TOC by
book => KLP Mishra

TOC

Automata theory - define the various mathematical models of computation and their properties

Computability theory - problem is computable or not

Complexity theory - Avoiding problems based on their hardness

Automata problem is computed without direct human contact
(automatic

(i) **Finite State Automata** - Analytically, a finite state automata or FSA may be defined as a 5 tuple $(Q, \Sigma, \delta, q_0, F)$ where

a) Q is a finite non empty set of states

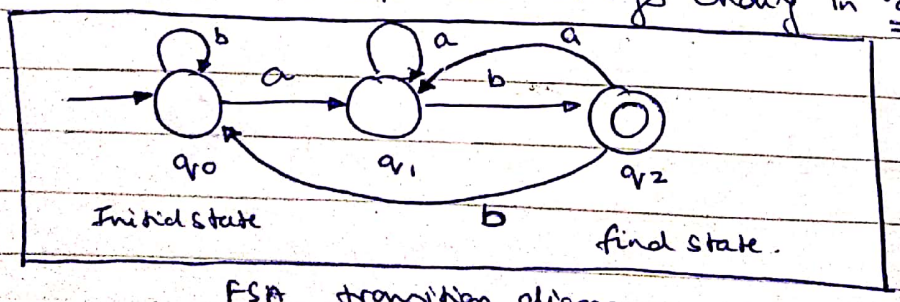
b) Σ is a finite non empty set symbols

c) δ is the state transition function $\delta : Q \times \Sigma \to Q$
(direct transition function)
$\delta(q, a) = q'$
Can be represented by a transition diagram or transition table

d) q_0 is initial state, $q_0 \in Q$

e) $F \subseteq Q$ is the set of final states.

Example => Q, $\Sigma = \{a, b\}$
The FSA has to accept all strings ending in 'ab'.



FSA transition diagram

→ ○ Initial state
(○) final state

Transition table

	δ	a	b
initial state →	q_0	q_1	q_0
	q_1	q_1	q_2
final state	(q_2)	q_1	q_0

Acceptibility of $\delta(q_0, abaaab)$

$= \delta(q_1, baaab) = \delta(q_2, aaab)$

$= \delta(q_1, aab) = \delta(q_1, ab) = \delta(q_1, b)$

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

empty string

q_2 is the final state.

∴ abaaab is accepted by FSA.

Indirect transition function $\delta^* : Q \times \Sigma^* \to Q$

Σ^* → all the possible string set over Σ

(2) Transition System — all other things are same, only transition function is changed to indirect transition function. $Q_0 \subseteq Q$

This was deterministic finite automata.

(3) Non deterministic finite automa → $Q \times \Sigma \to 2^Q$

2^Q = power set of Q

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

Transition system => $(Q, \Sigma, \delta, Q_0, F)$

~~Qs~~ $\delta : Q \times \Sigma^* \to Q$ $\quad Q_0 \subseteq Q$

Properties of transition function

(i) $\delta(q, \Lambda) = q \quad \forall q \in Q$ $\quad \Lambda$ = empty string

(ii) $\delta(q, xa) = \delta(\delta(q, x), a)$

$\delta(q, ax) = \delta(\delta(q, a), x)$ }

$x, y, z \in \Sigma^*$ (strings)

$a, b, c \in \Sigma$ (symbols)

Results : For any transition δ and for any 2 input strings x and y:

$\delta(q, xy) = \delta(\delta(q, x), y)$

Proof By PMI on $|y|$ ↳ length of string y.

Base Step $|y| = 1$ follows from property 2

Assume that result holds for all strings of length $\leq n$.

Let $|y| = (n+1)$ $\quad$ $|y_1| = n$ $\quad$ $y = y_1 a$ $\quad$ $, a \in \Sigma$

$$LHS = \delta(q, xy) = \delta(q, xy_1 a)$$
$$= \delta(\delta(q, xy_1), a) \quad \text{prop 2}$$
$$= \delta(\delta(\delta(q, x), y_1), a)$$
(using hypothesis)

$$RHS = \delta(\delta(q, x), y)$$
$$= \delta(\delta(q, x), y_1 a) \quad y = y_1 a$$
$$= \delta(\delta(\delta(q, x), y_1), a) \quad (\text{prop 2})$$

