Theory Of Computation

Computation

- Any system which perform some result on certain operation.
- Computation is any type of calculation.
- It can be Manual or Automatic.

Manual Computation

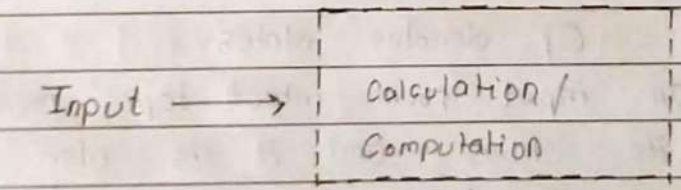
These computation involve human directly associated into the computation.

Ex :- Cloth washing, making tea

Automata :

A common name given to the family of automatic machine is called as Automata.

Automata Machine Model



Simple automata Model can be easily understand using a system which take some input, perform certain calculation / computation and gives some output.

Valid Input : These inputs are useful for computation.

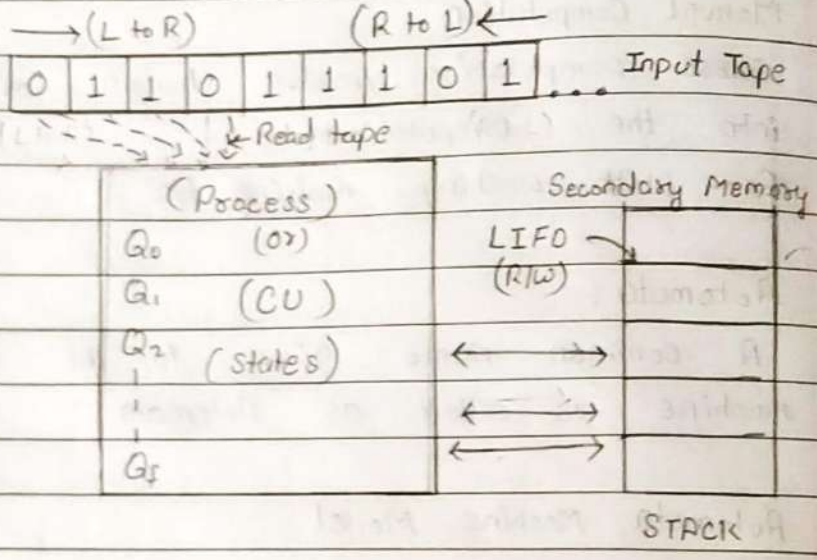
Invalid Input : These inputs are not useful under certain computation.

Date / /

States

- Finite Machines which take some input start at some point and goes to different instance / state.
- There will always be a starting point or starting state & always have a certain final state
- Ex:- Traffic light contains 3 state

Abstract Model of Computation / Digital Computation



→ Here $Q_0, Q_1, Q_2 \ldots Q_f$ denotes states

→ By reading an input from intput tape machine can change its states and it is also possible that state of machine not get change.

→ It is to be noted that all input symbols must be read / scanned and machine must reach to final state for finalization of computation.

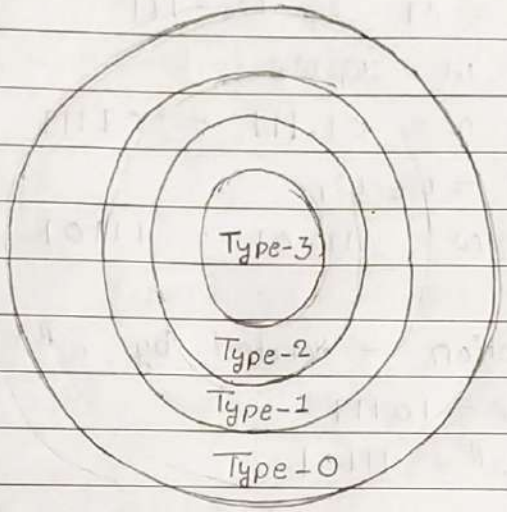
➔ The power of Automata or power of computation lies in secondary storage.

→ Secondary storage follow LIFO operation that is stack data structure.

→ On secondary storage ~~rig~~ write operation can also be done by an input tape, only read operation can be done in general.

CHOMSKY Hierarchy

Chomsky hierarchy is used for machine classification,



Type	Language	Grammar	Restriction	Range	Machine
Type 3	Regular Language	Regular Grammar	Most	Least	DFA & NFA
Type 2	Context Free Language	Context Free Grammar	Less than type 3	More than type-3	Push Down Automata (PDA)
Type 1	Context Senstive Language	Context Senstive Grammar	Less than type 2	More than type 2	Linear Bounded PDA
Type 0	Recursive Enumerable Language (REL)	Unristricted Grammar	No Restriction	Cover All	TURING Machine

String Operations

- String are basically the combination of alphabet
- In general string are denoted by 'ω'
- Ex:- 0, 1, 11, 00, 01, a, b, ab, ba

#1 Concatenation Operation - dot operator (.)

Ex:- $\omega_1 = 01$ & $\omega_2 = 111$

(1) $\omega = \omega_1 . \omega_2$

$\omega = 01 . 111 = 01111$

(2) $\omega = \omega_2 . \omega_1$

$\omega = 111 . 01 = 11101$

#2 Reverse Operation - denoted by ω^R

Ex:- $\omega = 10111$

$\omega^R = 11101$

#3 Length of String - Modules operator (|·|)

Ex:- $\omega = 1101$

$|\omega| = 4$

$\Rightarrow \boxed{|\omega| = |\omega_1| + |\omega_2|}$ $|\omega| = 5, |\omega_1| = 2, |\omega_2| = 3$

$|\omega| = 2+3 = 5$

$\Rightarrow \boxed{|\omega| = |\omega^R|}$

Empty String

- Empty string denotes a string having no length or zero length
- Generally empty string is shown to λ or ϵ

NoTe Empty string is highly useful and useful when state has to be change.

⇒ Few operation on empty string

- length always, $|\lambda| = |\epsilon| = 0$

Ex:- $\omega = 1101$

$\lambda \cdot \omega = \omega \cdot \lambda = 1101 \cdot \lambda$ or $1101 \cdot \epsilon$

$\Rightarrow \omega = 1101$

$[\omega = \lambda \cdot \omega = \omega \cdot \lambda]$

Here,

$[|\omega| = |\lambda \cdot \omega| = |\omega \cdot \lambda|]$

Kleene Closure

- Generally kleene closure is represented by (*) sign. symbol.
- It represent all length string and include empty string also.
- Ex:- $\Sigma = \{a\}$, $\Sigma^* = \{\epsilon, a, aa, aaa - - -\}$

(b) $\Sigma = \{a, b\}$, $\Sigma^* = \{\epsilon, b, a, aa, ab, bb, - - -\}$

- Kleene closure is always infinite in nature.

Positive Closure

- It is same as that of kleene closure except that it do not contain empty string (ϵ).
- It is also infinite in nature.
- Ex:- $\Sigma = \{a, b\}$, $\Sigma^+ = \{a, b, aa, ab, ba, - - -\}$

Language Operations

A language is a set of string all of which are chosen from some Σ^*, where Σ is a particular alphabet.

- A language is a subset of Σ^*.
- A language that can be formed over 'Σ' can be Finite or Infinite.
- Ex:- (a) Finite language

$L1 = \{$ set of 2 string $\}$

$L1 = \{ xy, yx, xx, yy \}$

(b) Infinite language

$L2 = \{$ set of all string (alphabet a-z) $\}$

$L2 = \{ \epsilon, a, ba, ab, bb, ac; \text{----} \}$

1 Concatenation

$L_1 = \{\omega_1\}, \quad L_2 = \{\omega_2\}$

Then, $L = L_1 \cdot L_2 = \{\omega_1 \cdot \omega_2\}$

Power or Closure

$L_1 = \{a, b\}, \quad L_1^{(3)} = ?$

$\Rightarrow L_1 = \{a, b\} \quad L_1^{(1)} = \{ab\}, \quad L_1^{(2)} = L^{(1)} \cdot L^{(1)} = \{abab\}$

$\Rightarrow L_1^{(3)} = L_1^{(1)} \cdot L_1^{(1)} \cdot L_1^{(1)} = \{ab\,ab\,ab\}$

Complement Operation

- It is denoted as L' (or) L^c for language L.
- Complement of language L is given as,

$$\boxed{L^c = L' = \{\Sigma^* - L\}} \quad \text{for } \Sigma$$

Ques:- Ex:- For alphabet set $\Sigma = \{a, b\}$ a language L is given $L = \{a, b, aa\}$ then find out the complement of given language.

Sol?

$L^c = \Sigma^* - L$

$L^c = \{a, b, \epsilon, ab, ba, aa, \text{---}\} - \{a, b, aa\}$

$L^c = \{\epsilon, ab, ba, bb, aaa, \text{---}\}$

Imp

Production Rules

Production rules allowed in type-3 machine are as given below,

P: 1.) $S \rightarrow aA$

2.) $A \rightarrow a$

3.) $A \rightarrow \epsilon$

Imp

NoTe: Type-3 machine are known as Finite State Machine (FSM)

The production rules are fixed in each type of machine under chomsky hierarchy.

Ques: For the given examples mention which production rules follows type-3 grammer

Ex 1:- P: $S \rightarrow a$ ✓
$S \rightarrow b$ ✓
$S \rightarrow \epsilon$ ✓
} follow type-3

Ex 2:- P: $S \rightarrow a$ ✓
$S \rightarrow bbX$
$S \rightarrow \epsilon$ ✓
} do not follow type-3

Ex 3:- P: $S \rightarrow aA$ ✓
$A \rightarrow \epsilon$ ✓
$A \rightarrow aaBX$
$B \rightarrow \epsilon$ ✓
} do not follow type-3

Ex 4:- P: $S \rightarrow aA$ ✓
$A \rightarrow \epsilon$ ✓
$A \rightarrow aB$ ✓
$BB \rightarrow aX$
} do not follow type-3

Ex 5:- P: $S \rightarrow aA$ ✓
$A \rightarrow \epsilon$ ✓
$A \rightarrow ab$ ✓
$bB \rightarrow aX$
} do not follow type-3 because only 1 non-terminal is allow in left side.

Ques: Based on given grammer whose production rule are given to rule no 1 to 3 as shown below

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

P: 1.) $S \to aS$
2.) $S \to \epsilon$
3.) $S \to b$

} or $S \to aS|\epsilon|b$

and what will be the language generated to the gammer.

Soln, First of all finding the possible derived string as shown below,

$S \xrightarrow{2} \epsilon$

$S \xrightarrow{3} b$

$S \xrightarrow{1} aS \xrightarrow{1} a$

$S \xrightarrow{1} aS \xrightarrow{3} ab$

$S \xrightarrow{1} aS \xrightarrow{1} aaS \xrightarrow{2} aa$

$S \xrightarrow{1} aS \xrightarrow{1} aaS \xrightarrow{3} aab$

Hence the language generated will be give as,

$L(G) = \{\epsilon, b, a, ab, aa, aab, aaa \ldots\}$

Ques: Mention the language generated by grammer.

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

$P : S \to aS|\epsilon$

Soln, Few possible string are shown below,

$S \to \epsilon$

$S \to a\boxed{S} \to a$

$S \to a\boxed{S} \to aa\boxed{S} \to aa$

Hence we can say that the language generated will be given as

$L(G) = \{ \epsilon, a, aa, aaa, --- \}$ - Informal notation

$L(G) = \{ a^n : n \geq 0 \}$ - Formal notation

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

$P: \quad S \rightarrow aS$

$A \rightarrow a$

Soln: We can clearly see that as per the given production rule it will never get terminated and hence no string can be generated $L(G) = \phi$

$S \rightarrow a[S] \rightarrow aa[S] \rightarrow aaa[S] - - -$

Also here A is a non-terminal which can never be reached hence it is a useless production.

Ques: $G = (\{S, A\}, \{0, 1\}, P, S)$

$P: \quad S \rightarrow 1S$

$S \rightarrow 1$

$S \rightarrow \epsilon$

Soln: $S \rightarrow \epsilon$

$S \rightarrow 1$

$S \rightarrow 1S \rightarrow 11$

$S \rightarrow 1S \rightarrow 11S \rightarrow 111$

$S \rightarrow 1S \rightarrow 11S \rightarrow 111S \rightarrow 1111$

$L(G) = \{ \epsilon, 1, 11, 111, - - - \}$ Informal notation

$L(G) = \{ 1^n : n \geq 0 \}$ Formal notation

Finite State Machines (FSM) / Finite State Automata

- Finite state machines are type-3 machine under Chomsky hierarchy.
- Under Finite state Machines, DFA & NFA
  DFA - Deterministic Finite Automata
  NFA - Non-Deterministic Finite Automata

- Under FSM or type-3 machine we have no external available memory or no secondary storage.

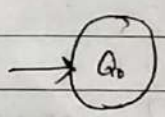
Note: Some internal memory or very small internal memory is available in finite state machine.

- The name finite state suggest that fixed number of state will be available in thes machines with no external storage.

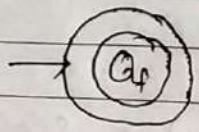
Deterministic Finite Automata (DFA)

Deterministic term in DFA signifies that the next possible move or next transition is fixed.

⇒ For DFA or NFA representation we use following symbol for representation,



Start state



Final State

⇒ Generally state is represent by Q

⇒ A DFA can be represented by 5 tuples

$$M = \{ Q, \Sigma, \delta, q_0, F \}$$

Q: set of all possible state
Σ: Alphabet set or Input available
δ: Transition or Transition Function $[Q \times \Sigma \to Q]$
q_0: Start state
F: Final state

V.Imp// In DFA from one state, when we are applying transition then we must remember two things -

1.> Each input alphabet at any state can be used only once.

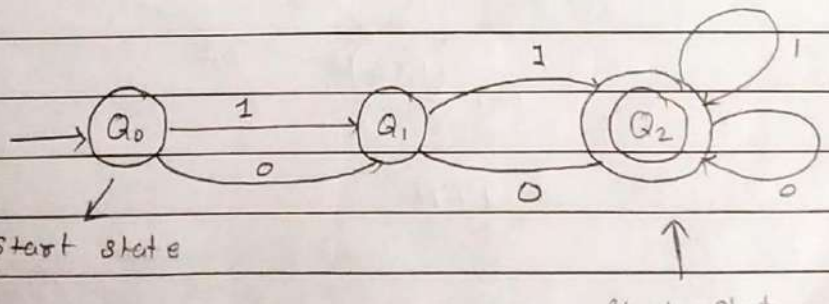
2.> No input can be leaved and useless input are generally used with the dead state (or) trap state.

⇒ Transition Diagram is generally known as DFA or NFA for the given question.

⇒ For any string acceptance, the transition must reach to final state starting from start state.

