

# Theory of Computation

## Introduction

The purpose of theory of computation is to develop formal mathematical models of computation that reflect real-world computers.

Theory of complexity can be divided into three areas.

- Complexity Theory
- Computability Theory
- Automata Theory

- Complexity Theory : classify problems according to their degree of "difficulty".
- Computability Theory : classify problems as being solvable or unsolvable by a computer.
- Automata Theory : deals with definitions and properties of different computational models. It also asks whether all models have same power or can some solve more problems than others.

## Pigeon-hole principle

If  $(n+1)$  or more objects are placed in  $n$  boxes, then atleast one box has more than one object.

In other words, if  $A$  &  $B$  are sets such that  $|A| > |B|$ , then there is no one-one function from  $A$  to  $B$ .

## Finite Automata and Regular languages

An example : a toll gate.

Suppose we want a computer to control a toll gate.

The gate is opened when  $\geq 25$  is paid. We accept  $\{5, 10\}$  & ~~25~~ coins.

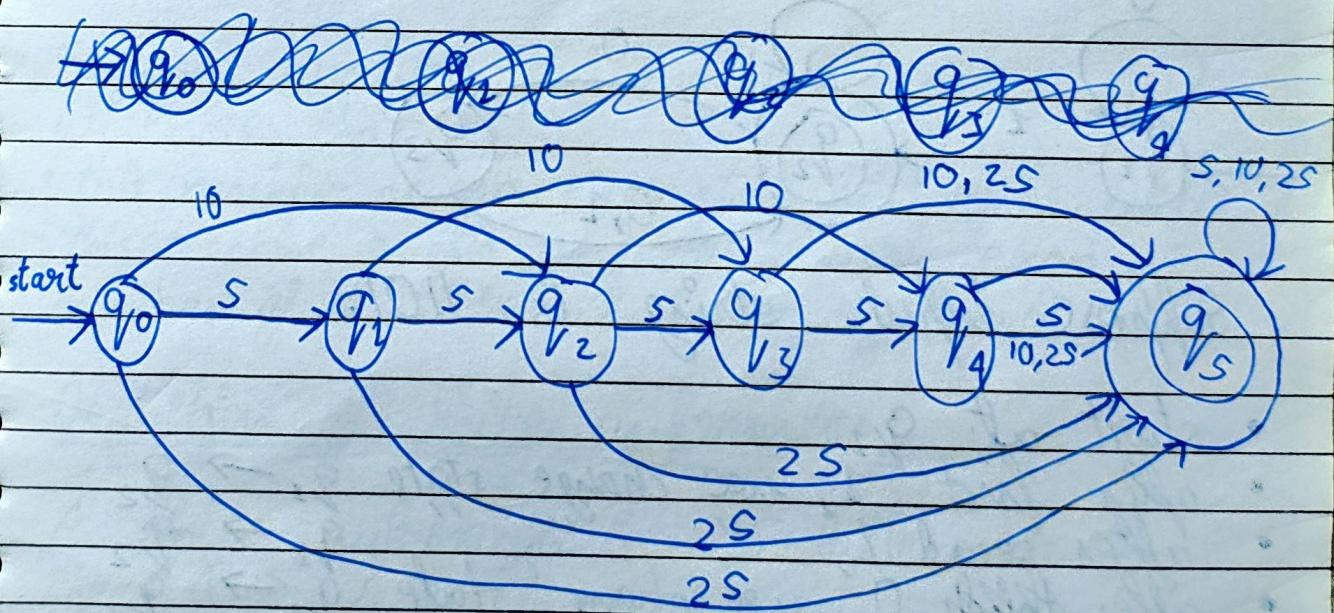
The computer will not return change.

In order to decide when to open gate, machine has to be in any of 6 states.

- machine is in state  $q_0$ , if it has collected no money.
- machine is in state  $q_1$ , if it has collected ₹ 5
- $q_2$  state when ₹ 10 collected
- $q_3$  state when ₹ 15 collected
- $q_4$  " " ₹ 20 "
- $q_5$  " " ₹ 25 collected or more.

Initially, state is  $q_0$ .

The states and transitions can be shown as follows.



The numbers above line show what coin was inserted

$q_5$  is shown with double circles because it is a special state. When it reaches state  $q_5$  it accepts payments and opens gate.

This state is called accept state.

Date \_\_\_\_\_

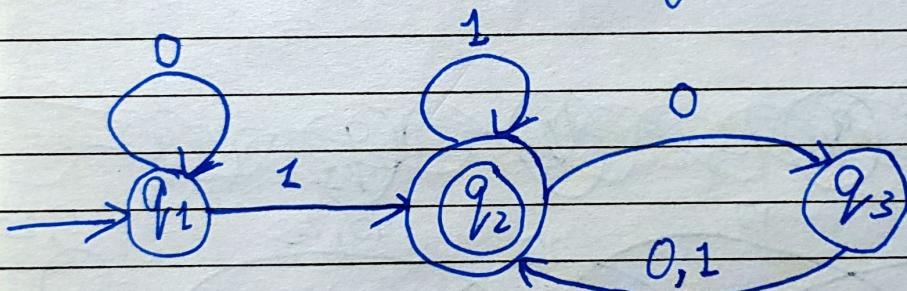
This machine only needs to remember which state it is in at any given time.

So to distinguish all 6 states, it only needs

$$\lceil \log_2 6 \rceil = 3 \text{ bits}$$

### Deterministic finite automata

Consider the state diagram



Suppose input string is 1101.

- start at  $q_1$
- after first 1, we change state  $q_1 \rightarrow q_2$
- after second 1, "
- after third 0, we change state  $q_2 \rightarrow q_3$
- finally after 1, we change  $q_3 \rightarrow q_2$

After processing 1101, we are at  $q_2$  which is accept state.

Therefore, we say string 1101 is accepted by the machine.

Suppose string 1010

- first 1 :  $q_1 \rightarrow q_2$
- second 0 :  $q_2 \rightarrow q_3$
- third 1 :  $q_3 \rightarrow q_2$
- fourth 0 :  $q_2 \rightarrow q_3$

After processing 1010, we do not end on an accept state.

We say that machine rejects string 1010.

If we analyze this machine, we will see that

- this machine accepts every string that ends with 1
- this machine accepts every string with even number of 0's at right end.

### Definition of finite automaton

A finite automaton is 5-tuple collection  
 $M = (Q, \Sigma, \delta, q_0, F)$

1.  $Q$  is a finite set of states
2.  $\Sigma$  is a finite set of symbols.  
This set is called an alphabet.
3.  $\delta: Q \times \Sigma \rightarrow Q$  is a function called transition function.

4.  $q_0$  is an element of  $Q$ ; it is the start state.
5.  $F$  is a ~~subset~~ subset of  $Q$ ; it is the set of accept states.

$\delta$  can be thought of as the program of the finite automaton. This function tells  $M$  what to do in "one step".

In our toll gate example.

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$$

$$\Sigma = \{5, 10, 25\}$$

$$\begin{aligned} q &= q_0 \\ F &= \{q_5\} \end{aligned}$$

$\delta$  is represented by a table for input and output

	5	10	25	← symbol input
$q_0$	$q_1$	$q_2$	$q_5$	
$q_1$	$q_2$	$q_3$	$q_5$	
$q_2$	$q_3$	$q_4$	$q_5$	
$q_3$	$q_4$	$q_5$	$q_5$	
$q_4$	$q_5$	$q_5$	$q_5$	
$q_5$	$q_5$	$q_5$	$q_5$	

↑  
state input

Therefore

As we can see transition function  $\delta$  has two inputs, one for current state ( $r$ ) and one for a symbol ( $a$ ).

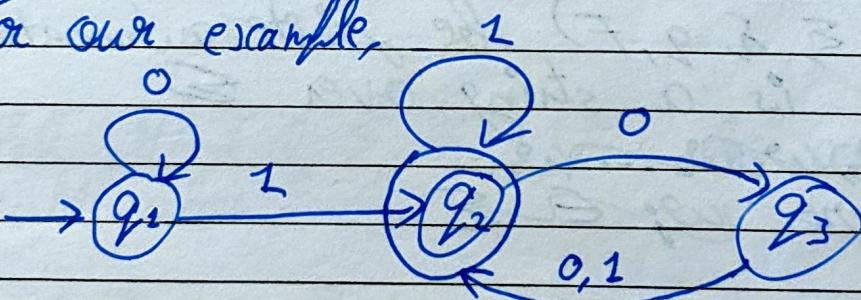
So we write transition function as

$$\delta(r, a)$$

↑
↖
symbol

current
state

For our example,



$\delta$  is:

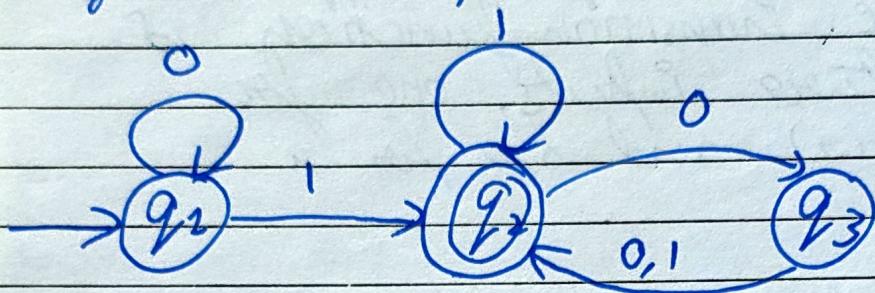
	0	1
q1	q2	q2
q2	q3	q2
q3	q2	q2

Language of a finite automaton

The language of  $M = (\mathcal{Q}, \Sigma, \delta, q_0, F)$

is set of all strings accepted by  $M$ . It is then written as  $L(M)$

so far our example machine



$L(M) = \{w : w \text{ contains at least one } 1 \text{ and ends with even no. of } 0's\}$

$\rightarrow x \rightarrow x \rightarrow x \rightarrow x \rightarrow x \rightarrow$

Let  $M = (\emptyset, \Sigma, \delta, q_0, F)$  be finite automata  
and  $w$  is a string over  $\Sigma$   
(i.e.  $w = w_0 w_1 w_2 \dots w_n$   
where  $w_i \in \Sigma$ )

i.e.  $w = w_0 w_1 w_2 \dots w_n$   
where  $w_i \in \Sigma$

Then we define sequence of  
states  $r_0, r_1, r_2 \dots r_n$  such that

\*  $r_0 = q_0$   
\*  $r_{i+1} = \delta(r_i, w_{i+1})$ , for  $i = 0, 1, \dots, n-1$

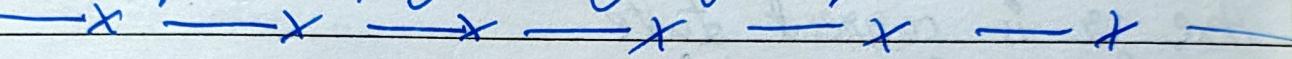
If  $r_n \in F$ , then we say  $M$  accepts  $w$   
If  $r_n \notin F$ , then we say  $M$  rejects  $w$ .

we can be an empty string. Empty string is denoted by ~~by~~ epsilon ( $\epsilon$ )

In this case, sequence  $q_0, q_1, \dots, q_n$  has length 1.

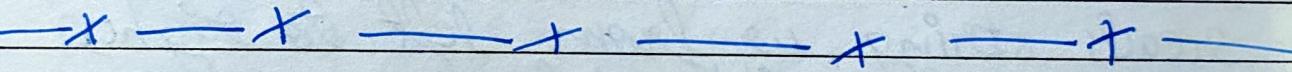
So it just has ~~an~~  $q_0 = q$ .

M accepts empty string if  $q \in F$ .