$LHS = RHS$ $\Rightarrow$ Hence proved

Acceptability of a string by a FSA

A string x is said to be accepted by a finite state automata n

$(x \in \Sigma^*)$ $\quad$ $M = (Q, \Sigma, \delta, q_0, F)$

if $\delta(q_0, x) = q$ $\quad$ for some $q \in F$

NDFA ⟶ DFA

State / Σ	0	1
→ (q0)	q0	q1
q1	q1	q0, q1

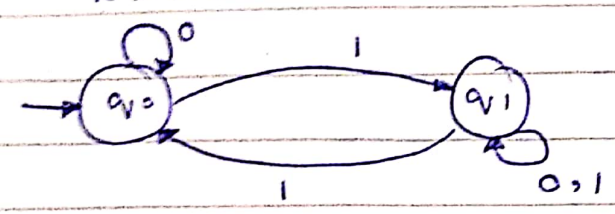
$\delta(q_1, 1) = \{q_0, q_1\}$

$\Rightarrow$ it is non deterministic

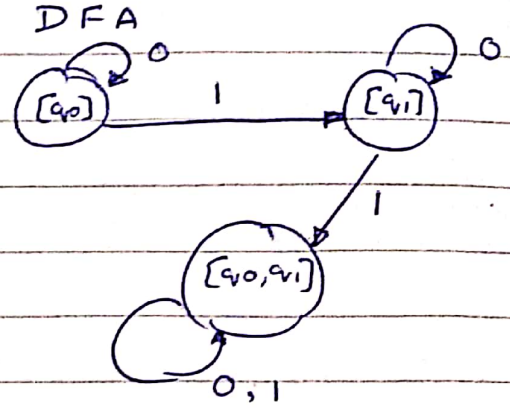
instead of multiple elements take a single set $[q_0]$ this will be the new initial state

State / Σ	0	1
→ [q0]	[q0]	[q1]
[q1]	[q1]	[q1, q0]
[q0, q1]	[q0, q1]	[q0, q1]

NDFA



DFA



→ DFA { $\omega \in \Sigma^*$ is accepted by a DFA
$\delta(q_0, \omega) = q \in F$

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

→ NDFA $\delta(q_0, \omega) = \{q_1, q_2, \ldots, q_k\}$
will be accepted if atleast one state in $\{q_1, q_2, \ldots q_k\}$ is in F

Theorem : For every NDFA ∃ a DFA accepting the same set of strings.

Convert from NDFA to DFA

State / Σ	a	b
→ q0	q0, q1	q0
q1	q2	q1
q2	q3	q3
(q3)	ϕ	q2

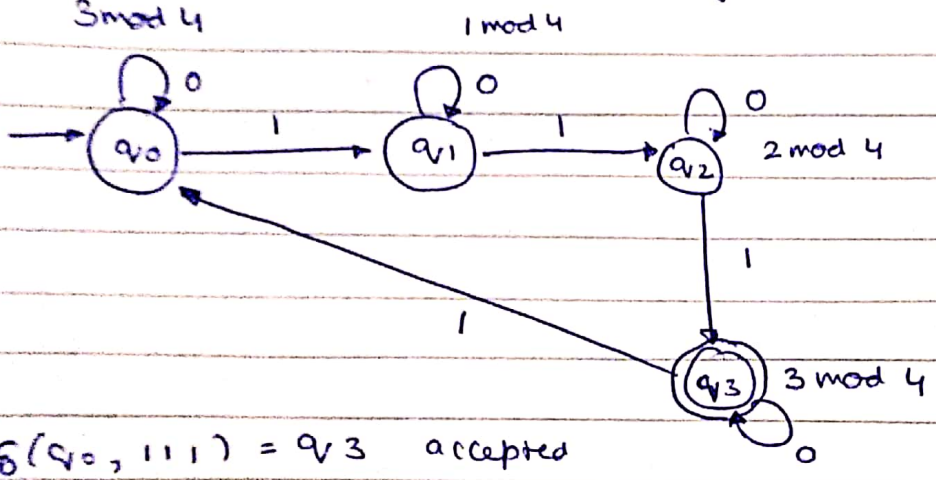
NDFA

State / Σ	a	b
→ [q0]	[q0, q1]	[q0]
[q0 q1]	[q0, q1, q2]	[q0, q1]
[q0, q1, q2]	[q0, q1, q2, q3]	[q0, q1, q3]
	DFA	

DFA

State / Σ	a	b
→ $[q_0]$	$[q_0, q_1]$	$[q_0]$
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0, q_1]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_3]$
$[q_0, q_1, q_3]$	$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$
$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_2, q_3]$

- Create a DFA that accepts strings over {0,1} when no of 1's is



$\delta(q_0, 111) = q_3$ accepted

$\delta(q_0, 11) = q_2$ not accepted

Mealy and Moore Machine we need some output here

$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ } this type of machines are particularly used for ciphers

Q = non-empty set of states

Σ = non empty set of input symbols

δ = transition function $Q \times \Sigma \to Q$

Δ = non empty set of output symbols

q_0 = initial state

λ = Output function

If λ depends only on the state and then gives an output.