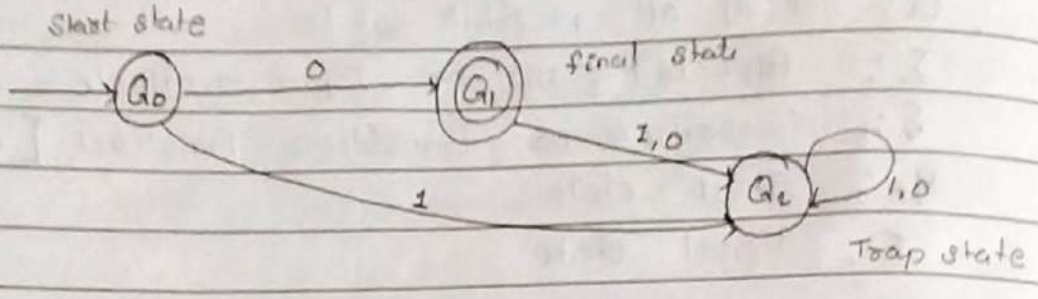
Ques: Draw a DFA that takes 2 string 0 and 1.

$\Sigma = \{0, 1\}$

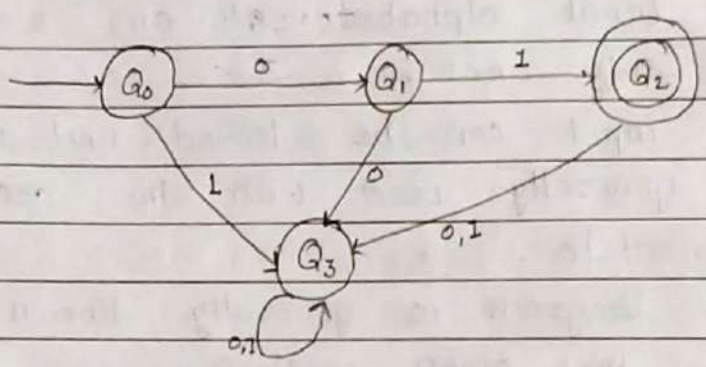


Ques: Write down a diterministic finite acceptor where the language accepted is,

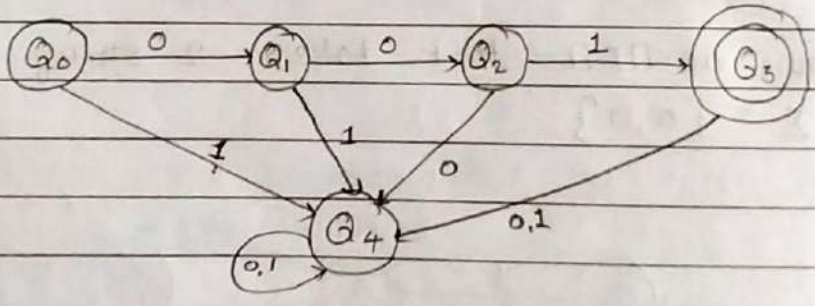
$$L = \{ 0 : \Sigma = \{0, 1\} \}$$



Ques: $L = \{ 0,1 : \Sigma = \{0,1\} \}$



Ques: $L = \{ 0^2 1 : \Sigma = \{0,1\} \}$



* DFA

Date / /

Ques: $L = \{ a^n b : \forall\ n \geq 0 , \Sigma = \{a, b\} \}$

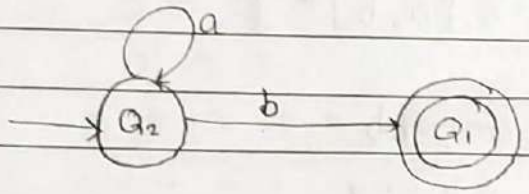
Soln, String accepted according to given language L are,

$\omega_1 = a^0 . b = b \quad n = 0$

$\omega_2 = a^1 b = ab \quad n = 1$

$\omega_3 = a^2 . b = aab \quad n = 2$

$\omega_4 = a^3 . b = aaab \quad n = 3$



* DFA

Ques: $L = \{ \Sigma^* : \Sigma = \{a, b\} \}$

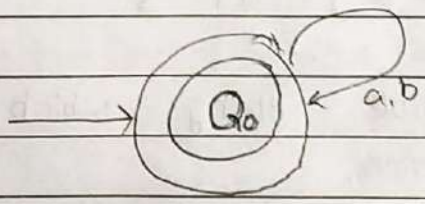
Soln, String accepted according to given language L are,

$\omega_1 = a^0 b^0 = \varepsilon$

$\omega_2 = a^1 b^0 = a$

$\omega_3 = a^1 b^0 = b$

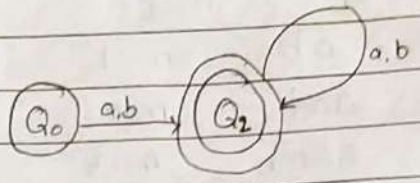
$\omega_4 \quad a^1 b^1 = ab$



* DFA

Ques: $L = \{ \Sigma^* : \Sigma = \{a, b\} \}$

Soln: String accepted according to given language L are, a, b, aa, bb, ab, ba - - - .



* DFA

Ques: (i) $L = \{ a\omega a : \omega \in \{a, b\}^* \}$

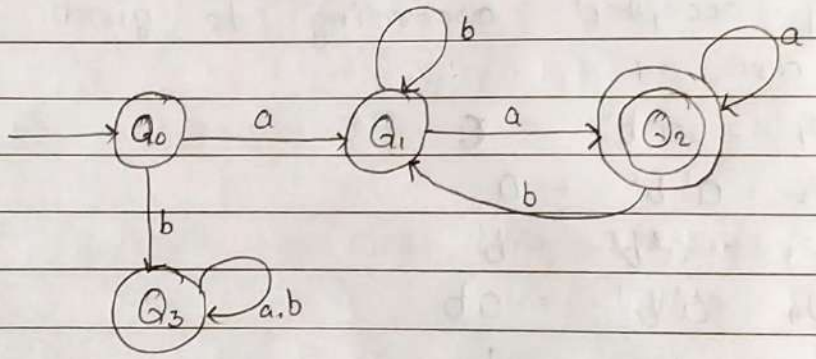
~~$\omega = \epsilon, a, b, aa, bb$~~

$\omega = aa$

$\omega = aaa$

$\omega = aba$

$\omega = abab a$



(ii) $L = \{ a \omega b : \omega \in \{a, b\}^* \}$

Soln: Few of the possible string which could be possible over here,

$\omega = \epsilon, a, b, ab, ba, aa, - - -$

$L = a \boxed{\omega} b = ab$

$L = a\,[w]\,b = aab$

$L = a\,[w]\,b = abb$

$L = a\,[w]\,b = aabb$

$L = a\,[w]\,b = abab$

Note: We will use self loop on state Q_1 in order to accept some acceptable string such as,

a b
a a b
a a a b
a a a a b
a a a a a b
- - - - -

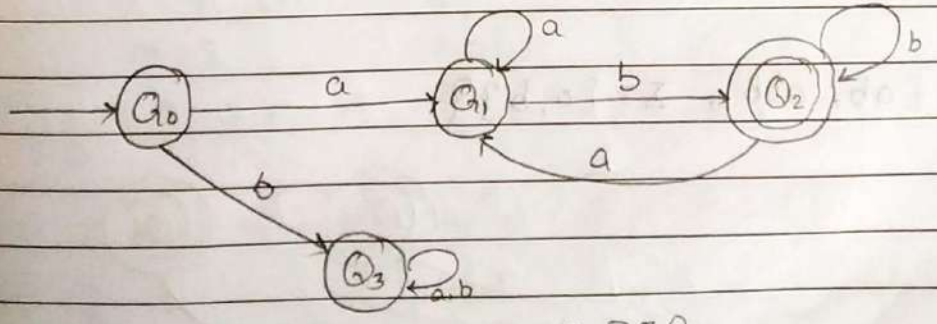
and similarly we will use self loop on state Q_2 with input b and so it will accept.

a b b
a b b b
a b b b b
a b b b b b
- - - -

Some more possible string could be -

abababb , aaabbbabab ... etc

Hence the input a of state Q_2 transition to Q_1

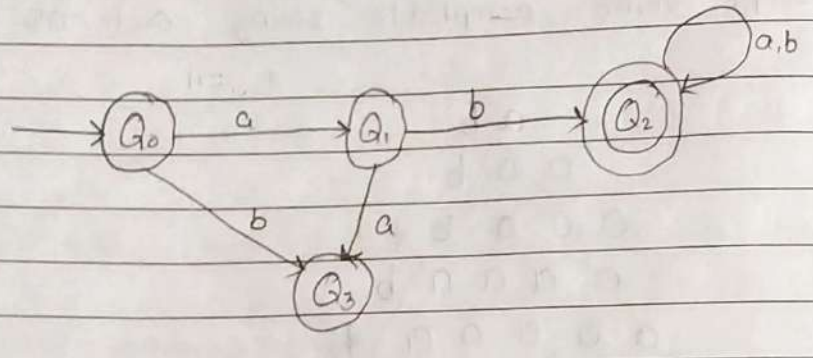


* DFA

Ques: Draw a DFA for $L = \{ abw, \Sigma = \{a, b\} \; w \in \{a, b\}^* \}$

(OR)

Draw a DFA for any language L and prefix must be ab



* DFA

Ques (i) $L = \{ wab, \Sigma = \{a, b\}, w \in \{a, b\}^* \}$

(ii) $L = \{ \epsilon, 0, 01, \Sigma = \{a, b\} \}$

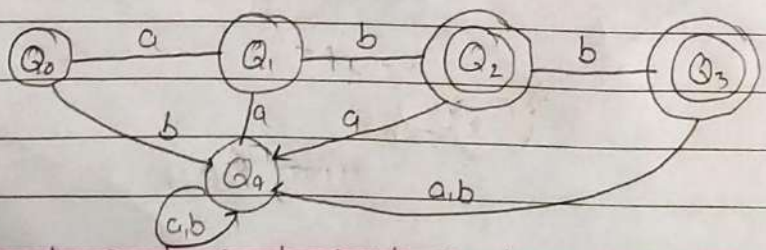
(iii) $L = \{ (ab)^n : n \geq 0 \; \Sigma = \{a, b\} \}$

(iv) $L = \{ ab, abb, \Sigma = \{a, b\} \}$

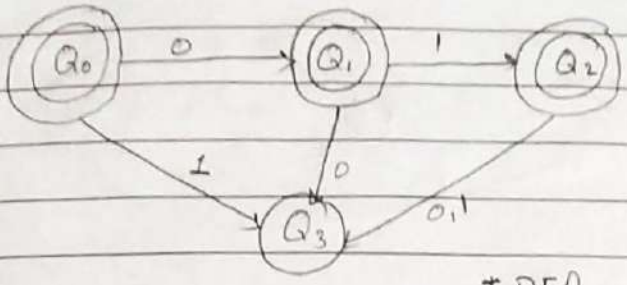
(v) $L = \{ (ab)^n a : n \geq 0, \Sigma = \{a, b\} \}$

Note: Remember that for any ~~string~~ machine we will always have a unique initial state however we can have more than one possible final state.

Sol^n // $L = \{ ab, abb, \Sigma = \{a, b\} \}$



$L = \{ \epsilon, 0, 01, \ \Sigma = \{0,1\} \}$



* DFA

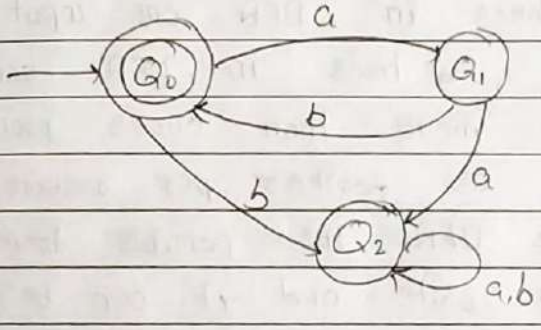
$L = \{ (ab)^n \ : \ n \geq 0, \ \Sigma = \{a,b\} \}$

Here put $n = 0$, $\omega = \epsilon$

$n = 1$, $\omega = ab$

$n = 2$, $abab$

$n = 3$, $ab\ ab\ ab$



* DFA

$L = \{ (ab)^n a \ : \ n \geq 0, \ \Sigma = \{a, b\} \}$

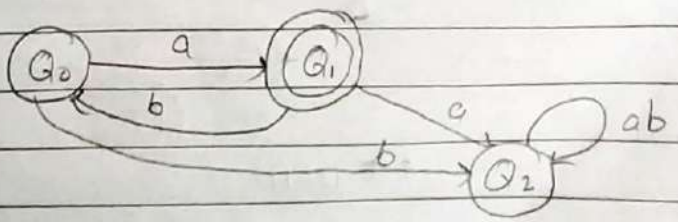
Sol^n

$n = 0$, $\omega = a$

$n = 1$, $\omega = aba$

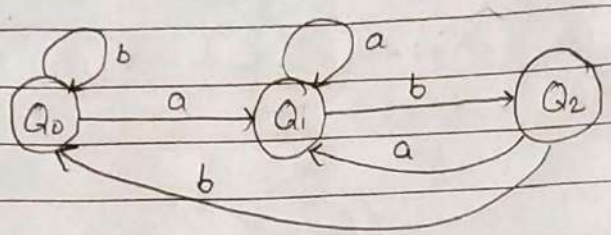
$n = 2$, $\omega = ababa$

$n = 3$, $\omega = ab\ ab\ ab\ a$



$L = \{ \omega ab : \omega \in \{a,b\}^*, \Sigma = \{a,b\} \}$

Here min. string possible is ab



Non-deterministic Finite Automata (NFA)

- In short denoted as NFA
- Basic state representation as same as that of DFA, however following few point must be obeyed while writing NFA

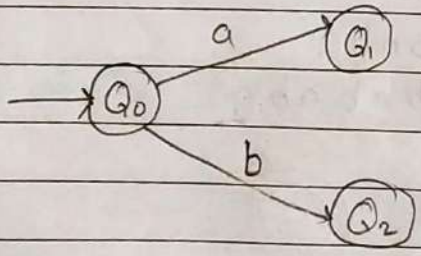
1.) Unlike DFA in NFA few inputs are allowed not to be taken

2.) Unlike DFA where in DFA one input can be taken onces only, where in NFA same input can be applied more than onces from any particular state as ~~further~~ per requirement.

3.) In NFA unlike DFA the possible transition is not a single state and it can be power set of total state.

4.) I NFA we can have empty string (ϵ) as an input.