A language  $A$  is called regular if there is a finite automata

$M$  such that  $A = L(M)$



Let  $M = (Q, \Sigma, \delta, q_0, F)$  be a finite automaton.

$\delta$  tells us that when  $M$  is in state  $r \in Q$  and reads a symbol  $a \in \Sigma$ , it switches from  $r$  to  $\delta(r, a)$ .

Let  $\Sigma^*$  be set of all possible strings over  $\Sigma$  (including empty string)

Then

$$\bar{\delta}: Q \times \Sigma^* \rightarrow Q$$

This function is defined as

~~that~~

$$\delta(r,w) = \delta(\delta(r,v), a)$$

$$\bar{\delta}(r,w) = \begin{cases} r \\ \delta(\delta(r,v), a) \end{cases}; \quad w = va \text{ and } 'a' \text{ is a symbol}$$

Here,  $r$  is current state  
and  $w$  is a string

$\bar{\delta}(r,w)$  is the state which  $M$  reaches  
when it starts at state  $r$  and  
reads string  $w$  from left to right,  
using  $\delta$  to switch states.

Therefore,

$$L(M) = \{w : w \text{ is a string over } \Sigma \text{ and } \bar{\delta}(q, w) \in F\}$$

## Examples of finite automaton

1.  $A = \{w : w \text{ contains an odd number of 1's}\}$

Prove that  $A$  is a regular language.  
 $\Sigma = \{0, 1\}$

We can ~~sm~~ read w from left to right and change state every time we read a 1 in w. Since, we need odd no. of 1's every other state is accepted.

But this will lead to infinite states but finite automata can only have finite states.

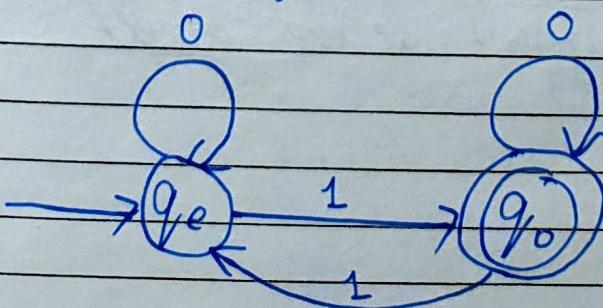
Since there is repeating pattern, we can have two states.

$q_0$ : odd no. of 1's  
 $q_e$ : even no. of 1's

Zero 1's is even, therefore initial state is  $q_e$

$\delta$ is	0	1
$q_0$	$q_0$	$q_e$
$q_e$	$q_e$	$q_0$

Finite automata is



2.  $A = \{w : w \text{ is binary string containing } 101 \text{ substring}\}$

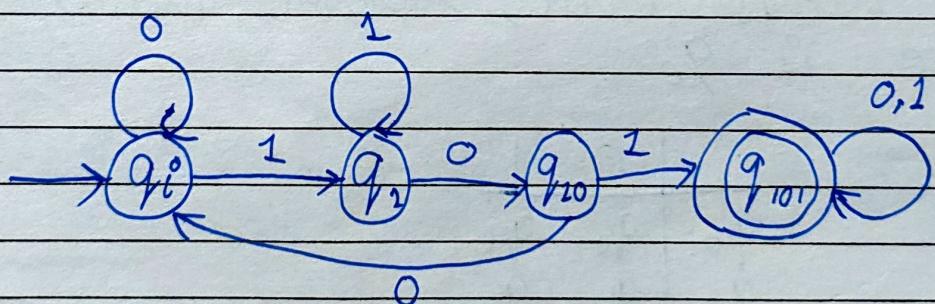
Prove that  $A$  is a regular language.

To test substring, we need to store bits as we read them in memory.

Since substring to test has three bits, we can use three states to store current substring ( $q_1, q_{10}, q_{101}$ )

We also need a initial state in which we know that there is no potential substring ( $q_i^0$ ).

The machine will be.



Once we read 101 consecutive in a string, we will enter accept state after which we won't leave accept state.

$\delta =$	0	1
$q_i^0$	$q_i^0$	$q_1$
$q_1$	$q_{10}$	$q_1$
$q_{10}$	$q_i^0$	$q_{101}$
$q_{101}$	$q_{101}$	$q_{101}$

3. A = {w : w has 1 in third position from right}

Show that A is a regular language.

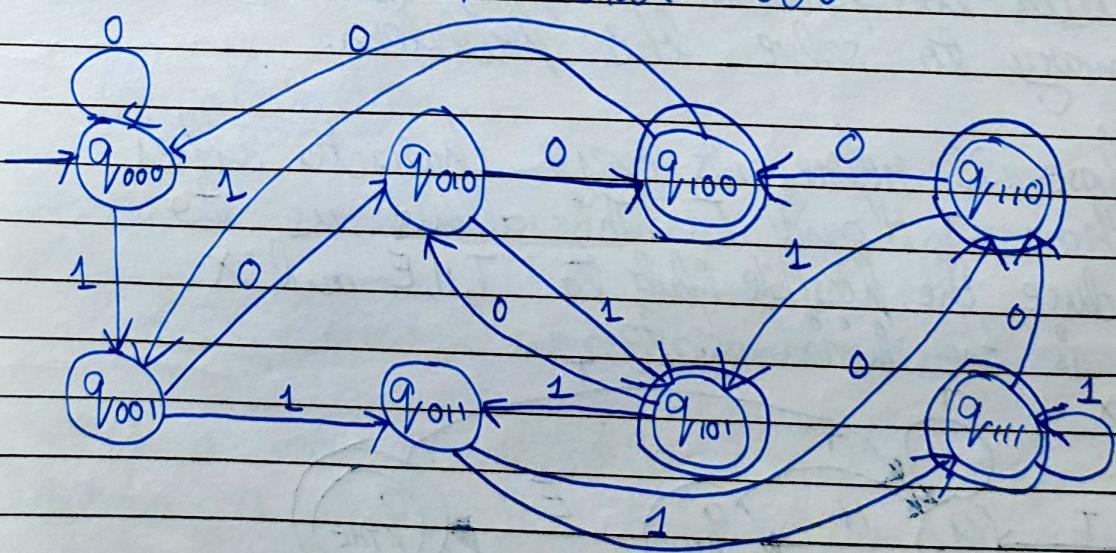
To make machine for this, we will need memory to store the last three bits that the machine has read.

But unlike last example, we can't predict when we have reached final accept state, so we need larger memory.

For three bits we will have states

$q_{000}, q_{001}, q_{010}, q_{011}, q_{100}, q_{101}, q_{110}, q_{111}$

Since we only worry about 1 bit, we can consider initial state 000



We consider initial state ~~q\_{000}~~  $q_{000}$  to avoid errors in case string length is less than 3.

By same logic, initial state will be  $q_{111}$  if we want M for 0 in third position from right.

$\delta =$	0	1
$q_{000}$	$q_{000}$	$q_{001}$
$q_{001}$	$q_{010}$	$q_{011}$
$q_{010}$	$q_{100}$	$q_{101}$
$q_{011}$	$q_{110}$	$q_{111}$
$q_{100}$	$q_{000}$	$q_{001}$
$q_{101}$	$q_{010}$	$q_{011}$
$q_{110}$	$q_{100}$	$q_{101}$
$q_{111}$	$q_{110}$	$q_{111}$

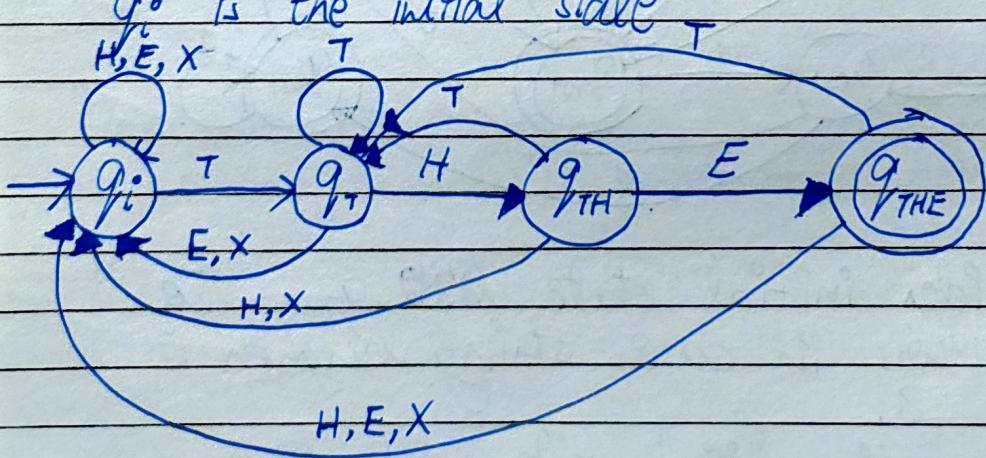
4. A =  $\{w : w \text{ does not end with } THE\}$

$$\Sigma = \{A, B, C, \dots, Z\}$$

Similar to previous example, we need memory to solve this problem.

Suppose  $X$  represents every character other than T, H and E. This allows us to reduce the possible inputs to T, H, E and X.

$q_0$  is the initial state



$s =$	T	H	E	X
$q_i$	$q_T$	$q_i$	$q_i$	$q_i$
$q_T$	$q_T$	$q_{TH}$	$q_i$	$q_i$
$q_{TH}$	$q_T$	$q_i$	$q_{THE}$	$q_i$
$q_{THE}$	$q_T$	$q_i$	$q_i$	$q_i$

## Regular operations

We define three operations of on languages.

### 1. Union

The union of language A and B is

$$\text{A} \cup \text{B} = \{w : w \in \text{A} \text{ or } w \in \text{B}\}$$

### 2. Concatenation

The concatenation of two languages A and B is

$$AB = \{ww' : w \in \text{A} \text{ and } w' \in \text{B}\}$$

AB is a set of all strings obtained by taking an arbitrary string from A and concatenating a string from set B.

$$A = \emptyset, 01, 102$$

$$B = \emptyset, 00, 112$$

Then we can find AB as

$AB =$	$\begin{array}{c cc c} & 00 & 11 \\ \hline 01 & 0100 & 1101 \\ 10 & 1000 & 1110 \end{array}$
--------	--

i.e.  $AB = \{0100, 1101, 1000, 1110\}$

Note that  $w \in A$  is always left and  $w' \in B$  is always at right.  
This operation is like cross product.

### 3. Star

The star of language A ( $A^*$ ) is defined as

$A^* = \{u_1 u_2 \dots u_k : k \geq 0 \text{ and } u_i \in A \text{ for all } i=1,2,\dots,k\}$

$A^*$  is obtained by taking  $k$  strings from set A and concatenating them.

A single string can be chosen multiple times and  $k$  can be anywhere from 0 to  $\infty$ .

Therefore  $A^*$  gives an infinite set.

In fact,

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

Then

(~~any string is a bit string~~)

$A^*$  is a set of all binary strings including  $\epsilon$  (when  $k=0$ ).

The only time  $A^*$  is not infinite is when

~~$A \in \emptyset \cup \{\epsilon\}$~~

$A = \emptyset$  in which case  $A^* = \emptyset \cup \{\epsilon\}$

$\epsilon$  is empty string

$\epsilon \in A^*$  for every  $A^*$ .

$-x - x - x - x - x - x -$

Another definition of  $A^*$  can be through concatenation.

For a language  $A$

$$A^0 = \emptyset \cup \{\epsilon\}$$

$$A^1 = A$$

$$A^2 = AA$$

$$A^3 = AAA$$

$$\vdots \quad \vdots$$

$$A^k = A A^{k-1}$$

So  $A^k$  is  $A$  concatenated with itself  $k$  times  
then

$$A^* = \bigcup_{k=0}^{\infty} A^k$$

$$A^* = A^0 U A^1 U A^2 U \dots U A^{\infty}$$

Set of regular languages is closed under Union

i.e if A is a regular language  
and B is a regular language.

