Dr. Vrinda Shetty

Automata Theory and computability

Module-1

Basic Notations and Terminologies used in Autom Theory:

Alphabet: A language consists of various symbols from which the words, statements etc., can be obtained. These symbols are called alphabets. Formally, an alphabet is defined as a finite non-em set of symbols. The symbol Σ denotes the set of alpha of a language.

Eg: $\Sigma = \{a, b\}$, $\Sigma = \{0, 1\}$

2.1 **String:** The sequence of symbols obtained from alphabets of a language is called a string. Formal a string is defined as a finite sequence of symbo from the alphabet Σ.

Note: An empty string is denoted by the symbol ε (ep or λ (lambda) & $\varepsilon \notin \Sigma$

Eg: Let $\Sigma = \{a, b\}$ is set of alphabets.

The various strings that can be obtained from Σ are

$\{a, b, aa, bb, aba, bba, \ldots\}$

Concatenation of Two strings: The concatenation of tw strings u and v is the string obtained by writing the letters of string u followed by the letters of st v (i.e appending the symbols of v to the right of u)

Eg: $u = a_1 a_2 a_3 \ldots a_n$ & $v = b_1 b_2 b_3 \ldots b_m$

Substring: Let w is a string obtained from the symbols in Σ. The string w if it can be decomposed into 'n' ~~xxxx~~ strings, such 'n' no of strings are called as sub strings.

Eg: $w = xyz$

then x is a substring, y is a substring & z is a substring of string w.

Prefix: A prefix is string of any number of leading symbols.

Eg In the string "Sita", the various prefixes are ε, "S", "Si", "Sit" and "Sita".

Suffix: The suffix is a string of any number of trailing symbols.

Eg In the string "Sita", the various suffixes are string ε, "a", "ta", "ita", "Sita".

Reversal of a string: The reversal of a string is obtained by writing the symbols in reverse order.

Eg $v = a_1 a_2 a_3 \cdots a_n$

The reverse of V is denoted by v^R & is given by

$$v^R = a_n a_{n-1} \cdots a_3 a_2 a_1.$$

Length of a string: The length of a string w is the number of symbols in w and is denoted by |w|.

Eg: $w = a_1 a_2 a_3 \cdots a_n$

then length of w is given by

$$|w| = n.$$

Power of an alphabet: The power of an alphabet is denoted by Σ^i is the set of words of length i.

(Eg) If $\Sigma = \{a, b\}$

Then

$\Sigma^0 = \{\varepsilon\}$ denotes empty string.

$\Sigma^1 = \{a, b\}$ " set of words of length 1

$\Sigma^2 = \{aa, ab, ba, bb\}$ " " " 2

$\Sigma^3 = \{aaa, aab, aba, abb, bba, bab, bbb, baa\}$ -- length 3

& so on.

2.2 **Language**: A language can be defined as a set of strings obtained from Σ^* where Σ is set of alphabets of a particular language.

In other words, a language is subset of Σ^* which is denoted by $L \subseteq \Sigma^*$.

(Eg) (i) $\{\varepsilon, 01, 0011, 000111, 0101 \ldots\}$ A language of strings consisting of equal number of 0's & 1's.

(ii) $\{\varepsilon, ab, aabb, aaabbb \ldots\}$ A language of strings consisting of n number of 'a's followed by 'n' number of 'b's.

Sentence: A string that belongs to a language is called word or sentence of that language.

(Eg): $\Sigma = \{0, 1\}$

$\{\varepsilon, 01, 10, 0011, 000111, \ldots\}$

Each word separated by comma is a sentence (also called word)

So 01 is a sentence

10 " "

0011 " " "

Language Hierarchy

Theorem: Concatenation & Reverse of Strings.

Theorem: If w and x are strings, then $(wx)^R = x^R w^R$

Eg: $(\text{nametag})^R = (\text{tag})^R (\text{name})^R$ = gateman

Proof: The proof is by induction on $|x|$:

Base case: $|x| = 0$

Then $x = \varepsilon$

and $(wx)^R = (w\varepsilon)^R = (w)^R = \varepsilon w^R = \varepsilon^R w^R = x^R w^R$

Prove: $\forall n \geq 0 \; ((|x| = n) \rightarrow ((wx)^R = x^R w^R)) \rightarrow ((|x| = n+1) \rightarrow ((wx)^R = x^R w^R)))$

Consider any string x, where $|x| = n+1$.

Then $x = ua$ for some character a and $|u| = n$.

So

$(wx)^R = (w(ua))^R$	rewrite x as ua
$= ((wu)a)^R$	associativity of concatenation
$= a(wu)^R$	definition of reversal
$= a(u^R w^R)$	induction hypothesis
$= (au^R)w^R$	associativity of concatenation
$= (ua)^R w^R$	definition of reversal
$= x^R w^R$	rewrite ua as x.

2.2.2 Techniques for defining Languages

We use variety of techniques for defining the language that we wish to consider. Since languages are sets, we can define them using any of the set-defining techniques.

Ex 2.3 : All a's precede All b's

Let $L = \{ w \in \{a,b\}^* :$ all a's precede all b's in $w\}$

The strings ε, a, aa, aabbb, bb are in L.

Ex 2.4 : Strings That End in a

Let $L = \{ x : \exists y \in \{a,b\}^* (x = ya)\}$

$L = \{$ a, aa, aaaa, bbaa & ba$\}$ are in L.

Ex 2.5 : The perils of Using English to describe languages

Let $L = \{x \# y : x, y \in \{0,1,2,3,4,5,6,7,8,9\}^*$ and when x and y are viewed as the decimal representations of natural numbers, Square(x) = y$\}$

The strings 3#9 and 12#144 are in L.

The strings 3#8, 12 and 12#12#12 are not in L.

Ex 2.6 : The empty language

Let $L = \{\} = \phi$, L is the language that contains no strings

Ex : Using Replication to Define a Language

Let $L = \{a^n : n \geq 0\}$

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

Languages are sets, the computational definition of language, could specify either:

* a language generator, which enumerates (lists) the elements of the language, or
* a language recognizer, which decides whether or not a candidate string is in the language and returns True if it is and False if it isn't.

Lexicographic order (written $<_L$):

In certain cases, when enumeration of language L is considered, we may care about the order in which elements of L are generated. If there exists a total order D of the elements of E_L, then we can use D to define on L a useful total order called lexicographic order (written $<_L$)

* Shorter strings precede longer ones:
  $\forall x(\forall y((|x|<|y|) \rightarrow (x <_L y)))$, and
* Of strings that are the same length, sort them in dictionary order using D.

We say that a program lexicographically enumerates the elements of L iff it enumerates them in lexicographic order

Ex Lexicographic Enumeration

Let $L = \{x \in \{a,b\}^* : \text{all a's precede all b's}\}$

The lexicographic enumeration of L is

$\varepsilon, a, b, aa, ab, bb, aaa, aab, abb, bbb, aaaa, aaab, aabb, abbb, bbbb, aaaaa \ldots$

2.2.3 What is the cardinality of language?

The smallest language over any alphabet is ϕ, whose cardinality is 0.

The largest language over any alphabet Σ is Σ^*.

$|\Sigma^*|$? Suppose $\Sigma = \phi$ then $\Sigma^* = \{\varepsilon\}$ & $|\Sigma^*| = 1$

Since any language over Σ is a subset of Σ^*, the cardinality of every language is at least 0 & at most $\aleph_0$. So all languages are either finite or countably infinite.

Theorem 2.2 : The cardinality of Σ^*

Theorem : If $\Sigma \neq \phi$ then Σ^* is countably infinite

Proof : The elements of Σ^* can be lexicographically enumerated by a straightforward procedure that:

* Enumerates all strings of length 0, then length 1, then length 2, and so forth.
* within the strings of a given length, enumerates them in dictionary order.

This enumeration is infinite since there is no longest string in Σ^*. Since there exists an infinite enumeration of Σ^*, it is countably infinite.

2.2.4 How many languages are there?

The set of languages defined on Σ is $P(\Sigma^*)$, the power set of Σ^* or the set of all subsets of Σ^*.

If $\Sigma = \phi$ then Σ^* is $\{\varepsilon\}$ & $P(\Sigma^*)$ is $\{\phi, \{\varepsilon\}\}$.

Theorem : An uncountable infinite number of languages

Theorem : If $\Sigma = \phi$ then the set of languages over Σ is uncountably infinite.

Proof : The set of languages defined on Σ is $P(\Sigma^*)$. By theorem the cardinality of Σ^*, Σ^* is countably infinite. If S is a countably infinite set, $P(S)$ is uncountably infinite. So $P(\Sigma^*)$ is uncountably infinite.

2.2.5 Functions on Languages

The set functions applied to languages are Union, Intersection, difference & complement to be useful. complement will be defined with Σ^* as the universe unless we explicitly state Otherwise.

Ex: Set Functions applied to languages

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

$L_1 = \{$Strings with an even number of a's$\}$

$L_2 = \{$Strings with no b's$\} = \{\varepsilon, a, aa, aaa, aaaa, aaaaa, \cdots\}$

$L_1 \cup L_2 = \{$all strings of just a's plus strings that contain b's & an even number of a's$\}$

$L_1 \cap L_2 = \{\varepsilon, aa, aaaa, aaaaaa, \cdots\}$

$L_2 - L_1 = \{a, aaa, aaaaa, aaaaaaa, \cdots\}$

$\neg(L_2 - L_1) = \{$Strings with at least one b$\} \cup \{$String with an even no. of a's$\}$.

operations defined on Strings are concatenation, Kleene Star & reverse.

Concatenation: Let L_1 & L_2 be two languages defined over some alphabet Σ. Then their concatenation, written $L_1 L_2$ is

$$L_1 L_2 = \{w \in \Sigma^* : \exists s \in L_1 (\exists t \in L_2 (w = st))\}.$$

Eg Let $L_1 = \{$Rat, cat, dog$\}$

$L_2 = \{$bone, food$\}$

$L_1 \cdot L_2 = \{$Ratbone, Ratfood, catbone, catfood, dogbone, dogford$\}$.

Kleene Star: Let L be a language defined over some alphabet Σ. Then the Kleene Star of L, written L^* is:

$$L^* = \{\varepsilon\} \cup \{w \in \Sigma^* : \exists k \geq 1 (\exists w_1, w_2 \cdots w_k \in L (w = w_1 w_2 \cdots w_k))\}.$$

In other words: L^* is the set of Strings that can be formed by concatenating together zero or more Strings from L.

