classmate
Date 21/06/23
Page 01

Introduction to theory of computation.

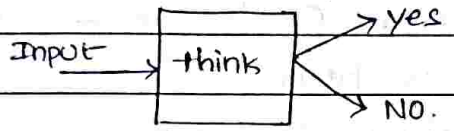
Introduction :-

⇒ One of the most fundamental courses of computer science.

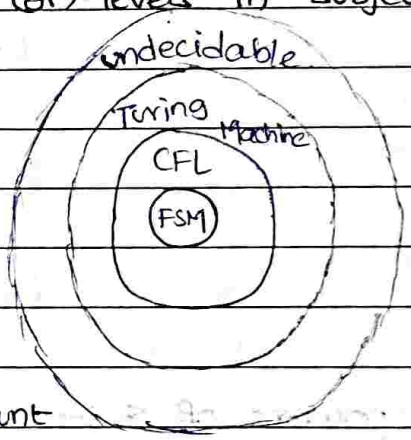
⇒ Will help you understand how people have thought about computer science as a science in past 50 years.

⇒ It is mainly about what kind of things can you really compute mechanically, how fast and how much space does it take to do so.

What we usually do :-



layers (or) levels in subject :-



FSM – Finite state machine. :-

→ it is the simplest model of computation

→ It has a very very limited amount of memory

→ It performs very low-level computations and calculations

CFL – context Free Language :-

→ It is little more powerful than FSM

→ It can perform some more higher level computations as compared to FSM

→ In this the word "language" means set of strings.

Turing machine :-

→ It can perform higher level computations & calculations.

→ It was designed by Alan Turning in 1940.

→ It is powerful than CFL & FSM.

undecidable :-

→ The problems that cannot be solved mechanically come under this

Finite state Machine (FSM)

prerequisites :-

Symbol — Anything like Alphabets, Numbers

Ex :- A, B, C, D (or) 1, 2, 0, - -

Alphabet — It is denoted by sigma (Σ)

It is a collection of Symbol's in curly Braces

Ex :- $\Sigma = \{a, b\}, \{d, e, f, g\}$

String — The sequence of Symbols

Ex :- a, b, 0, 1, aa, bb, ab, 01, - - -

Language — A set of strings

Ex :- $\Sigma = \{0, 1\}$

L_1 = set of all strings of length 2.

$= \{00, 01, 10, 11\}$ — finite sets

L_2 = set of all strings of length 3.

$= \{000, 001, 010, 100, 101, 110, 011, 111\}$ — finite set

L_3 = set of all strings ~~of~~ begin with 0.

$= \{0, 01, 001, 010, 011, 000, - - -\}$ — Infinite set

powers of Σ :-

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

Σ^0 = set of all strings of length 0. :- $\Sigma^0 = \{\epsilon\}$

ϵ = zero.

Σ^1 = set of all strings of length 1 :- $\Sigma^1 = \{0, 1\}$

Σ^2 = set of all strings of length 2 :- $\Sigma^2 = \{00, 01, 10, 11\}$

Σ^3 = set of all strings of length 3 :- $\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

⋮

Σ^n = set of all strings of length n :-

Cardinality :- Number of elements in a set

→ cardinality of $\Sigma^n = 2^n$ if ~~con~~ $\Sigma = \{0, 1\}$.

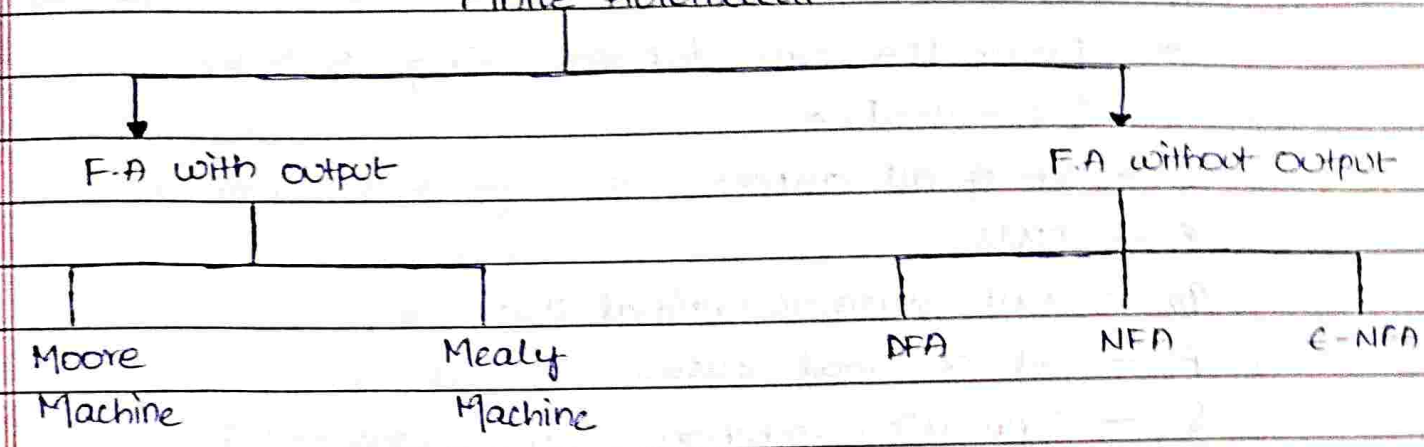
Σ^* :- $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 - - - -$

$= \{\epsilon\} \cup \{0, 1\} \cup \{00, 01, 10, 11\} \cup - - - -$

→ set of all possible strings of all lengths over $\{0, 1\}$

classmate
Date 21/06/23
Page 03

Finite state Machine (FSM).



→ Finite state Machine which is also known as Finite Automata

→ Finite Automata classified into two
  i) Finite Automata with output
  ii) Finite Automata without output.

→ Finite Automata with output classified into two.
  i) Moore Machine ii) Mealy Machine.

→ Finite Automata without output classified into two three.
  i) DFA (Deterministic Finite Automata)
  ii) NFA (Non-Deterministic Finite Automata).
  iii) NFA-ε (Non-Deterministic Finite Automata with Epsilon)

Date 21/02/23
Page 04

Deterministic Finite Automata (DFA) :-

DFA :-

→ Every DFA can defined using 5 tuples.

→ (Q, Σ, q_0, F, δ)

Q – set of all states.

Σ – inputs

q_0 – start state (or) initial state.

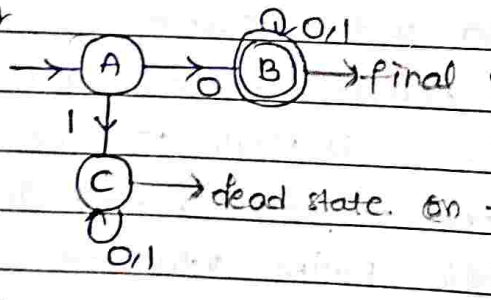
F – set of final state.

δ – Transition function from $Q \times \Sigma \longrightarrow Q$.

Examples :- 1

L1 = set of all strings that start with '0'.

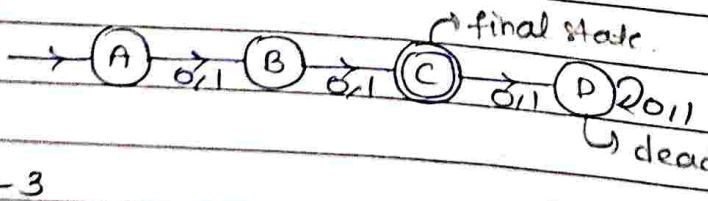
A) L1 = {0, 01, 00, 001, 010, 011, ---}



Example - 2

construct a DFA that accepts set of all strings over {0,1} of length 2.

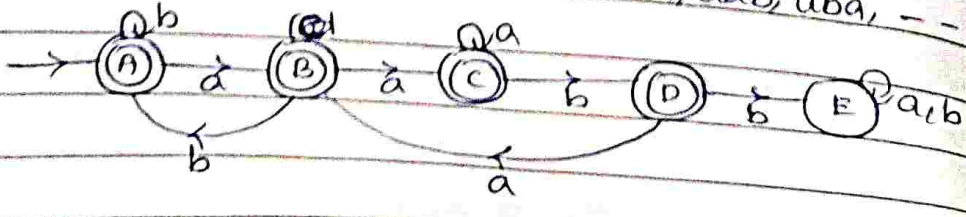
A) Σ = {0,1}, L = {00, 01, 10, 11}



Example - 3

construct a DFA that accepts any strings over {a, b} that does not contain the string aabb in it.

A) Σ = {0,1}, L = {a, b, ab, ba, aaa, aab, aba, ---}



Regular Languages

Regular Languages:

→ A language is said to be regular language if and only if some finite state Machine recognizes it.

So what languages are NOT REGULAR?

The languages

→ which are not recognized by any FSM

→ It which require memory

We know.

— Memory of FSM is very limited

— It cannot store or count strings.

Operations on Regular Languages :-

Union — $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$

Concatenation — $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$

STAR — $A^* = \{x_1 x_2 x_3 \dots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$

Example $A = \{pq, r\}$, $B = \{t, uv\}$

$A \cup B = \{pq, r, t, uv\}$

$A \circ B = \{pqt, pquv, rt, ruv\}$

$A^* = \{\epsilon, pq, r, pqr, rpq, pqpq, rr, pqpqpq, rrr, \dots\}$

Theorem 1:-

The class of Regular languages is closed under UNION

Theorem 2:-

The class of Regular Languages is closed under concatination.

Non-deterministic Finite Automata (NFA):-

Deterministic Finite Automata

↓

Determinism

→ In DFA, given the current state, We know what the next state will be.

→ It has Only One Unique next state.

→ It has no choices or randomness.

→ It is simple and easy to design.

Non-deterministic Finite Automata.

↓

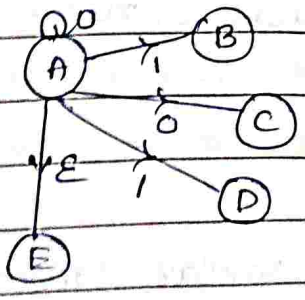
Non-determinism

→ In NFA, given the current state there could be multiple next states.

→ The next state may be chosen at random.

→ All the next states may be choosen in parallel

Example -for above :-



NFA – Formal Definition.

NFA :-

→ Every DFA can defined using 5 tuples

→ $(Q, \Sigma, q_0, F, \delta)$

Q — set of all states

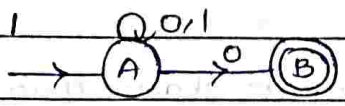
Σ — input

q_0 — initial state

F — Final state.

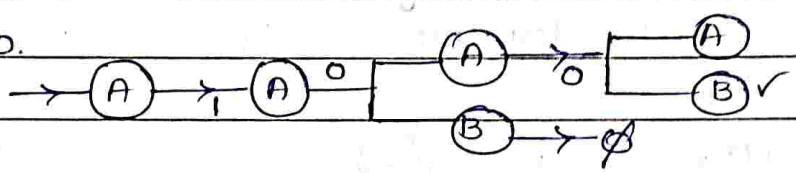
δ — $Q \times \Sigma \rightarrow 2^Q$

Example :- 1



L = {set of all strings that end with 0}

if 100.

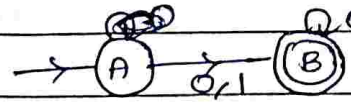


Note :- If there is any way to run the machine that ends in any set of states out of which atleast one state is a final state, then the NFA accepts.

Example :- 2

L = {set of all strings that starts with 0}

A) $L = \{0, 00, 01, 000, ---\}$

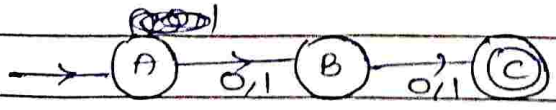


(we don't need to create dead state)

construct a NFA that accepts sets of all strings over {0,1} of length 2.

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

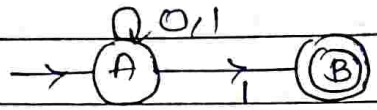
$L = \{00, 01, 10, 11\}$



Example - 3

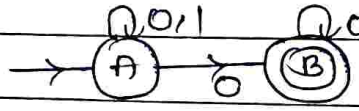
Q) L_1 = {set of all strings that ends with '1'}

A) L_1 = {1, 01, 001, ---}



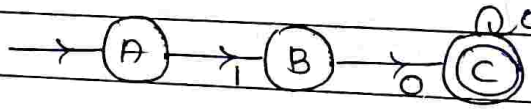
Q) L_2 = {set of all strings that contain '0'}

A) L_2 = {0, 00, 01, 10, ---}



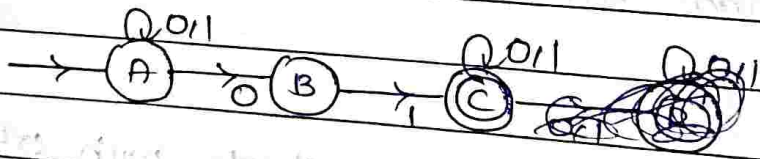
Q) L_3 = {set of all strings that start with '10'}

L_3 = {10, 100, 101, 1011, --}



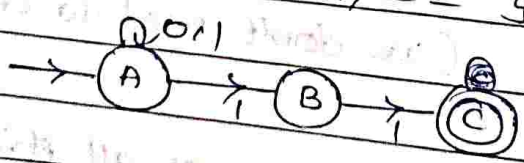
Q) L_4 = {set of all string contain '01'}

L_4 = {01, 001, 101, ---}



Q) L_5 = {set of all strings ends with '11'}

L_5 = {11, 011, 111, --}



Conversion of NFA to DFA

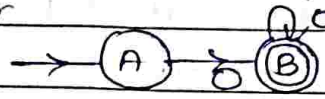
→ Every DFA is an NFA, but not vice versa. But there is an equivalent DFA for every NFA.

process:

L = { set of all strings over (0,1) that starts with '0' }

A) Σ = {0,1}

NFA:



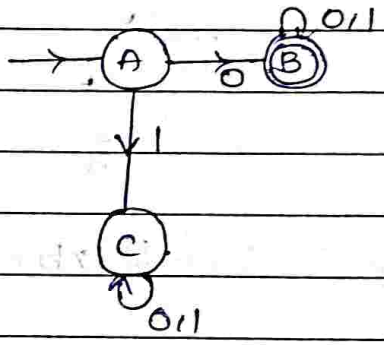
Transition table: NFA

	0	1
A	B	∅
B	B	B

Transition table: DFA.

δ	0	1
A	B	∅ (or) C
B	B	B
C(∅)	C(∅)	C(∅)

DFA:

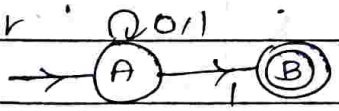


Example 1

L = { set of all strings over (0,1) that ends with '1' }

A) Σ = {0,1}

NFA:



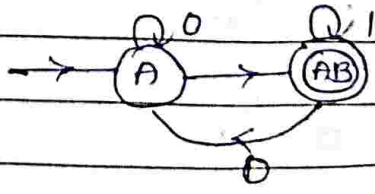
Transition table: NFA

	0	1
A	A	A, B
B	∅	∅

Transition table: DFA

δ	0	1
A	A	AB
AB	A	AB

DFA:



Note: The method of conversion this is sub set construction method.

Example 2-

Find the equivalent DFA for the NFA given by $M = [\{A, B, C\}, (a, b), \delta, A, \{C\}]$ where δ is given by:

A)

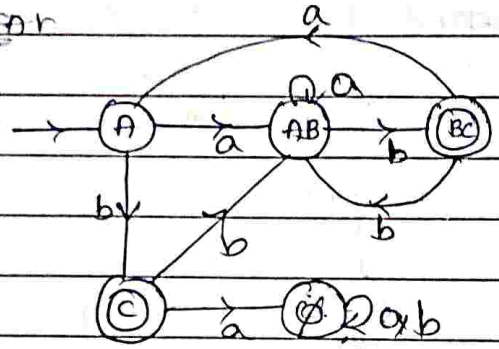
Transition table of NFA

δ	a	b
→ A	A, B	C
B	A	B
(C)	—	A, B

Transition table of DFA

δ	a	b
→ A	AB	C
AB	AB	BC
BC	A	AB
C	$\emptyset$	AB
$\emptyset$	$\emptyset$	$\emptyset$

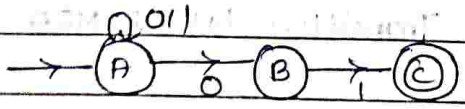
DFA:



Example 3

L = {set of all strings over (0, 1) that ends with '01'}

A) NFA:



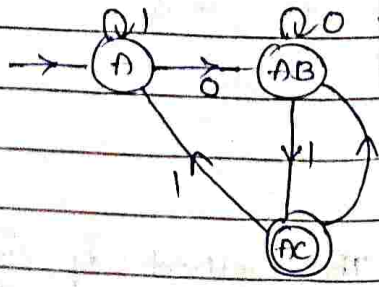
Transition table of NFA

δ	0	1
A	A, B	A
B	$\emptyset$	BC
C	$\emptyset$	$\emptyset$

Transition table of DFA:

δ	0	1
A	AB	A
AB	AB	AC
(AC)	AB	A
$\emptyset$	$\emptyset$	$\emptyset$

DFA:

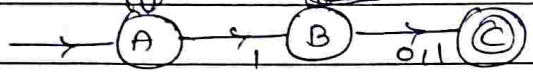


Example 4

Design an NFA for language that accepts all strings over {0,1} in which the second last symbol is always '1'. Then convert it to its equivalent DFA

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

NFA:



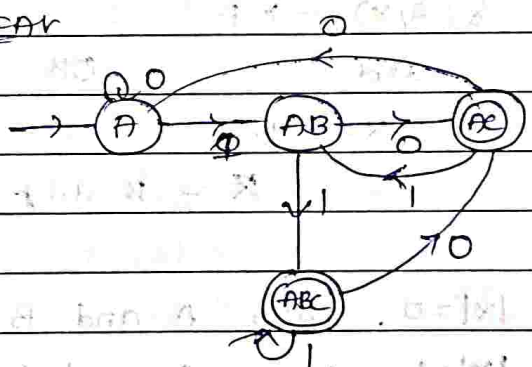
Transition table of NFA

δ	0	1
A	AB	AB
B	C	C
C	∅	∅

Transition table of DFA

δ	0	1
A	A	AB
AB	AC	ABC
AC	A	AB
ABC	AC	ABC
∅	∅	∅

DFA:



Minimization of DFA – partition method.

→ Minimization of DFA is required to obtain the minimal version of any DFA which consists of the minimum number of states possible.

→ We can combine two states into one state if they are equivalent.

→ Two states 'A' and 'B' are said to be equivalent if

$\delta(A, X) \rightarrow F$ and $\delta(B, X) \rightarrow F$ OR $\delta(A, X) \not\rightarrow F$ and $\delta(B, X) \not\rightarrow F$

X = is any input string.

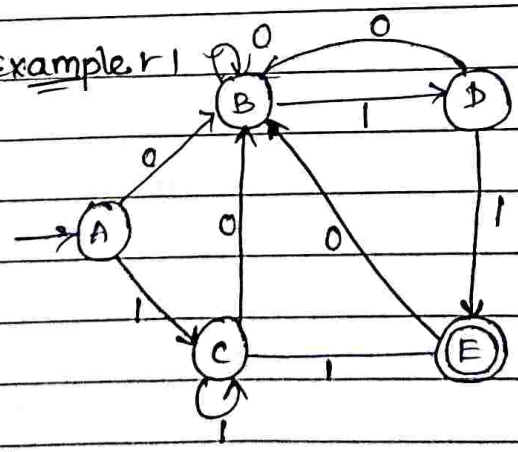
If $|x|=0$, then A and B are said to be 0 equivalent

$|x|=1$, then A and B are said to be 1 equivalent.

⋮

$|x|=n$, then A and B are said to be n equivalent.

Example 1



Transition table of DFA

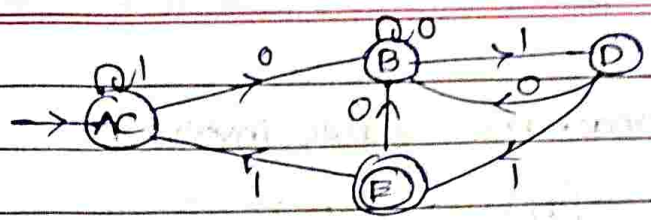
	0	1
→ A	B	C
B	B	D
C	B	C
D	B	E
(E)	B	C

0 Equivalence : {A, B, C, D}, {E}

1 Equivalence : {A, B, C} {D} {E}

2 Equivalence : {A, C} {B} {D} {E}

3 Equivalence : {A, C} {B} {D} {E}



Example 2

construct a minimum DFA equivalent to the DFA described by.

Transition table DFA

	0	1
→ q_0	q_1	q_5
q_1	q_6	q_2
(q_2)	q_0	q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

0 Equivalence

$\{q_0, q_1, q_3, q_4, q_5, q_6, q_7\}$ $\{q_2\}$

1 Equivalence

$\{q_0, q_4, q_6\}$ $\{q_1, q_7\}$ $\{q_2\}$

2 Equivalence

$\{q_0, q_4\}$ $\{q_6\}$ $\{q_1, q_7\}$ $\{q_2\}$, $\{q_2, q_5\}$

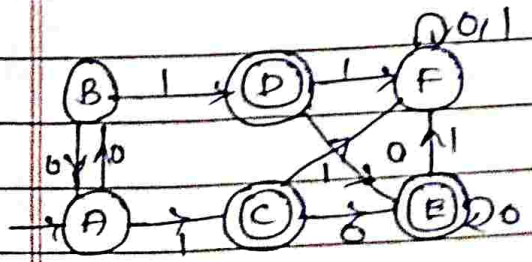
3 Equivalence

$\{q_0, q_4\}$ $\{q_6\}$ $\{q_1, q_7\}$ $\{q_2\}$ $\{q_2, q_5\}$

δ	0	1
→(q_0 q_4)	$\{q_1, q_7\}$	$\{q_5\}$
$\{q_6\}$	$\{q_6\}$	$\{q_4\}$
$\{q_1, q_7\}$	$\{q_6\}$	$\{q_2\}$
$\{q_2, q_5\}$	$\{q_2, \}$	$\{q_6\}$
$\{q_2\}$	$\{q_0\}$	$\{q_2\}$

Example 3

when there are more than one final state involved



	0	1
→A	B	C
B	A	D
(C)	E	F
(D)	E	F
(E)	E	F
F	F	F

0- Equivalence

$\{A, B, F\}$, $\{C, D, E\}$

1- Equivalence

$\{A, B\}$ $\{F\}$ $\{C, D, E\}$

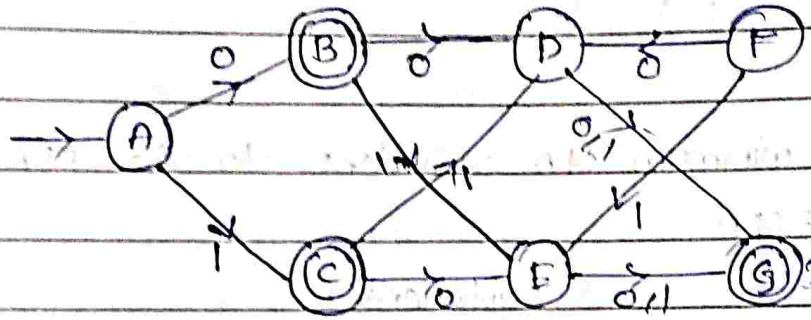
2- Equivalence

$\{A, B\}$ $\{F\}$ $\{C, D, E\}$

classmate
Date 25/06/23
Page 13

Example 4

When there are unreachable states involved.



Unreachable states

A state is said to be unreachable if there is no way it can be reached from the initial state.

Transition states

	0	1
→A	B	C
(B)	D	E
(C)	E	D
D	G	G
E	G	G
G	G	G

0 - Equivalence {A, D, E}, {B, C, G}

1 - Equivalence {A, D, E}, {B, C}, {G}

2 - Equivalence {A}, {D, E}, {B, C}, {G}

3 - Equivalence {A}, {D, E}, {B, C}, {G}

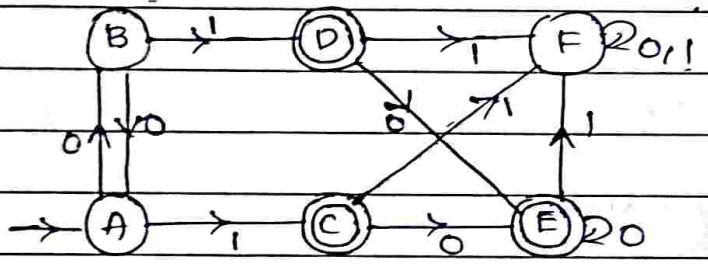
Minimization of DFA — Table Filling Method.

(Myhill – Nerode Theorem)

Steps:

1) Draw a table for all pairs of states (P, Q)
2) Marks all pairs where $p \in F$ and $Q \in F$
3) If there are any unmarked pairs (P, Q) such that $[\delta(P,x), \delta(Q,x)]$ is marked, then mark [P, Q] where 'x' is an input symbol

   Repeat this until No more markings can be made
4) Combine all the unmarked Pairs and make them a single state in the minimized DFA.



