Automata Theory

Power set → set of all subsets of a set

$S = \{1, 2\}$

$P(S)$ or $2^S = \{\phi, \{1\}, \{2\}, \{1, 2\}\}$

Sequence → list of objects in a particular order

$(7, 21, 51) \neq (7, 51, 21)$

$(7, 21, 51) \neq (7, 21, 51, 51)$

* consider order and repetitions.

Tuple → sequence of finite number of elements

k - tuple — sequence with k-elements

2 - tuple — pair

Cartesian Product (Cross Product)

$A = \{1, 2\}$

$B = \{3, 4\}$

$A \times B = \{(1,3), (1,4), (2,3), (2,4)\}$

Functions

$f: D \longrightarrow R$ ← range

↑ domain

Alphabet (Σ)

$\Sigma = \{a, b\}$ $\quad \sigma_1 = a \quad \sigma_2 = b$

Empty string = ε

$|\varepsilon| = 0$

$w = abba$

$\#_a w = 2$

$\#_b w = 2$

Σ^k = set of all strings with length k

$\Sigma^0 = \{\varepsilon\}$

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

$\Sigma^2 = \{00, 01, 10, 11\}$

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

Reverse String

$w = 10$ $\quad w^R = 01$

$s = abda$ $\quad s^R = adba$

Substring

- sequence of consecutive letters from a string

$w = 101$

$B(w) = \{\varepsilon, 1, 0, 10, 01, 101\}$

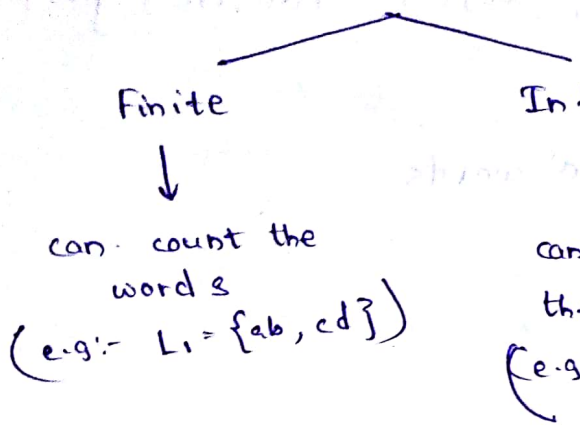
Kleene Closure (star)

Σ^* = The set of strings over that alphabet

$\{0, 1\}^* = \{\varepsilon, 0, 1, 01, 10, 11, 00, 000, \ldots\}$

Language = set of words (strings)

Notation ⟹ L or L_1



* Language is sets. we can apply set operations on languages.

Complement ⟶ all words doesn't belong to the lang.

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

Reverse Language ⟶ with all the reverse words

$$L^R = \{w \mid w^R \in L\}$$

e.g:- $L = \{abc, bc\}$

$L^R = \{cba, cb\}$

* for all binary words Language and reverse language are same

Concatenation ⟶ consider L_1 and L_2

$$L_1 \circ L_2 = L_1 L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\}$$

e.g:- $L_1 = \{ab, cd\}$ $L_2 = \{00, 1\}$

$L_1 L_2 = \{ab00, ab1, cd00, cd1\}$

[* order is important
$L_1 L_2 \neq L_2 L_1$]

$L_1 \circ L_2 \neq L_1 \times L_2$

Formal

e.g:- $A = \{0, 00\}$

$A \circ A = AA = \{00, 000, 0000, 0000\} \Rightarrow |A \circ A| = 3$

$A \times A = \{(0,0), (0,00), (00,0), (00,00)\} \Rightarrow |A \times A| = 4$

Empty language ($\emptyset$) ⟶ lang with '0' words

$L \circ \emptyset = \emptyset \circ L = \emptyset$

$L \circ \{\varepsilon\} = \{\varepsilon\} \circ L = L$

K-iteration : concatenate the lang with itself k times

L^K *

$L^1 = L$

$L^0 = \{\varepsilon\}$

$L_1 = \{\varepsilon, 00, 1\}$

$L_1^2 = \{\varepsilon, 00, 1, 00, 0000, 001, 100, 11\}$

$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$

$L^+ = L^1 \cup L^2 \cup L^3 \cup \ldots$

Languages

Natural Languages

Formal Languages

- a language designed by someone
(eg:- programming language)

Formal Language

(1) List all the words

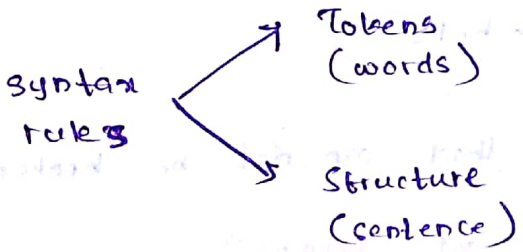
(2) Create set of rules

(3) Use set notation

$L = \{ x \mid x \in$ all words that can be generated using english alphabet $\}$

Alphabet (Σ)

Syntax : linguistic form of sentences in that lang.



Formal Grammar

- generating device which can generate and analyse words / strings of lang.

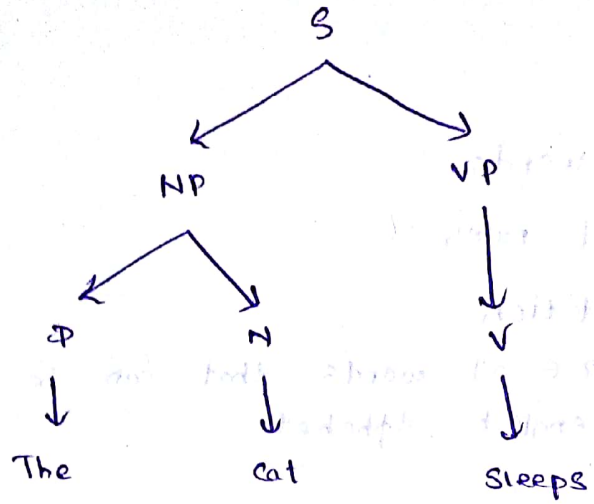
Grammar : Finite set of rules

↓

set of strings } = Formal language (Generated language)

S → NP VP
VP → V
NP → D N
V → sleeps
N → cat
D → The

S - sentence
NP - Noun phrase
VP - verb "
D - determinent
G - Grammer



$$G = (N, \Sigma, P, S)$$

- N - set of non-terminal symbols (eg:- S, NP, VP)
- Σ - set of terminal symbols

  ↓

  a symbol that can not be broken down further

  (eg:- The, cat, sleeps)

$$\Sigma \cap N = \phi$$

- S - a start symbol. $S \in N$
- P - set of production rules

$$v \rightarrow w$$
$$v \in (N \cup \Sigma)^+$$
$$w \in (N \cup \Sigma)^*$$

$$w = uxv$$

$x \rightarrow y$

rule $x \rightarrow y$ is aplicable to string w

$uxv \rightarrow uyv$

uxv derives uyv

uyv is derived from uxv

$w_1 \Rightarrow w_2 \Rightarrow w_3 \Rightarrow \ldots w_n$

$w_1 \Rightarrow^* w_n$

$w_1, w_2, w_3, \ldots w_n$ — Sentential forms of the derivation.

$N = \{S, B\}$

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

$S = S$

P

$S \rightarrow aS$

$S \rightarrow B$

$B \rightarrow bB$

$B \rightarrow \in$

$S \Rightarrow aS \Rightarrow aB \Rightarrow abB \Rightarrow abbB \Rightarrow abb$

$S \Rightarrow^* abb$

G $\quad$ $L(G)$

$G_1 = G_2$

All legal identifiers in Pascal is a language

$\langle id \rangle \longrightarrow \langle letter \rangle \langle rest \rangle$

$\langle rest \rangle \longrightarrow \langle letter \rangle \langle rest \rangle \; / \; \in$

