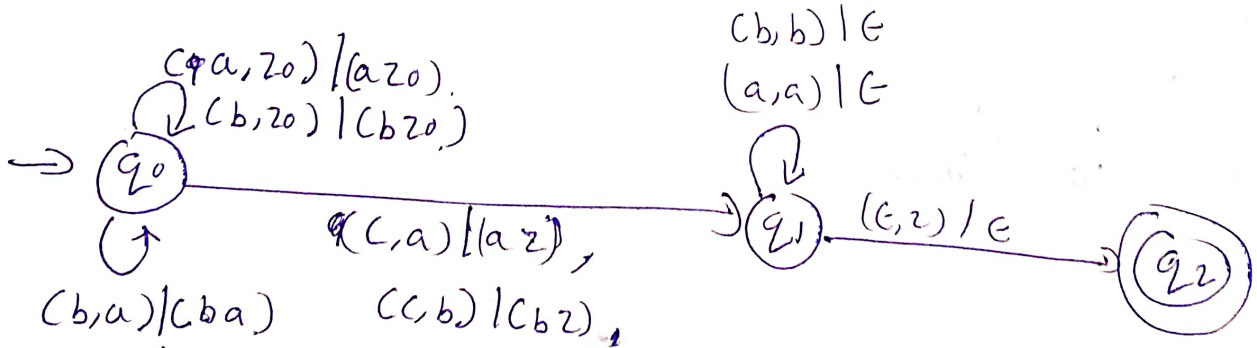
$\delta(q_0, c, b) = (q_1, bz_0)$

] string c is skip because it indicate middle term for PDA.

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

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

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



Must Cover ⇒ More question, Available in Mam's Notes

• Equivalence of Pushdown Automata and CFG:

Context-Free Grammers (CFGs) and Push Down Automata (PDAs) are two formalisms in Computer Science and automata theory that are equivalent in their language-defing power. This means that for every context-free language generated by a CFG, there exists a corresponding PDA that can recognize the same language and vice versa.

> **Note: CFG and PDA are equivalent in power:**
>
> A CFG generates a context free language and a PDA recognizes a context-free language. and the equivalent PDA to be used to implement its compiler. A language is context-free if some pushdown automata recognizes it.

- CFG are widely used tool for describing the syntax of programming languages and other structured languages. They consists of a set of variables (non-terminal), a set of terminals (symbols) a set of production rules, and a start symbol. These grammers are capable of generating context-free languages, which are a subset of the formal language
- On the other hand, PDAs are a type of automation equipped with a stack that allows them to recognize context-free languages. PDA have states, transitions, and a stack to store symbols. They can push and pop symbols from the stack during their operation, making them well-suited for recognizing nested structures in language.
- This equivalence plays a crucial role in study of formal languages, parsing and compiling programming languages.

Pending Sub topic:

(i) & CFG to PDA
(ii) PDA to CFG

(May be not in Syllabus)

No PYQ Available

Deterministic Pushdown Automata (DPDA):

DPDA is a variation of the pushdown automation used in automata theory. DPDAs are designed to recognize languages known as deterministic context-free languages (CFLs). Unlike non-deterministic pushdown automata (NPDAs), which can have multiple moves in each condition, ~~DPA~~ DPDAs have only one move from a given state and input symbol, making them deterministic in nature. This property simplifies the recognition process and allows for unambiguous parsing of context-free languages.

In TOC, DPDAs are commonly used to implement and parse context-free grammers (CFGs), especially in the context of compiler design and parsing algorithms. They are essential for processing the syntactic structure of programming languages and ensuring the correct interpretation of the source code.

DPDA are characterized by their ability to recognize context-free languages in a deterministic manner, making them a vital concept in formal language theory and automata Theory.