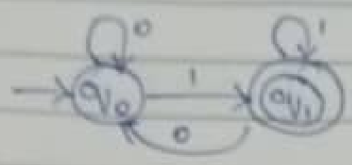
Wednesday

Date 1/3/2023
Page No.

Unit :- 1 Introduction

* Representation of DFA



· Input 0,1

· Transaction table.

δ	0	1
$\rightarrow q_0$	q_0	q_1
$* q_1$	q_0	q_1

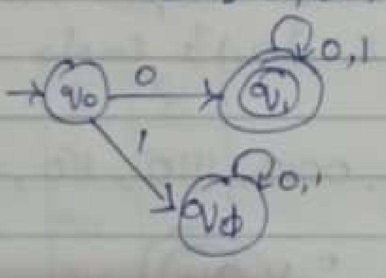
1. Construct a language accept the string with length 2 over sign equal to 0,1.

Soln:-

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

2. Construct a DFA which accept's all the strings over the alphabet $\Sigma = \{0,1\}$ start's with 0.

Soln:- $L = \{0, 01, 011, 0101, 0110, 0000 \ldots\ldots\}$

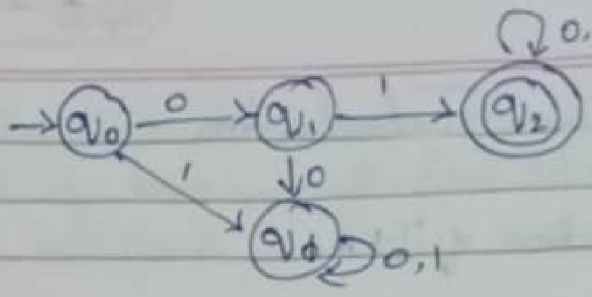


δ	0	1
$\rightarrow q_0$	q_1	q_ϕ
$* q_1$	q_1	q_1
q_ϕ	q_ϕ	q_ϕ

3. Construct DFA which accept the strings over the alphabet $\Sigma = \{0,1\}$ & starts 0,1.

Soln:-

$$L = \{01, 011, 0110, 0101 \ldots\ldots\}$$



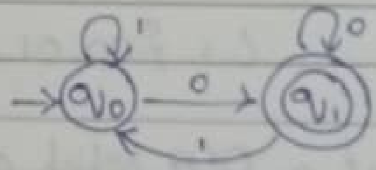
δ	0	1
→ q_0	q_1	q_ϕ
q_1	q_ϕ	q_2
* q_2	q_2	q_2
q_ϕ	q_ϕ	q_ϕ

4. Construct a language which accepts all string over of alphabet $\Sigma = \{0,1\}$ ends with 0.

Soln:-

$L = \{0, 00, 10, 010, 0110, 0010, 00110, 110, 1110 \ldots\}$

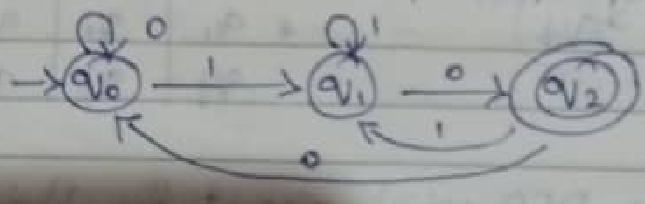
δ	0	1
→ q_0	q_1	q_0
* q_1	q_1	q_0



5. Construct a DFA which accept's all the strings over an alphabet $\Sigma = \{0,1\}$ ends with 10.

Soln:-

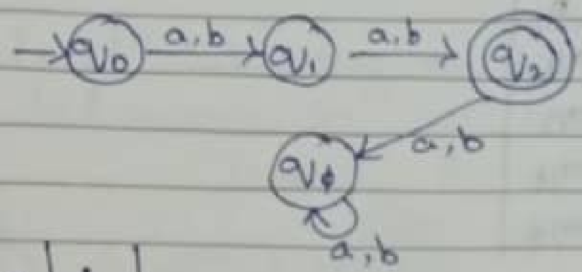
$L = \{10, 010, 0110, 0010, 1110, 110, 1010 \ldots\}$



δ	0	1
→ q_0	q_0	q_1
q_1	q_2	q_1
* q_2	q_0	q_1

6. Construct a DFA which accept all the Strings over the alphabet $\Sigma = \{a, b\}$ where the length of the string is 2.

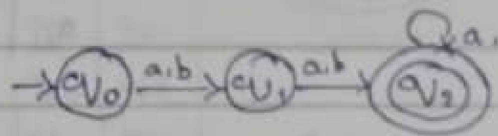
Soln:- $L = \{aa, bb, ab, ba\}$



δ	a	b
$\rightarrow q_0$	q_1	q_1
q_1	q_2	q_2
$* q_2$	q_ϕ	q_ϕ
q_ϕ	q_ϕ	q_ϕ

7. Construct a DFA which accept all the Strings over an alphabet $\Sigma = \{a, b\}$ where the length of the String ≥ 2.

Soln:- $L = \{aa, bb, aba, aab, baa \ldots\}$

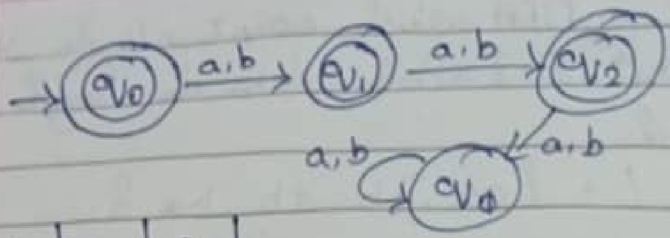


δ	a	b
$\rightarrow q_0$	q_1	q_1
q_1	q_2	q_2
$* q_2$	q_2	q_2

8. Construct a DFA which accept all the Strings over an alphabet $\Sigma = \{a, b\}$ where the length of the String ≤ 2.

Soln:-

$L = \{\epsilon, a, b, ab, ba, aa, bb\}$

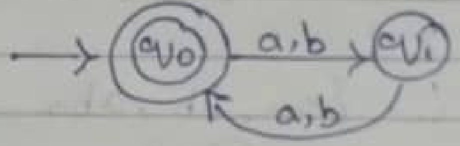


δ	a	b
$\rightarrow * q_0$	q_1	q_1
$* q_1$	q_2	q_2
$* q_2$	q_ϕ	q_ϕ
q_ϕ	q_ϕ	q_ϕ

9. Construct DFA which accept all the strings over an alphabet $\Sigma = \{a, b\}$ where the length of the string is Even.

Soln:-

$L = \{ab, ba, aa, bb, aaa, bbb, aaaa, bbbb, abab, aabb \ldots\}$



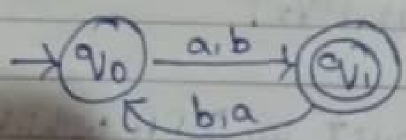
δ	a	b
$\rightarrow * q_0$	q_1	q_1
q_1	q_0	q_0

10. $\Sigma = \{a, b\}$ length of the string is Odd?

Soln:-

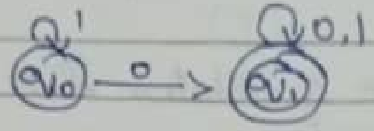
$L = \{a, b, aab, aba, bba, aaa, bbb \ldots\}$

δ	a	b
$\rightarrow q_0$	q_1	q_1
$* q_1$	q_0	q_0



11. Construct a DFA which accept all the strings over an alphabet $\Sigma = \{0, 1\}$ where the string contains 0 as sub string.

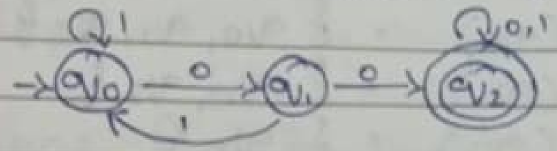
Soln:- $L = \{0, 01, 010, 011, 100, 1010, 1000, 1011 \ldots\}$



δ	0	1
$\rightarrow q_0$	q_1	q_0
$* q_1$	q_1	q_1

12. $\Sigma = \{0,1\}$ Substring as 00.