$\lambda : Q \to \Delta$ } Moore Machine

Mealy machine → output depends both on the input and state.

$\lambda : Q \times \Sigma \rightarrow \Delta$

No need to learn how to create a mealy or moore machine.
only conversion between the two

Present State	Next State		Output
	a = 0	a = 1	
→ q_0	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0

λ function depends only on the state

This is a Moore Machine.

Convert to a Mealy machine.

these output can be different.

State	a = 0	Output	a = 1	Output
→ q_0	q_3	0	q_1	1
q_1	q_1	1	q_2	0
q_2	q_2	0	q_3	0
q_3	q_3	0	q_0	0

Two machines are said to be equivalent if same set of strings can be accepted by both

Moore machine

$\delta(q_0, 0111) = (q_3, 111)$

$= (q_0, 11)$

$= (q_1, 1)$

$= (q_2, \wedge)$

output

= 00010

= length of input string + 1 in a Moore machine.

Mealy machine

$\delta(q_0, 0111) = (q_3, 111)$

$= (q_0, 11) = (q_1, 1) = (q_2, \wedge)$

Output = 0010 ~ equal to the length of the input string

→ Now, given a mealy machine convert to a moore machine

Mealy machine

Present state	a = 0	Output	a = 1	Output
→ q_1	q_3	0	q_2	0
q_2	q_1	1	q_4	0
q_3	q_2	1	q_1	1
q_4	q_4	1	q_3	0

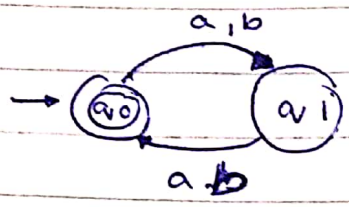
Moore machine.

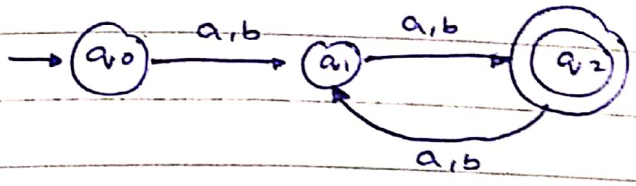
Present State	Next State a = 0	Next State a = 1	Output
→ q_1	q_3	q_2	1
q_2	q_1	q_4	1
q_3	q_2	q_1	0
q_4	q_4	q_3	0

Present	Next State a = 0	Next State a = 1	Output
q_1	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	0
q_{40}	q_{41}	q_3	0
q_{41}	q_{41}	q_3	1

TOC

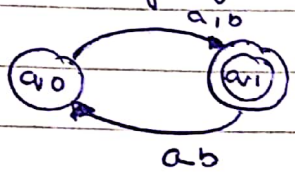
- $\Sigma = \{a, b\}$ all strings of even length over Σ

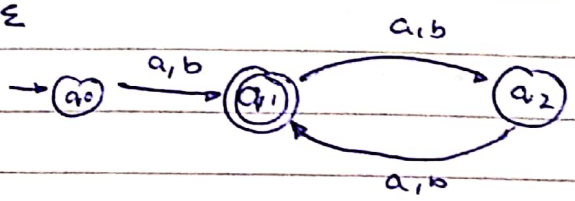




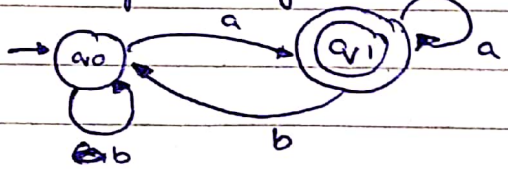
this is better, because no of states is less in this

=> all string of odd length over Σ





=> all strings ending in a



Minimization of automata

Equivalent ~~states~~ States

T(M) = Set of strings accepted by FSA M.

If M_1 and M_2 are 2 FSA's then $M_1 \equiv M_2$ if $T(M_1) = T(M_2)$ (equivalent)

2 states q_1 and q_2 are said to be equivalent ($q_1 \equiv q_2$) if $\delta(q_1, x)$ and $\delta(q_2, x)$ (transition functions) are both final or both non final states $\forall x \in \Sigma^+$ (for all strings)

k Equivalent States ($k \geq 0$)

Two states q_1 and q_2 are said to be k equivalent if $\delta(q_1, x)$ and $\delta(q_2, x)$ are both final states or both non final states for all strings of length k or less.

0 - equivalent

$\delta(q_1, \wedge) = q_1$ $\quad$ $\delta(q_2, \wedge) = q_2$

=> q_1 and q_2 will be zero equivalent if q_1 and q_2 both belong either to final state or both non final states.