Then A ∪ B is a regular language.

Suppose,

$$A = \mathcal{L}(M_1) ; M_1 = (\mathbb{Q}_1, \Sigma, \delta_1, q_1, F_1)$$

$$B = \mathcal{L}(M_2) ; M_2 = (\mathbb{Q}_2, \Sigma, \delta_2, q_2, F_2)$$

Then

$$A \cup B = \mathcal{L}(M) ; M = (\mathbb{Q}, \Sigma, \delta, q, F)$$

To define M, we will need a way to run both machines  $M_1$  and  $M_2$  simultaneously.

For this, we will arrange states of  $M_1$  &  $M_2$  in ordered pair using cartesian product.

$$\mathbb{Q} = \mathbb{Q}_1 \times \mathbb{Q}_2 = \{(q_1, q_2) : q_1 \in \mathbb{Q}_1 \text{ and } q_2 \in \mathbb{Q}_2\}$$

Then we can define  $\delta$  as

$$\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$$

for all  $q_1 \in \mathbb{Q}_1$ ,  $q_2 \in \mathbb{Q}_2$  and  $a \in \Sigma$

Therefore, for

$$A = \mathcal{L}(M_1) ; M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$$

$$B = \mathcal{L}(M_2) ; M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$$

There is machine M such that

$$A \cup B = \mathcal{L}(M)$$

and

~~M = (Q1 X Q2, Σ, δ1(q1, a), δ2(q2, a), (q1, q2), F1 ∪ F2)~~

$$M = (Q_1 \times Q_2, \Sigma, (\delta_1(q_1, a), \delta_2(q_2, a)), (q_1, q_2), F_1 \times F_2)$$

— x — x — x — x —

The concat and \* operators are also same. So if A and B are regular languages.

Then, AB is also regular language and  $A^*$  is also regular language.

But having this in a finite automata we studied so far is hard. We will need another type of finite automaton

— x — x — x — x —

Finite automata studied so far have deterministic state transitions for each alphabet. They are called Deterministic Finite Automaton (DFA)

## Non-deterministic Finite Automaton

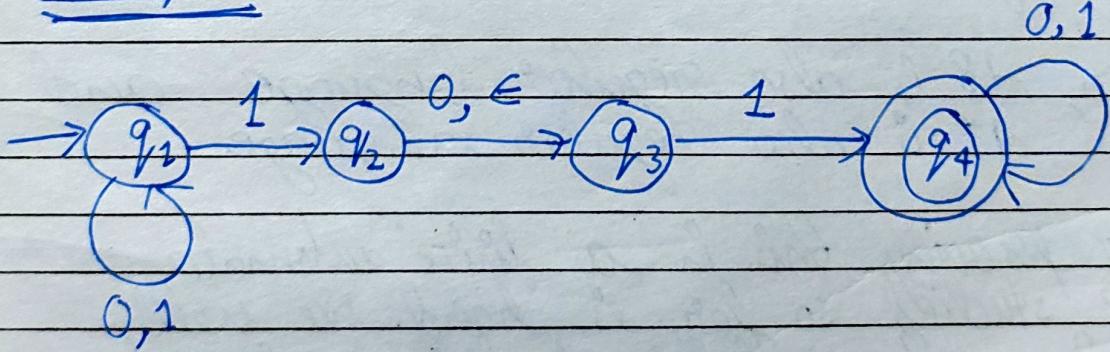
A NFA (Nondeterministic Finite Automaton) has two major additions

i) The machine can transition to any state from a set of states on reading a symbol.

i.e., same symbol can cause different state transitions

ii) There is an additional transition  $\epsilon$ , the machine can do this transition without reading a symbol.

### Example



- We can see that in state  $q_2$ , on reading 1, either machine will stay in  $q_2$ , or it will transition to  $q_3$ .
- Machine can transition from  $q_2$  to  $q_3$  without reading a symbol.

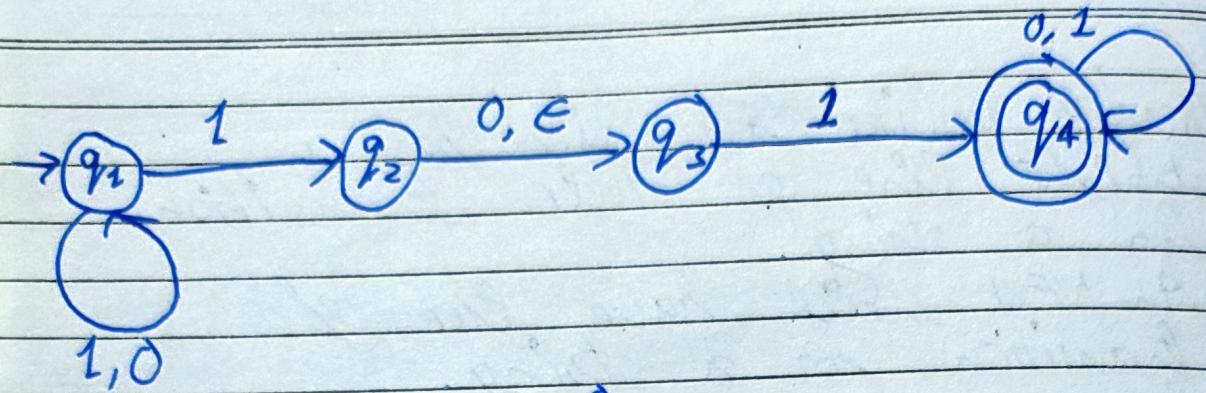
Date \_\_\_\_\_

Therefore, unlike DFA which has a straight line of state transitions for a string.

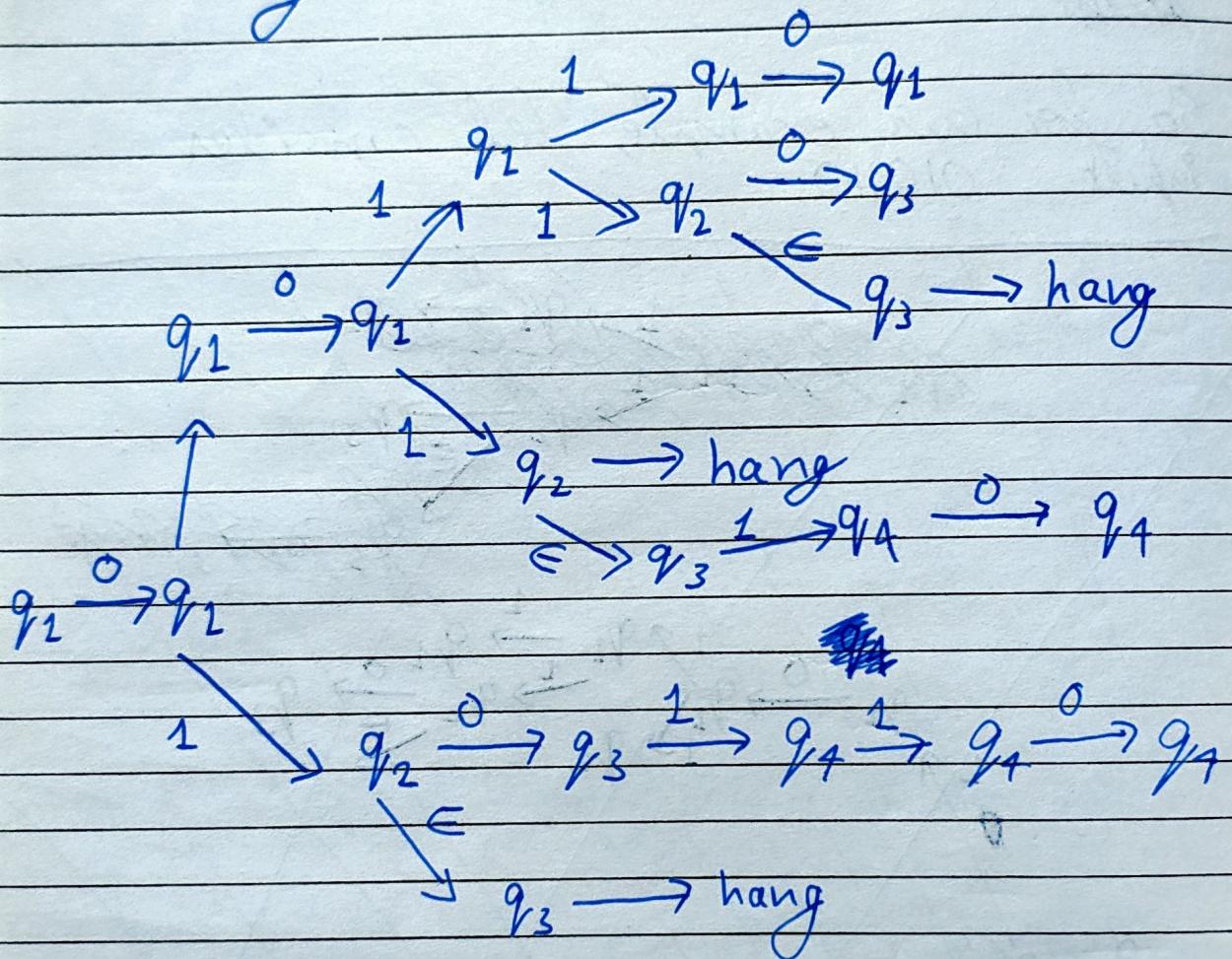
An NFA will have tree of transitions for a string.

Now

So for our example, let's consider input 010110.



For string 010110,



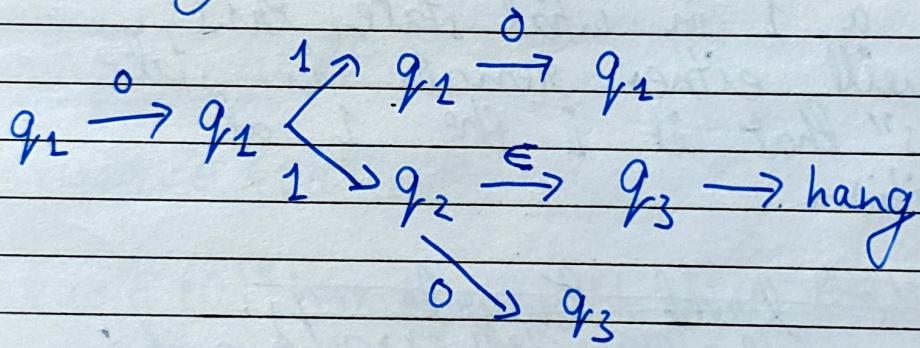
Two branches reach  $q_4$  at end.

If atleast one branch is ending at an accept state, we say that string is accepted by NFA

We can also see that some branches can't process the whole string because they get stuck on a state. We say that NFA hanged in this cases.

That is, NFA hangs when no further transitions are possible and symbols are remaining in string.

For string 010,



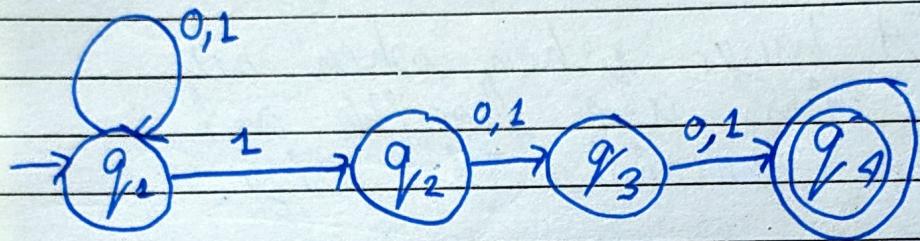
None of the branches lead to end at accept state.