Ex: Kleene star

Let $L = \{cat, rat, dog\}$ Then:

$L^* = \{\varepsilon, cat, rat, dog, catcat, catrat, catdog, \ldots$
$catratcatdog \ldots\}$

Reverse of L: The reverse of L, written L^R is:

$$L^R = \{w \in \Sigma^* : w = x^R \text{ for some } x \in L\}$$

In other words, L^R is the set of strings that can be formed by taking some string in L & reversing it.

Theorem: concatenation & Reverse of Languages

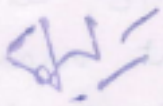
Theorem: If L_1 and L_2 are languages then

$$(L_1 L_2)^R = L_2^R L_1^R.$$

Proof: If x and y are strings, then

$$\forall x(\forall y((xy)^R = y^R x^R))$$

$$(L_1 L_2)^R = \{(xy)^R : x \in L_1 \text{ and } y \in L_2\}$$
$$= \{y^R x^R : x \in L_1 \text{ and } y \in L_2\}$$
$$= L_2^R L_1^R$$



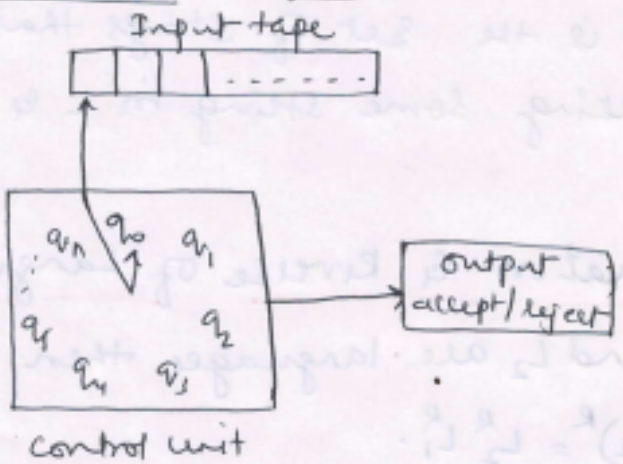
An abstract m/c is a conceptual or theoretical model of a computer h/w or s/w system which really does not exist.

Finite Automata: FSM FA are computing devices that accept/recognize regular languages. The FA acts as mathematical models and are used to study the abstract machines of abstract computing devices.

Any FA has three components

* Input tape
* control unit
* output.

Block diagram of FA



* Input tape: The i/p tape is divided into cells each of which can hold one symbol. The string to be processed is stored in these cells.
* control unit: This unit may be in any of the states denoted by q_0, q_1, q_2 ... so on. The state q_0 is considered as the start state. The m/c has at least one final state. Based on the current i/p symbol & the current state of the m/c, the state of the m/c can change.
* output: output may accept or reject. When end of input is encountered, the control unit may be in accept or reject state.

Types of finite Automata

There are three types of finite Automata

1) Deterministic finite ~~Automata~~ State machine (DFSM)
2) Non deterministic finite state machine (NDFSM)
3) Non deterministic finite state machine with ϵ-moves (ϵ-NDFSM)

Deterministic Finite State Machine

A Finite state machine (FSM) is a computational device whose input is a string and whose output is one of two values that we can call Accept & Reject. FSMs are also sometimes called finite state automata or FSAs.

A deterministic FSM (or DFSM) M is a quintuple

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

where

Q is non empty finite set of states
Σ is non empty finite set of input alphabets
δ is transition function which is a mapping from $Q \times \Sigma$ to Q based on the current state & input symbol.
$q_0 \in Q$ is the start state
$F \subseteq Q$ is the set of accepting states.

δ defines the operation of a DFSM M one step at a time we can use it to define the sequence of configurations that M will enter. We start by defining the relation yields-in-one-step, written $|-_M$. yields-in-one step relates configuration1 to configuration2 iff M can move from conf$_1$ to conf$_2$ in one step.

We can write

$$C_1 \vdash_M^* C_2$$

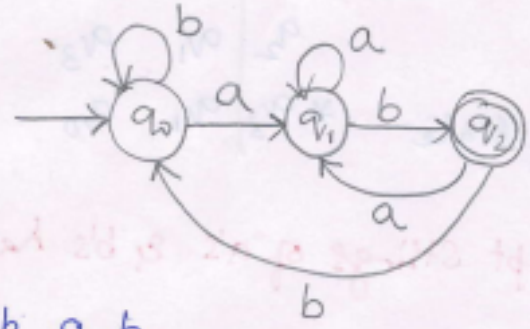
When configuration C_1 yields configuration C_2 iff M can go from C_1 to C_2 in zero or more steps.

A computation by M is a finite sequence of configurations $C_0, C_1 \cdots C_n$ for some $n \geq 0$ such that:

* C_0 is an initial configuration
* C_n is of the form (q, ε), for some state $q \in Q_M$ (i.e, the entire input string has been read) and
* $C_0 \vdash_M C_1 \vdash_M C_2 \vdash_M \cdots \vdash_M C_n$.

Eg Draw a DFSM to accept strings of a's & b's ending with the string ab.

Soln



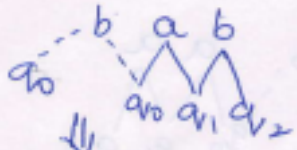
$M = (Q, \Sigma, \delta, q_0, F)$

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

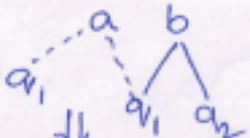
$\Sigma = \{a, b\}$

q_0 is start state

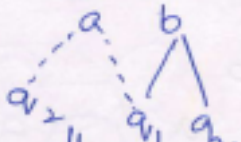
q_2 is final or accepting state



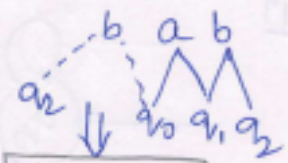
$\delta(q_0, b) = q_0$



$\delta(q_1, a) = q_1$



$\delta(q_2, a) = q_1$

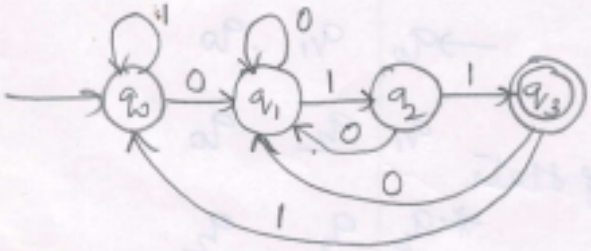


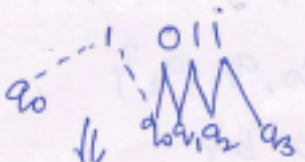
$\delta(q_2, b) = q_0$

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

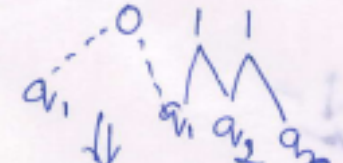
Eg Obtain a DFSM to accept strings of 0's & 1's ending with string 011.

Soln

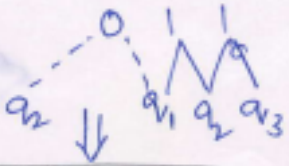




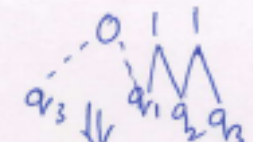
$\delta(q_0, 1) = q_0$



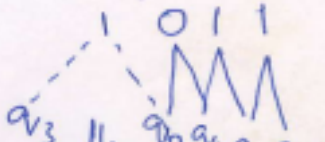
$\delta(q_1, 0) = q_1$



$\delta(q_2, 0) = q_1$



$\delta(q_3, 0) = q_1$



$\delta(q_3, 1) = q_0$

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

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

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

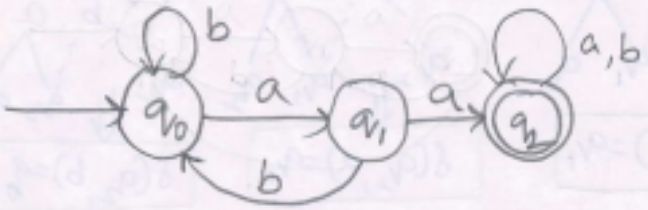
q_0 is start state

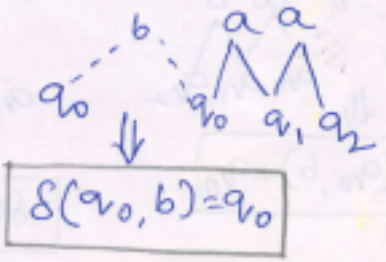
q_3 is final or accepting state

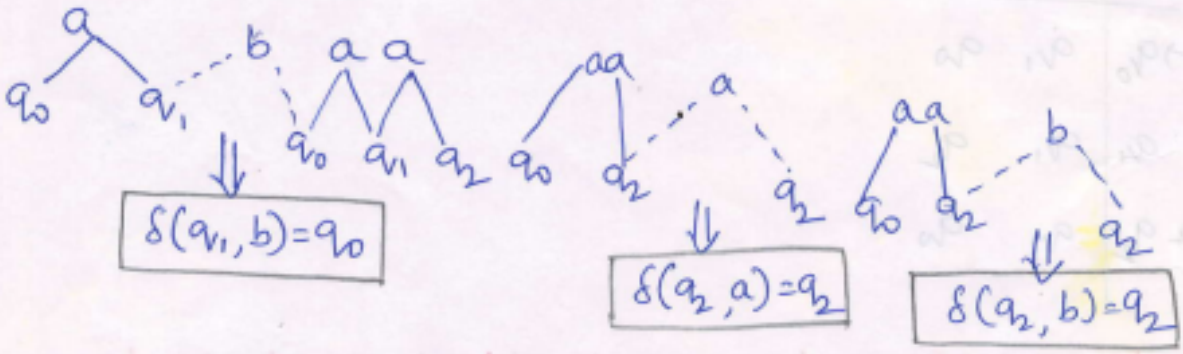
δ	0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_3
$*q_3$	q_1	q_0

Eg Obtain the DFSM to accept strings of a's & b's having substring aa

Sol:







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

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

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

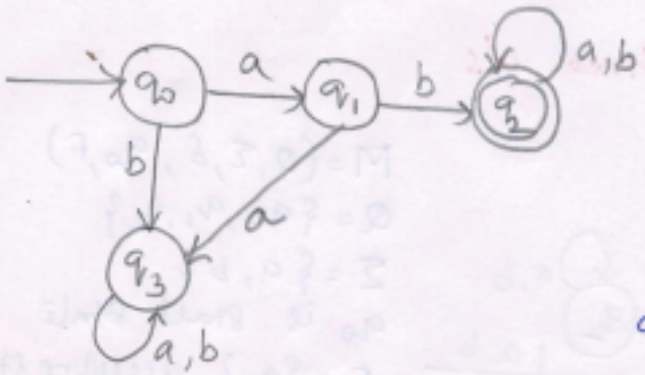
q_0 is start state

q_2 is final or accepting state

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

Eg Obtain a DFSM to accept the strings of a's & b's starting with string ab.