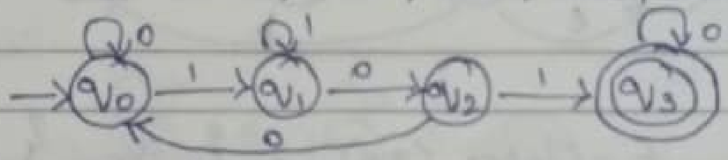
Soln:- $L = \{00, 000, 001, 100, 0000, 0001, 1001, 1100 \ldots\}$



δ	0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_2	q_0
$* q_2$	q_2	q_2

13. $\Sigma = \{0,1\}$ where string contains 101.

Soln:- $L = \{101, 0101, 1101, 1010, 00101, 01010 \ldots\}$

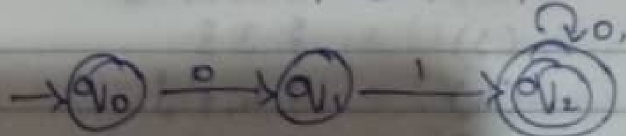


δ	0	1
$\rightarrow q_0$	q_0	q_1
q_1	q_2	q_1
q_2	q_0	q_3
$* q_3$	q_3	q_3

* NFA (Non-Deterministic finite automata)

1. Construct NFA which accept all the strings over an alphabet $\Sigma = \{0,1\}$ where the string starts 01.

Soln:- $L = \{01, 001, 011, 0011, 0101 \ldots\}$



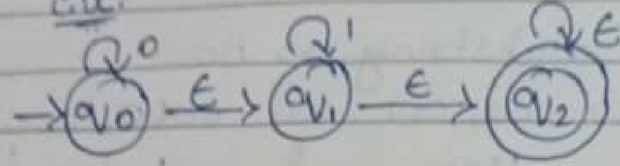
δ	0	1
$\rightarrow q_0$	q_1	ϕ
q_1	ϕ	q_2
$* q_2$	q_2	q_2

* ε-Closer / Epsolon Closer (ε)

- The set of state's that are reachable from the state q on ε-moves.

Ex:-

1.



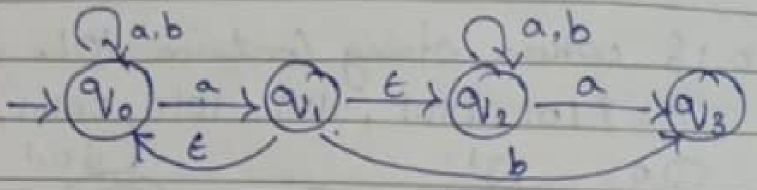
Soln:-

ε-closer (q_0) = { q_0, q_1, q_2 }

" (q_1) = { q_1, q_2 }

" (q_2) = { q_2 }

2.



Soln:-

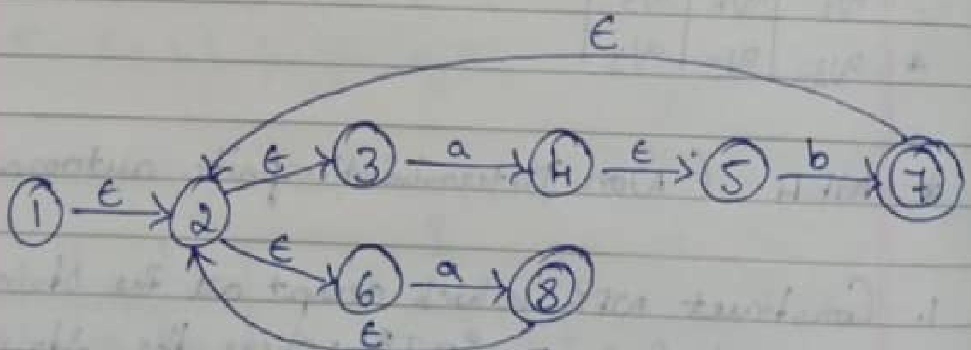
ε-Closer (q_0) = { q_0 }

" (q_1) = { q_1, q_0, q_2 }

" (q_2) = { q_2 }

" (q_3) = { q_3 }

3.



Soln:-

ε-Closer (1) = { 1, 2, 3, 6 }

(2) = { 2, 3, 6 }

(3) = { 3 }

(4) = { 4, 5 }

(5) = { 5 }

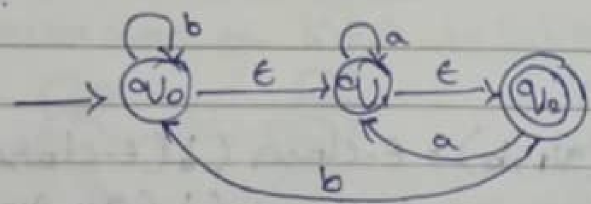
ϵ-closer (6) : {6}
" (7) : {7, 2, 3, 6}
" (8) : {8, 2, 3, 6}

* Conversion of NFA with Epsilon to NFA without Epsilon in automata theory.

Step1 :- Find Epsilon Closure.
Step 2 :- Find the transaction,
$\delta(q, a) = \epsilon\text{-closer}(\delta(\epsilon\text{-closer}(q), a))$
Step3 :- Find the finial state, $F' = F \cup \{q_0\}$
{ if ϵ-closure (q_0) is having F as a member }.

Ex:-

① (F)



Sol. a. Step1 :- find Epsilon closure.

ϵ-Closure (q_0) = {q_0, q_1, q_2}
" (q_1) = {q_2, q_1}
" (q_2) = {q_2}

b. Step2 :- Find the transaction.
$\delta'(q, a) = \epsilon\text{-closer}(\delta(\epsilon\text{-closer}(q), a))$
• (q_0, a)
$\delta'(q_0, a) = \epsilon\text{-closer}(\delta(\epsilon\text{-closer}(q_0)\ a))$
$\Rightarrow \epsilon\text{-closer}(\delta(q_0, q_1, q_2)\ a))$
$\Rightarrow \epsilon\text{-closer}(\delta[(q_0, a) \cup (q_1, a) \cup (q_2, a)])$
$\Rightarrow \epsilon\text{-closer}(\delta(\phi \cup q_1 \cup q_1)$

ϵ-closer $(q_1) \Rightarrow \{q_1, q_2\}$

- (q_0, b)

$\delta'(q_0, b) = \epsilon\text{-closer}(\delta(\epsilon\text{-closer}(q_0)\, b))$

$\Rightarrow \epsilon\text{-closer}(\delta(q_0, q_1, q_2), b))$.

$\Rightarrow \epsilon\text{-closer}(\delta(q_0, b) \cup (q_1, b) \cup (q_2, b))$

$\Rightarrow \epsilon\text{-closer}(\delta(q_0 \cup \phi \cup q_0)$

$\Rightarrow \epsilon\text{-closer}(q_0) = \{q_0, q_1, q_2\}$.

- (q_1, a)

$\delta'(q_1, a) = \epsilon\text{-closer}(\delta(\epsilon\text{-closer}(q_1)\, a))$

$\Rightarrow \epsilon\text{-closer}(\delta(q_1, q_2), a))$

$\Rightarrow \epsilon\text{-closer}(\delta(q_1, a) \cup (q_2, a))$

$\Rightarrow \epsilon\text{-closer}(\delta(q_1 \cup q_2)$

$\Rightarrow \epsilon\text{-closer}(q_1) = \{q_1, q_2\}$

- (q_1, b)

$\delta'(q_1, b) = \epsilon\text{-closer}(\delta(\epsilon\text{-closer}(q_1)\, b))$

$= \epsilon\text{-closer}(\delta(q_1, q_2), b))$

$= \epsilon\text{-closer}(\delta(q_1, b) \cup (q_2, b))$

$= \epsilon\text{-closer}(\delta(q_0, q_1, q_2)\}$.

- (q_2, a)

$\delta'(q_2, a) = \epsilon\text{-closer}(\delta(\epsilon\text{-closer}(q_2)\, a)$

$=$ " $(\delta(q_2), a))$

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

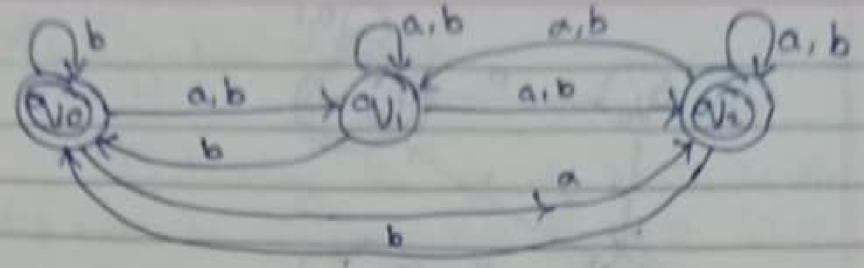
$=$ "

$= \epsilon\text{-closer}(\delta(q_1, q_2))$

- (q_2, b)

$= \epsilon\text{-closer}(\delta(q_0, q_1, q_2))$.