Classes of Grammars

Chomsky Hierarchy

Type 0: no restrictions on the grammer rules

→ All formal grammers

→ Type 0 languages

↓

recognised by Turing Machine

Type 1: context - sensitive grammer

$|\alpha_i| \leq |\beta_i|$

$AB \longrightarrow CBB$ ✓

$ABC \longrightarrow B$ ✗

- context sensitive language identified by linear bounded automation

Type 2 : context free grammar

for all $i \in \alpha_i$ are restricted to a single non-terminal symbol

$A \longrightarrow abc$

context - free lang

push down automata

Type 3 : regular grammar

has a single non-terminal in the left hand side

Right — single terminal possibly proceeded by a single non-terminal

right linear grammar

$A \longrightarrow \alpha B$

$A \longrightarrow \alpha$ and

left linear grammar

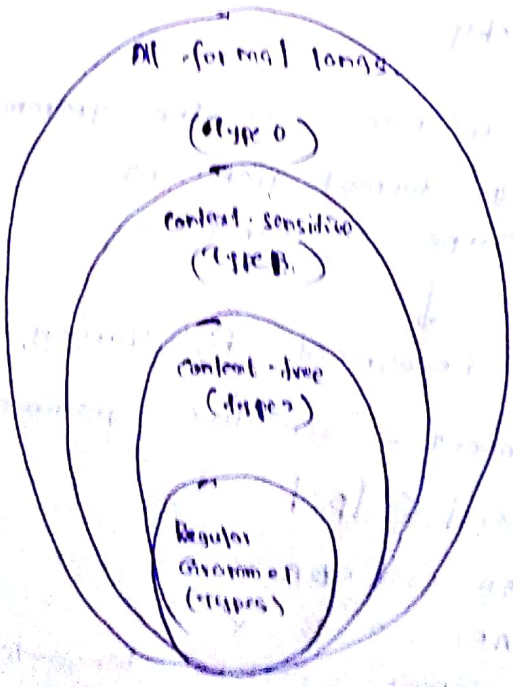
$A \longrightarrow B\alpha$

$A \longrightarrow \alpha$

recognised by finite Automata

Type 1 : is also type 0 - 1

for all $\alpha > 0$



$G_1 = (N, \Sigma, P, S)$

<while_stmt> ⟶ while(<logic_expr>) <stmt>

LHS — RHS

Left most derivation

- expand the left most non-terminal first

Right most derivation

- expand the rightmost non-terminal first

$S \to ABC$
$A \to aA$
$A \to \varepsilon$
$B \to bB$
$B \to \varepsilon$
$C \to cC$
$C \to \varepsilon$

$S \Rightarrow ABC \Rightarrow AbBC \Rightarrow AbC \Rightarrow aAbc \Rightarrow abC \Rightarrow abcC \Rightarrow abc$

Left Most

$S \Rightarrow ABC \Rightarrow aABC \Rightarrow aBC \Rightarrow abBC \Rightarrow abC \Rightarrow abcC \Rightarrow abc$

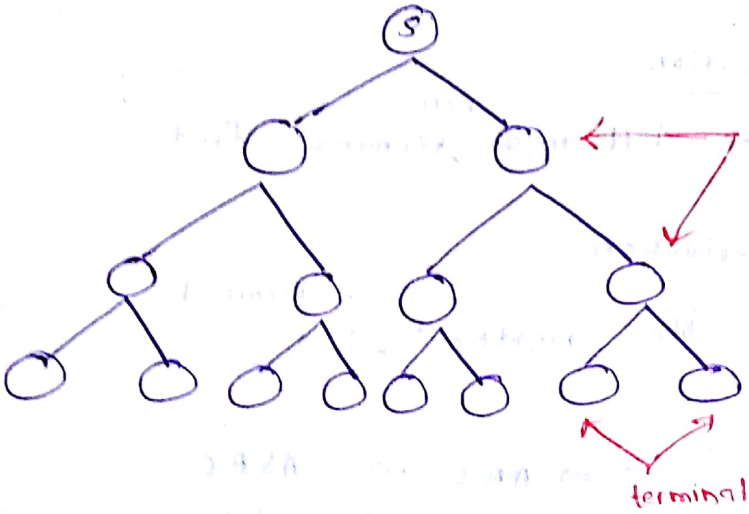
Right Most

$S \to ABC \Rightarrow ABcC \Rightarrow ABc \Rightarrow AbBc \Rightarrow Abc \Rightarrow aAbc \Rightarrow abc$