Soln



δ	a	b
$\rightarrow q_0$	q_1	q_3
q_1	q_3	q_2
$*q_2$	q_2	q_2
q_3	q_3	q_3

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

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

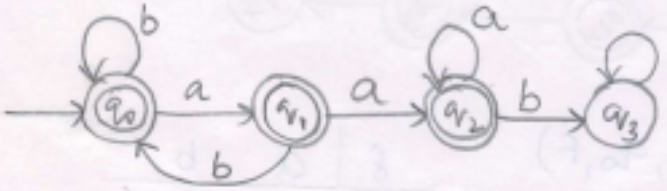
$\Sigma = \{a, b\}$

q_0 is start state

q_2 is final or accepting state

Eg Obtain a DFSM to accept a strings of a's & b's except those containing the substring aab

Soln



δ	a	b
$\rightarrow *q_0$	q_1	q_0
$*q_1$	q_2	q_0
$*q_2$	q_2	q_3
q_3	q_3	q_3

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

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

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

q_0 is start state

$F = \{q_0, q_1, q_2\}$ final/accepting state

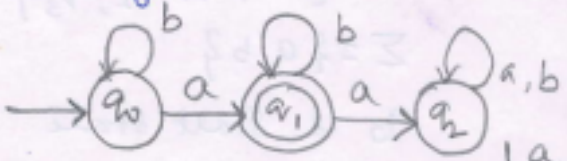
(Eg) Obtain a DFSM to accept strings of a's & b's having

i) Exactly one a

ii) atleast one a

iii) not more than three a's

Soln (i) Exactly one a



δ	a	b
→q_0	q_1	q_0
*q_1	q_2	q_1
q_2	q_2	q_2

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

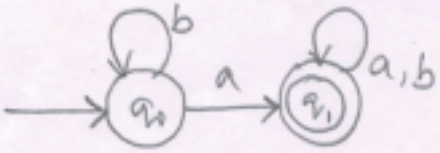
$Q = \{q_0, q_1, q_2\}$

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

q_0 is start state

$F = \{q_1\}$ accepting state

(ii) Atleast one a



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

$Q = \{q_0, q_1\}$

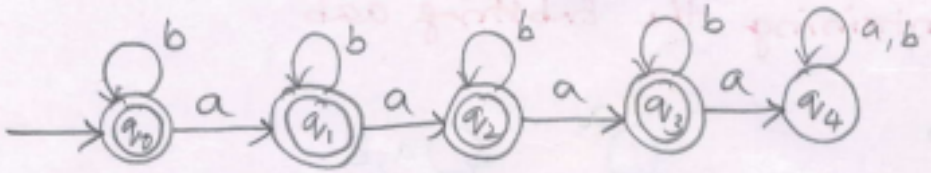
$\Sigma = \{a, b\}$

q_0 is start state

$F = \{q_1\}$ final/accepting state

δ	a	b
→q_0	q_1	q_0
*q_1	q_1	q_1

(iii) Not more than three a's



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

$Q = \{q_0, q_1, q_2, q_3, q_4\}$

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

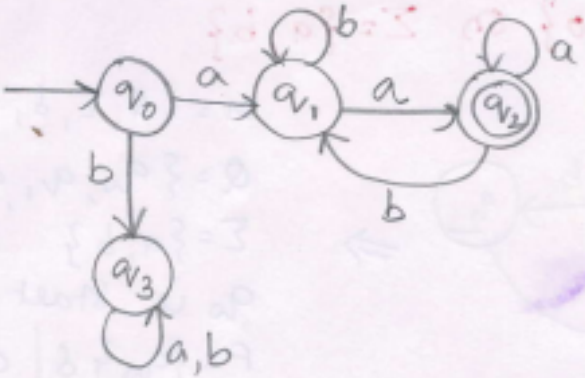
q_0 is start state

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

δ	a	b
→*q_0	q_1	q_0
*q_1	q_2	q_1
*q_2	q_3	q_2
*q_3	q_4	q_3
q_4	q_4	q_4

Eg Obtain DFSM to accept the language $L=\{awa \mid w \in (a+b)^*\}$

Soln



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

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

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

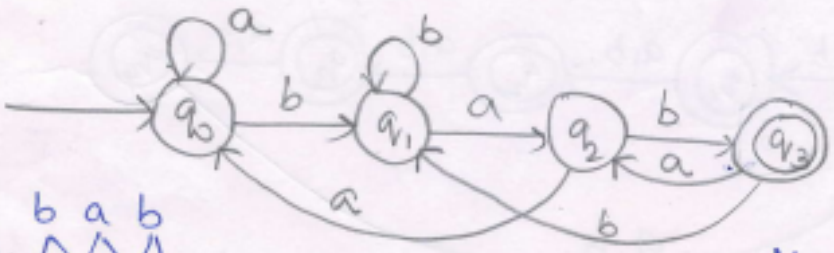
q_0 is start state

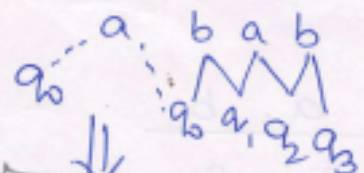
q_2 is Final state

δ	a	b
$\rightarrow q_0$	q_1	q_3
q_1	q_2	q_1
$*q_2$	q_2	q_1
q_3	q_3	q_3

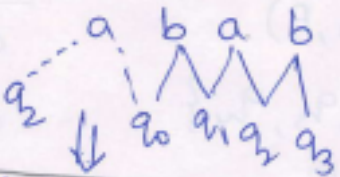
Eg Obtain a DFSM to accept the strings ending with bab (or) obtain a DFSM to accept the language $L = \{wbab \mid w \in (a+b)^*\}$

Soln

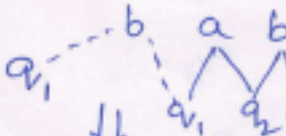




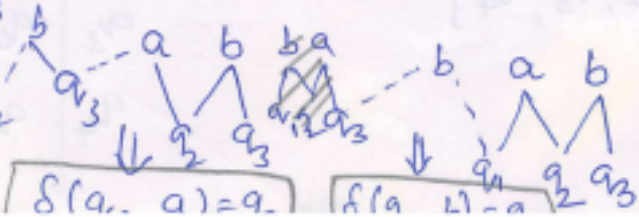
$\delta(q_0, a) = q_0$



$\delta(q_2, a) = q_0$



$\delta(q_1, b) = q_1$



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

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

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

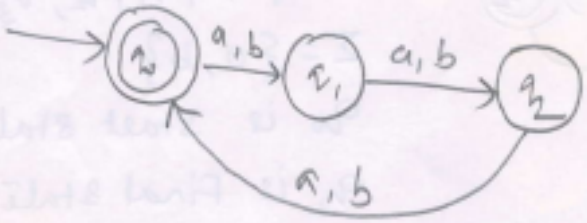
q_0 is start state

$F = \{q_3\}$

δ	a	b
$\rightarrow q_0$	q_0	q_1
q_1	q_2	q_1
q_2	q_0	q_3
$*q_3$	q_0	q_1

Eg. Obtain a DFSM to accept the language $L = \{w / |w| \bmod 3 = 0\}$ on $\Sigma = \{a, b\}$

Soln



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

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

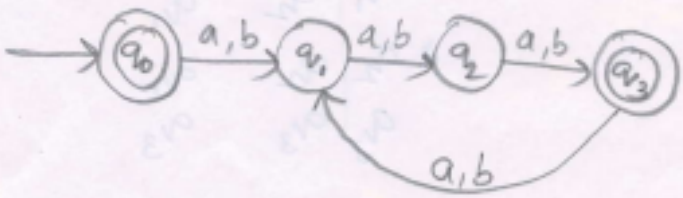
$\Sigma = \{a, b\}$

q_0 is start state

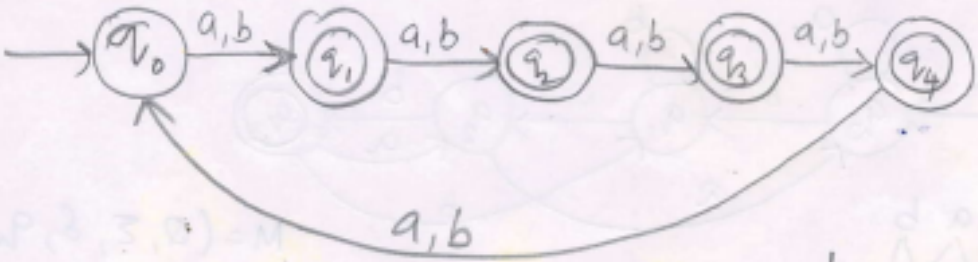
$F = \{q_0\}$

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

(or)



Eg Obtain a DFSM to accept the Language $L = \{w / |w| \bmod 5 \neq 0\}$ on $\Sigma = \{a, b\}$.



