Prof. Minakshi

Unit-2

⇒ Regular expressions and Regular Languages

- **Regular Language**

A language L is known as regular if and only if it is recognized by a finite automata (FA).

A language L is known as regular if and only if it is described by a regular expression (RE).

Regular Expression

- The language accepted by finite automata is easily expressed by simple expressions called regular expressions.
- A regular expression consists of strings of symbols from some alphabet Σ, parentheses ( ) and operators (union, concatenation, closure).

Regular expressions has 3 operations
- union (+, Or, |)
- concatenation (.)
- Kleene closure (*)

Definition of a regular expression:

Let Σ be a given alphabet, then the regular expressions over Σ are defined recursively

Let Σ be a given alphabet, then

- ϕ, ϵ, and $a \in \Sigma$ are all regular expressions. These are known as primitive regular expressions.
- **Recursive definition:**

  if r_1 and r_2 are regular expressions, then the following expressions are also regular expressions

  - (union) $r_1 + r_2 \Rightarrow r_1$ Or r_2
  - Concatenation $r_1 . r_2 \Rightarrow r_1$ followed by r_2
  - $r_1^* \Rightarrow r_1$ repeated zero or more times (closure)

- A string of symbols is a regular expression if and only if it can be derived from primitive regular expressions by finite applications of recursive definition.

Precedence of operators

Parentheses	( )	(Highest)
closure	*	↑ increases
Concatenation	.	
Union	+	(Lowest

Identities for regular expressions

I_1 $\phi + R = R = R + \phi$

I_2 $\phi R = R\phi = \phi$

I_3 $\epsilon R = R\epsilon = R$

$\epsilon + R = R + \epsilon = R$

I_4 $\epsilon^* = \epsilon$

$\phi^* = \epsilon$

I_5 $R + R = R$

I_6 $R^*R^* = R^*$

$R^*R^* = R^* = R^*(R + \epsilon)$

I_7 $RR^* = RR^*$

I_8 $(R^*)^* = R^*$

I_9 $\epsilon + RR^* = R^*$

I_{10} $(PQ)^*P = P(QP)^*$

I_{11} $(P^*R^*)^* = (P + R)^*$

Regular Expressions

- another type of language defining notations
- applications ⇒ in text-search
  ⇒ in compiler components (lexical analyzer)
  ⇒
- closely related to NFA.

- operations
  - Union
  - concatenation
  - closure

Precedence of Regular Expression operators:
- closure (highest)
- concatenation
- Union (lowest)

Examples of regular expressions

① set of strings of alternating 0's and 1's

$$\underbrace{(01)^* + (10)^*}_{\text{for even length strings}} + \underbrace{0(10)^* + 1(01)^*}_{\text{for odd length strings}}$$

or

$$\boxed{(\epsilon+1)\ (01)^*\ (0\epsilon+0)}$$

② a set of strings of length exactly 2.

$$\underset{\uparrow}{\underline{\ \ }}\ \underset{\uparrow}{\underline{\ \ }}$$
a,b a,b

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

③ a set of strings of length at least two

$$\underset{\uparrow}{\underline{\ \ }}\ \underset{\uparrow}{\underline{\ \ }}\ \underline{\text{many more...}}$$
a,b a,b 0 or more.

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

④ a set of strings of length at most 2. ⇒ can be 0, 1, 2.

$$\underset{\uparrow}{\underline{\ \ }}\ \underset{\uparrow}{\underline{\ \ }} \Rightarrow (\epsilon+a+b)(\epsilon+a+b)$$
ϵ,a,b. ϵ,a,b