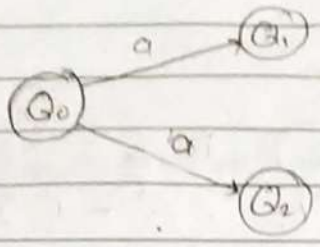
Point 1:



(No trap state)

* NFA

Point 2:



(taking same string)

Point 3:

In DFA,

$$[Q \times \Sigma \rightarrow Q]$$

In NFA,

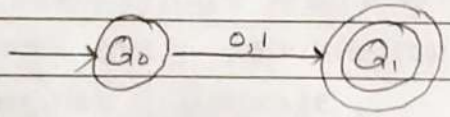
$$[Q \times \Sigma \rightarrow 2^Q]$$

Point 4:

$$Q \times \Sigma \cup \{\epsilon\} \rightarrow 2^Q$$

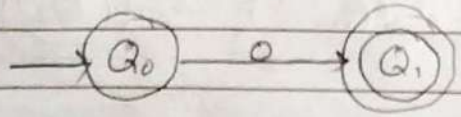
ϵ = empty string input

Ques: Draw an NFA that takes two string 0 or 1



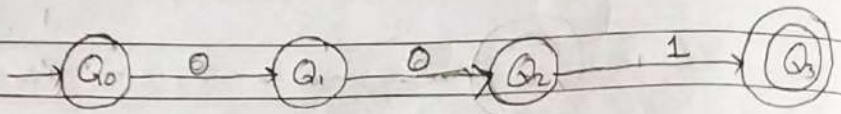
* NFA

Ques: $L = \{0 : \Sigma = \{0, 1\}\}$



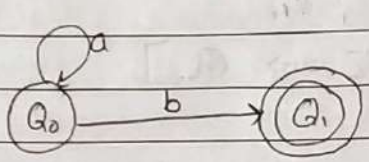
* NFA

Ques: $L = \{ 0^2.1 , \Sigma = \{0,1\} \}$



* NFA

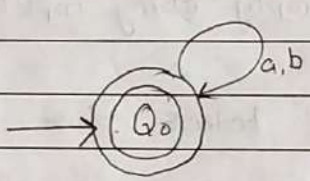
Ques: $L = \{ a^n b : n \geq 0 , \Sigma = \{a,b\} \}$



* NFA

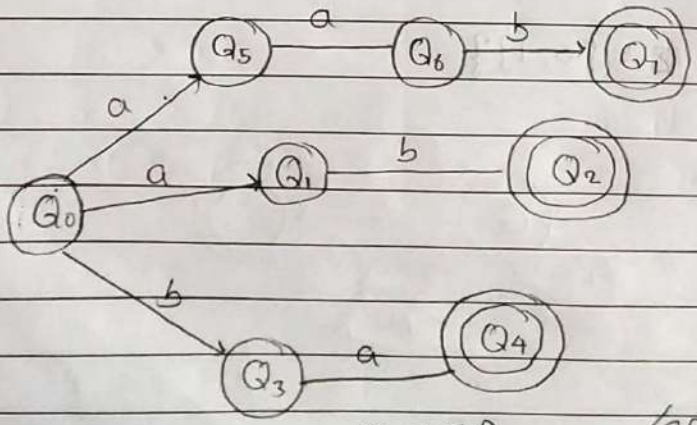
Ques: $L = \{ \Sigma^* : \Sigma = \{a,b\} \}$

$\Sigma^* = \{ \epsilon, a, b, aa, ab, ba, --- \}$

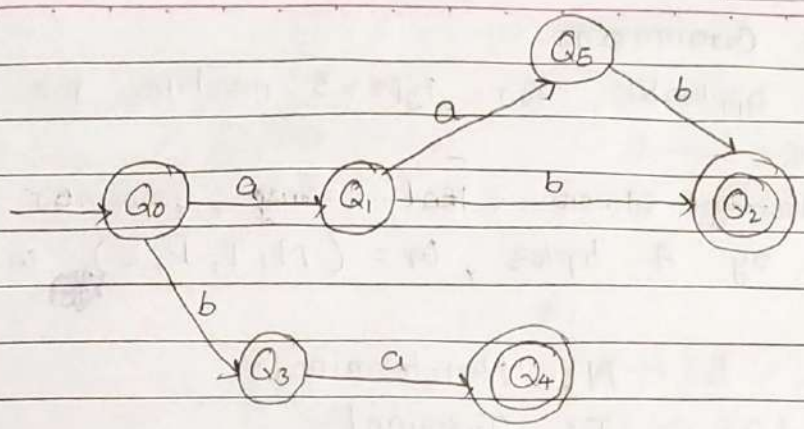


* NFA

Ques: $L = \{ ab , ba , aab ; \Sigma = \{a,b\} \}$

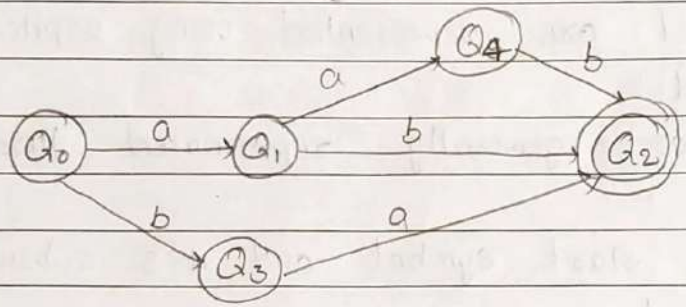


* NFA (OR)



* NFA

(or)



* NFA

Note : Remember that NFA and DFA are not unique for a given language, i.e we can have more than one possible DFA & NFA for a given regular language.

Most Imp

Note : 1.> If for a given language, we are able to draw or construct NFA or DFA then the given language will always be a regular language.

2.> Till now all question of DFA for given language and all the ques. of NFA for the given language in which we were able to draw NFA or DFA, those all languages are regular language.

Linear Grammar

It is applicable for type-3 machine for NFA | DFA machine.

We already know that any grammar G is define by 4 tuples, $G = (N, T, P, S)$ where,

N: Non-terminal
T: Terminal
P: Production Rule
S: Start Symbol $\in N$

- Non-terminal are represented using capital letter in general
- Terminal are generally represented through small letter.
- S denotes start symbol and it's subset of Non-terminal.

⇒ For linear Grammar or Type-3 Grammar we must remember that in production rule we can have atmost 1 Non-terminal on right hand side of production rule, i.e we can have the following possible representation,

P:

Right Linear	Left linear
1) $A \rightarrow xB$	1.) $A \rightarrow Bx$
2.) $A \rightarrow x$	2.) $A \rightarrow x$
$x \in T^*$	$x \in T^*$

Ques: Answer whether the given grammar shown through their production rule are linear grammar or not, If Yes then whether left linear or right linear.

1.) P:

$A \to 0B$ ×
$A \to BB$ ×
$B \to 1B$ ×

} Not linear

2.) P:

$A \to 01$ ✓
$A \to 001A$ ✓
$A \to 1$ ✓

} Right ~~Left~~ - linear

3.) P:

$A \to 10$ ✓
$A \to B0$ ✓
$B \to 111$ ✓

} left - linear

4.) P:

$A \to 1B$ ✓
$B \to 101B$ ✓
$B \to 1$ ✓

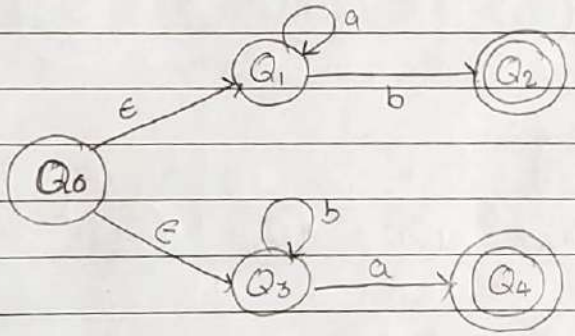
} Right linear

NFA with ϵ Transition

- We know that unlike DFA, in NFA we can have the ~~use~~ empty string (ϵ) as an input.
- Depending upon the need we can have the use of ϵ notation.

Ques: 2 Draw the NFA for given language

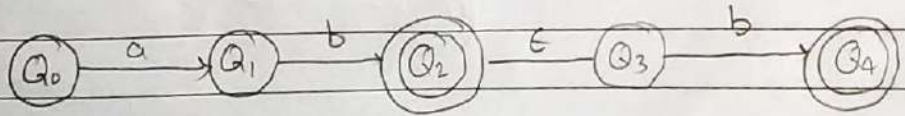
$L = \{\{a^n b\} \cup \{b^n a\}, \Sigma = \{a, b\}, n \geq 0\}$



≠ NFA

Ques: 1 Draw the NFA for the given language

$L = \{ab, abb\}$



≠ NFA

Date / /

Ques: 1. $L = \{ a^n b^n : n \geq 0, \Sigma = \{a, b\} \}$

2. $L = \{ a^n b^{2n} : n \geq 0, \Sigma = \{a, b\} \}$

3. $L = \{ a^{2n} b^{4n} : n \geq 0, \Sigma = \{a, b\} \}$

4. $L = \{ 0^{2n} 1^n : n \geq 0, \Sigma = \{0, 1\} \}$

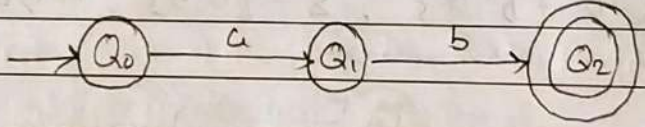
Soln, (i) $L = \{ a^n b^n : n \geq 0, \Sigma = \{a, b\} \}$

Some possible string generate from above given Language are,

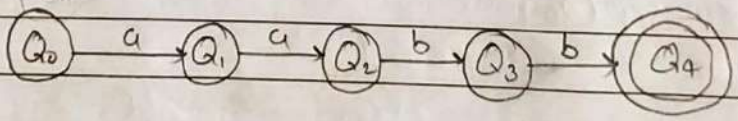
1. $L_1 = ab$
2. $L_2 = aabb$
3. $L_3 = aaabbb$

∞

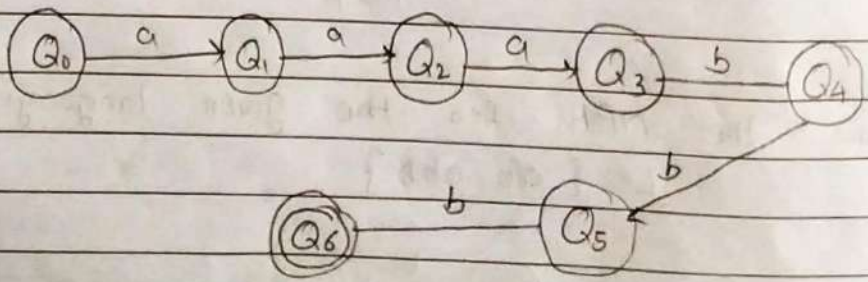
1.>



2.>



3.>



Camlin Page
Date / /

⇒ we already know that type-3 machine are not having any secondary memory.

Note: Since If for any given language if DFA is possible then NFA will also exist and vice-versa, that is both are equi-valent and same in term of performance or power.

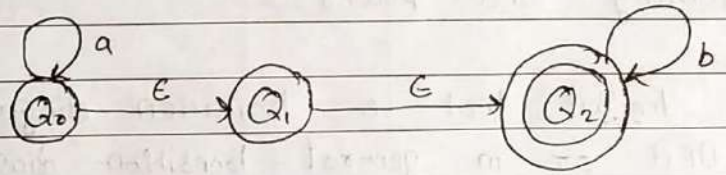
⇒ Since a & b are dependent on each other in terms of their occurence hence there is not upper bond and a & b occure large no. of time for infinite time.

An we know that in type-3 machine, their is no secondary or external storage Hence for the above 4 Question we can not draw corresponding NFA or DFA.

ϵ Closure

For any NFA we can write ϵ closure.

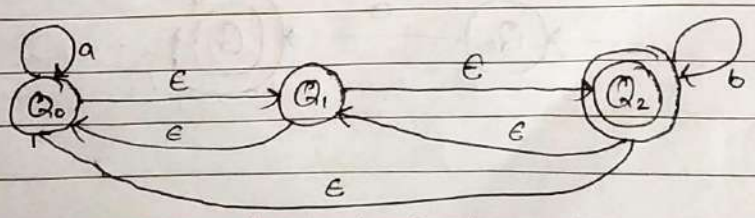
Ex:- (i)



Sol^n, ϵ closure,

$(Q_0) = \{Q_0, Q_1, Q_2\}$
$(Q_1) = \{Q_1, Q_2\}$
$(Q_2) = \{Q_2\}$

(ii)



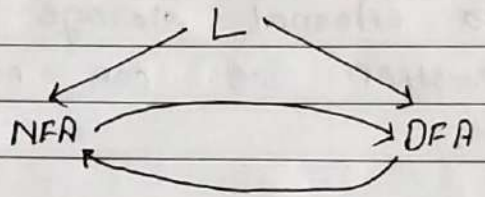
∈ Closure,

$(Q_0) = \{Q_0, Q_1, Q_2\}$
$(Q_1) = \{Q_1, Q_2, Q_0\}$
$(Q_2) = \{Q_2, Q_1, Q_0\}$

NFA to DFA Conversion

NFA $\longrightarrow Q \times \Sigma \cup \{\epsilon\} \longrightarrow 2^Q$
DFA $\longrightarrow Q \times \Sigma \longrightarrow Q$
Subset Construction

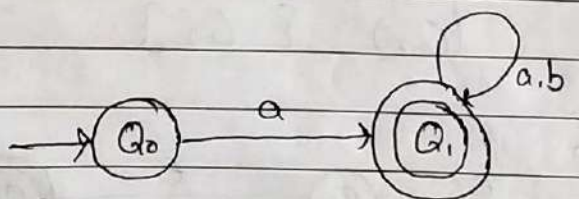
We know that any DFA and NFA is equivalent, i.e, if for any given language if NFA exist then DFA must exist and vice-versa.



i.e, NFA and DFA are equivalent in terms of compability and power.

NoTe: We known that a transition diagram denotes a DFA or in general transition diagram is known as DFA or NFA, however we can do the same thing i.e, can represent DFA or NFA through transition table

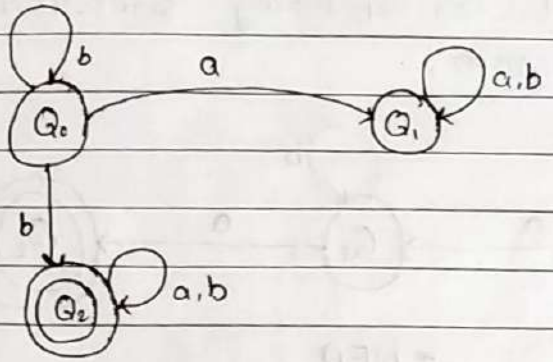
Example -



Transition Table

δ | Σ	a	b
→Q_0	Q_1	ϕ
Q_1^*	Q_1	Q_1

Ques: Write down the transition table for the given NFA



Transition Table

δ | Σ	a	b
→Q_0	Q_1	$\{Q_0, Q_2\}$
Q_1^*	Q_1	Q_1
Q_2^*	Q_2	Q_2

Stepts to convert NFA to DFA

Step 1: First of all find the initial state or start state of given NFA.