(F')



C. Step 3 :- Find the finial state

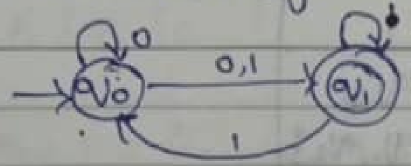
$f' = f \cup \{q_0\}$

{ if ε-closure (q_0) is having F as a member otherwise no }

ε-closure (q_0) = { q_0, q_1, q_2 }

[here q_2 is finial state in question. If there are any finial state in q_0 then consider q_0 as a finial state in second diagram.].

* Conversion of NFA to DFA in automata theory

1. 

δ	0	1
→ q_0	$q_0\ q_1$	q_1
* q_1	φ	{q_1, {q_0}

Soln:-

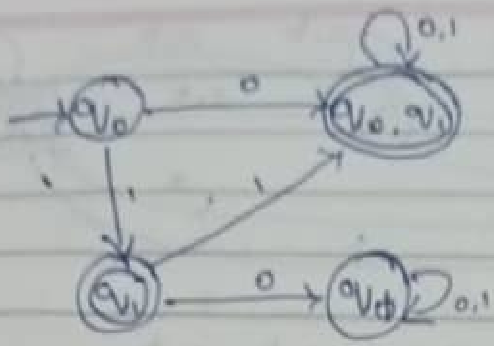
Step 1 :- find out the initial state. Since it is q_0 then find out the transaction's by using the i/p 0 & 1.

δ	0	1
→ q_0	{q_0, q_1}	q_1
{q_0, q_1}	{q_0 q_1}	{q_0, q_1}
* q_1	q_ϕ	{q_0, q_1}
q_ϕ	q_ϕ	q_ϕ

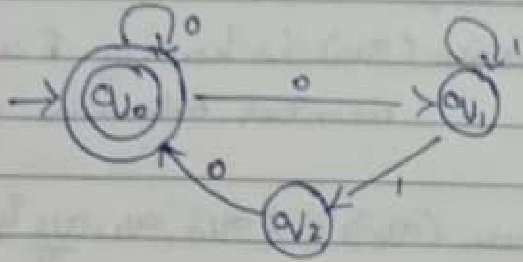
{q_0, q_1}

[{q_0, q_1} 0] = δ(q_0, 0) ∪ (q_1, 0)

= (q_0, q_1) ∪ φ = q_0

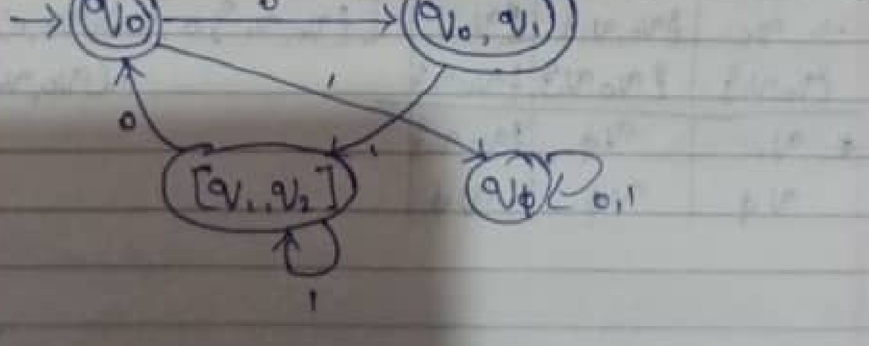


2.



Soln:- find the initial source. Since it is q_0 then find out transaction by using I/P 0 & 1.

δ	0	1
→ * q_0	$\{q_0 q_1\}$	q_ϕ
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_1, q_2\}$
$\{q_1, q_2\}$	q_0	$\{q_1, q_2\}$
q_ϕ	q_ϕ	q_ϕ

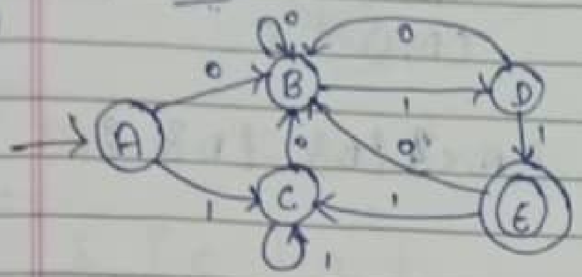


* Minimization of DFA.

Reducing the no of states in the given DFA none as Minimization of DFA

Ex:-

①



Soln:- Step1:- Construct transition table.

δ	0	1
→ A	B	C
B	B	D
C	B	C
D	B	E
* E	B	C

Step 2:- Generate two set's 1 with finial State & another 1 with non-finial state is Known as zero equivalent's.

Non-finial = {ABCD}
finial = {E}

One Equivalence. {A,B,C} {D} {E}

1. A,B
(A,0) = B (A,1) = c $\boxed{A=B}$ //:
(B,0) = B (B,1) = D

2. A,C

(A,0) = B   (A,1) = C $\boxed{A = C}$

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

3 A,D

(A,0) = B   (A,1) = C $\boxed{A \neq D}$

(D,0) = B   (D,1) = E

Two equivalence {A,C} {B} {D} {E}

A,B

(A,0) = B   (A,1) = C $\boxed{A \neq B}$

(B,0) = B   (B,1) = D

~~A,C~~

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

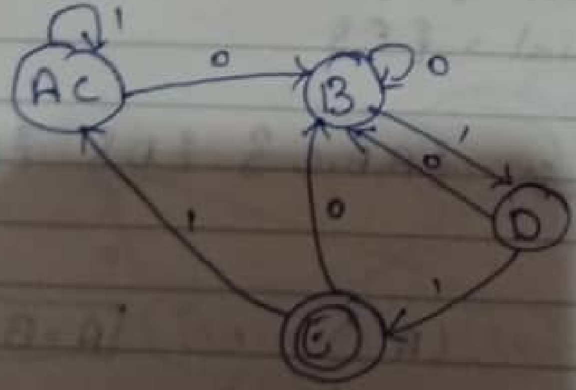
~~(C,0) = B~~   ~~(C,1) = C~~

Three Equivalence {A,C} {B} {D} {E}

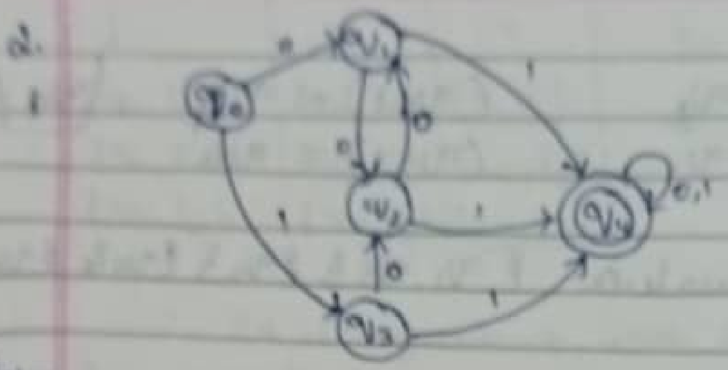
A = C

(A,0) = B   (A,1) = C $\boxed{A = C}$

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



2.



Soln:-

Step 1:- ~~Truth to~~ Transaction table.

δ	0	1
$\rightarrow q_0$	q_1	q_3
q_1	q_2	q_4
q_2	q_1	q_4
q_3	q_2	q_4
$* q_4$	q_4	q_4

Step 2:- Generate 2 state's with finial & non-finial states.