Therefore 010 is rejected by NFA

## Second Example,

$A = \{w : w \text{ has } 1 \text{ in third position from right}\}$

The machine for this in DFA very complex.  
In comparison NFA is very simple.



On reading a 1 in initial state, this machine will either remain in state 1 or "guess" that it is the 1 at third position.

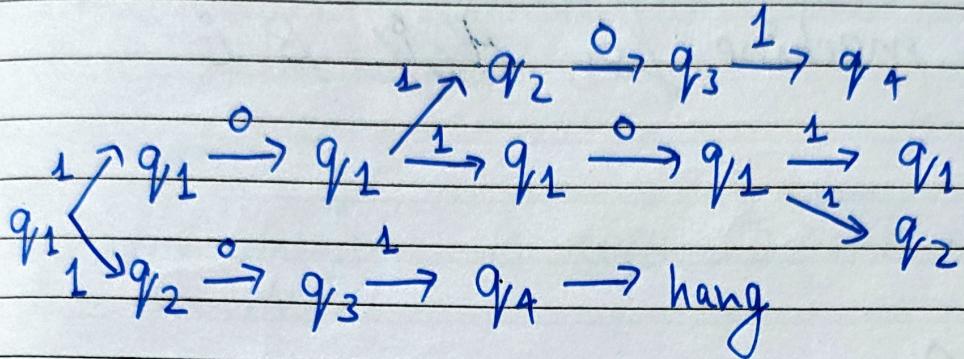
It will then travel towards  $q_4$ .

~~If it was not the 3rd symbol, string and rejects~~

After it guesses that it's third position '1'  
it will move towards  $q_4$ .

- if it was 1 in second or first position from right, string will be rejected as accept was at  $q_4$
- if there are more than three symbols after 1, machine will hang.

Example for 10101



A branch reaches  $q_4$ , thus 10101 is accepted.

### Third Example

~~A = B ∪ C ∪ D ∪ E ∪ F ∪ G ∪ H~~

$A = \{w \in \{0,1\}^*: w \text{ has multiple of 2 } \cancel{\text{or 3}} \text{ symbols}$   
 $\text{or } w \text{ has multiple of 3 symbols}\}$

This is a language made from union of two other languages

$B = \{w : w \text{ has multiple of 2 symbols}\}$

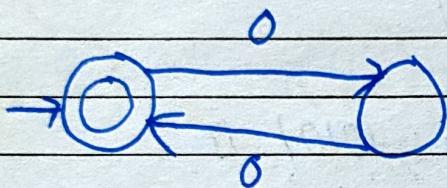
$C = \{w : w \text{ has multiple of 3 symbols}\}$

$A = B \cup C$

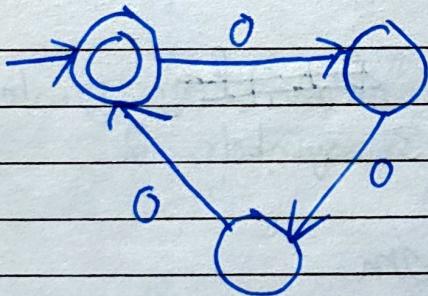
A DFA for both languages B and C is easy, but to build machine for A, we need to run them simultaneously in PFA as seen.

In ~~NFA~~ NFA, designing the machine is much easier.

- The ~~mach~~ machine for ~~A B~~ B is



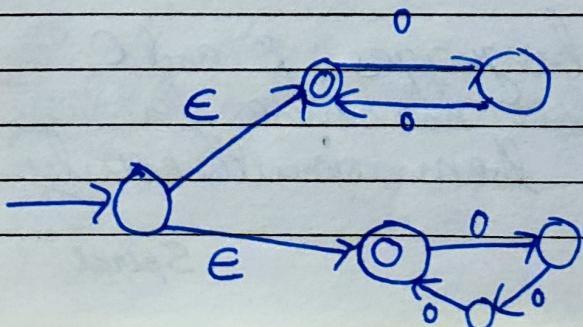
- and machine for ~~A C~~ C is



So machine for

$$A = B \cup C$$

~~mach~~ in NFA is



i.e. at ~~start~~ start, machine will guess whether it wants to check for B or check for C.

So, the union is easy to show in an NFA

### Formal Definition for NFA

A NFA is defined as 5-tuple

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

- $\mathcal{Q}$  is the set of states
- $\Sigma$  is the alphabet and elements of  $\Sigma$  are called symbols
- $\delta: \mathcal{Q} \times \Sigma \rightarrow P(\mathcal{Q})$  is the transition function

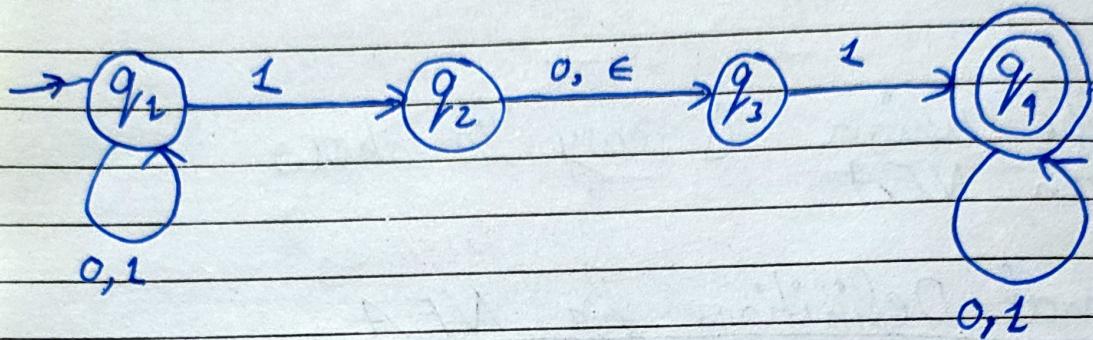
$$\Sigma^* = \Sigma \cup \{\epsilon\}$$

and  $P(\mathcal{Q})$  is powerset of  $\mathcal{Q}$ .

So  $\delta(q, a)$  takes current state, a symbol (including  $\epsilon$ ) and returns a set of next possible states.

- $q_0$  is the start state of machine
- $F$  is the set of accept states.

Let's consider NFA



For this machine

$$\begin{aligned} Q &= \{q_1, q_2, q_3, q_4\} \\ \Sigma &= \{0, 1, \epsilon\} \end{aligned}$$

$\delta =$	0	1	$\epsilon$
$q_1$	$\{q_1\}$	$\{q_2, q_3\}$	$\emptyset$
$q_2$	$\{q_3\}$	$\emptyset$	$\{q_3\}$
$q_3$	$\emptyset$	$\{q_4\}$	$\emptyset$
$q_4$	$\{q_4\}$	$\{q_4\}$	$\emptyset$

~~for the next part~~

$$q = q_1$$

$$F = \{q_4\}$$

## Equivalence of DFAs and NFAs

If there is a ~~valid~~ NFA called  $N$ , which accepts language  $L(N)$ , then there also exists a DFA called  $D$  which ~~accepts language~~ has language  $L(N)$ .

Similarly, if there is a DFA called  $D$  with language  $L(D)$ , there also exists an NFA with language  $L(D)$ .

We can convert a NFA to DFA and vice versa, with both having same behaviour.

This is called equivalence of DFA and NFA.

### Converting DFA to NFA

~~Suppose there is~~

The DFA is  $D = (\emptyset, \Sigma, S, q_f, F)$   
its equivalent NFA is

$N = (\emptyset, \Sigma, S', q_f, F)$  i.e. only  $S$  is changed

$$\delta'(q, a) = \begin{cases} \{\delta(q, a)\}, & \text{if } a \neq \epsilon \\ \emptyset, & \text{if } a = \epsilon \end{cases}$$

## Converting NFA To DFA

For a NFA,

~~REMARKS,~~

$$N = (\mathcal{Q}, \Sigma, \delta, q_0, F)$$

The equivalent DFA is

$$D = (\mathcal{Q}', \Sigma, \delta', q_0', F')$$

More is changed when converting from NFA to DFA than vice-versa.

The changes are made to simulate the many many possible transitions of an NFA in a DFA.

i)  $\mathcal{Q}' = P(\mathcal{Q})$

$\mathcal{Q}'$  is a powerset of  $\mathcal{Q}$ .

So the DFA will have set of states of NFA as its own states.

ii)  $S'(\mathcal{Q}, a) = \bigcup_{q \in \mathcal{Q}} C(S(q, a))$

iii)  $S'(\mathcal{Q}, a) = \bigcup_{q \in \mathcal{Q}} [C(S(q, a))]$

## $\epsilon$ -closure

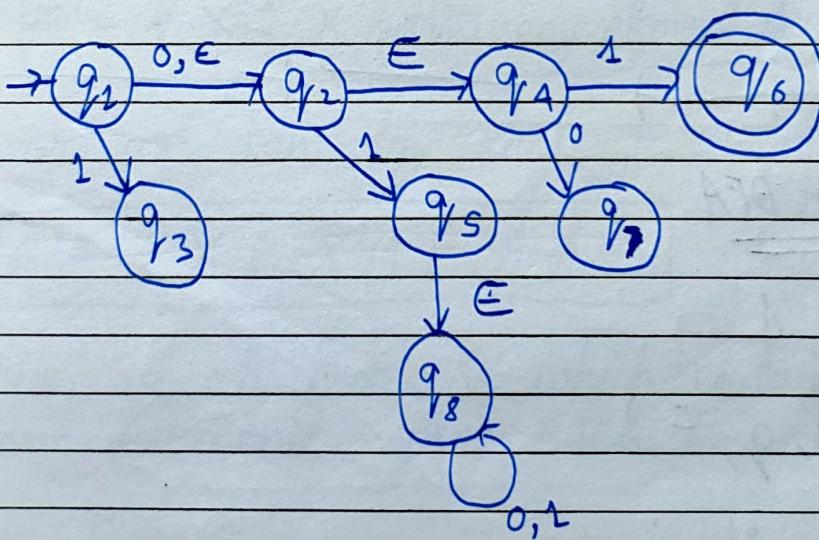
We will define  $\epsilon$  closure as

$C_{\epsilon}(R)$  as the set of all states we can reach using one or more  $\epsilon$  transitions from set of states  $R$ .

$R$  contains states of NFA, (i.e  $R \subseteq Q$ )

$\epsilon$  closure is used to get  $q'$  and  $S'$  when converting from NFA to DFA.

Example for NFA



Suppose  $R = \{q_1, q_5\}$

- States we can go from  $q_1$  by zero or more  $\epsilon$  transitions are  $\{q_1, q_2, q_4, q_6\}$
- States we can go from  $q_5$  from zero or more  $\epsilon$  transitions are  $\{q_5, q_8\}$

Therefore,

$$C_{\epsilon}(\epsilon_{q_1, q_5}) = \{q_1, q_2, q_4\} \cup \{q_5, q_8\}$$

$$C_{\epsilon}(\epsilon_{q_2, q_5}) = \{q_2, q_3, q_4, q_5, q_6\}$$

— X — X — X — X —

Converting NFA to DFA

~~Suppose the DFA is~~

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

Then the NFA

— X — X — X —

Converting NFA to DFA

~~Suppose the NFA is~~

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

Then the DFA for it is

~~Q, Q', Q'', Q''', Q''''~~

$$D = (Q', \Sigma, \delta', q', F')$$

i) For  $\mathcal{Q}'$

$$\mathcal{Q}' = P(\mathcal{Q})$$

$\mathcal{Q}'$  is the powerset of  $\mathcal{Q}$

ii) For  $\mathcal{q}'$

$$\mathcal{q}' = C_e(\{\mathcal{q}\})$$

$\mathcal{q}'$  is the  $E$ -closure of set  $\{\mathcal{q}\}$

iii) For  $F'$

$$F' \subset P(\mathcal{Q})$$

$$F' = \{X : X \text{ contains any element of } F\}$$

or we can say that

$$F' = \{X : X \cap F \neq \emptyset\}$$

i.e., all sets from  $\mathcal{Q}'$  which contain atleast one element from set  $F$ .

