AUTOMATA THEORY & COMPILER DESIGN

branch of computer sci
→ Abstract Mathematical model.
- → Finite automata
- → Push Down automata
- → Linear Bounded
- → Turing machine.

- Designing & checking the behaviour of logic
- Designing compiler circuits.
- scan large bodies of text.

Grammar → Type 0 ⇒ Turing Machine
Type 1 ⇒ Linear Bounded
" 2 ⇒ Push down automata
" 3 ⇒ Finite automata.

Language
REL, CSL, CFL, RL
- REL → Recursive enumerable lang.
- CSL → Context sensitive lang.
- CFL → Context free lang.
- RL → Regular lang.

Compiler → translates high level lang to low level lang.
(Goes through 7 phases of compiler).

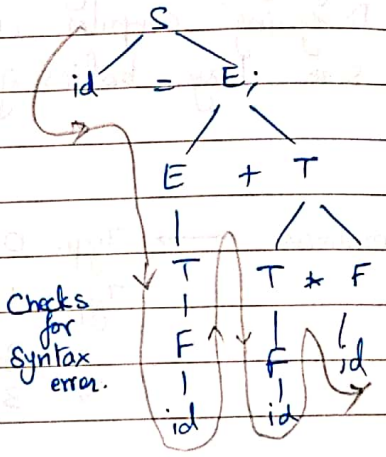
Automata theory concept
- Lexical Analysis (FE) → Syntax analysis (PDA)
- Syntax analysis → Semantic analysis
- Semantic analysis → Intermediate code generation (CFG)
- Intermediate code generation → Code optimization
- Code optimization → Target code generation (low level lang.)

Grammar = { V (or N), T, P, S }

V (or N) → Non terminals
T → Terminals
P → Production rules
S → Start Symbol.

S → id = E;
E → E + T | T
T → T * F | F
F → id

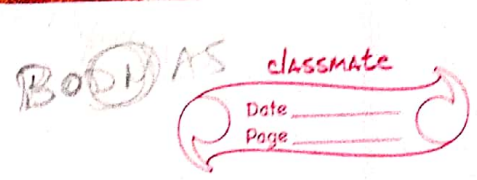
Parse Tree

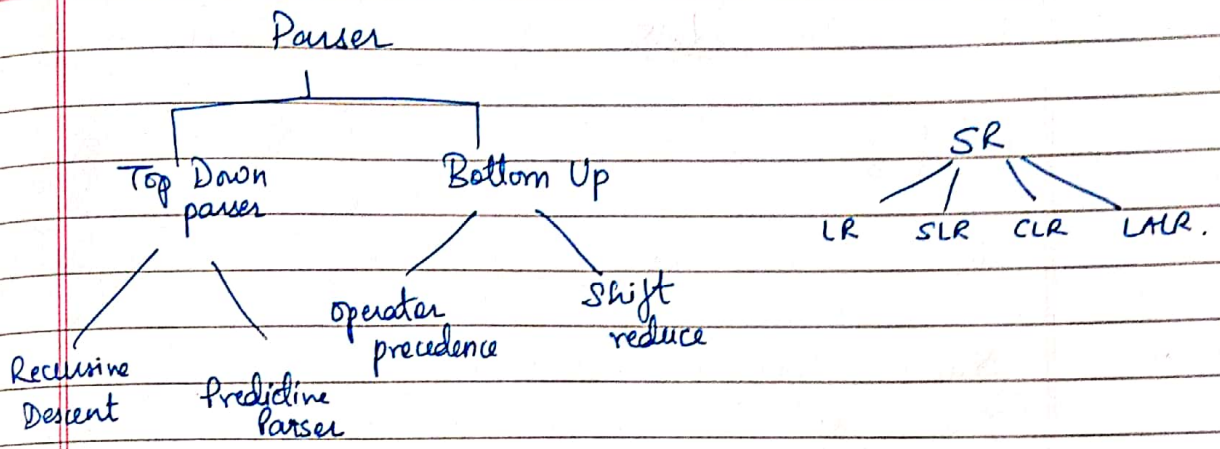


Q. Compiler vs Interpreter.

Compiler	Interpreter
• Translates HLL to LLL in 7 phases	• Translates HLL to LLL in 4 phases
• Compiles whole pgm at once & reports error at the end	• Compiles line by line & reports error as soon as it is encountered.
• Fast	• Slow.

{%% %%}





lex program → has 3 sections

① Definition : Initialization

② Rule Section : RE

③ Subroutine section : main()
{ yylex();
}

↓

generates lex.yy.c

yacc
↓
y.tab.c
y.tab.h

Lex

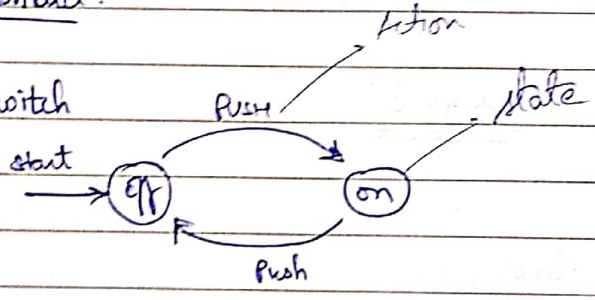
Sudo apt-get install bison
Sudo apt-get install flex
sudo apt-get install byacc

* Automata

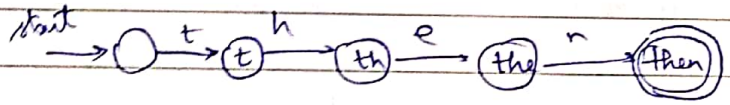
Study of abstract computing devices or "machines".

Finite automata:

- on/off switch



- Modelling word recognition



→ rewritten as
⇒ derived

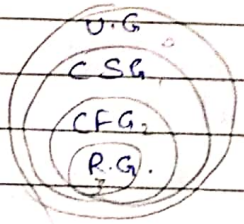
Derivation
→ Leftmost
→ Rightmost.

Types of Grammars

Type 0	Type 1	Type 2	Type 3	
Unrestricted phase structured grammar	Context Sensitive grammar	Context free grammar	Regular grammar	
$\alpha \rightarrow \beta$	$\alpha \rightarrow \beta$	$A \rightarrow \alpha$	Right linear	Left linear
No restriction	$\lvert\alpha\rvert \leq \lvert\beta\rvert$			
β may be ϵ		(left side should have only 1 symbol)	$A \rightarrow xB \mid x$	$A \rightarrow Bx \mid x$
α should have 1 non-Terminal				

- Regular Grammar
- Linear Grammar — middle linear grammar.

Chomsky Hierarchy.



* Diff b/w Empty & Null String.

$|\epsilon| = 0$.

Power of an alphabet.

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

$\Sigma^1 = \{a, b, c\}$ $\Sigma^2 = \{aa, ab, bc, bb, ba, cb ..\}$

$\Sigma^3 = \{aaa, aab, aac, abb, ... \}$

~~$\Sigma^* \quad \Sigma^0 (\Theta)$~~

$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup ..$

$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup ..$

q 1) $N = \{S\}$ $T = \{a, b\}$

$P = \{$ 1. $S \rightarrow aSb$
2. $S \rightarrow ab \}$

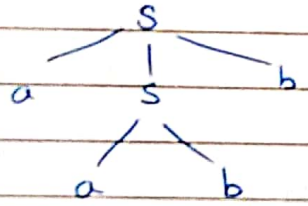
What is the language derived from this ?

$S \xRightarrow{2} ab$

$S \xRightarrow{1} aSb$
$\xRightarrow{2} aabb$

$L(G) = \{ ab, aabb, aaabbb \ldots \}$
$= \{ a^n b^n \mid n \in [1, \infty) \}$

Derivation Tree



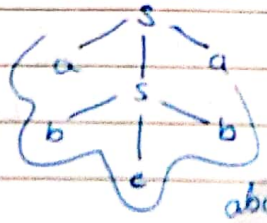
Grammar Type
Type 1, Type 2.

2) $N = \{S\}$ $T = \{a, b, c\}$
$P = \{$ 1. $S \rightarrow aSa$
2. $S \rightarrow bSb$
3. $S \rightarrow c$ Type 2

$S \Rightarrow c$
$S \Rightarrow aca$
$S \Rightarrow bcb$
$S \Rightarrow aacaa$
$S \Rightarrow abcba$
$S \Rightarrow bacab$

$L(G) = \{ wcw^R \mid w \in \{a, b\}^* \}$

Derivation tree



3) $N = \{S\}$ $T = \{a, b\}$

$P = \{ 1.\ S \to aS$

$2.\ S \to bS$ Type 3

$3.\ S \to a$

$4.\ S \to b \}$

$L(G) = \{ a, b, aa, ab, ba, bb, \ldots \}$

$= \{a, b\}+$

Regular grammar can be expressed as Regex.

Regular Expression

└ pattern Matching

1) $\hat{\ }_{\epsilon}, \phi$

2) $R_1 \quad R_2 \quad R_1 + R_2$

3) $R_1 \quad R_2 \quad R_1 . R_2$

4) $(R)^*$

If R is R.E then

L(R) is Regular lang.

ϕ —— $L(\phi) = \phi$

ϵ —— $L(\epsilon) = \{\epsilon\}$

a —— $L(a) = \{a\}$

$R_1 + R_2$ —— $L(R_1) \cup L(R_2)$

$R_1 . R_2$ —— $L(R_1) \cdot L(R_2)$

R^* —— $L(R)^*$

* Diff b/w ∈ & ϕ.