	A	B	C	D	E	F
A						
B						
C	✓	✓				
D	✓	✓				
E	✓	✓				
F	✓	✓	✓	✓	✓	

(B, A) — $\delta(B,0) = A$, $\delta(B,1) = D$ } unmarked
$\delta(A,0) = B$, $\delta(A,1) = C$ }

unmarked

(D, C) — $\delta(D,0) = E$, $\delta(D,1) = F$ } leave
$\delta(C,0) = E$, $\delta(C,1) = F$ } leave

(E, C) — $\delta(E,0) = E$, $\delta(E,1) = F$ } leave
$\delta(C,0) = E$, $\delta(C,1) = F$ } leave

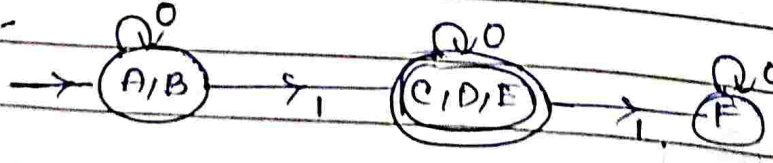
(E,D) — $\delta(E,0) = E$ } leave $\delta(E,1) = F$ } leave.
$\delta(D,0) = E$ } $\delta(D,1) = F$ }

(F,A) — $\delta(F,0) = F$ } leave $\delta(F,1) = F$ } mark it.
$\delta(A,0) = B$ } unmarked $\delta(A,1) = C$ }

	A	B	C	D	E	F
A						
B						
C	✓	✓				
D	✓	✓				
E	✓	✓				
F	✓	✓	✓	✓	✓	

(E,B) — $\delta(F,0) = F$ } mark it
$\delta(B,0) = A$ }

(A,B), (D,C) (E,C) (E,D)



Example :-

Minimize the following DFA using Table Filling Method.

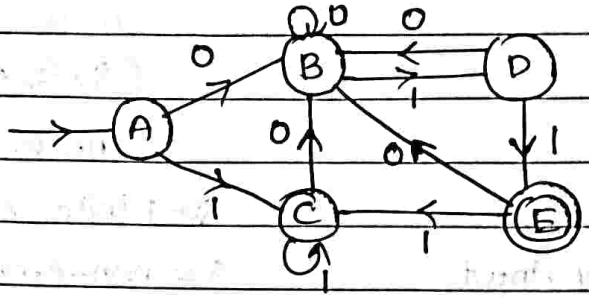


Table :-

	A	B	C	D	E
A					
B	✓				
C		✓			
D	✓	✓	✓		
E	✓	✓	✓	✓	

$(B,A) - \delta(A,0) = B \;\&\; \delta(A,1) = C$
$\delta(B,0) = B \;\&\; \delta(B,1) = D$

$(A,C) - \delta(A,0) = B \;\&\; \delta(A,1) = C$
$\delta(C,0) = B \;\&\; \delta(C,1) = C$

$(C,B) - \delta(C,0) = B \;\&\; \delta(C,1) = C$
$\delta(B,0) = B \;\&\; \delta(B,1) = D$

$(D,A) - \delta(D,0) = B \;\&\; \delta(D,1) = E$
$\delta(A,0) = B \;\&\; \delta(A,1) = C$

$(D,B) - \delta(D,0) = B \;\&\; \delta(D,1) = E$
$\delta(B,0) = B \;\&\; \delta(B,1) = D$

$(D,C) - \delta(D,0) = B \;\&\; \delta(D,1) = E$
$\delta(E,0) = B \;\&\; \delta(C,1) = C$

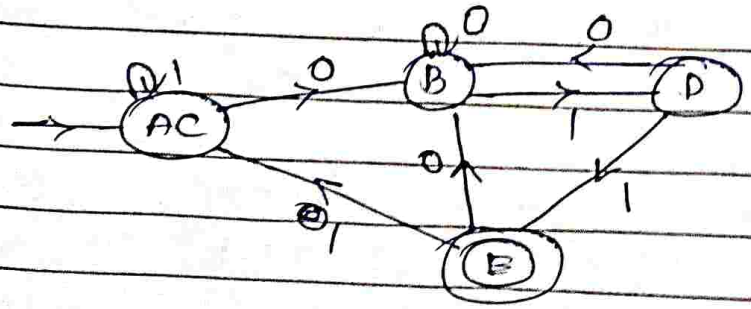
Again

$(B,A) - \delta(A,0) = B \;\&\; \delta(A,1) = D$
$- \delta(B,0) = B \;\&\; \delta(B,1) = E$

$(C,B) = \delta(C,0) = B \;\&\; \delta(C,1) = C$
$\delta(B,0) = B \;\&\; \delta(B,1) = D$

$(C,A) - \delta(C,0) = B \;\&\; \delta(C,1) = C$
$- \delta(A,0) = B \;\&\; \delta(A,1) = C$

AC B D E



classmate
Date 26/06/23
Page 18

Finite Automata with Outputs.

There are two types of FA with output.

Mealy Machine

$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$

where

Q = Finite set of states

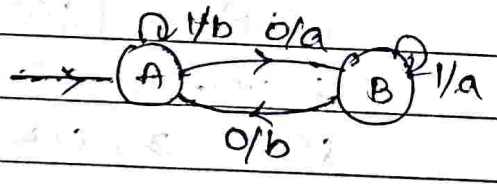
Σ = non-empty set of Input

Δ = The set of output Alphabets

δ = Transition function: $Q \times \Sigma \to Q$

λ = Output function: $\Sigma \times Q \to \Delta$

q_0 = initial state / start state



Moore Machine

$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$

where.

Q = Finite set of states

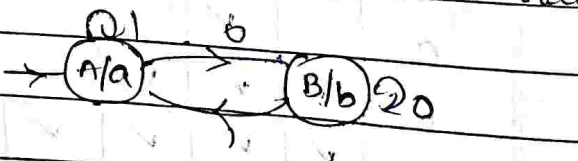
Σ = non-empty set of input

Δ = The set of output Alphabets

δ = Transition func :- $Q \times \Sigma \to Q$

λ = output func r $Q \to \Delta$

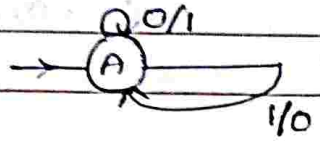
q_0 = initial state / start state



Construction of Mealy Machine

Example :-1

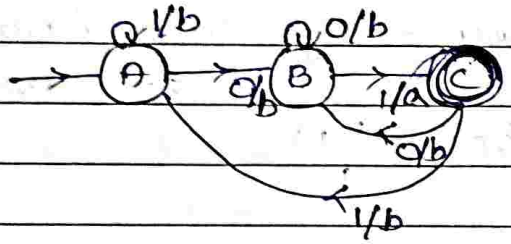
construct a Mealy Machine that produces the 1's complement of any binary input string.



Example 2

construct a Mealy Machine that prints 'a' whenever the sequence '01' is encountered in any input binary string.

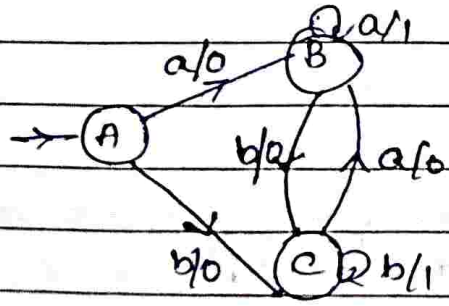
$\Sigma = \{0,1\}$, $\Delta = \{a,b\}$



Example 3

Design a Mealy Machine accepting the language consisting of strings from Σ^*, where $\Sigma = \{a,b\}$ and the strings should end with either aa or bb.

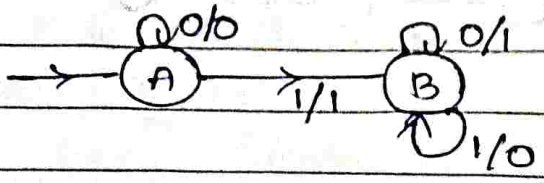
$\Sigma = \{a,b\}$



Example 4

construct a Mealy Machine that gives 2's complements of any binary input. (Assume that last carry is neglected)

2's complement = 1's complement + 1



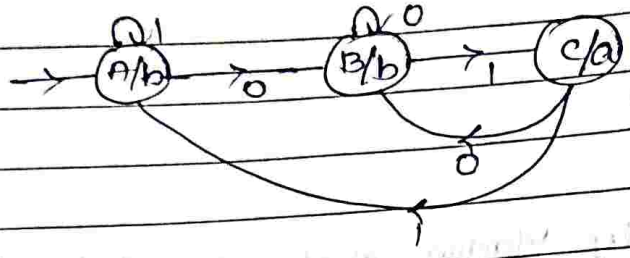
Date 28/06/23
Page 20

Construction of Moore Machine.

Example 1

construct a Moore Machine that prints 'a' whenever the sequence '01' is encountered in any input binary string.

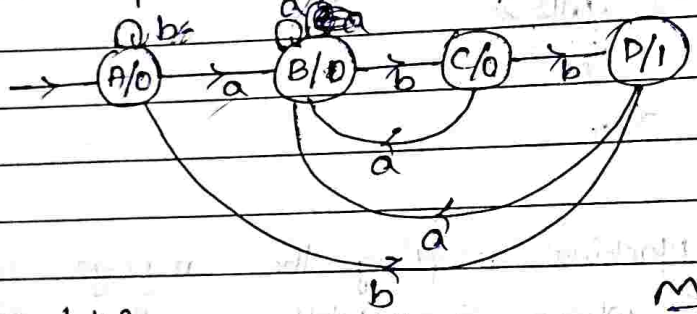
$\Sigma = \{0,1\}$



Example 2

construct a Moore Machine that counts the occurrences of the sequence 'abb' in any strings over {a,b}

$\Sigma = \{a,b\}$, $A = \{0,1\}$



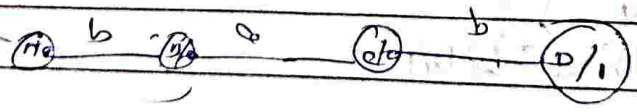
Moore to mealy

Example 3

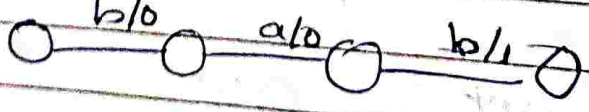
Not required

Moore

bab



Mealy machine (bab)

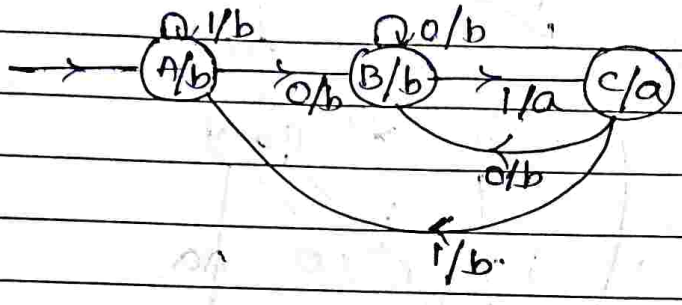


conversion of Moore Machine to Mealy Machine.

Example1

construct a Moore Machine that prints 'a' whenever the sequence '01' is encountered in any input binary string and then convert it into equivalent Mealy Machine.

Moore Machine



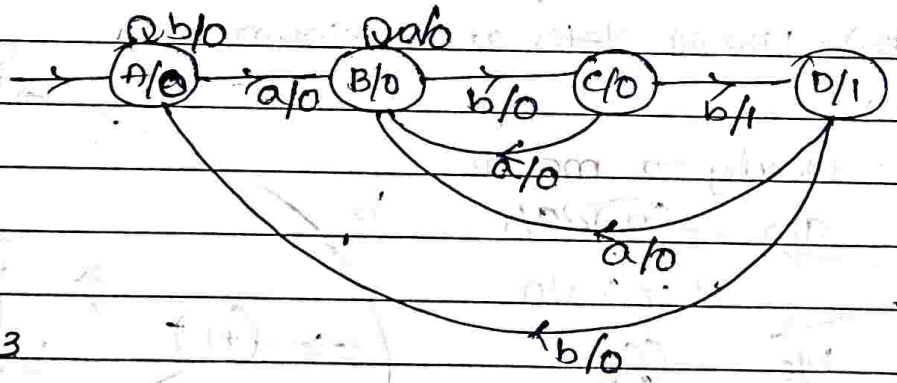
Moore Machine ⟷ Mealy Machine

To get Mealy add the same output of state consist to input values.

Example 2

The given Moore Machine counts the occurences of sequences 'abb' in any binary input strings over {a,b}. convert to Mealy

$\Sigma = \{a, b\}$, $\Delta = \{0, 1\}$



Example 3

convert the given Moore Machine to its equivalent Mealy Machine

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

State	0	1	output
→1 q_0	q_1	q_2	1
q_1	q_3	q_2	0
q_2	q_2	q_1	1
q_3	q_0	q_3	1

Mealy

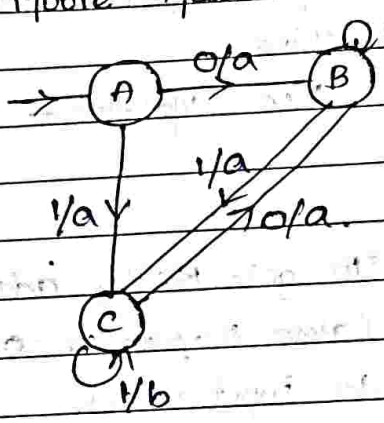
State	0	1
→1 q_0	q_1,0	q_2,1
q_1	q_3,1	q_2,1
q_2	q_2,1	q_1,0
q_3	q_0,1	q_3,1

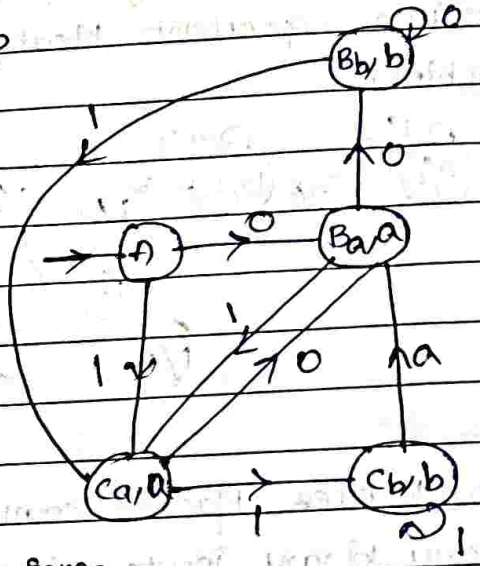
Check for $(q_0, 0) = q_1$ and q_1 output is 0 so keep $(q_1, 0)$ in Mealy Machine of $(q_0, 0)$.

conversion of Mealy Machine to Moore Machine

Example 1

convert the following Mealy Machine to its equivalent Moore Machine. $\Sigma = \{a, 1\}$, $\Delta = \{a, b\}$





Note:

Moore → Mealy

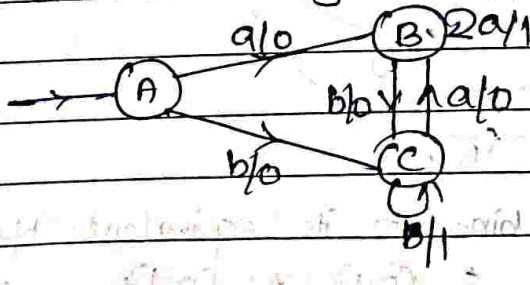
↳ no. of states were same.

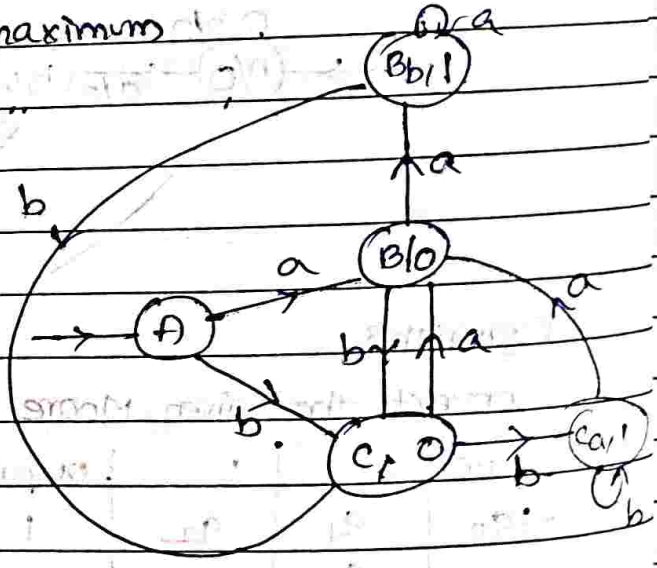
Mealy → Moore → no. of states increased.

↳ (n*y) → no. of states at maximum

Example 2

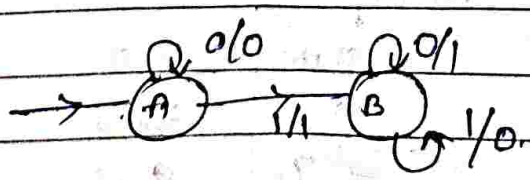
convert mealy to moore

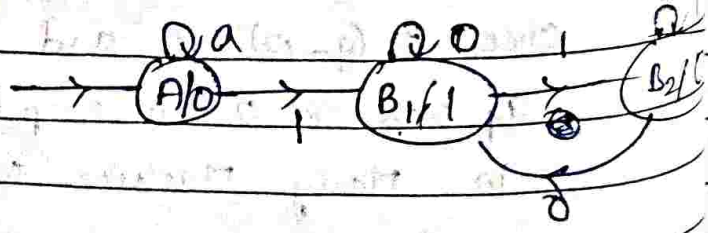




Example 3

convert mealy to moore?

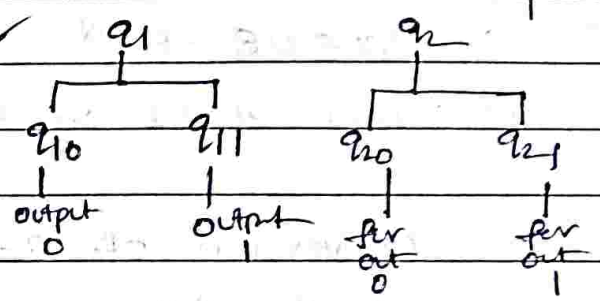




Example-4

Convert the given Mealy Machine to its equivalent Moore Machine.

State	a	b	
→ q_0	$q_3, 0$	$q_1, 1$	q_0, q_3
q_1	$q_0, 1$	$q_3, 0$	
q_2	$q_2, 1$	$q_2, 0$	
q_3	$q_1, 0$	$q_0, 1$	



State	a	b	output
q_0	$q_3, 0$	$q_{11}, 1$	1
q_{10}	$q_0, 1$	$q_3, 0$	0
q_{11}	$q_0, 1$	$q_3, 0$	1
q_{20}	$q_{21}, 1$	$q_{20}, 1$	1
q_{21}	$q_{21}, 1$	$q_{20}, 0$	1
q_3	$q_{10}, 0$	$q_0, 1$	0

(ϵ)

Epsilon - NFA

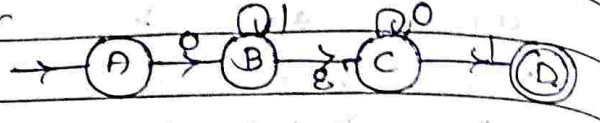
ϵ - NFA

↳ Empty symbols (or) null symbols.

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

$\delta = Q \times \Sigma \cup \epsilon \rightarrow 2^Q$

Ex:

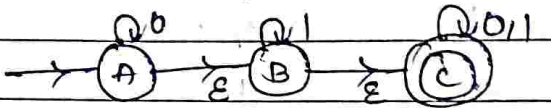


→ Every state on ϵ goes to itself.

conversion of NFA-ϵ to NFA with example & explanation.

Conversion of ϵ - NFA to NFA :-

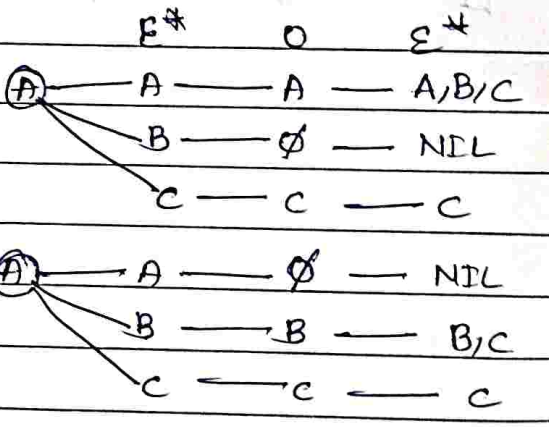
convert the following ϵ-NFA to its equivalent NFA.

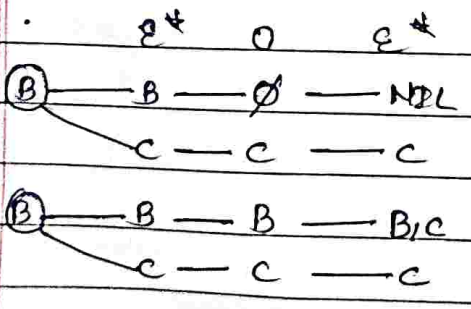


ϵ* (ϵ-closure) :-

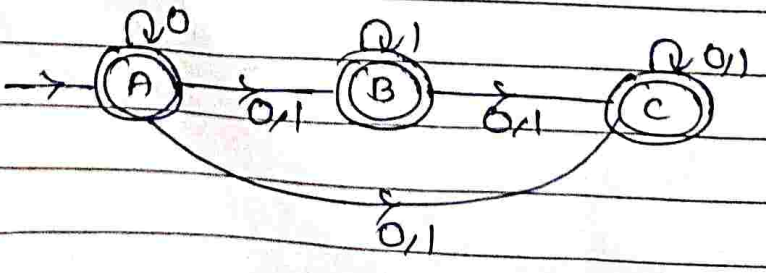
All the states that can be reached from a particular state only by seeing the ϵ symbol.

	0	1	ϵ
→A	A	∅	B
B	∅	B	C
C	C	C	∅



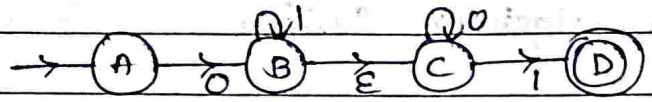


NFA :-



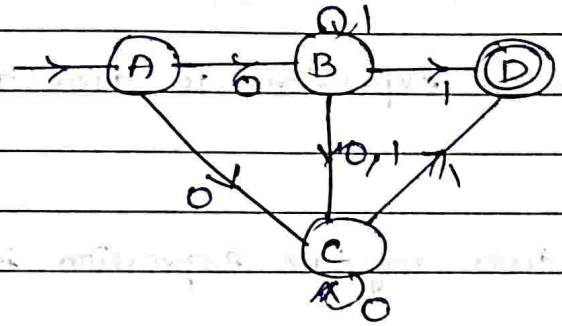
Example 1

Convert the following ε-NFA to its equivalent NFA.



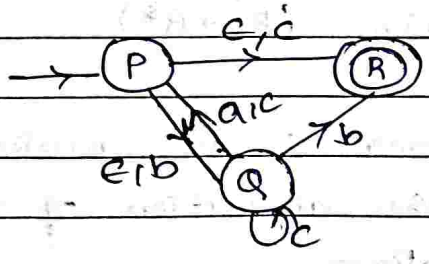
	0	1	ε
→ A	B	∅	∅
B	∅	B	C
C	C	D	∅
D	∅	∅	∅

NFA :

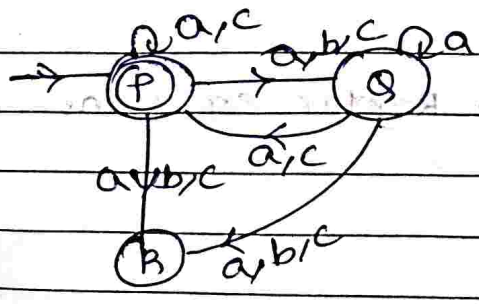


Example 2

Convert the following ε-NFA to NFA.



	a	b	c	ε
P	∅	Q	(R)	Q,R
Q	P	(R)	P,Q	∅
R	∅	∅	∅	∅



Regular Expression

→ Regular Expressions are used for representing certain set of strings in an algebraic fashion.

Rules:

1) Any terminal symbol i.e symbols $\in \Sigma$ including $\cdot$ $\wedge$ and ϕ are regular expressions.

2) The union of two regular expressions is also an regular expression.

3) The concatenation of two regular expression is also an regular expression.

4) The iteration (or closure) of regular expression is also an regular expression. $(R \rightarrow R^*)$

5) The regular expression over Σ are precisely those obtain recursively by the application of the above rules once or several times.

Examples:

Describe the following set of Regular expressions.

1) $\{0,1,2\}$ — $R = 0+1+2$

2) $\{\wedge, ab\}$ — $R = \wedge ab$

3) $\{abb, a, b, bba\}$ — $R = abb+a+b+bba$

4) $\{\wedge, 0, 00, 000, \ldots\}$ — $R = 0^*$

5) $\{1, 11, 111, 1111, \ldots\}$ — $R = \phi^* 1+$

Identities of Regular Expression

1) $\emptyset + R = R$

2) $\emptyset R + R\emptyset = \emptyset$

3) $\epsilon R = R\epsilon = R$

4) $\epsilon^* = \epsilon$ and $\emptyset^* = \epsilon$

5) $R + R = R$

6) $R^* R^* = R^*$

7) $RR^* = R^* R$

8) $(R^*)^* = R^*$

9) $\epsilon + RR^* = \epsilon + R^* R = R^*$

10) $(PQ)^* P = P(QP)^*$

11) $(P+Q)^* = (P^* Q^*)^* = (P^* + Q^*)^*$

12) $(P+Q)R = PR + QR$ and

$R(P+Q) = RP + RQ$.