Parse Tree

- ordered / rooted tree

$G = (N, \Sigma, P, S)$

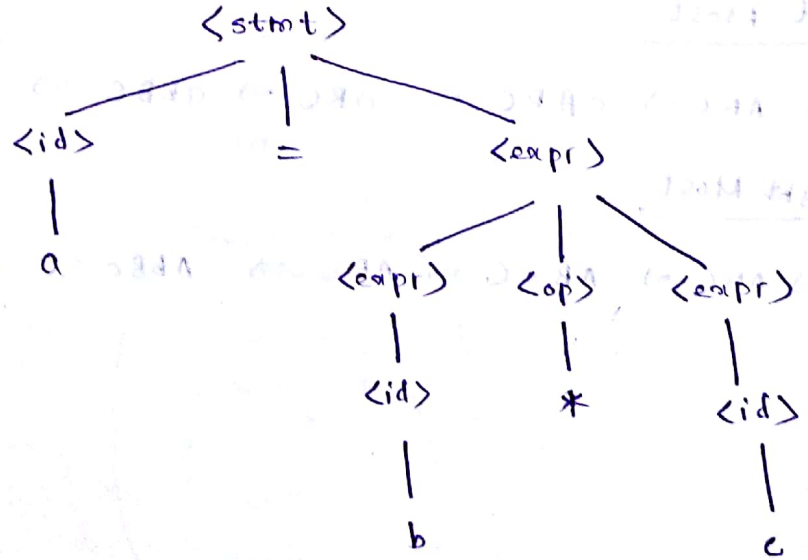


<stmt> ⟶ <id> = <expr>

<expr> ⟶ <expr> <op> <expr> | <id>

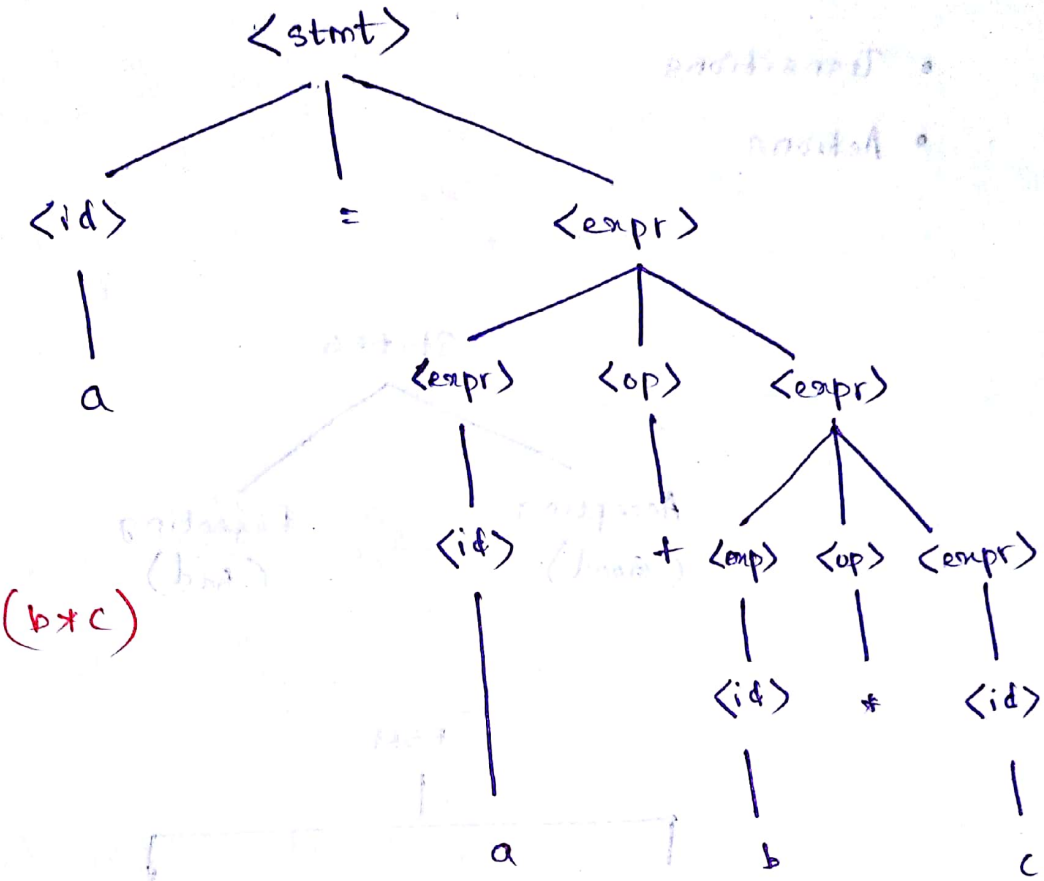
<op> ⟶ + | *

<id> ⟶ a | b | c

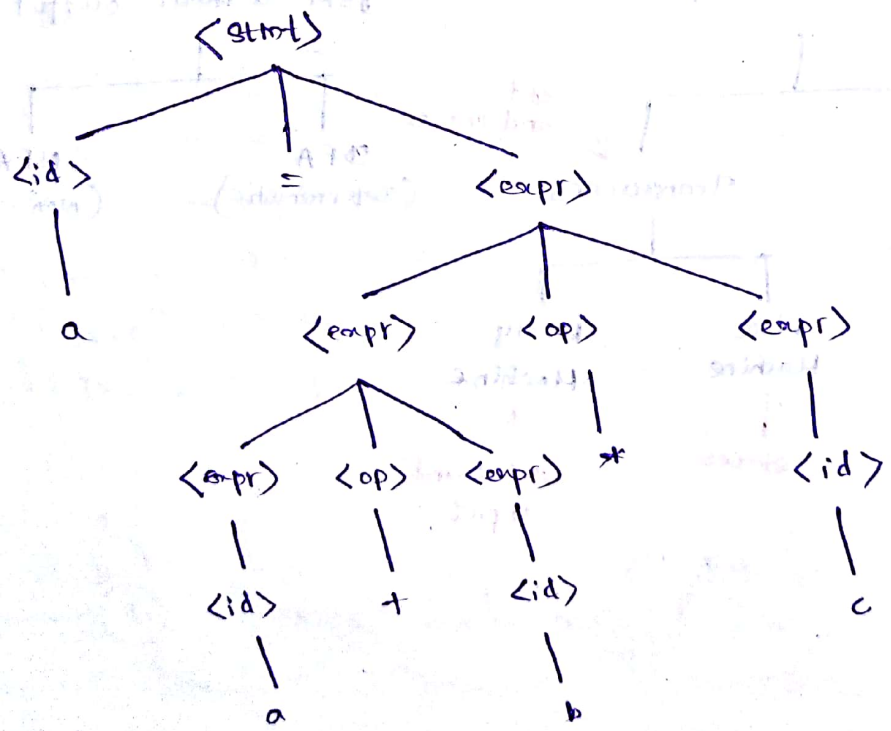


a = b * c

a = a + b * c



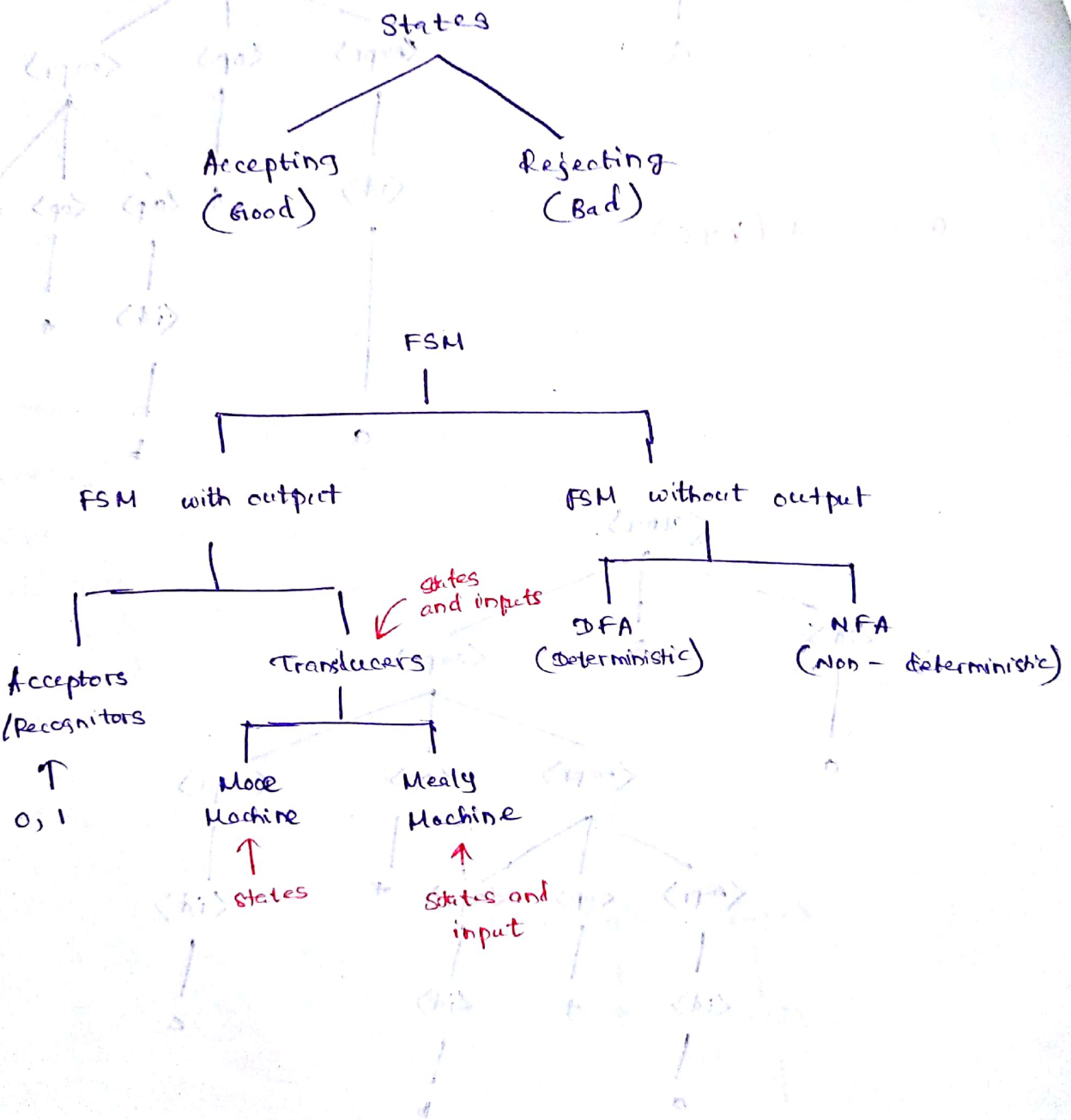
a = a + (b * c)



a = (a+b) * c

Finite State Machine

- States
- Transitions
- Actions

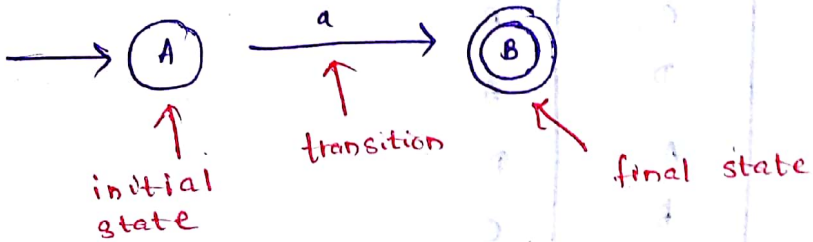


DFA

+ unique computation.

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

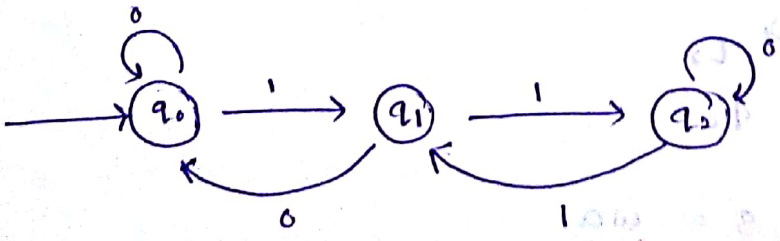
- Q : states
- Σ : input alphabet
- δ : transition action
- q_0 : initial state
- F : set of final states

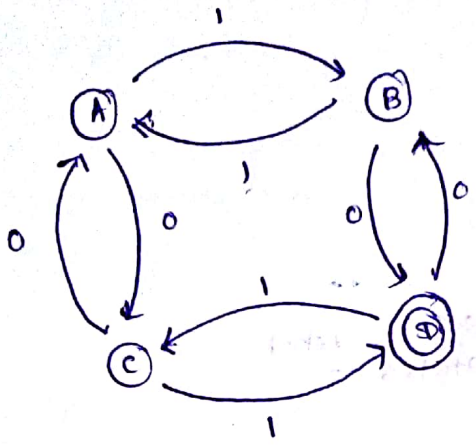


$$M = (\{q_0, q_1, q_2\}, \{0,1\}, \delta, q_0, \{q_1\})$$

δ is given by

$\delta(q_0, 0) \rightarrow q_0$
$\delta(q_0, 1) \rightarrow q_1$
$\delta(q_1, 0) \rightarrow q_0$
$\delta(q_1, 1) \rightarrow q_2$
$\delta(q_2, 0) \rightarrow q_0$
$\delta(q_2, 1) \rightarrow q_1$