Ques
→ at least one a and at least one b over alphabet $\Sigma = \{a, b, c\}$

$$(a+b+c)^* \underline{a} (a+b+c)^* \underline{b} (a+b+c)^* +$$
$$(a+b+c)^* \underline{b} (a+b+c)^* \underline{a} (a+b+c)^*$$

or $(a+b+c)^* [\, a(a+b+c)^* b + b(a+b+c)^* a\,] (a+b+c)^*$

Ques The set of strings over $\Sigma = \{0,1\}$, whose tenth symbol from right end is 1. any sub string

$$\underline{\downarrow} \; \underline{1} \; \underline{\downarrow} \; \underline{\downarrow} \; \underline{\downarrow} \; \underline{\downarrow} \; \underline{\downarrow} \; \underline{\downarrow} \; \underline{\downarrow} \; \underline{\downarrow} \; \underline{\downarrow}$$
$$0,1 \quad 0,1 \quad 0,1 \quad 0,1 \quad 0,1 \quad 0,1 \quad 0,1 \quad 0,1 \quad 0,1$$

$$\Rightarrow (0+1)^* 1 (0+1)(0+1)(0+1)(0+1)(0+1)(0+1)(0+1)(0+1)(0+1)$$
$$\Rightarrow (0+1)^* 1 (0+1)^9$$

Ques: at most one pair of consecutive 1's.

- at most one pair → ϵ, 11
- any number of zero's can be starting or ending.

$$(0+10)^* (\epsilon + 11) (0+01)^*$$

Ques: 3 consecutive zero's.

$$(0+1)^* 000 (0+1)^*$$

⇒ Regular expression for:- no two consecutive 1's except for possibly a string of 1's at end.

$$(0+01)^* 1^*$$

⇒ the set of strings in which every pair of adjacent zero's appears before any pair of adjacent ones.

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

Language associated with regular expression or Regular sets denoted by regular expressions

- If R is a regular expression then L(R) denotes the language associated with R.
- ϕ is a regular expression denoting empty set
  $R = \phi$ $\quad L(R) = \{\}$
- ϵ is a regular expression denoting $\{\epsilon\}$
  $R = \epsilon$ $\quad L(R) = \{\epsilon\}$
- for every $a \in \Sigma$, a is regular expression denoting $\{a\}$
  $R = a$ $\quad L(R) = \{a\}$
- if r_1 and r_2 are regular expressions, then
  $R = r_1 + r_2$ $\quad L(R) = L(r_1) \cup L(r_2)$
  $R = r_1 . r_2$ $\quad L(R) = L(r_1) . L(r_2)$
  $R = r_1^*$ $\quad L(R) = (L(r_1))^*$

Equivalence of regular expressions with finite automata

Regular expressions and finite automata are equivalent in their descriptive power that is
Any regular expression can be converted into finite automata that recognizes the language and vice versa.
Any FA can be converted to regular express (equivalent Eq.es) that describe the language.

- Regular expression to FA
- FA to regular expression.

DFA to regular Expressions

Converting DFA to regular expression

- Two methods
  - Arden's Method
  - State Elimination (Removal) Method)

Arden's Theorem:

- Arden's theorem is used to convert a given DFA to regular expression
- It states that "Let P and Q be two regular Expressions over Σ.
  If P does not contain epsilon or null string
  then $R = Q + RP$ has a unique solution

$$R = QP^*$$

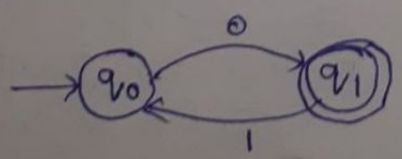
conditions:

- To use arden's theorem, the following conditions must be satisfied.
  1. The transition diagram must not have any ϵ transition.
  2. There must be only a single initial state.

Steps:

1. Form an equation for each state considering incoming transitions towards the state.
2. Add the epsilon ϵ in the initial state equation
3. Bring the final state in form $R = Q + RP$
4. if there exists multiple final states then
   - Write regular expression for each final state separately
   - Add (union) all the regular expressions to get the final regular expressions.

Example: convert DFA to regular expression.



1. Equations for each states considering incoming transitions

$$q_0 = q_1 \cdot 1 + \epsilon$$

$$q_1 = q_0 \cdot 0$$

Form final state in form ⇒ $R = Q + RP$

(R ↑ state)

$q_1 = q_0 \cdot 0$

$q_1 = (\epsilon + q_1 \cdot 1)0$ substituting

$q_1 = \epsilon \cdot 0 + q_1 \cdot 1 \cdot 0$

$q_1 = \underline{0} + q_1 \cdot \underline{10}$

$R = \underline{Q} + R\underline{P}$

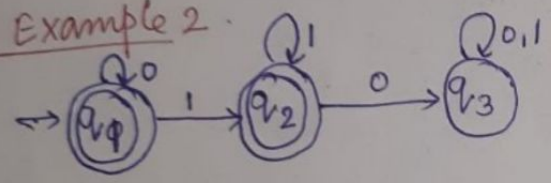
Solution

$R = QP^*$

$q_1 = 0.(10)^*$

The regular expression for given DFA ⇒ $0.(10)^*$

Example 2.