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

$Q = \{q_0, q_1, q_2, q_3, q_4\}$

q_0 is start state

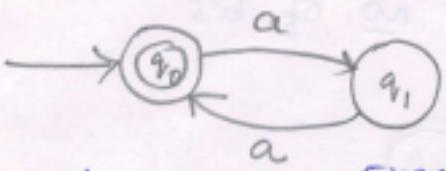
$F = \{q_1, q_2, q_3, q_4\}$

δ	a	b
q_0	q_1	q_1
q_1	q_2	q_2
q_2	q_3	q_3
q_3	q_4	q_4
q_4	q_0	q_0

Eg Obtain a DFSM to accept strings of a's & b's having
(i) Even no. of a's & even no. of b's
(ii) Even no. of a's & odd no. of b's
(iii) odd no. of a's & even no. of b's
(iv) odd no of a's & odd no. of b's

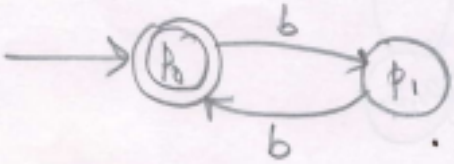
Sol: (i) Even no. of a's & even no. of b's

DFSM to accept even no. of a's



$Q_1 = \{q_0, q_1\}$

DFSM to accept ~~odd~~ Even no of b's



$Q_2 = \{p_0, p_1\}$

For Even no. of a's & Even no. of b's take the cross product of $Q_1 \times Q_2$

$\therefore Q_1 \times Q_2 = \{(q_0,p_0), (q_0,p_1), (q_1,p_0), (q_1,p_1)\}$

$\delta((q_0,p_0), a) = q_1 \cup p_0 = (q_1, p_0)$

$\delta((q_0,p_0), b) = \delta(q_0,b) \cup \delta(p_0,b) = q_0 \cup p_1 = (q_0, p_1)$

$\delta((q_0,p_1), a) = \delta(q_0,a) \cup \delta(p_1,a) = q_1 \cup p_1 = (q_1, p_1)$

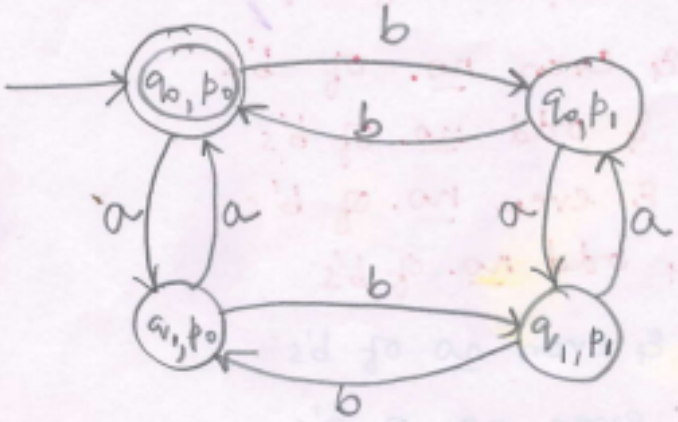
$\delta((q_0,p_1), b) = \delta(q_0,b) \cup \delta(p_1,b) = q_0 \cup p_0 = (q_0, p_0)$

$\delta((q_1,p_0), a) = \delta(q_1,a) \cup \delta(p_0,a) = q_0 \cup p_0 = (q_0, p_0)$

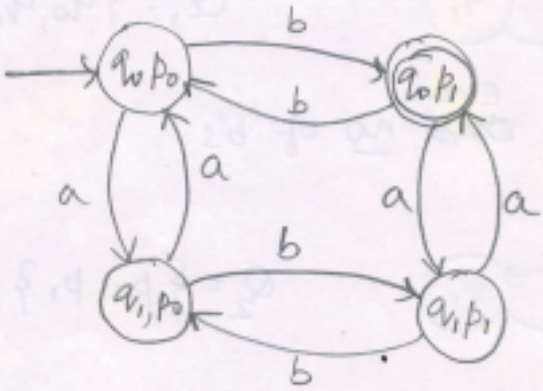
$\delta((q_1,p_0), b) = \delta(q_1,b) \cup \delta(p_0,b) = q_1 \cup p_1 = (q_1, p_1)$

$\delta((q_1,p_1), a) = \delta(q_1,a) \cup \delta(p_1,a) = q_0 \cup p_1 = (q_0, p_1)$

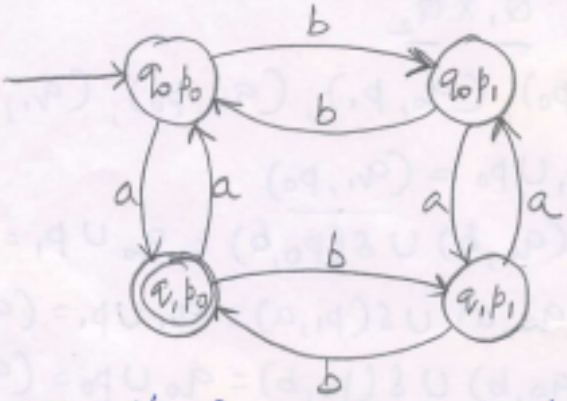
$\delta((q_1,p_1), b) = \delta(q_1,b) \cup \delta(p_1,b) = q_1 \cup p_0 = (q_1, p_0)$



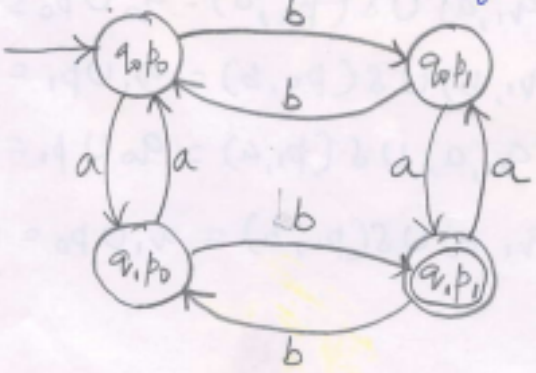
(ii) Even no of a's & odd no. of b's



(iii) Odd no. of a's & Even no. of b's

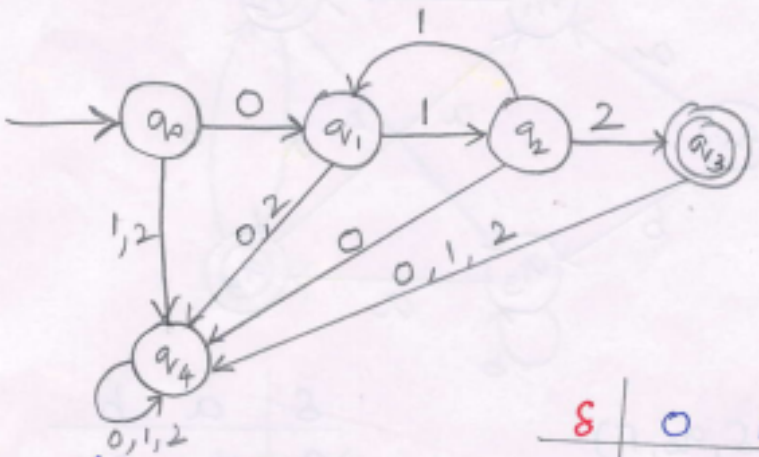


(iv) Odd no. of a's & odd no. of b's



Eg Obtain a DFSM to accept strings of 0's, 1's & 2's beginning with a '0' followed by odd no. of 1's & ending with a '2'.

Soln:



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

$Q = \{q_0, q_1, q_2, q_3, q_4\}$

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

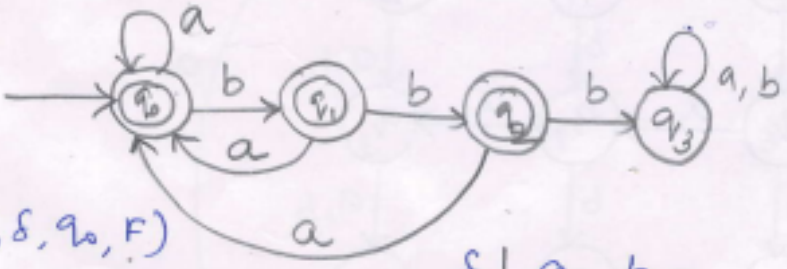
q_0 is start state

$F = \{q_3\}$

δ	0	1	2
$\rightarrow q_0$	q_1	q_4	q_4
q_1	q_4	q_2	q_4
q_2	q_4	q_1	q_3
$*q_3$	q_4	q_4	q_4
q_4	q_4	q_4	q_4

Eg Obtain a DFSM to accept strings of a's & b's with atmost two consecutive 'b's.

Soln



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

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

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

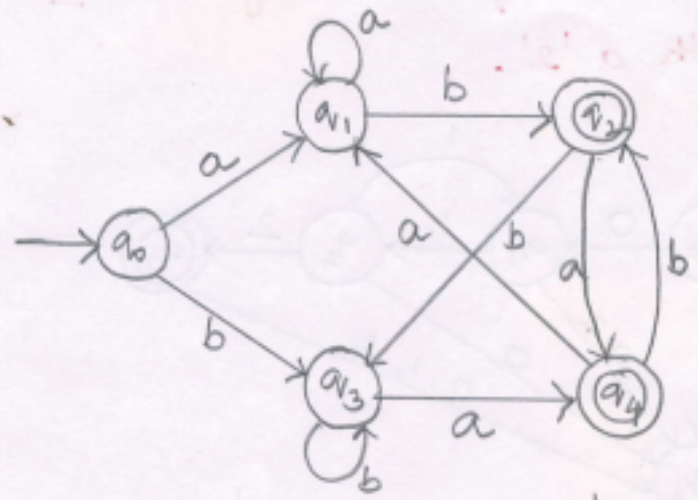
q_0 is start state

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

δ	a	b
$\rightarrow *q_0$	q_0	q_1
$*q_1$	q_0	q_2
$*q_2$	q_0	q_3
q_3	q_3	q_3

Eg Obtain a DFSM to accept the string ending with ab or ba

Soln



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

$Q = \{q_0, q_1, q_2, q_3, q_4\}$

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

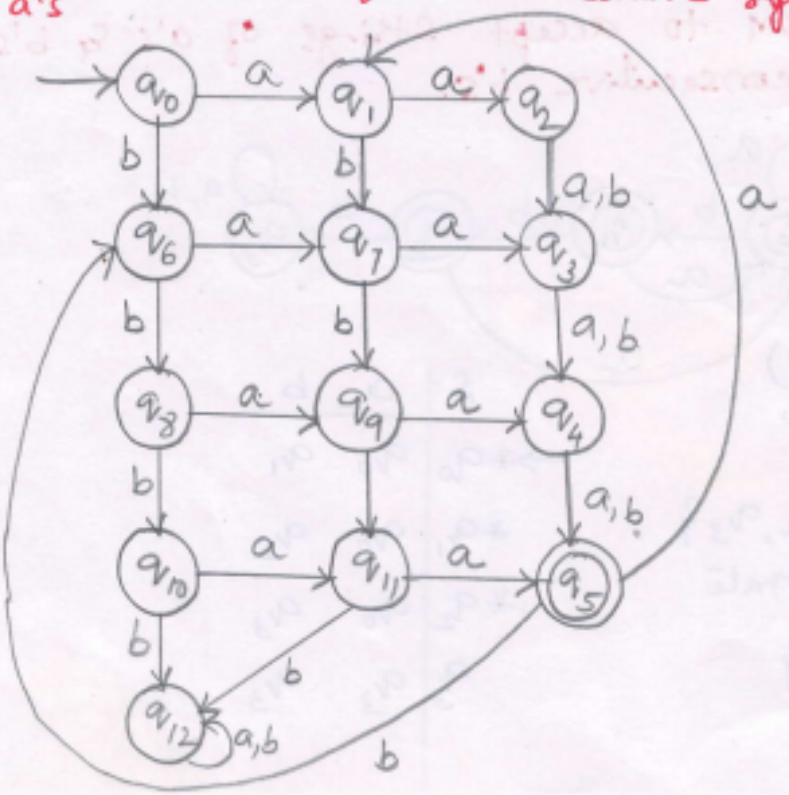
q_0 is start state

$F = \{q_2, q_4\}$

δ	a	b
$\rightarrow q_0$	q_1	q_3
q_1	q_1	q_2
$*q_2$	q_4	q_3
q_3	q_4	q_3
$*q_4$	q_1	q_2

Eg Obtain a DFSM to accept strings of a's & b's such that each block of 5 consecutive symbols have atleast 2 a's

Soln:



Three types of representing DFSM

* Pattern recognition / pattern matching
* Divisible by k problem.
* Modulo k counter problem.

