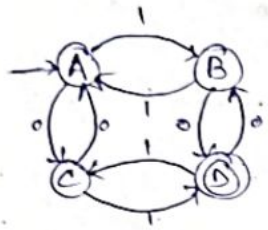
Finite State Machine

- Finite Automata
  - FA with output
    - Moore Machine
    - Mealy Machine
  - FA without output
    - DFA
    - NFA
    - ϵ-NFA

- DFA - Deterministic finite automata
  - It is the smallest unit of ~~automation~~ computation
  - It has very limited memory

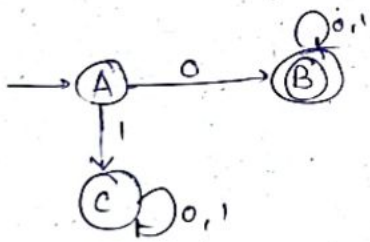


Q = Set of all states = $\{A, B, C, D\}$
Σ = Inputs = $\{0, 1\}$
q_0 = Initial State = $\{A\}$
F = Final State = $\{D\}$
δ = Transition function from $Q \times \Sigma \rightarrow Q$

	0	1
A	C	B
B	D	A
C	A	D
D	B	C

- Deterministic finite automata (DFA) (Example 1)

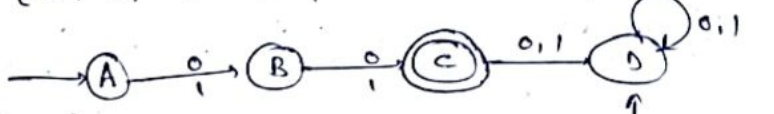
L = set of all strings that start with '0'
= $\{0, 00, 01, 000, 010, 001, 011, 0000, \ldots\}$



- Construct a DFA that accepts a sets of all strings over $\{0,1\}$ of length 2

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

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



$\Sigma = \{0, 1\}$
$q_0 = A$
$F = C$
$Q = \{A, B, C\}$

	0	1
A	B	B
B	C	C
C	D	D

Operations on Regular Languages

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

CONCATENATION $A \circ B = \{xy \mid x \in A \text{ or } y \in B\}$

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

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

- $A \cup B = \{pq, r, t, uv\}$
- $A \circ B = \{pqt, pquv, rt, ruv\}$
- $A^* = \{\epsilon, pq, r, pqr, rpq, pqpq, rr, pqpqpq, rrr \ldots\}$

Theorem 1: The class of Regular languages is closed under UNION.

Theorem 2: The class of Regular languages is closed under CONCATENATION.

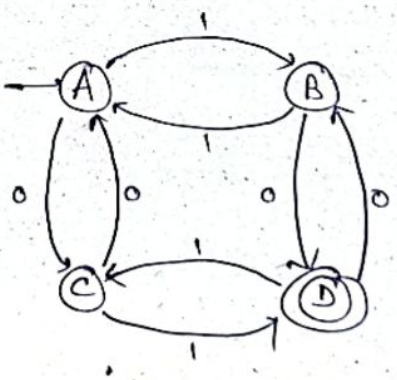
NON DETERMINISTIC FINITE AUTOMATA (NFA)

Deterministic Finite Automata

↓

DETERMINISM

- In DFA, given the current state, we know what the next step will be.
- It has only one unique next state.
- It has no choices or randomness.
- It is simple and easy to design.

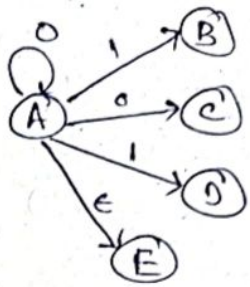


Non deterministic finite automata

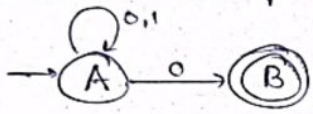
↓

NON DETERMINISM

- In NFA, given the current state, there could be multiple new states
- The next stage may be chosen at random.
- All the next states may be chosen in parallel



NFA - Formal Definition



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

Q = Set of all states = {A, B}

Σ = Inputs = {0, 1}

q_0 = Initial State = {A}

F = Final State = {B}

δ = Q × Σ → 2^Q

$$\begin{cases} A \times 0 \rightarrow A \\ A \times 0 \rightarrow B \\ A \times 1 \rightarrow A \\ B \times 0 \rightarrow \phi \\ B \times 1 \rightarrow \phi \end{cases}$$

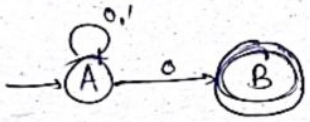
For example

If three states are there, A, B & C

∴ Total possibilities,

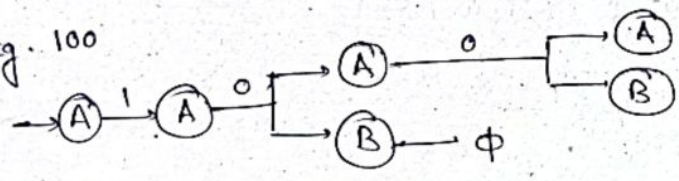
A, B, C, AB, BC, AC, ABC, ϕ

= 8 possibilities

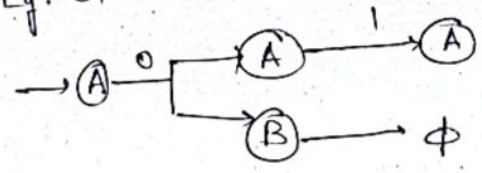


L = {Set of all strings that end with 0}

Eg. 100

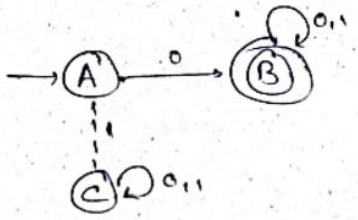


Eg. 01



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

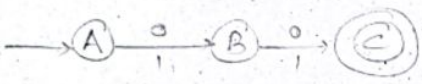
- L = { Set of all strings that start with 0 }
  = { 0, 00, 01, 000, ... }



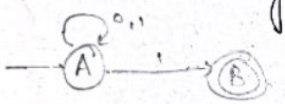
Construct a NFA that accepts set of all strings over {0,1} of length 2

→ $\Sigma = \{0,1\}$

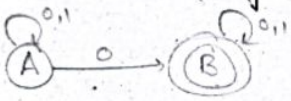
$L = \{00, 01, 10, 11\}$



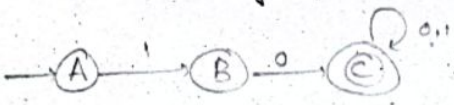
- L1 = { Set of all strings that ends with '1' }



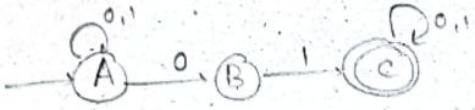
- L2 = { Set of all strings that contains '0' }



- L3 = { Set of all strings that starts with '10' }



- L4 = { Set of all strings that contain 01 }.



- L5 = { Set of all strings that ends with '11' }

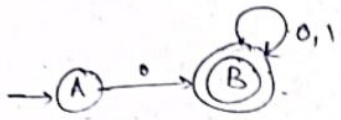
Conversion of NFA to DFA

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

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

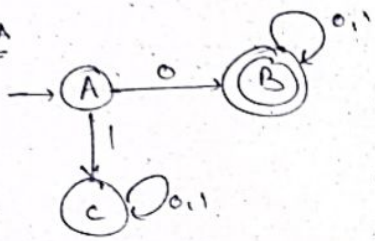
$\Sigma = \{0,1\}$

NFA



	0	1
A	B	ϕ
B	B	B

DFA

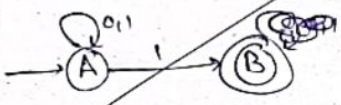


	0	1
A	B	C
B	B	B
C	C	C

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

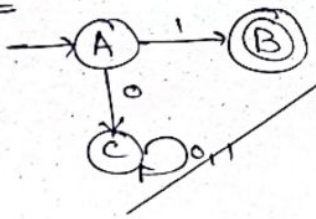
$\Sigma = \{0,1\}$

NFA



	0	1
A	{A}	{A,B}
B	ϕ	ϕ

DFA

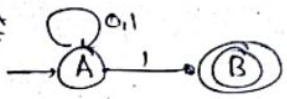


	0	1
A	{A}	{AB}
B		
C		

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

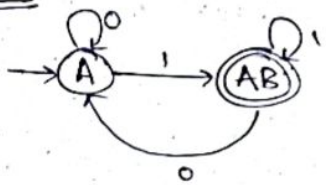
$\Sigma = \{0,1\}$

NFA



	0	1
A	{A}	{A,B}
B	ϕ	ϕ

DFA



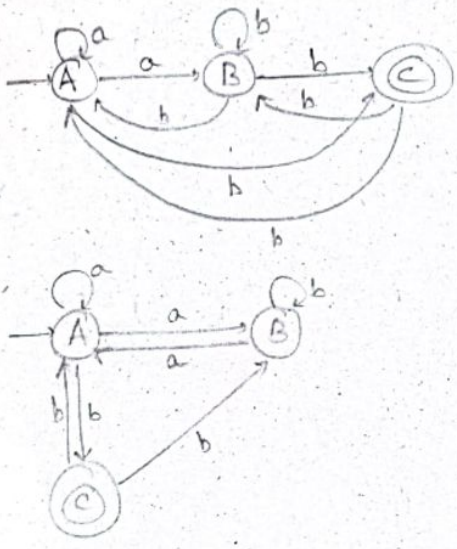
	0	1
A	{A}	{AB}
AB	{A}	{AB}

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

δ	a	b
→A	A,B	C
B	A	B
(C)	-	A,B

$\Sigma = (a,b)$
$Q = \{A,B,C\}$
$q_0 = A$
$F = C$

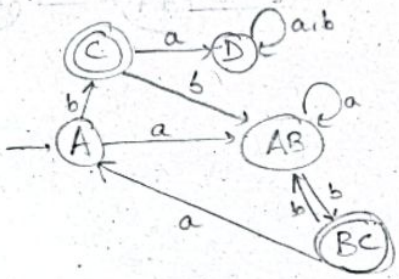
NFA



DFA

	a	b
A	AB	C
AB	AB	BC
(BC)	A	AB
(C)	D	AB
D	D	D

D = dead state

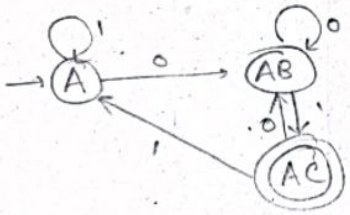


Given below is the NFA for a language
L = {Set of all strings over (0,1) that ends with '01'}. Construct its equivalent DFA

NFA

	0	1
A	A,B	A
B	ϕ	C
(C)	ϕ	ϕ

DFA

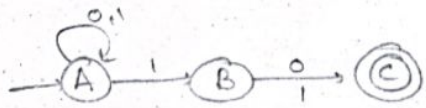


	0	1
A	AB	A
AB	AB	AC
(AC)	AB	A

Design an NFA for a language that accepts all strings over {0,1} in which the 2nd Last symbol is always '1'. Then convert it to its equivalent DFA

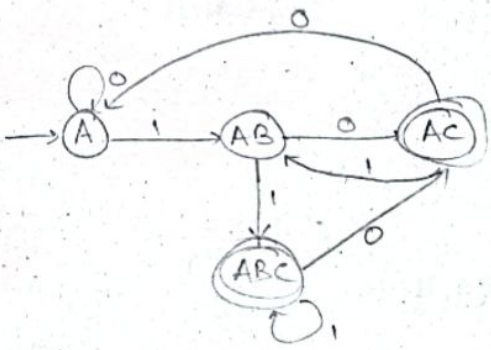
L = {10, 11, 010, 011, 111, ... }

NFA



	0	1
→ A	A	A,B
B	C	C
(C)	ϕ	ϕ

DFA



	0	1
A	A	AB
AB	AC	ABC
(AC)	A	AB
(ABC)	AC	ABC

MINIMIZATION OF DFA

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

Two states A and B are said to be equivalent if

$\delta(A,X) \longrightarrow F$ OR $\delta(B,X) \longrightarrow F$ AND $\delta(A,X) \longmapsto F$ OR $\delta(B,X) \longmapsto F$

where X is any input string

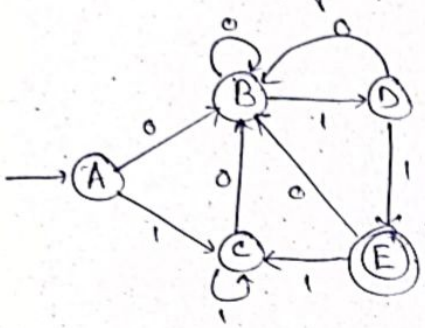
If $|x| = 0$, then A and B are said to be 0 equivalent