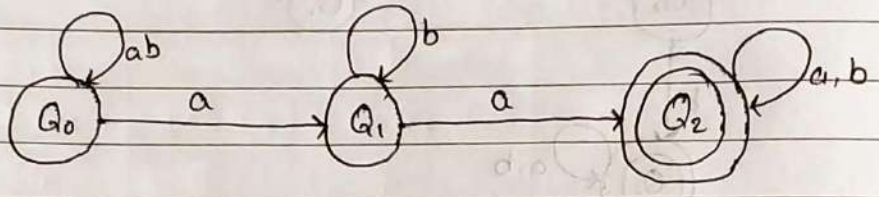
Step 2: Start performing transition to the given start state.

Step 3: Rename the new state by some name as received in step-2

Step4: Keep performing step number 3 until new state are forming.

Step 5: If new state carries any final state then make new state as also the final state.

Step 6: Write at last corresponding transition table and transition diagram.



*NFA

Transition Table

$\delta \mid \Sigma$	a	b
$\rightarrow Q_0$	$\{Q_0, Q_1\}$	Q_0
Q_1	Q_2	Q_1
Q_2^*	Q_2	Q_2

$\delta(Q_0, a) \longrightarrow \{Q_0, Q_1\}$ new state

$\delta(Q_0, b) \longrightarrow Q_0$

$\delta(\{Q_0, Q_1\}, a) = \delta(Q_0, a) \cup \delta(Q_1, a)$

$= \{(Q_0, Q_1)\} \cup \{Q_2\}$

$= \{Q_0, Q_1, Q_2\}$ – new state

$\delta(\{Q_0, Q_1\}, b) = \delta(Q_0, b) \cup \delta(Q_1, b)$

$= \{Q_0, Q_1\}$

$\delta(\{Q_0, Q_1, Q_2\}, a) = \delta(Q_0, a) \cup \delta(Q_1, a) \cup \delta(Q_2, a)$

$= (Q_0, Q_1) \cup (Q_2) \cup (Q_2)$

$\Rightarrow \{Q_0, Q_1, Q_2\}$

$\delta(\{Q_0, Q_1, Q_2\}, b) = \delta(Q_0, b) \cup \delta(Q_1, b) \cup \delta(Q_2, b)$

$= \{Q_0\} \cup \{Q_1\} \cup \{Q_2\}$

$= \{Q_0, Q_1, Q_2\}$

Since now there is no new state so the process will terminate

say -

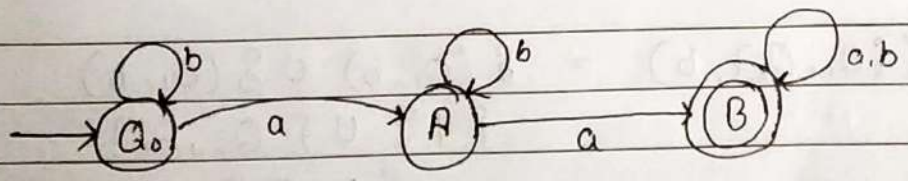
$\{Q_0, Q_1\} - A$

$\{Q_0, Q_1, Q_2\} \rightarrow B^*$

Now forming transition table

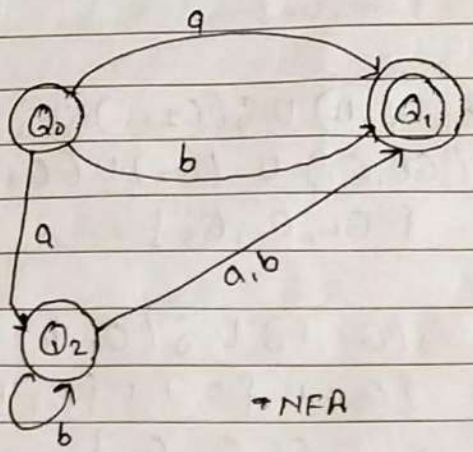
$\delta \mid \Sigma$	a	b
$\rightarrow Q_0$	A	Q_0
A	B	A
B^*	B	B

Transition Diagram,



* DFA

Ques: Convert the given NFA into DFA,



→NFA

Soln // Transition Table

δ|Σ	a	b
→Q_0	$\{Q_1, Q_2\}$	Q_1
Q_1	ϕ	ϕ
Q_2	Q_1	$\{Q_1, Q_2\}$

Now finding all the transition from starting state,

$\delta(Q_0, a) \rightarrow \{Q_1, Q_2\}$ → new state
$\delta(q_0, b) \rightarrow \{Q_1\}$

$\delta(\{Q_1, Q_2\} a) = \delta(Q_1, a) \cup \delta(Q_2, a)$
$\Rightarrow \{\phi\} \cup \{Q_1\}$
$\Rightarrow \{Q_1\}$

$\delta(\{Q_1, Q_2\}, b) = \delta(Q_1, b) \cup \delta(Q_1, b)$
$\{\phi\} \cup \{Q_2, Q_1\}$
$= \{Q_2, Q_1\}$

$\delta(Q_1, a) \rightarrow \phi$ → new state
$\delta(Q_1, b) \rightarrow \phi$

$\delta(\phi, a) \rightarrow \phi$

$\delta(\phi, b) \rightarrow \phi$

Since now there is no new state. so the process will terminate

Say,

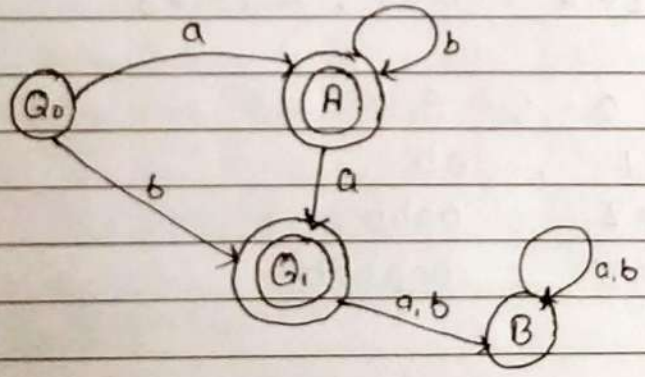
$\{Q_1, Q_2\} = A^*$

$\{\phi\} = B$

Now forming transition table

$\delta \mid \Sigma$	a	b
$\rightarrow Q_0$	A	Q_1
A^*	Q_1	A
Q_1	ϕ	ϕ
B	ϕ	ϕ

Transition diagram



* DFA

Note: In case if there is no transition then that state is known as empty state (ϕ) and we can rename it also by any other name.

Regular Language

The language accepted by Finite state Machine (FSM) contain NFA & DFA is known as Regular Language.

Ques: Prove that the given language is regular language.

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

Soln

*NFA

Since we are able to draw NFA, so the given language is regular language.

Ques: Show / Prove that the given language is regular, if it is not then why not ?

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

Soln

$n = 0$, $\in$

$n = 1$, ab

$n = 2$, $aabb$

$n = 3$, $aaabbb$

$n = aaa \cdots bbb \cdots$

We already know that Type-3 machine or FSM have limited memory and no extra external storage. Since the number of a's and no.

of 'b' are same, while writing a & b we must remember that how many 'a' were there and to write 2nd 'b' then also remember how many 'b' we have taken & This will be true for every attempts of 'b'.
Hence the give language is not a regular language.

Ques: $L = \{a^{2n} b^n : n \geq 0\}$, $\Sigma = \{a, b\}$

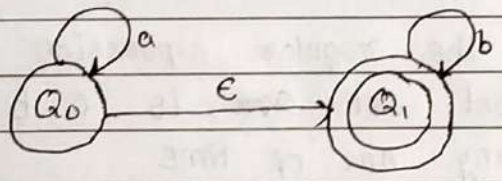
Ques: $L = \{0^{2n} 1^n, n \geq 0\}$, $\Sigma \{0, 1\}$

Soln: Same as above Question

Ques: $L = \{a^n b^n, \Sigma = \{a, b\}\}$

The NFA and corresponding DFA can be given for the following expression.

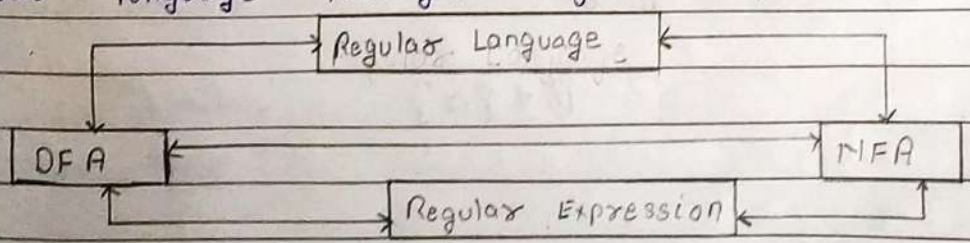
Ques: $L = \{a^n b^m : n, m \geq 0, \Sigma = \{a, b\}\}$



* NFA

Regular Expression

It 's a simple thing and used to denote the regular language through regular expression



Camlin | Page
Date / /

No Te: A regular expression can contain following operator '+', '.', '*', '()'

Ques: 1 a.) Write down the regular expression for empty string (ϵ), Null string (ϕ), any string a.

$$R_1 = \{\phi\}, \quad R_2 = \{\epsilon\}, \quad R_3 = \{a\}$$

b.) Write down the regular expression for the language $L = \{0, 1\}$

$$R = \{0+1\} \quad \text{or} \quad \{1+0\}$$

c.) $L = \{0, 1, 01\}$

$$R = \{0, +1+01\}$$

(OR)

$$R = \{1+0+01\}$$

Ques: 2 a.) Write down the regular expression for a language where alphabet set ~~are~~ is a, b and only a are allowed any no. of time

$$R = \{a^*\}$$

b.) Write down the regular expression for a language where alphabet set are a, b and it is given that this language must have a.

$$R = \{a\}$$

c.) same as a2|b part but the language must have 'b'.

$$R = \{ b \}$$

d.) String contain any number of a & b.

$$R = (a+b)^* \quad (or) \quad (b+a)^*$$

e.) Must contain a and there is no restriction before and after this.

$$R = \{ (a+b)^* . a . (a+b)^* \}$$

Ques: Write down the regular Expression for a language in which language start with two a (aa).

$$R = \{ aa . (a+b)^* \}$$

b.) Same as above, but double 'b' (bb) in last.

$$R = \{ (a+b)^* . bb \}$$

Ques: Atleast we can have either a or bb

$$R = \{ (a+b)^* . (a + bb) \}$$

For the above Regular expression we can investigate that,

bb ✓
aaa ✓
abb ✓
ba ✓

Ques: Write down a regular expression in which 'a' must occur even no. of times

$$R = \{(b)^* . (aa)^* . (b)^*\}$$

Ques: $\Sigma = \{a, b, c\}$ & one c must come.

$$R = \{(a+b+c)^* . c . (a+b+c)^*\}$$

Ques: Occurence of a is divisible by 3 & $\Sigma = \{a, b\}$

$$R = \{(b)^* . a . (b)^* . a . (b)^* . a . (b)^*\}$$

NoTe: The language which are not regular are non-regular language to prove that, the given language are not regular we can use Pumping Lemma Method.

⇒ Pumping Lemma is based on Pigeon Hole Principle

Pigeon Hole Principle

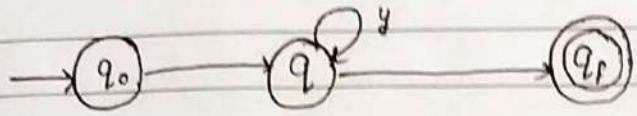
Suppose we have m object which are to be placed in 'n' number of boxes -
Boxes will be repeated in order to place object on to them, that is repetation of object will be their. ($m > n$)

- Unique Identification / Distribution not Possible
- Useful in finite automata to show non-regular property.

- Pigeon Hole Principle in Finite Automata tells about repetation of states, that is unique distribution is not possible

Pumping Lemma (Theorem) for Regular Language

- To show non-regular



- Middle position 'y' is pumped
- If 'L' is a regular language then after pumping all the string ω (say) $\in L$, must also $\in L$.

- Statement: If L is a regular language and $\exists$ n, a positive integer such that string $\omega \in L$ for string $x, y, z \in \Sigma^*$ where $\omega = xyz$

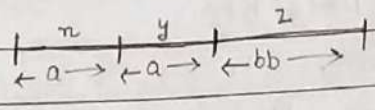
1. $|xy| \leq n$
2. $|y| \geq 1$
3. $\forall\ i \geq 1$, $xy^iz \in L$ (for all)

Ques: $L = \{a^n b^n : n \geq 0\}$, $\Sigma = \{a, b\}$ prove that the given language L is non-regular language using pumping Lemma for regular language

Soln: Assume that the given language L is a regular language ~~such that string $\omega \in L$~~ and $\exists$ n, a positive integer such that string $\omega \in L$ for string $x, y, z \in \Sigma^*$ where $\omega = xyz$

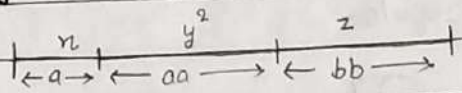
- $|xy| \leq n$
- $|y| \geq 1$
- $\forall\ i \geq 1$, $xy^iz \in L$

Case 1: Middle Portion contains 'a' only $i = 1$, $\omega = xyz$



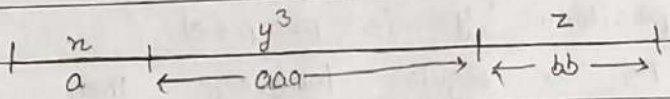
$\omega \in L$

$i = 2$, $\omega = xy^2z$



$\omega = xy^2z \notin L$

$i = 3$, $\omega = xy^3z$



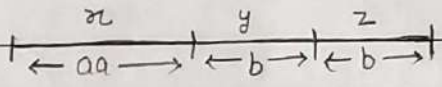
$\omega \notin L$

Similarly for $i = 4, 5, \ldots$ $\omega \notin L$

Hence L is not regular.

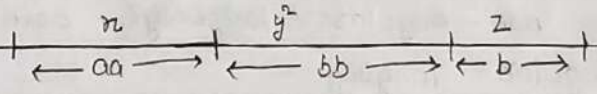
Case 2: Middle Portion Contain 'b' only

$i = 1$, $\omega = xyz$



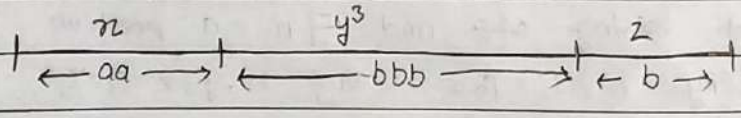
$\omega \in L$

$i = 2$, $\omega = xy^2z$



$\omega = xy^2z \notin L$

$i = 3$, $\omega = xy^3z$



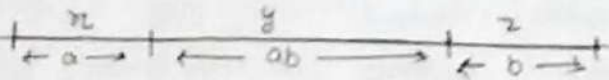
$\omega \notin L$

Similarly for $i = 4, 5, 6 \ldots$ $\omega \notin L$

Hence L is not regular.