ARDEN'S THEOREM

If p and Q are the Regular Expressions over Σ, and if p does not contain ϵ, then the following equation in R given by $R = Q + Rp$ has unique solution i.e. $R = Qp^*$

$R = Q + Rp$ —— (1)

$= Q + (Qp^*)p$ $\quad R = Qp^*$

$= Q(\epsilon + p^*p)$ $\quad \boxed{\epsilon + R^*R = R^*}$ — identity.

$\boxed{R = Qp^*}$ proved.

unique solution

$R = Q + Rp$

$R = Q + (Q + Rp)p \quad \because R = Q + Rp$

$R = Q + Qp + Rp^2$

$R = Q + Qp + (Q + Rp)p^2$

$= Q + Qp + Qp^2 + Rp^3$

$\vdots$

$= Q + Qp + Qp^2 + \cdots + Qp^n + Rp^{n+1}$

$R = Q + Qp + Qp^2 + \cdots + Qp^n + Qp^*p^{n+1}$

$R = Q[\epsilon + p + p^2 + \cdots + p^n + p^*p^{n+1}]$

$\boxed{R = Qp^*}$ (closure of p) $= p^*$

An Example Proof of using identities of R.E

prove that $(1+00^*1)+(1+00^*1)(0+10^*1)^*(0+10^*1)$ is equal to $0^*1(0+10^*1)^*$

$$\text{LHS} = (1+00^*1)+(1+00^*1)(0+10^*1)^*(0+10^*1)$$

$$= (1+00^*1)[\epsilon + (0+10^*1)^*(0+10^*1)] \quad (\because \epsilon + R^*R = R^*)$$

$$= (1+00^*1)[0+10^*1)^*$$

$$= (\epsilon \cdot 1+00^*1)(0+10^*1)^*$$

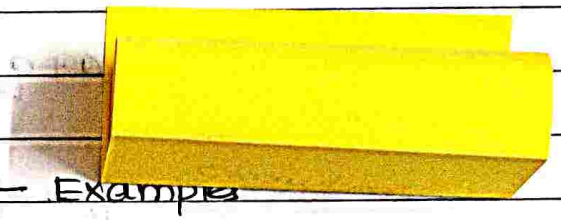
$$= (\epsilon+00^*)1(0+10^*1)^* \quad (\because \epsilon + RR^* = R^*).$$

$$= (0)^*1(0+10^*1)^*$$

$$= 0^*1(0+10^*1)^* = \text{RHS}$$

So

LHS = RHS



Designing Regular Expressions – Examples

Design R.E expression for following languages over $\{a,b\}$

1) language accepting strings of length exactly 2
2) language accepting strings of length atleast 2
3) language accepting strings of length atmost 2

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

1) $L_1 = \{aa, bb, ab, ba\}$

$R = aa+ab+ba+bb$

$= a(a+b) + b(a+b) = (a+b)(a+b)$

2) $L_2 = \{aa, ab, ba, bb, aaa, \ldots\}$

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

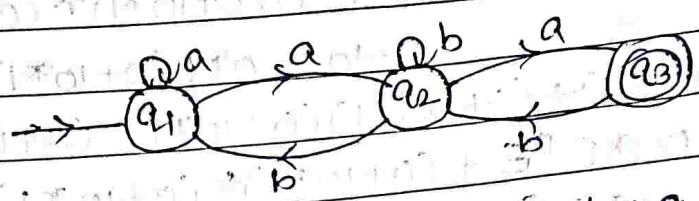
3) $L_3 = \{\epsilon, a, b, aa, ab, ba, bb\}$

$R = \epsilon + a+b+aa+ab+ba+bb$

$= (\epsilon+a+b)(\epsilon+a+b)$

Regular Expression for following NFA.

Example 1



$q_3 = q_2 a \longrightarrow (1)$, $q_2 = q_1 a + q_2 b + q_3 b \longrightarrow (2)$

$q_1 = \epsilon + q_1 a + q_2 b \longrightarrow (3)$

Lets take $q_3 = q_2 a$

$q_3 = (q_1 a + q_2 b + q_3 b) a$

$q_3 = q_1 aa + q_2 ba + q_3 ba \longrightarrow (4)$

Lets take $q_2 = q_1 a + q_2 b + q_3 b$ putting value of q_3 of (1)

$= q_1 a + q_2 b + q_2 ab$

$q_2 = q_1 a + q_2 (b + ab)$

$q_2 = (q_1 a)(b + ab)^* \longrightarrow (5)$

let (3), $q_1 = \epsilon + q_1 a + q_2 b$

$q_1 = \epsilon + q_1 a + ((q_1 a)(b + ab)^*) b$

$q_1 = \epsilon . (((q_1 a)(b + ab)^*) b)^*$

$q_1 = ((q_1 a)(b + ab)^* b)^*$

lets (3)

$q_1 = \epsilon + q_1 a + q_2 b$

$q_1 = \epsilon + q_1 a + ((q_1 a)(b + ab)^*) b$

$q_1 = \epsilon + q_1 (a + a(b + ab)^*) b$

$= \epsilon . (a + a(b + ab)^* b)^*$

$q_1 = (a + a(b + ab)^* b)^* \longrightarrow (6)$

Assignment
Date 29/08/22
Page 21

Final state.

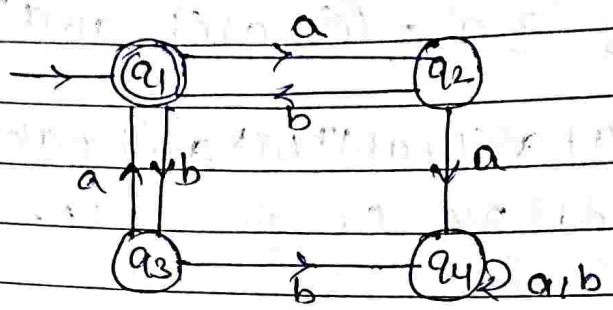
$$q_3 = q_2 a = ((q_1 a)(b+ab)^*)a$$

$$q_3 = (a+a(b+ab)^* b)^* a (b+ab)^*)a$$

= Required R.E for given NFA.

Regular Expression ~~for~~ following DFA.

Example:-2



$q_1 = \epsilon + q_2 b + q_3 a$ — (1)

$q_2 = q_1 a$ — (2)

$q_3 = q_1 b$ — (3)

$q_4 = q_2 a + q_3 b + q_4 a + q_4 b$ — (4)

(1) →, $q_1 = \epsilon + q_2 b + q_3 a$

$= \epsilon + q_1 ab + q_1 ba = \epsilon + q_1 (ab + ba)$

$q_1 = \epsilon \cdot (ab + ba)^*$

$q_1 = (ab + ba)^*$

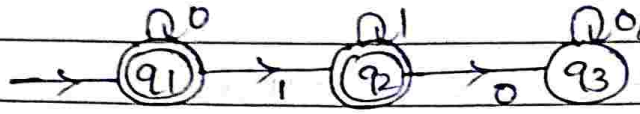
done.

DFA to R.E when single final state.

Regular Expression for following DFA

(When there are Multiple Final states)

Example



$q_1 = \epsilon + q_1 0$ — (1)

$q_2 = q_1 1 + q_2 1$ — (2)

$q_3 = q_2 0 + q_3 0 + q_3 1$ — (3)

Final state q_1

$q_1 = \epsilon + q_1 0$ $\quad (\because R = Q + RP)$ — (6)

$q_1 = \epsilon . 0^*$

$q_1 = 0^*$ — (4)

Final state q_2.

$q_2 = q_1 1 + q_2 1$

$= 0^* 1 + q_2 1$ — by same (6) formula.

$q_2 = 0^* 1 . 1^*$ — (5)

A = union of both final states.

$R = 0^* + 0^* 1 (1)^*$

$= 0^* (\epsilon + (1)^*)$

$= 0^* . (1)^*$ ⟶ Regular expression.

DFA to R.E when more than one final states.

Conversion of Regular Expression to Finite Automata

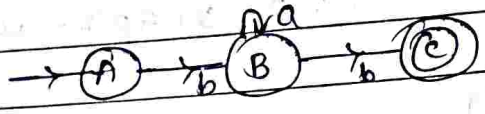
~~remember~~ Example 1

convert the following Regular Expressions to their equivalent Finite Automata:

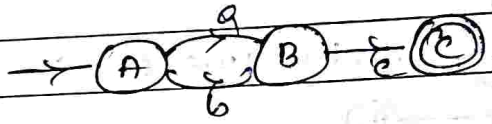
i) $b.a^*.b$ ii) $(a+b)c$ iii) $a(bc)^*$

1) ba^*b

$= \{bb, bab, baab, \dots\}$

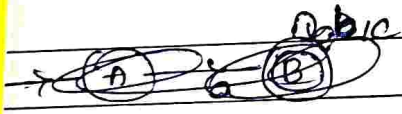


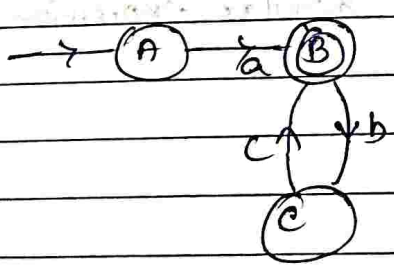
$(a+b).c = \{ac, bc\}$



+ and | are same usage.

$(bc)^* = \{a, abc, abcbc, \dots\}$



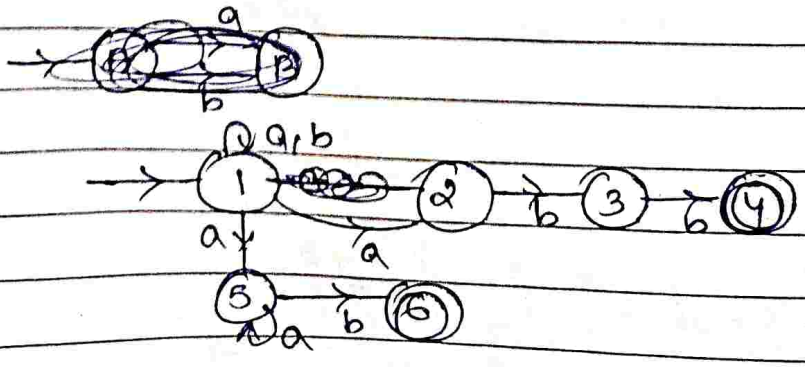


Example 2

convert the following R.E to its equivalent F.A

$(a|b)^*(abb|a+b)$

$(a+b)^*(abb+a+b)$



Conversion of Regular Expression to Finite Automata

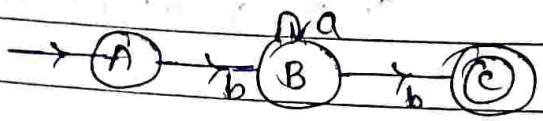
~~remember~~ Example 1

convert the following Regular Expressions to their equivalent Finite Automata.

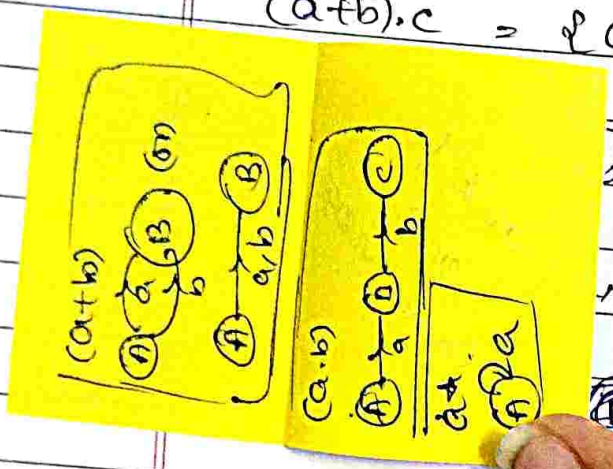
1) $b\,a^*\,b$ 2) $(a+b)c$ 3) $a(bc)^*$

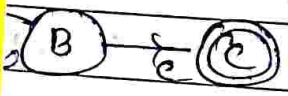
1) ba^*b

$= \{bb, bab, baab, \ldots\}$



$(a+b).c = \{ac, bc, \ldots\}$



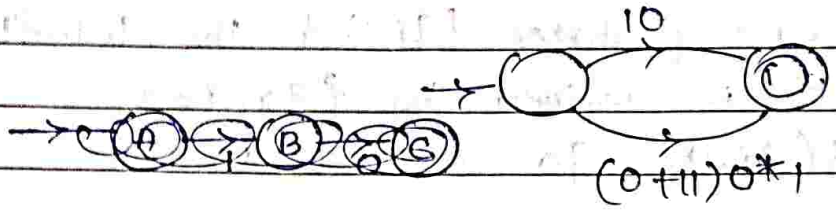


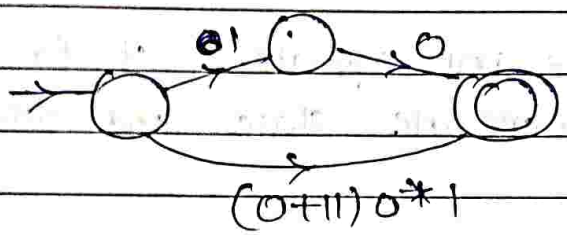
$\ldots, abc, abcbc, \ldots\}$

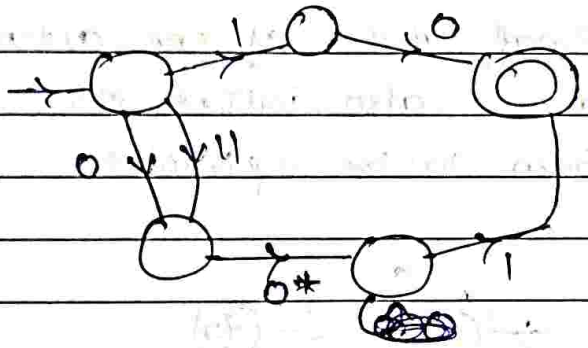
Example 3

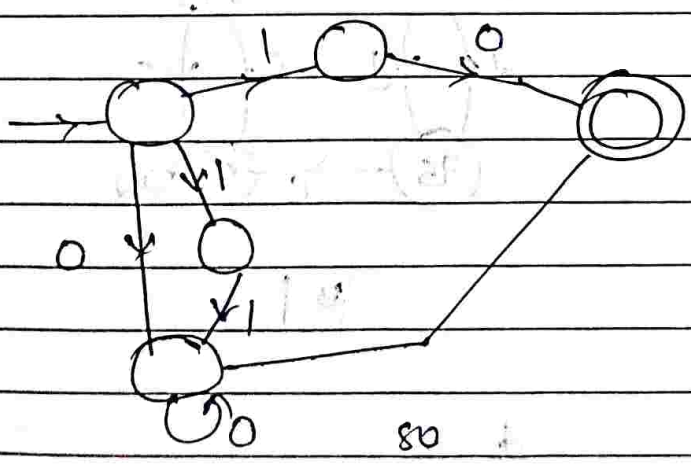
convert the following R.E to equivalent F.A.

$10 + (0+11)0^*1$



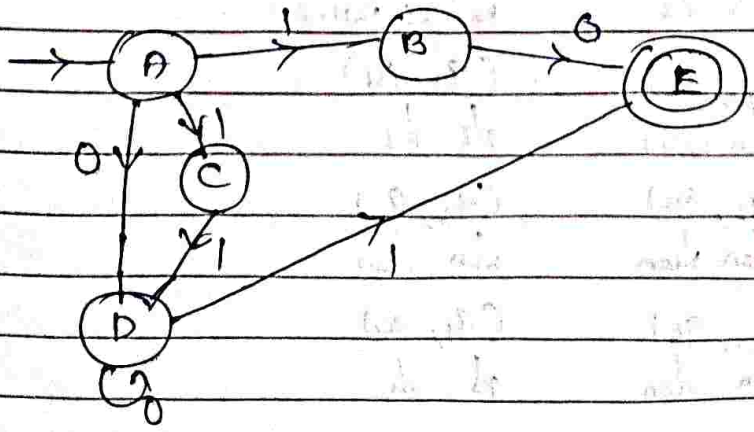






80

F.A is



Equivalence of two Finite Automata.

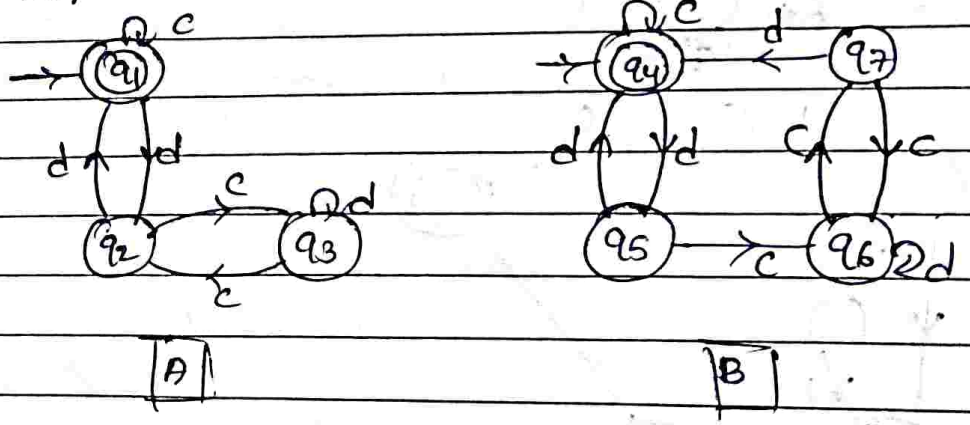
Steps to identify equivalence :-

1) For any pair of states $\{q_i, q_j\}$ the transition for input $a \in \Sigma$ is defined by $\{q_a, q_b\}$ where $\delta\{q_i, a\} = q_a$ & $\delta(q_j, a\} = q_b$

The two automata are not equivalent if for a pair $\{q_a, q_b\}$ one is Intermediate state and other is final state.

2) If initial state is Final state of one automaton, then in second automaton also initial state must be final state for them to be equivalent.

Example:-

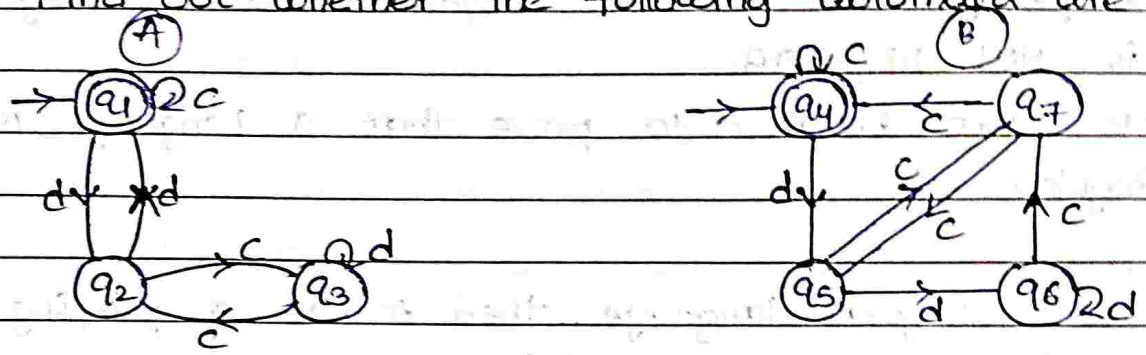


States	c	d
(q_1, q_4)	(q_1, q_4) FS FS	(q_2, q_5) NON-FS NON-FS
(q_2, q_5)	(q_3, q_6) Non Non	(q_1, q_4) FS FS
(q_3, q_6)	(q_2, q_7) Non Non	(q_3, q_6) Non Non
(q_4, q_7)	(q_3, q_6) non non	(q_4, q_4) FS FS

A & B are equivalent.

Example 1

Find out whether the following automata are equi (or) not.



States	c	d
(q_1, q_4)	(q_1, q_4)	(q_2, q_5)
	FS FS	Non Non
(q_2, q_5)	(q_3, q_7)	(q_1, q_6)
	Non Non	FS Non

Final state & non-final state cannot be in one pair.

So, A & B are not equivalent.

pumping lemma (For Regular languages)

→ pumping lemma is used to prove that a language is NOT REGULAR
→ It cannot be used to prove that a language is Regular.

If A is Regular language, then A has a pumping length 'p' such that any string 's'
→ where $|s| \geq p$ may be divided into 3 parts $s = xyz$ such that following conditions must be true:
(1) $xy^iz \in A$ for every $i \geq 0$
(2) $|y| > 0$
(3) $|xy| \leq p$.

Steps to prove language is not Regular
(we prove using contradiction)
→ Assume that A is Regular
→ It has to have a pumping length (say p)
→ All strings longer than p can be pumped $|s| \geq p$
→ Now find a string 's' in A such that $|s| \geq p$.
→ Divide s into xyz
→ Show that $xy^iz \notin A$ for some i
→ Then consider all ways that s can be divided into xyz
→ Show that none of these can satisfy all the 3 pumping conditions at same time.
→ s cannot be pumped == CONTRADICTION

Example 1

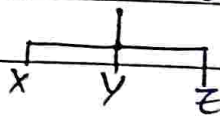
using pumping lemma prove that the language $A = \{a^n b^n \mid n \geq 0\}$ is not Regular.

A) $A = \{a^n b^n \mid n \geq 0\}$

Assume that A is Regular.

pumping length = p.

$S = a^P b^P \Rightarrow S =$ aaaaaaaabbbbbbbb.



If p=7

case 1 : The y is in the 'a' part

aaaaaaa bbbbbbb

x y z

$xy^1z \Rightarrow xy^2z$

aayyyyyyy bbbbbbb

11 ≠ 7

case 2 : The y is in 'b' part

aaaaaaabbbbbbbb

x y z

$xy^1z \Rightarrow xy^2z$

aaaaaaabbbbbbbbbbbb

7 ≠ 11

case 3 : The y is in 'a' & 'b' part

aaaaaaabbbbbbb

x y z

$xy^1z \Rightarrow xy^2z$

aaaaaaaaabbbbbbbbb

9 ≠ 7

$|xy| \leq p$ — case i — OK

Case - ii — NO

Case - iii — NO.

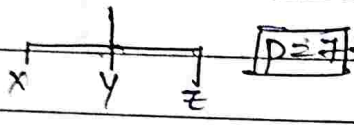
Example 12

using pumping lemma prove that the language $A = \{yy \mid y \in \{0,1\}^*\}$ is not Regular.

A) Assume A is regular

The pumping length = p

$S = 0^p 1 0^p 1$



$S = 0^7 1 0^7 1$

$= 00000001 00000001$

x y z

(1)

$xy^iz = xy^2z$

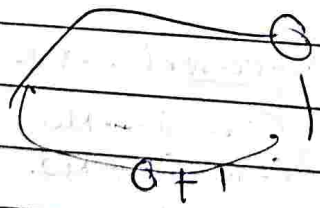
$000000000001 00000001$

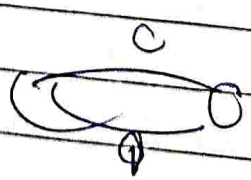
$\notin A$

$|y| > 0$ — (2)

$|xy| \leq p$ — (3).

A is not Regular for not satisfing (1).





Regular Grammar

→ Noam chomsky gave a Mathematical model of Grammar which is effective for writing computer languages.

There are four types of grammar according to Noam Chomsky!-

Grammar type	grammar Accepted	language Accepted	Automaton
Type-0	Unrestricted Grammar	Recursively Enumerable language	Turing machine
Type-1	context sensitive grammar	context sensitive language	Linear Bounded Automaton
Type-2	context free grammar	context Free language	pushdown Automaton
Type-3	Regular grammar	Regular language	Finite state Automaton

Grammar

A Grammar 'G' can be formally described using 4 tuples as $G = (V, T, S, P)$ where,

V = set of variables (or) Non-Terminal symbols

T = set of Terminal symbols

S = start symbol

P = production rules for Terminals and Non-Terminals

A production rule has the form $\alpha \rightarrow \beta$

where α, β are strings on VUT and atleast One symbol of α belongs to V,

Example

$G = (\{S, A, B\}, \{a, b\}, S, \{S \rightarrow AB, A \rightarrow a, B \rightarrow b\})$

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

$T = \{a, b\}$

$S = S$

$P = \{S \rightarrow AB, A \rightarrow a, B \rightarrow b\}$

Eqr

$S \rightarrow AB$

$\rightarrow aB$

$\rightarrow ab$

classmate
Date 30/08/23
Page 42

Regular Grammar

Regular Grammar can be divided into two types.

Right linear Grammar

→ A grammar is said to be Right linear if all productions are of the form

$A \longrightarrow xB$

$A \longrightarrow x$

where $A, B \in V$ & $x \in T$

Ex:

$S \rightarrow abS \mid b$ — Right linear

$Sbab \mid b$ — left linear

left linear Grammar

→ A grammar is said to be left linear if all productio are of the form

$A \longrightarrow Bx$

$A \longrightarrow x$

where $A, B \in V$ & $x \in T$

Derivations from a Grammar :-

→ The set of all strings that can be derived from a Grammar is said to be the language generated from that Grammar

Example 1

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

$S \to aAb$ [by $S \to aAb$]

$S \to aaAbb$ [by $aA \to aaAb$]

$S \to aaaAbbb$ [by $aA \to aaAb$]

$S \to aaa\epsilon bbb$ [by $A \to \epsilon$]

$S \to aaabbb$ [by $A \to \epsilon$]

Example 2

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

$S \to AB$

$S \to aB$ [by $A \to a$]

$S \to ab$ [by $B \to b$]

$L(G_2) = \{ab\}$

Example 3

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

$S \to AB$

$S \to ab$ [by $A \to aA/a$, $B \to bB/b$]

$S \to AB$

$S \to aAbB$ [by $A \to aA/a$, $B \to bB/b$]

$S \to aabb$ [by $A \to a$]

$S \to AB$

$S \to aAb$ [by $A \to aA/a$, $B \to bB/b$]

$S \to aab$ [by $A \to aA/a$.