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

⋮

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

- Minimisation of DFA



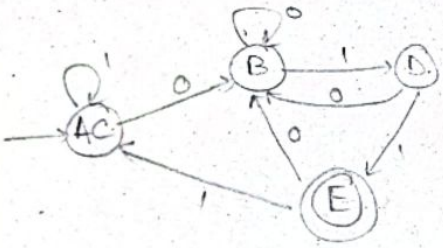
	0	1
→ A	B	C
B	B	D
C	B	C
D	B	E
(E)	B	C

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}



Construct a minimum DFA equivalent to the DFA described by:

	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:

$\{q_0, q_4, q_6\}$

$\{q_1, q_7\}$

$\{q_3, q_5\}\{q_2\}$

2 Equivalence

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

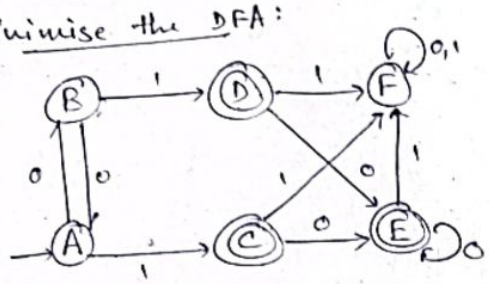
3 Equivalence

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

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

MINIMISATION OF DFA (When there are more than one final states involved)

Q Minimise the DFA:

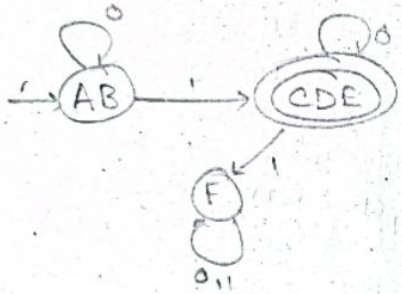


	0	1
→ A	B	C
B	A	D
C	E	F
D	E	F
E	E	F
F	F	F

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

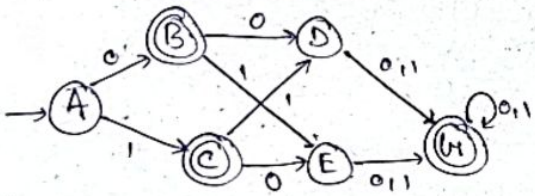
1 Equivalence : {A,B}{F}{C,D,E}

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



	0	1
{A,B}	{A,B}	{C,D,E}
{F}	{F}	{F}
{C,D,E}	{C,D,E}	{F}

MINIMISATION OF DFA (When there are unreachable states involved)



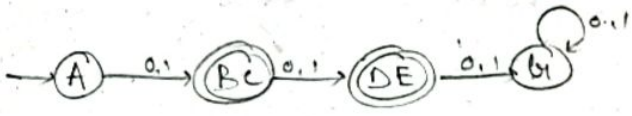
	0	1
→ A	B	C
B	D	E
C	E	D
D	G	G
E	G	G
G	G	G

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}



	0	1
{A}	{BC}	{BC}
{D,E}	{G}	{G}
{B,C}	{D,E}	{D,E}
{G}	{G}	{G}

Finite Automata with outputs

MEALY MACHINE

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

where

Q = Finite set of states

Σ = Finite non empty set of Input alphabets

Δ = Set of Output alphabets

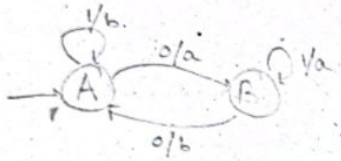
δ = Transition function

$Q \times \Sigma \to Q$

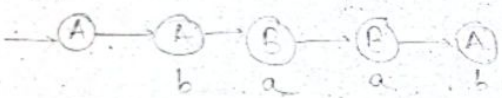
λ = Output function

$\Sigma \times Q \to \Delta$

q_0 = Initial state / Start State



Eg. 1010



Input = n

Output = n

MOORE MACHINE

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

where

Q = Finite set of states

Σ = Finite non empty set of Input alphabets

Δ = Set of output alphabets

δ = Transition function

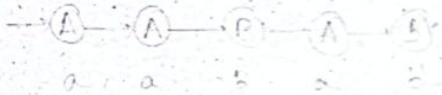
$Q \times \Sigma \to Q$

λ = Output function

$Q \to \Delta$

q_0 = Initial state / Start state

Eg. 1010



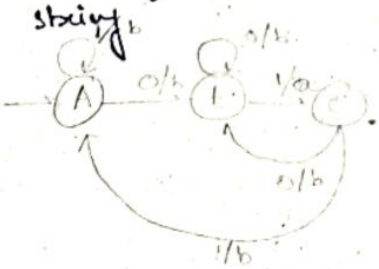
Input = n

Output = n+1

Construction of Mealy Machine

① Construct a mealy machine that produces the 1's complement of any binary input string

② Construct a mealy machine that prints 'a' whenever the sequence '01' is encounted in any input binary string



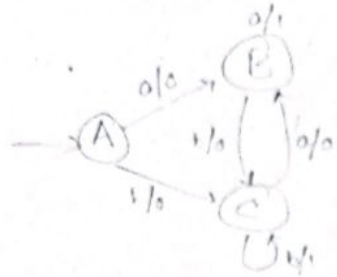
$\Sigma = \{0, 1\}$, $\Delta = \{a, b\}$

0110 → bbb

1000 → bbb

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.

aa – 1
bb – 1



Construct a mealy machine that gives 2's complement of any binary input. (Assume that the last carry bit is neglected)

2's complement = 1's complement + 1

Eg. 10100

1's complement: 01011

+1

2's complement: 01100

Eg. 11100

1's complement: 00011

+1

2's complement: 00100

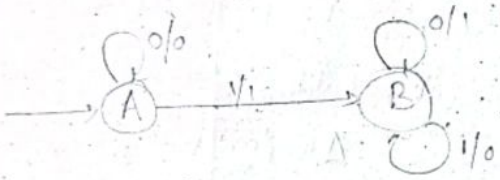
Eg. 1111

1's complement: 0000

+1

2's complement: 0001

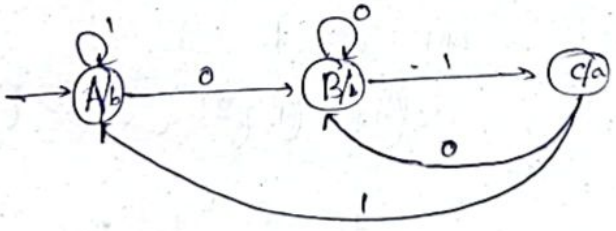
MSB ← LSB

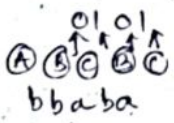


Construct a Moore machine that prints 'a' whenever the sequence '01' is encountered in any binary string

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

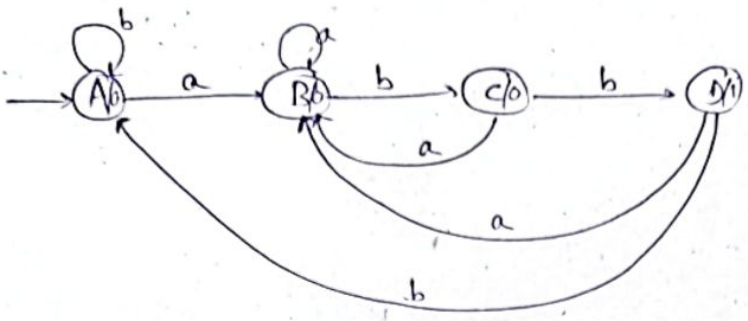
$\Delta = \{a, b\}$





bbaba

① Construct a Moore machine that counts the occurances of the sequence 'abb' in any input strings over {a,b}



a b b

A B C D

0 0 0 1

a b b a b b

A B C D B C D

0 0 0 1 0 0 1

② For the following moore machine the input alphabet is $\Sigma = \{a,b\}$ and the output alphabet is $\Delta = \{0,1\}$. Run the following input sequences and find the respective output

(i) aabab (ii) abbb (iii) ababb

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

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

States	a	b	Outputs
→ q_0	q_1	q_2	0
q_1	q_2	q_3	0
q_2	q_3	q_4	1
q_3	q_4	q_4	0
q_4	q_0	q_0	0

(i) a a b a b

q_0 q_1 q_2 q_4 q_0 q_2

0 0 1 0 0 1

(ii) a b b b

q_0 q_1 q_4 q_0 q_2

0 1 0 0 1

q_0 q_1 q_3 q_4 q_0

0 0 0 0 0

(iii) a b a b b

q_0 q_1 q_3 q_4 q_0 q_2

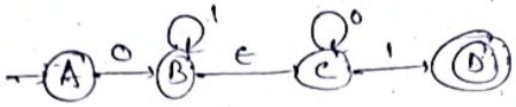
0 0 0 0 0 1

Epsilon (ε) - NFA

ε - NFA

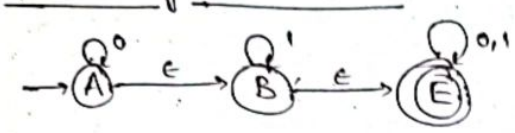
↳ empty symbols

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

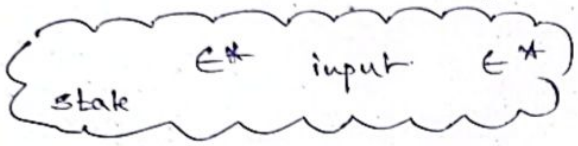


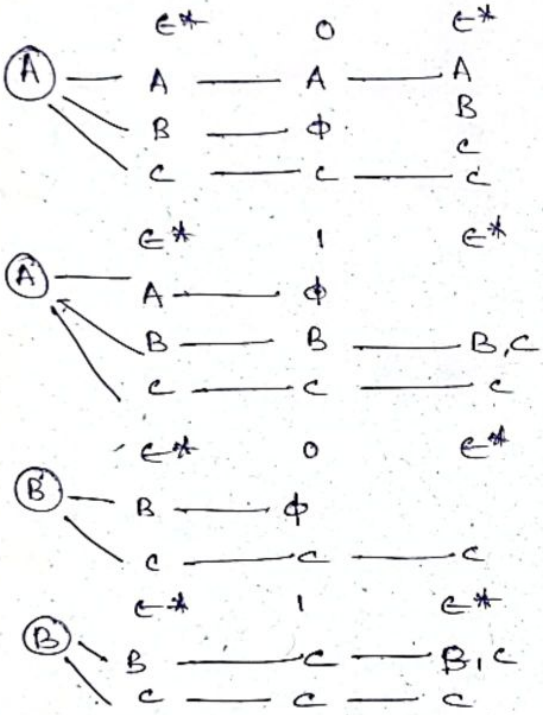
Every state on ε goes to itself.

Conversion of ε NFA to NFA

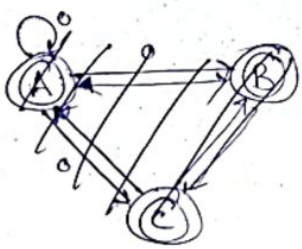


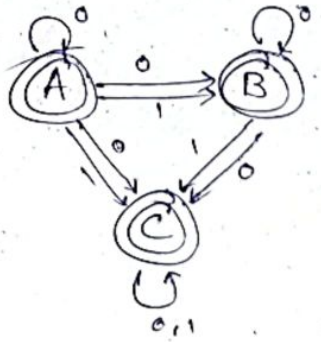
ε closure (ϵ^*) - All the states that can be reached from a particular state only by seeing ε symbol.