$Q = \{A, B, C, D\}$

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

$q_0 = A$

$F = \{D\}$

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

outgoing edges from a vertex = # symbols in the alphabet

Extended transition Function

$$\delta^* : Q \times \Sigma^* \longrightarrow Q$$

Σ^* – string

$\delta(q_0, a) = q_1$

$\delta(q_1, b) = q_2$

ab

$\delta^*(q_0, ab) = q_2$

$\delta^*(q_0, \epsilon) = q_0$

$s = wa$

string consist all but the final letter

final letter of the string.

$$\delta^*(q, wa) = \delta(\delta^*(q, w), a)$$

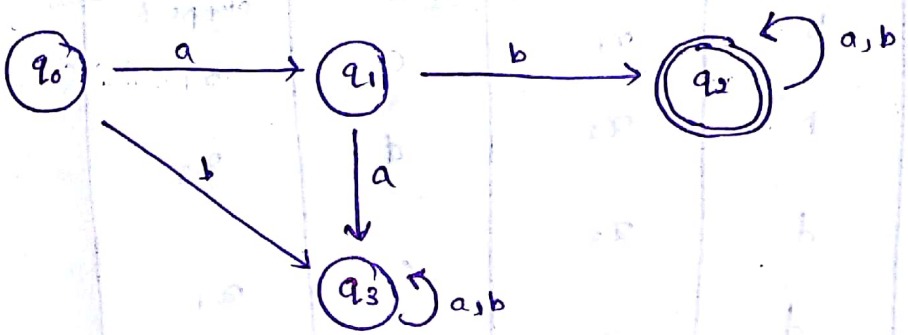
$$\delta^*(q_0, ab) = \delta(\delta^*(q_0, a), b)$$

$$\delta^*(q_0, a) = \delta(\delta^*(q_0, \varepsilon), a)$$

Find a DFA that recognises the set of all strings on $\Sigma = \{a, b\}$ starting with prefix ab.

$L = \{ab, abb, aba, abab, \ldots\}$

* we get the smallest word and draw the transition.

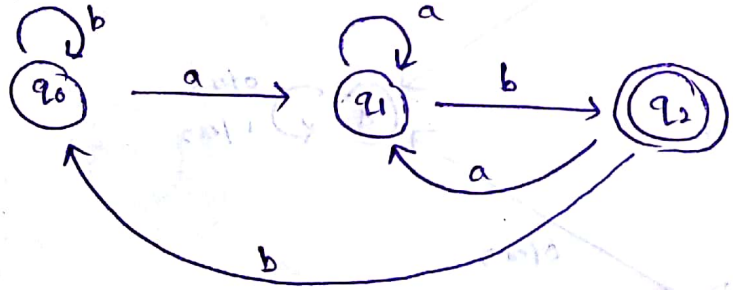


* input edges and output edges for a state should be same

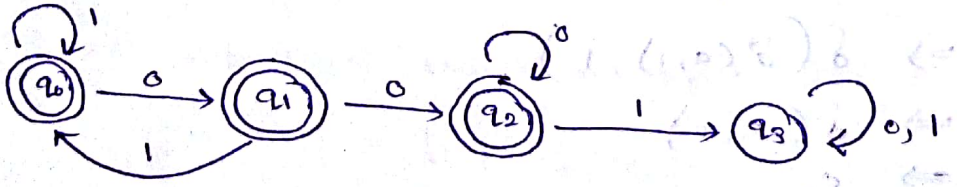
ab as suffix

$L = \{ab, aab, bab, abab, \ldots\}$

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



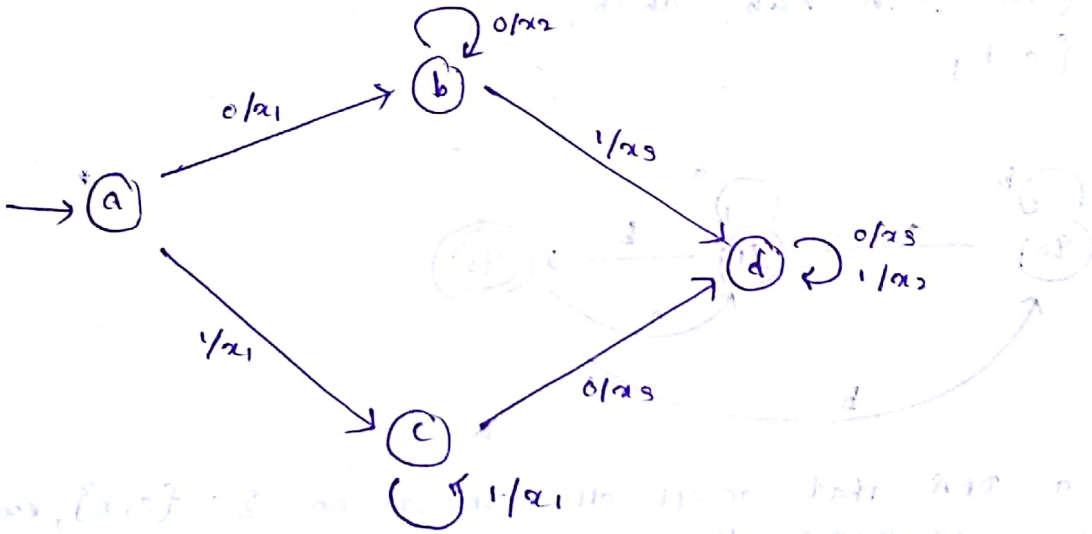
(Q2) Find a DFA that accept all strings on $\Sigma = \{0, 1\}$, except those containing the substring 001.



Transducers

Mealy Machine

Present State	Next State			
	input = 0		input = 1	
	state	output	state	output
→ a	b	x_1	c	x_1
b	b	x_2	d	x_3
c	d	x_3	c	x_1
d	d	x_3	d	x_2



Input : 11

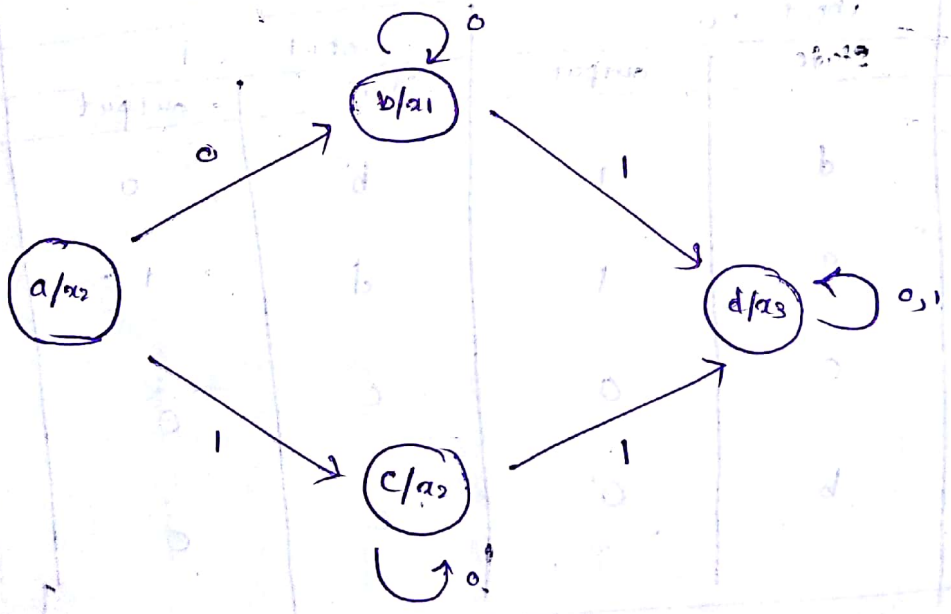
$$\delta^*(a, 11) \Rightarrow \delta(\delta^*(a, 1), 1)$$
$$\Rightarrow \delta(c, 1)$$
$$\Rightarrow c$$

output = $x_1 x_1$

|output| = |input|

Moore Machine

Present state	Next state		Output
	input = 0	input = 1	
→ a	b	c	x_2
b	b	d	x_1
c	c	d	x_2
d	d	d	x_3



length of output = length of input + 1

$$|output| = |input| + 1$$

Input : 11

$$\delta^*(a, 11) = \delta(\delta^*(a,1), 1)$$
$$= \delta(c, 1)$$
$$= d$$

output = $x_2 x_2 x_3$

Convert Moore Machine to Mealy Machine

Moore ⟶ Mealy

Present State	Next State		output
	0	1	
→ a	d	b	1
b	a	d	0
c	c	c	0
d	b	c	1

Present State	Next State			
	input = 0		input = 1	
	State	output	State	output
→ a	d	1	b	0
b	a	1	d	1
c	c	0	c	0
d	b	0	c	0