Identities of R.E

(1) $\phi + R = R$

(2) $\phi \cdot R + R \cdot \phi = \phi$

(3) $\epsilon \cdot R = R \cdot \epsilon = R$

(4) $\epsilon^* = \epsilon$ and $\phi^* = \epsilon$

Machines

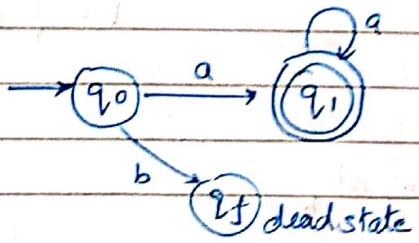
1. Finite Automata — DFA / NFA – Temporary storage
2. Push Down automata
3. Linear Bounded
4. Turing Machine — Alan Turing

DFA (Deterministic Finite Automata)

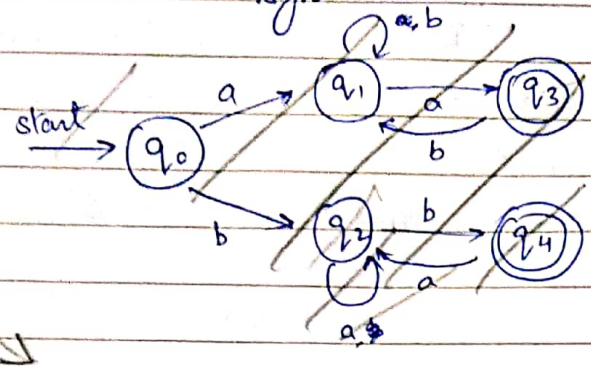
$\{ Q, \Sigma, \delta, q_0, F \}$

- Q : set of final states
- Σ : i/p
- δ : transition fun.
- q_0 : start state
- F : final state.

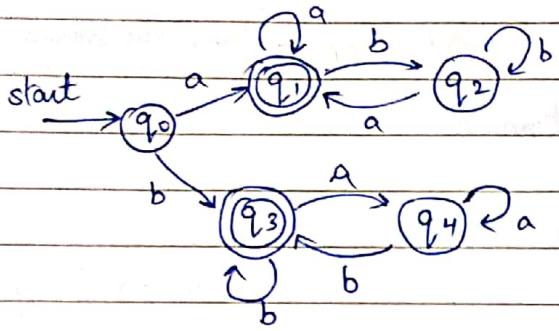
a^*



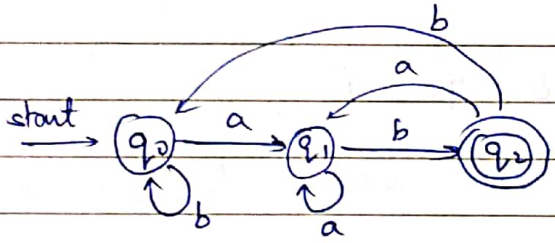
Q. Design a DFA over $\Sigma = \{a, b\}$ such that every string accepted must start & end with same symbol.



$L = \{a, b, aa, bb, aba \ldots\}$

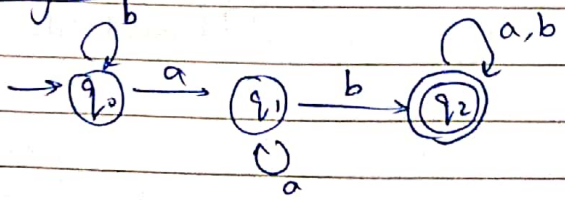


Q Ends with ab.

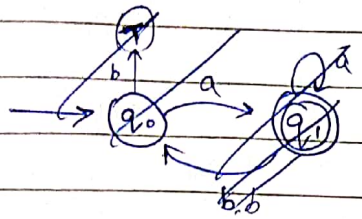


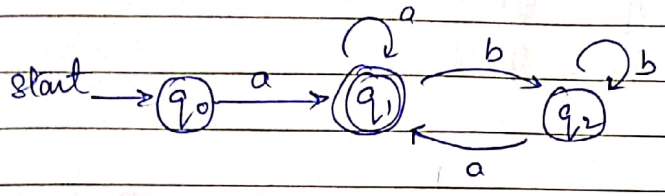
$L = \{ab, aab, baab, \ldots\}$

Q Substring ab

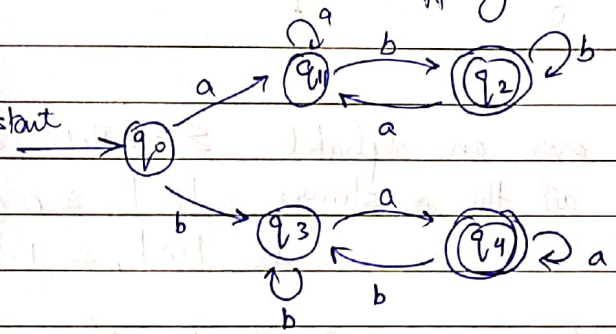


Q. Start & end with a





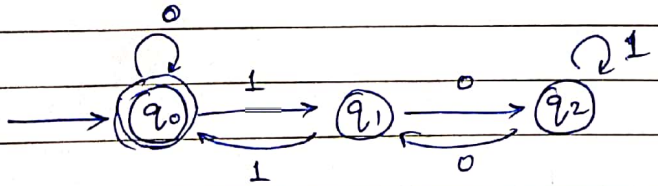
Q. Start & end with diff. symbols.



Q Construct a DFA which accepts set of all strings over $\{0,1\}$ which when interpreted as binary is divisible by 3.

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

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



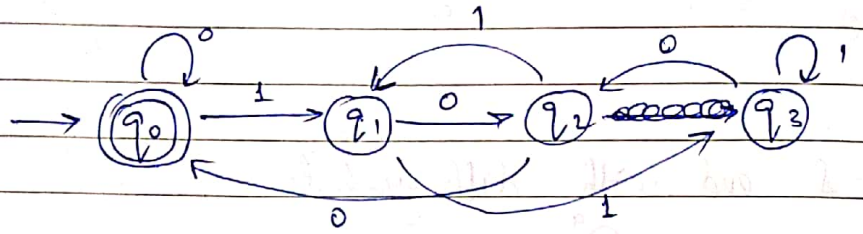
reminder: 0 | 1 | 2

0 | 01 | 10

11 | 111 | 101

Q. for divisible by 4.

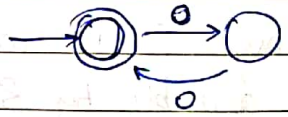
		0	1
0	→ q_0 *	q_0	q_1
1	q_1	q_2	q_3
2	q_2	q_0	q_1
3	q_3	q_2	q_3



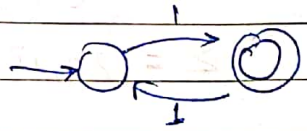
Q. Design a DFA over an alphabet $\Sigma = \{0,1\}$ such that it accepts all the strings $|w|_0 \equiv 0 \pmod 2$ && $|w|_1 \equiv 1 \pmod 2$

≡ there should be even no. of 0's & odd no. of 1's

DFA for even no. of 0's



DFA for odd no. of 1's



19/8/19

classmate
Date
Page

Epsilon - NFA

$(Q, \Sigma, \delta, q_0, F)$

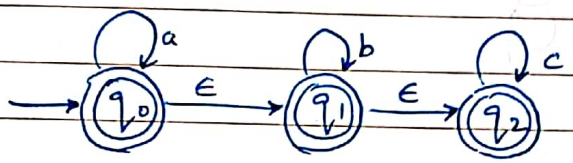
$\delta : Q \times \Sigma \cup \epsilon \rightarrow 2^Q$ (Epsilon - NFA)

$Q \times \Sigma \rightarrow 2^Q$ (NFA)

$Q \times \Sigma \rightarrow Q$ (DFA)

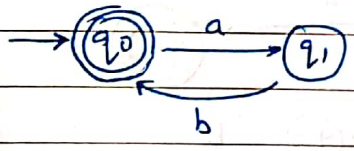
eg: $a^* b^* c^*$

$L = \{ a^n b^m c^i \mid n, m, i \geq 0 \}$

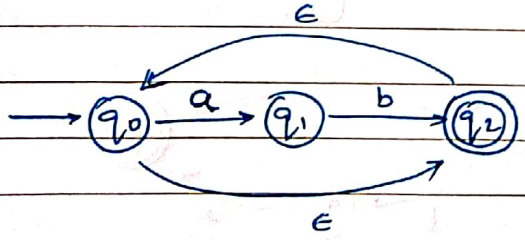


eg: $(ab)^*$

$L = \{ (ab)^n \mid n \geq 0 \}$



} NFA

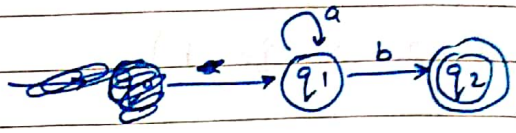


} ϵ-NFA

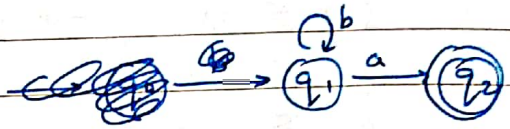
ab } both have length = 2.
aϵb }

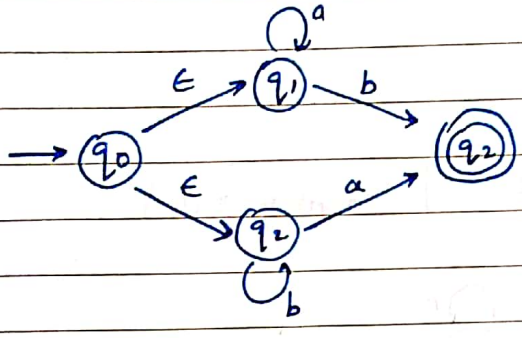
eg: $L = \{ a^n b \cup b^n a \mid n \geq 0 \}$

for $a^n b$ ⇒

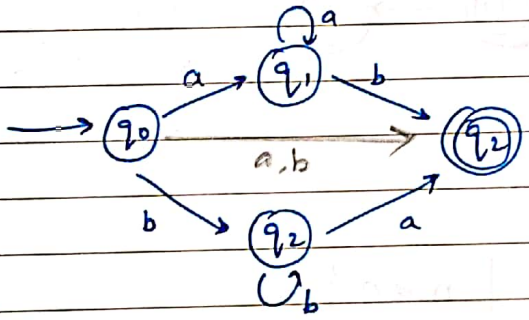


for $b^n a$ →

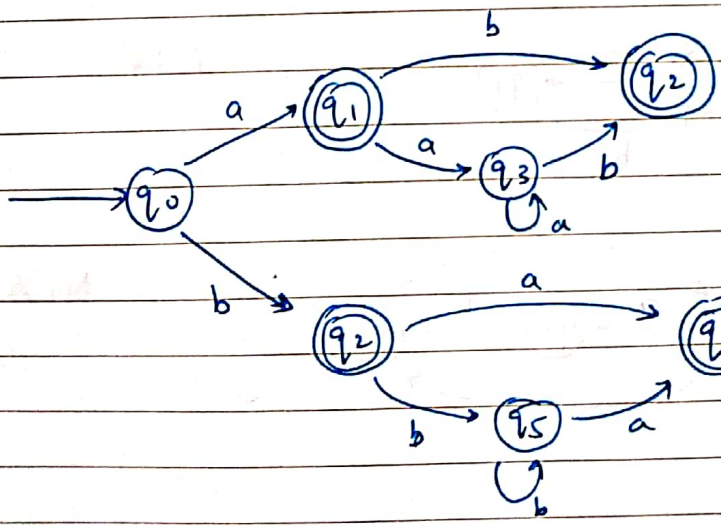




ϵ - NFA



NFA



DFA

∈-NFA to NFA

∈-closure :