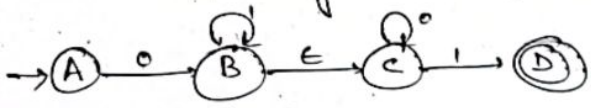


	0	1
→ A	{A, B, C}	{B, C}
B	{C}	{B, C}
C	{C}	{C}





Q Convert the following ∈NFA to its equivalent NFA

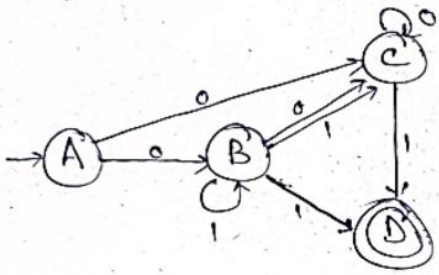


	∈*	0	∈*
A	A	B	B C
B	B C	ϕ	ϕ C
C	C	C	C
D	D	ϕ	ϕ

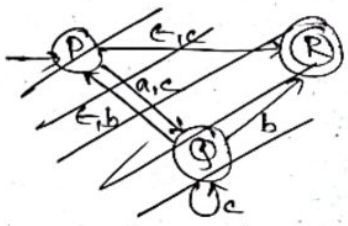
	∈*	1	∈*
A	A	ϕ	ϕ
B	B C	B D	B C D
C	C	D	D
D	D	ϕ	ϕ

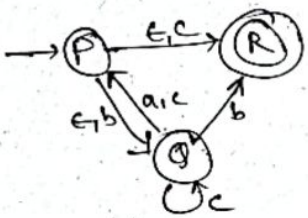
NFA

	0	1
→ A	{B,C}	{ϕ}
B	{ϕ,C}	{B,C,D}
C	{C}	{D}
(D) | ϕ | {ϕ}



Q Convert the following ∈NFA to its equivalent NFA



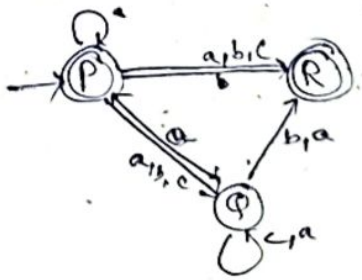


	∈*	a	∈*
P	P Q R	ϕ P ϕ	ϕ P Q R
Q	Q	P	P Q R
R	R	ϕ	-

	∈*	b	∈*
P	P Q R	Q R	Q R
Q	Q	R	R
R	R	ϕ	-

	∈*	c	∈*
P	P Q R	R Q ϕ	R Q -
Q	Q	Q	Q
R	R	ϕ	-

	a	b	c
→ (P)	{P,Q,R}	{Q,R}	{Q,R}
Q	{P,ϕ,R}	{R}	{Q}
(R)	ϕ	ϕ	ϕ



Regular expressions

Regular expressions are used for representing certain sets of strings in an algebraic fashion.

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

ii) The union of two regular expressions is also a regular expression

(iii) The concatenation of two regular expression is also a regular expression,

(iv) The iteration (or closure) of a regular expression is also a regular expression.

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

Q Describe the following sets as regular expression

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 $\{1, 11, 111, \ldots\}$

$R = 1^+$

IDENTITIES OF REGULAR EXPRESSIONS

- $\phi + R = R$
- $\phi R + R\phi = \phi$
- $\epsilon R = R\epsilon = R$
- $\epsilon^* = \epsilon \;\&\; \phi^* = \epsilon$
- $R + R = R$
- $R^* R^* = R^*$
- $RR^* = R^* R$
- $(R^*)^* = R^*$
- $\epsilon + RR^* = \epsilon + R^* R = R^*$
- $(PQ)^* P = P(QP)^*$
- $(P+Q)^* = (P^* Q^*)^* = (P^* + Q^*)^*$
- $(P+Q)R = PR + QR$ and $R(P+Q) = RP + RQ$

ARDENS THEOREM

If P and Q are two regular expressions over Σ and if P doesn't contain ϵ, then the following expression in R given by $R = Q + RP$ has a unique solution i.e. $R = QP^*$

$\longrightarrow$ $R = Q + RP \longrightarrow (1)$

$= Q + (QP^*)P$

$= Q(\epsilon + P^*P)$

$= QP^*$

Proved

$\longrightarrow$ $R = Q + RP$

$= Q + [Q + RP]P$

$= Q + QP + RP^2$

$= Q + QP + [Q + RP]P^2$

$= Q + QP + QP^2 + RP^3$

$\vdots$

$= Q + QP + QP^2 + QP^3 + \ldots QP^n + RP^{n+1}$

$= Q + QP + QP^2 + \ldots + QP^n + QP^*P^{n+1}$

$= Q[\epsilon + P + P^2 + \ldots P^n + P^*P^{n+1}]$

$= QP^*$

Q PT: $(1+00^*1) + (1+00^*1)(0+10^*1)^*(0+10^*1) = 0^*1(0+10^*1)^*$

$(1+00^*1)[\epsilon + (0+10^*1)^*(0+10^*1)]$

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

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

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

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

Proved.

Q Design regular expression for the following languages over $\{a,b\}$

(i) languages accepting strings of length exactly 2

(ii) " " " " atleast 2

(iii) " " " " atmost 2

(i) $L = \{aa, ab, ba, bb\}$

$R = aa + ab + ba + bb$

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

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

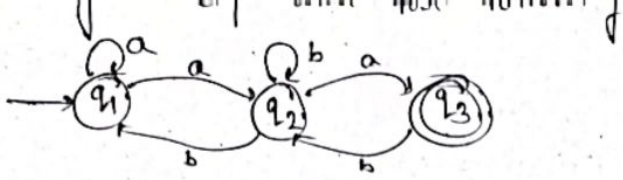
(ii) $L = \{aa, ab, ba, bb, aaa, aab, \ldots\}$

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

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

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

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



$q_3 = q_2 a$ ——— ①

$q_2 = q_1 a + q_2 b + q_3 b$ ——— ②

$q_1 = \epsilon + q_1 a + q_2 b$ ——— ③

$q_3 = q_2 a$

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

$= q_1 aa + q_2 ba + q_3 ba$ ——— ④

From ②,

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

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

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

$R = Q + RP$

$q_2 = (q_1 a)(b + ab)*$ ——— ⑤

From ③,

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

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

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

$R = Q + RP$
$R = QP*$ } Arden's theorem

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

$= ((a + a(b + ab)*) b)*$ ——— ⑥

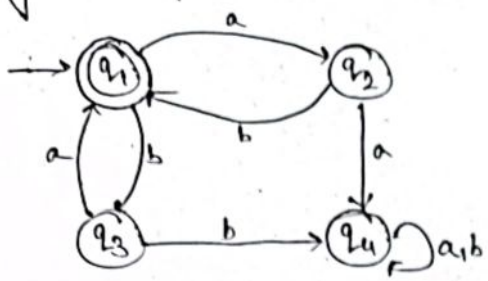
Final state $q_3 = q_2 a$

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

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

= Required regular expression for given NFA

Q Find regular expression for following DFA



$q_1 = \epsilon + q_2 b + q_3 a$ — (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)

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

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

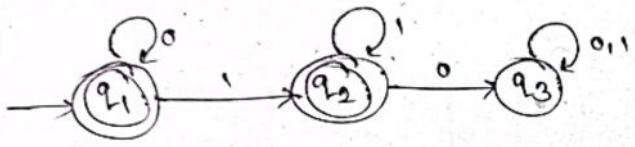
$q_1 = \epsilon + q_1 (ab + ba)$

$R = Q + RP$
$R = QP*$ } Ardens theorem $\Rightarrow$ $q_1 = \epsilon (ab+ba)*$

$\boxed{q_1 = (ab+ba)*}$ — (5)

⟶ Regular expression

Q



$q_1 = \epsilon + q_1 0$ — (1)

$q_2 = q_1 1 + q_2 1$ — (2)

$q_3 = q_2 0 + q_3 0 + q_2 1$ — (3)

FINAL STATE (q_1)

(1) $\Rightarrow$ $q_1 = \epsilon + q_1 0$

$R = Q + RP$
$R = QP*$ } Ardens theorem $\Rightarrow$ $q_1 = \epsilon 0*$

$\boxed{q_1 = 0*}$ — (4)

Final State (q_2)

$q_2 = q_1 1 + q_2 1$

$q_2 = 0*1 + q_2 1$

$R = Q + RP$
$R = QP*$ } Ardens theorem $\Rightarrow$ $q_2 = ((0*1)1)*$

R = Union of both final states

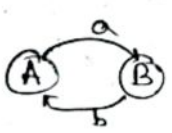
$= 0* + 0*11*$

$= 0*(\epsilon + 11*)$

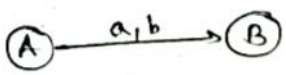
$= 0*1*$

Conversion of regular expression to finite automata

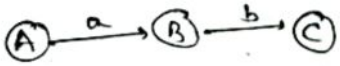
(a+b)



or



(a.b)



a*

Q Convert the following regular expression to their equivalent finite automata

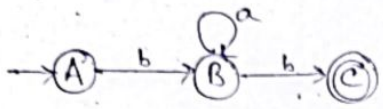
(i) ba^*b

(ii) $(a+b)c$

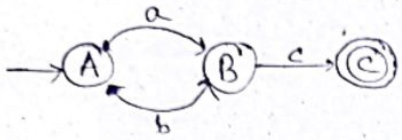
(iii) $a(bc)^*$

(i) ba^*b

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

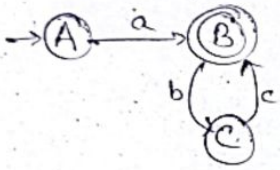


(ii) $(a+b)c$



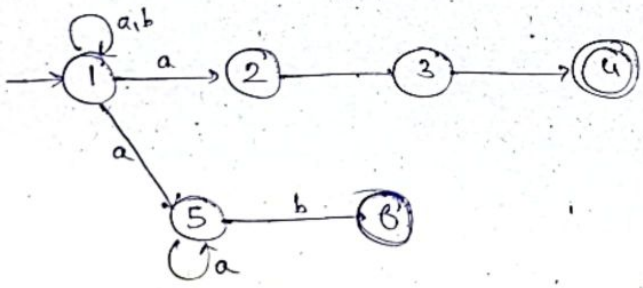
(iii) $a(bc)^*$

$L = \{a, abc, abebc, abcbcbc \ldots\}$



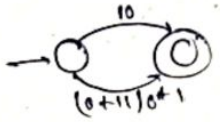
Q Convert the following regular expression to its equivalent finite automata.

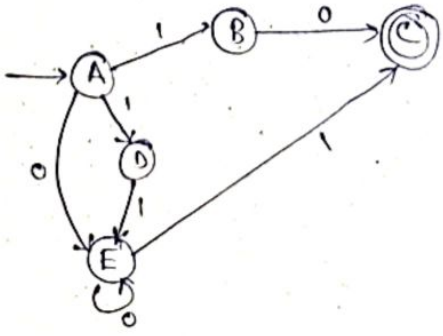
$(a|b)^*(abb|a^*b)$



Q Convert the following regular expression to its equivalent finite automata

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





Equivalence of two finite automata

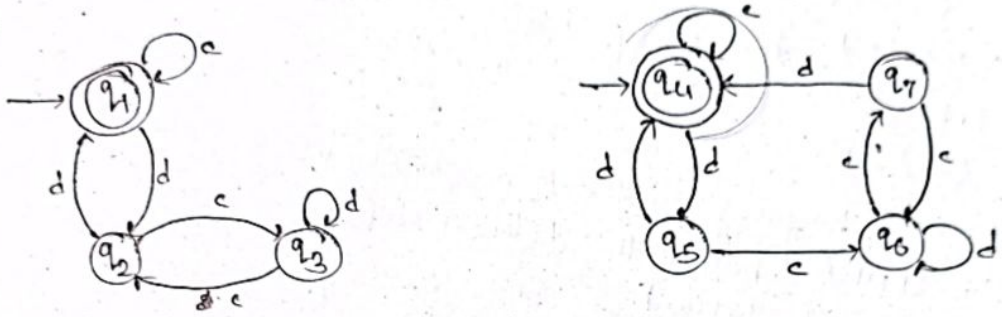
Steps to identify equivalence

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 $\delta\{q_i, a\} = q_a$ and $\delta\{q_j, a\} = q_b$.