~~For  $\mathcal{R}$~~

~~$\mathcal{S}(\mathcal{R})$~~

~~$\mathcal{F}(\mathcal{R}) = \{x_1, x_2, x_3, \dots, x_n\}$~~

~~$S'(\mathcal{R}) = S(x_1)$~~

iv) ~~For  $\delta'$~~

~~For any  $R = \{r_1, r_2, r_3, \dots, r_n\}$~~

~~$$\delta'(R, a) = C_e(\delta(r_1, a)) \cup C_e(\delta(r_2, a)) \cup \dots \cup C_e(\delta(r_n, a))$$~~

v) For  $\delta'$

For any  $R = \{r_1, r_2, r_3, \dots, r_n\}$  state of DFA

$$\delta'(R, a) = C_e(\delta(r_1, a)) \cup C_e(\delta(r_2, a)) \cup \dots \cup C_e(\delta(r_n, a))$$

" i.e,

$$\delta'(R, a) = \bigcup_{r \in R} C_e(\delta(r, a))$$

Since  $\emptyset$  has no elements,

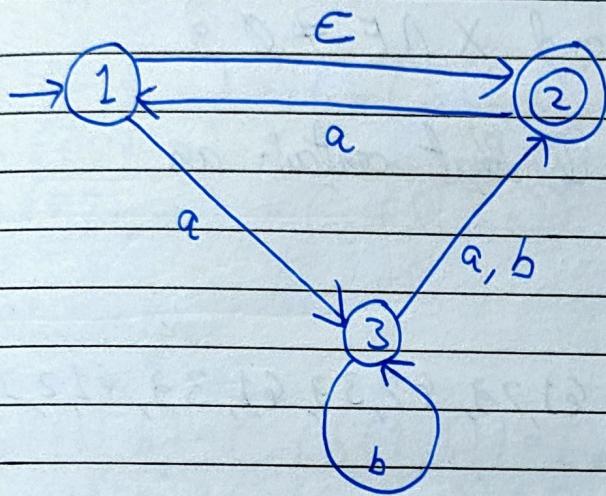
$$\delta'(\emptyset, a) = \emptyset$$

\*  $\delta'(A, a) \cup \delta'(B, a) = \delta'(A \cup B, a)$

This property can help get  $\delta'$  transitions faster for all states

## Example of converting NFA to DFA

NFA is  $(Q, \Sigma, \delta, q_1, F)$



$$Q = \{1, 2, 3\}$$

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

$$q_1 = 1$$

$$F = \{2\}$$

$S =$	$a$	$b$	$\epsilon$
1	$\{3\}$	$\emptyset$	$\{2\}$
2	$\{1\}$	$\emptyset$	$\emptyset$
3	$\{2\}$	$\{2, 3\}$	$\emptyset$

The DFA will be  $(Q', \Sigma, \delta', q'_1, F')$

i)  $Q' = P(Q)$

$$\begin{aligned}
 Q' = & \{ \{1\}, \{2\} \\
 & \{3\}, \{1, 2\} \\
 & \{2, 3\}, \{1, 3\} \\
 & \{1, 2, 3\}, \emptyset
 \end{aligned}$$

3

$$\text{ii)} \quad q' = C_e(\{q\}) = C_e(\{1\})$$

$$q' = \{1, 2\}$$

$$\text{iii)} \quad F' = \{X : X \in Q' \text{ and } X \cap F \neq \emptyset\}$$

so all elements of  $Q'$  that contain an element from  $F$

$$F = \{2\}$$

$$Q' = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{2, 3\}, \{1, 3\}, \{1, 2, 3\}\}$$

Therefore

$$F' = \{\{2\}, \{1, 2\}, \{2, 3\}, \{1, 3\}, \{1, 2, 3\}\}$$

iv) For finding  $\delta'$

We need to get the complete table

	a	b
$\emptyset$		
$\{1\}$		
$\{2\}$		
$\{3\}$		
$\{1, 2\}$		
$\{2, 3\}$		
$\{1, 3\}$		
$\{1, 2, 3\}$		

\*  $\delta'(\emptyset, a) = \emptyset$  and  $\delta'(\emptyset, b) = \emptyset$

\* Now we can get  $\delta'$  for all states having single element

$$\begin{aligned}\delta'(\{1\}, a) &= C_e(\delta(1, a)) = C_e(\{3\}) = \{3\} \\ \delta'(\{2\}, a) &= C_e(\delta(2, a)) = C_e(\{1\}) = \{1\} \\ \delta'(\{3\}, a) &= C_e(\delta(3, a)) = C_e(\{2\}) = \{2\}\end{aligned}$$

Similarly,

$$\begin{aligned}\delta'(\{1\}, b) &= C_e(\delta(1, b)) = C_e(\emptyset) = \emptyset \\ \delta'(\{2\}, b) &= C_e(\delta(2, b)) = C_e(\emptyset) = \emptyset \\ \delta'(\{3\}, b) &= C_e(\delta(3, b)) = C_e(\{2, 3\}) = \{2, 3\}\end{aligned}$$

\* Now we can use the property,

$$\delta'(A \cup B, x) = \delta'(A, x) \cup \delta'(B, x)$$

So to get,

$$\delta'(\{1, 2\}, a), \text{ we can get}$$

$$\delta'(\{1, 2\}, a) = \delta'(\{1\}, a) \cup \delta'(\{2\}, a)$$

$$\delta'(\{1, 2\}, a) = \{3\} \cup \{1\}$$

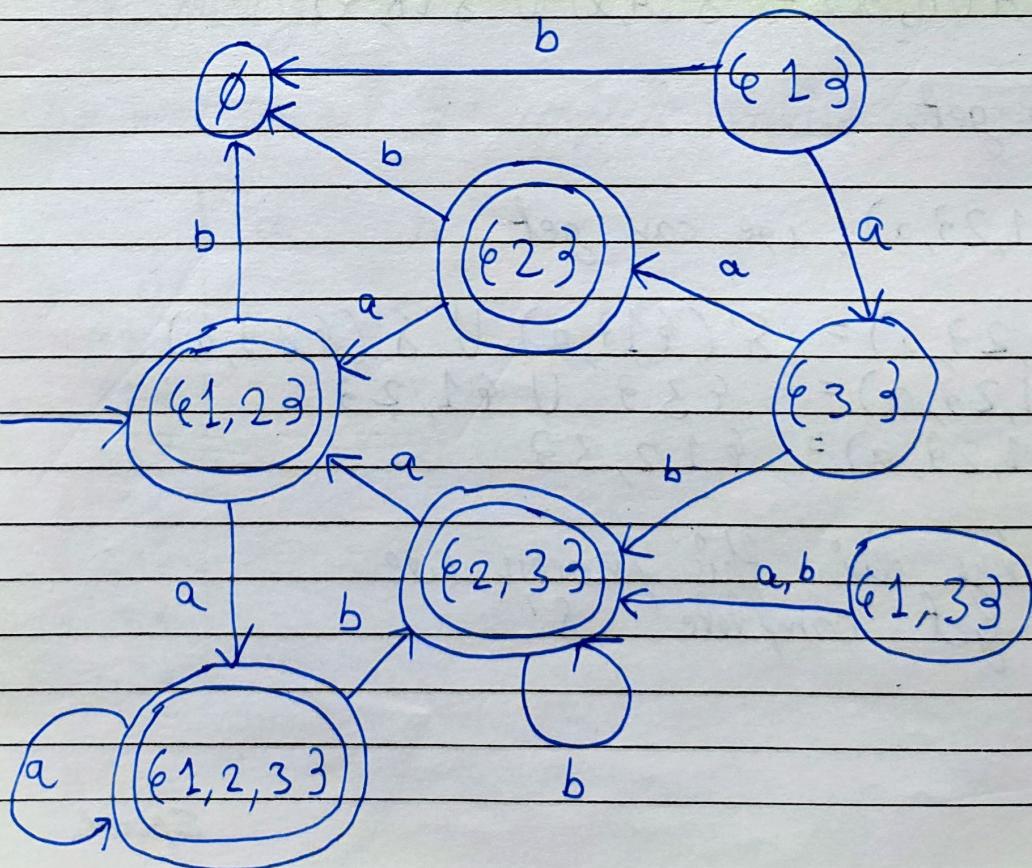
$$\delta'(\{1, 2\}, a) = \{1, 2, 3\}$$

Now by doing this process we can get complete  $\delta'$

$\delta'$  is

	a	b
$\emptyset$	$\emptyset$	$\emptyset$
$\{\epsilon_1\}$	$\{\epsilon_3\}$	$\emptyset$
$\{\epsilon_2\}$	$\{\epsilon_1, \epsilon_2\}$	$\emptyset$
$\{\epsilon_3\}$	$\{\epsilon_2\}$	$\{\epsilon_2, \epsilon_3\}$
$\{\epsilon_1, \epsilon_2\}$	$\{\epsilon_1, \epsilon_2, \epsilon_3\}$	$\emptyset$
$\{\epsilon_2, \epsilon_3\}$	$\{\epsilon_1, \epsilon_2\}$	$\{\epsilon_2, \epsilon_3\}$
$\{\epsilon_1, \epsilon_3\}$	$\{\epsilon_2, \epsilon_3\}$	$\{\epsilon_2, \epsilon_3\}$
$\{\epsilon_1, \epsilon_2, \epsilon_3\}$	$\{\epsilon_1, \epsilon_2, \epsilon_3\}$	$\{\epsilon_2, \epsilon_3\}$

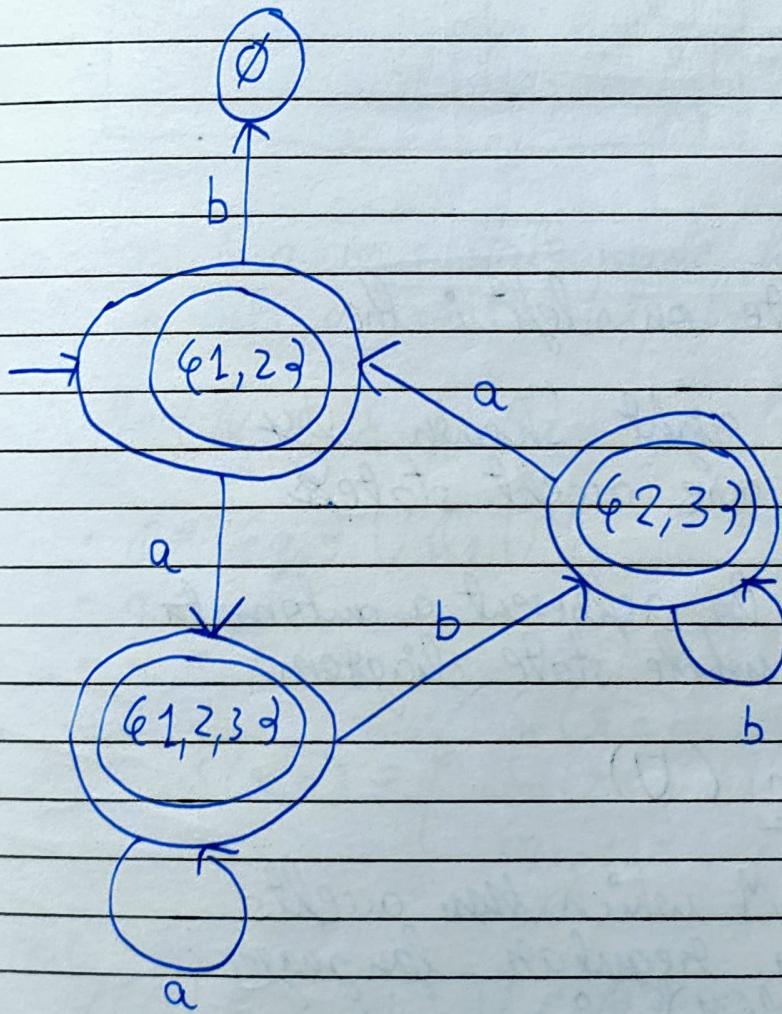
So we get the DFA,  
state diagram for this DFA is



This DFA works perfectly, but we can minimize the number of states.