Mealy ⟶ Moore

Present state	Next State			
	input = 0		input = 1	
	state	output	state	output
→ a	d	0	b	1
b	a	1	d	0
c	c	1	c	0
d	b	0	c	1

⇓

state	# of outputs
a	1 (1)
b	2 (0,1) ← $b_0 = 0$, $b_1 = 1$
c	2 (0,1) ← $c_0 = 0$, $c_1 = 1$
d	1 (0)

present state	Next state		output
	0	1	
a	d	b_1	1
b_0	a	d	0
b_1	a	d	1
c_0	c_1	c_0	0
c_1	c_1	c_1	1
d	b_0	c_1	0

? there's a mistake.

present	Next			output
	state	output	state	output
a	d	0	b_1	1
b_0	a	1	d	0
b_1	a	1	d	0
c_0	c_1	1	c_0	0
c_1	c_1	1	c_1	1
d	b_0	0	c_1	1

Non-Deterministic Finite Automata (NFA)

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

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

$q_0 = a$

$F = \{c\}$

$\delta(a, 0) = \{a, b\}$

$\delta(a, 1) = \{b\}$

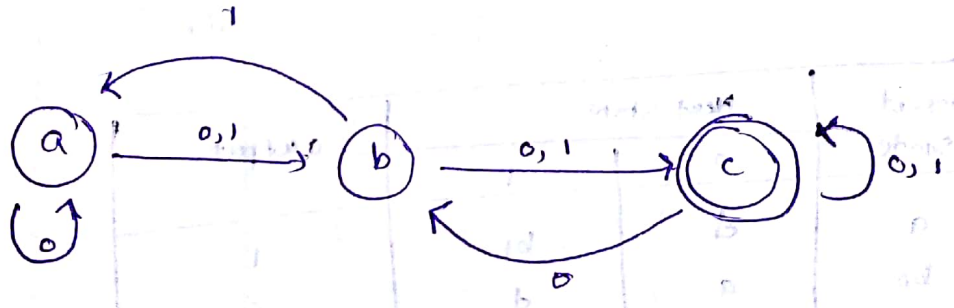
$\delta(b, 0) = \{c\}$

$\delta(b, 1) = \{a, c\}$

$\delta(c, 0) = \{b, c\}$

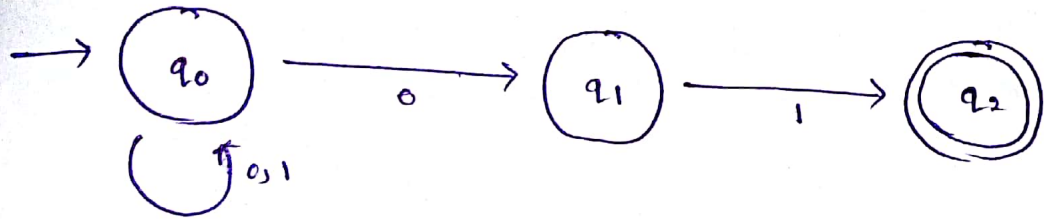
$\delta(c, 1) = \{c\}$

Current State	Next State	
	0	1
a	{a, b}	{b}
b	{c}	{a, c}
c	{b, c}	{c}



* Every DFA is a NFA
* but not every NFA is a DFA

Example



Is the string 00101 is accepted by this NFA?

00101

$\rightarrow q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0$ ✗

$\rightarrow q_0 \xrightarrow{0} q_1$ ✗

$\rightarrow q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_2$ ✓

$\delta^*(q, \varepsilon) = \{q\}$

$w = x\,a_1$

x: rest, a_1: final letter of w

$x_1, x_2, x_3, \ldots x_k$

$\delta^*(q, w)$

$\delta^*(q, x) = \{p_1, p_2, \ldots p_k\}$

$\bigcup_{i=0}^{k} \delta(p_i, a) = \{x_1, x_2, \ldots, x_m\}$

$\delta^*(q, w) = \{x_1, x_2, \ldots x_m\}$

$\delta^*(q_2, xa) = \delta(\delta^*(q, x), a)$

$= \delta(\{p_1, p_2, \ldots, p_k\}, a)$

$= \delta(p_1, a) \cup \delta(p_2, a) \cup \ldots \cup \delta(p_k, a)$

$= \bigcup_{i=1}^{k} \delta(p_i, a)$

$= \{x_1, x_2, \ldots, x_m\}$.

DFA = Have a unique path for a given input (direct path)

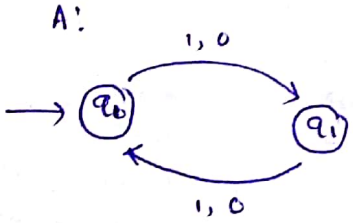
NFA = Can have several paths, for a given input.

- have transaction for ε (Null, Empty) inputs
- can have several states we can go. Even one path is going to final state, so it will an accepted otherwise rejected.

* DFA and NFA equivalent if they accept the same language

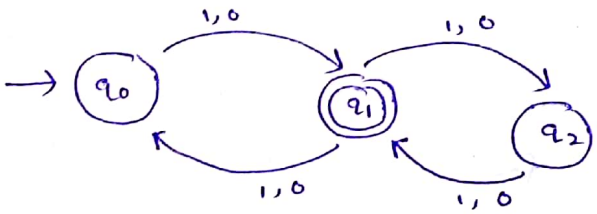
DFA

A:



NFA

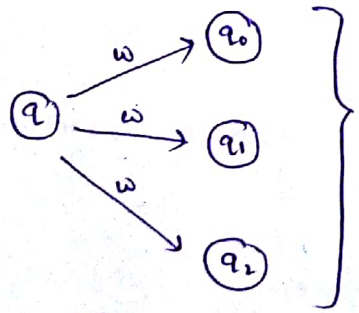
B:



These two accept the same language.
A and B are equivalent automata

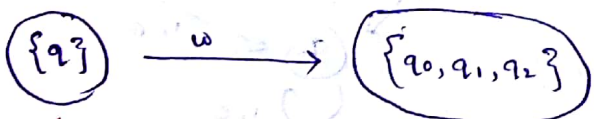
Convert NFA to DFA

Case 1:



NFA

combine this 3 and convert to single state in DFA



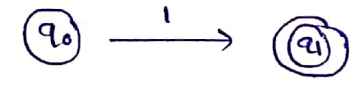
DFA

can be represented in a transition function.

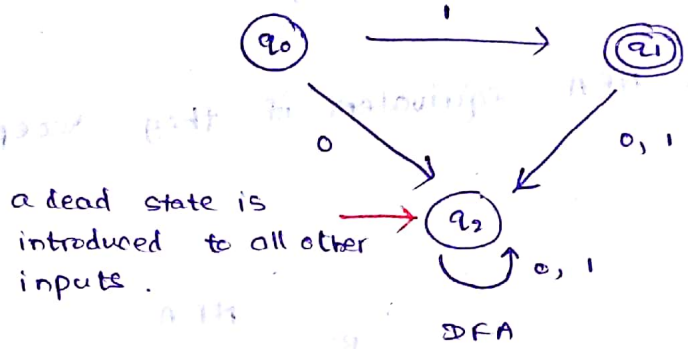
$$\delta(q, w) = \{q_0, q_1, q_2\}$$

case 2:

* Define transition for only one input
* DFA = Must have same as number of output as size as alphabet.