$L(G_3) = \{ab, a^2b^2, a^2b, ab^2, \ldots\}$
$= \{a^m b^n\}$

$\therefore \{a^m b^n | m \geq 0 \text{ \& } n \geq 0\}$

$S \to AB$

$S \to abB$ [by $A \to aA/a$, $B \to bB/b$]

$S \to abb$

context Free languages

→ In formal language theory, a context free language is a language generated by some context free grammar

→ The set of context free language is identical to the set of languages accepted by pushdown Automata.

→ context free grammar is defined by 4 tuples as $G = \{V, \Sigma, S, P\}$ where.

V = set of Variables or Non-Terminal Symbols

Σ = set of Terminal Symbols

S = Start Symbol

P = production Rule.

Context Free Grammar has production rule of form

$$A \rightarrow \alpha$$

where, $\alpha = (V \cup \Sigma)^*$ and $A \in V$.

Example

For generating a language that generates equal num of a's and b's in form of $a^n b^n$, the CFL can be defined as $G = \{(S, A), (a, b), S, (S \rightarrow aAb, A \rightarrow aAb/\epsilon)\}$

A) $S \rightarrow aAb$

$S \rightarrow aaAbb$ [by $A \rightarrow aAb/\epsilon$]

$S \rightarrow aaaAbbb$ [by $A \rightarrow aAb/\epsilon$]

$S \rightarrow aaabbb$ [by $A \rightarrow \epsilon$]

$\rightarrow a^3b^3$

$\rightarrow a^n b^n$

so $S \rightarrow a^n b^n$

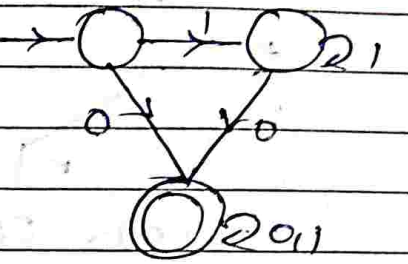
GATE-2013

Solved problem (1)

classmate
Date 07/07/23
Page 43

Regular languages & Finite-Automata

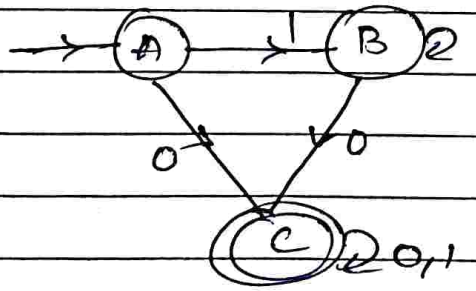
consider the DFA given below:-
which of following is FALSE?
1) complement of L(A) is context-free
2) L(A) = L((11*0+0)(0+1)*0*1*)
3) For the language accepted by A, A is the minimal DFA
4) A accepts all strings over {0,1} of length at least 2.



A) 1 & 3 only B) 2 & 4 only
C) 2 & 3 only D) 3 & 4 only.

1) True, because any DFA is regular it is context-free. The compliment of context free is also context free.

2) L(A) = L((11*0+0)(0+1)*0*1*) — True



$C = A0 + B0 + C0 + C1$ — (1)

$B = A1 + B1$ — (2)

$A = \epsilon$ — (3)

take B

$B = A1 + B1 \Rightarrow B = \epsilon 1 + B1$

$B = 1 + B1$
A Q A P

$B = A1^*$

(i) $C = A0 + B0 + C0 + C1$

$= \epsilon.0 + 11^*0 + C0 + C1$

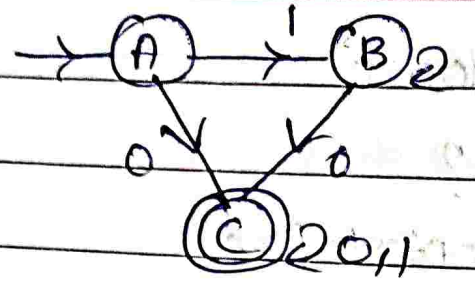
$= 0 + 11^*0 + C0 + C1$

~~$= C(0+0(\epsilon+11^*)+\epsilon)$~~

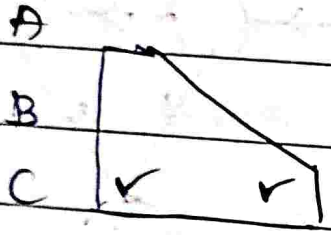
$= 0 + 11^*0 + C(0+1)$
Q R P

$C = Qp^* = 0 + 11^*0(0+1)^*$

$C = 11^*0 + 0(0+1)^*$ ✓

(8) 

— False.

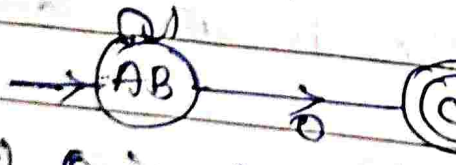


$(B,A) = (B,0) = C$, $(B,1) = B$

$(A,0) = C$ $(A,1) = B$

(c) 0 equivalence $\{A,B\}, \{C\}$

1 equivalence $\{A,B\}, \{C\}$



D) D is false.

GATE-2013

Solved problem – (2)

Regular language & Finite Automata.

Consider the languages $L_1 = \emptyset$ & $L_2 = \{a\}$. which one of the following represents $L_1L_2^* \cup L_1^*$?

A) ϵ ✓

B) $\emptyset$

C) a^*

D) $\{\epsilon, a\}$

$L_1L_2^* \cup L_1^*$

$L_1L_2^* = (\emptyset).(a)^*$

$(\emptyset).(a)^* \cup (\emptyset)^*$

$\emptyset \cup (\emptyset)^*$

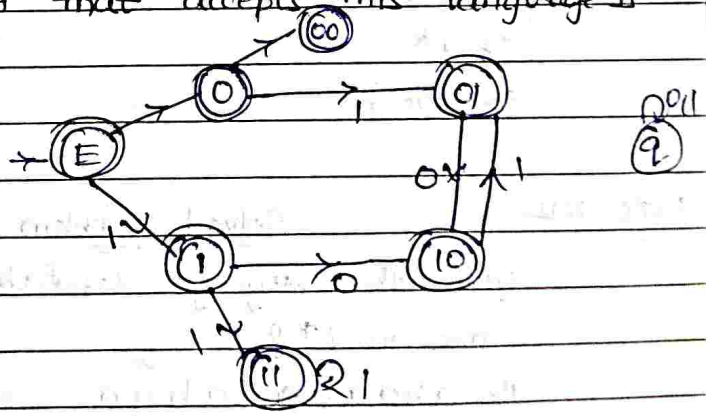
$\emptyset \cup \epsilon = \epsilon$

Solved problem – (3)

GATE – 2013

consider the set of strings over $\{0,1\}$ in which, every substring of 3 symbols has at most two zeros. For example, 001110 & 011001 are in language, but 100010 is not. All strings of length less than 3 are also in language. A partially completed DFA that accepts this language is shown below.

A) D.



GATE-2009

Solved problem-(4)

which one of the following languages over the alphabet {0,1} is described by regular expression $(0+1)^*0(0+1)^*0(0+1)^*$?

A) The set of all strings containing substring 00

B) The set of all strings containing at most two 0's

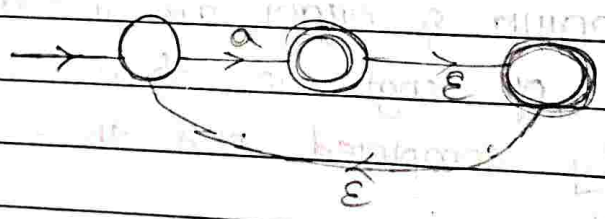
C) The set of all strings containing at least two 0's

D) The set of all strings that begin with 0 & end with either 0 & 1.

GATE-2012

Solved problem - (5)

what is the complement of language accepted by NFA shown below?



A) ϕ ✗

B) {ε} ✓

C) a^*

D) {a, ε}

GATE-2011

Solved problem - (6)

Given the language $L = \{ab, aa, baa\}$, which of the following are in L^*?

1) abaabaaabaa ✓

2) aaaabaaaa ✓

3) baaaaabaaaab ✗

4) baaaaabaa ✓

A) 1, 2 & 3

B) 2, 3 & 4

C) 1, 2 & 4 ✓

D) 1, 3 & 4

Method to Find whether a string belong to grammar or not.

Steps:

1) Start with the start symbol and choose the closest production that matches to the given string.
2) Replace the Variables with its most appropriate production. Repeat the process until the string is generated (or until no other productions are left.

Example:

Verify whether the Grammar $S \rightarrow 0B|1A$, $A \rightarrow 0|0S|1AA|\wedge$, $B \rightarrow 1|1S|0BB$ generates the string 00110101.

A) given 00110101

$S \rightarrow .0B$ $(S \rightarrow 0B)$

$\rightarrow 00BB$ $(B \rightarrow 0BB)$

$S \rightarrow 001B$ $(B \rightarrow 1)$

$\rightarrow 001\$S$ $(B \rightarrow 1S)$

$S \rightarrow 00110B$ $(S \rightarrow 0B)$

$S \rightarrow 001101S$ $(B \rightarrow 1S)$

$S \rightarrow 00110101B$ $(S \rightarrow 0B)$

$S \rightarrow 00110101$ $(B \rightarrow 1)$

So, 00110101 string can generated by grammar.

Example:

Verify whether the Grammar $S \rightarrow aAb$, $A \rightarrow aAb|\wedge$ generates the string aabbb

A) $S \rightarrow aAb$

$S \rightarrow aaAbb$ $(A \rightarrow aAb)$

$S \rightarrow aaaAbbb$ $(A \rightarrow aAb)$

$S \rightarrow aaabbb$ $(A \rightarrow \wedge)$

∴ This string cannot be possible.

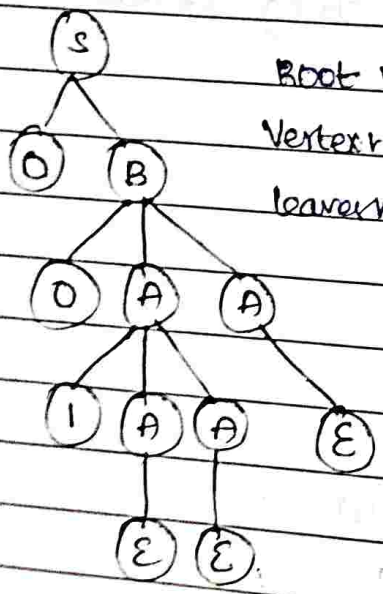
Derivation Tree

→ A Derivation Tree (or) parse Tree is an Ordered rooted tree graphically represents the semantic information of strings derived from a Context Free Grammar.

Example

For the grammar $G = \{V, T, P, S\}$

where $S \rightarrow 0B$, $A \rightarrow 1AA | \varepsilon$, $B \rightarrow 0AA$



Root Vertex Must be labelled by start symbol
Vertex labelled by Non-Terminal symbol
leaves labelled by Terminal symbols (or) ε

There are two types of derivation trees

left-derivation tree

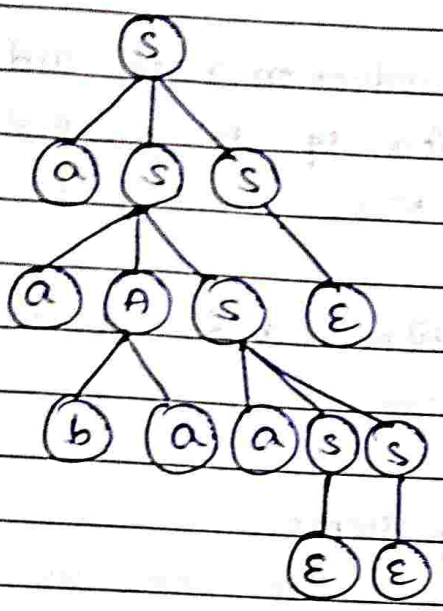
A left-derivation tree is obtained by applying production to the leftmost Variable in each step.

Right-derivation tree

A Right-derivation tree is obtained by applying productions to the rightmost variable in each step.

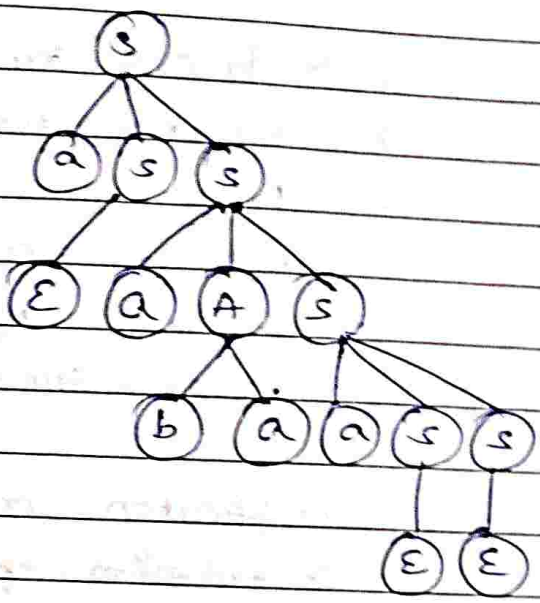
Ex: For generating string aabaa from grammar $S \rightarrow aAs | ass | \epsilon$, $A \rightarrow sbA | ba$

left-derivation tree



aabaa

Right-derivation tree.



aabaa

Ambiguous Grammar

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

Example

G = {{S}, {a+b, +, *}, P, S}, where P consists of S → S+S | S*S | a | b The string a+a*b can be generated as

S → S+S
→ a+S (∵ S → a)
→ a+S*S (∵ S → S*S)
→ a+a*S (∵ S → a)
→ a+a*b (∵ S → b)

S → S*S
→ S+S*S (∵ S → S+S)
→ a+S*S (∵ S → a)
→ a+a*S (∵ S → a)
→ a+a*b (∵ S → b)

Thus, this grammar is Ambiguous.

Date 11/07/23
Page 52

Simplification of context Free Grammar

Reduction of CFG:-

→ In CFG, sometimes all the production rules and symbols are not needed for derivation of strings. Besides this, there may also be some NULL productions and UNIT productions.

→ Elimination of these productions and symbols is called simplification of CFG.

Simplification consists of following steps

1. Reduction of CFG.
2. Removal of unit productions
3. Removal of NULL productions

Reduction of CFG

CFG can reduced in two phases.

phase 1:-

Derivation of an equivalent grammar 'G', from the CFG, G, such that each variable derives some terminal string.

Derivation procedure:-

Step 1-1

Include all symbols W_1, that derives some terminal and initialize $i=1$

Step 1-2

Include symbols W_{i+1}, that derives W_i

Step 1-3

increment i and repeat step 2, until $W_{i+1} = W_i$

step 1-4

Include all production rules that have W_i in it.

phase 2:-

Derivation of an equivalent grammar 'G', from the CFG, G, such that each symbol appears in a sentential form.

Derivation procedure:-

Step 1-1

Include the start symbol in Y_1 & initialize $i=1$

Step 1-2

Include all symbols Y_{i+1}, that can be derived from Y_i and include all productions rules have been applied.

Step 1-3

Increment i and repeat step 2, until $Y_{i+1} = Y_i$

Example:

Find a reduced grammer equivalent to the grammar G, having production rules.

P: S → AC | B, A → a, C → c | BC, E → aA | e.

A) phase 1

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

$W_1 = \{A, C, E\}$, ~~$W_2 = \{S, C, E\}$~~

$W_2 = \{A, C, E, S\}$

$W_3 = \{A, C, E, S\}$

$G^1 = \{(A, C, E, S), \{a, c, e\}, P, (S)\}$

P: S → AC, A → a, C → c, E → aA | e.

phase 2

$Y_1 = \{S\}$, $Y_2 = \{S, A, C\}$

$Y_3 = \{S, A, C, a, c\}$

$Y_4 = \{S, A, C, a, c\}$

$G^4 = \{(A, C, S), \{a, c\}, P, \{S\}\}$

P: S → AC, A → a, C → c

classmate
Date
Page

Removal of unit productions

→ Any Production Rule of the form $A \rightarrow B$ where A, B ∈ non Terminals is called unit productions.

procedure for Removal:

step-1

To remove $A \rightarrow B$, add production $A \rightarrow X$ to the grammar rule whenever $B \rightarrow X$ occurs in the grammar. [$X \in$ Terminal, X can be Null]

step-2

Delete $A \rightarrow B$ from grammar.

step-3

Repeat from step 1 until all unit productions are removed.

Example:-

Remove unit productions from the Grammar whose production rule is given by

P: $S \rightarrow XY$, $X \rightarrow a$, $Y \rightarrow Z|b$, $Z \rightarrow M$, $M \rightarrow N$, $N \rightarrow a$

A) $Y \rightarrow Z$, $Z \rightarrow M$, $M \rightarrow N$

1) since $N \rightarrow a$, we add $M \rightarrow a$

P: $S \rightarrow XY$, $X \rightarrow a$, $Y \rightarrow Z|b$, $Z \rightarrow M$, $M \rightarrow a$, $N \rightarrow a$

2) Since $M \rightarrow a$, we add $Z \rightarrow a$

P: $S \rightarrow XY$, $X \rightarrow a$, $Y \rightarrow Z|b$, $Z \rightarrow a$, $M \rightarrow a$, $N \rightarrow a$

3) Since $Z \rightarrow a$, we add $Y \rightarrow a$

P: $S \rightarrow XY$, $X \rightarrow a$, $Y \rightarrow a|b$, $Z \rightarrow a$, $M \rightarrow a$, $N \rightarrow a$

Remove the unreachable symbols.

P: $S \rightarrow XY$, $X \rightarrow a$, $Y \rightarrow a|b$

Removal of NULL productions

→ In a CFG, a Non-Terminal Symbol 'A' is a nullable variable if there is a production $A \to \epsilon$ or there is a derivation that starts at 'A' and leads to ϵ (like $A \to \ldots\ldots \to \epsilon$).

procedure of Removal:

Step 1
→ To remove $A \to \epsilon$, look for all productions whose right side contains A.

Step 2
Replace each occurrence of 'A' in each of these productions with ϵ.

Step 3
Add the resultant productions to the Grammar.

Example:
Remove NULL productions from following grammar
$S \to ABAC$, $A \to aA \mid \epsilon$, $B \to bB \mid \epsilon$, $C \to c$

1) $A \to \epsilon$, $B \to \epsilon$

1) To eliminate $A \to \epsilon$

$S \to ABAC$

$S \to ABEC$

$S \to ABC \mid BAC \mid BC$

$A \to aA$

$A \to a$

New productions: $S \to ABAC \mid ABC \mid BAC \mid BC$

$A \to aA \mid a$

$B \to bB \mid \epsilon$, $C \to c$

2) To eliminate $B \to \epsilon$

$S \to ABAC$

$S \to AAC \mid AC \mid BC$, $B \to b$

New productions: $S \to ABAC \mid ABC \mid BAC \mid BC \mid AAC \mid AC$

$A \to aA \mid a$, $B \to bB \mid b$, $C \to c$

chomsky Normal Form

→ In chomsky Normal Form (CNF) we have restriction on the length of RHS; which is; elements in RHS should either be two variables or a Terminal.

→ A CFG is in chomsky Normal Form if the productions are in following forms:-

$A \longrightarrow a$

$A \longrightarrow BC$

→ where A, B & C are non-terminals and 'a' is a terminal.

Steps to convert a CFG to chomsky Normal Form:-

Step-1

If the start symbol S occurs on some right side, create a new start symbol S' and a new production $S' \longrightarrow S$

Step-2

Remove Null productions. (Using Null production removal in previous page).

Step-3

Remove unit productions. (Using unit production removal in previous page).

Step-4

Replace each production $A \longrightarrow B_1 \ldots B_n$ where $n > 2$, with $A \longrightarrow B_1 C$ where $C \longrightarrow B_2 \ldots B_n$.

→ Repeat this step for all productions having two (or more symbols on right side.

Step-5

If the right side of any production is in the form $A \longrightarrow aB$ where 'a' is a terminal & A, B are non, then the production is replaced by $A \longrightarrow XB$ & $X \longrightarrow a$.

Repeat this step for every production which is in the form $A \longrightarrow aB$.

Example

classmate
Date 17/07/21
Page 58

conversion of CFG to Chomsky normal form.

convert the following CFG to CNF :-

P: S → ASA | aB, A → B|S, B → b|ϵ

A) given

P: S → ASA | aB, A → B|S, B → b|ϵ

1) since S appears in RHS, we add a new state S' and S' → S is added in production.

P: S' → S, S → ASA|aB, A → B|S, B → b|ϵ

2) Remove the Null production : B → ϵ & A → ϵ :

i) To emilate A → ϵ

S' → ASA|aB

S → AS|aB, SA|aB, S|aB,

S' → AS|aB|SA|S, A → B|S, B → b|ϵ

ii) To elimate B → ϵ

S' → AS|aB|SA|S, A → B|S, B → b|ϵ

→ AS|a|SA|AS|S|aB, A → B|S, B → b

∴ After elimnateing A → ϵ

S → S, S' → ASA|aB|a|SA|AS|S, A → B|S, B → b

∴ After elimnating B → ϵ

S → S, S → AS|aB|a, A → B|S|ϵ, B → b

Continue →

3> Remove the unit productions: $S \to S$, $S' \to S$, $A \to B$ & $B \to S$:

After removing $S \to S$:

P: $S' \to S$, $S \to AS|AB|a|SA|AS$, $A \to B|S$, $B \to b$

After removing $S' \to S$:

P: $S' \to AS|AB|a|SA|AS$, $A \to B|S$, $B \to b$

$S \to AS|AB|a|SA|AS$

After removing $A \to B$:

P: $S' \to AS|AB|a|SA|AS$, $A \to b|S$, $B \to b$

$S \to AS|AB|a|SA|AS$

After removing $A \to S$:

P: $S' \to AS|AB|a|SA|AS$, $A \to b|AS|AB|a|SA|AS$, $B \to b$

$S \to AS|AB|a|SA|AS$

4> Now Find out the productions that has more than two variables in Rhs.

$S' \to ASA$, $S \to ASA$ & $A \to ASA$.

After removing these,

P: $S' \to AX|AB|a|SA|AS$

$S \to AX|AB|a|SA|AS$

$A \to b|AX|AB|a|AS|SA$

$B \to b$

$X \to SA$.

5> Now change the productions $S' \to AB$, $S \to AB$. & $A \to AB$

P: $AX|YB|a|SA|AS$

$S \to AX|YB|a|SA|AS$

$A \to b|AX|YB|a|AS|SA$

$B \to b$

$X \to SA$

$Y \to a$

Greibach Normal Form (GNF)

→ A CFG is Greibach Normal Form if productions are in following

$$A \rightarrow b$$

$$A \rightarrow bC_1C_2,\dots,C_n.$$

where $A, C_i, \dots C_n$ are Non-Terminals and b is non-Terminal.

Steps to convert CFG to GNF:

Step 1

Check if the given CFG has any unit productions (or) NULL productions and remove if there any.

Step 2

Check whether CFG is chomsky Normal form (CNF) and convert to CNF if not.

Step 3

Change the names of Non-Terminal symbols into some A_i in ascending order of i.

Example: $S \rightarrow CA \mid BB$
$B \rightarrow b \mid SB$
$C \rightarrow b$
$A \rightarrow a$

Replace S with A_1
C with A_2
A with A_3
B with A_4

we get

$A_1 \rightarrow A_2A_3 \mid A_4A_4$
$BA_4 \rightarrow b \mid A_1A_4$
$A_2 \rightarrow b$
$A_3 \rightarrow a$

classmate
Date
Page

step 4:

Alter the rules so that Non-Terminals are in ascending order, such that, If the production is in form of $A_i \rightarrow A_j X$, then $i<j$ and should never be $i \geq j$

$A_1 \rightarrow A_2A_3 | A_4A_4$ ✓

$i=1, j=2$ (✓) ; $i=1, j=4$ (✓)

$A_4 \rightarrow A_2A_3b | A_1A_4$

$i=4, A_1=1$ (✗)

so.

$A_4 \rightarrow b | A_1A_4$

$A_4 \rightarrow b | A_2A_3A_4 | A_4A_4A_4$

$A_4 \rightarrow b | bA_3A_4 | A_4A_4A_4$

$A_4 \rightarrow b | bA_3A_4 | A_4A_4A_4A_4$

step 5 — left recursion.

Remove left Recursion.

Introduce a New variable to remove left Recursion.

$A_4 \rightarrow b | bA_3A_4 | A_4A_4A_4$

$Z \rightarrow A_4A_4Z | A_4A_4$

$A_4 \rightarrow b | bA_3A_4 | bZ | bA_3A_4Z$

Now grammar is

$A_1 \rightarrow A_2A_3 | A_4A_4$

$A_4 \rightarrow b | bA_3A_4 | bZ | bA_3A_4Z$

$A_2 \rightarrow b$, $Z \rightarrow A_4A_4Z | A_4A_4$

$A_3 \rightarrow a$

NOW

$A_1 \rightarrow bA_3 | bA_4 | bA_3A_4A_4 | bZA_4 | bA_3A_4ZA_4$

$A_4 \rightarrow b | bA_3A_4 | bZ | bA_3A_4Z$

$Z \rightarrow bA_4Z | bA_3A_4A_4Z | bZA_4Z | bA_3A_4ZA_4Z | bA_4 | bA_3A_4A_4 | bZA_4 | bA_3A_4ZA_4$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

pumping lemma (FOR context Free language).