$A = \{q_0, q_1, q_2, q_3\}$
$B = \{q_4\}$

One equivalence. $\{q_0, q_2, q_3\}$ $\{q_1\}$ $\{q_4\}$

1. q_0, q_1

$(q_0, 0) = q_1$ $(q_0, 1) = q_3$ $q_0 \neq q_1$

$(q_1, 0) = q_2$ $(q_1, 0) = q_4$

q_0, q_2

~~$(q_0, 0) = q_1$~~ ~~$(q_0, 1) = q_3$~~

$(q_2, 0) = q_1$ $(\cancel{q_2}, 1) = q_4$

Two Equivalence $\{q_0, q_3\}$ $\{q_2\}$ $\{q_0\}$ $\{q_4$

q_1, q_2

$(q_1, 0) = q_2$ $\quad$ $(q_1, 1) = q_4$ $\quad$ $\boxed{q_1 \neq q_2}$

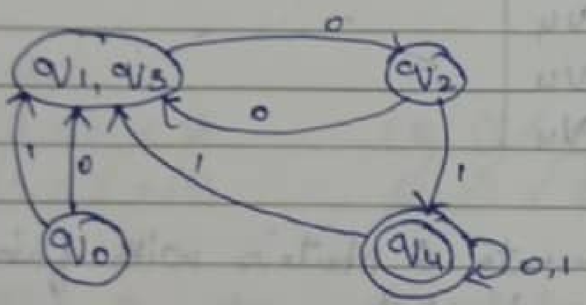
$(q_2, 0) = q_1$ $\quad$ $(q_2, 1) = q_4$

Three Equivalence $\{q_1, q_3\}$ $\{q_2\}$ $\{q_0\}$ $\{q_4\}$

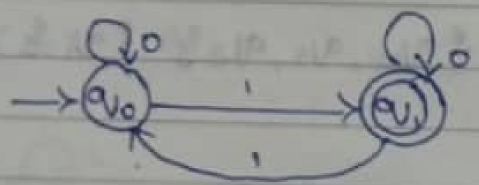
q_1, q_3

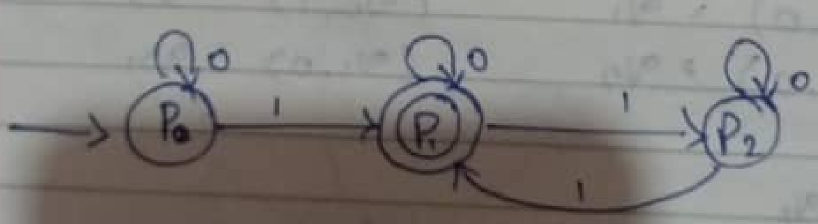
$(q_1, 0) = q_2$ $\quad$ $(q_1, 1) = q_4$ $\quad$ $\boxed{q_1 \neq q_3}$

$(q_3, 0) = q_2$ $\quad$ $(q_3, 1) = q_4$



* Equivalence of DFA in automata theory.





Soln:-

d	0	1
$\{q_0, P_0\}$	$\{q_0, P_0\}$	$\{q_1, P_1\}$
$\{q_1, P_1\}$	$\{q_1, P_1\}$	$\{q_0, P_2\}$
$\{q_0, P_2\}$	$\{q_0, P_2\}$	$\{q_1, P_1\}$

So Hence proved.

It is equivalent with 2 states. Each & Every state is having the transaction with 0 & 1

Find the finial & non-finial states from the transaction table.

from the transaction table no one state is having the combination of finial & non finial states hence both the FSM are equal.

6/8/2022

Unit:- 2

• Regular Expression

It is defined as Epsilon ϵ & regular expression corresponding to $L = \{\epsilon\}$

ϕ is a regular expression corresponding to $L = \{\}$

x is a reg corresponding to $L = \{x\}$

if x reg Exp over the lang $L(x)$ & y is a regular exp over the language $L(y)$ then $x+y$ is a reg exp corresponding to $L(x) \cup L(y)$ where

$$[L(x \cup y) = L(x) \cup L(y)]$$

if x y is a regular expression then ~~$L(x^*) = (L(x))^*$~~ $L(x) = L(y)$ where $(x.y) = L(x).L(y)$

• R-closure is a regular expression then $L(R^*) = (L(R)^*)$

The language used for writing the regular expression is called regular language (RL)

RL can be defined by DFA, NFA with ϵ, NFA & with regular expression.

Ex:-

1. RE denoted the lang with strings having any no of a's $\Sigma = \{a\}$

Soln:- $RE = \{a^*\}$

2. RE denoting language with the string's any no of a's & b's over $\Sigma = \{a, b\}$

Soln:- $RE = (a+b)^* b$

3. RE denoting the lang with the string starts with a & end's with b' and having no.

Soln:- $RE = a(a+b)^* b$

- Rules of RE

If P, Q, R then

Rule 1:- $\phi + r = r$
Rule 2:- $\phi . r = r . \phi \Rightarrow \phi$
Rule 3: $\epsilon . r = r . \epsilon \Rightarrow r$
Rule 4:- $r + r \Rightarrow r$
Rule 5:- $r^* . r^* \Rightarrow r^*$
Rule 6:- $r^* r = r r^* \Rightarrow r^*$
Rule 7:- $(r^*)^* = r^*$
Rule 8:- $\epsilon + r^* r = \epsilon + r r^* \Rightarrow r^*$
Rule 9:- $(Pq)^* P = P(qP)^*$

* Adream's theorem.

If P & q are two regular expression under P does not have Epsolon (ϵ) transati... then the R = Q + RP will have a unique solution $R = QP^*$

P Q

RE

Theorme Proof !

1. Proof 1 :- $R = Q + RP \rightarrow$ Eqn ①
$R = QP^* \rightarrow$ Eqn ②

Soln. $R = Q + RP \rightarrow$ ①
$\Rightarrow Q + (QP^*)P$
$Q + QP^*P$
$Q(1 + P^*P)$
$Q(\epsilon + P^*P)$
$\boxed{R \Rightarrow QP^*}$ //:

2. Proof 2 !

Substitute Equ Eqn ① in $R = Q + RP$.

Soln:- $R = Q + RP \rightarrow$ ①
$R = Q + QP + RP^2 \rightarrow$ ②

$\Rightarrow Q + (Q + RP)P$
$\Rightarrow Q + QP + RP^2 \rightarrow$ ②

$R = Q + QP + RP^2 + \rightarrow$ ②
$Q + QP + (Q + RP)P^2$
$Q + QP + QP^2 + RP^3 \rightarrow$ ③

$R \Rightarrow Q + QP + QP^2 + RP^3 \rightarrow$ ③
$Q + QP + QP^2 + (Q + RP)P^3$
$Q + QP + QP^2 + QP^3 + RP^4 \rightarrow$ ④
$\Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \ldots\ldots \Sigma^n$

in eqn ④ 'Q' in common
$R = Q(1 + P + P^2 + P^3 + P^4 \ldots P^n)$

Consider 'i' & G

• Conversion of Finite automata to regular expression.

There method are used to Convert finite Automata to RE.

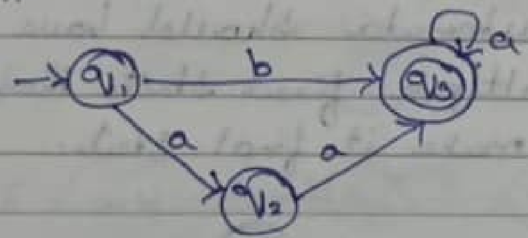
1. Arden's methods.
2. State ~~Enimation method~~. Elemination method.

1. Arden's methods

① Equation for each state based on incoming Edge.
② An ε to the Equation of initial state.
③ Symplify the equation using the Arden's method & find RE for final state.

Ex:-

a.



Soln:-

$q_1 = \epsilon \rightarrow ①$

$q_2 = q_1 . a \rightarrow ②$

$q_3 = q_1 . b + q_2 . a + q_3 . a \rightarrow ③$

Substitute equation 1 & 2 in equation 3.