Equations for each state

$q_1 = \epsilon + q_1 \cdot 0$

$q_2 = q_1 \cdot 1 + q_2 \cdot 1$

$q_3 = q_2 \cdot 0 + q_3 \cdot (0+1)$

$q_1 = \epsilon + q_1 \cdot 0 \Rightarrow \underline{R} = Q + \underline{R}P$

$q_1 = \epsilon 0^*$ ⇒ $R = QP^*$

$q_1 = 0^*$

$q_2 = q_1 \cdot 1 + q_2 \cdot 1$

$q_2 = 0^* \cdot 1 + q_2 \cdot 1$

$q_2 = 0^* \cdot 1 \cdot 1^*$

q_1 and q_2 are final states
But q_3 is not final
So we do not need to solve q_3.

$q_1 + q_2 = 0^* + 0^*11^*$

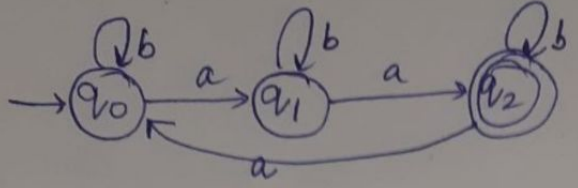
$= 0^*[\epsilon + 11^*]$

$= 0^*[\epsilon + 1^+] \Rightarrow 0^*1^*$ (Regular Expression for given DFA)

Language

any number of zero's followed by any number of one's.

Example 3



Equations

$q_0 = \epsilon + q_0 \cdot b + q_2 \cdot a$

$q_1 = q_0 \cdot a + q_1 \cdot b$

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

q_2 is final state

$q_0 = \epsilon + q_2 a + q_0 b$

$R = Q + RP$

$R = QP^*$

$q_0 = (\epsilon + q_2 a) b^*$

$\boxed{q_0 = \epsilon b^* + q_2 a b^*} \Rightarrow \boxed{q_0 = b^* + q_2 a b^*}$

$q_1 = q_0 \cdot a + q_1 \cdot b$

$q_1 = (b^* + q_2 a b^*) \cdot a + q_1 \cdot b$

$R = Q + RP \Rightarrow R = QP^*$

$q_1 = (b^* + q_2 a b^*) \cdot a \cdot b^*$

$\boxed{q_1 = b^* a b^* + q_2 a b^* a b^*}$

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

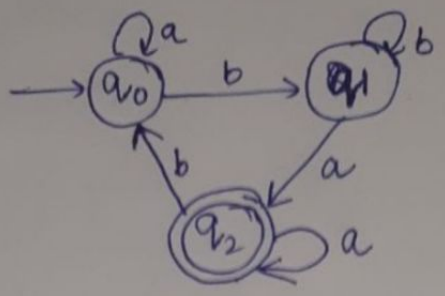
$q_2 = (b^* a b^* + q_2 a b^* a b^*) a + q_2 \cdot b \Rightarrow b^* a b^* a + q_2 a b^* a b^* a + q_2 b$

$q_2 = b^* a b^* a + q_2 [a b^* a b^* a + b]$

$\boxed{q_2 = b^* a b^* a (a b^* a b^* + b)^*}$

Practice Questions

Example 4



Re ⇒ ?