→ pumping lemma (FOR CFL) is used to prove that a language is NOT context Free.

→ If A is a context Free languages, then, A has a pumping length 'p' such that any string 's', where $|s| \geq p$ may be divided into 5 pieces $s = uvxyz$ such that following conditions must true:-

(1) uv^ixy^iz is in A for every $i \geq 0$

(2) $|vy| > 0$

(3) $|vxy| \leq p$

To prove that a language is Not context Free using pumping lemma (FOR CFL) follow steps below (WE prove using CONTRODICTION).

→ Assume that A is context Free.

→ It has to have pumping

→ All strings longer than p can be pumped $|s| \geq p$

→ Now find a string 's' in A such that $|s| \geq p$

→ Divide s into uvxyz

→ Show that $uv^ixy^iz \notin A$ for some i

→ The consider the ways that 's' can be divided into uvxyz

→ Show that none of these can satisfy all the 3 pumping conditions at same time.

→ s cannot be pumped = = CONTRADICTION

pumping lemma (For context Free language) - Example

→ show that $L = \{a^N b^N c^N \mid N \geq 0\}$ is not context Free.

1) → Assume that L is context Free

→ L must have a pumping length (say P)

→ Now we take a string S such that $S = a^P b^P c^P$

→ We divide S into parts u v x y z

Ex: $p=4$, so, $S = a^4 b^4 c^4$

CASE I: v & y each contain only one type of symbol

aaaabbbbcccc
u v x y z

Now (1) $uv^i xy^i z$ $(i=2)$

uv^2xy^2z

aaaaaabbbbccccc $\Rightarrow a^6 b^4 c^5 \notin L$

CASE II

Either v or y has more than one kind of symbols.

aaaabbbbcccc
u v x y z

Now (1) $uv^i xy^i z$ $(i=2)$

uv^2xy^2z

aaaabbaabbbbbcccc $\neq a^N b^N c^N \notin L$

pumping lemma (FOR CFL) – Example (2).

→ show that $L = \{ww \mid w \in \{0,1\}^*\}$ is NOT context Free.

→ Assume that L is context Free.

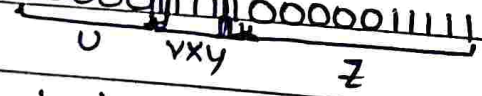
→ L must have pumping length (say p)

→ Now take a string s that $s = 0^p 1^p 0^p 1^p$.

→ Now divide s into uvxyz

Ex: P=5, so, $s = 0^5 1^5 0^5 1^5$

CASE I: vxy does not straddle a boundary



Now $uv^i xy^i z$ (i≥2)

uv^2xy^2z

000001111111100001111

$0^5 1^7 0^5 1^5 \notin L$

∴ L is not context free language.

Pushdown Automata (Introduction)

→ A pushdown Automata (PDA) is a way to implement a context Free Grammar in a similar way we design Finite Automata for Regular Grammar.

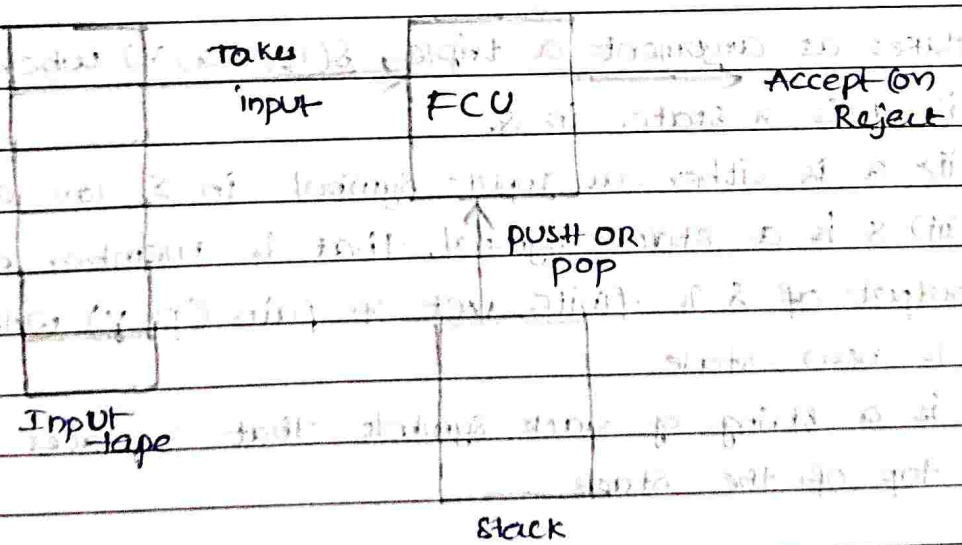
→ It is more powerful than FSM

→ FSM has a very limited memory but PDA has more memory

→ PDA = Finite state Machine + stack

A pushdown Automata has 3 components

1) An input tape
2) A finite Control unit
3) A stack with infinite size



Pushdown Automata (Formal definition)

→ A pushdown Automata is formally defined by 7 Tuples as shown below.

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

where

Q = A finite set of states

Σ = A finite set of Input symbols

Γ = A finite stack Alphabet

δ = The transition function

q_0 = The start state

z_0 = The start stack symbol

F = Final state

δ takes as argument a triple $\delta(q, a, X)$ where:

i) q is a state in Q.

ii) a is either an Input symbol in Σ (or $a = \epsilon$

iii) X is a stack symbol, that is member of Γ

The output of δ is finite set of pairs (p, γ) where:

→ p is new state

→ γ is a string of stack symbols that replaces X at the top of the stack.

Eg:

If $\gamma = \epsilon$ stack is popped

If $\gamma = X$ stack in unchanged

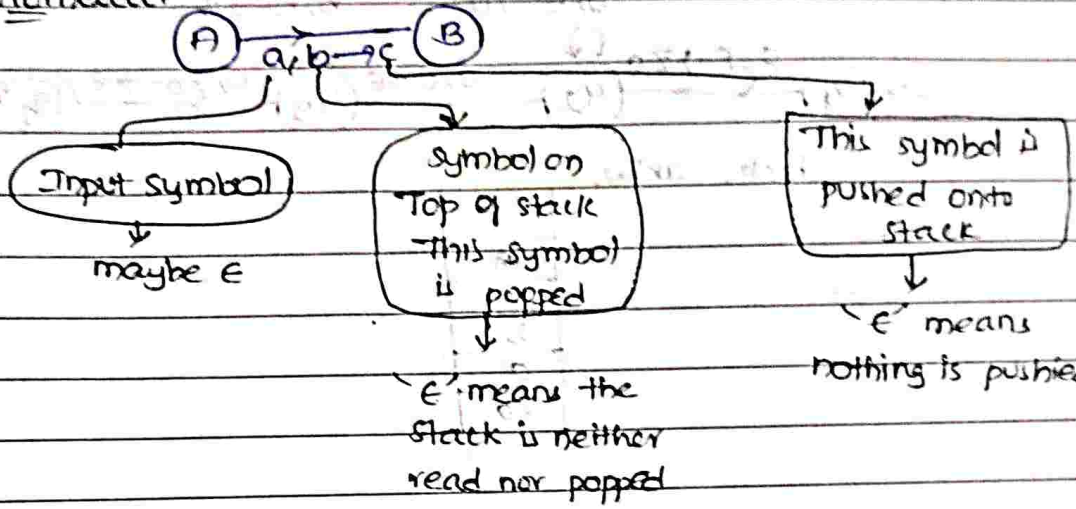
If $\gamma = YZ$ then X is replaced by Z and Y is pushed onto the stack.

classmate
Date
Page

PUSHDOWN AUTOMATA (Graphical Notation)

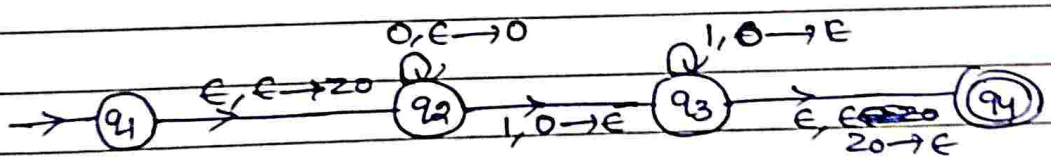
Finite state machine:

pushdown automata:



Example:

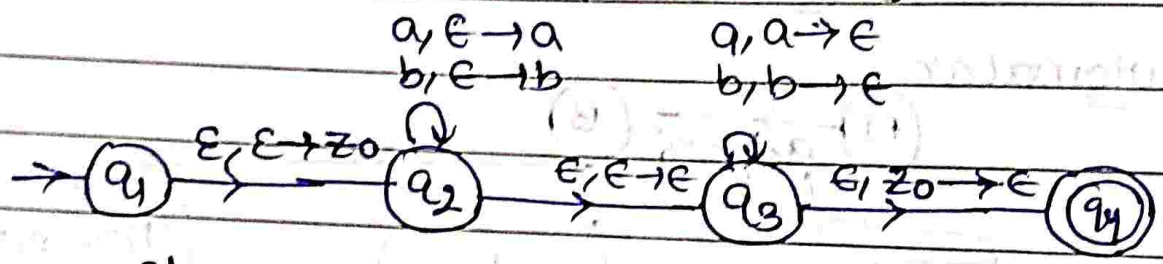
Construct a PDA that accepts $L = \{0^n 1^n \mid n \geq 0\}$



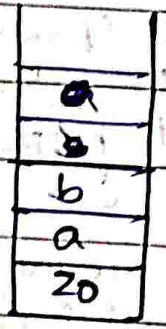
Pushdown Automata - Example (Even palindromes)

→ Construct a PDA that accepts Even palindrome of form

$$L = \{ ww^R \mid w = (a+b)+ \}$$



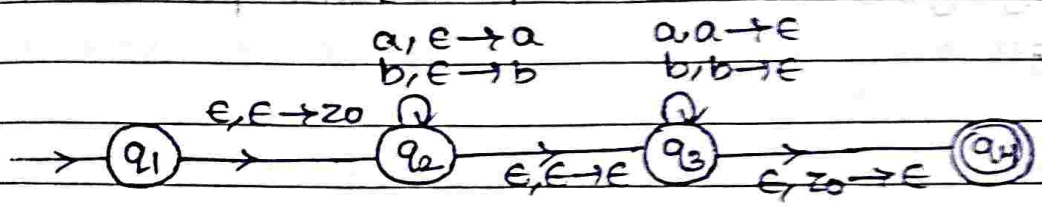
Check abba



Pushdown Automata – Example (Even palindrome) part-2

Construct a PDA that accepts even palindrome of form

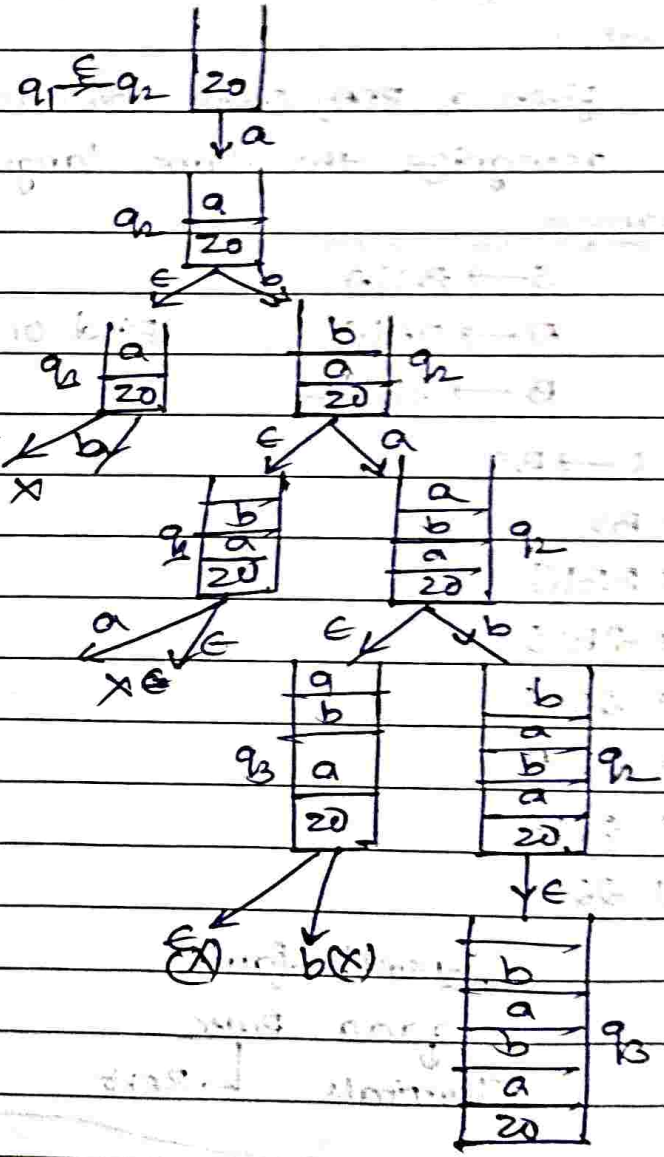
$L = \{ w w^R \mid w = (a+b)^+ \}$



Example:

a b a b

ε a ε b ε a ε b ε



Not accepted

Equivalence of CFG & PDA (part-1)

Theorem:

A language is context free if some pushdown Automata recognizes it.

proof:

part 1:

Given a CFG, show how to construct a PDA that recognize it.

part 2:

Given a PDA, show how to construct a CFG that recognize the same language.

Example

$S \rightarrow BS \mid A$

$A \rightarrow 0A \mid \epsilon$

$B \rightarrow BB1 \mid 2$

Find or build a PDA.

→ S → BS

→ BS

→ BBIS

→ 2BIS

→ 22IS

→ 22IA

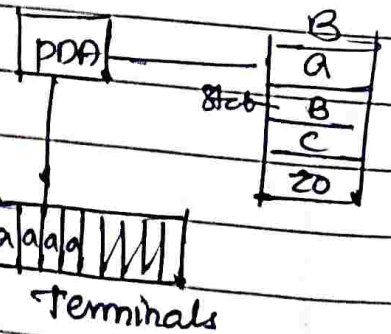
→ 22IE

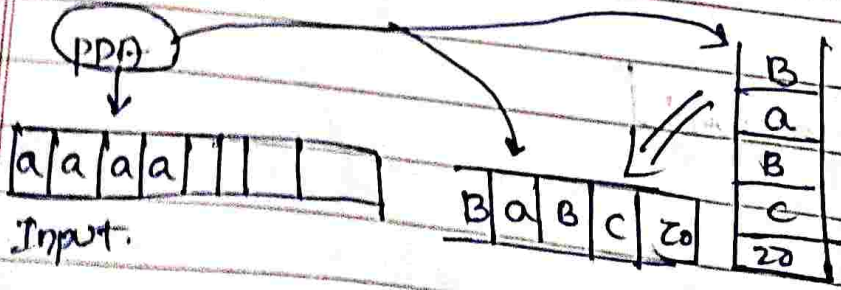
→ 221

General form:

aaaa BaBc

Terminals — Rest





left Most Derivation : S ⟶ ... a a a a | B | a | B | C |

AT Each step Expand left-most derivation

Ex Rule : B ⟶ ASA x BA

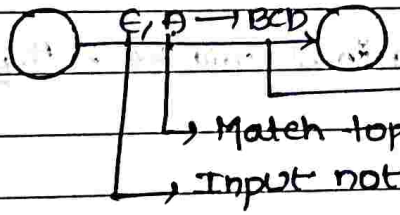
⟶ a a a a | A | S | A | x | B | A | a | B | C |

→ Match stack Top to a Rule

→ pop stack

→ push Right hand side of Rule onto stack.

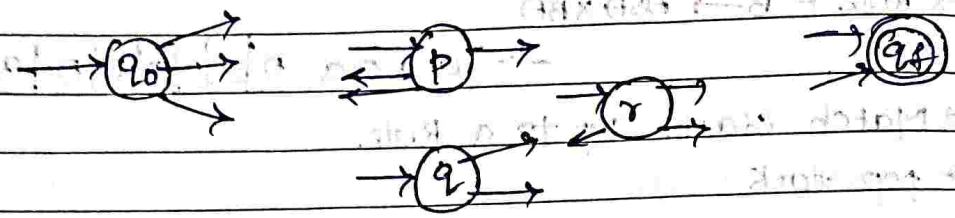
Rule: A ⟶ BCD Add this to the PDA:



classmate
Date
Page

(part-2) :

Given a PDA --> Build a CFG from it.



Step1: Simplify the PDA

Step2: Build the CFG

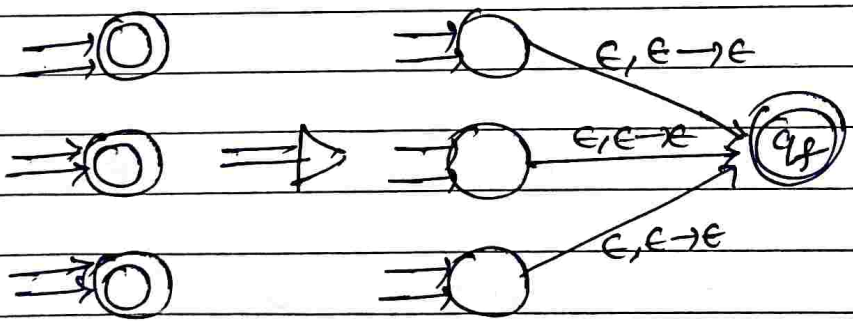
→ There will be non-Terminal for every pair of state.

$A_{pq}, A_{qr}, A_{rq_0}, \ldots$

→ The starting Non-Terminal will be : $A_{q_0 q_f}$

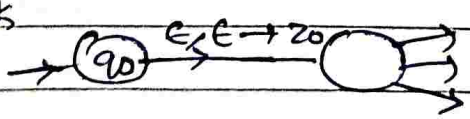
Simplifying the PDA:

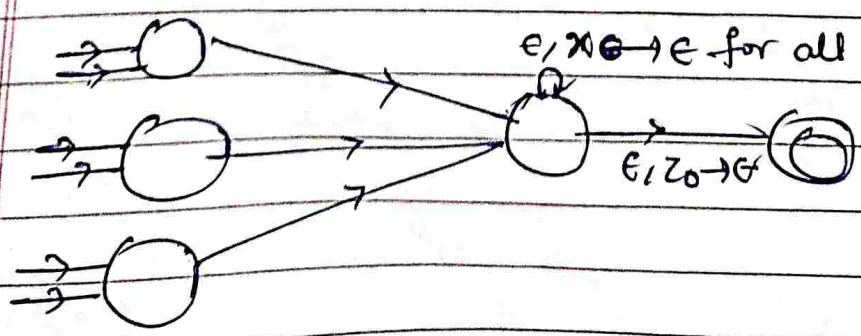
1) The PDA should have only one Final/accept state



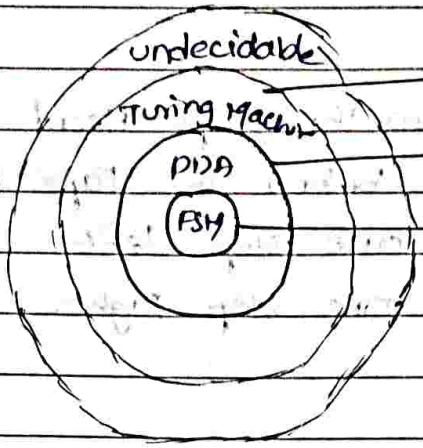
2) The PDA should empty its stacks before accepting

→ Create a new start state q_0 which pushes z_0 to the stack





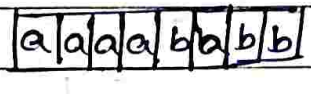
Turning machine – Intro (part-1)



→ Recursively Enumerable languages.
→ content free languages
→ Regular languages

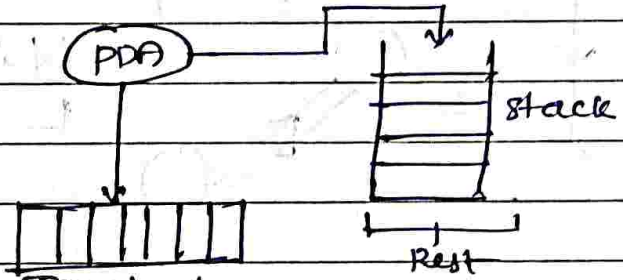
Data structures:

FSM: The input string



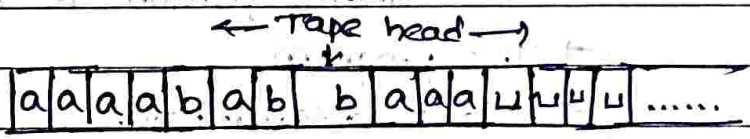
PDA:

→ The input string
→ Stack



Turing machine:

→ A Tape

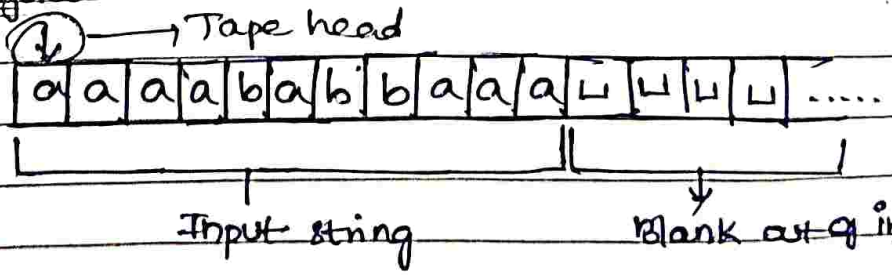


Tape Alphabet: $\Sigma = \{0, 1, a, b, X, Z_0\}$

The Blank ⊔ is a special symbol $\sqcup \notin \Sigma$

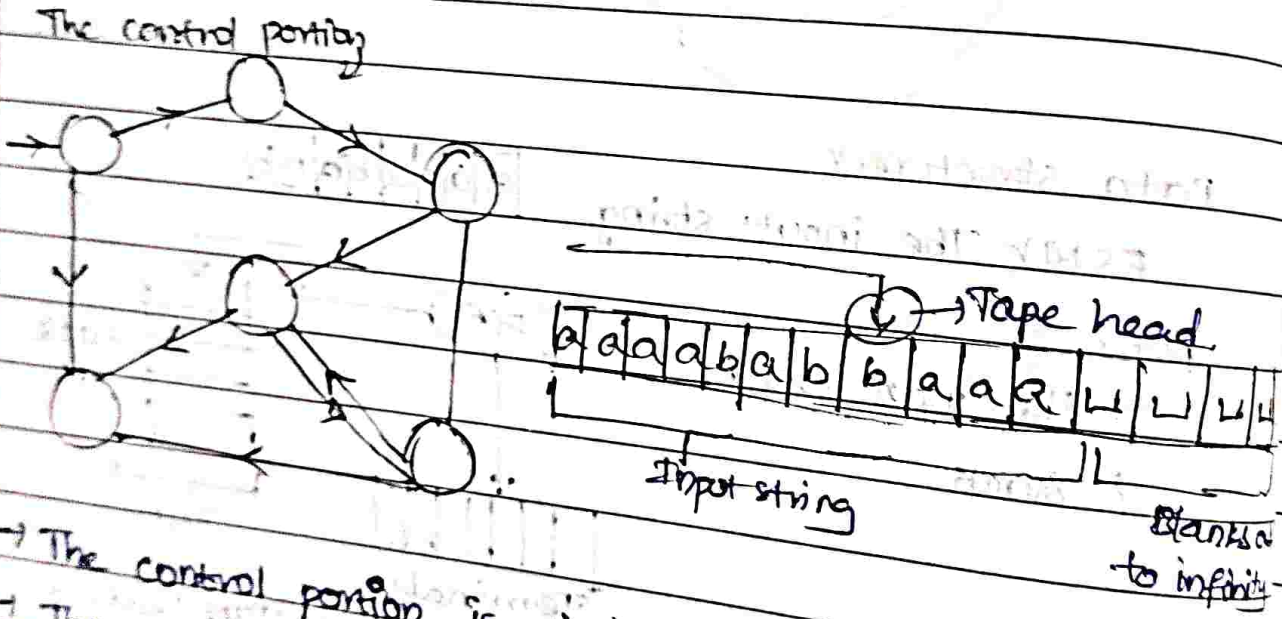
The blank is a special symbol used to fill infinite tape.

Initial configuration:



operations on the Tape

→ Read / Scan symbol below the Tape Head
→ update / write a symbol below the Tape Head
→ Move the Tape Head one step left
→ Move the Tape Head one step Right.



→ The control portion is similar to FSM or PDA
→ The program
→ It is deterministic.

Rules of operations

Rule of operation: 1

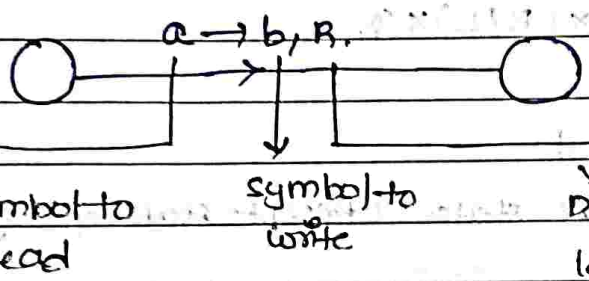
At each step of the computation.