$q_3 = q_1 . b + q_2 . a + q_3 . a$

$= \epsilon b + q_1 aa + q_3 a \quad R = QP^*$

$= (b + aa) + q_3 a$

$Q = RP$

$= (b+aa) + q_3 a$

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

$= (b+aa) + \epsilon b + q_1 aa + q_3 . a$

$= (b+aa) + b + aa + q_3 a$

$= (b+aa) + (b+aa) + q_3 . a$

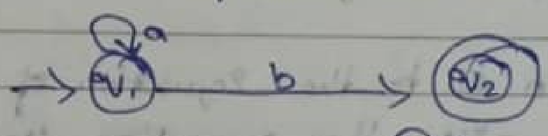
$= (b+aa) + q_3 . a$

2. State Elemination method.

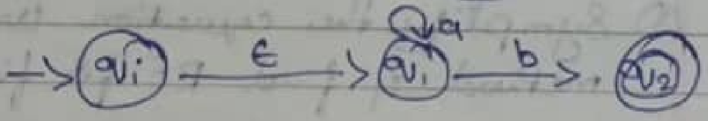
① In a finite automata initial state should not contain any incoming edge, if exist Create new state & make it as initial state

Ex:-

①

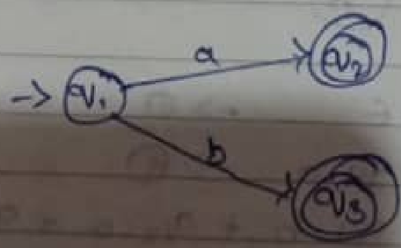


Sol:-

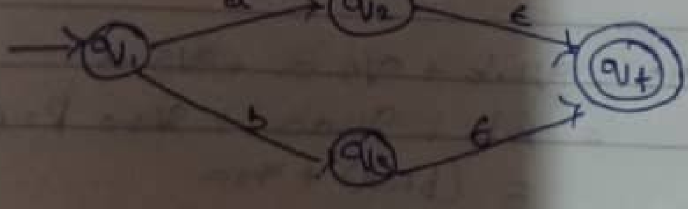


② The finite automata should have only one state if multiple final state then & Create new state & make it final state.

②



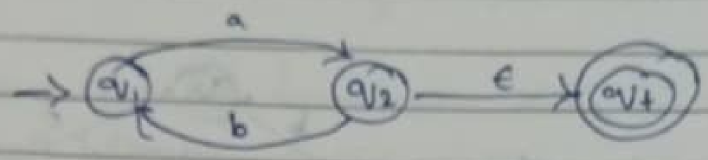
Sol:-



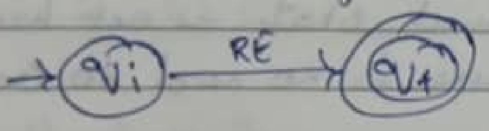
③ final state should not have out going edge if exist create a new state & make it as final state.

①

Sol:-

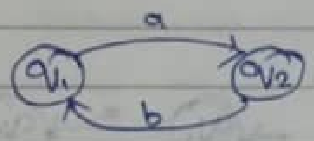


④ finite automata should have only 2 state after eliminating every state one after another. It should be in the format of.

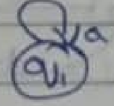


Basic terms.

1. RE = a+b



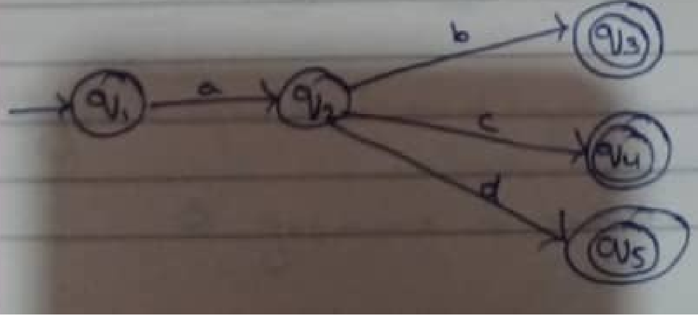
2. RE = a^*



3. RE = ab

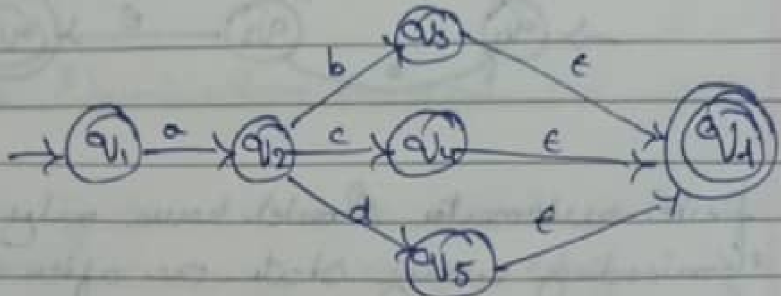
* Problems.

1.



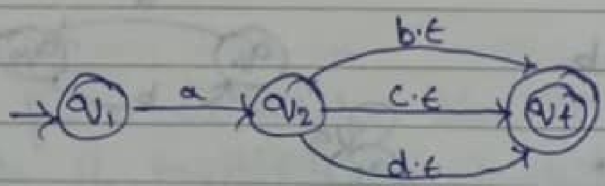
S1. q_1 is the initial stack without any incoming edge. so there is no need to create new state.

S2. multiple finial state exist. So create new finial state q_f

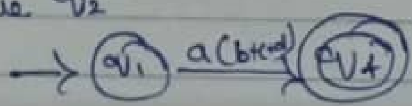


S3. Since the finial state is not having any out going so there is no need to create new state.

Eliminate / Remove the q_3, q_4, q_5



Remove q_2



* Regular Grammer.

It is a set of quadraple element
G = {V, T, P, S}

1. V is defined as set of non terminals
   NT = {A, B, C, D} capital letters (which is similar to state's in finite automata)

2. T is a set of terminals which is represented in small letter's
   T = {a, b, c, d ... 0, 1, 2}
   Similar to input Elements Σ in finite automata.

3. P is production rule.

4. S start state.

→ Production rule it is Known as grammer by having both RHS & LHS.

a. LHS it contain's only non terminals.
b. RHS is a Combination of ε & a string containing a terminal & non-terminals.

LHS → RHS
A → aB
└→ is used by input 'a'

Eoc :- → $A \xrightarrow{a} B$

* Regular Grammer.

A grammer is set be regular grammer

based on the production rule.
LHS -> RHS

* Production rule.
1. LHS :- It contains single non-terminal ?
2. RHS :- It contains Epsolon, terminal, terminal followed by non-terminal, non-terminal followed by terminal.

LHS -> RHS
S -> ε
A -> b
C -> bZ
X -> Yc

Not a RG
Z -> aB d

R.G was divided into 2 types

1. Left linear grammer.
2. Right linear grammer.

1. left linear grammer.

LHS -> RHS
A -> Aa

2. Right linear grammer.

~~RHS -> LHS~~ LHS -> RHS
B -> dE

* Conversion of FA to RG

1. 

Soln:- RG = {V, T, P, S}

V = {A, B} T = {a, b}

S = {A}

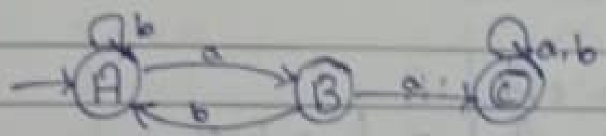
P (Production rule)

A → aA A → a

A → bB A → b

B → aB B → a

B → bB B → b

2. 

Soln:- RG = {V, T, P, S}

V = {A B C} T = {a, b}

S = {A}.

A → aB

A → bA A → a

B → aC A → b

B → bA B → a

C → aC B → b

C → bC C → a C → b

* Conversion of RG to FA

1. The no of state's in FA should be equal to the no of non-terminals + 1
2. Each state in automata represent non-terminal in RG.