∈-closure of q_0 = { q_0, q_1, q_2 }

∈-closure of q_1 = { q_1, q_2 }

∈-closure of q_2 = { q_2 }

$\in^* a \in^*$

q_0	q_0	q_0	q_0, q_1, q_2
	q_1	ϕ	ϕ
	q_2	ϕ	ϕ

$\in^* b \in^*$

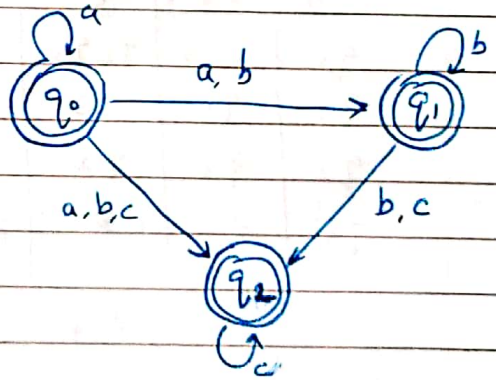
q_0	q_0	ϕ	ϕ
	q_1	q_1	q_1, q_2
	q_2	ϕ	ϕ

$\in^* c \in^*$

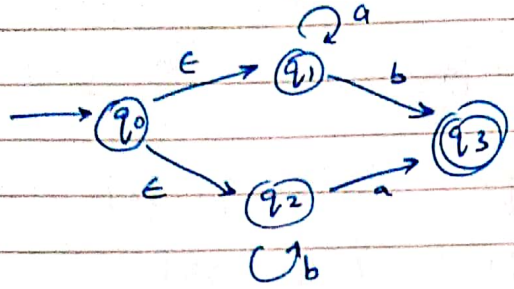
q_0	q_0	ϕ	ϕ
	q_1	ϕ	ϕ
	q_2	q_2	q_2

	a	b	c
q_0	q_0, q_1, q_2	q_1, q_2	q_2
q_1	ϕ	q_1, q_2	q_2
q_2	ϕ	ϕ	q_2

Transition diagram.



eg: Convert the following ϵ-NFA to NFA.



ϵ-closure of q_0 = $\{q_0, q_1, q_2\}$

of q_1 = $\{q_1\}$

of q_2 = $\{q_2\}$

of q_3 = $\{q_3\}$

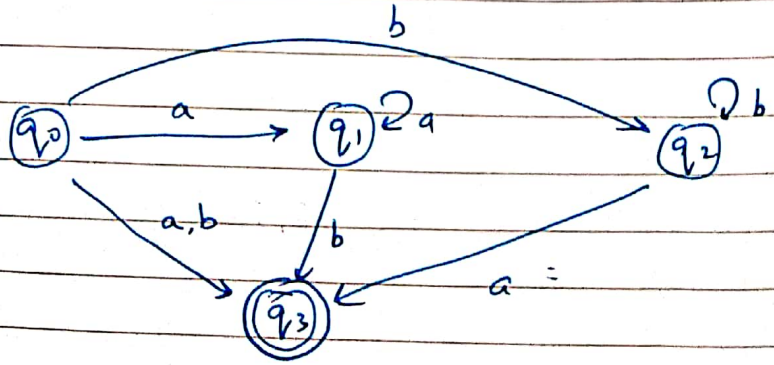
$\epsilon^* a \epsilon^*$

q_0	q_0	ϕ	ϕ
	q_1	q_1	q_1
	q_2	q_3	q_3

$\epsilon^* a \epsilon^*$

q_1	q_0	q
	q_1	
	q_2	

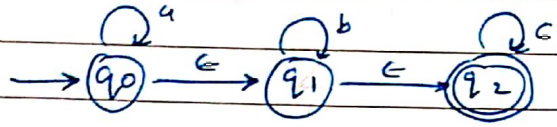
	a	b
q_0	$q_1 q_3$	$q_2 q_3$
q_1	q_1	q_3
q_2	q_3	q_2
q_3	ϕ	ϕ



∈-NFA to DFA

m-1 ∈-NFA → NFA → DFA.

m-2 ∈-closure.



A = ∈-closure of q_0 = { q_0, q_1, q_2 }

$\delta(A, a)$ = ∈-closure ({ q_0, q_1, q_2 }, a)
= ∈-closure ($(q_0, a) \cup (q_1, a) \cup (q_2, a)$)
= ∈-closure (q_0) = { q_0, q_1, q_2 } = A.

$\delta(A, b)$ = ∈-closure ({ q_0, q_1, q_2 }, b)
= ∈-closure ($(q_0, b) \cup (q_1, b) \cup (q_2, b)$)
= ∈-closure (q_1) = { q_1, q_2 } = B

B = ∈-closure of q_1 = { q_1, q_2 }

$\delta(B, a)$ = ∈-closure ({ q_1, q_2 }, b)
= ∈-clos ($(q_1, b) \cup (q_2, b)$)
= ∈-closure (

$\delta(A, c)$ = ∈-closure ({ q_0, q_1, q_2 }, c)
= ∈-closure (q_3) = C

B = ∈-closure of q_1 = { q_1, q_2 }

$\delta(B, a)$ = ∈-closure ({ q_1, q_2 }, ∅)
= ∈-closure ($(q_1, \emptyset) \cup (q_2, \emptyset)$)
= ∈-closure (∅, ∅)
= ∅

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

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

$= \epsilon\text{-closure}(q_1, \phi)$

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

$= B$

$\delta(b, c) = C$

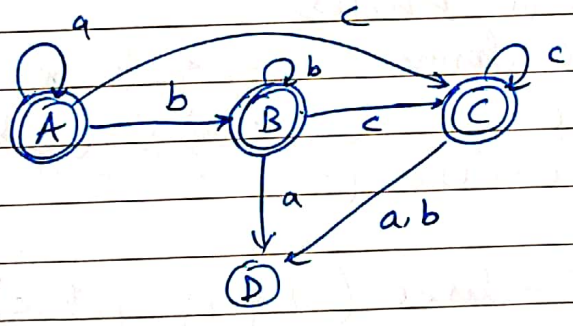
$\delta(C, a) = \phi$

$\delta(C, b) = \phi$

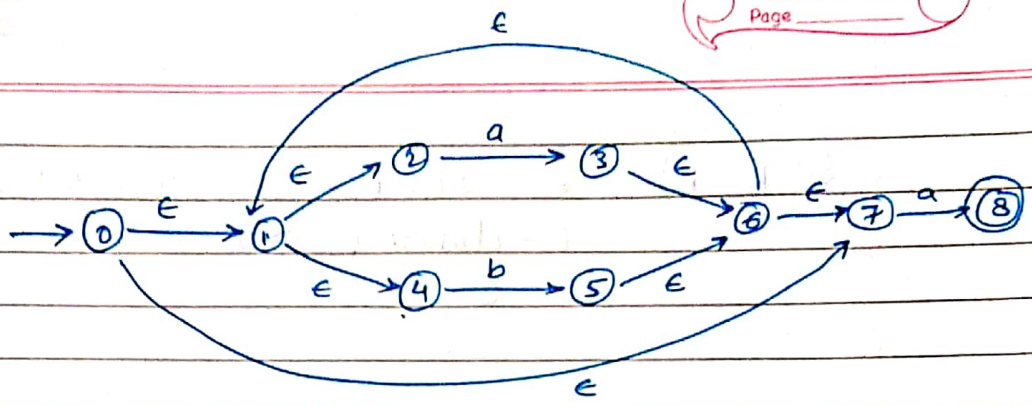
$\delta(c, c) = C$

	a	b	c
A	A	B	C
B	ϕ	B	C
C	ϕ	ϕ	C

$\phi \equiv$ D state



Ex:



A = ε-closure of start state i.e 0.

ε-closure (0)

= {0, 1, 2, 4, 7}

$\delta(A, a)$ = ε-closure ({0, 1, 2, 4, 7}, a)

= ε-closure ((0, a) ∪ (1, a) ∪ (2, a) ∪ (4, a) ∪ (7, a))

= ε-closure (∅ ∪ ∅ ∪ 3 ∪ ∅ ∪ 8)

= ε-closure (3 ∪ 8)

= {(3, 6, 1, 2, 4, 7) ∪ (8)}

= {1, 2, 3, 4, 6, 7, 8} = B

$\delta(A, b)$ = ε-closure ({0, 1, 2, 4, 7}, b)

= ε-closure ((0, b) ∪ (1, b) ∪ (2, b) ∪ (4, b) ∪ (7, b))

= ε-closure (∅ ∪ ∅ ∪ ∅ ∪ 5 ∪ ∅)

= ε-closure (5)

= {5, 6, 1, 2, 4, 7} = C

	a	b	c
A	A	B	A

~~B = ε-closure~~

D = ε-closure (1)

= {1, 2, 4}

Since we get B & C as new states from A, we check states possible from B & C as well.