→ Read the current symbol

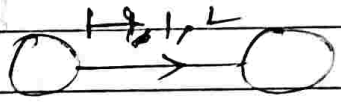
→ update (i.e write) the same cell.

→ Move exactly one cell either left or RIGHT.

*) If we are at left end of tape, and trying to move left, then do not move. stay at left end.



If you dont want to update the cell just write the same symbol.



Rule of operation: 2

→ control is with a sort of FSM

→ Initial state.

→ Final states: (there are two final states)

1) The Final state

2) The Reject state.

→ computation can either

1) HALT & ACCEPT

2) HALT & REJECT

3) LOOP (the machine fails to HALT).

Turing machine - (Formal definition).

A Turing machine can be defined as a set of 7 tuples $(Q, \Sigma, \Gamma, \delta, q_0, b, F)$.

Q – Non empty set of states.
Σ – Non empty set of symbols.
Γ – Non empty set of Tape symbols.
δ – Transition function defined as
$Q \times \Sigma \rightarrow \Gamma \times (R/L) \times Q$.
q_0 – Initial state.
b – Blank symbol
F – set of Final states (Accept state & Reject state)

Thus, the production rule of Turing machine will be as
$\delta(q_0, a) \longrightarrow (q_1, Y, R)$.

Turing Thesis

→ Turing's Thesis states that any computation that can be carried out by mechanical means can be performed by some Turing Machine.

Few arguments for accepting this thesis are

1) Anything that can be done on existing digital computer can also be done by Turing machine

2) If there is a problem that you have and if you are able to write an algorithm for solving that problem, then you will also able to design a Turing machine program which solve the same problem.

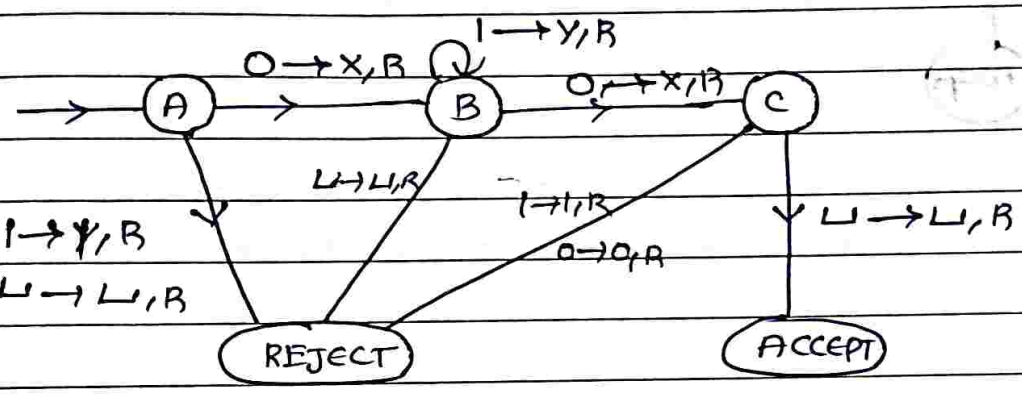
Recursively Enumerable Languages.

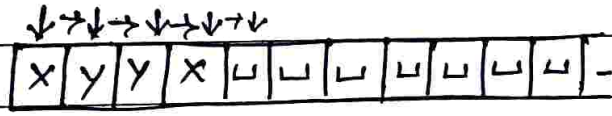
A language L and Σ is said to be Recursively Enumerable if there exists a Turing Machine that accepts it

Turning Machine - Example (part-1).

Design a Turning Machine which recognizes the language $L = 01^*0$.

A)





Tape.

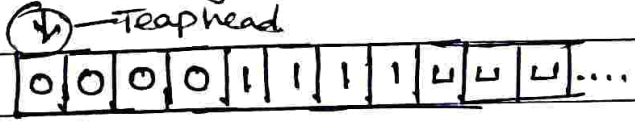
Eg. 0110.

--- Infinity tape.

Example - 2:

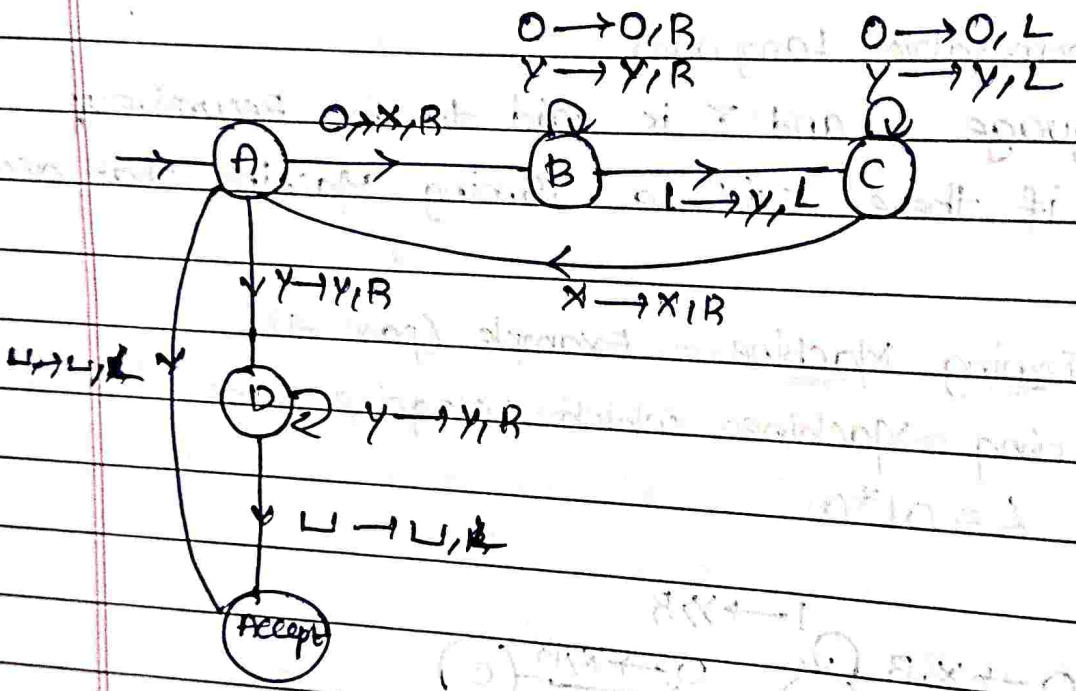
Design a Turning Machine which recognizes the language $L = 0^N 1^N$.

A)



Algorithm:

* change "0" to "x"
* Move RIGHT TO FIRST "1"
  If none r REJECT
* change "1" to "y"
* Move LEFT to leftmost "0"
* Repeat the above steps until no more "0's
* Make sure no more "1's remain.


0 → 0, R
Y → Y, R
0 → 0, L
Y → Y, L
0 → X, R
A
B
C
1 → Y, L
Y → Y, R
X → X, R
D
Y → Y, R
U → U, R
Accept

The CHURCH – TURING Thesis

→ what does ~~countable~~ computable mean?

Alonzo church – LAMBDA CALCULUS

Allen Turning – Turing machine

Several variations of Turning Machines

→ One Tape (or) many

→ Infinite on both ends

→ Alphabets: only {0,1} or more?

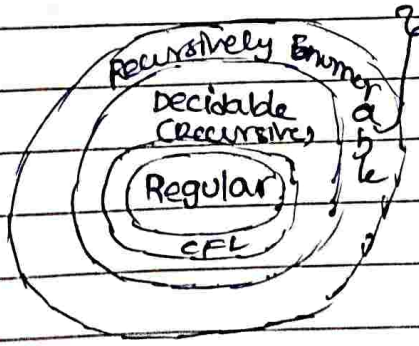
→ can the head also stay in the same place?

→ Allow non-Determinism

* All these variations are equivalent (equal) in computing capability.

Turing machine & lambda calculus are also equivalent powers.

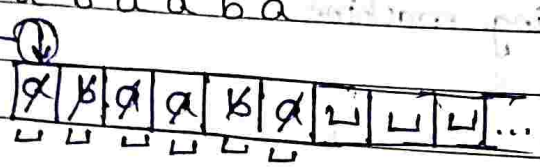
The different classes of languages

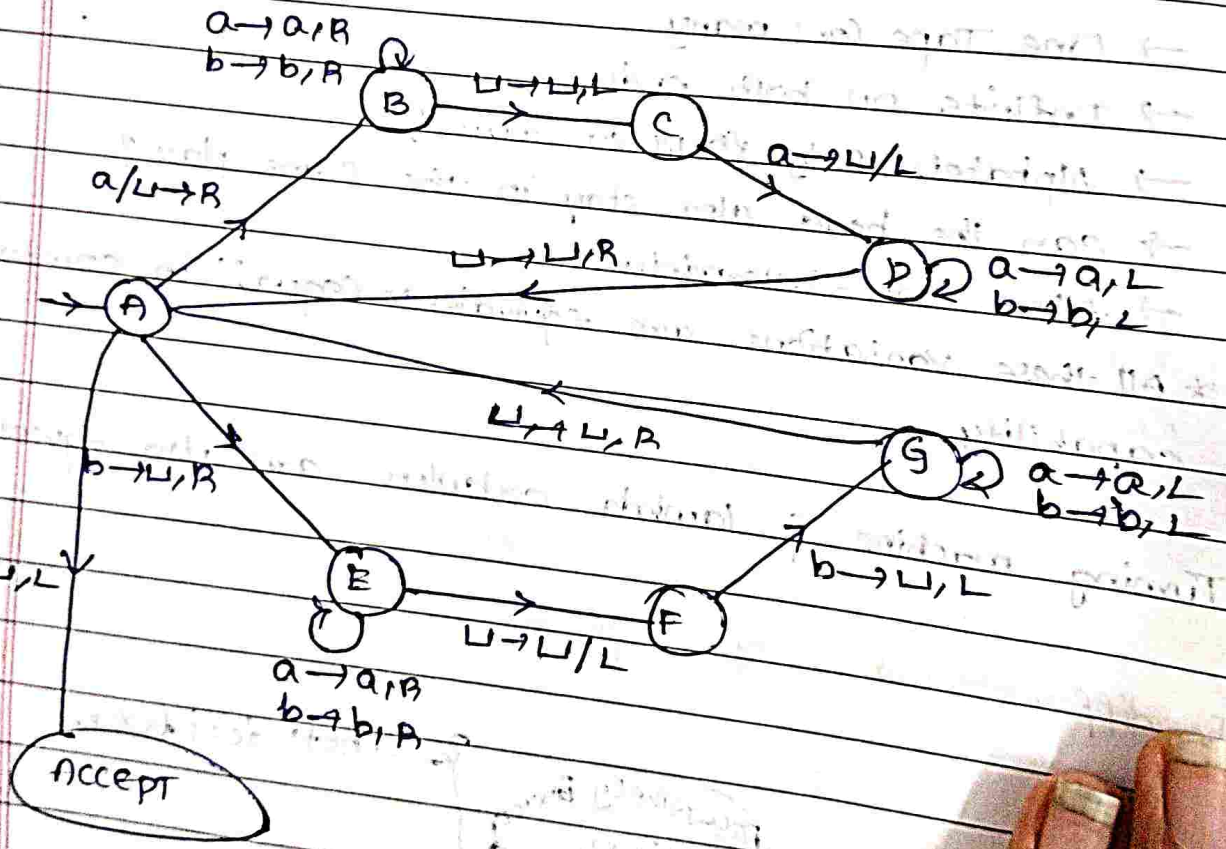


Turing Machine for even palindromes

Design a Turing Machine that accepts Even palindromes over the alphabet $\Sigma = \{a, b\}$.

Ex: a b a a b a





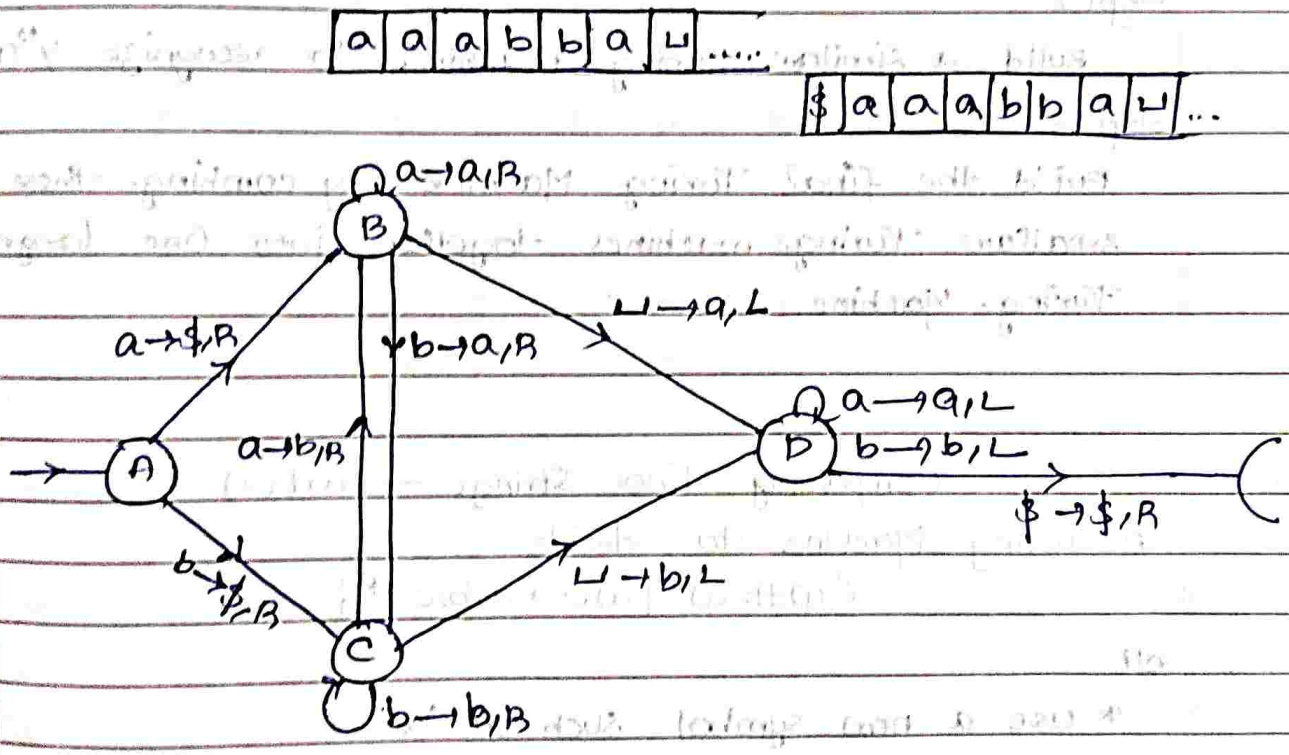
Turing machine programming Techniques (1)

problem:

How can we recognize the left end of the Tape of a Turing machine.

solution:

put a special symbol $ on the left end of the Tape and shift the input over one cell to the right.



Example – 2 (part-2)

Build a Turing Machine to recognize the language

$0^N 1^N 0^N$

Step-1

00000 11111 00000

xxxxx yyyyy 00000

Step-2

Build a similar Turing machine to recognize $y^N 0^N$

Step-3

Build the final Turing Machine by combing these two smaller Turing machines together, into one larger Turing Machine.

Comparing Two Strings – part(2)

A Turing Machine to decide

$\{w\#w \mid w \in \{a,b,c\}^*\}$

Soln

* use a new symbol such as 'x'
* Replace each symbol into an x after it has been examined.

Eg: a b b a c # a b b a c

x b b a c # x b b a c

x x b a c # x x b a c

x x x a c # x x x a c

x x x x c # x x x x c

x x x x x # x x x x x

problem:

can we do it non-destructively? i.e., without loosing the original strings?

sol:

Replace each unique symbol with another unique symbol instead of replacing all with the same symbol.

Ex: a → p, b → q, c → r

a b b a c # a b b a c

p b b a c # p b b a c

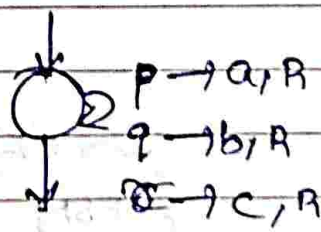
p q b a c # p q b a c

p q q a c # p q q a c

p q q p c # p q q p c

p q q p r # p q q p r

Restore the original string if required.

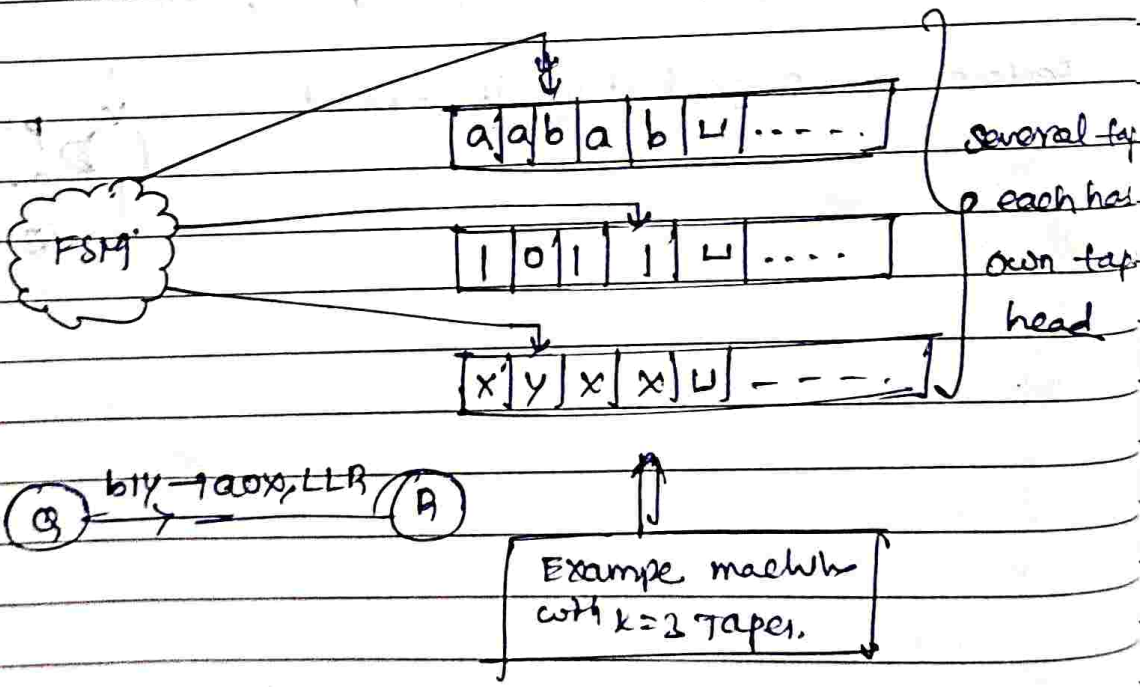


Multitape Turing Machine

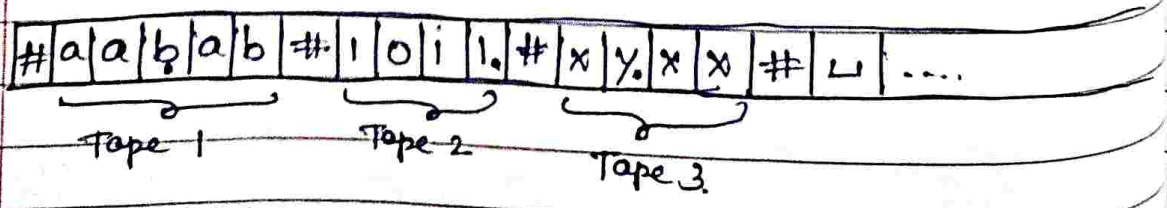
Theorem:
Every multitape Turing machine has an equivalent single Tape Turing Machine.

proof:
Given a Multitape Turing Machine, show how to build a single tape Turing Machine.

1. • Need to store all tapes on single tape.
   • show data representation.
2. Each tape has a tape head.
   show how to store that info.
3. Need to transform a move in the multitape TM into one or more in single tape TM



single tape TM:



* Add dot (•) to each tape in single tape head.

* To simulate a transition from state q_i, we must scan our Tape to see which symbols are under the k Tape heads.
* Once we determine this ε_p are ready to make the transition, we must scan across the tape again to update the cells and move the dots.
* whenever one head moves off the right end, we must shift our tape so we can insert (␣).

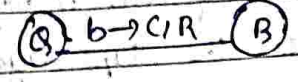
Non-determinism in Turing Machine - (part-1)

Only transition function change in this.

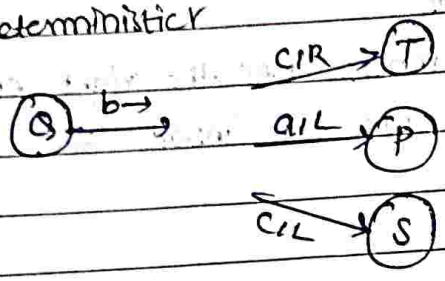
$$\delta : Q \times \Sigma \rightarrow P(\Gamma \times (R/L) \times Q)$$

Ex:-

Deterministic:



non-deterministic:



Configuration:

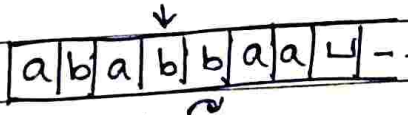
→ A way to represent the entire state of TM at a movement during computation

→ A string which capturers

→ The current state.

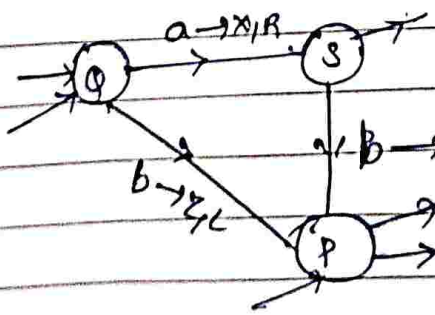
→ The current position of head

→ The entire tape contents.

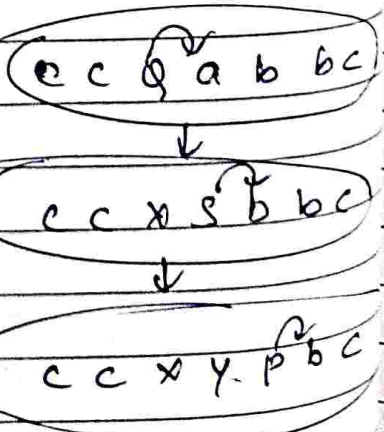


a b a Q b b a a

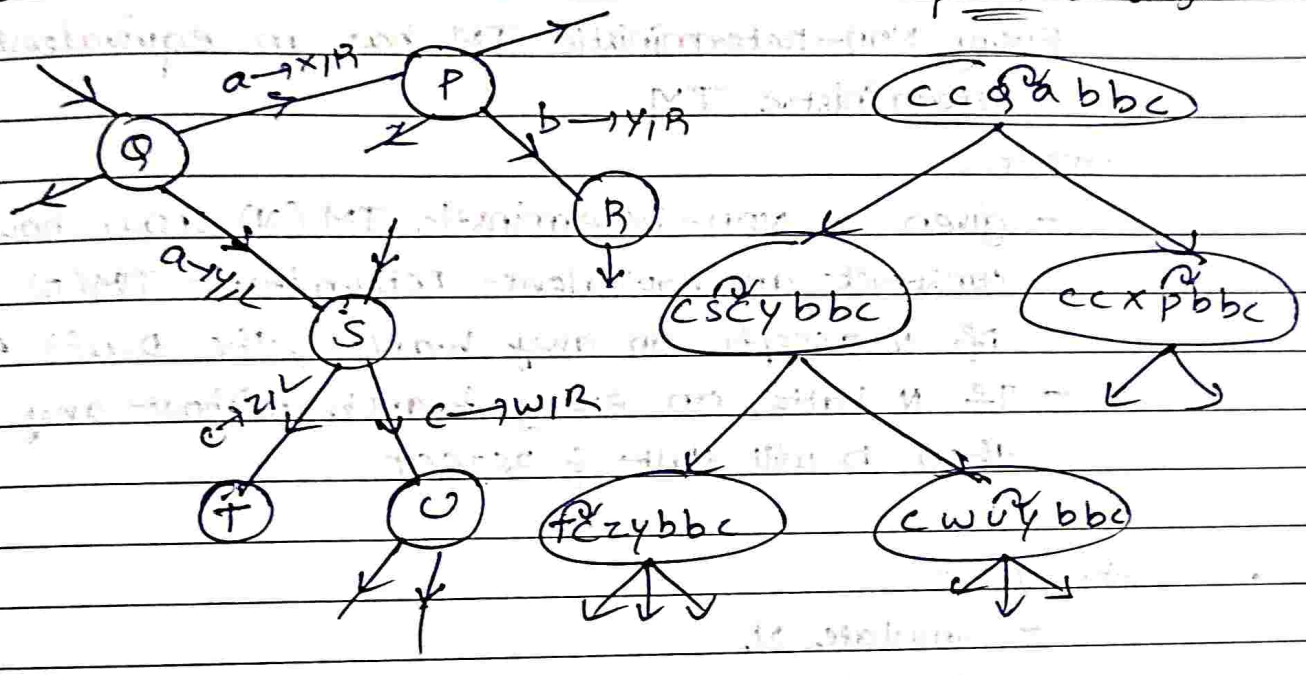
Deterministic TM:



← History of it



Non-Deterministic TM:-



outcomes of Non-Deterministic computation:

Accept:- If any branch of computation accepts, then the non-deterministic TM will accept.

Reject:- If all blocks of computation halt or reject then non-deterministic TM will reject.