The two automata aren't equivalent if for a pair $\{q_a, q_b\}$, one is intermediate state and the 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.

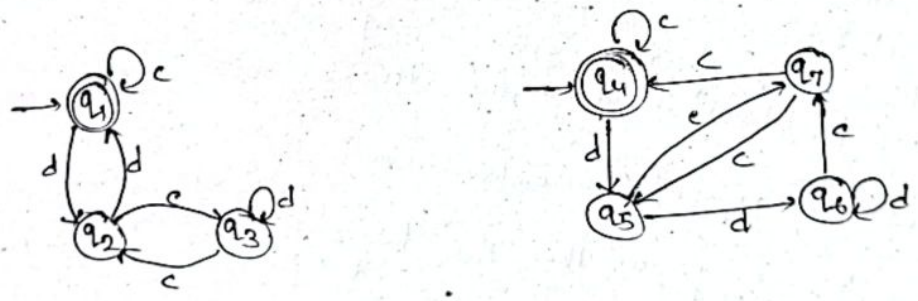
Ex



States	c	d
$\{q_1, q_4\}$	$\{q_1, q_4\}$	$\{q_2, q_5\}$
$\{q_2, q_5\}$	$\{q_3, q_6\}$	$\{q_1, q_4\}$
$\{q_3, q_6\}$	$\{q_2, q_7\}$	$\{q_3, q_6\}$
$\{q_2, q_7\}$	$\{q_3, q_6\}$	$\{q_4, q_4\}$

A and B are equivalent

Ex



States	c	d
$\{q_1, q_4\}$	$\{q_1, q_4\}$	$\{q_2, q_5\}$
$\{q_2, q_5\}$	$\{q_3, q_7\}$	$\{q_1, q_6\}$
$\{q_3, q_6\}$	$\{q_2, q_7\}$	$\{q_3, q_6\}$

A and B aren't 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 a 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 may be true.

(i) $xy^iz \in A$ for every $i \geq 0$
(ii) $|y| > 0$
(iii) $|xy| \leq P$

To prove that a language is not regular using PUMPING LEMMA, follow the below steps:
(We prove using contradiction)

(i) Assume that A is regular
(ii) It has to have a pumping length (say P)
(iii) All strings longer than P can be pumped $|s| \geq P$
(iv) Now find a string 's' in A such that $|s| \geq P$
(v) Divide s into x, y, z
(vi) Show that $xy^iz \notin A$ for some i
(vii) Then consider all ways that s can be divided into x y z
(viii) Show that none of these can satisfy all 3 pumping conditions at the same time.
(ix) s cannot be pumped == contradiction

Q Using pumping lemma, prove that the language $A = \{a^n b^n \mid n \geq 0\}$ is not regular

Assume that A is regular
Pumping length = P

$s = a^P b^P \Rightarrow s = aaaaaaabbbbbbb$

Case 1: y is in 'a' part

aa aaaaa abbbbbbb
x y z

$xy^iz \Rightarrow xy^2z$
aaaaaaaabbbbbbb
$11 \neq 7$

Case 2 : Y is in 'b' part

aaaaaaaabbbbbbb
X Y Z

$XY^0Z \Rightarrow XZ$

aaaaaaaabbbbbbb

7 ≠ 11

Case 3: Y is in 'a' and 'b' part

aaaaaaaabbbbbbb
X Y Z

$XY^0Z \Rightarrow XY^1Z$

aaaaaaaabbbbbbb

$a^n b^n$

$|XY| \leq P$

■ Pumping Lemma to prove that the language $A = \{yy \mid y \in \{0,1\}^*\}$ is not regular

Assume that A is regular

Then it must have pumping length = P

$S = 0^P 1 0^P 1$

X Y Z

P=7

00000001000000001

X Y Z

$XY^iZ \Rightarrow XY^2Z$

000000000010000 0001 $\in A$

$|Y| \geq 0$

$|XY| \leq P$ A is not regular

Set of all string over $\Sigma = \{a\}$

- $(a+b)^* \longrightarrow \varepsilon, a, b, ab, aa, ba, bb, \ldots$

  ↓

  Set of all strings over $\Sigma = \{a,b\}$

- $aa^* \longrightarrow a^+$
- $a^*a \longrightarrow a^+$
- $(a+\varepsilon)^+ \longrightarrow \varepsilon, a, a^2, \ldots \longrightarrow a^*$
- $a^*b^* \longrightarrow \varepsilon, a, b, aa, ab, bb \ldots \longrightarrow \{a^m b^n \mid m,n \geq 0\}$

■ How to write regular expression ?

① $L = \{\cdot\}$ over $\Sigma = \{a,b\}$

$R = \phi.a = \phi = \phi.b$
$= \phi.0^*$
$= (ab).\phi$

② $L = \{\varepsilon\}$ over $\Sigma = \{a,b\}$

$R = \varepsilon = a^0 = b^0 = a^0b^0 = \varepsilon^* = \phi^*$

③ L = Set of all strings over $\Sigma = \{a,b\}$

$R = (a+b)^* = \Sigma^*$
$= (a+b)^+ + \varepsilon$
$= (a^*+b)$

④ $L = \{w \mid w \in \{a,b\}^*, |w| > 0\}$

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

⑤ L = Set of all binary strings starting with 1.

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

⑥ L = Set of all binary strings ending with 1.

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

⑦ $L = \{w \mid w \in \{a,b\}^*, w \text{ contains 'a'}\}$

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

⑧ $L = \{w \mid w \in \{a,b\}^*, |w| = 2\}$
$=$ Set of all 2 length strings over $\Sigma = \{a,b\}$

$R = (a+b)(a+b) = aa + ab + ba + bb$
$= \Sigma \cdot \Sigma$
$= \Sigma^2$
$= (a+b)^2$

(20) $L = \{ w \mid w \in \{a,b\}^* , n_a(w) = odd \}$ → Number of a = odd

$R = [b^* \mid b^* ab^* ab^*]^* b^* ab^*$

$= [b^* ab^* [b^* ab^* ab^*]^* b^*]$

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

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

$R = b^* (b^* ab^* ab^*)^* b^*$

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

$R = b^* (ab^* ab^*)^*$

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

(22) $L = \{ w,x,y \mid w,x,y \in \{a,b\}^* , |w| = 1, x = a \}$

(23) L = 3rd symbol from begining 'a' over $\Sigma = \{a,b\}$

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

$R = [(a+b)^2 a (a+b)^*]$

(24) L = 3rd symbol from ending is 'a' over $\Sigma = \{a,b\}$

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

$R = [(a+b)^2 a (a+b)^2]$

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

$R = (00)^*$

$L = \{ w \mid w \in \{0,1\}, n_0(w) = even , n_1(w) = 0 \}$

- $\Sigma = \{0\}$

$R = (00)^*$

$L = \{ w \mid w \in \{0\}, n_0(w) = even \}$

- $L = \{ w \in (a,b)^* : \#a(w) \leq 3 \}$ → na

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

- $L_1 = a^* + b^*$ $\quad L_2 = a^* b^*$

$L_1^* = (a^* + b^*)^*$ $\quad L_2^* = (a^* b^*)^*$

L_1^*: $\varepsilon, a, b, aa, ba$ $\quad$ L_2^*: $\varepsilon, a, b, aa, ba$

$$L_1^* = L_2^*$$

Note: $a^* + b^* \neq a^* b^*$

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

- $L_1 = a^+ + b^+$ $\quad L_2 = a^* + b^*$

$[L_1 \cup L_2 = L_2]$

- $L_1 = a^+$ $\quad L_2 = a^*$

$L_1^* = (a^+)^* = \varepsilon, a, aa, aaa, \ldots$

$L_2^* = (a^*)^+ = \varepsilon, a, aa, aaa, \ldots$

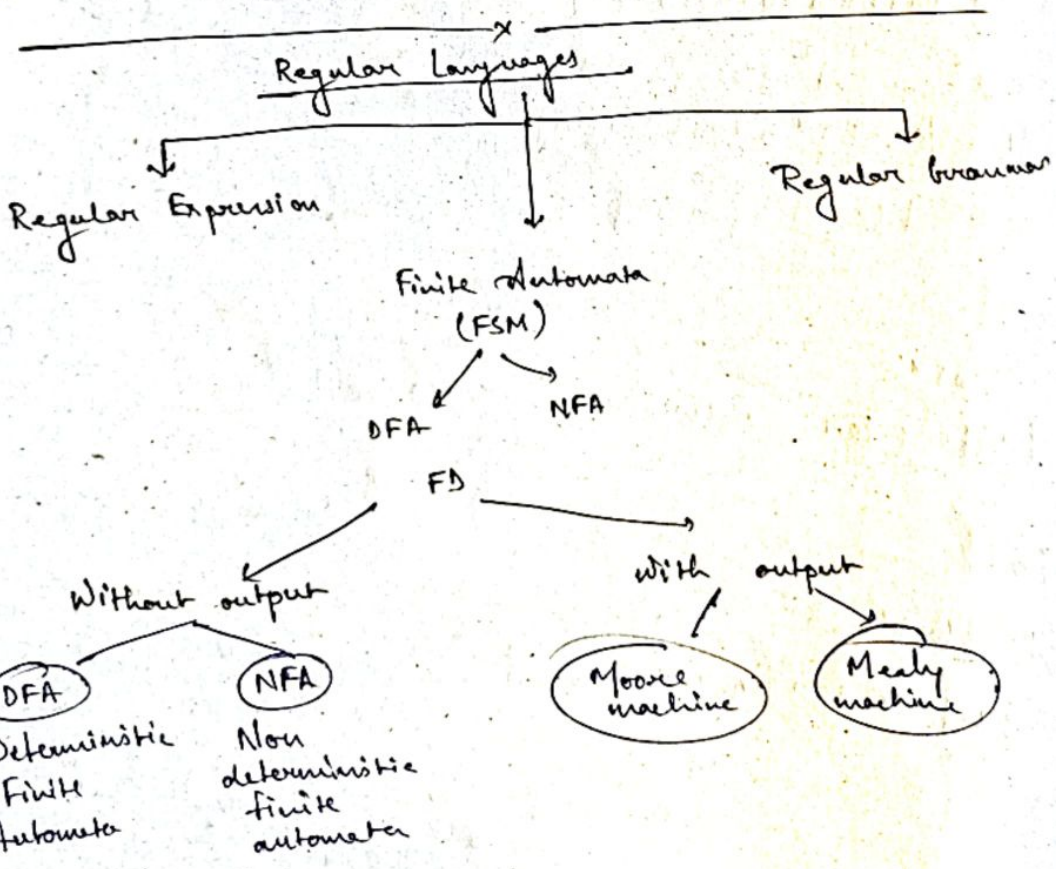
$$\boxed{L_1^* = L_2^+}$$

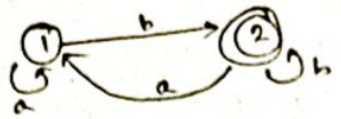
- $A = aa^*$ $\quad B = bb^*$

$(A \cup B)^* = (aa^* + bb^*)^*$

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

$= (a+b)^*$





$Q = \{1, 2\}$ = Set of State

$\Sigma = \{a, b\}$ = Set of Symbols

q_0 = Initial State = $\{1\}$

F = Final State = $\{2\}$

δ =

	a	b
1	{1}	{2}
2	{1}	{2}

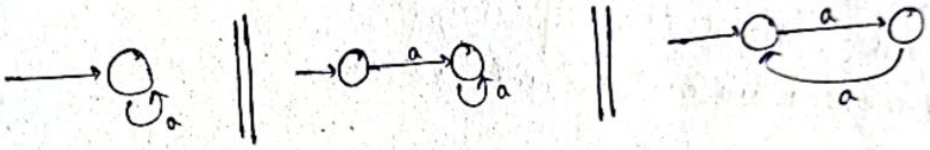
■ If DFA has 5 states and 3 input symbol, then how many transition are there?

State × Symbol = Transition

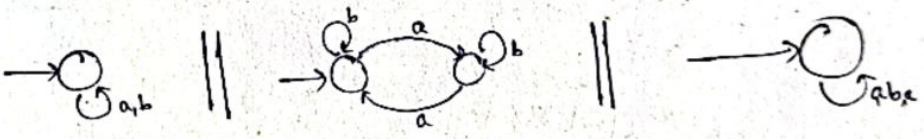
$5 \times 3 = 15$ Ans.