$\delta(B, a)$ = ε-closure ({1, 2, 3, 4, 6, 7, 8}, a)

= ε-closure (∅ ∪ 3 ∪ 9 ... 8)

= ε-closure (3, 8)

= B

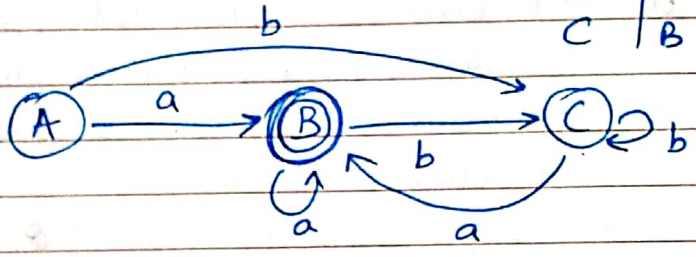
$\delta(B, b) = \epsilon\text{-closure}(\{1,2,3,4,6,7\}, b)$
$= \epsilon\text{-closure}(5)$
$= C$

$\delta(C, a) = \epsilon\text{-closure}(\{1,2,4,5,6,7\}, a)$
$= \epsilon\text{-closure}(3,8)$
$= B$

$\delta(C, b) = C$

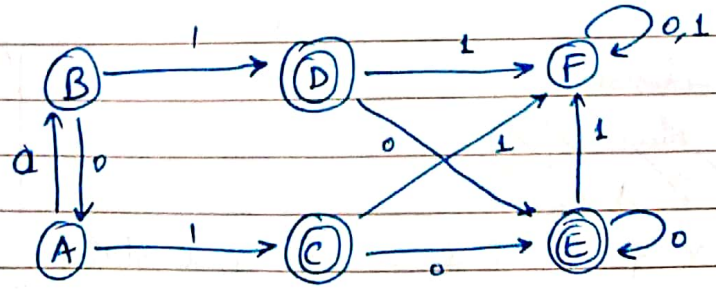
Transition table

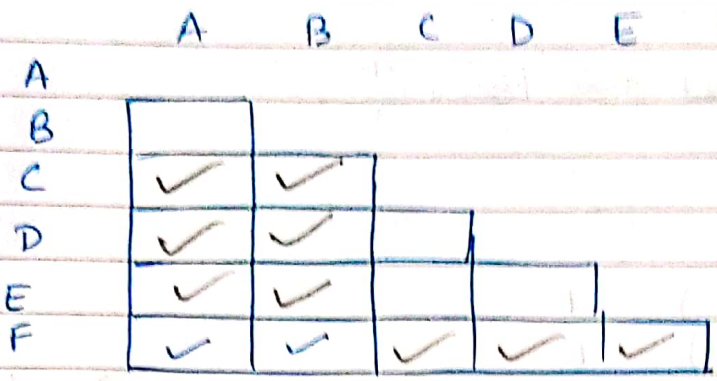
	a	b
A	B	C
B	B	C
C	B	C



Minimization of DFA

① Draw a table for all pairs of states (P, Q)





② * Mark all pairs where P∈ final & Q ∉ final

③ If there are any unmarked pairs (P,Q) such that [δ(P,x), δ(Q,x)] is marked, then mark (P,Q) where x is an input symbol.

Q: B,A

δ(B,0) = A
δ(A,0) = B } (A,B) Not marked

δ(B,1) = D
δ(A,1) = C } can't mark (B,A) coz (D,C) not marked.

D, C

δ(D,0) = E
δ(C,0) = E } (E,E) Check.

δ(D,1) = F
δ(C,1) = F } (F,F)

E, C

δ(E,0) = E
δ(C,0) = E
δ(E,1) = F
δ(C,1) = F } ∅

E, D

~~Kuch aur hoga~~ .

E
E
F
F

F, A

$\delta(F,0) = F$
$\delta(A,0) = B$
$\delta(A,1) = C$ } marked in table ⇒ Mark (F, A)
$\delta(F,1) = F$

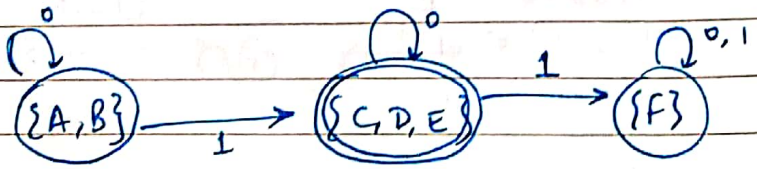
F, B

$\delta(F,0) = F$
$\delta(B,0) = A$
$\delta(F,1) = F$ } Marked ⇒ Mark (F, B).
$\delta(B,1) = D$

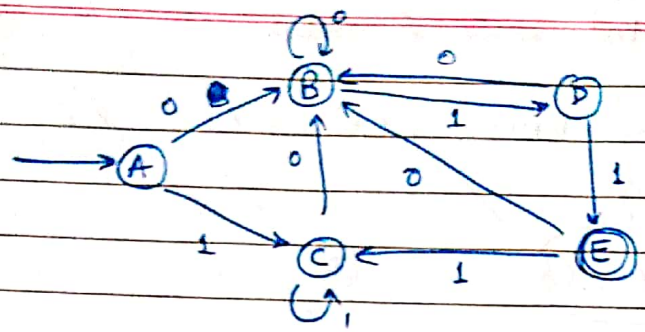
Equivalent State (P, Q)

$\delta(P,w) \neq F$
$\delta(Q,w) \neq F$

Unmarked States ⇒ (B, A) , (D, C) , (E, C) , (E, D)



Ex:



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

* All pairs.

(A, B)

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

} None marked

(A, D)

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

} Marked. Mark (A, D)

(A, C)

$\delta(A,0) = B$
$\delta(C,0) = B$
$\delta(A,1) = C$
$\delta(E,1) = C$

} None marked

(B, D)

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

✓ Mark (B, D)

(B, C)

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

} None marked

(C, D)

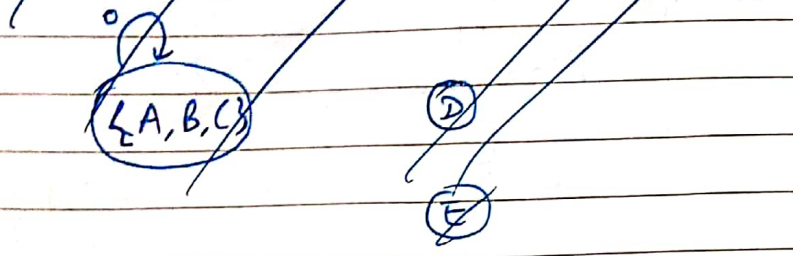
$\delta(C,0) = B$
$\delta(D,0) = B$
$\delta(C,1) = C$
$\delta(D,1) = E$

✓ Mark (C, D)

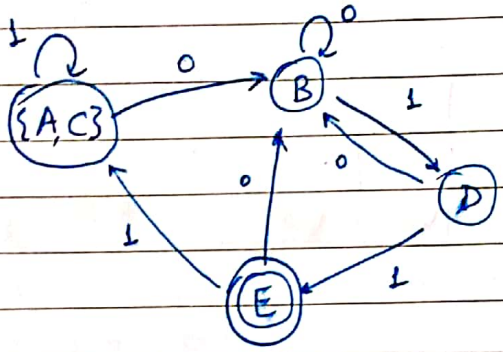
Second Iteration required.

* Since (C, D) is marked, (A, B) & (C, B) should be checked again & marked.

Unmarked pairs (A,B), (A,C), (B,C)



Unmarked pairs : (A,C)



Finite Automata (FA) without o/ps.

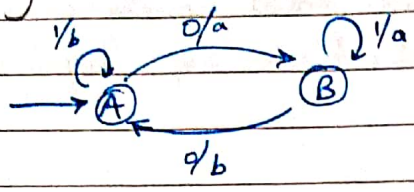
mealy & moore machines
↓
No final state.

- Mealy Machine

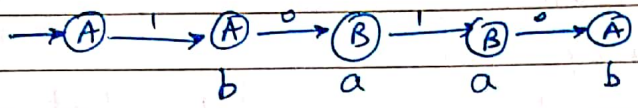
$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$

- Q : finite state
- Σ : i/p symbol
- Δ : o/p symbol
- δ : transition fun. $Q \times \Sigma = Q$
- λ : output fun. $\Sigma \times Q = \Delta$
- q_0 : start state

Mealy



eg: 1010



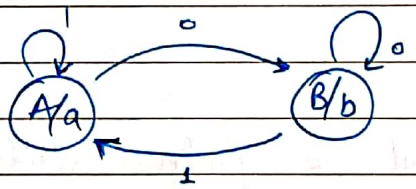
⇒ baab

*(output length = input length)

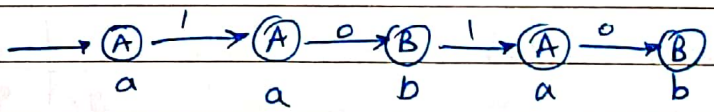
Moore Machine

Only o/p changes : $Q \rightarrow \Delta$

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



eg: 1010



⇒ aabab

*(output length = I/P length + 1)

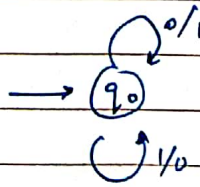
Q. Construction of a mealy machine that produces 1's complement of any binary input string.

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

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

$Q = q_0$

$q_0 = q_0$

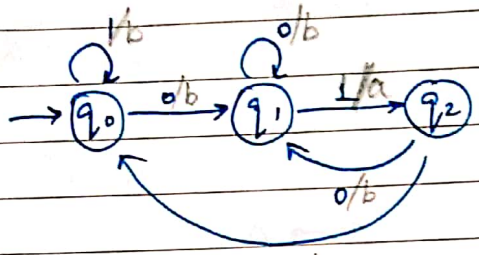


Q. Construct a mealy machine that prints 'a' whenever sequence '01' encountered in any input binary string.

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

~~$\Delta = \{0, 1, a\}$~~

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



first, construct 0,1 seq for DFA machine.

at 01 → a
remaining → b

eg: 101010
bbabab
01 01

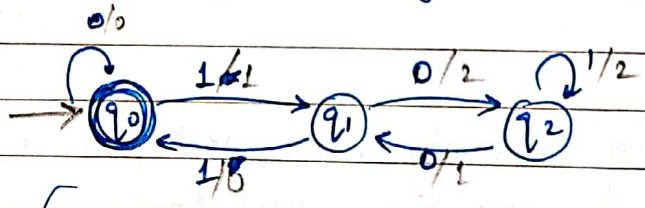
Q. binary nos. as input & produces residue modulo '3' as o/p

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

$\Delta = \{0, 1, 2\}$

	0	1
q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

DFA machine for divisible by '3'.