Loop:- Accept cannot be done of Reject also.

Theorem:

Every Non-Deterministic TM has an equivalent Deterministic TM.

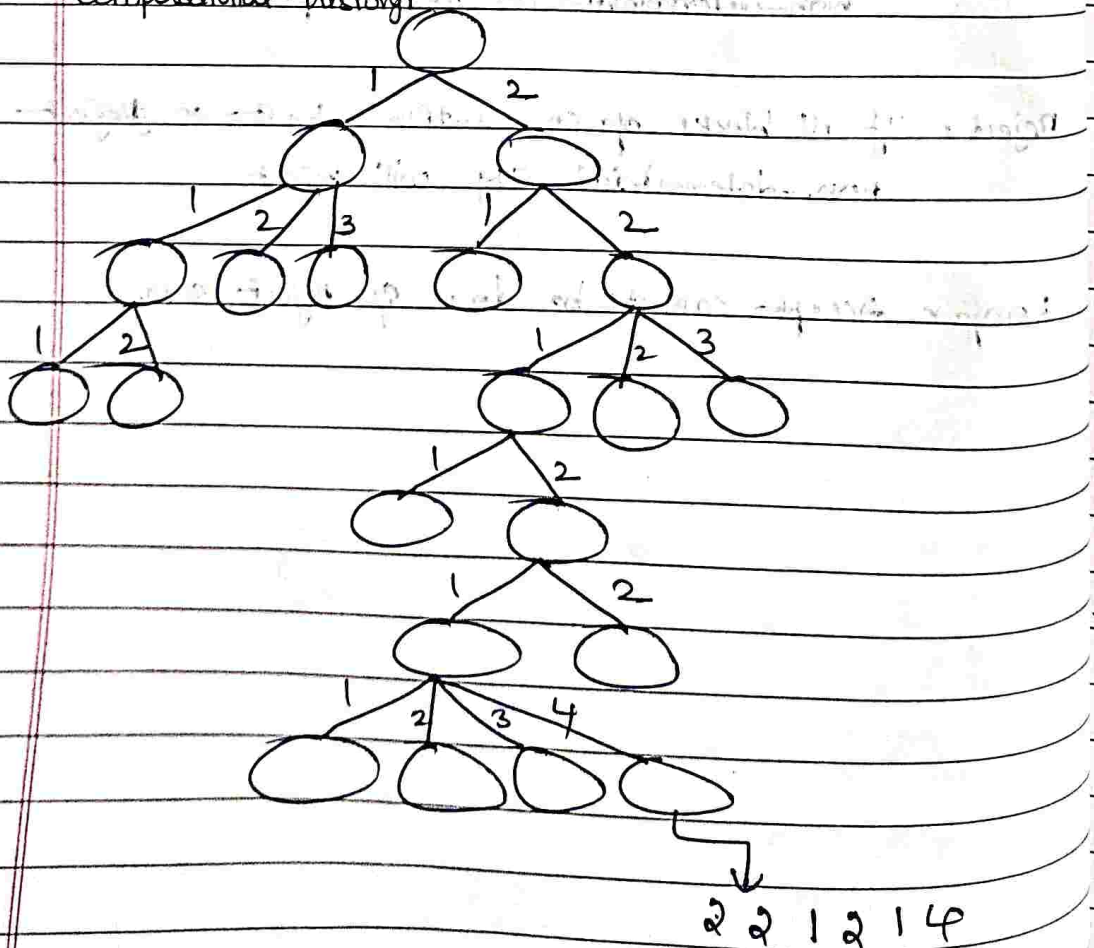
proof:

- given a Non-Deterministic TM (N) show how to construct an equivalent Deterministic TM (D).
- If N accepts on any branch, the D will Accept
- If N halts on every branch without any accept then D will Halt & REJECT

Approach:

- simulate N.
- simulate all branches of computation.
- Search for any way N can Accept.

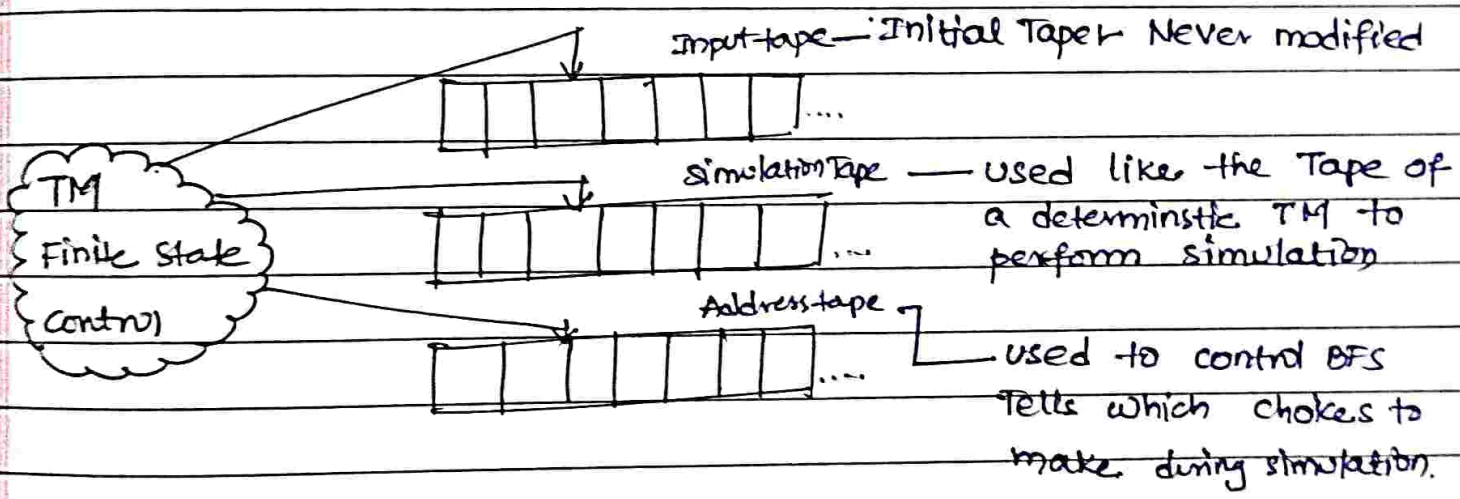
computational history:



→ A path to any node is given by a number
→ Search the tree looking for looking.

Search Order:
- DFS ✗
- BFS ✓

To examine a node:
- perform the entire computation from scratch.
- The path number tells which of many non-deterministic choices to make.

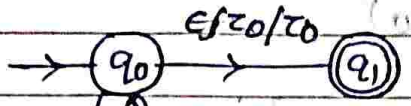


Module - 4

Part - A

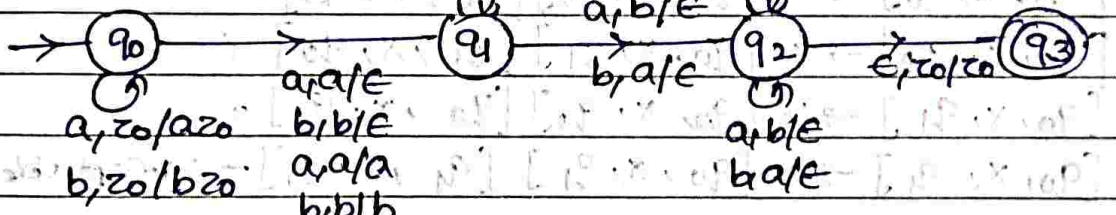
1) Construct PDA for equal number of x's & y's. eg: xyyxxy.

2) PDA for equal no. of x's & y's. (x = a, y = b)



a, z0/az0
b, z0/bz0
a, a/aa , b, b/bb
a, b/ε , b, a/ε

2) $L = \{ w \neq w^R / w \in (x+y)^* \}$



a, z0/az0
b, z0/bz0
a, a/aa
b, b/bb
a, b/ab
b, a/ba

3) Convert the following PDA to CFG

$\delta(q_0, 0, z_0) = (q_0, xz_0)$, $\delta(q_0, 0, x) = (q_0, xx)$, $\delta(q_0, 1, x) = (q_1, \epsilon)$
$\delta(q_1, 1, x) = (q_1, \epsilon)$, $\delta(q_1, \epsilon, x) = (q_1, \epsilon)$,
$\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$.

A) given that

$\delta(q_0, 0, z_0) = (q_0, xz_0)$
$\delta(q_0, 0, x) = (q_0, xx)$
$\delta(q_0, 1, x) = (q_1, \epsilon)$
$\delta(q_1, 1, x) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, x) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$.

4) i) $S \rightarrow (q_0, z_0, q_0)$

$S \rightarrow (q_0, z_0, q_1)$

ii) $\delta(q_0, \overset{0}{0}, \overset{I_0}{z_0}) = (q_0, xz_0)$

$[q_0, z_0, q_0] = 0[q_0, x, q_0][q_0, z_0, q_0]$

$[q_0, z_0, q_0] = 0[q_0, x, q_1][q_1, z_0, q_0]$

$[q_0, z_0, q_1] = 0[q_0, x, q_0][q_0, z_0, q_1]$

$[q_0, z_0, q_1] = 0[q_0, x, q_1][q_1, z_0, q_1]$ - acceptable.

iii) $\delta(q_0, 0, x) = (q_0, xx)$

$[q_0, x, q_0] \rightarrow 0[q_0, x, q_0][q_0, x, q_0]$

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

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

$[q_0, x, q_1] \rightarrow 0[q_0, x, q_1][q_1, x, q_1]$ - acceptable.

iv) $\delta(q_0, 1, x) = (q_1, \epsilon)$

$[q_0, x, q_1] \rightarrow 1$

v) $\delta(q_1, 1, x) = (q_1, \epsilon)$

$[q_1, x, q_1] \rightarrow 1$

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

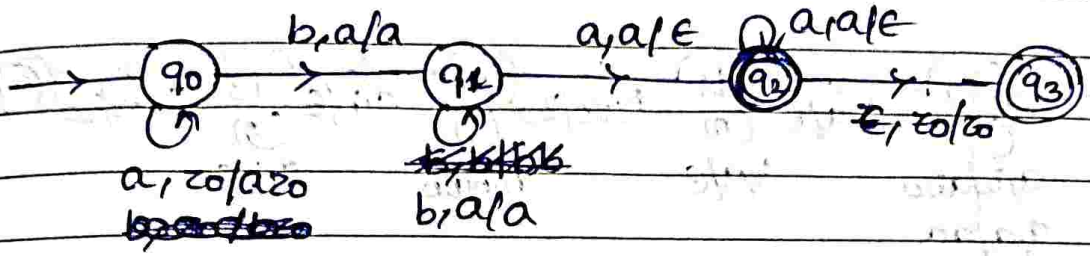
$[q_1, z_0, q_1] \rightarrow \epsilon$

vii) $\delta(q_1, \epsilon, x) = (q_1, \epsilon)$

$[q_1, x, q_1] \rightarrow \epsilon$

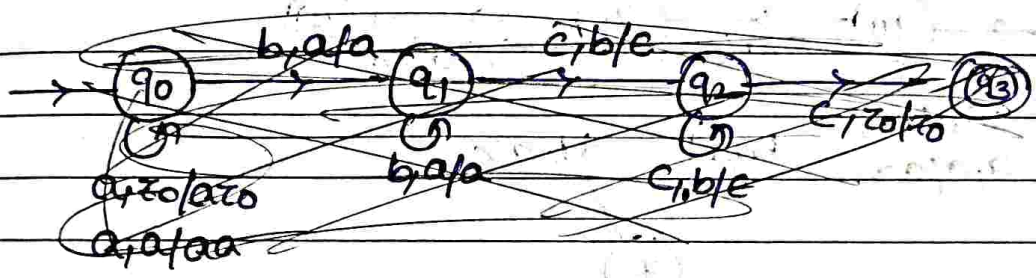
i) $\{a^n b^m a^n \mid m, n \in N\}$

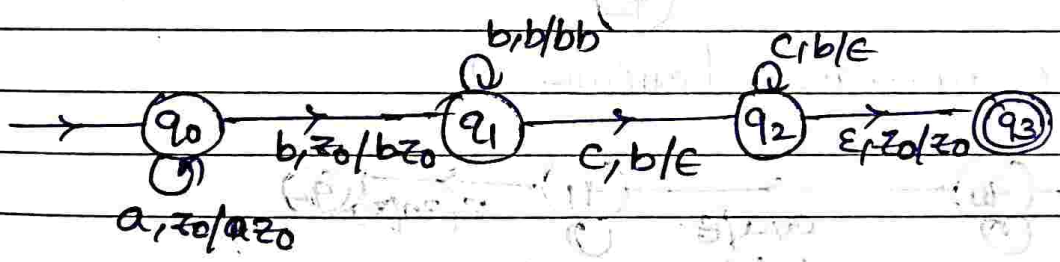
$L = \{a^n b^m a^n \mid m, n \in N\}$



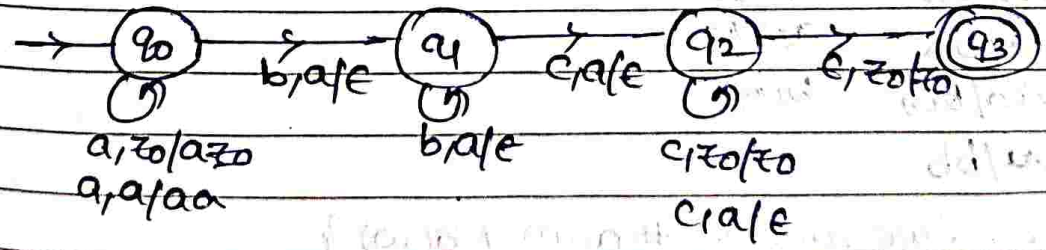
ii) $\{a^n b^m c^m / m, n \in N\}$

$L = \{a^n b^m c^m / n, n \in N\}$.

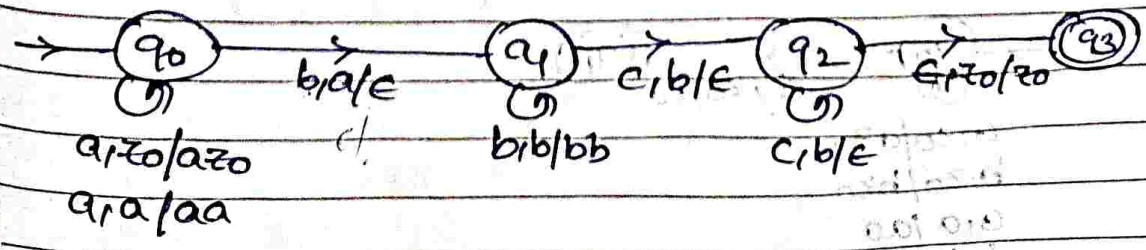




iii) $L = \{a^i b^j c^k \mid i, j, k \in N, i > j\}$

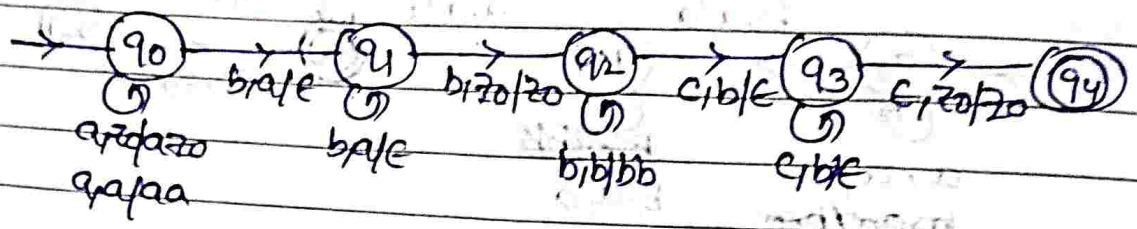


iv) $L = \{a^i b^j c^k \mid i, j, k \in N, i+j=k\}$

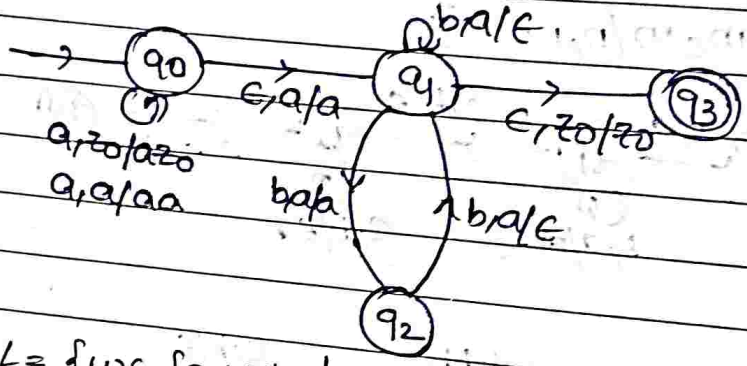


v) $L = \{a^i b^j c^k \mid i,j,k \in N,\ i+k=j\}$

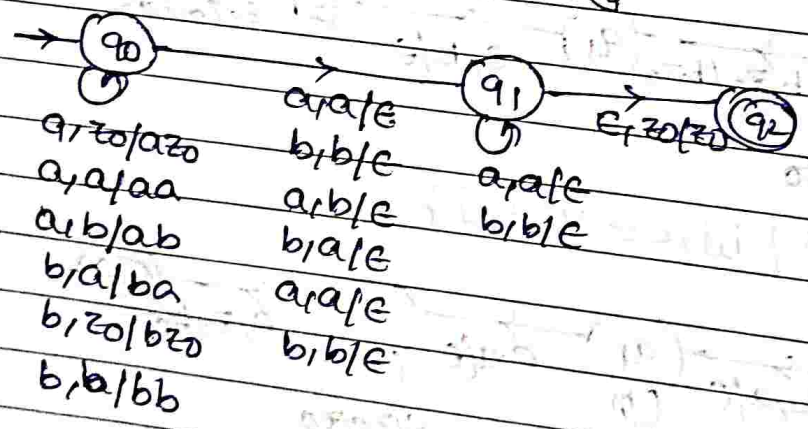
$L = \{a^i b^j c^k \mid i,j,k \in N,\ i+k=j\}$



vi) $L = \{a^n b^m \mid n \le m \le 2n\}$

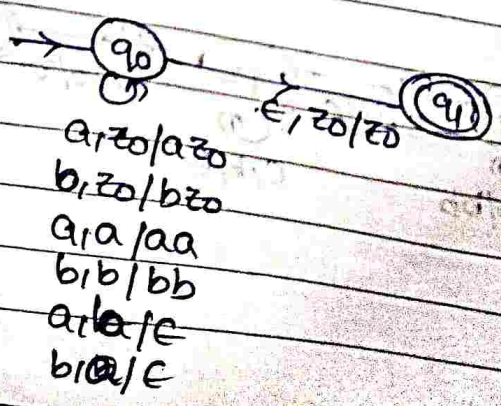


vii) $L = \{w \in \{a,b\}^* \mid n_a(w) = n_b(w)\}$

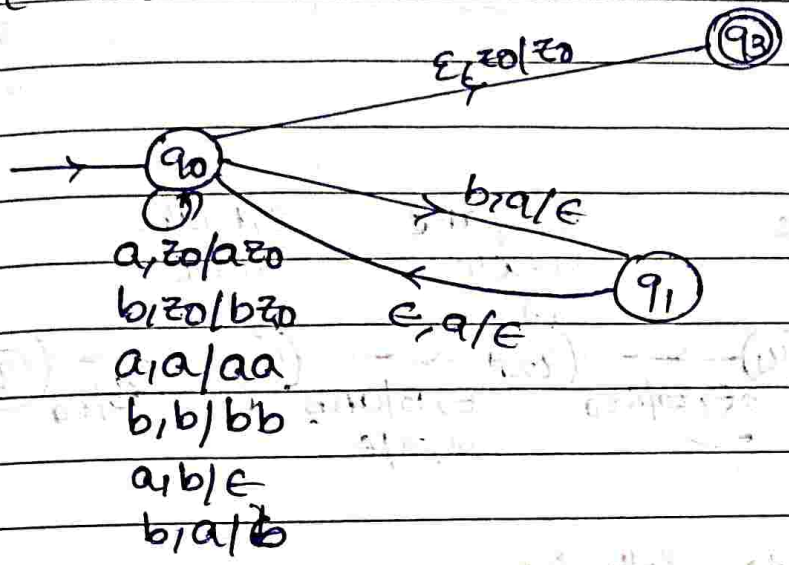


viii) $L = \{w \in \{a,b\}^* \mid \#a(w) \neq b(w)\}$

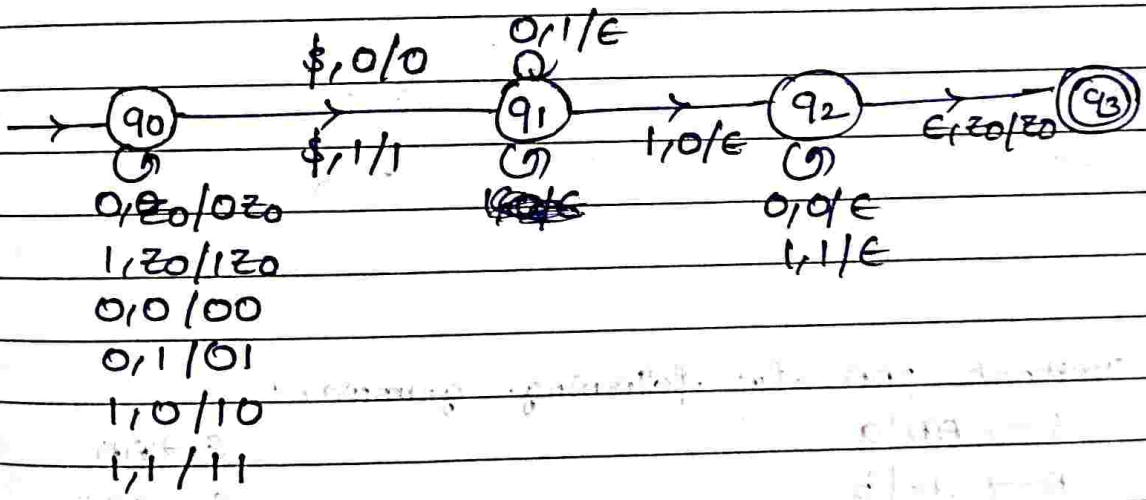
$L = \{w \in \{a,b\}^* \mid \#a(w) \neq b(w)\}$



iv) $\{w \in \{a,b\}^* \mid \#a(w) = 2 \neq b(w)\}$



6) $L = \{ bin(i) \$ mir(bin(i+1)) \mid i \geq 0, \in \{0,1\}^* \}$



7) construct CFG corresponding to PDA whose transition mapping is

$\delta(S,a,X) = (S,A,X)$

$\delta(S,b,A) = (S,AA)$

$\delta(S,a,A) = (S,AA)$

given $\delta(S,a,X) = (S,A,X)$

$\delta(S,b,A) = (S,AA)$

$\delta(S,a,A) = (S,AA)$.

$X \longrightarrow aAX$

$A \longrightarrow bAA$

$A \longrightarrow aAA$

$[S,X,S] = a[S,A,S][S,X,S]$

$[S,A,S] = b[S,A,S][S,A,S]$

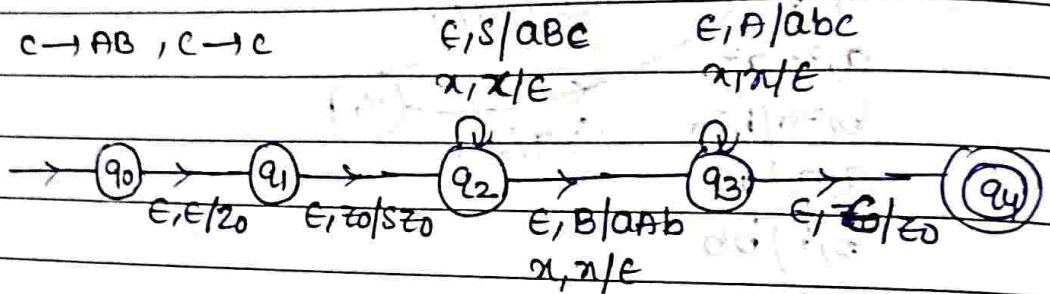
$[S,A,S] = a[S,A,S][S,A,S]$

8) show that given CFG with following productions.

$S \to aBc$
$A \to abc$
$B \to aAb$
$C \to AB$, $C \to c$

$S \to aBc$
$S \to aaAbc$
$S \to aaabcbc$

A)



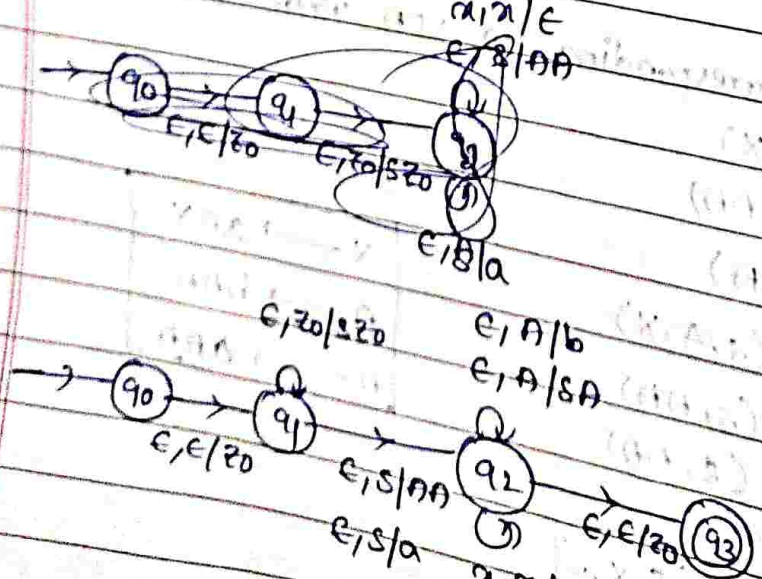
9) construct PDA for following.