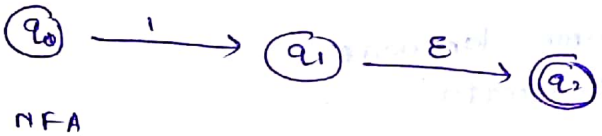
NFA



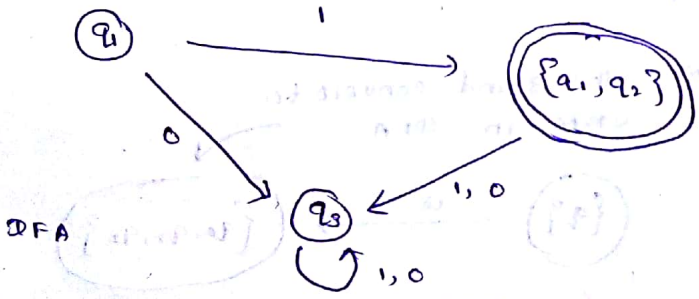
DFA

case 3:

* Transition for ε

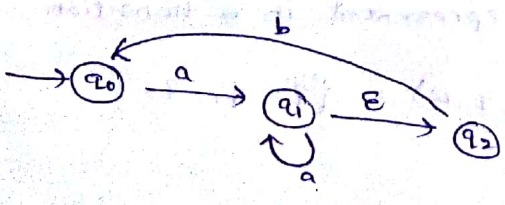


NFA

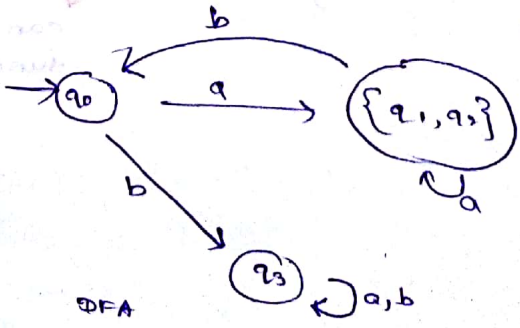


DFA

Example



NFA



DFA

Convertion of NFA into DFA

NFA = {A, B, C}
↑
States

possible state of DFA = {A}, {B}, {C}, {A,B}, {B,C}, {A,C}, {A,B,C}, {ϕ}

These are possible states. No need to have them all.

States = 2^n

DFA transition Function define

Single input for DFA state

Can have many NFA states

$R = \{r_1, r_2, r_3, \ldots\}$

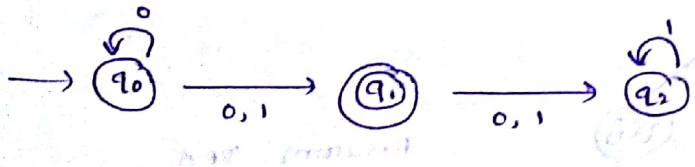
R ↑ State of DFA

$r_1, r_2, r_3, \ldots$ ↑ NFA state

$$= \Sigma(\delta_n(r_1, \sigma)) \cup \Sigma(\delta_n(r_2, \sigma))$$

↑ bcz of ε transition will happen for NFA

Example step 1



$q_0 = \varepsilon(q_0) = q_0$

↑ ε transition closure mean number of transitions can do with ε closure with state itself.

→ q_0

* lets find transition for inputs.

$$\delta(q_0, 0) = \bigcup_{r \in R} \varepsilon(\delta_n(q_0, 0))$$

$$= \varepsilon(q_0, q_1)$$

$$= \varepsilon(q_0) \cup \varepsilon(q_1)$$

$$= \{q_0, q_1\}$$

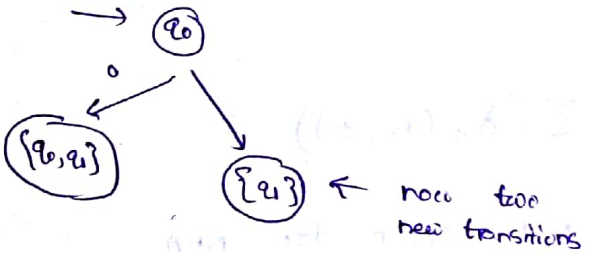
→ q_0 —0→ $\{q_0, q_1\}$

Now for input 1 at q_0

$$\delta(q_0, 1) = \bigcup_{r \in R} \varepsilon(\delta_n(q_0, 1))$$

$$= \varepsilon(q_1)$$

$$= \{q_1\}$$



← now two new transitions

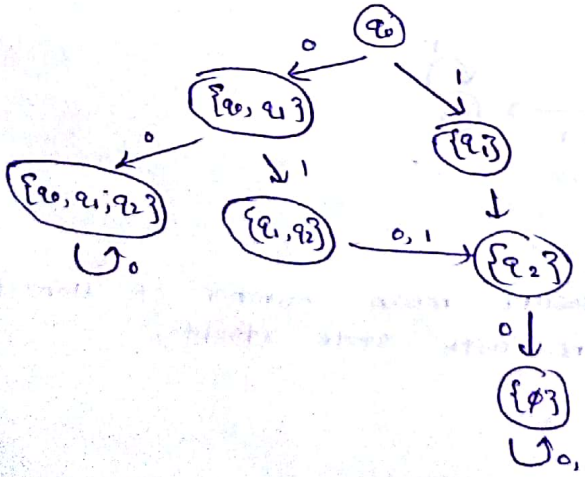
$$\delta(\{q_0, q_1\}, 0) = \bigcup_{r \in R} E(\delta_n(\{q_0, q_1\}, 0))$$

$$= \varepsilon(\delta_n(q_0, 0)) \cup \varepsilon(\delta_n(q_1, 0))$$

$$= \varepsilon(q_0, q_1) \cup \varepsilon(q_2)$$

$$= \{q_0, q_1, q_2\}$$

Do same for input 1;



Resulting DFA

$$\varepsilon(\{q_0,q_1,q_2\},0) = \bigcup_{i \in P} \varepsilon(\delta_N(i,0))$$

$$= \varepsilon(\delta_N(q_0,0)) \cup \varepsilon(\delta_N(q_1,0)) \cup \varepsilon(\delta_N(q_2,0))$$

$$= \varepsilon(q_0,q_1) \cup \varepsilon(q_1) \cup \varepsilon(\phi) \quad \text{the null}$$

$$= \{q_0,q_1,q_2\}$$

Another way

Transition table for NFA

State	0	1
q_0	$\{q_0,q_1\}$	q_1
q_1	q_2	q_2
q_2	—	q_2

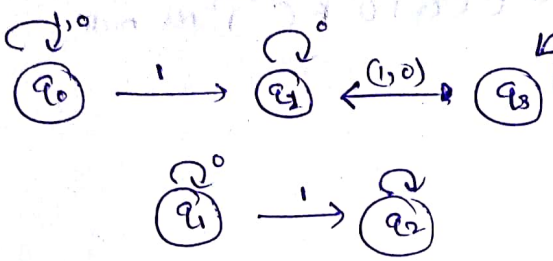
DFA

State with ε put ε[q0]	State	0	1
→	q_0	$\{q_0,q_1\}$	$\{q_1\}$
	$\{q_0,q_1\}$	$\{q_0,q_1,q_2\}$	$\{q_1,q_2\}$
	$\{q_1\}$	q_2	q_2
	$\{q_0,q_1,q_2\}$	$\{q_0,q_1,q_2\}$	$\{q_1,q_2\}$
	$\{q_1,q_2\}$	$\{q_2\}$	$\{q_1\}$
	$\{q_2\}$	ϕ	ϕ

First go see what is the state when 0, Then q_1 and see what is state when 0.

Reducing number of state in finite automata

* remove states which can not be reached from



* Merge equivalent states
  - think all state are equivalent. then prove which are different.

* indistinguishable (equivalent) state
  - The states that give same output

$$\forall \omega,\ \delta^*(p,\omega) \in F \iff \delta^*(q,\omega) \in F$$

(p: start from p state; q: comes to Final state)

& above is true when

$\delta^*(p,\omega) \in F$ and $\delta^*(q,\omega) \in F$

OR

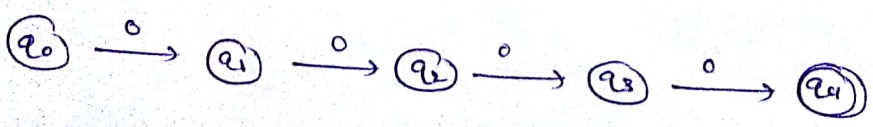
$\delta^*(p,\omega) \notin F$ and $\delta^*(q,\omega) \notin F$

* Must both belong to final state or must not both belong to final state.

* if p and q are not equivalent due to coord ω, so ω seperation word.