3. Additional state will be the final state.

* Transaction's in Automata (δ)

1. For every production $A \to aB$, it should be return as $\delta(A, a) \to B$

2. For every production $A \to b$ that it should be written as $\delta(A, b) \to$ final state.

Ex:-

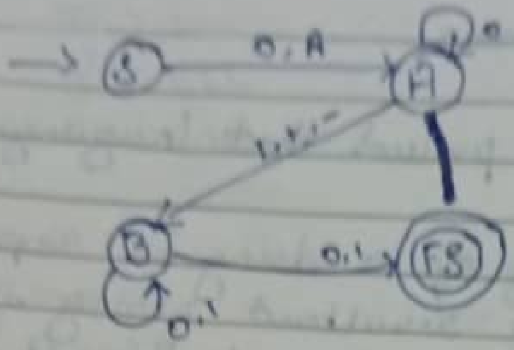
$S \to 0A \,/\, 1B \,/\, 0 \,/\, 1$
$A \to 0S \,/\, 1B \,/\, 1$
$B \to 0A \,/\, 1S$

a. $S \to 0A \mid 1A$
$A \to 0A \mid 1B \mid +B \mid -B$
$B \to 0B \mid 1B \mid 0 / 1$

Soln:-

① $S \to 0A$ $B \to 0B$
$S \to 1A$ $B \to 1B$
$A \to 0A$ ~~$B \to 0$~~
$A \to 1B$ $B \to 0$
$A \to +B$ $B \to 1$
$A \to -B$

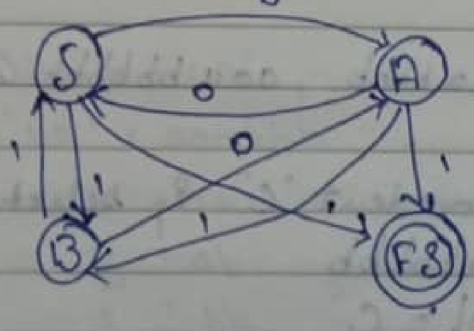
② $\delta(S0) \to A$ $B \to 0 \Rightarrow \delta(B, 0) \Rightarrow FS$
$\delta(S1) \to A$ $B \to 1 \Rightarrow \delta(B, 1) \Rightarrow FS$
$\delta(A0) \to A$
$\delta(A1) \to B$
$\delta(A+) \to B$
$\delta(A-) \to B$
$\delta(B0) \to B$
$\delta(B1) \to B$



b. $S \to 0A \mid 1B \mid 0/1$
$A \to 0S \mid 1B \mid 1$
$B \to 0A \mid 1S$

<u>Soln:-</u>

$S \to 0A$	$\delta(S\ 0) \Rightarrow A$
$S \to 1B$	$\delta(S\ 1) \Rightarrow B$
$S \to 0$	$\delta(S\ 0) \Rightarrow FS$
$S \to 1$	$\delta(S\ 1) \Rightarrow FS$
$A \to 0S$	$\delta(A\ 0) \Rightarrow S$
$A \to 1B$	$\delta(A\ 1) \Rightarrow B$
$A \to 1$	$\delta(A\ 1) \Rightarrow FS$
$B \to 0A$	$\delta(B, 0) \Rightarrow A$
$B \to 1S$	$\delta(B, 1) \Rightarrow S$



Pumping Lemma

1. It is used to prove that language is not regular.
2. If substring of a string is repeated many times & if the resultant string is also available in language L then it is called regular.

Step 1 :- Consider a language as regular.
Step 2 :- Assume a constant 'c' and select the string 'w' from 'L' such that $|w| \geq c$
Step 3 :- Divide W as x, y, z where $|y| > 0$ case 1
case 2 $|xy| \leq c$
Step 4 :- For every $i \geq 0$, every string of the form $x y^i z \in L$

Example. 1-4

1. Prove $L = \{a^n b^{2m} \mid m, n > 0\}$ & Prove that it is not regular

Soln:-

(S1) $L = \{abb, aabbbb, aaa\,bbbbbb, aaaa\,bbbbbbbb \ldots\}$

(S2) Assume constant 'c' & select the string 'w'
aaa bbbbbb
$|w| \geq c$
$|aaabbbbbb| \geq 9$

(S3) Divide 'w'
w = aaa bbbbbb

$x = aaa$
$y = bbb$
$z = bbb$

Case 1
$|y| \geq 0$
$|bbb| > 0$

Case 2
$|xy| \leq c$
$|aaabbb| \leq 9$

(S4) for every $i \geq 0$, xy^iz.
Assume $i = 0$
aaa bbb is not regular
Hence proved.

2. Prove the lang $L = \{a^n b^n \mid n \geq 1\}$

Soln:-

(S1) $L = \{ab, aabb, aaabbb, aaaa, bbbb \ldots\}$

(S2) Constant 'c' & select the string 'w'
aaabbb
$|w| \geq c$
$|aaabbb| \geq 6$

(S3) Divide 'w'
'w' = aaabbb
$x = aa$
$y = ab$
$z = bb$

Case 1
$|y| > 0$
$|ab| > 0$
$2 > 0$

Case 2
$|xy| \leq c$
$|aaab| \leq 6$
$4 \leq 6$

(Sh) for every $i \geq 0$ as xy^iz
Assume $i = 0$
aabb ~~regular~~ (regular)
$i = 1$ xy^1z ~~hence proved~~
aaabbb (regular)
$i = 2$ xy^2z
aaa abab bbb is not regular
hence proved.

* Context free grammar (CFG)

It consist of 4 tupls G = {V T P S} where
V is known as non-terminals
T is known as Terminals
P is known as Production rule

LHS = RHS

LHS ↓ only one single non terminal

RHS ↓ It can contain ε, terminal, non-terminal & combination of terminal & non-terminal

S is know as Start Symbole

* Problems

1. Construct a CFG which accept the string having at least 2 a's over Σ = {a, b}

Sol:-

~~2a's = baba → string's~~

S → TaTaT
T → aT | bT | ε

Assume ↓

2a's = baba → strings

(b)

S -> TaTaT

bTaTaT

bεaTaT

baTaT

babTaT

babεaT

babaT

bababT

bababε

babab

2. Generate the string 1000111 using the gramm

S -> TOOT

T -> 0T | 1T | ε

Soln:- S -> TOOT

-> 1000111

1T000111

10T000111

100T000111

1000T000111

10001T000111

100011T000111

1000111ε

1000111

* Difference b/w RG & CFG.

RG	CFG
1. It consist of 4 Tupls {VTPS} where the production rule is defined with ε, NT, T, T & NT	1. It consist of 4 tuple {VTSP} where the production rules is defined as ε, NT, T, T&NT
2. It is not suitable for Parsing.	2. It is suitable for Parsing
3. It is used to construct DFA	3. It is used to construct PDA
4. It is known as type-3 grammer.	4. It is known as Type-2 grammer.
5. RG is a subset of CFG. Every RG can be context free grammer.	5. Every CFG may not be RG.
6. Syntax of any programming lang. can't ~~bee~~ be represented completely by regular grammer.	6. Syntax of any programming lang. can be represented by CFG.

* Types of Derivation / Parse tree.

There are 2 types of trees:-

1. Left most derivation (LMD).
2. Right most derivation (RMD).

1. LMD

The derivation will be done from left most non-terminal in production rule.

2. RMD

The derivation will be done from right most non-terminal in the production rule.