① $L = \phi$ over $\Sigma = \{a\}$

No final state for ϕ

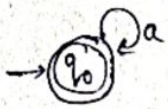


② $L = \phi$ over $\Sigma = \{a, b\}$



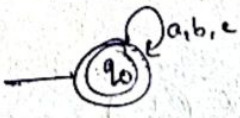
③ $L = \Sigma^*$ over

(i) $\Sigma = \{a\}$

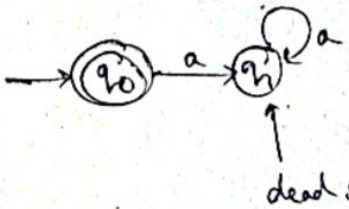


(ii) $\Sigma = \{a, b\}$

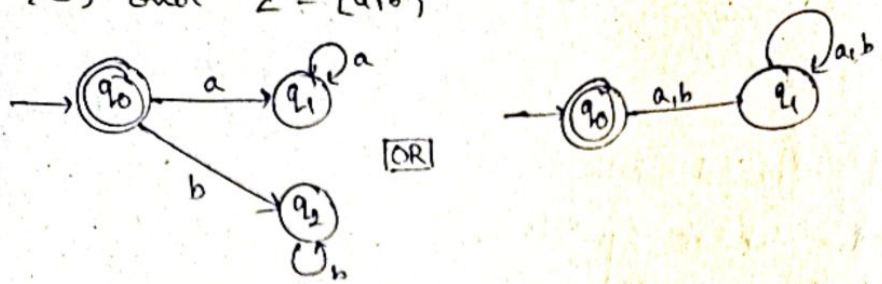
(iii) $\Sigma = \{a, b, c\}$



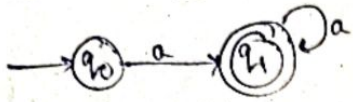
④ $L = \{\varepsilon\}$ over $\Sigma = \{a\}$



$L = \{\varepsilon\}$ over $\Sigma = \{a, b\}$

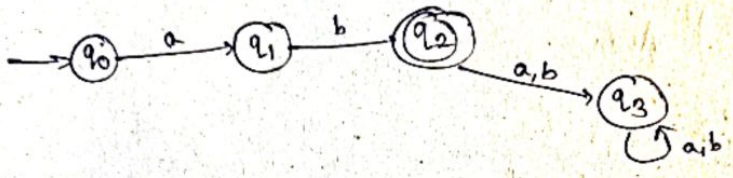


(5) $L = \Sigma^*$ over $\Sigma = \{a\}$



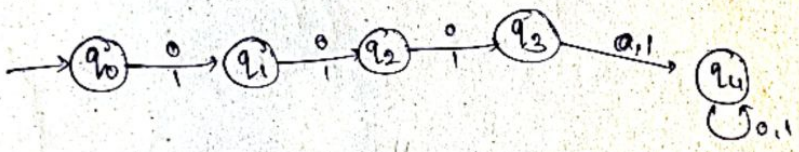
Number of State = length + 1

$L_1 = \{w \mid w \in \{a,b\}^*, |w| = 2\}$



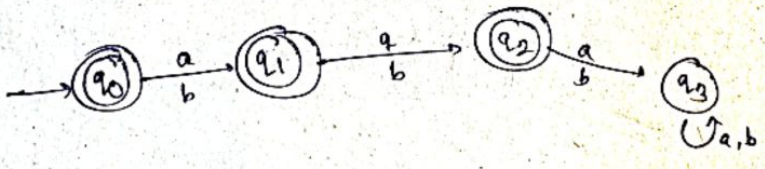
(1) Set of all 3 length binary strings

$L = \{000, 001, 010, 011, 100, 101, 110, 111\}$



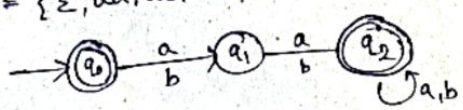
(2) $L = \{w \mid w \in \{a,b\}^*, |w| \leq 2\}$

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



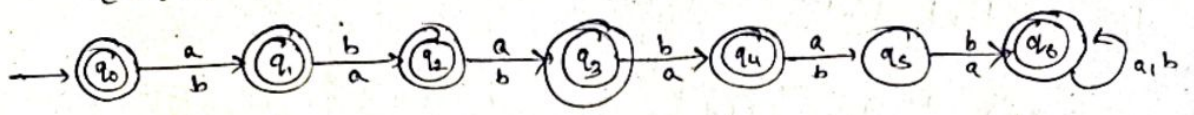
(3) $L = \{w \mid w \in \{a,b\}^*, |w| \geq 2\}$

$L = \{\varepsilon, aa, ab, ba, bb, aaa, \ldots\}$



④ $L = \{ w \mid w \in \{a,b\}^*, |w| \neq 5 \}$

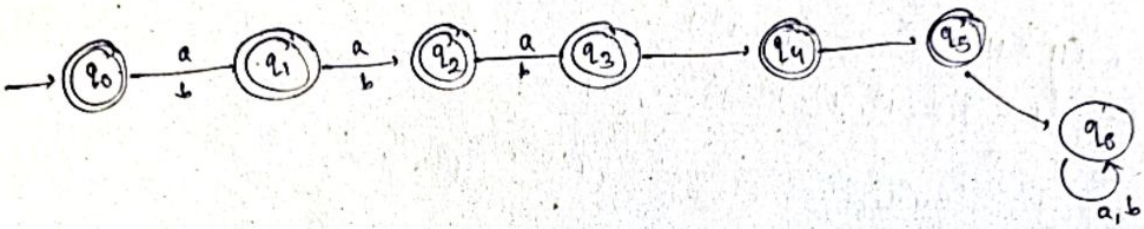
$L = \{\epsilon, a, b, aa, ab, ba, bb, aaa \ldots\} - \{ccccc \ldots 11111\}$



⑤ $L = \{ w \mid w \in \{a,b\}^*, |w| \neq 7 \}$

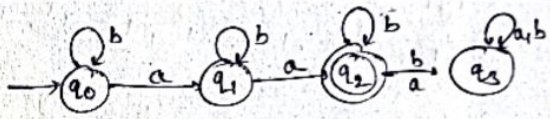
⑥ $L = \{ w \mid w \in \{a,b\}^*, |w| \neq 9 \}$

⑦ $L = \{ w \mid w \in \{a,b\}^*, |w| < 6 \}$



$L_1 = \{ w \mid w \in \{a,b\}^*, n_a(w) = 2 \}$

$R = [b^* a b^* a b^*]$



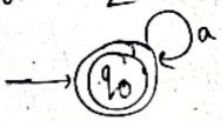
$L_2 = \{ w \mid w \in \{a,b\}^*, n_a(w) \leq 2 \}$

$R = [b^* (a+\epsilon) b^* (a+\epsilon) b^*]$

$L_3 = \{ w \mid w \in \{a,b\}^*, n_a(w) \geq 2 \}$

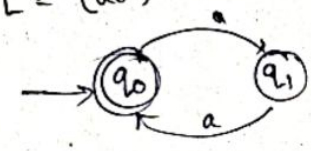
$R = [(a+b)^* a (a+b)^* a (a+b)^*]$

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



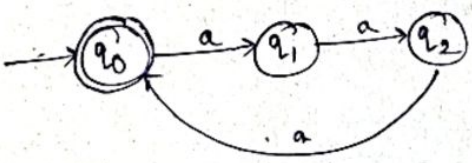
$L = \{a, aa, aaa, \ldots\}$

- $L = (aa)^*$



$L = \{aa, aaaa, aaaaaa, \ldots\}$

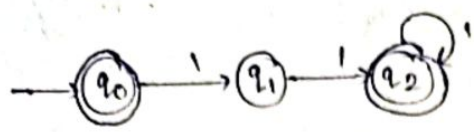
- $L = (aaa)^*$



$L = \{aaa, aaaaaa, \ldots\}$

- $L = \{(11+111)^*\}$ $L = \{(1)^*$ Not $1\}$

  $L = \{\varepsilon, 1^2, 1^3, 1^4, 1^5, 1^6, \ldots\}$

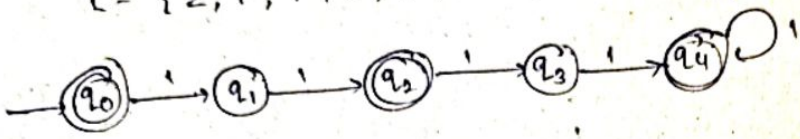


- $(11+1111)^*$

  $L = \{11, 1111, \varepsilon, 111111, 1^8, 1^{10}, \ldots\}$

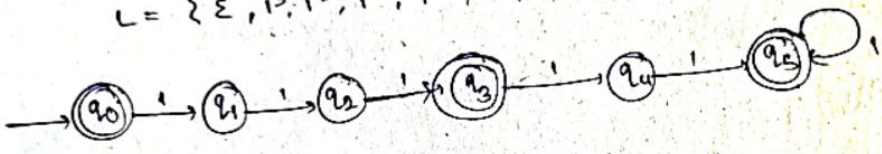
- $(11+11111)^* = (1^2+1^5)^*$

  $L = \{\varepsilon, 1^2, 1^4, 1^5, 1^6, \ldots\}$



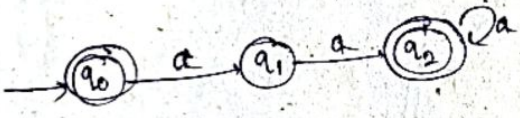
- $(111+11111)^*$

  $L = \{\varepsilon, 1^3, 1^5, 1^6, 1^8, 1^9, 1^{10}, \ldots\}$



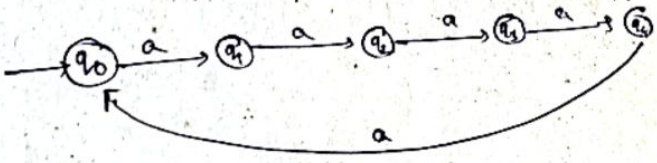
- $(a^{\text{prime}})^*$

  $L = \{\varepsilon, a^2, a^3, a^5, a^7, a^{11}, \ldots\}$



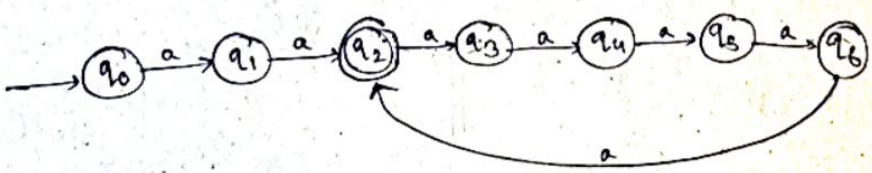
- $\{a^{5n} \mid n \geq 0\}$

  $L = \{\varepsilon, a^5, a^{10}, a^{15}, \ldots\}$

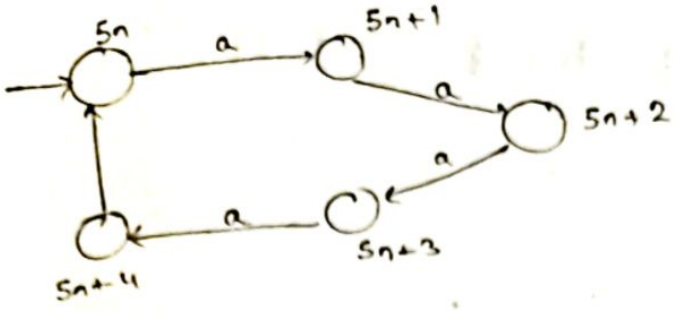


- $\{a^{5n+2} \mid n \geq 0\}$

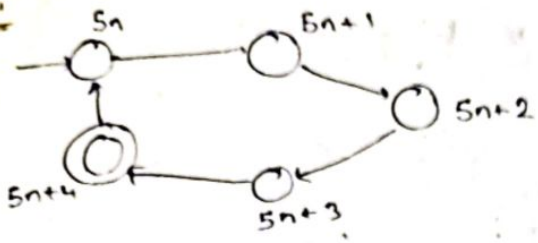
  $L = \{a^2, a^7, a^{12}, \ldots\}$



- $5n+k$ // Make the $5n+k$ as final



- $5n+4$

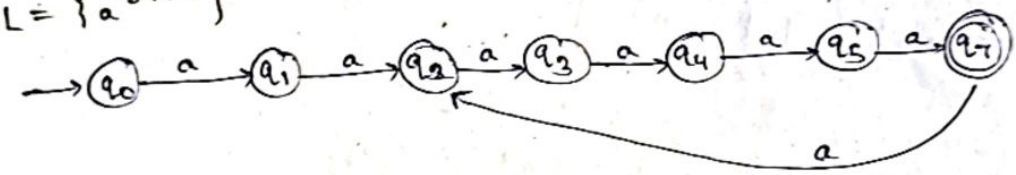


General formula: $\{a^{k_1 n + k_2} \mid n \geq 0\}$

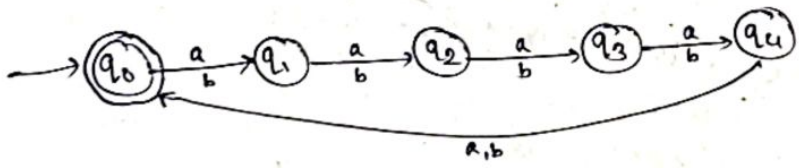
(i) If $k_2 < k_1$ ⟶ k_1 states
q_{k_2} ⟶ final states