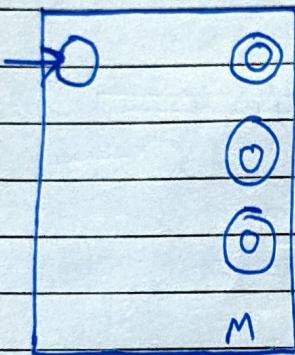
- States  $\emptyset$  and  $\{1, 3\}$  have no incoming transitions so they can be removed
- After previous step,  $\{3\}$  has no incoming transition so it is also removed
- After previous steps,  $\{2\}$  has no incoming transitions, so it also removed

~~Final State~~ The minimized DFA is



## Closure of regular languages under regular operations

Suppose we represent a finite automata  
~~as~~  $M$  as



- The single state on left is the start state
- and states on right shown by double circles are accept states

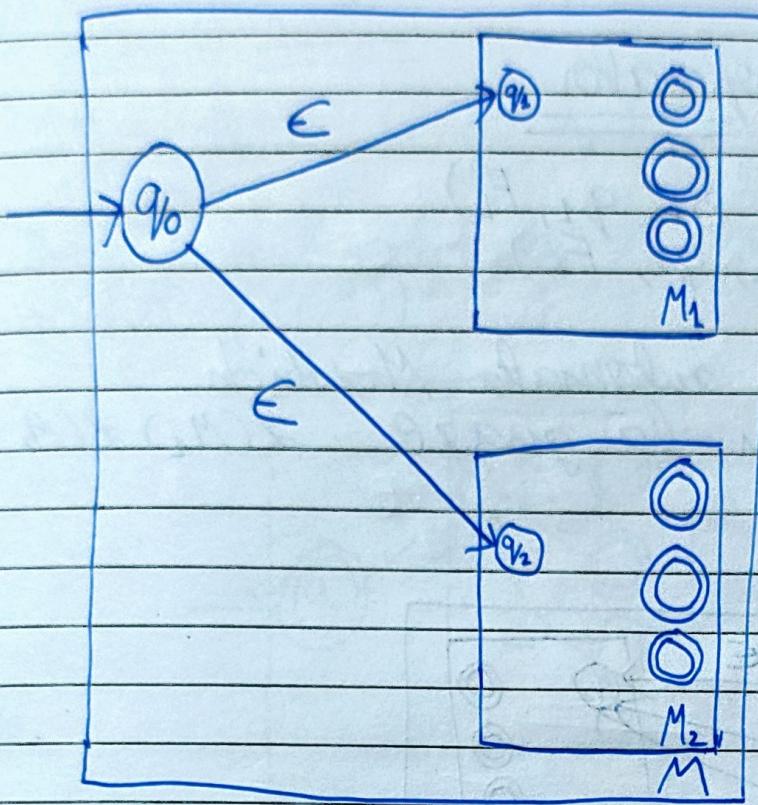
This allows us to represent a automata without drawing whole state diagram.

### i) Union operator (U)

The machine  $M$  which accepts union of two regular languages  $L(M_1)$  and  $L(M_2)$  is.

$$M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$$

$$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$$



$q_0$  is a new state made for machine  $M$   
 $q_0$  acts as start state for  $M$ .

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

- $Q = \{q_0\} \cup Q_1 \cup Q_2$

- $q_0 = q_0$

- $F = F_1 \cup F_2$

- $\delta(r, a) = \begin{cases} \delta_1(r, a) & \text{if } r \in Q_1 \\ \delta_2(r, a) & \text{if } r \in Q_2 \\ \{q_1, q_2\} & \text{if } r = q_0 \text{ & } a = \epsilon \\ \emptyset & \text{if } r = q_0 \text{ & } a \neq \epsilon \end{cases}$

Here,

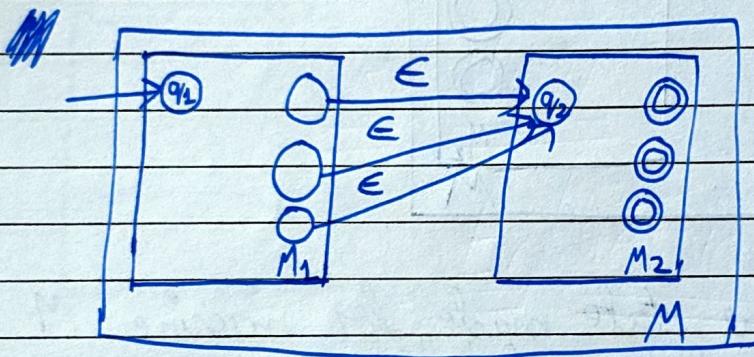
$$\mathcal{L}(M) = \mathcal{L}(M_1) \cup \mathcal{L}(M_2)$$

## ii) Concatenation operator

$$M_1 = (\mathcal{Q}_1, \Sigma, \delta_1, q_1, F_1)$$

$$M_2 = (\mathcal{Q}_2, \Sigma, \delta_2, q_2, F_2)$$

The the finite automata  $M$  which accept regular language  $L(M_1)L(M_2)$  is,



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

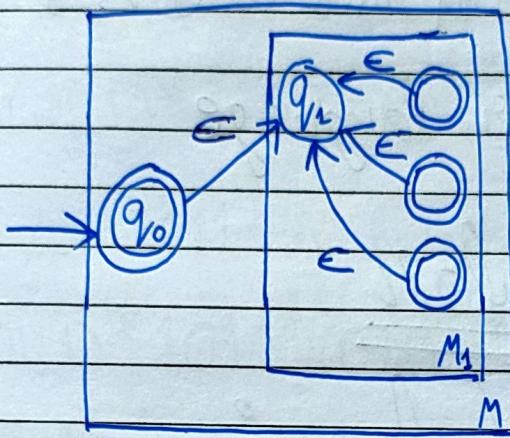
- $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$
- $q = q_1$
- $F = F_2$  (only accept states of  $F_2$ )
- $\delta(r, a) = \begin{cases} \delta_1(r, a); & \text{if } r \in \mathcal{Q}_1 \text{ and } a \neq \epsilon \\ \delta_1(r, a) \cup \{q_2\}; & \text{if } r \in \mathcal{Q}_1 \text{ and } a \in F_1 \\ \delta_2(r, a); & \text{if } r \in F_1 \text{ and } a = \epsilon \end{cases}$

$$L(M) = L(M_1)L(M_2)$$

iii) Star operator

$$M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$$

The finite automata  $M$  which accepts  $(\mathcal{L}(M_1))^*$  is,



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

- $Q = Q_1 \cup \{q_0\}$
- $q = q_0$
- $F = F_1 \cup \{q_0\}$
- $\delta(r, a) = \begin{cases} \delta_1(r, a) & ; \text{ if } r \in Q_1 \text{ and } a \notin E \\ \delta_1(r, a) & ; \text{ if } r \in F_1 \text{ and } a \neq \epsilon \\ \delta_1(r, a) \cup \{q_0\} & ; \text{ if } r \in F_1 \text{ and } a = \epsilon \\ \{q_1\} & ; \text{ if } r = q_0 \text{ and } a = \epsilon \\ \emptyset & ; \text{ if } r = q_0 \text{ and } a \neq \epsilon \end{cases}$

iv) Complement

If  $A$  is regular language over alphabet  $\Sigma$ , then

$\bar{A} = \{w : w \in \Sigma^* \text{ and } w \notin A\}$  is also a regular language

## v) Intersection

If  $A_1$  and  $A_2$  are two regular languages over alphabet  $\Sigma$ , then

$$A_1 \cap A_2 = \{ w : w \in A_1 \text{ and } w \in A_2 \}$$

is also a regular language

Regular

Expressions

Regular expressions are means to describe languages.

The class of languages described by regular expressions are - the class of regular languages.

A regular expression is

$$(0 \cup 1) 01^*$$

- This regular expression describes the language
- start with either 0 or 1. Indicated by  $(0 \cup 1)$
- has 0 as second symbol. Indicated by 0
- end with zero or more 1s. Indicated by  $1^*$

Here are a few more examples

i)  $\{w : w \text{ contains exactly two zeros}\}$

$$1^* 0 1^* 0 1^*$$

ii)  $\{w : w \text{ contains atleast two } 0\}$

$$(01)^* 0 (01)^* 0 (01)^*$$

iii)  $\{w : \text{1011 is substring of } w\}$

$$(01)^* 1011 (01)^*$$

iv)  $\{w : \text{length of } w \text{ is even}\}$

$$(01)(01)^*$$

v)  $\{w : \text{length of } w \text{ is odd}\}$

$$(01)((01)(01))^*$$

vi)  $\{w : \text{first and last symbols of } w \text{ are equal}\}$

vii)  $\{w : w \text{ is } 1011, 0\}$

$$1011 \cup 0$$

Note: + is also used instead of  $\cup$  in some places

# Properties of regular expressions

~~For any alphabet  $\Sigma$~~

1.  $\epsilon$  is a regular expression
2.  $\emptyset$  is a regular expression
3. For all  $a \in \Sigma$ ,  $a$  is a regular expression.

Rules that can be used to combine regular expressions are

1. If  $R_1$  and  $R_2$  are regular expressions, then  $R_1 \cup R_2$  is also a regular expression.
2. If  $R_1$  and  $R_2$  are regular expressions, then  $R_1 R_2$  is also a regular expression. (this is simple concatenation)
3. If  $R$  is a regular expression, then  $R^*$  is a regular expression.

What is  $\emptyset$  and  $\epsilon$ ?

~~$\emptyset$  is used to show a regular expression with no accepted language,~~

~~whereas  $\epsilon$  shows a regular expression with one accepted language,  
i.e.,  $\epsilon \in \epsilon$~~

i.e  
 Language of regular expression  $\emptyset = \emptyset$   
 Language of regular expression  $\epsilon = \{\epsilon\}$

## Language described by regular expressions

For regular expression  $R_1$  describing language  $L_1$ . And  $R_2$  describing  $L_2$ ,

Regular Expression	Language
$\emptyset$	$\emptyset$
$\epsilon$	$\{\epsilon\}$
$(\text{for some } a \in \Sigma); a$	$\{\epsilon, a\}$
$R_1 \cup R_2$	$L_1 \cup L_2$
$R_1 R_2$	$L_1 L_2$
$R^*$	<del><math>L^*</math></del>

Example, suppose regular expression is

$(0 \cup \epsilon)(1 \cup \epsilon)$

We know

$$L(0) = \{0\}$$

$$L(\epsilon) = \{\epsilon\}$$

$$L(1) = \{1\}$$

So

$$\begin{aligned}
 L((0 \cup \epsilon)(1 \cup \epsilon)) &= (\{0\} \cup \{\epsilon\})(\{1\} \cup \{\epsilon\}) \\
 &= (\{0, \epsilon\})(\{1, \epsilon\}) \\
 &= \{01, 0, 1, \epsilon\}
 \end{aligned}$$

Spiral

## Converting regular expression to NFA

We can convert regular expression to NFA by following inductive steps.