If $q_1, q_2 \in F$ then q_1 and q_2 are zero equivalent.

$Q = \{q_0, q_1, q_2, \ldots q_n\}$

q_0 → initial state. $\quad F = \{q_m, q_1, q_n\}$ = final states

Q/F = Non-final states.

$\delta(q_i, \wedge) = q_i$

$\delta(q_j, \wedge) = q_j$

they will be zero equivalent if $q_i, q_j \in F$ or both $q_i, q_j \in Q/F$

F and Q/F are 0-equivalent

Reflexive $\quad q_1 \equiv q_1$

Symmetric $\quad q_1 \equiv q_2 \Rightarrow q_2 \equiv q_1$

Transitive $\quad q_1 \equiv q_2 \quad q_2 \equiv q_3 \Rightarrow q_1 \equiv q_3$

} Equivalence relation

Equivalence relation can be divided into equivalence classes.

Π_q = equivalence classes.

Minimization =>

Step 1) Construction of Π_0 =>

By definition of 0-equivalence

$\Pi_0 = \{q_1^0, q_2^0\}$ where

q_1^0 = set of final states $\quad q_2^0 = Q \mid q_1^0$

Step 2) Construction of Π_{k+1} from Π_k, let Q_{ik} be any subset of Π_k. If q_1 and q_2 are in Q_{ik}, they are (k+1) equivalent provided $\delta(q_1, a)$ and $\delta(q_2, a)$ are k equivalent. Find out whether $\delta(q_1, a)$ and $\delta_2(q_2, a)$ are

in same equivalence classes in Π_k. $\forall a \in \Sigma$. If so, q_1 and q_2 are $(k+1)$ equivalent. In this way, $Q_{i,k}$ is divided into $(k+1)$ equivalence classes. Repeat this for every $Q_{i,k}$ in Π_k to get all elements of Π_{k+1}

Step 3) Construct Π_k for $n = 1, 2, \ldots$ until $\Pi_k = \Pi_{n+1}$

Ques:

State / Σ	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

Q_1^0 Q_2^0

$\Pi_0 = \{ \{q_2\}, \{q_0, q_1, q_3, q_4, q_5, q_6, q_7\} \}$

two equivalence classes

final states, non final states.

Q_1^0 Q_2^0

if a class has single element carry on

otherwise check for 1 equivalent.

Consider strings of length 1 (as of length 0 are already considered)

$\delta(q_0, 0) = q_5 \in Q_2^0$
$\delta(q_1, 1) = q_2 \in Q_1^0$
} different equivalent classes they are not on equivalent.

$\delta(q_3, 0) = q_2$ } final state ⇒ different state
this is not equivalent to any of the present state.

$\Pi_1 = \{ \{q_2\}, \{q_0, q_4, q_6\}, \{q_1, q_7\}, \{q_3, q_5\} \}$

now check for two equivalence.

$\Pi_2 = \{ \{q_2\}, \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\} \}$

already considered length 1 ⇒ we need to check for only 1 alphabet.

At any step, classes can only be further divided, they cannot be ~~mix~~ shuffled.

~~while dividing~~

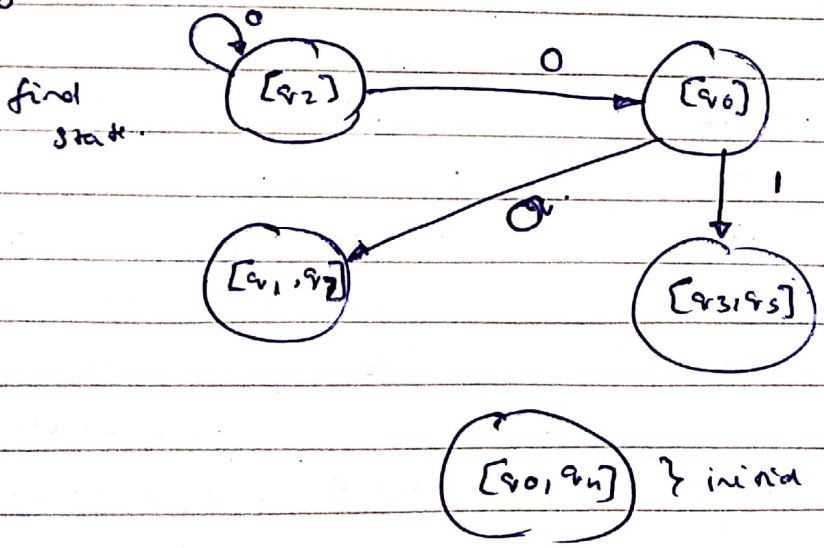
while creating Π_2, check with states Π_1 and not of Π_2

$\Pi_3 = \{\{q_2\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}, \{q_0, q_4\}\}$

$\Pi_3 = \Pi_2$ => we will stop.