(ii) If $k_2 \geq k_1$ ⟶ $k_2 + 1$ state

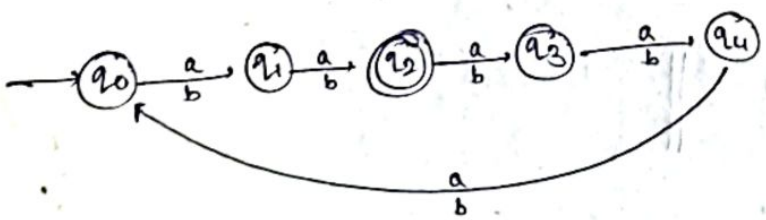
- $L = \{a^{5n+6}\}$



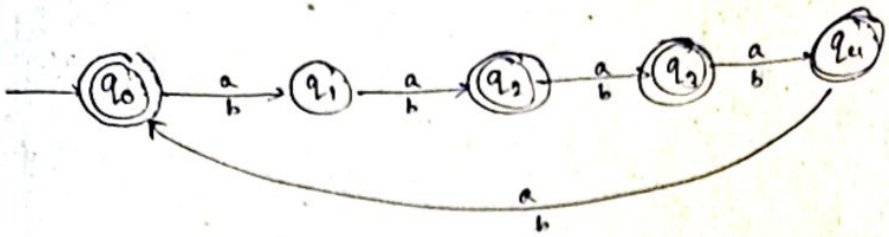
- $L = \{w \mid w \in \{a,b\}^*$, $|w|$ is divisible by $5\}$
⟶ $|w| \% 5 = 0$
$|w| = 5n$, $n \geq 0$



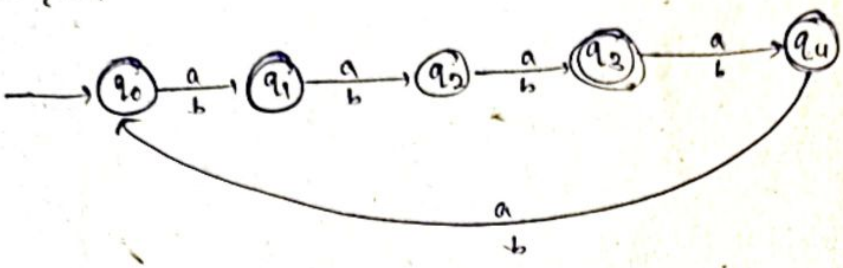
- $L = \{w \mid w \in \{a,b\}^*$, $|w| \% 5 = 2\}$
⟶ $|w| \% 5 = 2$
$|w| = 5n+2$



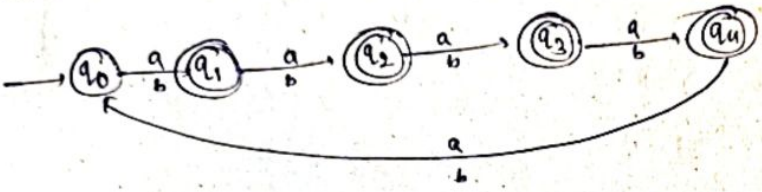
- $\{w \mid w \in \{a,b\}^*, (w \% 5 \gg 2)\}$
  $5n+2$



- $\{w \mid w \in \{a,b\}^*, (w \% 5) \neq 2\}$

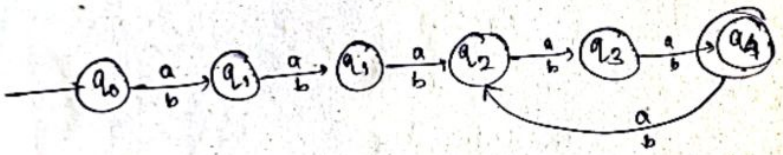


- $|w|$ not divisible by 5



- $\{w \mid w \in \{a,b\}^*, |w| = 3n+5\}$

$n=0$ ⑤
$n=1$ 8
$n=2$ 11



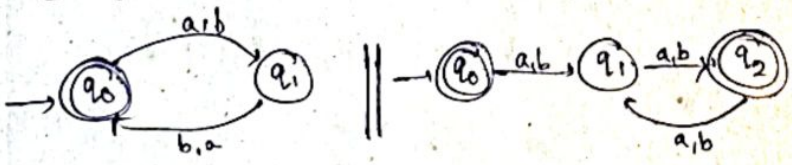
General formula

$\{w \mid w \in \{a,b\}^*, |w| = K_1 n + K_2, n \gg 0\}$

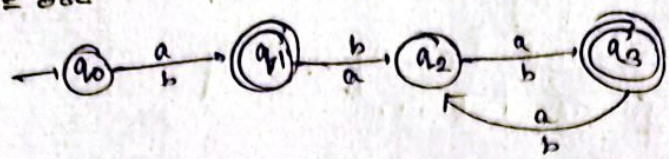
- If $K_2 < K_1 \longrightarrow K_1$ state
- If $K_1 \gg K_2 \longrightarrow K_2+1$ state

- $\{w \mid w \in \{a,b\}^*, |w| = \text{even}\}$

$L = \{0, aa, ab, ba, bb, aaaa, abab, babb, \ldots\}$

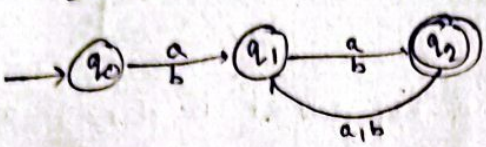


- $|w|$ = odd L = {a, [illegible]}



- $\{w \mid w \in \{a,b\}^+ \;.\; |w| = \text{even}\}$

  $L = \{aa, ab, ba, bb, aabb, abba \ldots\}$



Regular Grammar

Noam Chomsky gave a mathematical model of grammar which is effective for writing computer languages.

- There are four types of grammar:

Grammar type	Grammar Accepted	Language Accepted	Automation
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 automata
Type 3	Regular Grammar	Regular language	Finite State automaton

Grammar

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

V = Set of variables or non terminals symbols

T = Set of Terminal Symbols

S = Start Symbol

P = Production rules for terminals and non terminals

A production rules has the form $\alpha \rightarrow \beta$ where α and β are strings on VUT and atleast one symbol of α belongs to V.

Ex: $Gr = (\{S, A, B\}, \{a, b\}, S, \{S \rightarrow AB, A \rightarrow a, B \rightarrow b\})$

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

$T = \{a, b\}$

$S = S$

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

$S \rightarrow AB$
$\rightarrow aB$
$\rightarrow ab$

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

$A \rightarrow xB$
$A \rightarrow x$

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

Left Linear Grammar

A grammar is said to be left Linear if all productions are of the form $A \rightarrow Bx$
$A \rightarrow x$

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

Ex: $S \to abS \mid b$ — RLG

$S \to Sbb \mid b$ — LLG

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.

Q: Consider the grammar $G_1 = (\{S, A\}, \{a, b\}, S, \{S \to aAb, aA \to aaAb, A \to \in\})$

$S \to aAb$
$\to aaAbb$
$\to aaaAbbb$
$\to aaabbb$

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

$S \to AB$
$\to aB$
$\to ab$

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

$S \to AB$ $\to ab$

$S \to AB$ $\to aABB$ $\to aabb$

$S \to AB$ $\to aAb$ $\to aab$

$S \to AB$ $\to abB$ $\to abb$

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

$= \{a^m b^n \mid m \geq 0 \text{ and } n \geq 0\}$

Context Free Language

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

The set of all CFL 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 / Non terminal symbols

Σ = Set of terminal symbols

S = start symbol

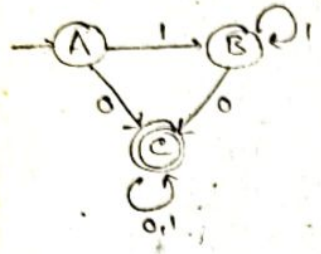
P = Production rule.

CFG has production rule of the form $A \to \alpha$ where $\alpha = \{V \cup \Sigma\}^*$ and $A \in V$

Ex: For generating a language that generates equal number of a's and b's in the form $a^n b^n$, the context free grammar will be defined as:

$$G = \{(S, A), (a, b), (S \to aAb, A \to aAb \mid \epsilon)\}$$

$S \to aAb$
$\to aaAbb \quad (A \to aAb)$
$\to aaaAbbb \quad (A \to aAb)$
$\to aaabbb \quad (A \to \epsilon)$
$\to a^3 b^3 \Rightarrow a^n b^n$

Q 

$C = A0 + B0 + C0 + C1$ — (i)
$B = A1 + B1$ — (ii)
$A = \epsilon$ — (iii)

(ii) $B = A1 + B1$
$B = \epsilon 1 + B1$
$B = 1 + B1$
$B = 11^*$
} By Arden's Theorem

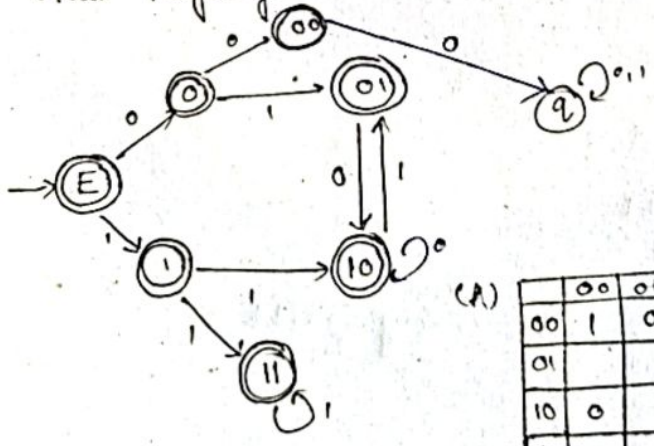
(i) $C = A0 + B0 + C0 + C1$
$C = \epsilon 0 + 11^*0 + C(0+1)$
$C = \underbrace{0 + 11^*0}_{R} + \underbrace{C}_{q}\underbrace{(0+1)}_{P}$
$R \quad q \quad R \quad P$

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

Q Consider the languages $L_1 = \phi$ and $L_2 = \{a\}$. Which one represents $L_1 L_2^* \cup L_1^*$?

$L_1 L_2^* \cup L_1^*$
$= \phi a^* \cup L_1^*$
$= \phi \cup \phi^*$
$= \epsilon$

Q Consider the set of strings on $\{0, 1\}$ in which every substring of 3 symbols has atmost two zeroes. For example 001110 and 011001 are in the language but 100010 is not. All strings of length less than 3 are accepted and in the language. A partially completed DFA that accepts this language is shown below.