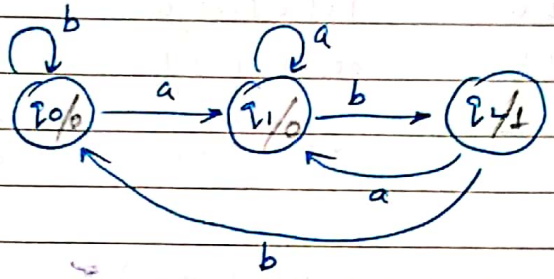
No end state.

eg: 101011
o/p → 122101

Q. Construction of moore machine whenever 'ab' encountered it should print '1'.

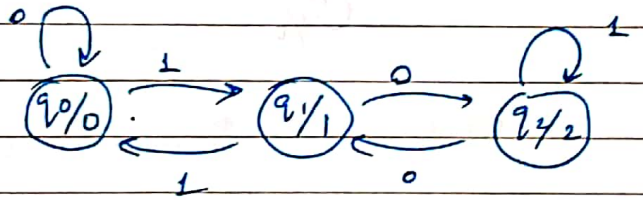
$\Sigma = \{a, b\}$

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



* Construct DFA first & then change the node's state.

Q. input → binary nos.
out → residue modulo '3'.

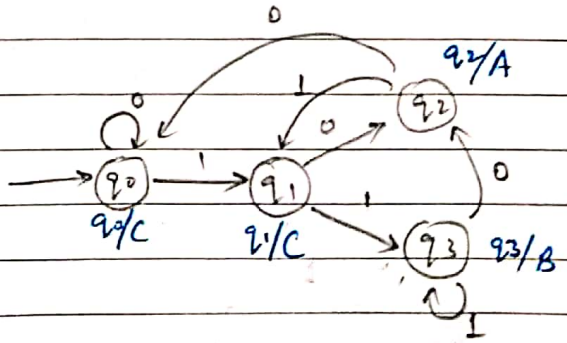


Q. Moore
input: all strings over $\{0, 1\}$
output: produces 'A' as o/p if i/p ends with '10' or produces 'B' as o/p if i/p ends with '11'. otherwise C.

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

$\Delta = \{A, B, C\}$

10 → A
11 → B
C

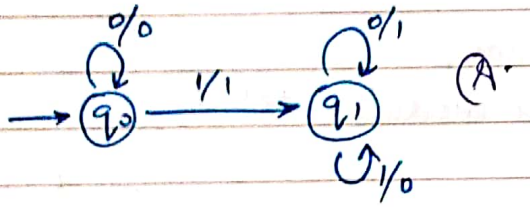


Q. Construct a mealy m/c that takes binary no. as i/p & produces 2's complement of that no. as o/p. Assume that string is read LSB to MSB and end carry is discarded.

MSB ← LSB

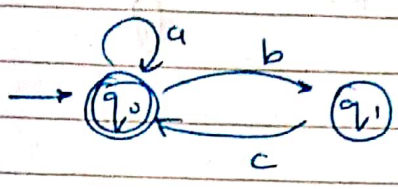
1100	11101100	1011
0011	00010011	0100
+ 1	+ 1	+ 1
0100	00010100	0101

* from LSB to MSB (←). Keep all zeroes & one 1, & then invert all bits

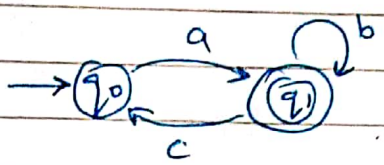


(A.

Conversion from DFA, NFA, ε-NFA to RE

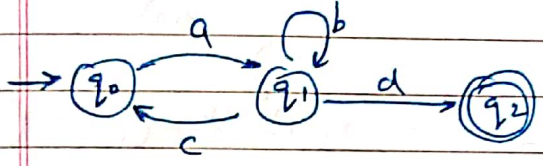


$(a|(bc)^*$

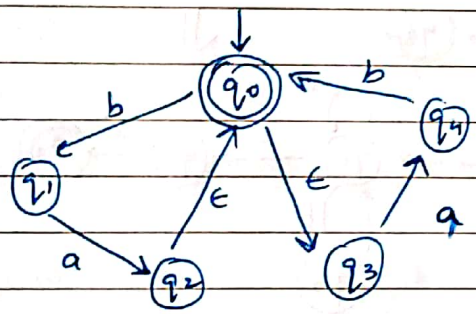


~~$(a|c|ac|abc)$~~

$a(b|ca)^*$



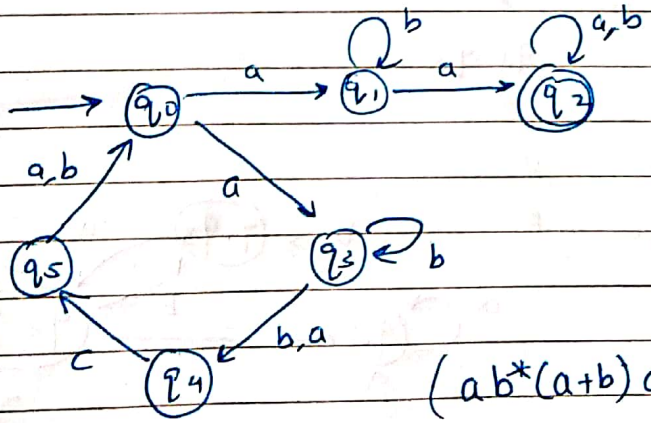
$a(b|ca)^* d$



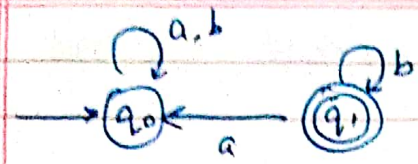
~~$(b|a)$~~

$((ba\epsilon)^* + (\epsilon ab)^*)^*$

$\rightarrow (ba|ab)^*$

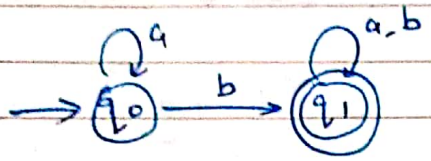


$(ab^*(a+b)c(a+b))^* ab^*a(a+b)^*$

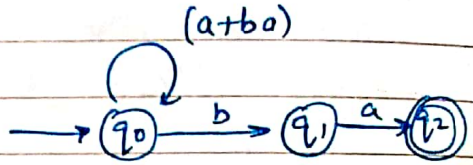


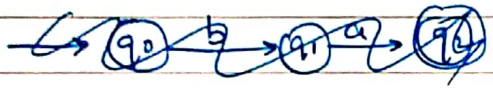
RE → $\emptyset$ (8.

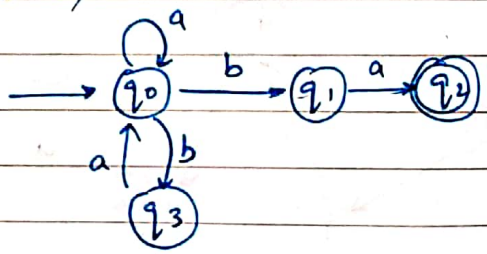
⇒ $a^*b(a+b)^*$



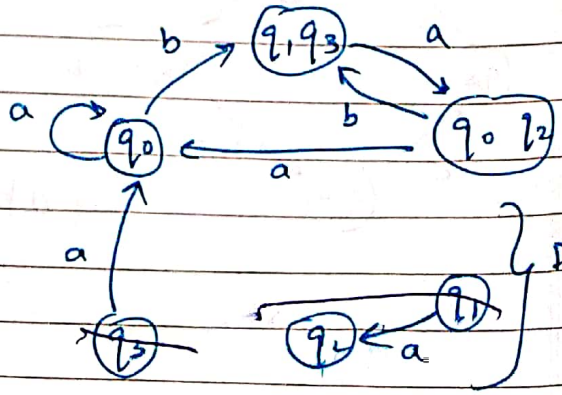
⇒ $(a+ba)^*ba$



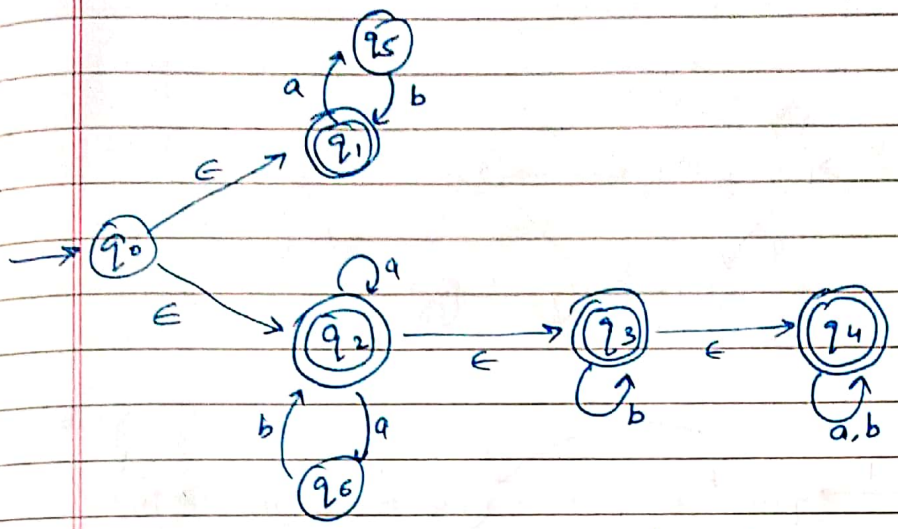




	a	b
q_0	q_0	q_1, q_3
$q_1 q_3$	$q_2 q_0$	$\emptyset$
$q_0 q_2$	q_0	q_1, q_3
q_1	q_2	$\emptyset$
q_2	$\emptyset$	$\emptyset$
q_3	q_0	$\emptyset$