Divisible by k problem

The steps to be followed to find finite state automata machine for divisible by k problems is given below.

Step 1 : Identify the radix, input alphabet & the divider k.

Step 2 : compute the possible remainders, these remainders represents the states of DFSM

Step 3 : find the transition

$$\delta(q_i, a) = q_j$$

where $j = (r \times i + d) \bmod k$

Step 4 : construct a DFSM using the transitions obtained in step 3.

Eg Construct a DFSM which accepts strings of 0's & 1's where the value of each string is represented as a binary no. only. The strings representing 0 modulo 5 should be accepted. Ex: 0000, 0101, 1010, 1111 etc.

Soln:

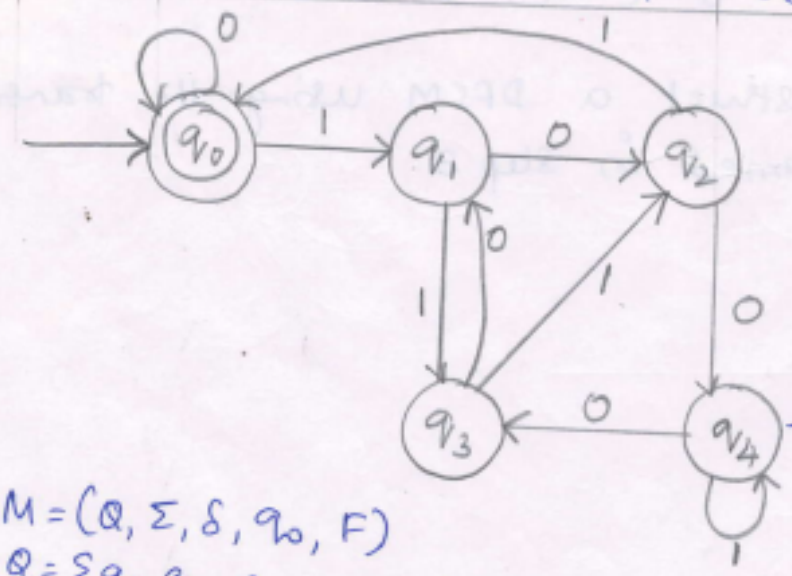
$r = 2$

$d = 0, 1$

$k = 5$

$i = 0, 1, 2, 3, 4$ (remainders of 5)

i	d	$(r \times i + d) \bmod k = j$	$\delta(q_i, a) = q_j$
$i=0$	0	$(2\times0+0) \bmod 5 = 0$	$\delta(q_0, 0) = q_0$
	1	$(2\times0+1) \bmod 5 = 1$	$\delta(q_0, 1) = q_1$
$i=1$	0	$(2\times1+0) \bmod 5 = 2$	$\delta(q_1, 0) = q_2$
	1	$(2\times1+1) \bmod 5 = 3$	$\delta(q_1, 1) = q_3$
$i=2$	0	$(2\times2+0) \bmod 5 = 4$	$\delta(q_2, 0) = q_4$
	1	$(2\times2+1) \bmod 5 = 0$	$\delta(q_2, 1) = q_0$
$i=3$	0	$(2\times3+0) \bmod 5 = 1$	$\delta(q_3, 0) = q_1$
	1	$(2\times3+1) \bmod 5 = 2$	$\delta(q_3, 1) = q_2$
$i=4$	0	$(2\times4+0) \bmod 5 = 3$	$\delta(q_4, 0) = q_3$
	1	$(2\times4+1) \bmod 5 = 4$	$\delta(q_4, 1) = q_4$



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

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

$Q = \{q_0, q_1, q_2, q_3, q_4\}$

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

q_0 is start state

$F = \{q_0\}$

Eg Draw a DFSM to accept decimal strings divisible by 3.

Soln $r = 10$ (decimal nos)

$d = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$

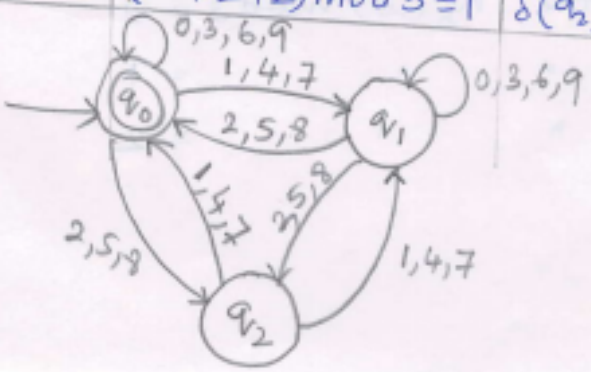
$k = 3$

$i = 0, 1, 2$

Note: For sake of convinence, let us put the digits from 0 to 9 based on the remainders, we get after dividing by 3 as given below.

* $\{0, 3, 6, 9\}$ with '0' as the remainder
  So δ from $\{0, 3, 6, 9\} \Rightarrow \delta$ from $\{0\}$
* $\{1, 4, 7\}$ with '1' as the remainder
  So δ from $\{1, 4, 7\} \Rightarrow \delta$ from $\{1\}$
* $\{2, 5, 8\}$ with '2' as the remainder
  So δ from $\{2, 5, 8\} \Rightarrow \delta$ from $\{2\}$

i	d	$j = (r*i+d) \bmod k$	$\delta(q_i, a) = q_j$
$i=0$	0	$(10*0+0) \bmod 3 = 0$	$\delta(q_0, 0) = q_0 \Rightarrow \delta(q_0 \{0,3,6,9\}, 0) = q$
	1	$(10*0+1) \bmod 3 = 1$	$\delta(q_0, 1) = q_1 \Rightarrow \delta(q_0 \{1,4,7\}, 1) = q$
	2	$(10*0+2) \bmod 3 = 2$	$\delta(q_0, 2) = q_2 \Rightarrow \delta(q_0 \{2,5,8\}, 2) = q_2$
$i=1$	0	$(10*1+0) \bmod 3 = 1$	$\delta(q_1, 0) = q_1 \Rightarrow \delta(q_1 \{0,3,6,9\}, 0) = q_1$
	1	$(10*1+1) \bmod 3 = 2$	$\delta(q_1, 1) = q_2 \Rightarrow \delta(q_1 \{1,4,7\}, 1) = q_2$
	2	$(10*1+2) \bmod 3 = 0$	$\delta(q_1, 2) = q_0 \Rightarrow \delta(q_1 \{2,5,8\}, 2) = q_0$
$i=2$	0	$(10*2+0) \bmod 3 = 2$	$\delta(q_2, 0) = q_2 \Rightarrow \delta(q_2 \{0,3,6,9\}, 0) = q_2$
	1	$(10*2+1) \bmod 3 = 0$	$\delta(q_2, 1) = q_0 \Rightarrow \delta(q_2 \{1,4,7\}, 1) = q_0$
	2	$(10+2+2) \bmod 3 = 1$	$\delta(q_2, 2) = q_1 \Rightarrow \delta(q_2 \{2,5,8\}, 2) = q_1$



Modulo k counter problem

Eg: Obtain a DFSM to accept strings of a's & b's such that no. of a's is divisible by 5 & no. of b's divisible by 3.

Soln:

$L = \{ w / w \in (a+b)^* \}$, such that $n_a(w) \bmod 5 = 0$ & $n_b(w) \bmod 3 = 0$

$n_a(w) \bmod 5$ gives the remainder 0, 1, 2, 3, 4 & it can be represented as

$Q_1 = \{q_0, q_1, q_2, q_3, q_4\}$

$n_b(w) \bmod 3$ gives the remainder 0, 1, 2 & it is represented as

$Q_2 = \{p_0, p_1, p_2\}$

Since each state of DFSM should keep track of $n_a(w) \bmod 5$ & $n_a(w) \bmod 3$ the possible state of the DFSM can be obtained by $Q_1 \times Q_2$ (cross product) & represented as given below:

$Q_1 \times Q_2 = \{(q_0,p_0), (q_0,p_1), (q_0,p_2), (q_1,p_0), (q_1,p_1), (q_1,p_2), (q_2,p_0), (q_2,p_1), (q_2,p_2), (q_3,p_0), (q_3,p_1), (q_3,p_2), (q_4,p_0), (q_4,p_1), (q_4,p_2)\}$