For base case, we will show automatas for

- i) Finite automata for regular expression  $\epsilon$
- ii) Finite automata for regular expression  $\emptyset$
- iii) Finite automata for regular expression  $a$  (for all  $a \in \Sigma$ )

For inductive step, we show

- i) If regular expression  $R_1$  has equivalent finite automata  $M_1$  & regular expression  $R_2$  has equivalent automata  $M_2$ . Then a finite automata exists for  $R_1 \cup R_2$ .

- ii) ~~the~~ Finite automata for  $R_1 R_2$  exists if ~~exists~~ finite automata for  $R_1$  &  $R_2$  exists.

iii) Finite automata  $R^*$  exists if  
finite automata for  $R$  exist.

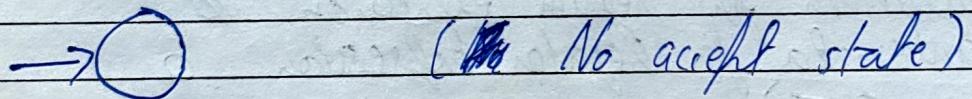
Proof:

Base Cases:

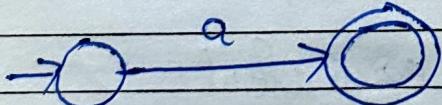
- NFA for regular expression  $\epsilon$



- NFA for regular expression  $\emptyset$



- NFA for regular expression  $a$

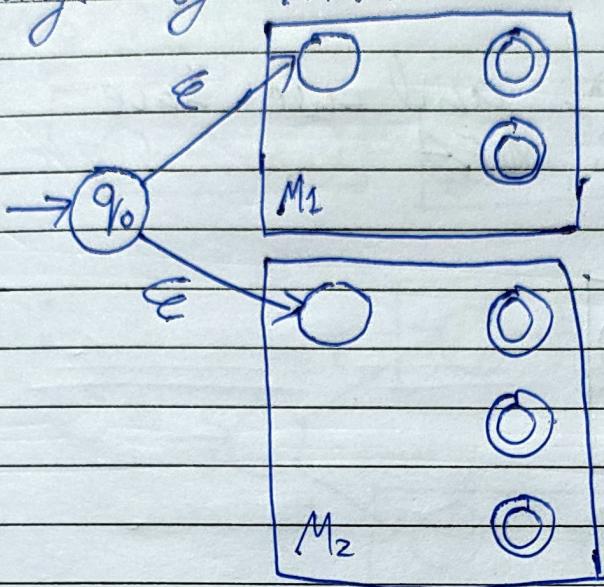


Induction steps

- NFA for  $R_1 \cup R_2$

$R_1$  has language  $L_1$  and NFA  $M_1$   
 $R_2$  has language  $L_2$  and NFA  $M_2$

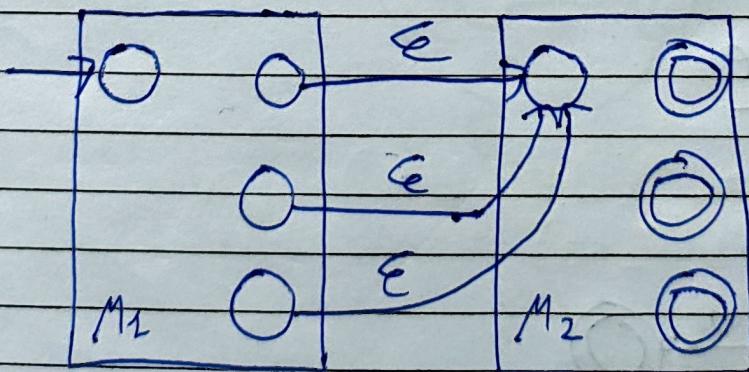
We know that  $R_1 \cup R_2$  describes  $L_1 \cup L_2$  and have shown that  $L_1 \cup L_2$  is language of NFA



### NFA for $R_1 R_2$

$R_1$  has language  $L_1$  and NFA  $M_1$   
 $R_2$  has language  $L_2$  and NFA  $M_2$

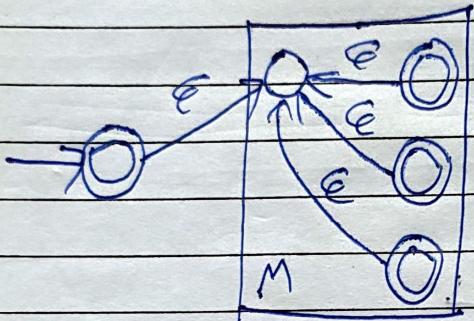
$R_1 R_2$  has language  $L_1 L_2$  and we have shown that  $L_1 L_2$  is described by NFA



• NFA for  $R^*$

$R$  has language  $L$  and NFA  $M$

$R^*$  has language  $L^*$ , and we have shown that  $L^*$  is NFA

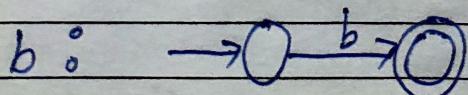
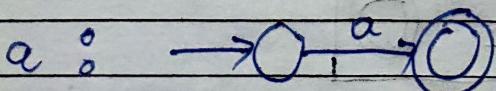
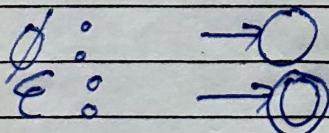


Example,  
convert regular expression  
 $(ab \cup a)^*$

to NFA

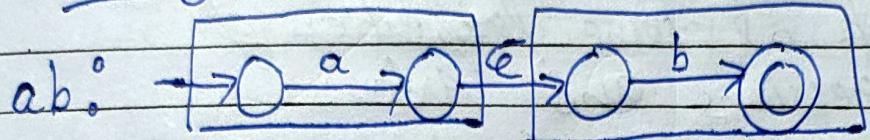
$$\Sigma = \{a, b\}$$

Base cases are

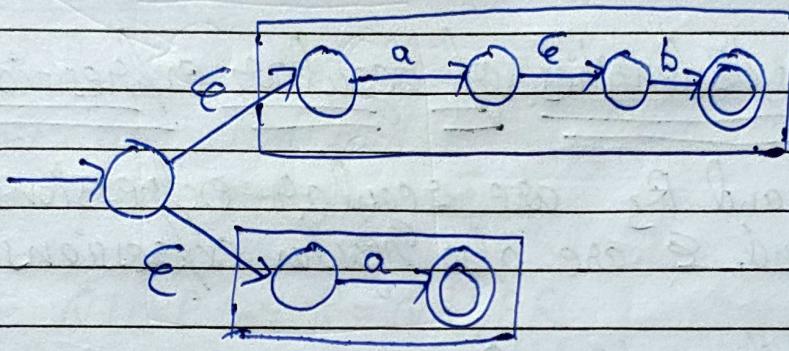




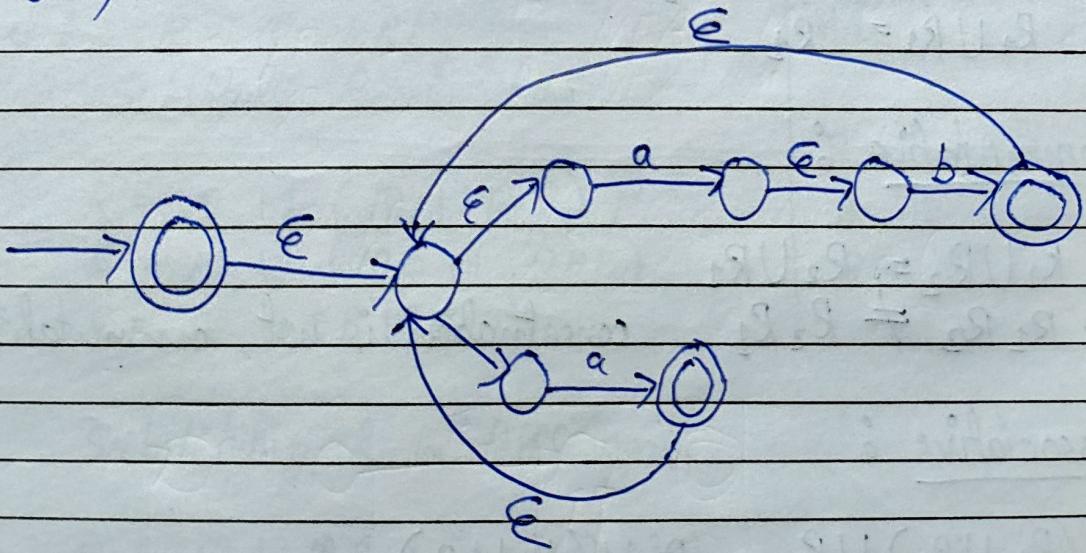
Creating NFA



ab U a<sup>\*</sup>:



(ab U a)<sup>\*</sup>:



# Converting DFA to regular expression

To convert DFA to regular expression, we need to look at some properties of regular expressions and Arden's theorem.

## Properties / Identities of regular expressions

$R_1, R_2$  and  $R_3$  are regular expressions.  
 $\emptyset$  and  $\epsilon$  are also regular expressions

1.  $R_1 \emptyset = \emptyset R_1 = \emptyset$
2.  $R_1 \epsilon = \epsilon R_1 = R_1$
3.  $R_1 \cup \emptyset = \emptyset \cup R_1 = R_1$
4.  $R_1 \cup R_1 = R_1$

### Commutative :

5.  $R_1 \cup R_2 = R_2 \cup R_1$
6.  $R_1 R_2 \neq R_2 R_1$  concatenation is not commutative

### Associative :

7.  $(R_1 \cup R_2) \cup R_3 = R_1 \cup (R_2 \cup R_3)$
8.  $(R_1 R_2) R_3 = R_1 (R_2 R_3)$

### Distributive :

$$9. R_1(R_2 \cup R_3) = R_1R_2 \cup R_1R_3$$

$$10. (R_1 \cup R_2)R_3 = R_1R_3 \cup R_2R_3$$

$$11. R_1 \cup (R_2 R_3) \neq (R_1 \cup R_2)(R_2 \cup R_3)$$

Union is not distributive over concatenation

### Arden's theorem.

For alphabet  $\Sigma$ , the three ~~three~~ regular expressions are  $P, Q$  and  $R$  and  $P$  does not contain  $\epsilon$ .

If  $R = QURP$  or  $R = QUPR$   
 Then ~~the~~  $R$  has a unique solution  
 $R = QP^*$  or  $R = P^*Q$  respectively

Proof:  $R = Q \cup RP$

Substitute  $R$  in RHS

$$R = Q \cup (Q \cup RP)P$$

$$R = Q \cup (QP \cup RPP)$$

$$R = Q \cup QP \cup RP^2$$

Hence

$$R = P \cup QR$$

~~then~~  $\Downarrow$

$$R = Q^*P$$

Substituting  $R$  in RHS again

$$R = Q \cup QP \cup (QURP)P^2$$

$$R = Q \cup QP \cup QP^2 \cup RP^3$$

Therefore, by repeatedly substituting

$$R = Q \cup QP \cup QP^2 \cup QP^3 \cup \dots$$

Using distributive property

$$R = Q(\epsilon U P U P^2 U P^3 U \dots)$$

~~Using property  $P U P^2 U P^3 U \dots = P^*$~~

$$\underline{R = Q(\epsilon U P^*)}$$

~~Since,  $\epsilon U R_1 = R_1$~~

Using property  $\epsilon U P U P^2 U P^3 U \dots = P^*$

$$\boxed{R = Q P^*}$$

So if given  $R = Q U P$ , and  $P$  does not contain  $\epsilon$   
 Solution ~~is~~ is  $R = Q P^*$  ( $R = Q U P R$  ~~solve to~~ ~~reduces to~~  $R = P^* Q$ )

Conversion from DFA to regular expression

We can now use Arden's theorem for this conversion.