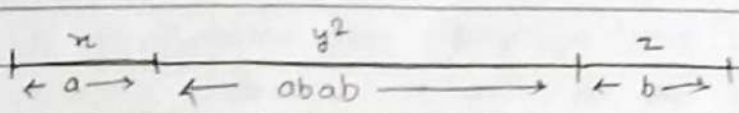
Case 3 Middle portion contains combination of a and b.

$i = 1$, $\omega = xyz$



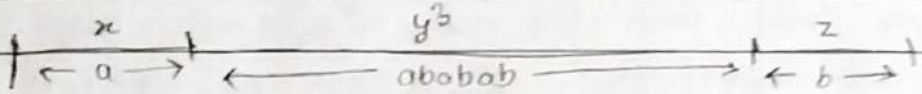
$\omega \in L$

$i = 2$, $\omega = xy^2z$



$\omega \notin L$

$i = 3$, $\omega = xy^3z$



$\omega \notin L$

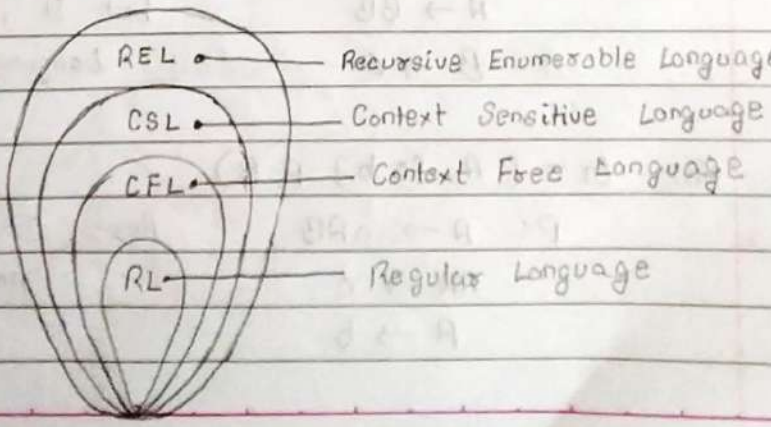
Similary for $i = 4, 5, 6 \text{ --- } \omega \notin L$

Hence L is not regular

Context Free Language

- While defining context free language there is no restriction on right hand side of Production Rule.
- Most of the Programming Language follows context free language properties.
- Context free language falls under type-2 of Chomskey hierarchy.
- In Type-2 or context free language we shall be having secondary storage as well in the form of Stack (LIFO).
- Ambigoity should not be present while defining context free language.
- Like Regular language where we have used Finite State Machine (NFA, DFA). In context free language Push-Down Automata or Push-Down Machine are used for processing

Type	Language	Grammar	Machine
Type-3	Regular Language	Regular Grammar	Finite State Machine (NFA, DFA)
Type-2	Context Free language	Context Free Grammar	Push Down Automata (PDA)



⇒ All regular language are subset of Context Free language however vice-versa is not true

⇒ The Grammar is known as Context-Free Grammar containing 4 tuples $G = (N, T, P, S)$

G: Context Free grammar where Production Rule is define as $P: S \rightarrow (NUT)^*$

⇒ Since there is no restriction on right hand side of production rule Hence this language is known as Context Free language.

Note: Most of the programming language follows Context Free Properties example C, etc.

- The next application of Context Free language are in COMPILER CONSTRUCTION, PHRSE TREE generation.

Ques: Some Production Rule are given classified them into Regular Grammar, Context Free Grammar.

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

P: $A \rightarrow aA$
$A \rightarrow a$

Ans: both regular and CFL as well

(ii) $G = [\{A, B\}, \{0, 1\}, P, A\}$

P: $A \rightarrow aAb$
$A \rightarrow BB$
$B \rightarrow \epsilon$

Ans: It is not regular language but it is Context Free Language

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

P: $A \rightarrow aAB$
$AA \rightarrow a$
$A \rightarrow b$

Ans: It is neither Regular nor Context Free Language.

Generation of Derivation Tree

- Derivation tree are also known as Parse tree and these process is known as Parsing.
- While defining semantics or language representation Compiler construct Parse tree or derivation tree in order to define the meaning.
- The generation of derivation tree or Parse tree can be performed in left most manner or right most manner.
- The left most conversion is known as Left most derivation tree and right most conversion is known as Right most derivation tree.
- Both in leftmost or in right most one non-terminal will be replace by production rule one at a time and either of 2 approaches will be perform.

Ques: Generate the derivation tree or Parse tree for the given Context Free Grammar.

$G = \{\{S\}, \{a, b\}, P, S\}$

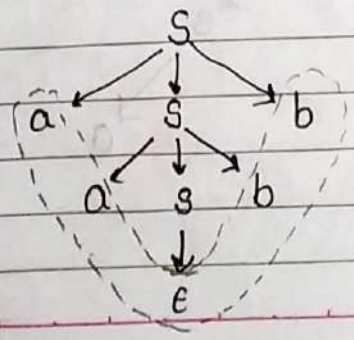
P: 1.> $S \rightarrow aSb$

2.> $S \rightarrow \epsilon$

For string $\omega = aabb$

Sol^n, Derivation tree or Parse tree,

$$S \xrightarrow{1} aSb \xrightarrow{1} aaSbb \xrightarrow{2} aabb$$



* Derivation Tree

Date / /

NoTe: Here in the derivation tree representation ; since only one non-terminal is used on right hand side.

Ques: For the question mentioned below write down left-most and right most derivation tree.

$G_1 = \{\{S\}, \{a, b\}, P, S\}$ $\quad w_1 = aa$

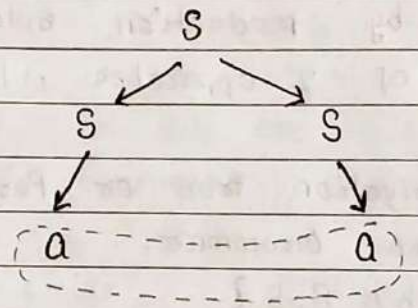
P: 1) $S \rightarrow SS$

2) $S \rightarrow a$

Sol^n,

Left most derivation Tree

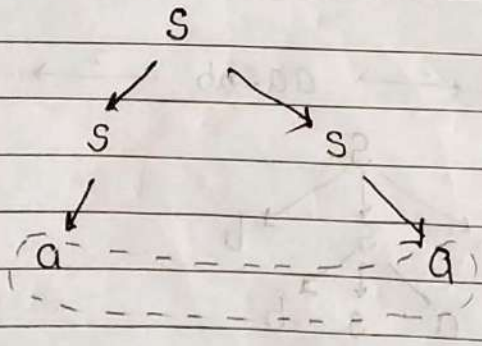
$$S \xrightarrow{1} SS \xrightarrow{2} aS \xrightarrow{2} aa$$



* Left most derivation Tree

Right most derivation tree,

$$S \xrightarrow{1} SS \xrightarrow{2} Sa \xrightarrow{2} aa$$



* Right most derivation tree

v.v imp

Ques: Write down the derivation tree for given Context Free Grammar

$G = \{ \{S\}, \{a, b, c\}, P, S \}$

$G = \{ N, T, P, S \}$

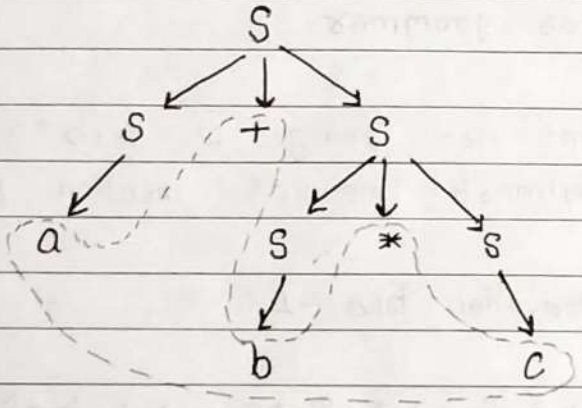
P: $S \rightarrow S+S \mid S-S \mid S*S \mid a \mid b \mid c$

$\omega = a+b*c$

Soln, Left most derivation Tree

$S \longrightarrow S+S \longrightarrow a+S \longrightarrow a+S*S \longrightarrow a+b*S$

$a+b*c \longleftarrow$

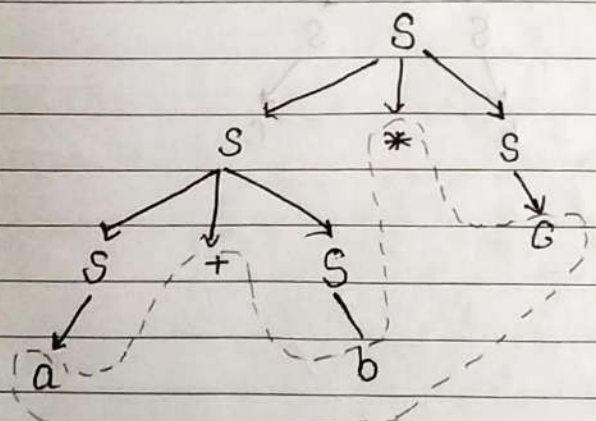


* Left most derivation Tree

Right most derivation Tree

$S \longrightarrow S*S \longrightarrow S*c \longrightarrow S+S*c \longrightarrow S+b*c$

$a+b*c \longleftarrow$



* Right most derivation tree

Ambiguity in Context Free Grammar

- Ambiguity means anything which is having more than one meaning.
- In Context Free gammar can not be ambiguous.
- In order to find out the ambiguity in given grammar, if we are able to draw more than one leftmost or right most derivation tree then given grammar will be ambiguous grammar.

Ques: Show that the given grammar,

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

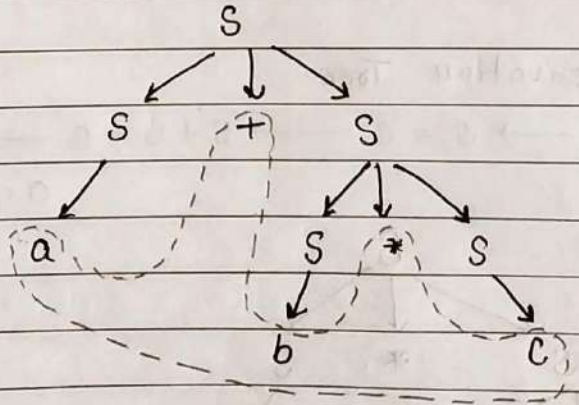
$P: S \rightarrow S+S \mid S-S \mid S*S \mid a \mid b \mid c$

is ambiguous grammar

Soln, Let us assume for string $\omega = a+b*c$, we are generating leftmost tree as mention below,

Left most derivation tree -1

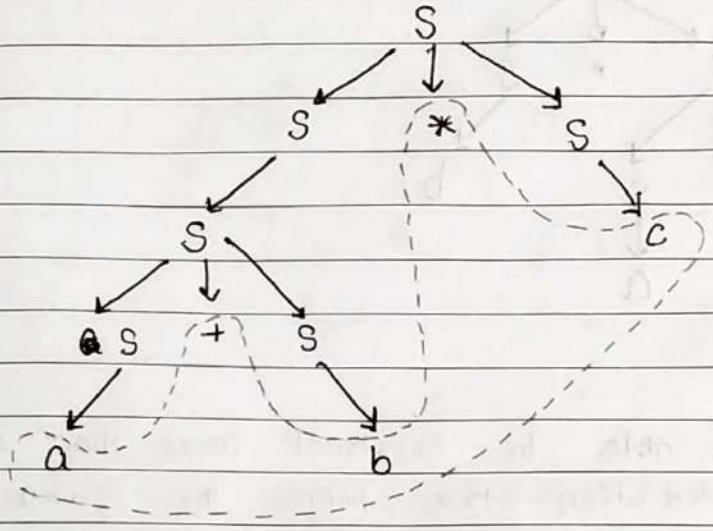
$S \rightarrow \boxed{S}+S \rightarrow a+\boxed{S} \rightarrow a+\boxed{S}*S \rightarrow a+b*\boxed{S} \rightarrow a+b*c$



Left most derivation tree - 2

$S \rightarrow \boxed{S} * S \rightarrow \boxed{S} + S * S \rightarrow a + \boxed{S} * S \rightarrow a + b * \boxed{S}$

$a + b * c$ ←



Since we are able to construct more than one left most tree hence the given grammar is ambiguous.

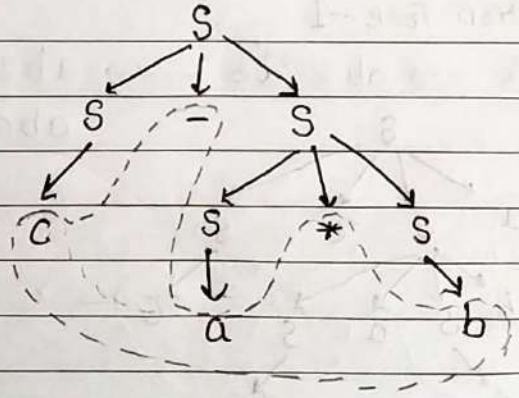
H/W Part of previous Question,

Ques: For string $\omega = c - a * b$, Right most derivation tree

Soln, Right most derivation Tree - 1

$S \rightarrow S - \boxed{S} \rightarrow S - S * \boxed{S} \rightarrow S - S * b$

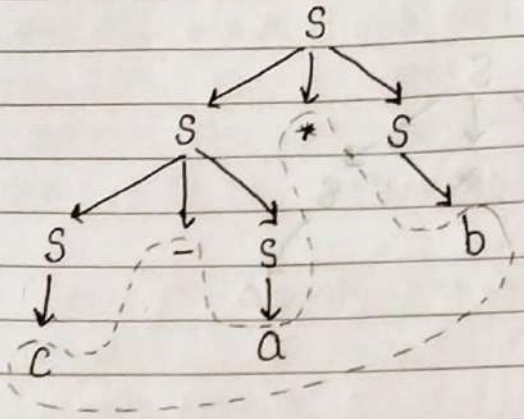
$c - a * b \leftarrow \boxed{S} - a * b \leftarrow$



Right most ~~tree~~ derivation Tree - 2

$S \longrightarrow S*S \longrightarrow S*b \longrightarrow S-S*b \longrightarrow S-a*b$

$c-a*b$



Since we are able to construct more than one right most derivation tree, hence the given grammar is ambiguous.