where q_0 indicates that n_a in $w \bmod 5 = 0$

q_1 " " " " " " = 1

q_2 " " " " " " = 2

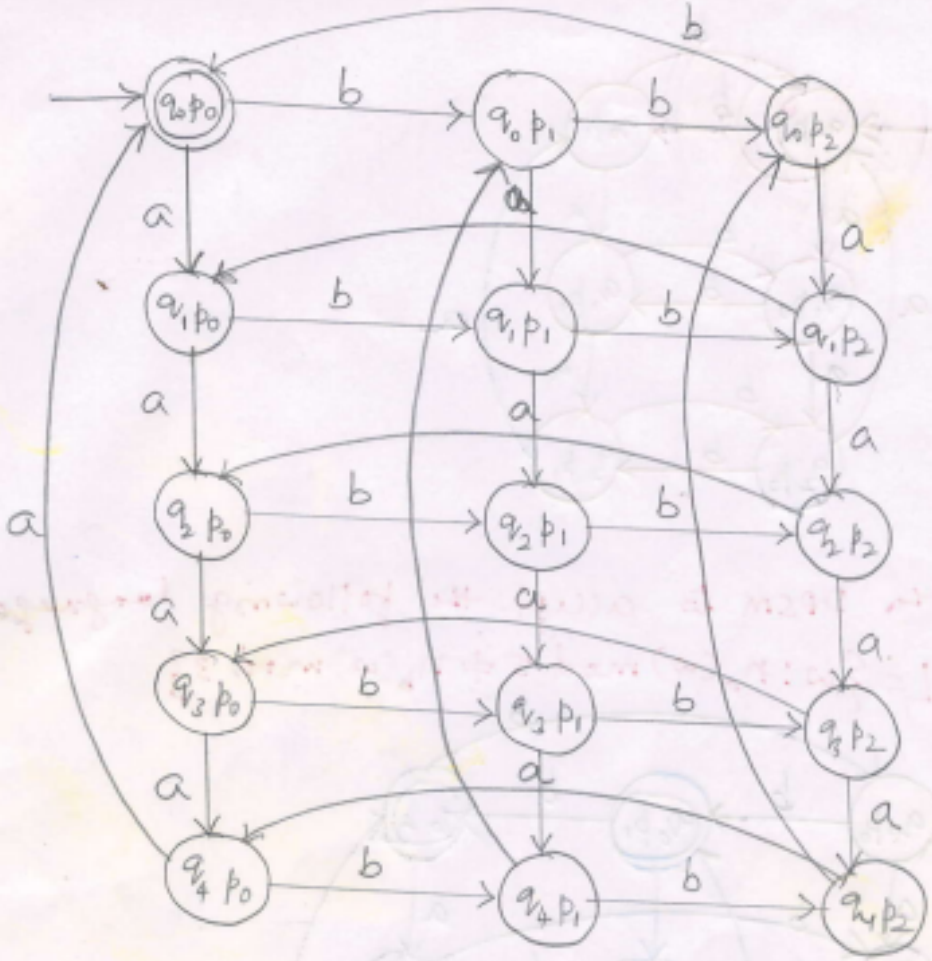
q_3 " " " " " " = 3

q_4 " " " " " " = 4

p_0 " the $n_b(w) \bmod 3 = 0$

p_1 " " " = 1

p_2 " " " = 2



Eg Obtain a DFSM to accept strings of a's & b's such that $L = \{w \mid w \in (a+b)^*$ such that $n_a(w) \bmod 3 = 0$ & $n_b(w) \bmod 2 = 0\}$

Soln:

$Q_1 = \{q_0, q_1, q_2\}$

$Q_2 = \{p_0, p_1\}$

$\therefore Q_1 \times Q_2 = \{(q_0,p_0), (q_0,p_1), (q_1,p_0), (q_1,p_1), (q_2,p_0), (q_2,p_1)\}$

$\delta(q_0, a) = q_1$

$\delta(q_1, a) = q_2$

$\delta(q_2, a) = q_0$

$\delta(p_0, b) = p_1$

$\delta(p_1, b) = p_0$

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

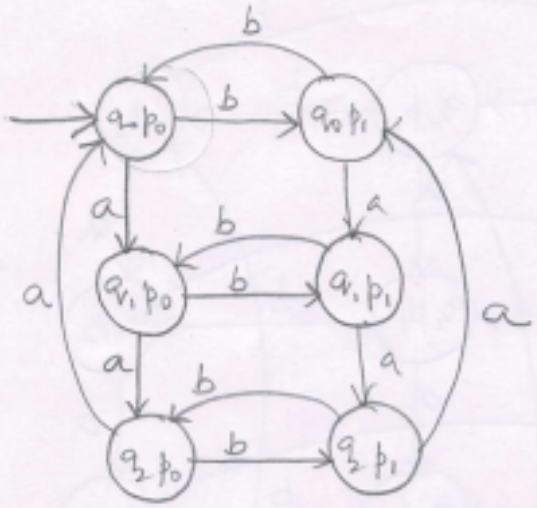
$Q = \{q_0, q_1, q_2, p_0, p_1\}$

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

(q_0, p_0) is start state

$F = (q_0, p_0)$ is accepting state

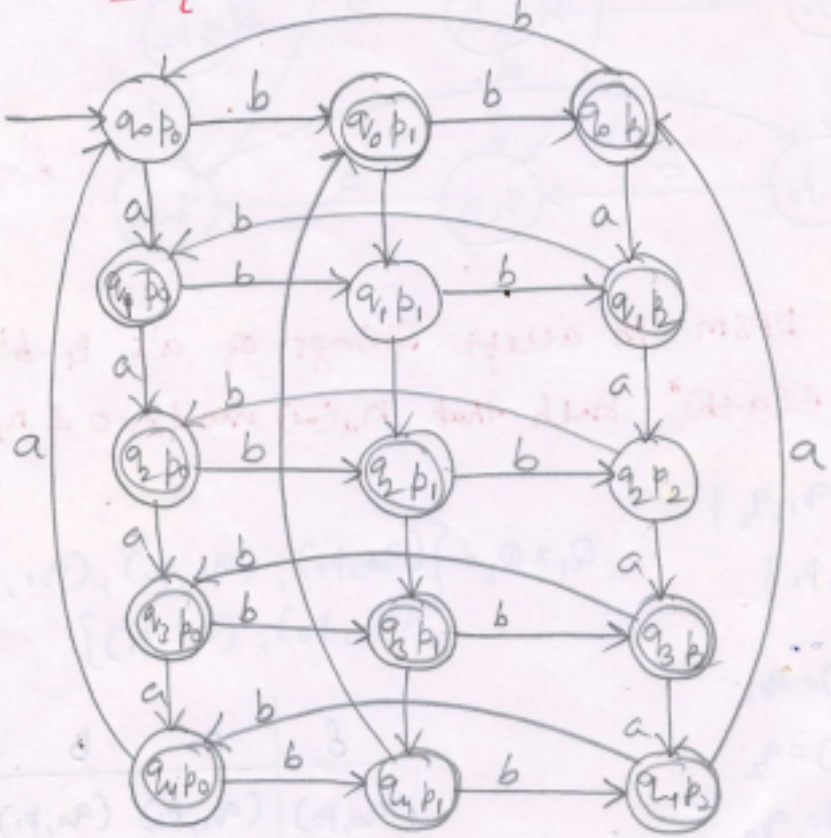
δ	a	b
(q_0,p_0)	(q_1,p_0)	(q_0,p_1)
(q_0,p_1)	(q_1,p_1)	(q_0,p_0)
(q_1,p_0)	(q_2,p_0)	(q_1,p_1)
(q_1,p_1)	(q_2,p_1)	(q_1,p_0)
(q_2,p_0)	(q_0,p_0)	(q_2,p_1)
(q_2,p_1)	(q_0,p_1)	(q_2,p_0)



(Eg) Construct the DFSM to accept the following language

$$L = \{w : n_a(w) \bmod 5 \neq n_b(w) \bmod 3\}$$

Soln:



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

$Q = \{(q_0,p_0), (q_0,p_1), (q_0,p_2), (q_1,p_0), (q_1,p_1), (q_1,p_2), (q_2,p_0), (q_2,p_1), (q_2,p_2), (q_3,p_0), (q_3,p_1), (q_3,p_2), (q_4,p_0), (q_4,p_1), (q_4,p_2)\}$

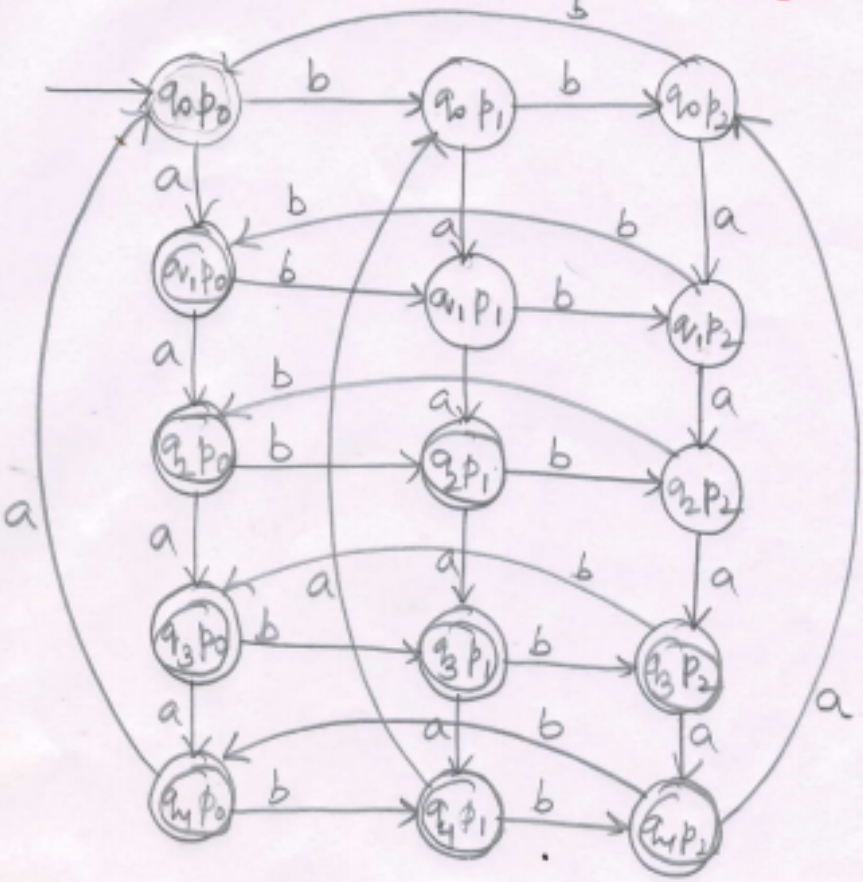
$\Sigma = \{a, b\}$

(q_0,p_0) is start state

$F = \{(q_0,p_1), (q_0,p_2), (q_4,p_0), (q_1,p_2), (q_2,p_0), (q_2,p_1), (q_3,p_0),$

Eg Construct the DFSM to accept the following language

$L = \{ w : n_a(w) \bmod 5 > n_b(w) \bmod 3 \}$



5.4 Non Deterministic finite state Machine (NFSM) (NDFSM)

A non deterministic FSM (or NDFSM) M is a quintuple

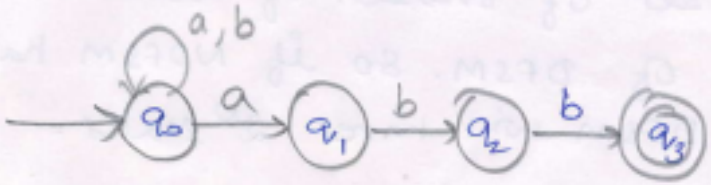
$$M = (Q, \Sigma, \delta, q_0, F)$$

where:

* Q is a finite set of states
* Σ is an alphabet
* $q_0 \in Q$ is a start state
* δ is transition function which is mapping from $Q \times \Sigma$ to subsets of 2^Q.
* $F \subseteq$ is the set of final state.

Eg Construct an NDFSM to accept strings of a's & b's ending with the string abb.