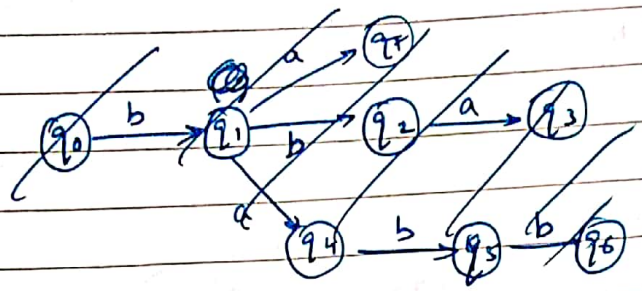


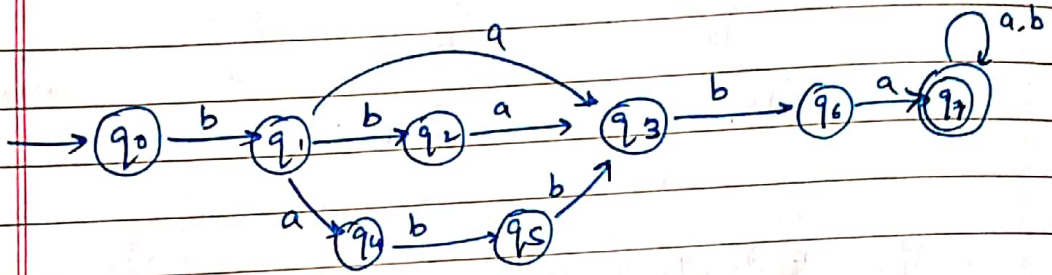
⇒ $(ab)^* + (a + ab)^* b^* (a+b)^*$



	a	b
q_0	q_5, q_2, q_6, q_4, q_3	q_3, q_4
$q_2 q_3 q_4 q_5 q_6$		

⇒ $b(a + ba + abb)\ (ba(a+b)^*)$



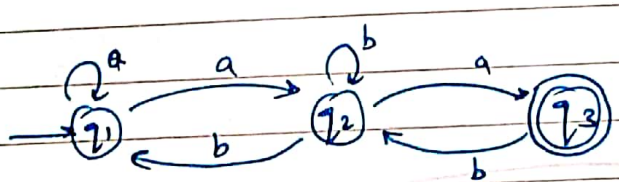


Arden's Theorem

ϵ-NFA / DFA → RE.

If P and Q are two Regular Expressions and if P does not contain ϵ, then the following eqn in R given by $R = Q + RP$ has a unique sol $R = QP^*$

eg:



add ϵ to incoming node.

initial state → $q_1 = q_1 \cdot a + q_2 \cdot b + \epsilon$

$q_2 = q_2 \cdot b + q_3 \cdot b + q_1 \cdot a$

$q_3 = q_2 \cdot a$

} Incoming edges.

$q_2 = q_2 \cdot b + q_3 \cdot b + q_1 \cdot a$

$= q_2 \cdot b + (q_2 \cdot a) b + q_1 \cdot a$

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

$R = R(P) + Q$

$\Downarrow$

Sol$^n \Rightarrow R = QP^*$

$$q_2 = q_1 \cdot a (b + ab)^* \quad - (1)$$

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

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

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

$R = R(P) + Q$

$\therefore \quad q_1 = (a + a(b+ab)^* b)^* \quad - (2)$

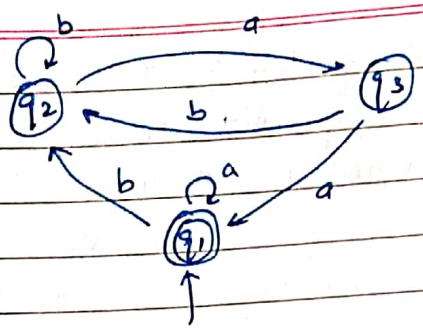
from (1) & (2)

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

$\therefore \quad q_3 = q_2 \cdot a$

$$\Rightarrow \boxed{q_3 = (a + a(b+ab)^* b)^* a (b+ab)^* a} \quad - (3)$$

eg:



Use Arden's theorem

Sol":

$$q_1 = q_3 \cdot a + q_1 \cdot a + \epsilon$$
$$q_2 = q_2 \cdot b + q_3 \cdot b + q_1 \cdot b$$
$$q_3 = q_2 \cdot a$$

$$\Rightarrow q_2 = q_2 \cdot b + q_2 \cdot a \cdot b + q_1 \cdot b$$

$$q_2 = q_2 (b + ab) + q_1 \cdot b$$

$$q_2 = q_1 \cdot b (b+ab)^*$$

$$\Rightarrow q_3 = (q_1 \cdot b (b+ab)^*) \cdot a$$

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

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

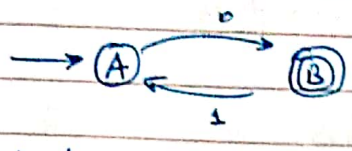
$$q_1 = (a + b(b+ab)^* aa)^*$$

* We have to find the eqn of the end-state.

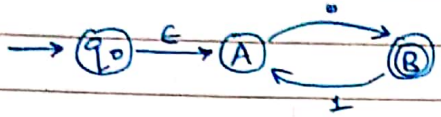
$R = RP + q$

$R = qP^*$

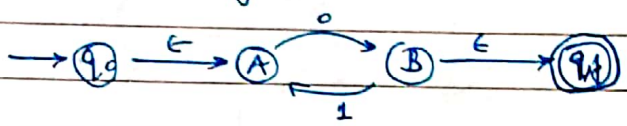
State Elimination method.



→ Check initial state. if it has any incoming state, make another Initial state.

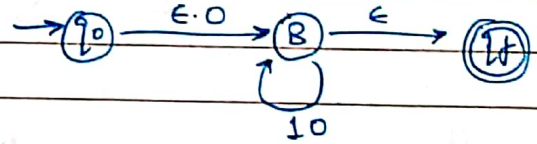


→ Check for final state. if it has any outgoing state. make another final state

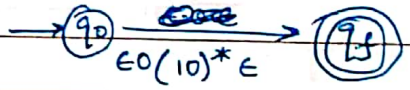


→ Eliminate Intermediate States.

Eliminating (A) ⇒

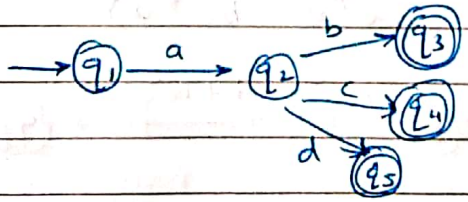


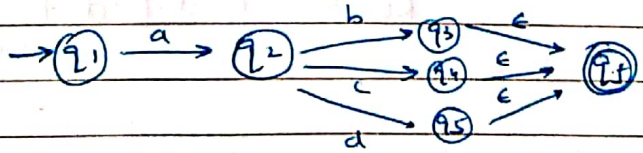
Eliminating (B) ⇒



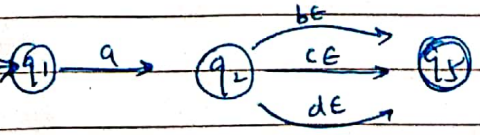
$$R.E = 0(10)^*$$

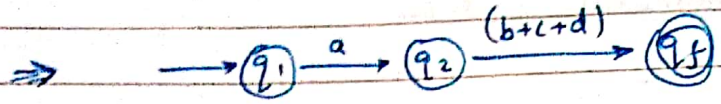
eg:

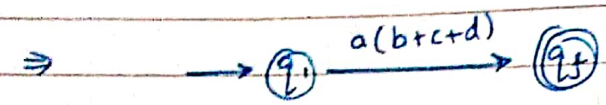




⇒

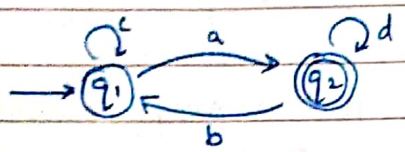




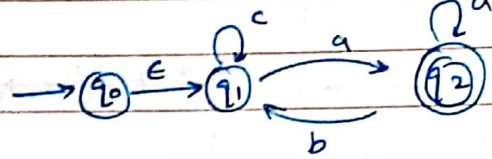


$\therefore$ RE = $a(b+c+d)$

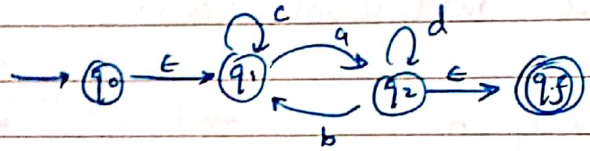
q:



Step 1 :

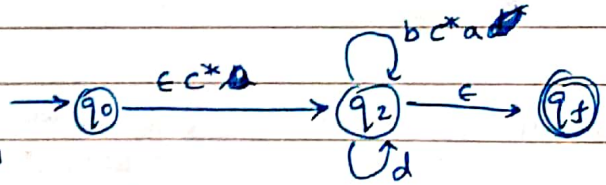


Step 2:

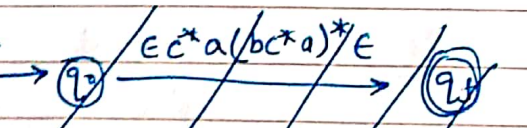


Step 3 :

eliminating q_1

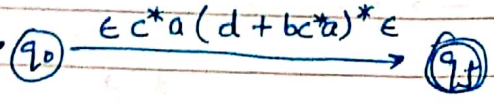


eliminating q_2



RE = $c^*a(bc^*a)^*$

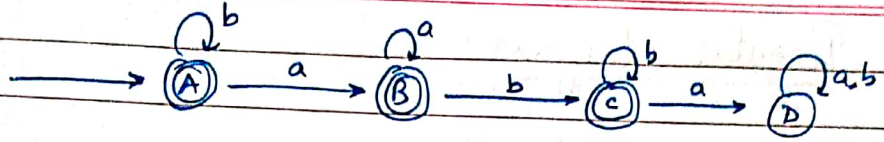
eliminating q_2



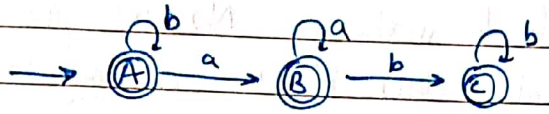
$\therefore$ RE $\Rightarrow$ $c^*a(d+bc^*a)^*$

classmate
Date
Page

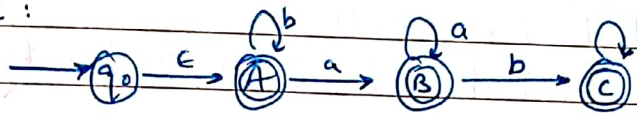
eg.