$S \to 0A$, $A \to 0AB$, $B \to 1$.

10) construct PDA for following grammar.

$S \to AA \mid a$
$A \to SA \mid b$

$S \to AA$
$S \to SAA$
$\to SAb$
$S \to abb$



classmate
Date
Page

Part-B

2) Describe the grammar for following PDA.

$M=(\{q_0,q_1\},\{0,1\},\{X,z_0\},q_0,z_0,\phi)$ where δ is given by.

$\delta(q_0,0,z_0)=\{(q_0,Xz_0)\}$

$\delta(q_0,0,X)=\{(q_0,XX)\}$

$\delta(q_0,1,X)=\{(q_1,e)\}$

$\delta(q_1,1,X)=\{(q_1,e)\}$

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

$\delta(q_1,e,z_0)=\{(q_1,e)\}$

A) given transistions are.

$\delta(q_0,0,z_0)=\{(q_0,Xz_0)\}$

$\delta(q_0,0,X)=\{(q_0,XX)\}$

$\delta(q_0,1,X)=\{(q_1,e)\}$

$\delta(q_1,1,X)=\{(q_1,e)\}$

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

$\delta(q_1,e,z_0)=\{(q_1,e)\}$

Now $S \longrightarrow [q_0,z_0,q_0]$, $S \longrightarrow [q_0,z_0,q_1]$ ✓

$\delta(q_0,0,z_0)=\{(q_0,Xz_0)\}$

$[q_0,z_0,q_0]=0[q_0,X,q_0][q_0,z_0,q_0]$ ×

$[q_0,z_0,q_0]=0[q_0,X,q_1][q_1,z_0,q_0]$ ×

$[q_0,z_0,q_1]=0[q_0,X,q_0][q_0,z_0,q_1]$ ×

$[q_0,z_0,q_1]=0[q_0,X,q_1][q_1,z_0,q_1]$ ✓

For $\delta(q_0,0,X)=\{(q_0,XX)\}$

$[q_0,X,q_0]=0[q_0,X,q_0][q_0,X,q_0]$ ×

$[q_0,X,q_0]=0[q_0,X,q_1][q_1,X,q_0]$ ×

$[q_0,X,q_1]=0[q_0,X,q_0][q_0,X,q_1]$ ×

$[q_0,X,q_1]=0[q_0,X,q_1][q_1,X,q_1]$. ✓

Now take $\delta(q_0,1,X)=\{(q_1,e)\}$

$[q_0,X,q_1] \rightarrow 1$

Now take. $\delta(q_1, 1, X) = \{(q_1, e)\}$

$[q_1, X, q_1] \rightarrow 1$.

Now take $\delta(q_1, e, X) = \{(q_1, e)\}$

$[q_1, X, q_1] \rightarrow e$

Now take $\delta(q_1, e, z_0) = \{(q_1, e)\}$

$[q_1, z_0, q_1] \rightarrow e$.

Finally

$S \rightarrow [q_0, z_0, q_1]$

$[q_0, z_0, q_1] = 0[q_0, X, q_1][q_1, z_0, q_1]$

$[q_0, X, q_1] = 0[q_0, X, q_1][q_1, X, q_1]$

let

$[q_0, z_0, q_1] = A$, $[q_0, X, q_1] = B$, $[q_1, z_0, q_1] = C$

$[q_1, X, q_1] = D$.

So,

$S \rightarrow A$

$A \rightarrow OBC$

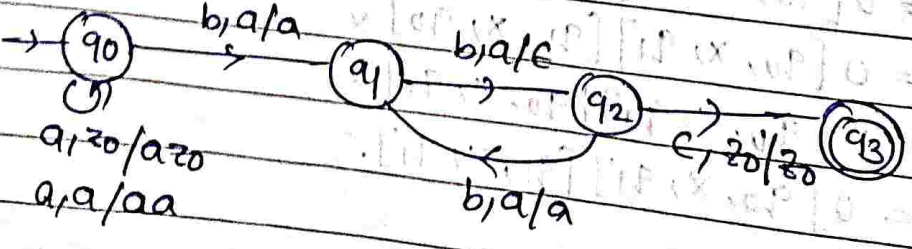
$B \rightarrow OBD / 1$

$D \rightarrow 1 / e$

$C \rightarrow e$.

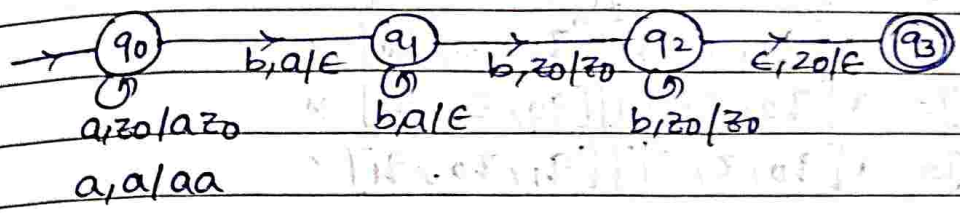
3) Describe PDA for string of form $a^n b^{2n}$

$a^n b^{2n}$, abb, aabbbb



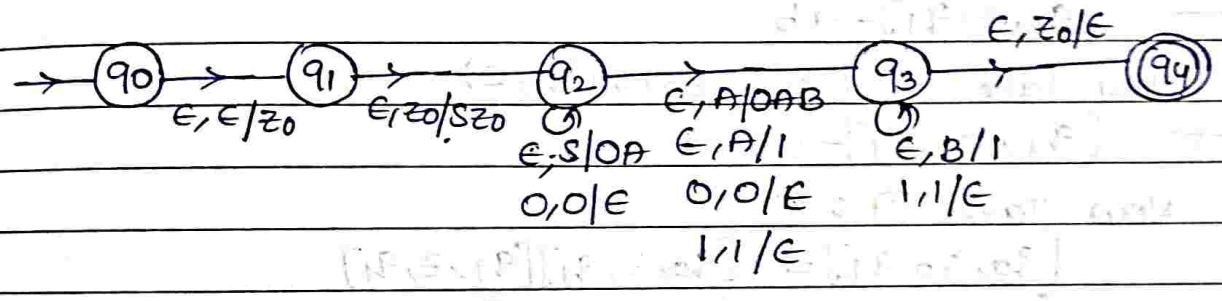
5) Describe PDA for language that accepts $\{a^m b^n / n > m\}$

$a^m b^n / n > m$ ex: a^1b^2, a^1b^3, a^2b^3, a^2, b^4



6) Describe PDA for following grammar $S \to 0A$, $A \to 0AB/1$, $B \to 1$

$S \to 0A$, $A \to 0AB/1$, $B \to 1$



7) convert following PDA to CFG.

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

$\delta(q_0, a, z_0) = (q_0, zz)$, $\delta(q_0, a, z) = (q_0, zz_0)$, $\delta(q_0, b, z) = (q_1, \epsilon)$

$\delta(q_1, b, z) = (q_1, \epsilon)$, $\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$.

A) given δ transistions are.

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

$\delta(q_0, a, z) = (q_0, zz_0)$

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

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

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

Note:

$S \to [q_0, z_0, q_0]$ ✗

$S \to [q_0, z_0, q_1]$ ✓

Now Take. $\delta(q_0, a, z_0) = (q_0, zz)$

$[q_0, z_0, q_0] = a[q_0, z, q_0][q_0, z, q_0]$ ✗

$[q_0, z_0, q_0] = a[q_0, z, q_1][q_1, z, q_0]$ ✗

$[q_0, z_0, q_1] = a[q_0, z, q_0][q_0, z, q_1]$ ✗

(1) $[q_0, z_0, q_1] = a[q_0, z, q_1][q_1, z, q_1]$ ✓

Now Take. $\delta(q_0, a, z) = (q_0, z z_0)$

$[q_0, z, q_0] = a[q_0, z, q_0][q_0, z_0, q_0]$ ✗

$[q_0, z, q_0] = a[q_0, z, q_1][q_1, z_0, q_0]$ ✗

$[q_0, z, q_1] = a[q_0, z, q_0][q_0, z_0, q_1]$ ✗

② $[q_0, z, q_1] = a[q_0, z, q_1][q_1, z_0, q_1]$ ✓

Now Take. $\delta(q_0, b, z) = (q_1, \epsilon)$

③ $[q_0, z, q_1] \rightarrow b$

Now Take $\delta(q_1, b, z) = (q_1, \epsilon)$.

④ $[q_1, z, q_1] \rightarrow b$

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

⑤ $[q_1, z_0, q_1] \rightarrow \epsilon$

Now Take ① & ②.

$[q_0, z_0, q_1] = a[q_0, z, q_1][q_1, z, q_1]$

$[q_0, z, q_1] = a[q_0, z, q_1][q_1, z_0, q_1]$

$A = [q_0, z_0, q_1]$, $B = [q_0, z, q_1]$, $C = [q_1, z, q_1]$

$D = [q_1, z_0, q_1]$

So,

$S \rightarrow A$,

$A \rightarrow aBC$

$B \rightarrow aBD$

$B \rightarrow 1$

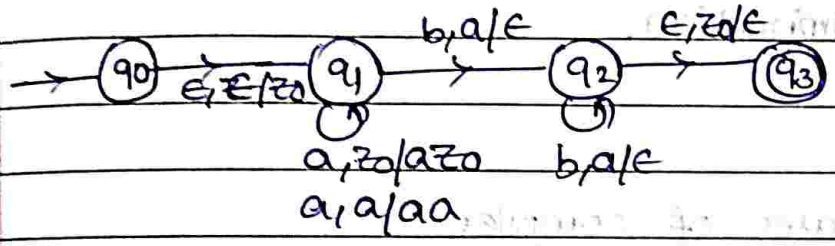
$\Big\}$ $B \rightarrow aBD \mid 1$

$C \rightarrow b$

$D \rightarrow \epsilon$

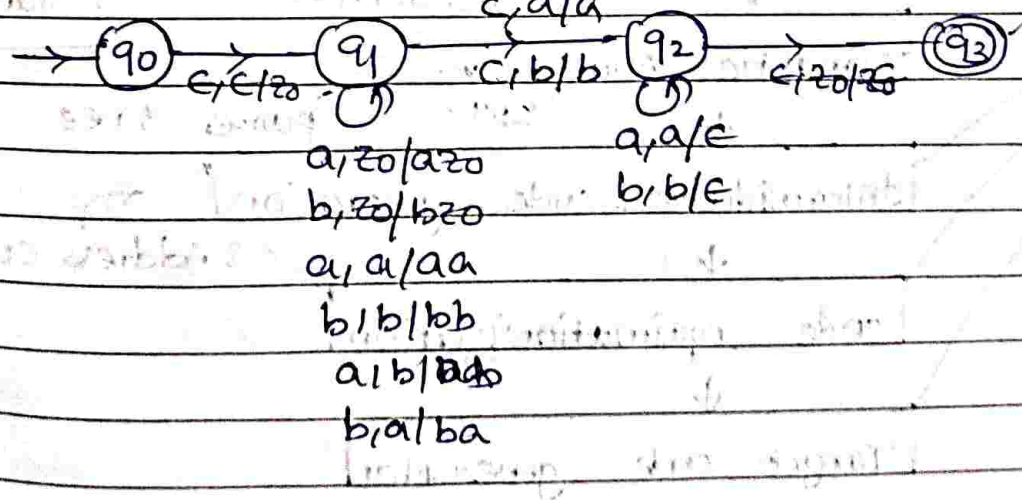
8) Describe PDA for following language.
$L = \{w/w \text{ of form } a^n b^n\}$

A) $L = \{w/w \text{ of form } a^n b^n\}$



9) Describe PDA for language.
$L = \{xcx^r / x \in \{a,b\}^*\}$

A) $L = \{xcx^r / x \in (a,b)^*\}$

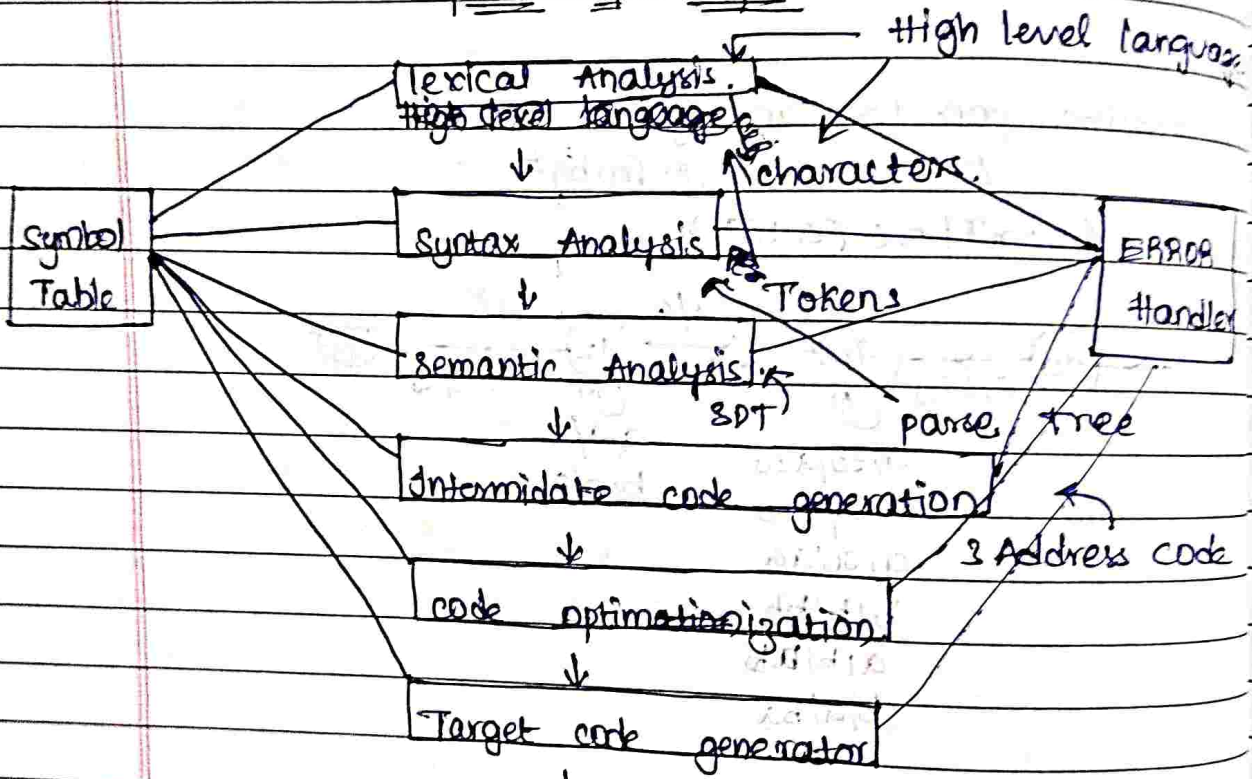


compiler Design

1) lexical Analysis
★★★ 2) parser (Syntax Analyzer)
3) Semantic Analysis
★ 4) Intermidate code generator.
★ 5) code optimization.

lectures:

phases of compiler:-



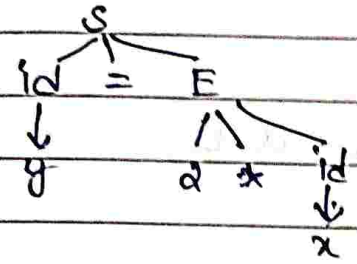
Symbol Table:-

Symbol Table stores the output of the lexical Analysis and gives the data to syntax Analysis as input.

Syntax Analysis :-

It checks Tokens are syntatically organized (or) not.

Ex:- y = 2*x



Semantic Analysis :-

Semantic Analysis will check for logical error. It generally analysis in compile time where any lexical error, syntax error (or) even logical error then they are checked here.

The output of it is SDT

→ SDT stands for syntax directory Transition. SDT has combined with grammar rules with actions.

Front end :-

Front end part contains

1) lexical Analysis.
2) Syntax Analysis.
3) Semantic Analysis.
4) Intermidate code generation.

back end :-

Back end part contains

1) code optimization.
2) Target code generator.

Intermidate code generation :-

Intermidate code generation will combi the grammar rules with actions into three addre code.

Example

$z = a + b * c$

$t_1 = b * c$, $t_2 = a + t_1$

$\boxed{z = t_2}$

code optimization :

code optimization makes code efficient.

Example

$t_1 = 2 * x$

$y = t_1$

$t_1 = x + x$

$y = x + x$

Lexical Analysis

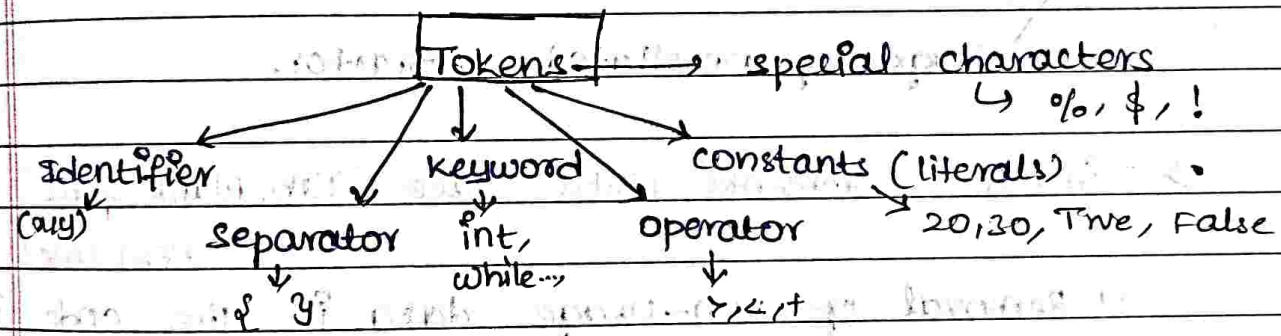
→ lexical analysis is the first phase in compiler design.

→ lexical analysis takes input as a stream of characters which are written in high-level-language.

→ Each and every character are converted into Tokens as output of it.

→ This process of conversion is called Tokenization

→ The process (or) program where dividing the tokens are called as lexer, tokenizer, scanner.



Example

```
int main (){
  /* Find Max */
  int a=20; b=30;
  if (a<b)
      return (b);
  else
      return (a);
}
printf ("i=%d, &i=%x", i, &i);
```

a, b are identifiers
( ), { } are separators.
int, main, return, if, else are keywords.
=, < are operators.
20, 30 are constants (literals)
%, & are special characters.

↳ Tokenization

4) Give Error messages.
   a) Exceeding length.
   b) unmatched string.
   c) Illegal character.

   Usage of more length than original identifier be then exceeding length, error message occurs.

   Usage of unrellated string at syntax can leads to this.

   Usage of unrellated character.

5) Eliminate Comments, white space (Tab, Blank space, new line)
   → Removal of non-usage data in the code is called the process.

We use the Finite automata, because it can do any program to Tokens & NFA or DFA can also be used.

First & Follow

First(A) :-

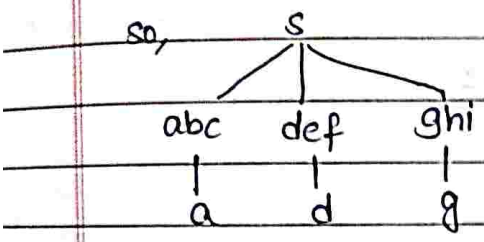
contains all terminals present in first place of every string derived by A.

Rules:

1) S → abc | def | ghi
2) First (terminal) = terminal
3) First (ϵ) = ϵ

example:

S → abc | def | ghi



so first of S is a, d, g.

example 1

S → ABC | ghi | jkl	first of S is A, g, j so a, b, c, g, j
A → a | b | c	first of A is a, b, c
B → b	first of B is b
D → d	first of D is d

example 2

S → ABC	first of S is ABC so a, b, c, d, e, f, ϵ
A → a | b | ϵ	first of A is a, b, ϵ
B → c | d | ϵ	first of B is c, d, ϵ
C → e | f | ϵ	first of C is e, f, ϵ

example 3

E → TE'	first of E is id, (
E' → *TE' | ϵ	first of E' is *, ϵ
T → FT'	first of T is F so id, (
T' → ϵ | +FT'	first of T' is ϵ, +
F → id | (E	first of F is id, (

Follow ():-

Follow(A) contains set of all terminals present immediate in right of 'A'

Rules:-

1) Follow of start symbol is $

Fo(A) = {$}

2) S → ACD

C → a|b

Fo(A) = First(C) = {a, b}

Fo(D) = Follow(S) = {$}

3) S → aSbS | bSaS | ∈

* Follow never contains ∈

example

S → ACB | CbB | Ba

A → da | BC

B → g | ∈

C → h | ∈

Follow(S) → {$}

Follow(A) → ~~First(C) → First~~ {C}

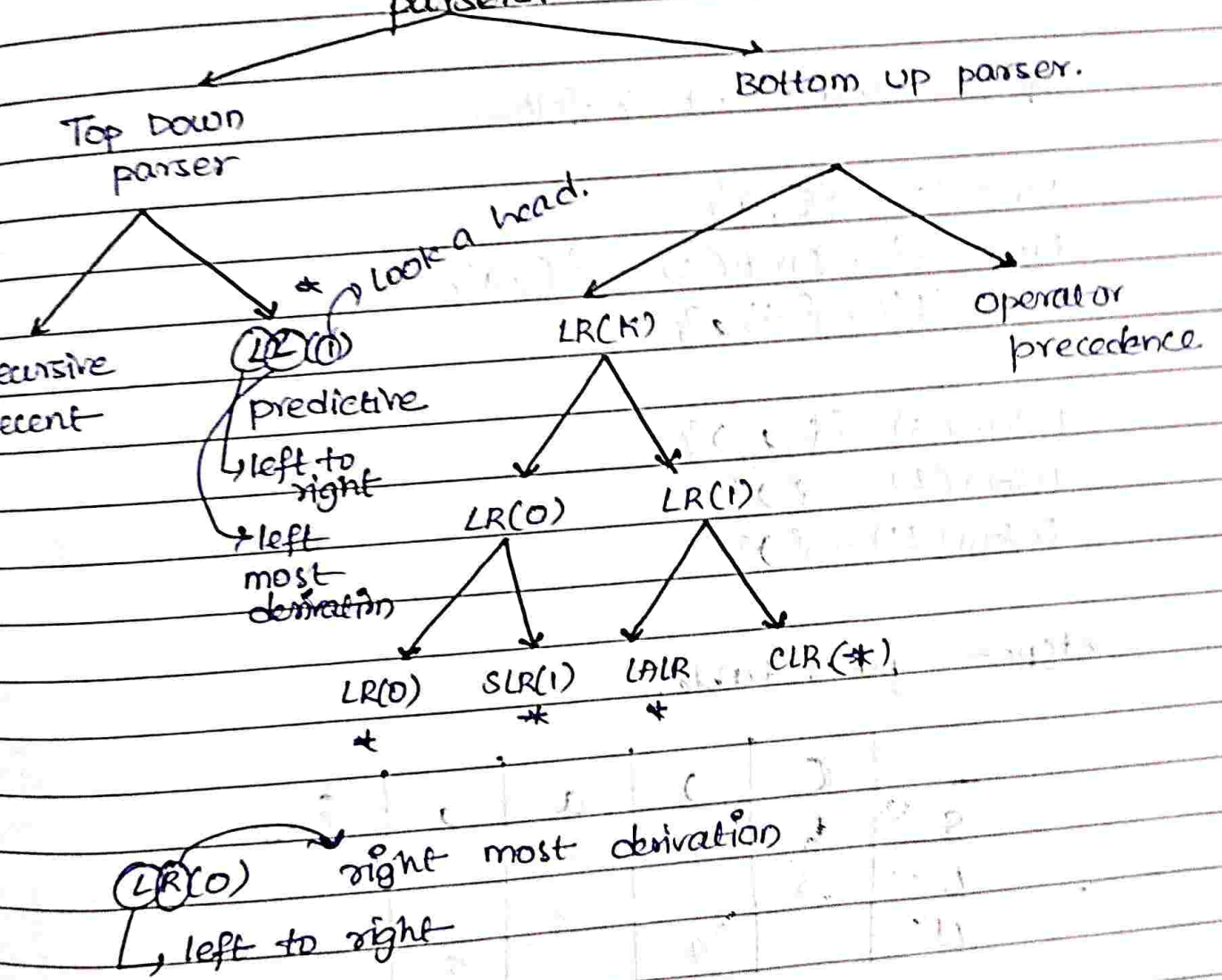
Follow(B) → {$, a, h}

Follow

What is parsing & Types of parser.

parsing:-

It is a process of deriving string from a given grammar.



LL(1) grammar parsing Table

Question

$S \rightarrow (L) / a$

$L \rightarrow SL'$

$L' \rightarrow \epsilon / , SL'$

Step1 Find First & follow.

First (S) = {(, a}

First (L) = First (S) = {(, a}

First (L') = {ϵ, ,}

Follow (S) = {$, , ,)}

Follow (L) = {)}

Follow (L') = {)}

Step2 parse table.

	(	)	a	,	$
S	1		2		
L	3		3		
L'		4		5	

① $S \rightarrow (L)$ ② $/ a$

③ $L \rightarrow SL'$

④ $L' \rightarrow \epsilon$ ⑤ $/ , SL'$

If in any cell there exists more than one entry then it not LL(1).

This LL(1).