We do this using a series of regular expression equations and then solving using ~~after~~ Arden's theorem.

Let  $M = (Q, \Sigma, \delta, q_f, F)$  be DFA to convert.

We create a equation for every state in this DFA.

For a state  $r$  in set  $Q$ , regular expression  
is

$$R_r = \begin{cases} \bigcup_{a \in \Sigma} a R_{s(r,a)} & \text{if } r \notin F \\ \left( \bigcup_{a \in \Sigma} a R_{s(r,a)} \right) \cup \epsilon, & \text{if } r \in F \end{cases}$$

$R_r$  is the regular expression for  $M$  if  
 $M$  had start state  $r$ .

Reason why

$R_r$  is  $\left( \bigcup_{a \in \Sigma} a R_{s(r,a)} \right)$  is because, if

$M$  does a transition for a symbol ' $a$ ', the  
regular expression for such can become  
 $a R_{s(r,a)}$  so it matches the first  
processed symbol and regular expression  
from resulting state  $s(r,a)$  i.e  $R_{s(r,a)}$

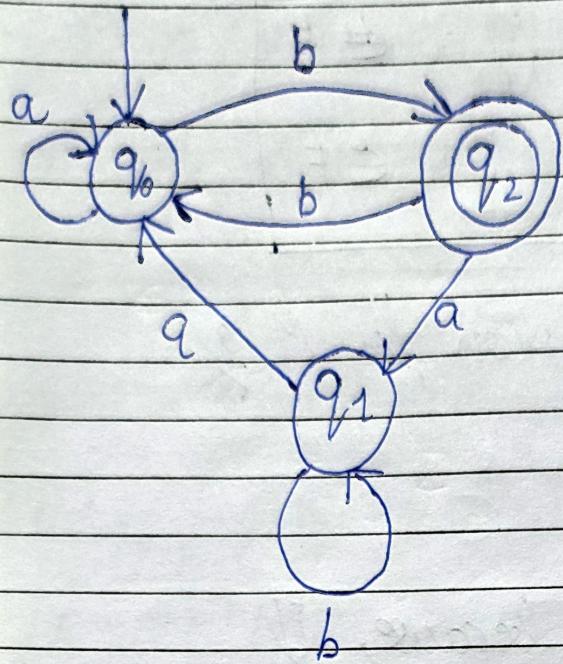
Accounting for all outgoing transitions  
given

$$R_r = \left( \bigcup_{a \in \Sigma} a R_{s(r,a)} \right)$$

Reason why we also have  $\epsilon$  if  $r \in F$   
is because if  $M$  starts at  $r$ , it can  
accept  $\epsilon$ .

These set of equations are then used  
to solve for  $R_r$

Example,



For this DFA, regular expressions are

$$\begin{aligned} R_{q_0} &= aR_{q_0} \cup bR_{q_2} \\ R_{q_1} &= aR_{q_0} \cup bR_{q_1} \end{aligned}$$

— (i)  
— (ii)

For final states

$$R_{q_2} = aR_{q_1} \cup bR_{q_0} \cup \epsilon \quad — (iii)$$

Since actual ~~start~~ start state is  $q_0$ , we need to solve  $R_{q_0}$

Putting eq (iii) in eq (i)

$$R_{q_0} = aR_{q_0} \cup b(aR_{q_1} \cup bR_{q_0} \cup \epsilon)$$

$$R_{q_0} = a R_{q_0} \cup b a R_{q_1} \cup b b R_{q_0} \cup b \quad \text{(ii)}$$

~~R<sub>q\_0</sub> = a R<sub>q\_0</sub>~~

~~R<sub>q\_1</sub>~~

Taking eq (ii)

$$R_{q_1} = a R_{q_0} \cup b R_{q_1}$$

Using Arden's theorem

$$R_{q_1} = \underbrace{a R_{q_0}}_R \cup \underbrace{b}_{P} \underbrace{R_{q_1}}_R$$

~~R<sub>q\_1</sub> = a R<sub>q\_0</sub>~~

$$R_{q_1} = b^* a R_{q_0}$$

Putting this result in eq (iv)

$$R_{q_0} = a R_{q_0} \cup b a b^* a R_{q_0} \cup b b R_{q_0} \cup b$$

$$R_{q_0} = (a \cup b a b^* a \cup b b) R_{q_0} \cup b$$

Using Arden's theorem

$$R_{q_0} = \underbrace{(a \cup b a b^* a \cup b b)}_R R_{q_0} \cup \underbrace{b}_{Q}$$

$$R_{q_0} = (a \cup b a b^* a \cup b b)^* b$$

## Pumping Lemma

Pumping lemma states that for a regular language  $\mathcal{L}$ , there is a constant  $p$  such that any string  $w \in \mathcal{L}$  can be split into three substrings if  $|w| \geq p$  so that

$$w = xyz ; y \neq \epsilon \text{ and } |xy| \leq p$$

such that  $y$  can be pumped and it will still be part of language i.e,

~~xyz, xyxz, xyyz, ... , xy...yz<sup>n</sup>~~

$$\{xyz, xyxz, xyyz, \dots, xy\dots yz^n\} \subset \mathcal{L}$$

This property is exclusive to regular languages thus it can be used to distinguish regular and non regular languages.

\* Pumping lemma thus shows that all languages with finite no. of strings are regular; since we can simply choose  $p$  to be larger than longest string

## Applications of pumping lemma

The most important application of pumping lemma is to show that a language is non-regular.

First Example,  $A = \{w : 0^n 1^n \text{ and } n \geq 0\}$

We proof by contradiction that  $A$  is non-regular.

Suppose  $A$  is regular

$\therefore$  it has pumping length ( $p$ ) which is some positive integer.

Let's consider ~~0<sup>p</sup> 1<sup>p</sup>~~ or ~~1<sup>p</sup>~~

$\Rightarrow$  Its length is greater than pumping length

So it must have  $y$  component which can be pumped such that  $y \neq \epsilon$  and ~~0<sup>p</sup> 1<sup>p</sup>~~  $|xy| \leq p$

So the string is

$000 \dots (p \text{ times}) 111 \dots (p \text{ times})$

Since ~~1<sup>p</sup>~~  $|xy| \leq p$ ,  $|z| \geq (2p - p)$   
 $\therefore$  So

$000 \dots \underbrace{111 \dots}_{\frac{1}{2}},$

All the 1's ~~are~~ on right ~~so~~ have to be a part of  $|z|$  to satisfy this condition.

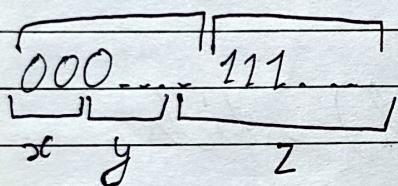
Therefore,  $y$  is only made up of 0's

So

$x \Rightarrow$  zero or more 0's

$y \Rightarrow$  one or more 0's

$z \Rightarrow$  ~~between~~ zero ~~of~~ or more 0's then 1's  
p times p times



Now if we pump  $y$ , the number of 0's will be greater than 1's

So ~~xy<sup>i</sup>z~~  $\notin A$  for  $i > 1$

Therefore by contradiction  $A$  is non-regular.

Second Example

$A = \{w : xx \text{ where } x \in \{0,1\}^*\}$   
i.e., ~~the~~ every  $w \in A$  can be split into two equal strings.

Assume  $A$  is a regular language with pumping length  $p$ .

Then,  $0^p 1 0^p$  belongs to  $A$  since it is symmetrical.

$\underbrace{000\dots 0}_{p \text{ times}} \underbrace{1}_{\text{1}} \underbrace{000\dots 0}_{p \text{ times}} 1$

$$\text{Total length} = 2p + 2$$

$$\text{Since } |xy| \leq p \text{ then } |z| \geq (2p+2) - p$$

$$|z| \geq p+2$$

So

$\underbrace{000\dots 0}_{p \text{ times}} \underbrace{1}_{\text{1}} \underbrace{000\dots 0}_{p \text{ times}} 1$

$x \quad y \quad z$

$y$  is always only 0's.

If we try to pump  $y$ , string no longer belongs to  $A$

So,  $xy^iz \notin A$  for  $i > 1$

So by contradiction,  $A$  is not regular.

Third example  $A = \{1^n\}^{\infty} : n \geq 0$

i.e., 1 repeated  $n^2$  times

Assume  $A$  is regular, so assume pumping length is  $p$

$1^{p^2}$  has length  $p^2$  which is bigger than  $p$

So,  $1^{p^2}$  should be in  $A$  according to assumption.

$1^{p^2}$  has three components  $x, y$  and  $z$  such that  $y \neq \epsilon$  and  $|xy| < p$

$$\text{so } |xyz| = p^2$$

Now,

$$|x y^2 z| = |x y y z| = |y| + |xyz|$$

$$|x y^2 z| = |y| + p^2 \quad \text{--- (1)}$$

Since,  $|xy| \leq p$   
 $1 \leq |y| \leq p$

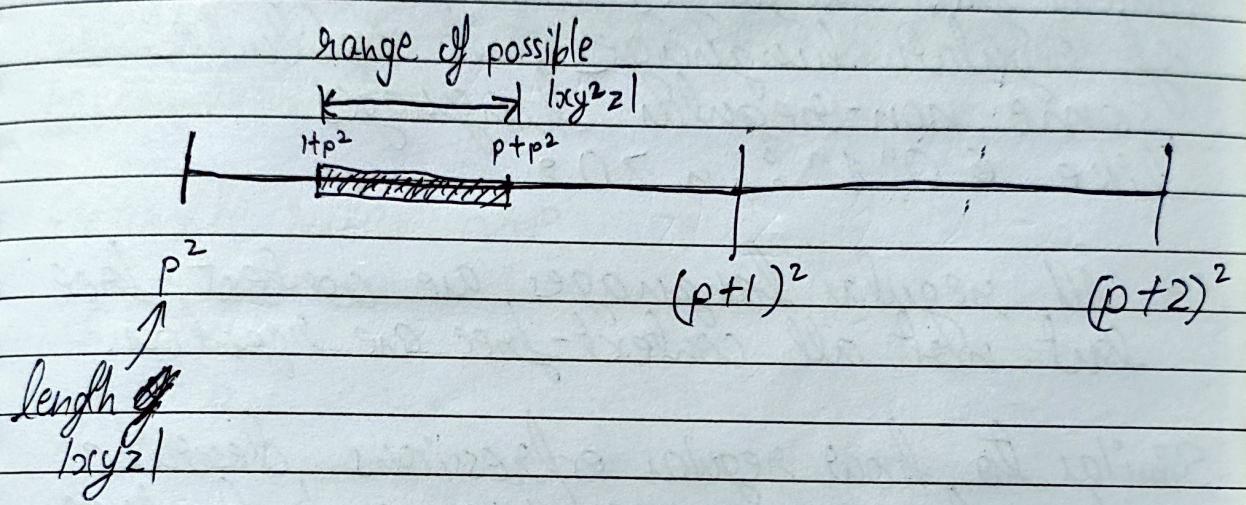
$$1 + p^2 \leq |y| + p^2 \leq p + p^2$$

Using (1)

$$1 + p^2 \leq |x y^2 z| \leq p + p^2$$

Nay, since  $|xyz|$  will have longer length than  $|bcyz|$  it must fall in range  
should cover atleast one complete square

But it does not fall in any square



So,  $|bcyz|$  cannot be a larger square

$$xyz \in A$$

So by contradiction, A is not regular.