K- equivalence

* Mean there is no word $|\omega| \in K$ which is a seperation word. so $P \equiv q$

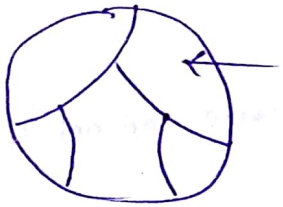


$q_0, q_1 = 2$ - equivalence

$q_0, q_1 \neq 3$ - equivalence

Equivalence relation

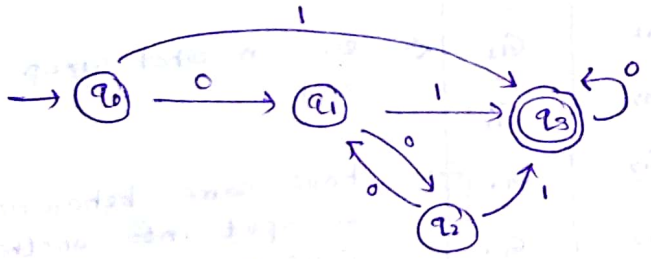
* seperate set of state into disjoint sets.



equivalence class

Equivalence class = Equivalent state

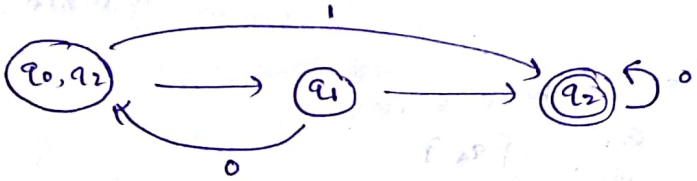
* Cannot have two equivalence in seperate class



$(q_0, 0) = q_1$ $\quad$ $(q_2, 0) = q_1$

$(q_0, 1) = q_3$ $\quad$ $(q_2, 1) = q_3$

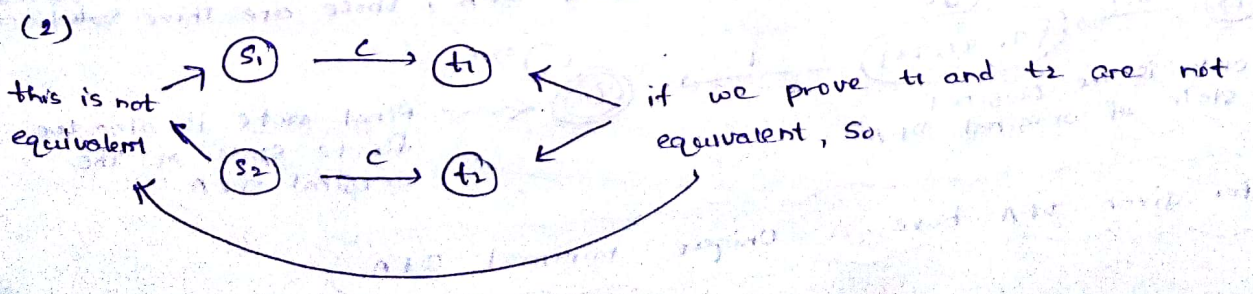
q_0 and q_1 are equivalent state



The rules that we use

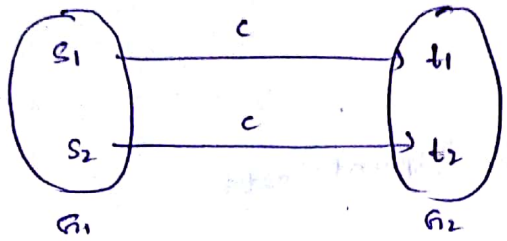
(1) distinguishable

(2)



DFA Minimization Algorithm

Group is consistance if,



Example

Initial groups $G_1 = \{q_4\}$ ← Here one state can not split further

↑ Final state

$G_2 = \{q_0, q_1, q_2, q_3\}$

↑ Non final state

Now take G_2 and see whether consistance

	0	1
q_0	G_2	G_2
q_1	G_2	G_1
q_2	G_2	G_1
q_3	G_2	G_1

← see in what group they will end

← have same behaviours so split into another group category

$G_3 = \{q_0\}$

↑ This can not split so, its ok

$G_4 = \{q_1, q_2, q_3\}$

↑ Get q_4 and create a table.

G_4	0	1
q_1	G_4	G_1
q_2	G_4	G_1
q_3	G_1	G_1

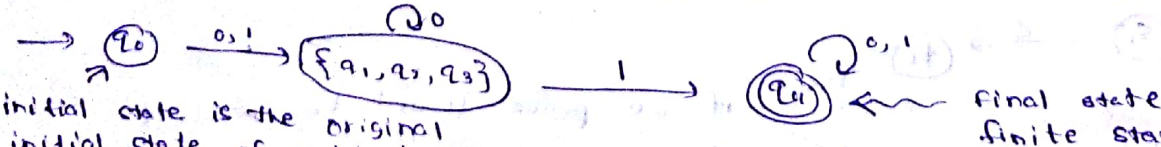
So we have 3 classes

$G_1 = \{q_4\}$

$G_3 = \{q_0\}$

$G_4 = \{q_1, q_2, q_3\}$

* in our minimized DFA, those are three states.

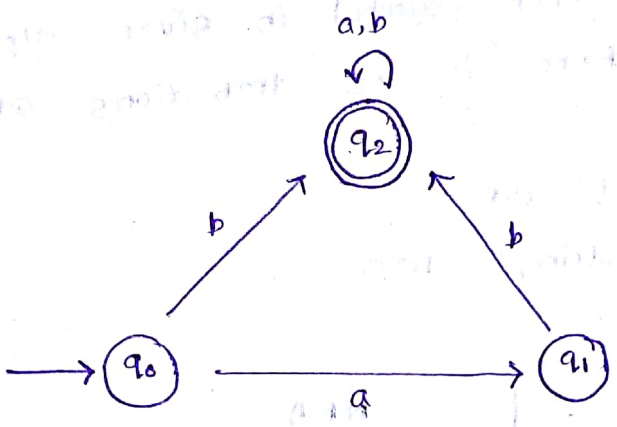


initial state is the original initial state of original DFA

Final state is also the finite state of the original DFA

* For given DFA have a unique minimal DFA

Examples



a) Not a DFA

There is no transition for input a at state q_1

b)

State	a	b
q_0	q_1	q_2
q_1	—	q_2
q_2	q_2	q_2

(02) a) $= \{Q, q_0, F, \delta, \Sigma\}$

Q - states

Σ - alphabet (Finite set of symbols)

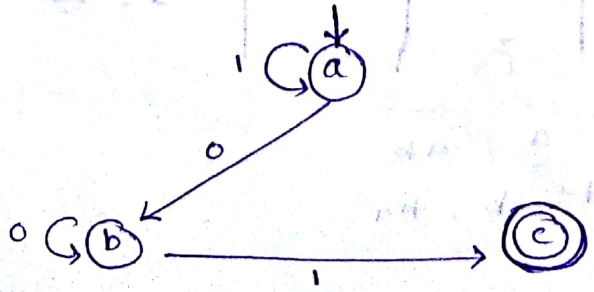
q_0 - initial state

Q - Final states

δ - transition Function

$$\delta : Q \times \Sigma \rightarrow Q$$

(b)



(c) It is not a DFA. In a DFA every states must have a transition for each symbol in given alphabet. But here for input 1, 0 there is no transitions at state C.

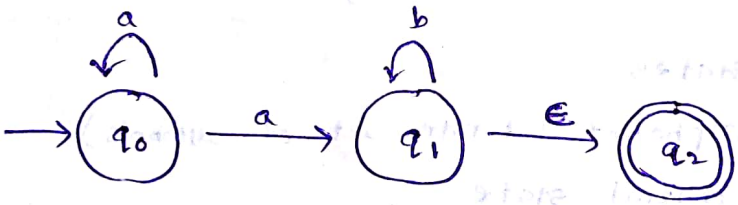
(d) Accepting String : 01

Not accepting string : 100

(03) (a)

DFA	NFA
* All transitions are defined	* There can be transitions that are not defined
* For one input, there is only one state	* There can be several states for one input
* Do not accept empty string transitions (ϵ)	* accept ϵ string transitions.
* Accept string if the last state is a final state	* Accept strings if ~~the~~ one of the last state is a final state

(b)

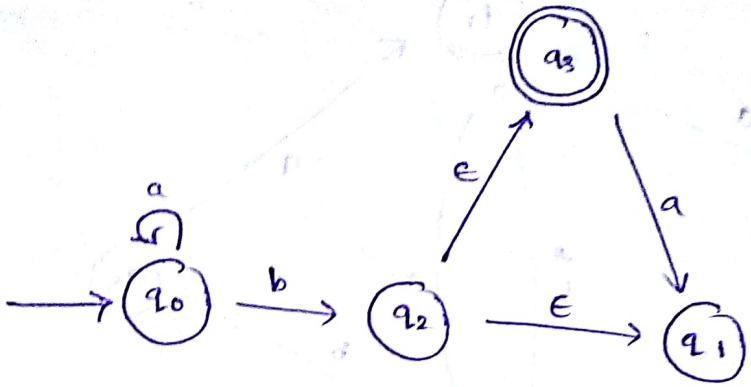


(i)

State	a	b	ϵ
q_0	q_0, q_1	–	–
q_1	–	q_1	q_2,
q_2	–	–	–

(ii) accepted : a , ab

not accepted : b , ba



(a) NFA.