$$q_0 = q_0a + q_2b + \epsilon$$

$$q_1 = q_0b + q_1b$$

$$q_2 = q_2a + q_1a$$

Solve q_1 Arden's Theorem

$$q_1 = \underbrace{q_0b}_{Q} + q_1\underbrace{b}_{P} \qquad q_1 = q_0bb^*$$

Solve q_2 $\quad q_2 = q_2a + \underline{q_0bb^*a}$

$$= \underbrace{q_0bb^*a}_{Q} + q_2\underbrace{a}_{P} \qquad q_2 = q_0bb^*aa^*$$

Solve q_0 $\quad q_0 = q_0a + q_0bb^*aa^* + \epsilon$

$$q_0 = q_0\underbrace{(a + bb^*aa^*)}_{P} + \underbrace{\epsilon}_{Q}$$

$$q_0 = \epsilon(a+bb^*aa^*)^* \Rightarrow q_0 = (a+bb^*aa^*)^*$$

Solve q_2

$$q_2 = q_0bb^*aa^*$$

$$q_2 = (a+bb^*aa^*)^*bb^*aa^*$$

regular expression

$$(a + bb^*aa^*)^*bb^*aa^*$$

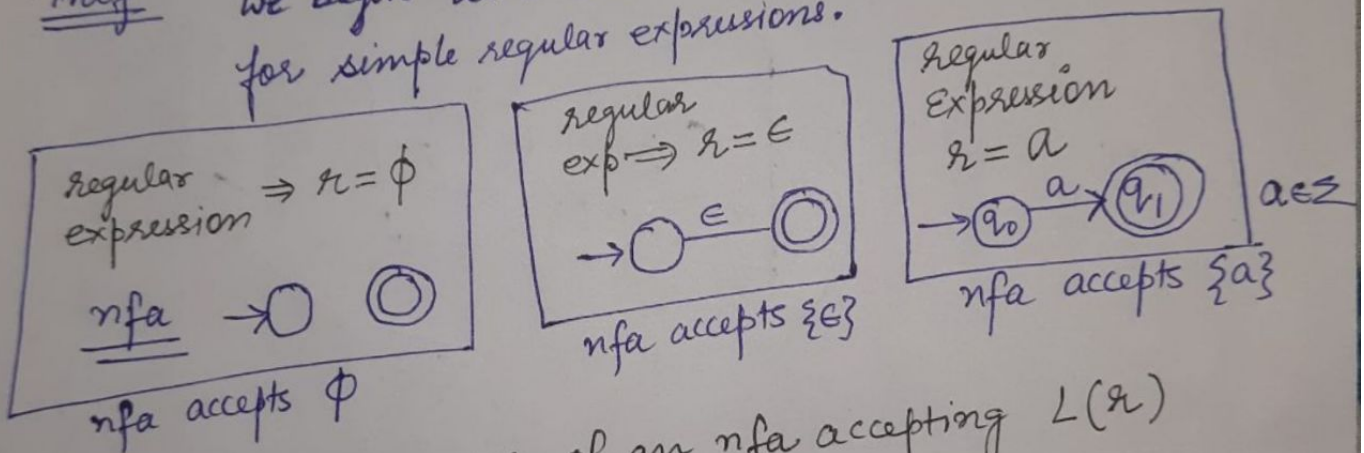
∈-NFA from Regular ~~Law~~ Expressions

- Let r be a regular expression
- Let $L(r)$ is a regular language described by the regular expression r.

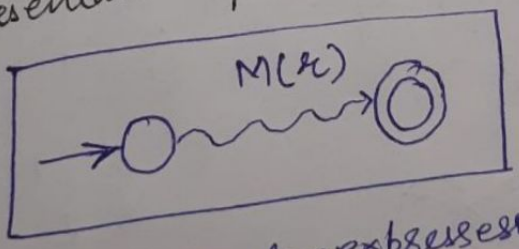
Note for every regular expression, there is a regular language for every regular language, there is a regular expression

Theorem: Let r be a regular expression. Then there exists some non-deterministic finite accepter that accepts $L(r)$. consequently $L(r)$ is a regular language.

Proof we begin with automata that accept languages for simple regular expressions.



schematic representation of an nfa accepting $L(r)$



Let r_1 and r_2 are two regular expresses

$L(r_1)$ language described by r_1

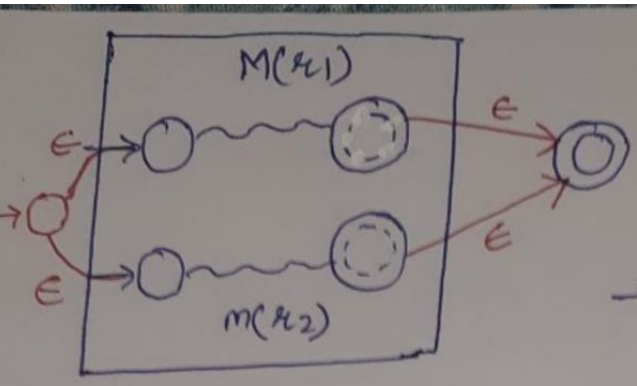
$L(r_2)$ language described by r_2

$r_1 + r_2 \Rightarrow$ union of r_1 and r_2

$L(r_1 + r_2) \Rightarrow$ language described by union of r_1, r_2.