Ex:-

$E \to E+E$

$E \to E*E$

$E \to a|b|c.$

String = $a+b*c$

Ans:-

LMD

E

$\downarrow$

$E+E$

$\downarrow$

$a+E$

$\downarrow$

$a+E*E$

$\downarrow$

$a+b*E$

$\downarrow$

$a+b*c$

RMD

E

$\downarrow$

$E*E$

$\downarrow$

$E*c$

$\downarrow$

$E+E*c$

$\downarrow$

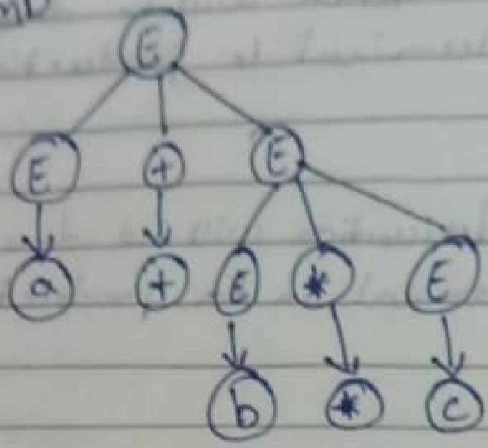
$E+b*c$

$\downarrow$

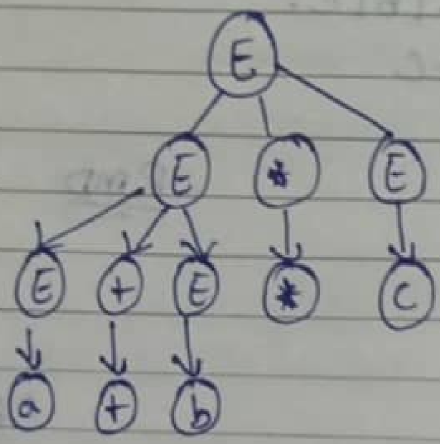
$a+b*c$

tree

LMD



RMD



2. Derive the string 1000111 by using the 2 derivation trees by using the gramm

S -> TOOT

T -> OT

T = 1T

T -> ε

Soln:- LMD

S -> TOOT

↓

1TOOT

↓

10TOOT

↓

RMD

S -> TOOT

↓

TOO1T

↓

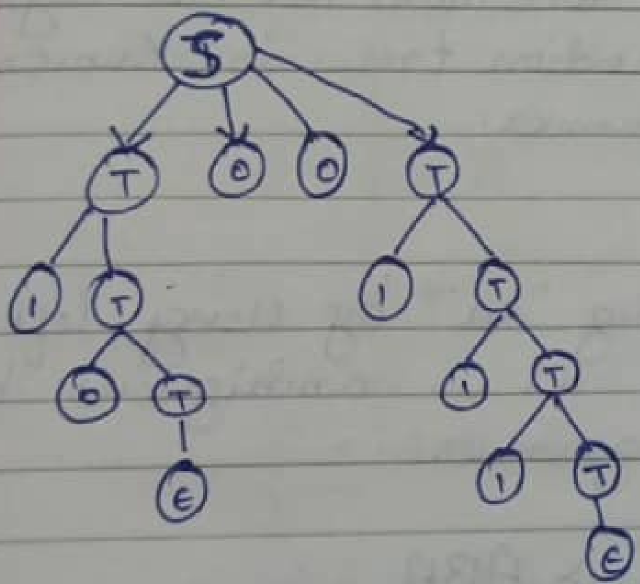
TOO11T

10ε00T
1000T
↓
10001T
↓
100011T
↓
1000111T
↓
1000111 ε
1000111

T00111T
↓
T00111 ε
T00111
↓
T00111
↓
~~100111~~
10T00111
~~10100111~~
10ε00111
↓
1000111

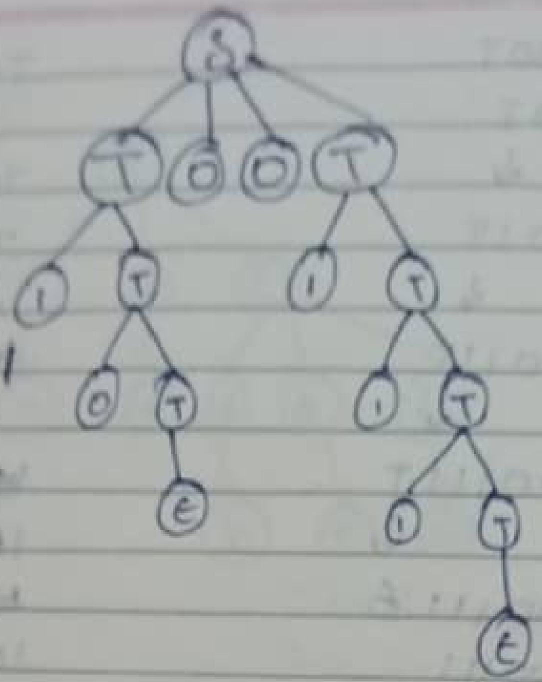
Trees

LMD.



RMD

11/08/2023



* Ambiguous Grammer.

If there is more than one left most derivation tree (or) more then one right most parse or derivation tree is known as ambigous grammer.

Ex:-

1. Generate a string 'aa' by using leftmost parse tree & prove it is ambiguous by using the following grammer.

$$S \rightarrow ABA$$
$$A \rightarrow aA \mid \epsilon$$
$$B \rightarrow bB \mid \epsilon$$

Soln:-

$S \rightarrow ABA$	ABA
$\downarrow$	$\downarrow$
$aABA$	ϵBA
$\downarrow$	$\downarrow$
$aaABA$	ϵA
$\downarrow$	$\downarrow$

E B A	a A
↓	↓
G A	a a A
↓	↓
E	E
aa	aa

In this grammer deriving the string aa by using LMD tree more them once's. Hence it is proved as ambiguous.

2. Derive the string id + id * id by using RMD & prove it is ambiguous.

$$E \rightarrow E + E$$
$$E \rightarrow E * E$$
$$E \rightarrow id.$$

Soln:-

E -> E + E	E -> E * E
↓	↓
E + E * E	E * id
↓	↓
E + E * id	E+E * id
↓	↓
E + id * id	E + id * id
↓	↓
id + id * id ///	id+id * id

hence a given string id+id * id has been proved that it has ambiguous

3. Derive the string ibtibtibtaea by Using the LMD and prove the given grammer is ambiguous.

Soln:-

S -> icts
S -> ictSeS
S -> a
c -> b

ibt ibt ibtaea

S -> ictSeS
↓
ibtSeS
↓
ibt ibtSeS
↓
ibtibtSes
↓
ibtibtictSes
↓
ibtibtibtSes
↓
ibtibt ibtaes
↓
ibtibt ibtaea

S -> ictS
↓
ibtS
~~ibt~~ ↓
ibt ictS
↓
ibtibtS
↓
ibtibt ictSes
↓
~~ibtibtibtSes~~
↓

~~ibt ibt ibt ictes~~

↓ ↓

~~ibt ibt ibt~~

ibt ibt ibt SeS

↓

ibt ibt ibt aEes

ibt ibt ibtaes

↓

ibt ibt ibtaea

* Simplification of CFG.

To reduce the productions it can be done using 3 ways.

1. Removing useless symbols.

Ex:- $S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow b$

$C \rightarrow c$ -> useless

2. Removing null Productions. or ϵ production

3. Removing Unit productions.

Ex:- $A \rightarrow C$ it is known as unit production

* Problems.

1. T -> aaB | abA | T (Removing useless symbols)
   A -> aA
   B -> ab | b
   C -> ad

Soln:-

T -> aaB | abA | T
A -> aA -> Non-terminal removed
B -> ab | b

T -> aaB | T
B -> ab | b

2. S -> AB | a
   A -> b

Soln:-

S -> AB | a
A -> b

~~S -> AB~~ | a   S -> a

* Removal of null or ε-production

1. S -> aMb
   M -> aMb
   M -> ε

Soln:- Apply 'ε' in the place of M.

S -> aMb
  -> aMb | ab
M -> aMb
  -> aMb | ab

2. $S \to XYX$
$X \to 0X/\epsilon$
$Y \to 0Y/\epsilon$

Soln:
① $X \to \epsilon$
$S \to XYX$
$\to XYX / YX / XY / Y$
$X \to 0X/\epsilon$
$\to 0X/0$

$S \to XYX / YX / XY / Y$
$X \to 0X/0$
$Y \to 0Y/\epsilon$

② $Y \to \epsilon$
$S \to XYX / YX / XY / Y$
$\Rightarrow XYX / YX / XY / Y / XX / X /$
$S \Rightarrow XYX / YX / XY / Y / XX / X$
$X \to 0X/0$
$Y \to 0Y/0.$

* Removal of Unit production.

1. $S \to XYX / YX / XY / Y / XX / X$
$X \to AB$
$Y \to C$

Soln:
$Y \to C$ is unit production.

2. S -> 0A/1B/C
A -> 0S/00
B -> A/A
C -> 01

Soln:-

S -> C   Replace C -> 01
B -> A   Replace A -> 0S/00

S -> 0A/1B/01
A -> 0S/00
B -> 1/0S/00
C -> 01.

3. S -> AC
A -> a
C -> B/d
B -> D
D -> E
E -> b

Soln:-

C -> B $\boxed{C \to b}$
B -> D   D -> b. $\boxed{B \to b}$
D -> E $\boxed{E \to b}$

S -> AC
A -> a
C -> b/d
$\boxed{\begin{array}{l} B \to b \\ D \to b \\ E \to b \end{array}}$ -> Useless Remove.

$S \to AC$
$A \to a$
$C \to b/d.$

* Removal of left Recursion

Recursion was divided into 3 types.

1. Normal recursion
2. Left "
3. Right "

1. Normal Recursion

The same non terminal on LHS appear's the same on RHS than it is called normal recursion.

Ex:- $S \to ASB$

2. Left Recursion

$S \to Sa$

3. Right Recursion

$S \to aS$

* Remove the left recursion for the following Grammar

$A \to Aa/b.$

Soln:-

$A \to bA'$
$A' \to aA'/\epsilon$

* Chomsky Normal form (CNF)

Rule 1 :- The start symbole can have Epsilon production ($S \to \epsilon$).