Minimized automata =>

5 states instead of 8



$M = (\{\{q_2\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}, \{q_0, q_4\}\}, \{0, 1\}, \delta, [q_0, q_4], \{[q_2]\})$

ToC

U2: formal languages

Language = { } set of strings accepted by a machine (M)

Grammar

A grammar (may be) is defined by a 4-tuple (V_N, Σ, P, S)

where

i) V_N : non empty set of symbols called variables

ii) Σ : non empty set of symbols called terminals

iii) $V_N \cap \Sigma = \phi$

iv) S : starting symbol $S \in V_N$

v) P : An element in P is of the form

$\alpha \rightarrow \beta$; $\alpha, \beta \in (V_N \cup \Sigma)^*$

α must contain at least one variable

P is called set of Production Rules

$L = \{a^n b^{2n} \ (n \geq 1)\}$

$(\ , \Sigma = \{a, b\}, P, S)$

$P: \begin{cases} S \rightarrow \Lambda \\ S \rightarrow a S b^2 \end{cases}$

$S \xrightarrow{*} ab^2$

Reflexive transitive closure

Language generated by Grammar G = L(G)
language accepted by machine M = T(M)

G_1 & G_2 are equivalent if $L(G_1) = L(G_2)$

$$L(G) = \{ w \in \Sigma^* \mid S \xrightarrow{*} w \}$$

$G = (\{S\}, \{0,1\}, \{S \to 0S1, S \to 01\}, S)$

$$L(G) = \{0^n 1^n \ (n \geq 1)\}$$

$G = (\{S, A_1, A_2\}, \{a, b\}, P, S)$

$P = \{S \to aA_1A_2a, A_1 \to baA_1A_2b, A_2 \to A_1ab,$
$aA_1 \to baa, bA_2b \to abab\}$

test if $w = baabbabaaaabbaba \in L(G)$

$S \to aA_1A_2a \to baaA_2a \to baaA_1aba$
$baaA_1aba \to baababA_1A_2baba \to baabbaaA_2baba$
$baabbaaA_2baba \to baabbaaA_1ababaa \to baabbabaaabbaba = w$
$\Rightarrow w \in L(G)$

Chomsky Classification of Languages

(i) There is no restriction on set of production rules
⟹ Type 0 Grammar

(ii) $\phi A \psi \rightarrow \phi \alpha \psi$; $A \in V_N$; $\alpha \in (V_N \cup \Sigma)^+$

ϕ : left context
ψ : right context

(only one variable is replaced)

⟹ Type 1 Grammar / Context sensitive grammar

(iii) $A \rightarrow \alpha$; $A \in V_N$; $\alpha \in (V_N \cup \Sigma)^*$
⟹ Type 2 Grammar / Context free grammar

(iv) $A \rightarrow aB$; $A \rightarrow a$
⟹ Type 3 Grammar / Regular Grammar

Type 3 $\subseteq$ Type 2 $\subseteq$ Type 1 $\subseteq$ Type 0 } Language generated

★ ϵ for Type 1 and Type 3 grammar.
★ if $S \rightarrow \Lambda$ is a product^n rule
★ S must NOT be in RHS of ANY production rule

① $S \xrightarrow{T2} Aa$, $A \xrightarrow{T3} c \mid \underset{T2}{Ba}$, $B \xrightarrow{T2} abc$ } Type 2, Grammar

② $S \xrightarrow[T2]{} ASB \mid \underset{T3}{d}$, $A \xrightarrow[T3]{} aA$ } Type 2, Grammar

Can determine Grammar (4 tuple) if know exhaustive product^n rules.

Let G be a type 0 grammar, then we can find an equivalent grammar G_1 in which each production is either of the form $\alpha \to \beta$; $\alpha, \beta \in V_N^*$ or $A \to a$ $A \in V_N$, $a \in \Sigma$

G_1 is of type 1, 2 or 3 according to G's type (G_1 is of the same type as G)

if $\exists$ any terminal in the product^n rule, replace it by a variable (newly introduced) e.g. $\underset{(\Sigma)}{a} \to \underset{(V_n)}{C_a}$

$aABC \to abcC \longrightarrow C_aABC \to C_aC_bC_cC$

$C_a \to a, C_b \to b, C_c \to c$

for type i language (i) L_1, L_2

$(L_1 \cup L_2)$ } $(L_1 L_2)$ (L_1^T) — }