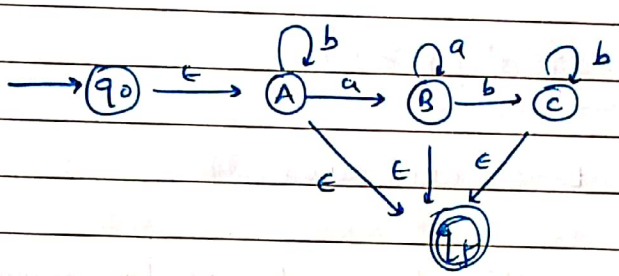
Sol^n: Remove the dead state.



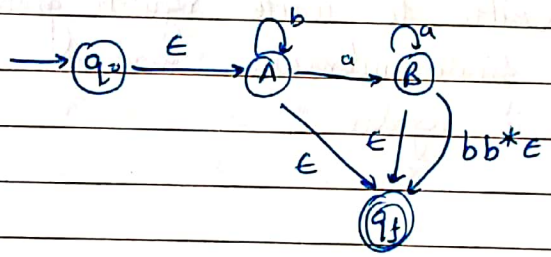
Step 1 :



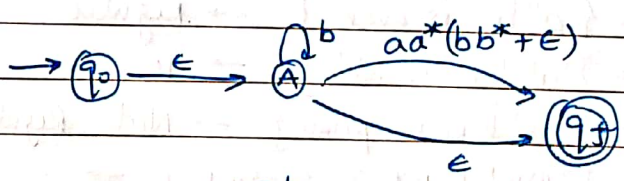
Step 2 :



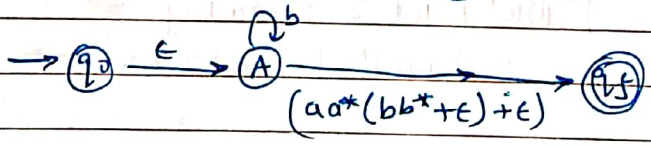
→



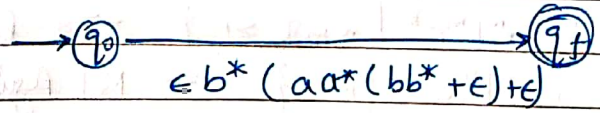
⇒



⇒



⇒



RE ⇒ $b^*(aa^*(bb^*))$

Regular languages

Finite — Regular language

* If it is possible to make a Machine for (FA-M) DFA finite Automata, then it is a regular lang.

eg: L = { 00, 01, 10, 11 }
L = { ab, abab, ababab, ... } — Regular

eg: L = { a^n | n ≥ 1 }

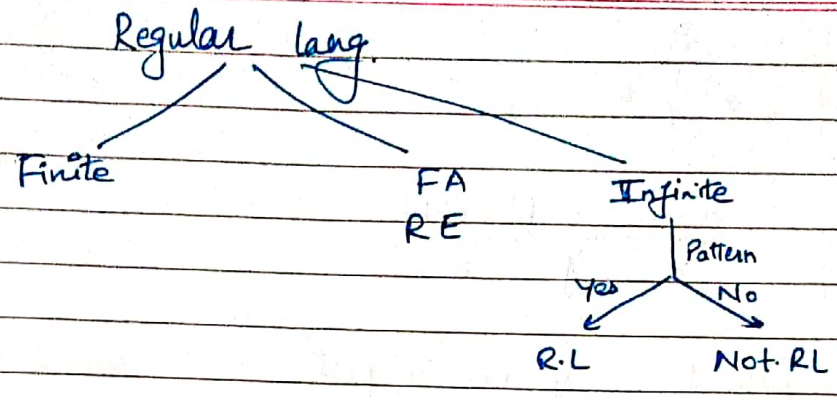
L = { a, aa, aaa, ... }

* If it is possible to write regular grammar or regex or draw finite automata machine, then the language is regular.

eg: L = { a^n | n is even } → Regular
L = { a^n | n is odd } → "
L = { a^n | n is prime } → Not Regular
L = { ww^R | |w| = 2 Σ = {a, b} }
= { ww^R | w ∈ {a, b} } → Not Regular (Since length of string is not bounded)

L = { $a^n b^m c^k$ | n, m, k ≥ 1 } → Regular
L = { $a^n b^n c^n$ | n ≥ 1 } → Not Regular (because we'll have to compare what is the n for all characters)

L = { $a^i b^{2j}$ | i, j ≥ 1 } → Regular [$aa^* bb(bb)^*$]
L = { $a^i b^{4j}$ | i, j ≥ 1 } → Regular [$a+(bbbb)+$]



Pumping lemma
(Negativity Test)

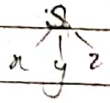
→ Proves that a grammar is not Regular

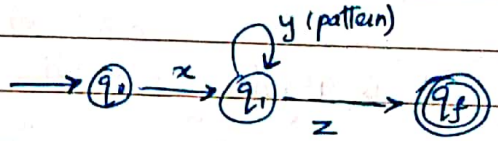
⇒ If A is Regular language, then A has a pumping length 'p' such that any string 's' where $|s| \geq p$ may be divided into 3 parts, $s = xyz$ such that the following conditions must be true

① $xy^iz \in A \quad \forall \; i \geq 0$

② $|y| > 0$

③ $|xy| \leq p$





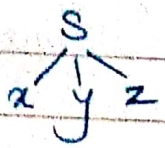
Q. Prove that $L = \{a^n b^n \mid n \geq 1\}$ is not Regular.

Sol": ① Assume given L is Regular

② pumping length 'p' = 7

$a^n b^n \xrightarrow{p=7} a^7 b^7$

aaaaaaa bbbbbbb = S



$\underbrace{aaaaaaaa}_{x}\underbrace{aaa}_{y}\underbrace{bbbbbbbb}_{z}$

Case 1: y is 'a' part. Case 2:

$xy^iz \in A$

let $i=2$ $\quad p = m$

xy^2z

$(aaaa)(aaa)^2(bbbbbbbb)$

$\rightarrow \underbrace{aaaaaaaaaaa}_{10}\underbrace{bbbbbbb}_{7} \notin A$

* Check for all three conditions.

Q. $L = \{ vv^R : v \in \Sigma^* \}$

$w = \underbrace{a^m b^m}_{v}\underbrace{b^m a^m}_{v^R}$

① L is Regular

② $w = \overbrace{\underbrace{a\ldots a}_{x}\ldots\underbrace{a-a}_{y}}^{m}\overbrace{b\ldots b\ldots b\ldots b}^{m}\overbrace{a\ldots a}^{m}$

$y = a^k$

$w = xy^iz$

let $i=2$

$\overbrace{\underbrace{a\ldots a}_{x}\underbrace{(a\ldots aa\ldots a)}_{y}}^{m+k}\overbrace{\underbrace{bb\ldots b}_{z}\underbrace{a\ldots a}}^{m}$

$\Rightarrow a^{m+k} b^m b^m a^m \notin A$

$a^n b^m$ ✓

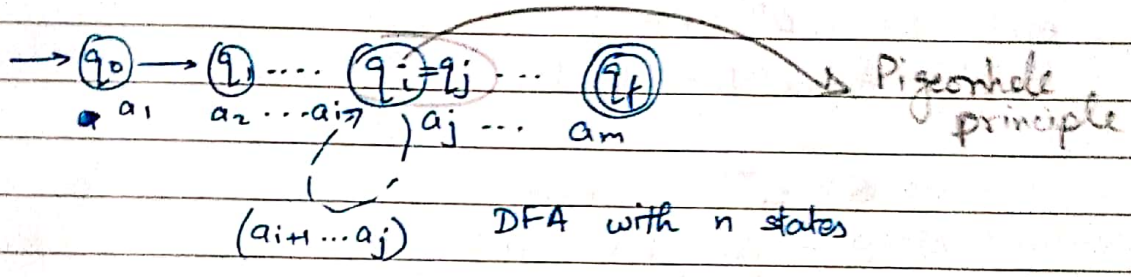
$xy^iz \notin A$
$|xy| \leq m$
& $|y| \geq 0$
} Not Regular.

7/7/17

ex: $S = a_1 a_2 \cdots a_m$

$q_0\ q_1\ q_2 \cdots q_m$

if $n > n$
$(m+1)$ states



$a_1 \cdots a_i a_j \cdots a_m$

$\underbrace{a_1 \cdots a_i}_{x}\ \underbrace{(a_{i+1} \cdots a_j)}_{y}\ \underbrace{a_j \cdots a_m}_{z}$

P $xyyz \in R$ $|xy| \leq n$ $xy^{i=2}z$

$xyyyz \in R$ $|y| \geq 1$

eg: $L = \{ a^n b^l c^{n+l} : n, l \geq 0 \}$

Assume L is regular

$S = a^m b^m c^{2m}$

$= \underbrace{a \cdots a}_{m}\ \underbrace{b \cdots b}_{m}\ \underbrace{c \cdots \cdots c}_{2m}$

$x \quad y \quad z$

xy^iz & $i=2$

$\underbrace{(a\cdots a)}_{m}\underbrace{(b\cdots\cdots b)^2}_{m+k}\underbrace{c\cdots\cdots c}_{2m}$

$\Rightarrow a^m b^{m+k} c^{2m}$

$2m \neq m+m+k$

$\therefore$ Not regular

eg- $L = \{ a^{n!} : n \geq 0 \}$

Assume L is Regular

$S = a^{m!}$ $\quad n=m$

$S = \underbrace{a\cdots a}_{m}\underbrace{a\cdots a a\cdots a}_{m!-m}$

$S = \underbrace{\overbrace{a..aa..a}^{m}}_{x\ y}\ \overbrace{a----aa----a}^{m!-m}$ (z)

$xy^iz \quad i=2$

$S = (a\cdots a)(a\cdots a)^2(a\cdots aa\cdots a)$

$= \underbrace{(a\cdots a)(a\cdots a)^k}_{m+k}\ \underbrace{(a\cdots aa\cdots a)}_{m!-m}$

$m+k+m!-m$

$= k+m!$ Not Regular

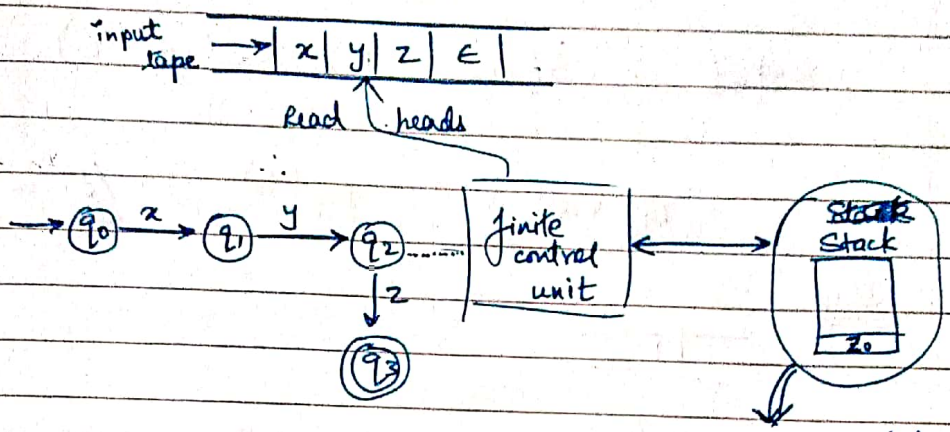
Disadv. of Finite Automata

→ can't store/count infinite values (can't store n values)

Limitations of FA

It can't store and count.