Rule 2 :- A non-terminal generating single terminal ($A \to b$) $\to$ CFG

($B \to ac$) $\to$ not CNF

Rule 3 :- A non-terminal generating two non terminals ($A \to BC$) $\to$ CNF

($B \to CDE$) $\to$ not CNF.

Step's to be followed in the Conversion of CFG to CNF.

S1 :- Remove start Symbole of the RHS with another non-terminal & Create new production.

Example :-

$S \to ASB$

$S_1 \to S$

$S \to ASB$

S2 :- Remove null, Unite production & useless production.

S3 :- Replace terminals on RHS with Non-terminal & Create new production.

Ex :-

$A \to aC$

$A \to XC$

$X \to a$

S4 :- Reduce the no of non terminals on RHS by Creating new production.

Ex:- $B \to CDE$

$B \to YE$

$Y \to CD$

1. Convert the following Grammar in CFG to CNF

$S \to ASB / aB$

$A \to B/S$

$B \to b/\epsilon$

Soln:-

Step 1:- Remove the start symbol by generati[ng] new non-terminal.

$S \to ASB | aB$

$S_1 \to S$

$S \to ASB | aB.$

Step 2:- Simplification of

$S_1 \to S$

$S \to ASB | aB$

$A \to B | S$

$B \to b | \epsilon$

$B \to \epsilon$

$S \to ASB | aB | AS | a$

$A \to B | S | \epsilon$

$B \to b$

$S_1 \to S$

$A \to \epsilon$

$S \to ASB | aB | AS | a$

$A \to B | S | \epsilon$

$B \to b$

$S_1 \to S$

$S \to ASB \mid aB \mid AS \mid a \mid SB \mid S \mid$
$A \to B \mid S$
$B \to b$
$S_1 \to S$

Eleminate unit productions in

$S \to ASB \mid aB \mid AS \mid a \mid SB \mid S$
$A \to B \mid S$
$B \to b$
$S_1 \to S$

$\therefore$ the unit productions are :-
$S \to S$
$A \to B$
$A \to S$
$S_1 \to S$

① $S \to S$
$S \to ASB \mid aB \mid AS \mid a \mid SB$
② $A \to B$
$A \to b$
③ $A \to S$
$S \to ASB \mid aB \mid AS \mid a \mid SB$
$A \to ASB \mid aB \mid AS \mid a \mid SB.$
④ $S_1 \to S$
$S_1 \to ASB \mid aB \mid AS \mid a \mid SB.$

$S_1 \to ASB \mid aB \mid AS \mid a \mid SB$
$S \to ASB \mid aB \mid AS \mid a \mid SB$
$A \to b \mid ASB \mid aB \mid AS \mid a \mid SB$
$B \to b$

Scanned with OKEN Scanner

Step 3:- Replace terminal with new non terminal.

$S_1 \rightarrow ASB \mid XB \mid AS \mid a \mid SB$
$S \rightarrow ASB \mid XB \mid AS \mid a \mid SB$
$A \rightarrow b \mid ASB \mid XB \mid AS \mid a \mid SB$
$B \rightarrow b$
$X \rightarrow a$

Step 4:- Reduce the no of non-terminals.

$S_1 \rightarrow YB \mid XB \mid AS \mid a \mid SB$
$S \rightarrow YB \mid XB \mid AS \mid a \mid SB$
$A \rightarrow b \mid YB \mid XB \mid AS \mid a \mid SB$
$B \rightarrow b$
$X \rightarrow a$
$Y \rightarrow AS$

* Greibach Normal form (GNF)

a. Rule's in GNF

1. Start Symbole can generate Epsclon ($S \rightarrow \epsilon$)
2. A non-terminal generating single terminal ($A \rightarrow b$)
3. A non-terminal generating terminal followed by any no of terminals & non-terminals.
($A \rightarrow aACbDEfGH$)

b. Step's to Convert CFG to GNF

S1:- CFG is in CNF
S2:- Re-Name non-terminal with numeric Variables

Scanned with OKEN Scanner

in ascending order in which they appear.

$$\left\{ \begin{array}{l} A \to BC \\ A_1 \to A_2 A_3 \end{array} \right\}$$

S3:- Consider A_i generates A_j where $i<j$, $i>j$ apply substution method.

$i=j$ means apply left recursion.

S4:- Remove left recursion.

S5:- Make the production's following the GNF rule's.

Example:-

1. $S \to CA \mid BB$

$B \to b \mid SB$

$C \to b$

$A \to a$.

Soln:-

S1:- The grammar given is in CNF.

S2:- Re-name non-terminals to numeric Variable's.

$S \to CA \mid BB \Rightarrow A_1 \to A_2 A_3 \mid A_4 A_4$

$B \to b \mid SB \Rightarrow A_4 \to b \mid A_1 A_4$

$C \to b \Rightarrow A_2 \to b$

$A \to a \Rightarrow A_3 \to a$.

S3:- Consider 'i' value & 'j' Value.

$A_1 \to A_2 A_3 \mid A_4 A_4$

$A_4 \to b \mid A_1 A_4$

$A_2 \to b$

$A_3 \to a$

Case 1:-

$i > j$

$A_4 \to b \mid A_1 A_4$

Apply substution method by substuting

A_1
$A_4 \to b \mid A_2 A_3 A_4 \mid A_4 A_4 A_4$

Apply substitution method by substituting A_2
replace $A_2 \to b$.

$A_4 \to b \mid b A_3 A_4 \mid A_4 A_4 A_4$

Apply substitution method by substituting A_3
replace $A_3 \to A_3$

$A_4 \to b \mid b A_3 A_4 \mid A_4 A_4 A_4.$

$i = j$ Apply left recursion.

$A_4 \to bZ \mid b A_3 A_4 Z \mid$
$Z \to A_4 A_4 Z / \epsilon$

Remove $Z \to \epsilon$

$A_4 \to b\epsilon \mid b A_3 A_4 \epsilon \mid b \mid b A_3 A_4$
$Z \to A_4 A_4 Z / A_4 A_4.$

∴ Substitute A_4 in the Z

$\Rightarrow$ $A_4 \to bz / b A_3 A_4 z / b / b A_3 A_4$
$Z \to A_4 A_4 Z / A_4 A_4$

$\Rightarrow Z \to \underline{bZA_4}Z \mid \underline{bA_3A_4}ZA_4Z \mid \underline{b}A_4Z \mid \underline{bA_3A_4}A_4Z$
$\underline{bZ}A_4 \mid \underline{bA_3A_4}ZA_4 \mid \underline{b}A_4 \mid bA_3A_4A_4$

This is in GNF

Scanned with OKEN Scanner

$i < j$

$A_1 \rightarrow A_2 A_3 / A_3 A_4$

apply substitution method by A_2

$A_1 \rightarrow b A_3 / A_4 A_4$

$A_1 < A_4$

substitue A_4 in A_1

$A_1 \rightarrow bA_3 / bZA_4 / bA_3 A_4 Z A_4 / bA_4 /$
$b / A_2 A_4 A_4$

This in GNF.

* Sentential form. (Section-A)

It produces string that has special rule from the stack symbole in definition.

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

$(V \cup T)^* / S \Rightarrow \omega \rightarrow$ string

If the $S \overset{*}{\underset{lm}{\Rightarrow}} \omega$ it is Known as left most sentential form.

If the $S \overset{*}{\underset{Rm}{\Rightarrow}} \omega$ it is Known as Right most sentential form.

Ex:-

1. Consider the grammer $E \rightarrow E+E$
$E \rightarrow E*E$
$E \rightarrow id / \epsilon$

Using left & Right sentential form & derive the string 'ω' as $id * id + id$.

Solu:-

$E \overset{*}{\underset{Lm}{\Rightarrow}} E*E$ (Left sententia

$\overset{*}{\underset{Lm}{\Rightarrow}} id * E$

$\overset{*}{\underset{Lm}{\Rightarrow}} id * E + E$

$\overset{*}{\underset{Lm}{\Rightarrow}} id * id + E$

$\overset{*}{\Rightarrow} id * id + id$

(Right sentitable)

$E \xRightarrow[Rm]{*} E + E$

$E \xRightarrow[Rm]{*} \boxed{E} + id$

$E \xRightarrow[Rm]{*} \boxed{E} * E + id$

$E \xRightarrow[Rm]{*} E * id + id$

$E \xRightarrow[Rm]{*} id * id + id.$

Scanned with OKEN Scanner