Sol"



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

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

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

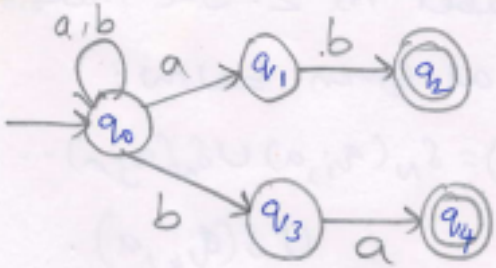
q_0 is start state

$F = \{q_3\}$

δ	a	b
$\rightarrow q_0$	$\{q_0, q_1\}$	q_0
q_1	$\emptyset$	q_2
q_2	$\emptyset$	q_3
$*q_3$	$\emptyset$	$\emptyset$

Eg Construct an NDFSM to accept the strings of a's & b's ending with string ab or ba

Sol"



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

$Q = \{q_0, q_1, q_2, q_3, q_4\}$

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

$F = \{q_2, q_4\}$

δ	a	b
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0, q_3\}$
q_1	$\emptyset$	q_2
$*q_2$	$\emptyset$	$\emptyset$
q_3	q_4	$\emptyset$
$*q_4$	$\emptyset$	$\emptyset$

Conversion from NDFSM to DFSM

Two methods

1) Subset construction method
2) Lazy evaluation method.

Subset construction method

Step 1: The start state of NDFSM is the start state of DFSM.

Step 2: The input alphabets of NDFSM are the input alphabets of DFSM.

Step 3: The set of subsets of NDFSM will be the states of DFSM. So if NDFSM has N states then DFSM will have 2^N states.

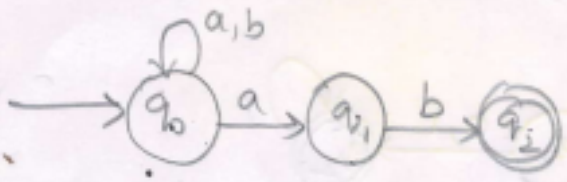
Step 4: Identify the final states of DFSM
If $\{q_i, q_j \cdots q_k\}$ is a state in DFSM. Then $\{q_i, q_j \cdots q_k\}$ will be final state of DFSM provided one of the $\{q_i, q_j \cdots q_k\}$ is the final state of NDFSM.

Step 5: Identify the transition of DFSM for each state $\{q_i, q_j \cdots q_k\}$ in DFSM for each input alphabet in Σ. The transition can be obtained as given below:

$$\delta_D(\{q_i, q_j \cdots q_k\}, a) = \delta_N(q_i, a) \cup \delta_N(q_j, a) \cdots \cup \delta(q_k, a).$$

Thus DFSM can be obtained using subset construction method.

Eg Obtain DFSM for the following NDFSM using subset construction method.



Subset

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

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

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

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

q_0 is start state

δ	a	b
→ $\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	ϕ	$\{q_2\}$
* $\{q_2\}$	ϕ	ϕ
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
* $\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
* $\{q_1, q_2\}$	ϕ	$\{q_2\}$
* $\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

$\delta((q_0,q_1),a) = \{q_0,q_1\} \cup \phi = \{q_0,q_1\}$

$\delta(\{q_0,q_1\},b) = \{q_0\} \cup \{q_2\} = \{q_0,q_2\}$

$\delta(\{q_0,q_2\},a) = \{q_0,q_1\} \cup \phi = \{q_0,q_1\}$

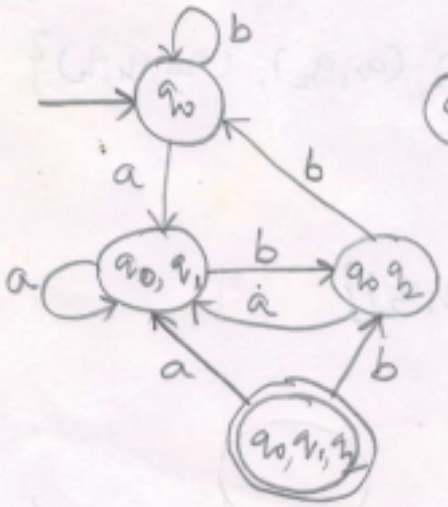
$\delta(\{q_0,q_2\},b) = \{q_0\} \cup \phi = \{q_0\}$

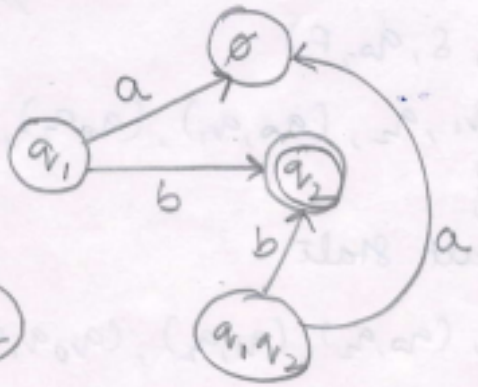
$\delta(\{q_1,q_2\},a) = \phi \cup \phi = \phi$

$\delta(\{q_1,q_2\},b) = \phi \cup \phi = \{\phi_2\}$

$\delta(\{q_0,q_1,q_2\},a) = \{q_0,q_1\} \cup \phi \cup \phi = \{q_0,q_1\}$

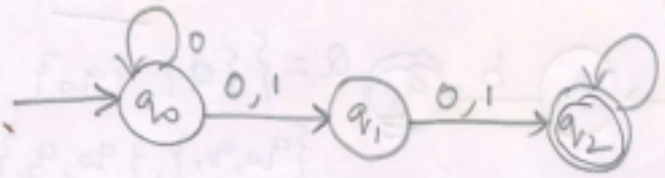
$\delta(\{q_0,q_1,q_2\},b) = \{q_0\} \cup \{q_2\} \cup \phi = \{q_0,q_2\}$





Note: From start state, if it is not possible to reach to any of the state then state can be eliminated.

Eg Obtain DFSM for the following NDFSM using subset construction method.

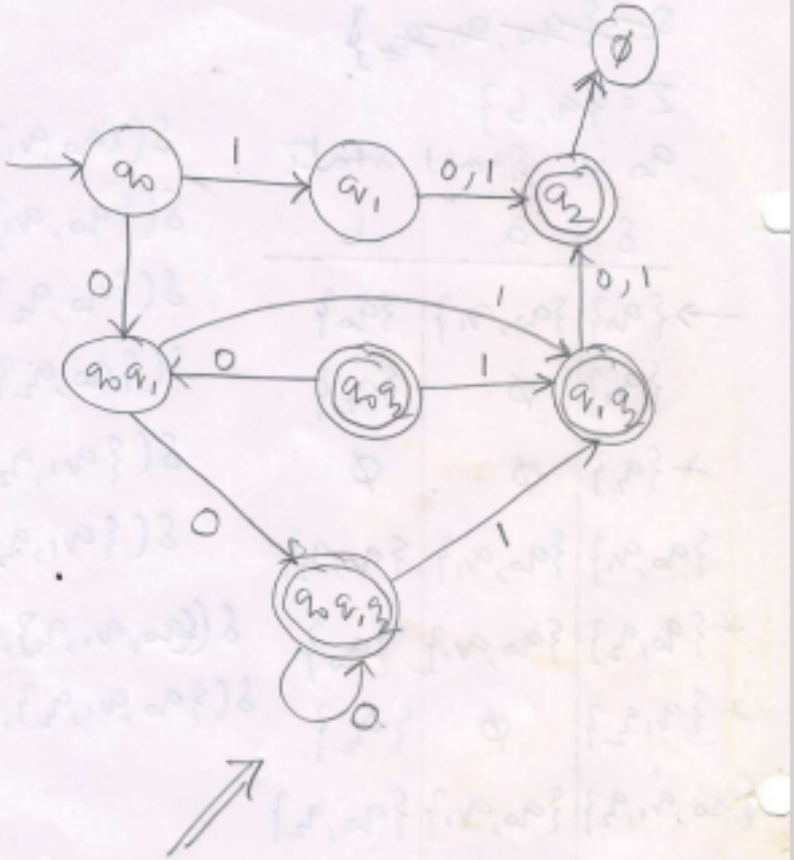


Soln

q_0 is start state

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

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



DFSM

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

$Q = \{q_0, q_1, q_2, (q_0, q_1), (q_0 q_2), (q_1, q_2), (q_0, q_1, q_2)\}$

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

q_0 is start state

$F = \{(q_2), (q_0, q_2), (q_1, q_2), (q_0 q_1 q_2)\}$

Finite state transducers

FST is an extension of FA where it has both an input & output.

Definition : A finite state machine where an output is associated with either a transition or a state is called finite state transducer. The finite state transducers are associated with output and are classified as given below:

* Mealy machine
* Moore machine

Mealy machine : A mealy machine is a finite state machine whose output values are determined both by its current state & the current inputs. A Mealy machine is a deterministic finite-state transducer for each state & input, at most one transition is possible.

Defn: A finite state machine where an output is associated with each transition is called Mealy machine.

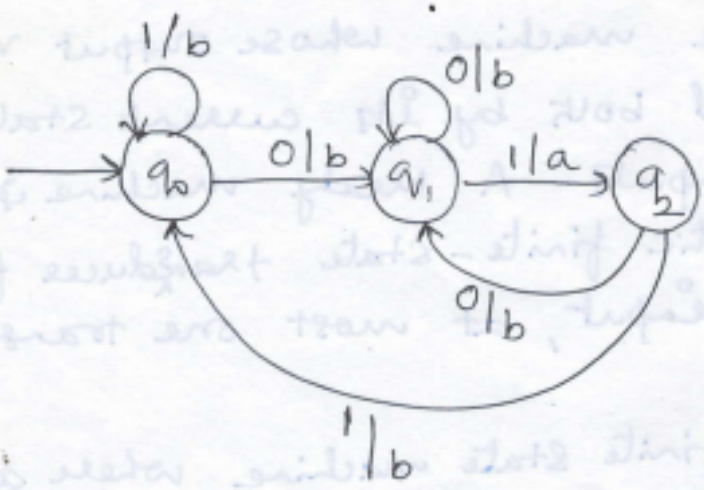
$$M = (Q, \Sigma, I, \delta, \lambda, q_0)$$

where

Q is finite set of states
Σ is set of input alphabets
I is set of output alphabets
δ is transition function from Q×Σ to Q
λ is an output function which is a mapping from Q×Σ to I
q_0 is the start state of machine.