PUSH DOWN AUTOMATA (FA + Stack)



- zero address data structure
- No indexing req. (Z_0 ⇒ empty stack)

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

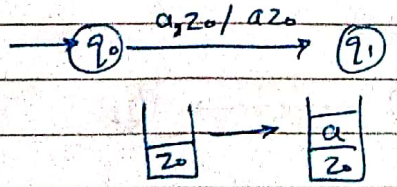
- Q: set of finite states
- Σ: set of input alphabets
- F: bottom ele. of the stack
- Z_0: top of the stack
- Γ: transition func.
- δ

DPDA, NPDA $\rightarrow$ $Q \times \Sigma \times \Gamma \rightarrow Q \times \Gamma^*$ — push, pop, skip.

$Q \times \Sigma \times \Gamma = 2^{Q \times \Gamma^*}$

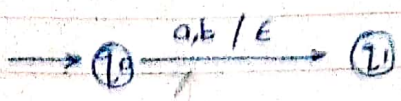
Push operation

|a|b|



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

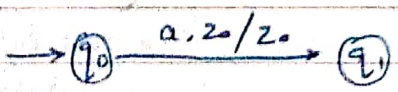
Pop operation



$\delta(q_0, a, b) = \delta(q_1, \in)$

On reading a, pop(a) if /ε is there

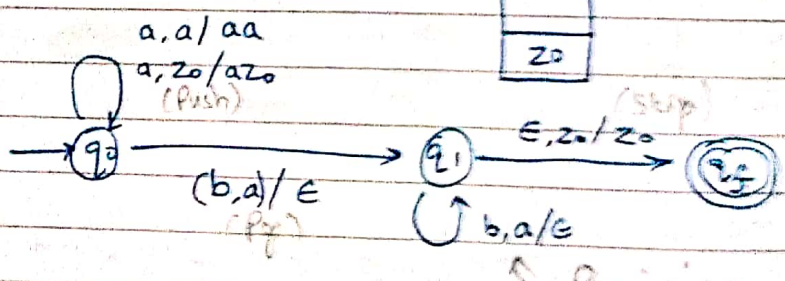
Skip



Do Nothing on reading a

eg: $a^n b^n \quad : n \geq 1$

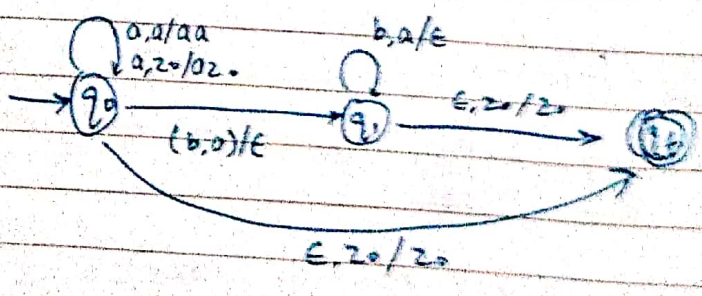
let n=2 : aabb ∈



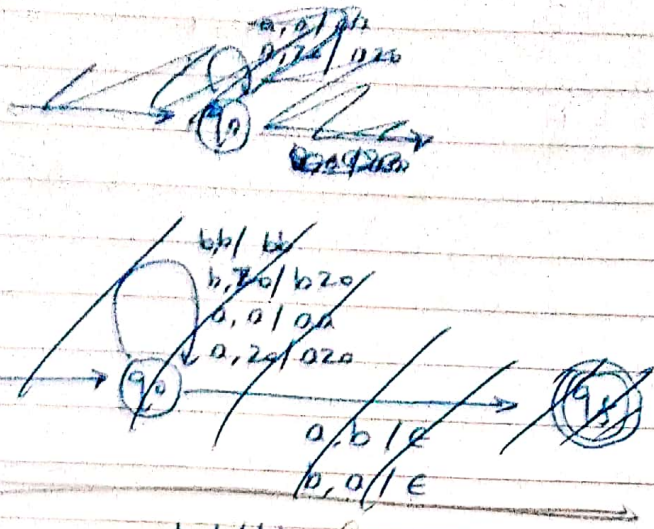
Pop a on reading b

→ Acceptance by final state
→ Acceptance by empty state

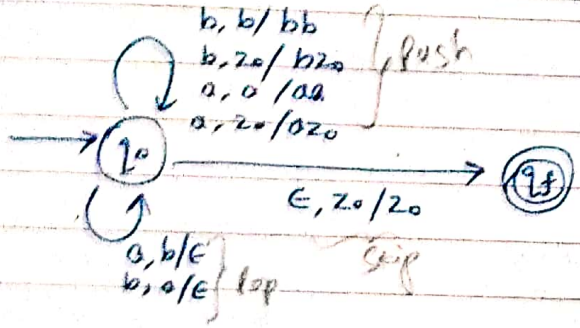
eg: $a^n b^n \quad ; n \geq 0$



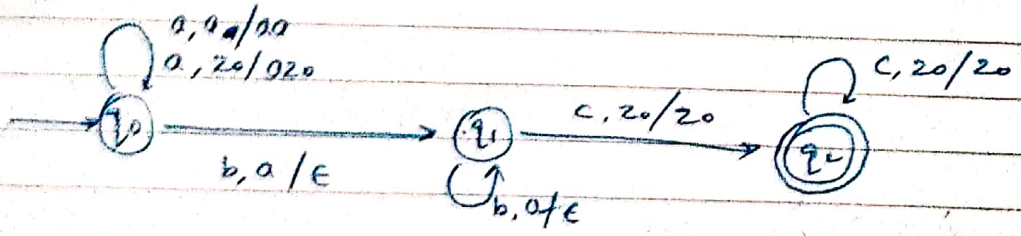
eg: $L = \{ w \mid n_a(w) = n_b(w) \}$



Soln



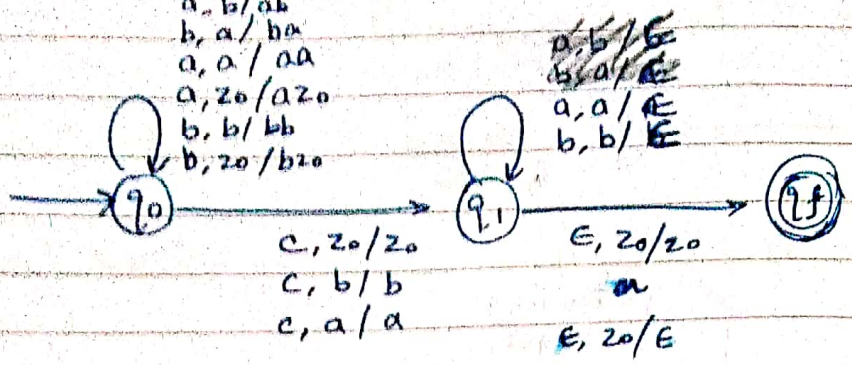
eg: $L = \{ a^n b^n c^m \mid n, m \geq 1 \}$



Limitation

for $a^n b^n c^n$ → can't verify for 'c' also.

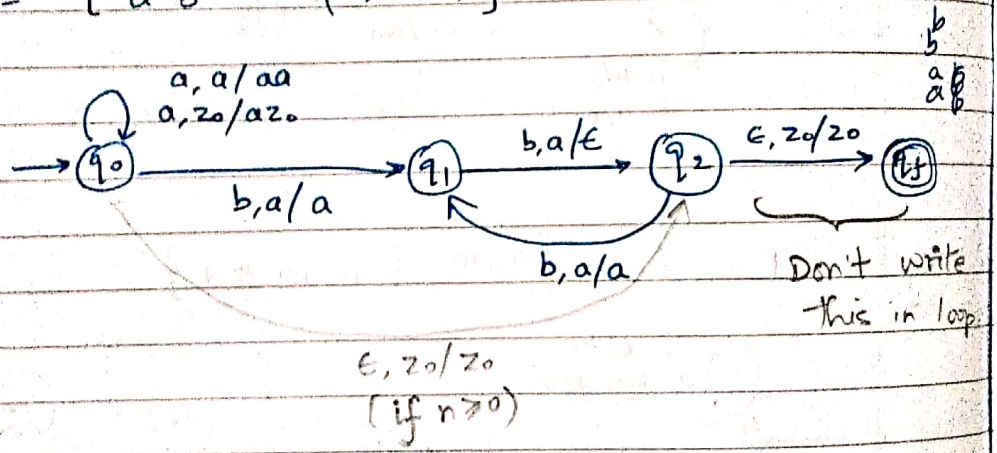
eg: $L = \{ wcw^R \mid w \in \{a,b\}^* \}$



eg: $L = \{ wcw \}$

↑ Not CFL (Context Free Lang.)

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



eg: $L = \{ ww^R \}$

↳ How do we get to know w has ended

∴ Not CFL ?