The missing arcs in the DFA are
(a) A
(b) B
(c) C
(d) D

(A)

	00	01	10	11	q
00	1	0			
01				1	
10	0				
11			0		

(B)

	00	01	10	11	q
00		0			1
01		1			
10			·	0	
11		0			

(C)

	00	01	10	11	q
00		1			0
01		1			
10			0		
11		0			

(D)

	00	01	10	11	q
00		1			0
01				1	
10	0				
11			0		

Q Which one of the following languages over ... is described by the regular expression: (0+1)*0(0+1)*0 (0+1)*

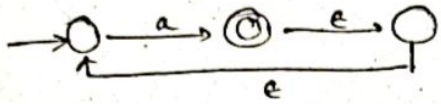
(A) The set of all strings containing the substring 00

(B) The set of all strings containing atmost two ~~zero~~ 0's.

(C) The set of all strings containing atleast two 0's.

(D) The set of all strings that begin and end with either 0 or 1.

Q What is the complement of the language accepted by the NFA shown below ?



L = a+

Complement of a+ will be ∈

Method to find whether a string belongs to a grammar or not

(i) Start with a start symbol and choose the closest production that matches to the given string.

(ii) Replace the variables with its most appropriate production. Repeat the process until the string is generated or until no other productions are left.

Q Verify whether the grammar S → 0B|1A, A → 0|0S|1AA, B → 1|1S|0BB generates the string 00110101

S → 0B (S → 0B)
→ 00BB (B → 0BB)
→ 001B (B → 1)
→ 0011S (B → S)
→ 00110B (~~B → 0~~) (S → 0B)
→ 001101S (B → 1S)
→ 0011010B (S → 0B)
→ 00110101 (B → 1)

Q Verify whether the grammar S → aAb, A → aAb | ^ generates the string aabbb

S → aAb
→ aaAbb (A → aAb)
→ aabb (A → ^)

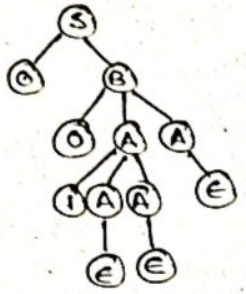
aaAbb (A → aAb)
→ aaaAbbb (A → aAb)
→ aaabbb (A → ^)

Doesn't generate the string aabbb

Derivation Tree

A derivation tree or Parse tree is an ordered rooted tree that graphically represents the semantic information of strings derived from a context free grammar.

Ex: For the grammar $G = \{V, T, P, S\}$ where $S \to 0B$, $A \to 1AA \mid \epsilon$, $B \to 0AA$



Root vertex: Must be labelled by the start symbol

Vertex: Labelled by non terminal symbols

Leaves: Labelled by terminal symbols or ϵ

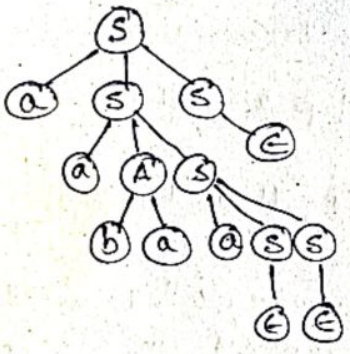
Left Derivation Tree

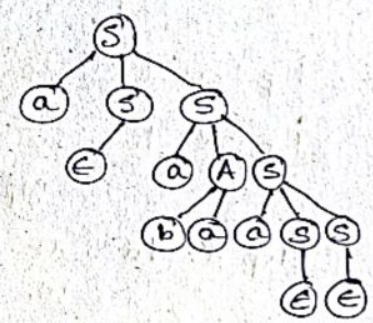
A left derivation tree is obtained by applying the production to the leftmost variable in each step.

Right Derivation Tree

A right derivation tree is obtained by applying the production to the rightmost variable in each step.

For generating the string aabaa from the grammar $S \to aAS \mid aSS \mid \epsilon$, $A \to SbA \mid ba$





Ambiguous Grammar

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

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

The string a + a * b can be generated as

$S \to S+S$
$\to a+S$ $(S \to a)$
$\to a+S*S$ $(S \to S*S)$
$\to a+a*S$ $(S \to a)$
$\to a+a*b$ $(S \to b)$

$S \to S*S$
$\to S+S*S$ $(S \to S+S)$
$\to a+S*S$ $(S \to a)$
$\to a+a*S$ $(S \to a)$
$\to a+a*b$ $(S \to b)$

Thus grammar is ambiguous

Simplification of context free grammar

Reduction of CFGs

In CFGs, sometimes all the production rules and symbols are not needed for the 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 CFGs.

Simplification consists of following steps:

(I) Reduction of CFGs
(II) Removal of Unit productions
(III) Removal of NULL productions.

REDUCTION OF CFGs

CFGs are reduced in two phases:

Phase 1: Derivation of an equivalent grammar G' from the CFG such that each variable derives some terminal string.

Derivation Procedure:

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

Step 2: Include symbols W_{i+1} that derives W_i

Step 3: Increment i and repeat Step 2 until $W_{i+1} = W_i$

Step 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 the sequential form.

Derivation procedure:

Step 1: Include the start symbol in Y_1 and initialize $i = 1$.

Step 2: Include all symbols Y_{i+1} that can be derived from Y_i and include all production rules that have been applied.

Step 3: Increment i and repeat Step 2 until $Y_{i+1} = Y_i$

Ex: Find a reduced grammar equivalent to the grammar G, having production rules P: S → AC|B, A → a, C → c|Bc, E → aA|ε

(M) T = {a, c, ε}

W_1 = {A, C, E}

W_2 = {A, C, E, S}

W_3 = {A, C, E, S}

G' = {(A, C, E, S), {a, c, ε}, P, (S)}

P: S → AC, A → a, C → c, E → aA|ε

(P2) Y_1 = {S}

Y_2 = {S, A, C}

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

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

G'' = {(A, C, S), {a, c}, P, {S}}

P: S → AC, A → a, C → c

Removal of Unit Productions

Any production rule of the form A → B where A, B ∈ Non terminals is called Unit Production.

Procedure for removal

Step 1: To remove A → B, add production A → x to the grammar rule whenever B → x occurs in the grammar [x ∈ Terminal, x can be NULL]

Step 2: Delete A → B from the grammar

Step 3: Repeat from Step 1 until all unit productions are removed

Ex: Remove unit productions from the grammar whose production rule is given by P: S → XY, X → a, Y → Z|b, Z → M, M → N, N → a

P: S → XY, X → a, Y → Z|b, Z → M, M → N, N → a

Y → Z Z → M M → N

i) Since N → a, we add M → a

P: S → XY, X → a, Y → Z|b, Z → M, M → a, N → a

ii) Since M → a, we add Z → a

P: S → XY, X → a, Y → Z|b, Z → a, M → a, N → a

iii) Since Z → a, we add Y → a

P: S → XY, X → a, Y → a|b, Z → a, M → a, N → a

Remove the unreachable symbols

P: S → XY, X → a, Y → a|b

Removal of NULL productions

Procedure for removal

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

Step 2: Replace each occurances of 'A' in each of these productions with ϵ

Step 3: Add the resultant productions to the grammar.

Ex: Remove NULL predictions from the following grammar

$S \to ABAC$, $A \to aA \mid \epsilon$, $B \to bB \mid \epsilon$, $C \to c$

$A \to \epsilon$, $B \to \epsilon$

i) To eliminate $A \to \epsilon$

$S \to ABAC$

$S \to ABC \mid BAC \mid BC$ $\quad (A \to \epsilon)$

$A \to aA$

$A \to a$

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

$A \to aA \mid a$

$B \to bB \mid \epsilon$

$C \to c$

ii) To eliminate $B \to \epsilon$

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

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

$A \to aA \mid a$

$B \to bB \mid b$

$C \to c$

CHOMSKY NORMAL FORM

In chomsky normal form (CNF) we have a 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 the following forms.

$A \to a$

$A \to BC$

where A, B and C are non terminals and a is a terminal.

Steps to convert a given CFG to Chomsky Normal Form

(i) If the start symbol S occurs on some right side, create a new start symbol S and a new production $S' \to S$

(ii) Remove NULL productions

(iii) Remove Unit productions

(iv) Replace each production $A \to B_1 \ldots B_n$ where $C \to B_2 \ldots B_n$. Repeat this step for all productions having two or more symbols on the right side.

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

Repeat this step for every production which is of the form $A \to aB$

Ex: Convert the following CFG to CNF

P: $S \to ASA \mid aB$, $A \to B \mid S$, $B \to b \mid \epsilon$

P: $S' \to S$, $S \to ASA \mid aB$, $A \to B \mid S$, $B \to b \mid \epsilon$

Since S appears in RHS, we add a new state S' and $S' \to S$ is added to the production.

(ii) Remove the NULL predictions: $B \to \epsilon$, $A \to \epsilon$

After removing $B \to \epsilon$: P: $S' \to S$, $S \to ASA \mid aB \mid a$, $A \to B \mid S \mid \epsilon$, $B \to b$

After removing $A \to \epsilon$: P: $S' \to S$, $S \to ASA \mid aB \mid a \mid AS \mid SA \mid S$, $A \to B \mid S$, $B \to b$

(iii) After removing unit productions: $S \to S$, $S' \to S$, $A \to B$, $A \to S$

After removing $S \to S$: P: $S' \to S$, $S \to ASA \mid aB \mid a \mid AS \mid SA$, $A \to B \mid S$, $B \to b$

After removing $S' \to S$: P: $S' \to ASA \mid aB \mid a \mid AS \mid SA$
$S \to ASA \mid aB \mid a \mid AS \mid SA$
$A \to B \mid S$, $B \to b$

After removing $A \to B$: P: $S' \to ASA \mid aB \mid a \mid AS \mid SA$
$S \to ASA \mid aB \mid a \mid AS \mid SA$
$A \to b \mid S$, $B \to b$

After removing $A \to S$: P: $S' \to ASA \mid aB \mid a \mid AS \mid SA$
$S \to ASA \mid aB \mid a \mid AS \mid SA$
$A \to b \mid ASA \mid aB \mid a \mid AS \mid SA$
$B \to b$

(iv) Now find out the production that has more than two variables in RHS

$S' \to ASA$, $S \to ASA$ and $A \to ASA$

After removing these, we get, P: $S' \to AX \mid aB \mid a \mid AS \mid SA$
$S \to AX \mid aB \mid a \mid AS \mid SA$
$A \to b \mid AX \mid aB \mid a \mid AS \mid SA$
$B \to b$
$X \to SA$

Now change the productions $S' \to aB$, $S \to aB$ and $A \to aB$

∴ P: $S' \to AX \mid YB \mid a \mid AS \mid SA$
$S \to AX \mid YB \mid a \mid AS \mid SA$
$A \to b \mid AX \mid YB \mid a \mid AS \mid SA$
$B \to b$
$X \to SA$
$Y \to a$

Greibach Normal Form

A CFG is in Greibach normal form if the productions are in the following forms

$A \rightarrow b$

$A \rightarrow bC_1C_2 \cdots C_n$

where $A, C_1 \ldots C_n$ are non-terminals and b is a terminal

- Steps to convert a given CFG to GNF

Step 1: Check if the given CFG has any unit productions or Null productions and remove if there are any.

Step 2: Check whether the CFG is already in Chomsky Normal form (CNF) and convert it into CNF if it is not.

Step 3: Change the names of the Non terminal symbols into some A_i in ascending order of i

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

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

We get $A_1 \rightarrow A_2A_3 \mid A_4A_4$
$A_4 \rightarrow b \mid A_1A_4$
$A_2 \rightarrow b$
$A_3 \rightarrow a$

Step 4: Alter the rules so that the Non terminals are in ascending order. Such that, if the production is of the form $A_i \rightarrow A_jx$, then $i < j$ and should never be $i \geq j$

$A_4 \rightarrow b \mid A_1A_4$
$A_4 \rightarrow b \mid A_2A_3A_4 \mid A_4A_4A_4$
$A_4 \rightarrow b \mid bA_3A_4 \mid A_4A_1A_4$
↳ left recursion

Step 5: Remove left recursion
- Introduce a new variable to remove left recursion

$A_4 \rightarrow b \mid bA_3A_4 \mid A_4A_4A_4$
$Z \rightarrow A_4A_4Z \mid A_4A_4$
$A_4 \rightarrow b \mid bA_3A_4 \mid bZ \mid bA_3A_4Z$

Now the grammar is $A_1 \rightarrow A_2A_3 \mid A_4A_4$
$A_4 \rightarrow b \mid bA_3A_4 \mid bZ \mid bA_3A_4Z$
$Z \rightarrow A_4A_4 \mid A_4A_4Z$
$A_2 \rightarrow b$
$A_3 \rightarrow a$

$A_1 \rightarrow bA_3 \mid bA_4 \mid bA_3A_4A_2 \mid bZA_4 \mid bA_3A_4ZA_4$

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

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

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Pumping Lemma (for CFL)

Pumping Lemma (for CFL) is used to prove that a language is not context free

If A is a context Free language, 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 the following conditions may be 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 CFL using pumping lemma (for CFL) follow the steps given below (prove using CONTRADICTION)

(i) Assume that A is context free

(ii) It has to have a pumping length (say P)

(iii) All strings longer than P can be pumped. $|S| \geq P$

(iv) Now find a string S in A such that $|S| \geq P$

(v) Divide S into uvxyz

(vi) Show that $uv^ixy^iz \notin A$ for some i

(vii) Then consider the ways that S can be divided into uvxyz

(viii) Show that none of these can satisfy all the pumping conditions at the same time.