(Eg) Construct a Mealy machine that prints 'a' whenever the sequence '01' is encountered in any input binary string

Soln:



$\lambda = Q \times \Sigma \to I$
$(q_0, 0) \to b$
$(q_0, 1) \to b$
$(q_1, 0) \to b$
$(q_1, 1) \to a$
$(q_2, 0) \to b$
$(q_2, 1) \to b$

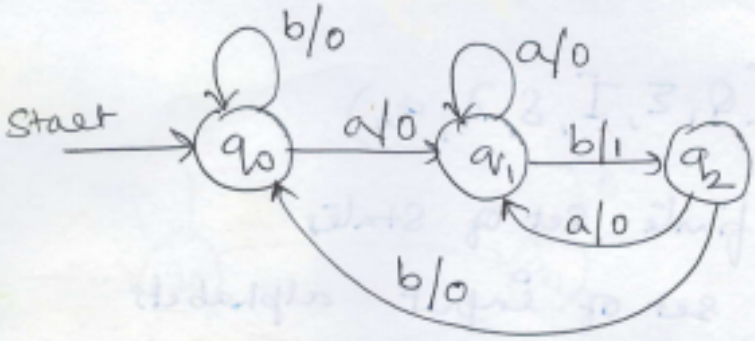
(Eg)

0 1 1 0
b a, b b

1 0 0 0
b b b b
no 'a'

(Eg) Construct a Mealy machine which accepts strings of a's & b's & count the number of times the pattern "ab" is present in the string.

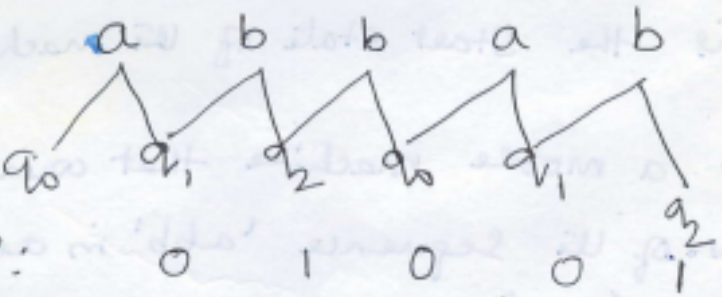
Soln:



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

output $= \{0, 1\}$

The sequence of moves made by the Mealy machine for the input string abbab is :



Outputs: 0 1 0 0 1

Observe that in the output two 1's are present so the pattern "ab" is present 2 times in the given string abbab.

Moore machine

Defn: A finite state machine where an output is associated with each state is called Moore machine.

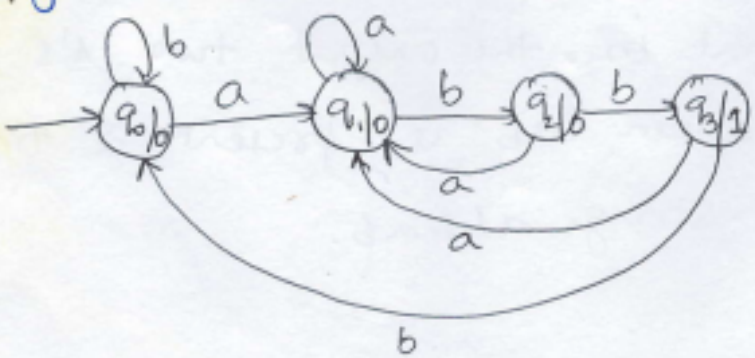
$$M = (Q, \Sigma, I, \delta, \lambda, q_0)$$

Q is finite set of states
Σ is set of input alphabets
I is set of output alphabets
δ is the transition function from $Q \times \Sigma$ to Q.
λ is an output function which is mapping from Q to I
q_0 is the start state of the machine.

n → bit string
n+1 → o/p

Eg Construct a moore machine that counts the occurences of the sequence 'abb' in any input strings over $\{a,b\}$

Sol:

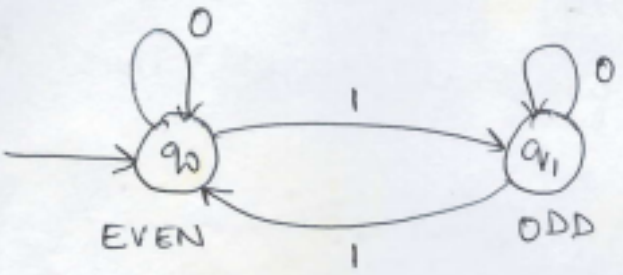


$\Sigma = \{a, b\}$
$\lambda = \{0, 1\}$
$\lambda : Q \to I$
$q_0 \to 0$
$q_1 \to 0$
$q_2 \to 0$
$q_3 \to 1$

a b b
q_0 q_1 q_2 q_3
0 0 0 1 → occurs 1 time.

Eg Construct a Moore machine which accepts strings of 0's & 1's output EVEN when the input string has even number of 1's & output ODD when the input string has odd number of 1's.

Sol?:



Machine based hierarchy of language classes

The hierarchy of language classes is given below

* Regular language [accepted by FSM]
* Context free language [accepted by PDA]
* Decidable language [accepted by TM]
* Semi decidable language [accepted by TM]

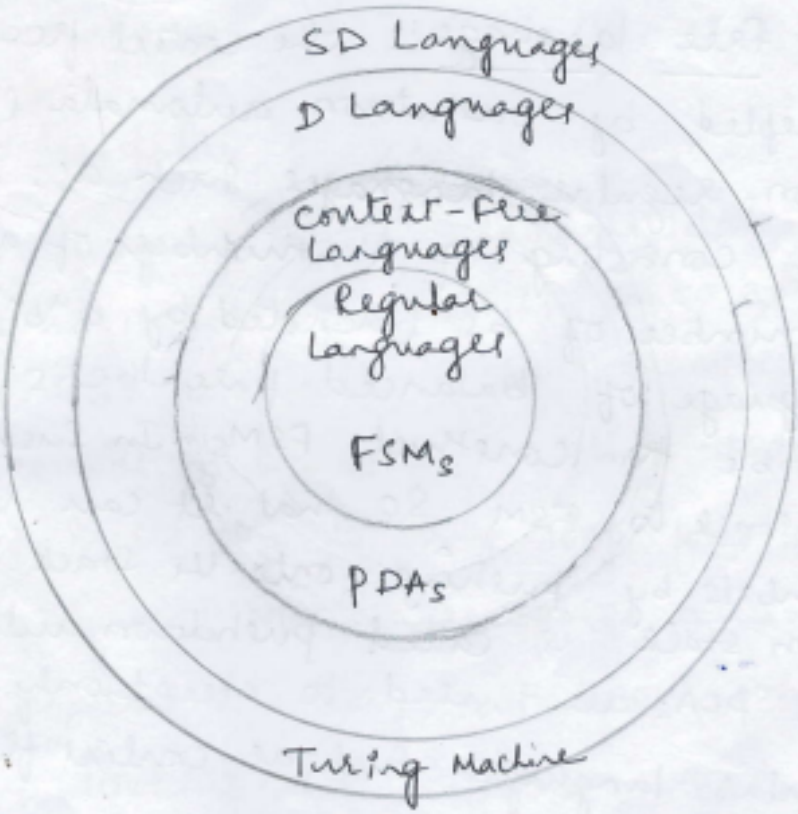


Fig: A hierarchy of language classes.

Regular language: The regular languages are accepted by FSM (Finite State machine). The FSM starts from the start state & accepts one character at a time & changes the state based on current state & input symbol.

After scanning the entire string, if the machine is in a final state, the string is accepted by the machine. Otherwise, the string is rejected by the machine. The language accepted by FSM is called regular language.

fig: A Simple FSM.

Context free language: The context free languages are accepted by pushdown automata (PDA). For non-regular languages such as: The language consisting of 'n' number of a's followed by 'n' number of b's (denoted by $a^n b^n$) or the language of balanced parenthesis. It is not possible to construct FSMs. In such situation, we add stack to FSM so that it can remember some symbols by pushing onto the stack. The FSM with stack is called pushdown automation (or PDA). PDAs are used to accept only the non regular languages such as context free languages.

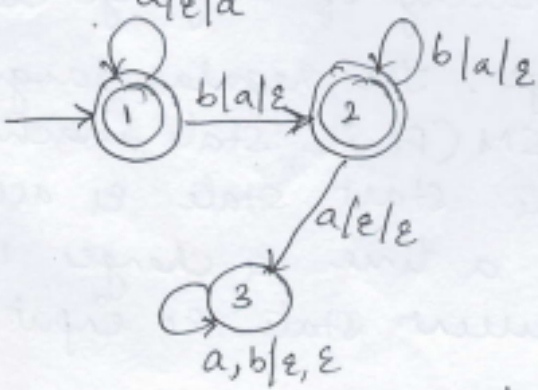


Fig: A Simple PDA that accepts $a^n b^n$.

Decidable languages: For non-context free languages such as $a^n b^n c^n$ it is not possible to construct a PDA. For such problems we can construct Turing machine. Turing machine can accept all types of languages including regular language & context free language

A language L is decidable iff there exists a Turing machine M that halts on all inputs, accepts all strings that are in L, and rejects all strings that are not in L. In other words, M can always say yes or no, as appropriate.

Semidecidable Languages

A language L is semidecidable iff there exists a Turing Machine M that accepts all strings that are in L & fails to accept every string that is not in L. Given a string that is not in L, M may reject or it may loop forever.
In other words, M can recognize a solution & then say yes, but it may not know when it should give up looking for a solution & say no.

Decision problem :

The problem for which we must take yes/no decision is called decision problem. The algorithm that is used to solve a decision problem is called decision procedure. The output of the decision algorithm is always a Boolean value i.e., either true or false & halts on all inputs.

(Eg) Write a decision procedure to check whether given number is even or not.

A) The program to check whether a given number is even or not is a decision problem.. The decision procedure is given below:

```
even (x: integer)
{
 if (x%2==0) return true; //program halts: returns true when x is even.
 return false;            // program halts: returns false when x is odd.
 }
```

Since the program halts on all inputs & returns the correct answer whenever the number is even or odd, it is a decision procedure.

(Eg) Write a decision procedure to check whether given number is prime or not.

```
prime(n: integer)
 { for i = 2 to √n
    if (n%i==0) return false
   end for
   return true;
```

Decidable, undecidable & semi-decidable decision problem:

A decision problem is decidable if and only if the answer to this problem is yes.

A decision problem is undecidable if and only if the answer to this problem is no.

A decision problem is semi-decidable if and only if there exists an algorithm that halts & returns true if and only if the answer is true. When false is the answer, the algorithm may halt & return false or it may loop infinitely without returning false.