Ques: Show that the given Grammar is ambiguous grammar -

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

$S \rightarrow aSbS \mid bSaS \mid \epsilon$

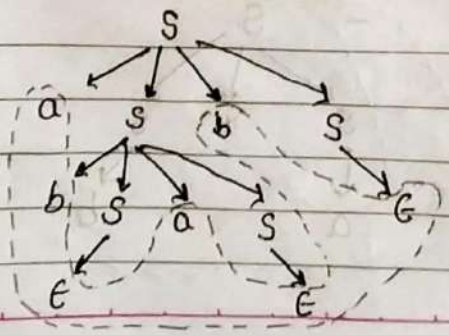
Soln. We can show ambiguity through string.

$\omega = abab$

Left most derivation Tree - 1

$S \rightarrow aSbS \longrightarrow abSaSbS \longrightarrow abaSbS \longrightarrow ababS$

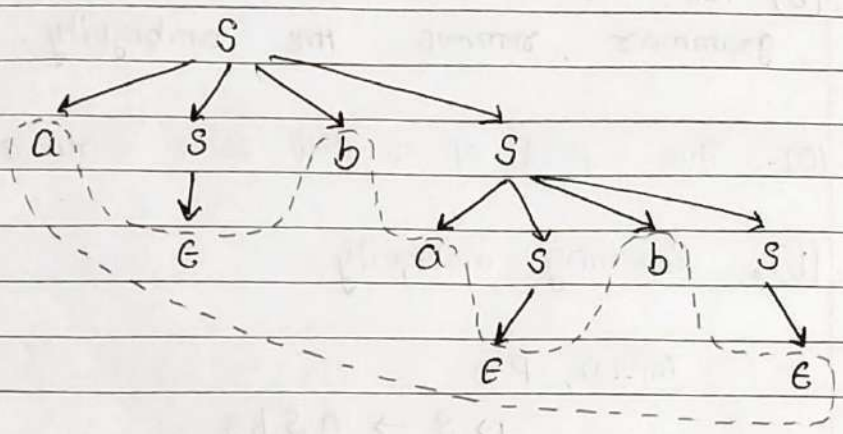
$abab$



Left most derivation Tree - 2

$S \rightarrow aSbS \rightarrow a\epsilon bS \rightleftharpoons abS \rightarrow abaSbS \rightarrow ababS$

$abab$



Since we are able to construct more than one left most derivation tree, hence the given grammar is ambiguous.

Ambigoity Removal in Context Free Grammar

Step 1: If the given grammar is ambiguous then keep replacing one of the same Non-terminal used in right hand side of production by some other non-terminal.

Step 2: The Non-Terminal which is replaced by the new Non-terminal is shown through the next production rule.

Step 3: Keep following step 1 & 2 till reaching to the terminals.

Ques: (a) For the given grammar show that the grammar is ambigous.

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

P: 1.) $S \rightarrow aSbS$

2.) $S \rightarrow bSaS$

3.) $S \rightarrow \epsilon$

(b) For the above question which is an ambiguous grammar, remove the ambiguity.

Soln (a). This part of question solve earlier.

(b), Removing ambiguity

Given, P:

1.) $S \rightarrow aSbS$

2.) $S \rightarrow bSaS$

3.) $S \rightarrow \epsilon$

Modified Rule,

P':

1.) $S \rightarrow aSbT$

2.) $S \rightarrow T$

3.) $T \rightarrow bTaU$

4.) $T \rightarrow U$

5.) $U \rightarrow \epsilon$

$G' = (\{S, T, U\}, \{a, b\}, P', S)$

Ques: a.) Show that the given grammar is ambiguous grammar, P: $S \rightarrow S+S \mid S-S \mid S*S \mid a \mid b \mid c$

(b) Remove ~~ab~~ ambiguity ~~fo~~rom the above given grammar.

Sol^n, a.) This part of solution already done earlier.

b.) Removing ambiguity,

Modified Rule,

P':

1.) $S \rightarrow S + T$

2.) $S \rightarrow T$

3.) $T \rightarrow T - U$

4.) $T \rightarrow U$

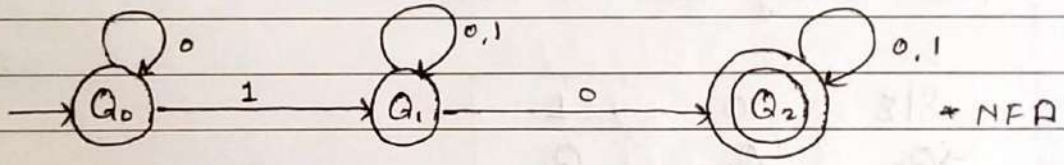
5.) $U \rightarrow U * V$

6.) $U \rightarrow V$

7.) $V \rightarrow a | b | c$

$G' = (\{S, T, U, V\}, \{a, b\}, P', S)$

Ques: Convert the given NFA into corresponding DFA using sub-set construction



Sol^n Transition Table

$\delta \mid \Sigma$	0	1
$\rightarrow Q_0$	Q_0	Q_1
Q_1	$\{Q_1, Q_2\}$	Q_1
Q_2	Q_2	$\{Q_1, Q_2\}$

Now finding all the transition from start state,

$\delta(Q_0, 0) \rightarrow Q_0$

$\delta(Q_0, 1) \rightarrow Q_1$

$\delta(Q_1, 0) \rightarrow \{Q_1, Q_2\}$ ⇒ new state

$\delta(Q_1, 1) \rightarrow Q_1$

$\delta(\{Q_1, Q_2\}, 0) = \delta(Q_1, 0) \cup \delta(Q_2, 0)$

$\Rightarrow \{Q_1, Q_2\} \cup \{Q_2\}$

$\Rightarrow \{Q_1, Q_2\}$

$\delta(\{Q_1, Q_2\}, 1) = \delta(Q_1, 1) \cup \delta(Q_2, 1)$

$\{Q_1\} \cup \{Q_1, Q_2\}$

$\{Q_1, Q_2\}$

Since now their is no new state, so the process will terminate

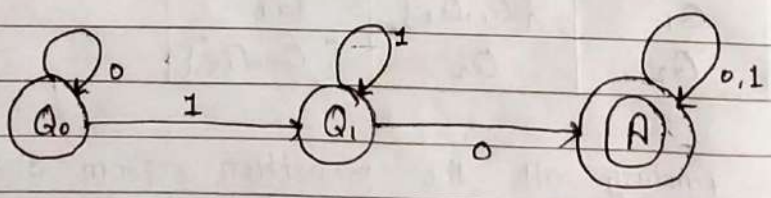
Say,

$\{Q_1, Q_2\} = A^*$

Now forming transition table

$\delta \mid \Sigma$	0	1
$\rightarrow Q_0$	Q_0	Q_1
Q_1	A	Q_1
A^*	A	A

Transition Diagram,



* DFA

Ques: P: $S \to asb \mid \epsilon$

Following Production rule given for any grammar G.

(i) What is the formal notation of language generated by given grammar

(ii) Mention whether the given grammar is regular grammar are not.

(iii) Mention whether the given grammar is context Free grammar are not.

Soln (i) Given, P: 1.) $S \to asb$

2.) $S \to \epsilon$

String generated are,

$S \xrightarrow{1} asb \xrightarrow{2} ab$

$S \xrightarrow{1} asb \xrightarrow{1} aasbb \xrightarrow{2} aabb$

$S \xrightarrow{1} asb \xrightarrow{1} aasbb \xrightarrow{1} aaasbbb \xrightarrow{2} aaabbb$

Let the language generate is L

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

(ii) No the above given grammar is not a regular grammar because for above grammar, required a secondary memory and Type-3 machine did not have any external memory.

(iii) It is simply a context Free grammar because in context Free grammar, on right hand side of production rule, any combination of terminal & Non-terminal occure.

Simplification of Context Free Grammar (CFG)

Any context free grammar can be simplified by using following,

1.) Removal of Useless Production
2.) Removal of Unit Production
3.) Removal of ϵ Production

Removal of Useless Production

In given context free grammar two type of Non-terminal are useless non-terminal.

(i) Those which never end to any string or get terminate.

(ii) Non-terminals which are not reachable from start symbol.

Ques. (i) Perform the removal of useless production for the given production role.

$P: S \rightarrow aB/a/b/\epsilon$

$B \rightarrow cD/e$

$A \rightarrow aA$

Sol^n, Removing useless production,

$S \rightarrow aB/a/b/\epsilon$

$S \rightarrow cD/\epsilon$

$A \rightarrow aA$

Hence the simplified context Free grammar is

$S \rightarrow aB/a/b/\epsilon$

$B \rightarrow \epsilon$

(2) $S \to aB|a|b|\epsilon$
$B \to CD$
$A \to a$

Soln,
$S \to a|b|\epsilon$

Removal of Unit Production

The productions of the form '$A \to B$' are known as unit production

Ques: Remove the unit Production from the production rule given below

(i) P: $S \to aA|B|a|b$
$B \to ab$
$A \to a$

Soln,
$S \to aA|B|a|b$ (B → ab)
$\therefore S \to aA|ab|a|b$
$A \to a$

(ii) P: $S \to aB|A|b|\epsilon$
$A \to a|D$
$D \to DE|a$
$E \to a|b|ab|F$
$F = \epsilon|a|b$
$B \to \epsilon$

Soln,
$S \to aB|b|\epsilon|a|ab$
$B \to \epsilon$

Removal of ϵ Production

Those production which are of the form '$A \to \epsilon$' are known as ϵ production.
We shall use the ϵ-production rule in all the sub-sequent Non-terminal production rule.

Ques: P: $S \to aA|aB|b|a$
$A \to \epsilon$
$B \to b$

Soln,
$S \to aA|aB|b|a$
$B \to b$

(2) P: $S \to aA \mid aB \mid b \mid a$
$A \to \epsilon \mid a \mid b$
$B \to b$

Sol^n,
$S \to aA \mid aB \mid b \mid a$
$A \to a \mid b$
$B \to b$

(3) P: $S \to aA \mid abbA \mid aAb \mid bbA$
$A \to \epsilon$

Sol^n, $S \to aA \mid abbA \mid aAb \mid bbA$
$S \to a \mid abb \mid ab \mid bb$

Ques: Apply the simplification rule for the given production rule for the context Free Grammar
P: $S \to aA \mid B \mid a \mid b \mid aSb \mid \epsilon \mid D$
$B \to aaa \mid aa$

Sol^n, First of all removing useless production, so we will have simplified grammar production rule as mention below,

$S \to B \mid a \mid b \mid aSb \mid \epsilon$
$B \to aaa \mid aa$

Now, Removing unit production, hence the simplified grammar production rule will be as mention below,

$S \to a \mid b \mid aSb \mid \epsilon \mid aaa \mid aa$

Now Removing ϵ-production, hence we will have the simplified production rule as shown below

$S \to a \mid b \mid ab \mid aaa \mid aa \mid aSb$

This is final simplified context free grammar.

Ques: Let $S = \{ab, ba\}$ and Let $T = \{ab, bb, bbbb\}$. Show that $S^* \neq T^*$

Soln: Some string generated by S^*,

$S^* = \{ab, ba, abab, baba, abba, baab, ----\}$

Now the string generate by T^*

$T^* = \{ab, bb, bbbb, abab, abbbbb, abbbbbb---\}$

We can clearly see that the both string generated is not equal to each others Hence it is proved that $S^* \neq T^*$ for the given production rule.

Ques: Write the regular expression for the language containing string over $\{a, b\}$ in which it must contain one 'a' at the begining and two 'b' at the last.

Soln: $R = \{a.(a+b)^* bb\}$

Ques: Apply the simplification of CFG to the given Production Rule

(i) P: $S \rightarrow 0A \mid 1B \mid C$
$A \rightarrow 0S \mid 00$
$B \rightarrow 1 \mid A$
$C \rightarrow 01$

(ii) P: $S \rightarrow XYX$
$X \rightarrow 0X \mid \epsilon$
$Y \rightarrow 1Y \mid \epsilon$

Normal Forms

In context Free grammar we know that their is no restriction on right hand side of production rule, however we can convert the given CFG into some mixed notation on right hand side production rule and this is known as Normalization or conversion of CFG into some other standard form.

V. Imp

Remember that while doing this, the meaning of given CFG will not change and will remain the same.

=> There are ~~me~~ two main normal form,

1.) Chomsky Normal Form (CNF)

2.) Greibach Normal Form (GNF)

Chomsky Normal Form (CNF)

This allows following production,

1.) $S \rightarrow \epsilon$ where S is start symbol

2.) $A \rightarrow BC$

3.) $B \rightarrow a$

4.) $C \rightarrow b$

NoTe

- While converting the given CFG to CNF first of all apply simplification of CFG (Removal of useless, unit and ϵ-production)
- After that we can use new non-terminal and can act accordingly
- There can be more than one possible CNF for the given grammar

Ques: For the given ~~CFG~~ CFG, convert this into CNF

CFG: 1.) $S \to \epsilon$

2.) $S \to ABC$

3.) $C \to a$

4.) $B \to bE$

5.) $E \to a$

6.) $A \to a$

Soln, Since the given grammar does not require any simplification, hence we will know apply CNF conversion,

1.) $S \to \epsilon$

2.) $S \to FC$

3) $F \to AB$

4) $C \to a$

5) $B \to GE$

6) $G \to b$

7) $E \to a$

8) $A \to a$

Equivalent Grammar

If two grammar G & G' represent the same language then both are equivalent grammar that is,

$$L(G) = L(G')$$

⇒ If grammar represent by CFG is denoted as G & the converted grammar into CNF or GNF is G' then again,

$$L(G) = L(G')$$

Ques Convert the given CFG into CNF.