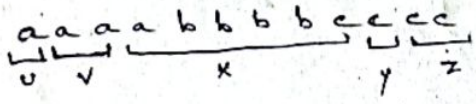
(ix) S cannot be pumped == CONTRADICTION

Ex Show that $L = \{a^Nb^Nc^N \mid N \geq 0\}$ is NOT Context Free

- Assume that L is context free
- L must have a pumping length (say P)
- Now we take a string S such that $S = a^Pb^Pc^P$
- We divide S into parts u v x y z.

Eg. P = 4 , so $a^4b^4c^4$

case 1: v and y each contain only one type of symbol



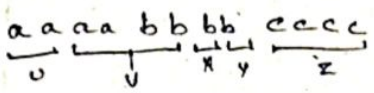
Now for $i=2$, $uv^ixy^iz \Rightarrow uv^2xy^2z$

$\Rightarrow aaaaaabbbbccccc$

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

Hence L is not context free language.

Case II: Either v or y has more than one kind of symbol



Now for $i=2$, $uv^ixy^iz \Rightarrow uv^2xy^2z$

$\Rightarrow aaaaaabbbbbbbcccc$

$a^6b^7c^4 \notin L$

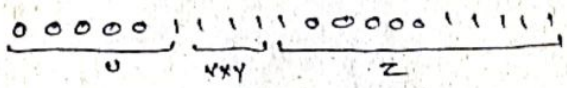
Hence L is not context free Language

Ex: Show that $L = \{ww \mid w \in \{0,1\}^*\}$ is NOT Context Free

- Assume that L is context free
- L must have a pumping length (say P)
- Now we take a string S such that $S = 0^P1^P0^P1^P$
- We divide S into parts uvxyz

Eg. $P=5$, So $S = 0^51^50^51^5$

Case 1: vxy does not straddle a boundary



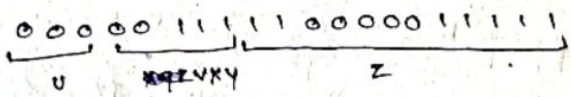
Now for $i=2$, $uv^ixy^iz \Rightarrow uv^2xy^2z$

$\Rightarrow 000001111111100000111111$

$0^51^70^51^5 \notin L$

Hence L is not context free Language

Case 2(a): vxy straddles the first boundary



Now for $i=2$, $uv^ixy^iz \Rightarrow uv^2xy^2z$

$\Rightarrow 00000001111111000000 11111$

$0^71^70^51^5 \notin L$

Hence L is not context free Language.

Case 2(b) : xyz straddles the third boundary

$\underbrace{00000111110000}_{u}\underbrace{00111}_{v\,*\,x\,y}\underbrace{11}_{z}$

For $i=2$, $uv^ixy^iz \Rightarrow uv^2xy^2z$

$\Rightarrow 0000011111100000001111111$

$0^{5,5}0^{7,7} \notin L$

Hence L is not context free language.

Case 3 : xyz straddles the middle point

$\underbrace{00000}_{u}\underbrace{111111}_{v\,x}\underbrace{00000}_{y}\underbrace{111111}_{z}$

For $i=2$, $uv^ixy^iz \Rightarrow uv^2xy^2z$

$\Rightarrow 00000111111100000000111111$

$0^{5,7}0^{7,5} \notin L$

Hence L is not context free language

Pushdown Automata

A pushdown automata (PDA) is a way to implement a context free grammar in a similar way we design Finite Automata

- It is more powerful than FSM
- FSM has less memory but PDA has more memory
- PDA = Finite State Machine + A stack

A stack is a way we arrange elements on top of one another

A stack does two basic operations

Push: A new element is added at the top of stack

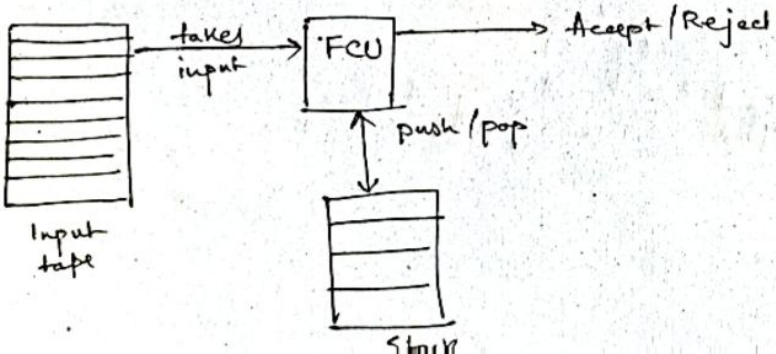
Pop: The top element of the stack is read and removed.

A pushdown automata has 3 components:

(I) An input tape

(II) A finite control unit

(III) A stack with finite size



A pushdown automata is formally defined by 7 tuples as shown below:

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

where

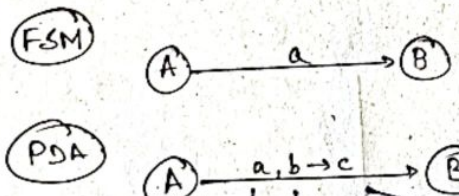
Q = Finite set of States
Σ = Finite set of input symbols
Γ = Finite stack alphabet
δ = Transition function
q_0 = Start State
$\$/Z_0$ = Start Stack Symbol
F = Final / Accepting States

δ takes as argument a triple $\delta(q, a, X)$ where:
(a) q is a state in Q
(b) a is either an input symbol in Σ or $a = \epsilon$
(c) X is a stack symbol that is a member of Γ

The output of δ is finite set of pairs (p, γ) where
- p is a new state
- γ is a string of stack symbols that replaces X at top of the stack.

Ex: If $\gamma = \epsilon$, then stack is popped
If $\gamma = X$, then stack is unchanged
If $\gamma = YZ$, then X is replaced by Z and Y is pushed onto the stack.

- Pushdown Automata (Graphical Notation)



Input Symbol → Maybe ϵ

Symbol on top of stack → This symbol is popped → ϵ means the stack is neither read nor popped.

This symbol is pushed onto the stack → ϵ means nothing is pushed

---

$\delta(q, a, \gamma) = \delta(q', \alpha)$

current state, input, top of stack, new state, string of stack symbols in stack

Q Construct a PDA for $L = \{a^n b^n \mid n \geq 1\}$

$L = \{ab, aabb, aaabbb, \ldots\}$

a	a	a	b	b	b	∈

$\Sigma = \{a, b\}$
$Q = \{q_0, q_1, q_2\}$
q_0 = initial state
z_0 = top of stack
$F = \{q_2\}$

a – push
b – pop (a)
unchange

$(q', \epsilon) \rightarrow$ pop

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

$\delta(q_0, a, a) = (q_0, aaz_0)$ ← top

$\delta(q_0, a, a) = (q_0, aaaz_0)$ ← top

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

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

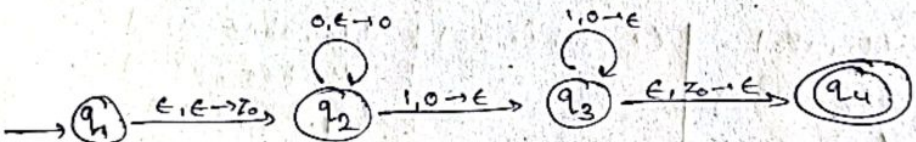
$\delta(q_1, b, a) = (q_1, \epsilon)$

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

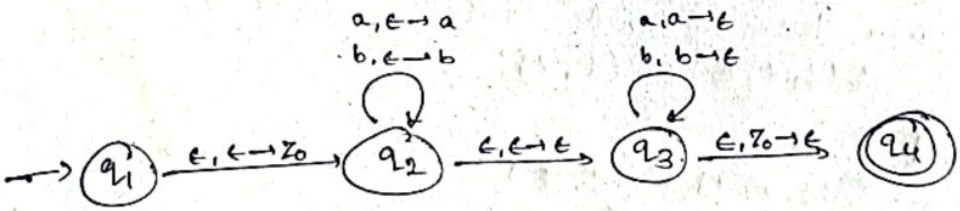
$P = \{\{q_0, q_1, q_2\}, \{a, b\}, \{z_0, a\}, \delta, \{q_0\}, \{z_0\}, \{q_2\}\}$
$Q \quad \Sigma \quad \Gamma \quad \delta, q_0 \quad z_0, \quad F$

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

$L = \{\epsilon, 01, 0011, 000111, \ldots\}$



Q Construct a PDA that accepts even palindromes of the form $L = \{ww^R \mid w = (a+b)^+\}$



Instantaneous description (ID) of PDA

ID of PDA

- Triple (q, w, γ) → content of the stack
  - q → current state
  - w → input string to be processed
- representation (Italian T) → turnstile notation
  - $\vdash$ single move
  - $\vdash^*$ multiple / sequence of moves

Q $L = \{a^n b^n \mid n \geq 1\}$

$L = \{ab, aabb, aaabbb, \ldots\}$

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, a, a) = (q_0, aaZ_0)$
$\delta(q_0, b, a) = (q_1, \epsilon)$ } pop
$\delta(q_1, b, a) = (q_1, \epsilon)$ }
$\delta(q_1, \epsilon, Z_0) = (q_2, \epsilon)$

$(q_0, aaabbb, Z_0) \vdash (q_0, aabbb, aZ_0)$
$\vdash (q_0, abbb, aaZ_0)$
$\vdash (q_0, bbb, aaaZ_0)$
$\vdash (q_1, bb, aaZ_0)$
$\vdash (q_1, b, aZ_0)$
$\vdash (q_1, \epsilon, Z_0)$
$\vdash (q_2, \epsilon)$

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

$L = \{abc, aabcc, aabbcc, aaabccc, \ldots\}$

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, a, a) = (q_0, aaZ_0)$
$\delta(q_0, b, a) = (q_1, aaZ_0)$
$\delta(q_1, c, a) = (q_2, \epsilon)$
$\delta(q_2, \epsilon, Z_0) = (q_3, \epsilon)$

$(q_0, aabcc, Z_0) \vdash (q_0, abcc, aZ_0)$
$\vdash (q_0, bcc, aaZ_0)$
$\vdash (q_1, cc, aaZ_0)$
$\vdash (q_2, c, aZ_0)$
$\vdash (q_2, \epsilon, Z_0)$
$\vdash (q_3, \epsilon)$

Conversion of CFG to PDA

- For non terminal, $A \rightarrow \alpha$
  $\delta(q, \epsilon, A) = (q, \alpha)$
- For terminals
  $\delta(q, a, a) = (q, \epsilon)$

Q Construct PDA for the grammar

$S \rightarrow AB$
$A \rightarrow 0S \mid 10$
$B \rightarrow 1S \mid 1$

$S \rightarrow AB$, $\delta(q, \epsilon, S) = (q, AB)$

$A \rightarrow 0S$, $\delta(q, \epsilon, A) = (q, 0S)$

$A \rightarrow 0$, $\delta(q, \epsilon, A) = (q, 0)$

$B \rightarrow 1S$, $\delta(q, \epsilon, B) = (q, 1S)$

$B \rightarrow 1$, $\delta(q, \epsilon, B) = (q, 1)$

$\delta(0, 0) = (q, \epsilon)$

$\delta(q, 1, 1) = (q, \epsilon)$