have empty strings.
all transitions are not defined

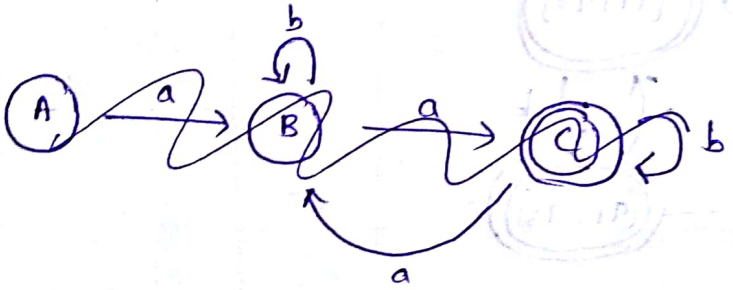
(b)

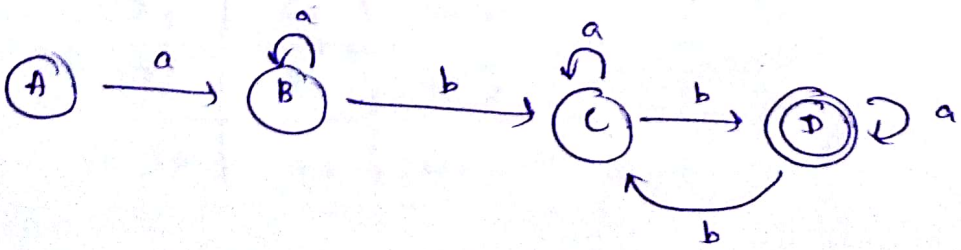
State	a	b	ϵ
q_0	q_0	q_2	—
q_2	—	—	q_3, q_1
q_1	—	—	—
q_3	q_1	—	—

(c) accepted: b, ab

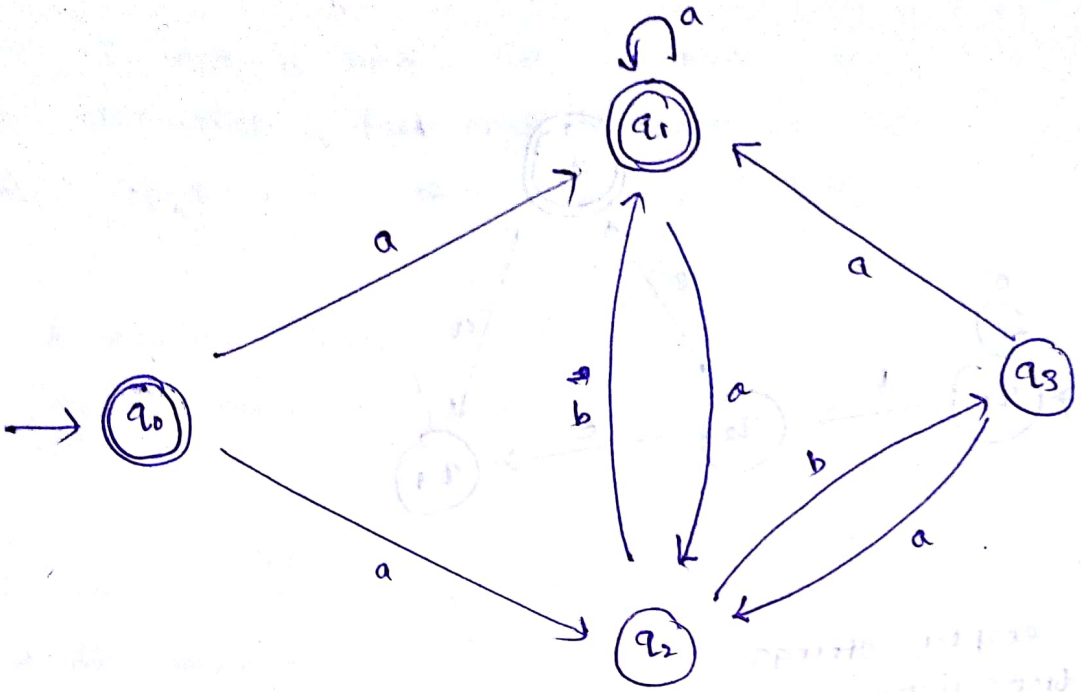
not accepted: aba

(05)





(6b)

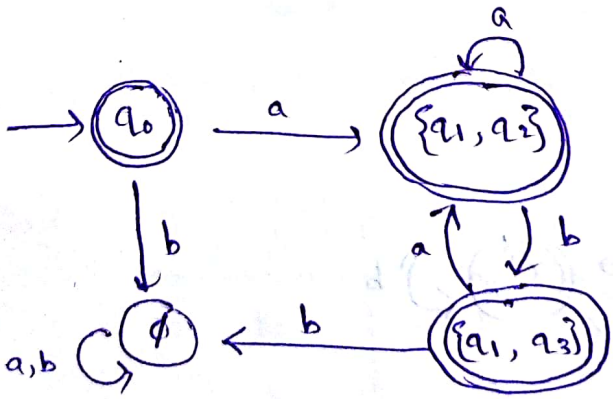


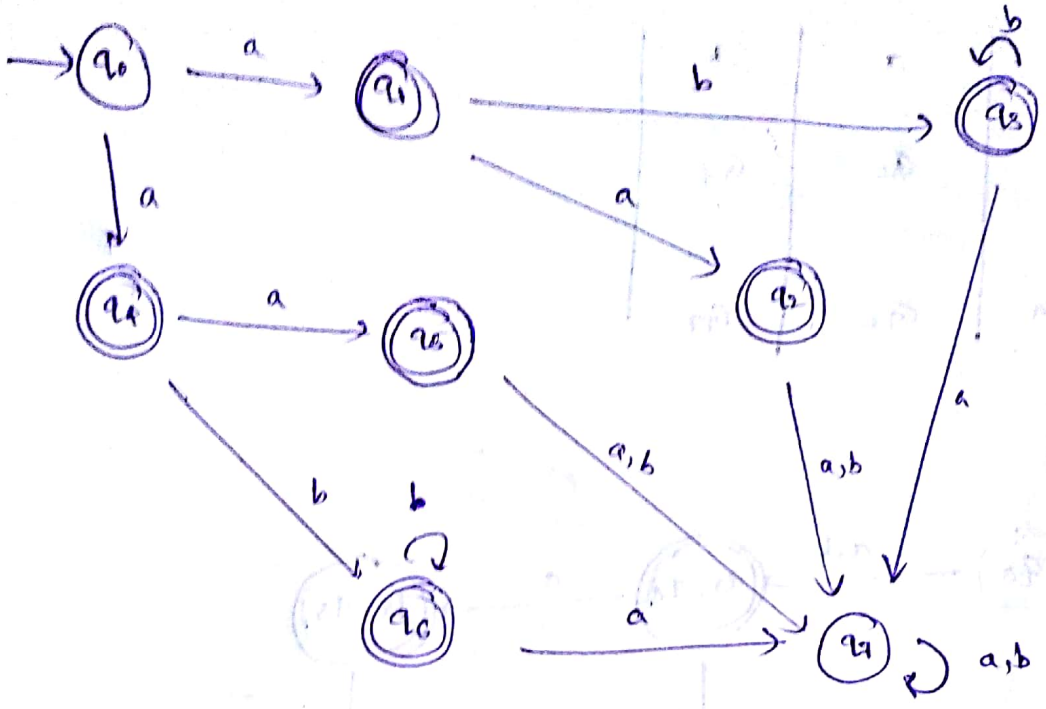
NFA

State	a	b
q_0	$\{q_1, q_2\}$	—
q_1	$\{q_1, q_2\}$	—
q_2	—	$\{q_1, q_3\}$
q_3	$\{q_1, q_2\}$	—

DFA

State	a	b
q_0	$\{q_1, q_2\}$	ϕ
$\{q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_1, q_3\}$
$\{q_1, q_3\}$	$\{q_1, q_2\}$	ϕ





G_1 : $\{q_1, q_2, q_3, q_4, q_5, q_6\}$ ⇐ final states

G_2 : $\{q_0, q_7\}$ ⇐ non final states.

G_2	a	b
q_0	G_1	G_1
q_7	G_2	G_2

$G_3 = \{q_6\}$

$G_4 = \{q_7\}$

G_1	a	b
q_1	G_1	G_1
q_2	G_4	G_4
q_3	G_4	G_1
q_4	G_1	G_1
q_5	G_4	G_4
q_6	G_4	G_1

$G_5 = \{q_1, q_4\}$

$G_6 = \{q_2, q_5\}$

$G_7 = \{q_3, q_6\}$

G_5	a	b
q_1	G_6	G_7
q_4	G_6	G_7