CFG => P : 1) $S \to \epsilon \mid aG$

2) $S \to ABCD$

3) $A \to a$

4) $B \to b$

5) $C \to aE$

6) $E \to b$

7) $D \to F$

8) $F \to a$

Sol Applying simplification,

Removing useless production, so we get

1) $S \to \epsilon$

2) $S \to ABCD$

3) $A \to a$

4) $B \to b$

5) $C \to aE$

6) $E \to b$

7) $D \to F$

8) $F \to a$

Now removing unit production, so we get

1) $S \to \epsilon$

2) $S \to ABCD$

3) $A \to a$

4) $B \to b$

5) $C \to aE$

6) $E \to b$

7) $D \to a$

Now the given grammar is simplified, hence we will apply CNF conversion,

$G' \rightarrow$ 1) $S \rightarrow \epsilon$
2) $S \rightarrow AH$
3) $H \rightarrow BI$
4) $I \rightarrow CD$
5) $A \rightarrow a$
6) $B \rightarrow b$
7) $C \rightarrow JE$
8) $J \rightarrow a$
9) $E \rightarrow b$
10) $D \rightarrow a$

Greibach Normal Form (GNF)

This allow following production,

1) $S \rightarrow \epsilon$
2) $A \rightarrow a$
3) $B \rightarrow aN^*$

Ques: Apply the GNF conversion for the given CFG,

P: $S \rightarrow AbA$
$S \rightarrow \epsilon$
$S \rightarrow C$
$C \rightarrow a$
$C \rightarrow abD$

Sol^n, Applying simplification, removing useless production,

1) $S \rightarrow \epsilon$
2) $S \rightarrow C$
3) $C \rightarrow a$

Now, removing unit production,

1) $S \rightarrow \epsilon$
2) $S \rightarrow a$

Now after simplification, applying CNF conversion

1) $S \rightarrow \epsilon$
2) $S \rightarrow a$

Ques Convert given CFG to GNF

P: 1) $S \rightarrow Aaab$
2) $S \rightarrow \epsilon$
3) $S \rightarrow c$
4) $\epsilon \rightarrow a$
5) $A \rightarrow a$

Sol^n, Removing unit production,

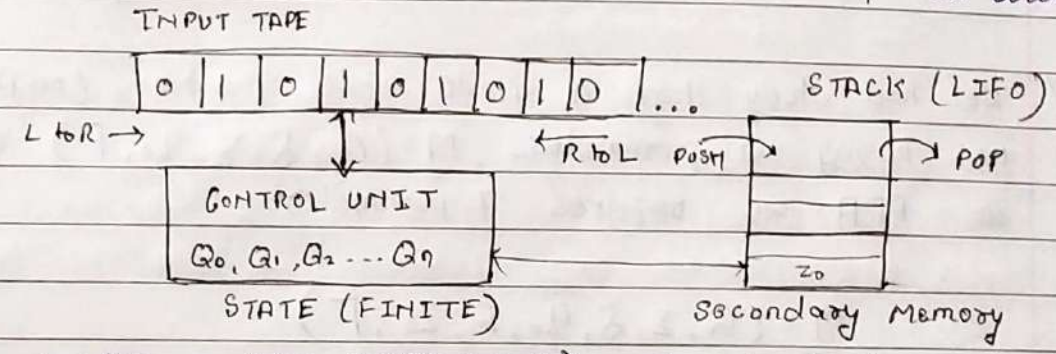
1) $S \rightarrow Aaab$
2) $S \rightarrow \epsilon$
3) $S \rightarrow a$
4) $A \rightarrow a$

Now applying GNF conversion,

1) $S \rightarrow aPQA$
2) $P \rightarrow a$
3) $Q \rightarrow b$
4) $S \rightarrow \epsilon$
5) $S \rightarrow a$
6) $A \rightarrow a$

Push Down Automata (PDA)

- Push Down Automata, PDA is a finite automata having secondary storage or extra storage.
- The secondary storage or extra storage is in the form of STACK (LIFO Property)
- The PDA is for Context Free Language processing.
- A general PDA representation can be shown as below,



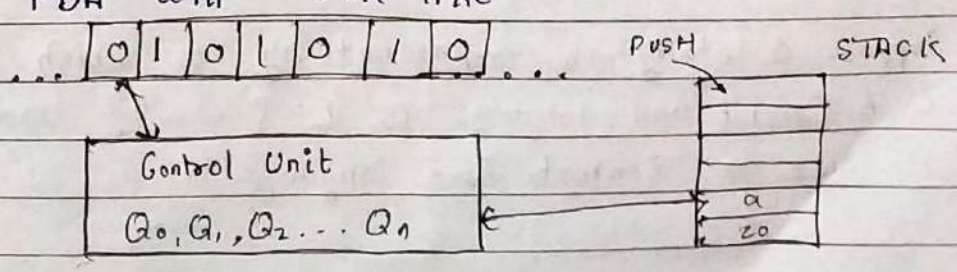
⇒ In regular machine (NFA & DFA) only input tape and control unit state was used for defining the transition (δ). However here in PDA transition will involve 3 ~~three~~ parameters. The 3rd extra parameter will be the symbol on to the top of the stack.

⇒ Hence we can understand the transition state (δ) for above PDA representation as mention below,

$$\delta(Q_0, 0, Z_0) \Rightarrow (Q_1, aZ_0)$$
$$\delta(Q_0, 0, Z_0) \Rightarrow (Q_1, \epsilon)$$

⇒ We very much know that 2 operation PUSH & POP are useful when defining the ~~state~~ Stack.

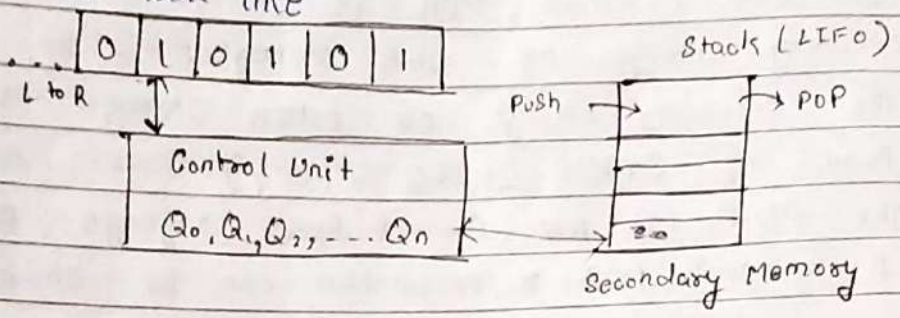
⇒ So, for PUSH operation $\delta(Q_0, 0, Z_0) \Rightarrow (Q_1, aZ_0)$ and PDA will look like



For PoP operation, the transition (δ) can be written as

$$\delta(Q_0, 0, Z_0) = (Q_1, G)$$

& Stack will look like



Imp// We had study that a finite state machine (NFA & DFA) was having 5 parameters, $M = (Q, \delta, \Sigma, q_0, F)$ while in PDA we requires 7 parameter,

$$M = (Q, \Sigma, \delta, q_0, F, Z_0, \underset{\text{TAU}}{\Gamma})$$

where Z_0 is default or initial input symbol of stack and TAU (Γ) is the symbol allowed on stack.

=> We know that certain languages such as,

(1) $L = a^n b^n ; n \geq 0$ can not be expressed or process through finite state machine (DFA & NFA) and can ~~not~~ be expressed through push Down Automata and the language are Context Free Language.

2-> $L = \{a^n b^{2n} : n \geq 0\}$ } are Context Free language

3-> $L = \{0^n 1^n : n \geq 0\}$ }

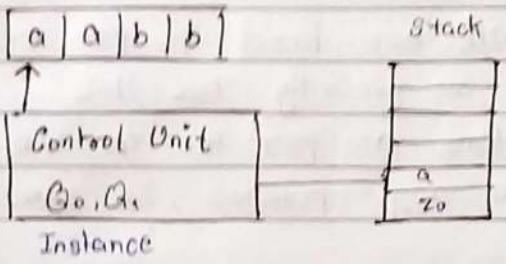
Imp// 4-> Suppose a string is represent by 'w' which belong to $w \in \{0, 1\}$ and language is $L = \{w w^R\}$ and $\Sigma = \{0, 1\}$ is also a Context Free language

5) Palindrom numbers of any length are also a example of context free language.

#) Note: we know that in stack we can have 3 possible operation - PUSH, POP & SKIP or (no operation), let understand one by one,

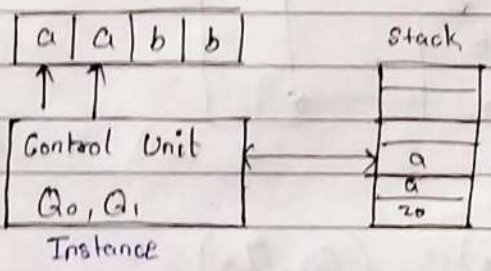
#1 PUSH

(i)



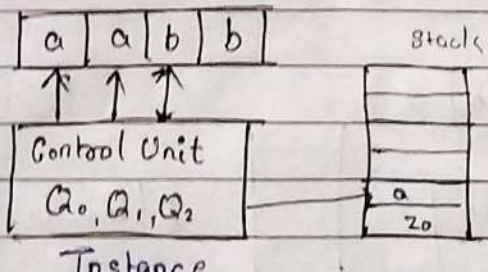
$\delta(Q_0, a, Z_0) = (Q_1, aZ_0)$

(ii) $\delta(Q_0, a, a) = (Q_1, aa)$



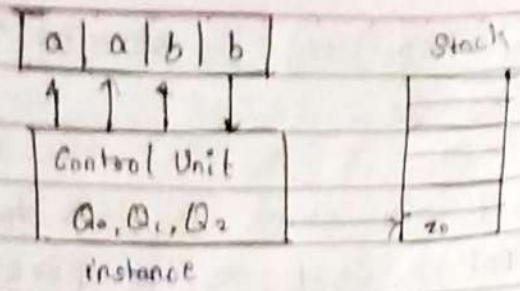
#2 POP

(iii)



$\delta(Q_0, b, a) = (Q_2, \epsilon)$

(iv)

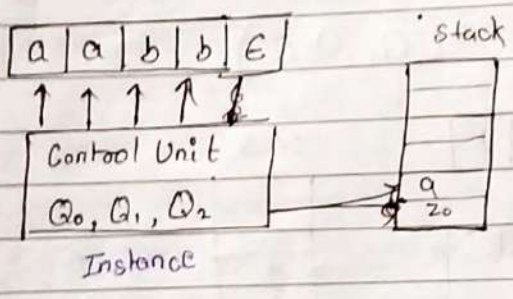


$$\delta(Q_2, b, a) = (Q_2, \epsilon)$$

#3 SKIP Operation

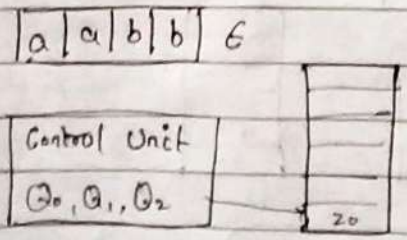
- This is based on our need, as per our convinience for state change we generally use this
- And also some time as per the question when we don't want perform any operation, we can use this.

⇒ For the following representation, we can understand skip operation as mentioned below,



$$\delta(Q_2, \epsilon, Q_0) = (Q_2, Q_0)$$

⇒ Suppose we have a following rep^n of the PDA then we will have skip operation written as bellow



$$\delta(Q_2, \epsilon, Z_0) = (Q_2, Z_0)$$

⇒ In general the transition delta (δ) for PDA machine will be expressed as general notation is shown below,

$$[Q \times \Sigma \cup \{\epsilon\} \times \Gamma \Rightarrow Q \times \Gamma^*]$$

Acceptance of String by PDA

Acceptance by Final State (say Q_f or q_f)
Acceptance by Empty Stack

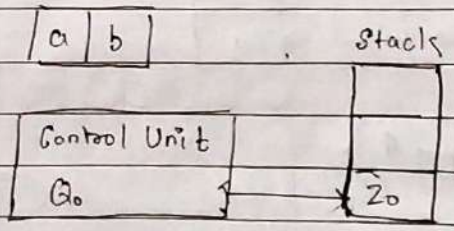
⇒ The first approach we had already studied in the regular grammar, however the other approach is acceptance by empty stack.

NoTe

1. Remember almost all the step are same except the last state by accepting through final state or accepting through empty stack.

2. In both the approaches, the input tape must be empty atleast that is no symbol is remaining.

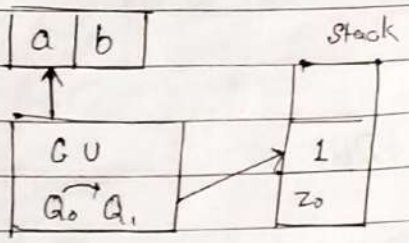
Ques Describe the acceptance by final state for the string $\omega = ab$ through PDA.



(Fig 1)

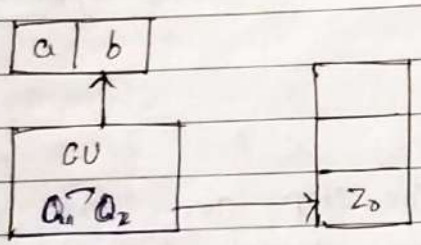
Let $\Gamma = \{Z_0, 1, 0\}$

(ii) $\delta(Q_0, a, Z_0) = \{Q_1, 1Z_0\}$



(Fig 2)

(iii) $\delta(Q_1, b, 1) = (Q_1, \epsilon)$



(Fig 3)

(iii) $\delta(Q_2, \epsilon, Z_0) = (Q_f, Z_0)$

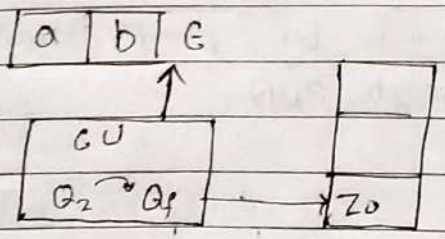
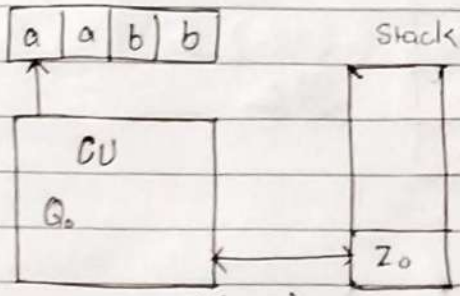


Fig (4)

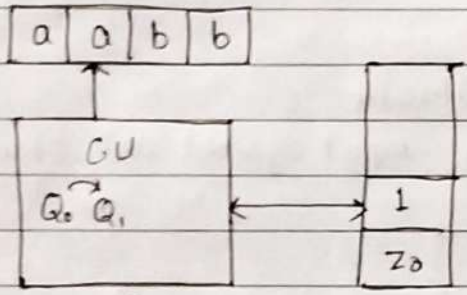
Ques Show the acceptance of string, $\omega = aabb$ using final state in PDA

Sol^n,



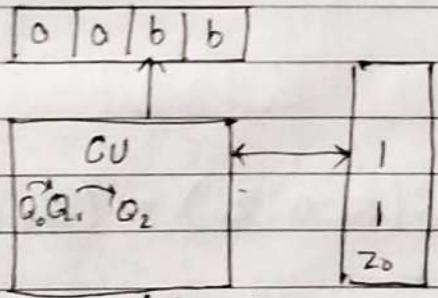
(Fig. 1)

$\delta(Q_0, a, Z_0) = (Q_1, 1Z_0)$



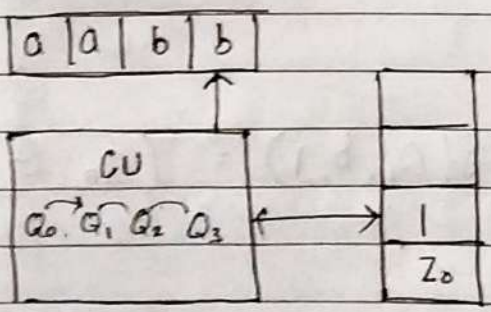
$\delta(Q_1, a, 1) = (Q_2, 11)$

(Fig. 2)



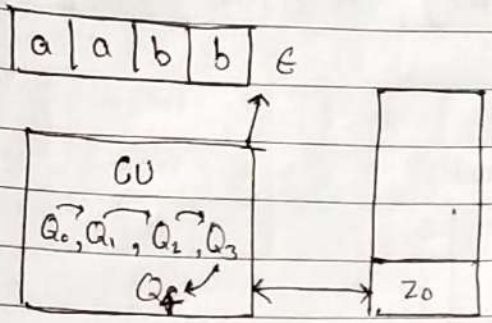
$\delta(Q_2, b, 1) = (Q_3, \epsilon)$

(Fig. 3)



$\delta(Q_3, b, 1) = (Q_f, \epsilon)$

(Fig. 4)



(Fig. 5)

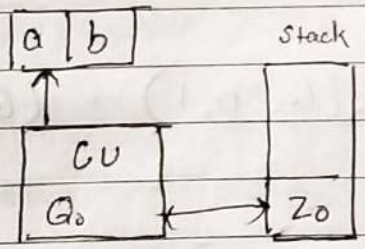
Here we have reach to the final state and input tape is also empty.

Acceptance by Empty stack

In this approach both input symbol and stack must become empty.

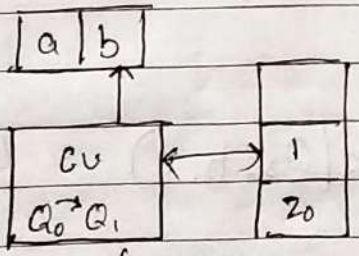
Ques: Show the acceptance by empty stack for string, $w = ab$ using PDA

Soln



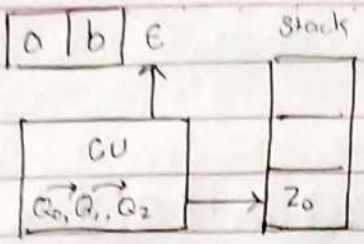
$\delta(Q_0, a, Z_0) = (Q_1, 1Z_0)$

(Fig. 1)



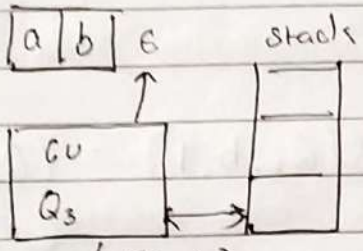
$\delta(Q_1, b, 1) = (Q_2, \in)$

(Fig. 2)



$\delta(Q_2, \epsilon, Z_0) = (Q_3, \epsilon)$

(Fig.3)

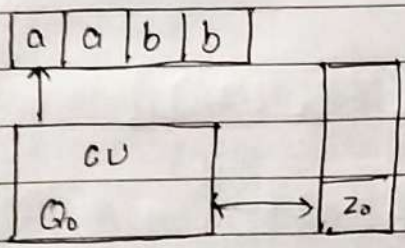


(Fig.4)

Here the input symbol and stack both are empty.

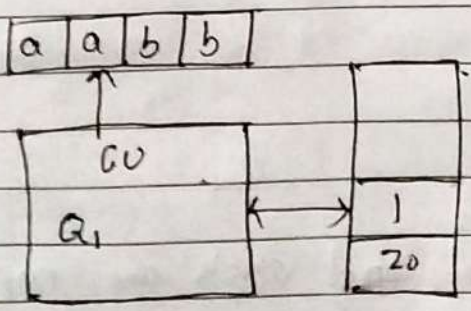
Ques Accept using empty stacks for the string, $w = aabb$ using PDA

Soln



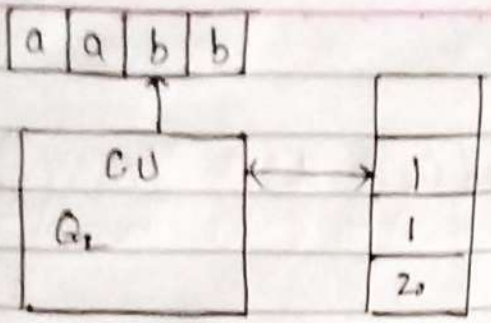
$\delta(Q_0, a, Z_0) = (Q_1, Z_0 1)$

(Fig.1)



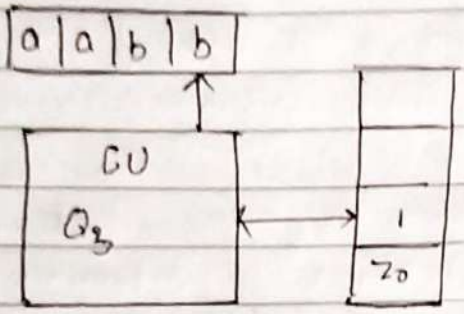
$\delta(Q_1, a, 1) = (Q_2, 1, 1)$

(Fig.2)



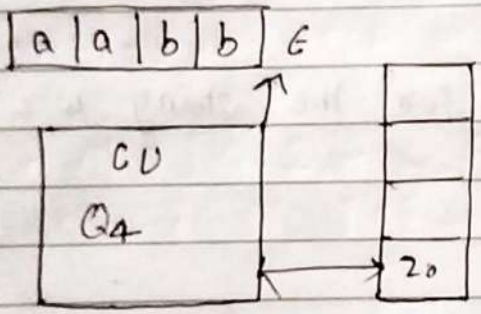
$\delta(Q_2, b, 1) = (Q_3, \epsilon)$

(Fig. 3)



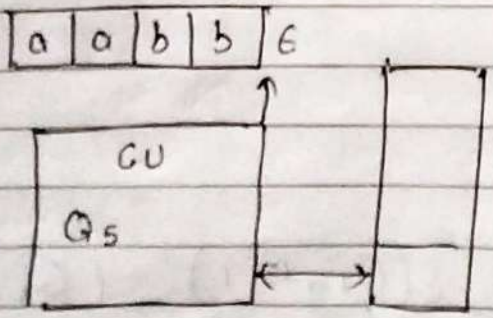
$\delta(Q_3, b, 1) = (Q_4, \epsilon)$

(Fig. 4)



$\delta(Q_4, \epsilon, Z_0) = (Q_5, \epsilon)$

(Fig. 5)



Here both the input symbol and stack are empty.

Camlin Page
Date / /

Ques: Show the acceptance using PDA for the language,
$L = \{a^n b^n : n \geq 1\}$
Write down the state representation and also mention Instantaneous Description (ID) for any assumed string following the given language.

Instantaneous Description (ID)

- We know that we generally process these Context Free language with needs large amount of secondary storage or memory
- In order to process them we require instantaneous description for any small size finite string which is following that limit.
- To show instantaneous description, we use the symbol "$\vdash$" between two consecutive transition.
- ID, again describe for acceptance by Final state and acceptance by empty stack.

Ques: For string, $\omega = aabb$, following the language
$L = \{a^n b^n : n \geq 1\}$
Shows the instantaneous description for acceptance using Final State & Empty stack.

Sol^n: For the final state Machine,

$$(Q_0, aabb, Z_0) \vdash (Q_1, abb, 1Z_0) \vdash (Q_2, bb, 11Z_0) \vdash$$

$$(Q_3, b, 1Z_0) \vdash (Q_f, \epsilon, Z_0)$$

Now doing the same, for empty stack,

$$(Q_0, aabb, Z_0) \vdash (Q_1, abb, 1Z_0) \vdash (Q_2, bb, 11Z_0)$$
$$\vdash (Q_3, b, 1Z_0) \vdash (Q_4, \epsilon, Z_0) \vdash (Q_5, \epsilon, \epsilon)$$

Ques What do you mean by Deterministic PDA and Non-deterministic PDA?

Sol" We had already studied the basic meaning of deterministic & Non-deterministic.

- In non-deterministic we can have more than one or no transition for any input symbol.

Example:- Two PDA machine M & M' are describe below,

For $M = \{Q, \Sigma, \delta, q_0, F, Z_0, \Gamma\}$

$Z_0 = \{0\}$, $\Gamma = \{Z_0, 1\}$

$$\delta(Q_0, a, Z_0) = \{(q_1, \epsilon), (q_2, 1Z_0)\}$$

$$\delta(Q_0, b, 1) = \{Q_2, \epsilon\}$$

⇒ Here in M, few transition are left out and few are used more than one. Hence M is non-deterministic PDA.

Example (i) $L = \{w w^R : \Sigma = \{0, 1\}\}$

(ii) $L = \{wcw^R : \Sigma = \{0, 1\}\}$

Sol^n, In first one we don't know when w is going to end and when w^R is going to start.

Assuming $w = 100$, So $w^R = 001$

Hence $ww^R = 100001$

$\Rightarrow$ So after process 1st one & zero for the next symbol we are not aware whether this is part of w or w^R.

$\Rightarrow$ But, in example 2 any thing before C will be the part of w and after C will be the part of w^R.

Ques: Complete PDA for $L = \{a^n b^n : n \geq 1\}$

Sol^n, Let PDA is represented by M

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

Let $Z_0 = 0$, $\Gamma = \{z_0, 0, a\}$

Transition

PUSH $\begin{cases} \delta(Q_1, a, Z_0) = (Q_1, aZ_0) \\ \delta(Q_1, a, a) = (Q_1, aa) \end{cases}$ Assume Q_1 start as start state

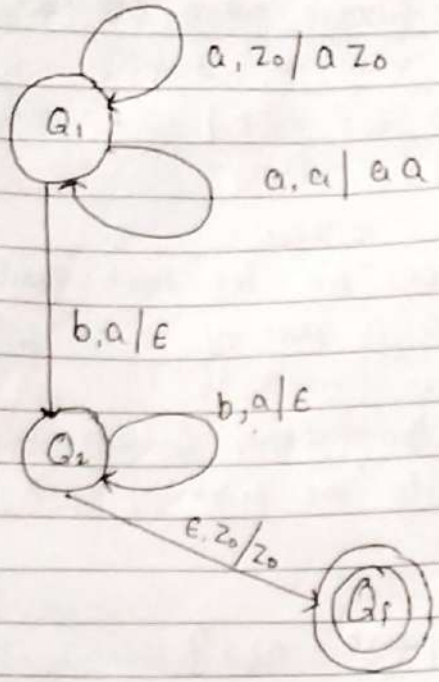
For PUSH, assume Q_1 state

POP $\begin{cases} \delta(Q_1, b, a) = (Q_2, \epsilon) \\ \delta(Q_2, b, a) = (Q_2, \epsilon) \end{cases}$ For POP, assume Q_2 state

Final state accept. $\delta(Q_2, \epsilon, Z_0) = (Q_F, Z_0)$

Empty stack acceptance $\delta(Q_2, \epsilon, Z_0) = (Q_3, \epsilon)$ Q_f is final state

Q_3 is state on empty stack transition



* PDA representation (acceptance using final state)

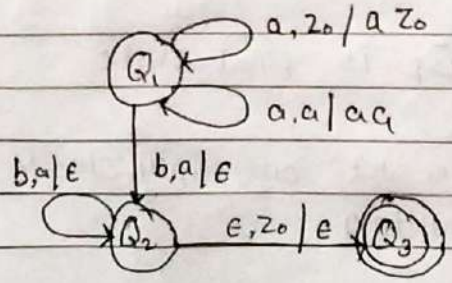
Let $w = aabb \in L$

ID

$$(Q_1, aabb, Z_0) \vdash (Q_1, abb, aZ_0) \vdash (Q_1, bb, aaZ_0)$$

$$\vdash (Q_2, b, aZ_0) \vdash (Q_2, \epsilon, Z_0) \vdash (Q_f, \epsilon, Z_0)$$

$\Rightarrow$ The same above question can also be solve using acceptance by empty stack.



* PDA representation (acceptance using empty stack)

ID for $\omega = aabb \in L$

$$(Q_1, aabb, Z_0) \vdash (Q_1, abb, aZ_0) \vdash (Q_1, bb, aaZ_0)$$

$$\vdash (Q_2, b, aZ_0) \vdash (Q_2, \epsilon, Z_0) \vdash (Q_3\ \epsilon, \epsilon)$$

Ques: Design a PDA for $L = \{a^n b^n : n \geq 0\}$

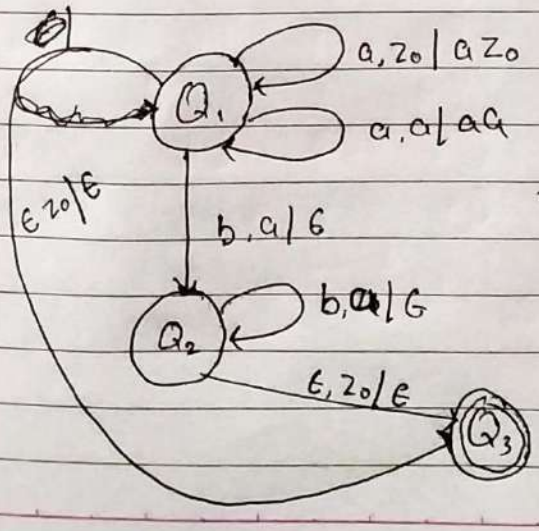
Sol: For this question since empty string (ϵ) can also come hence we should show either of the transaction along with all the previous transition & their two representation are mentioned below

(i) $\delta(Q_1, \epsilon, Z_0) = (Q_3, \epsilon)$

(ii) $\delta(Q_1, \epsilon, Z_0) = (Q_f, Z_0)$

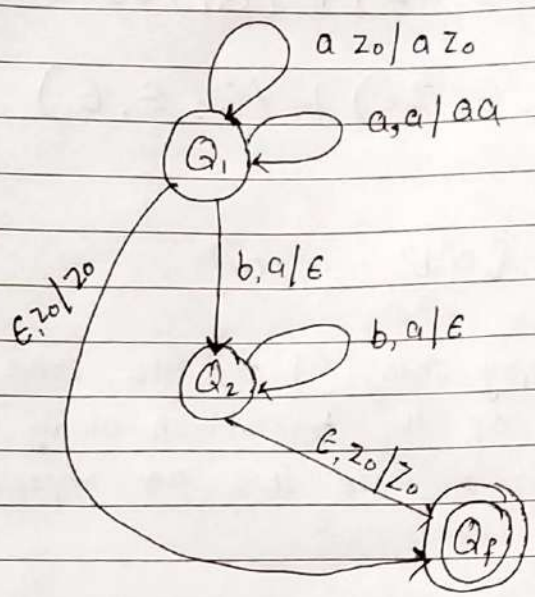
⇒ Here first one is for empty stack acceptance and 2nd one is for final state acceptance.

⇒ Also the PDA state representation will get change and correspondence PDA is show below,



* PDA representation (by empty stack)

Now for the acceptance by final state, the PDA will be like as mention below



* PDA Representation
(By Final State Acceptance)