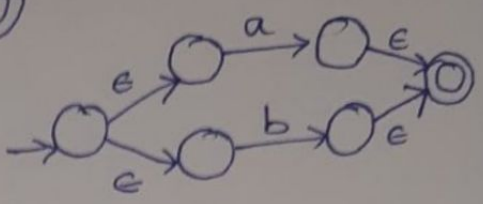
$L(r_1 \cdot r_2) \Rightarrow$ " " " concatenation of r_1 and r_2

ϵ-NFA or automata for $L(r_1 + r_2)$

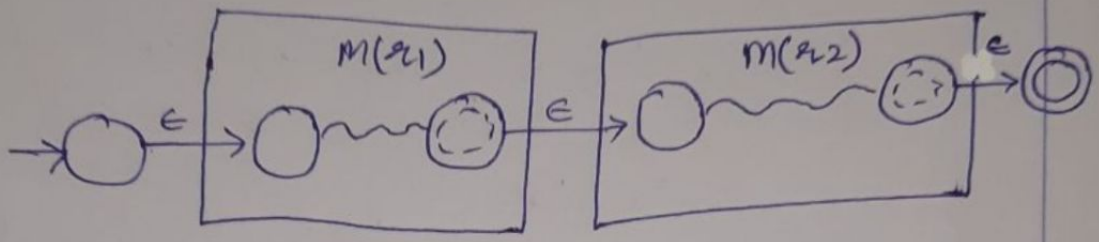


Example

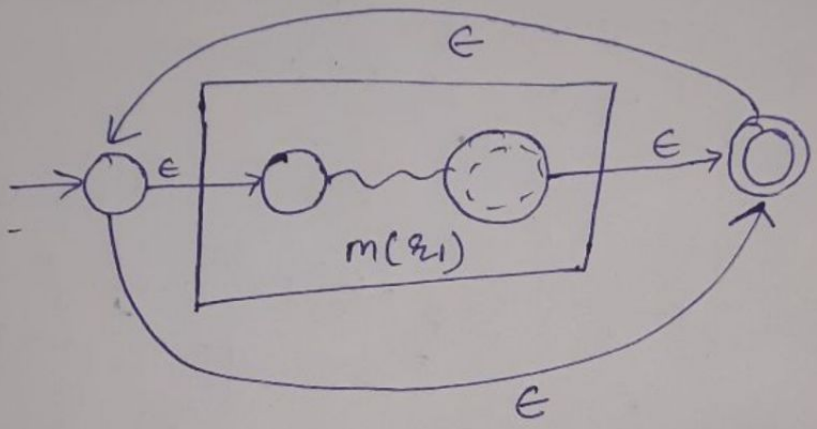
$r = a + b$



ϵNFA or Automata for $L(r_1 r_2)$

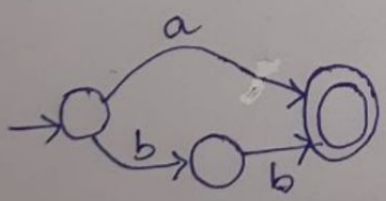


ϵNFA or automata for $(L(r_1^*))$

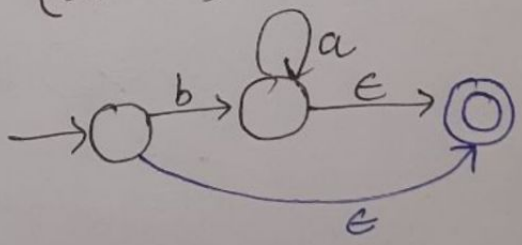


Ques: Draw automata for regular expressions, directly

$a + bb$

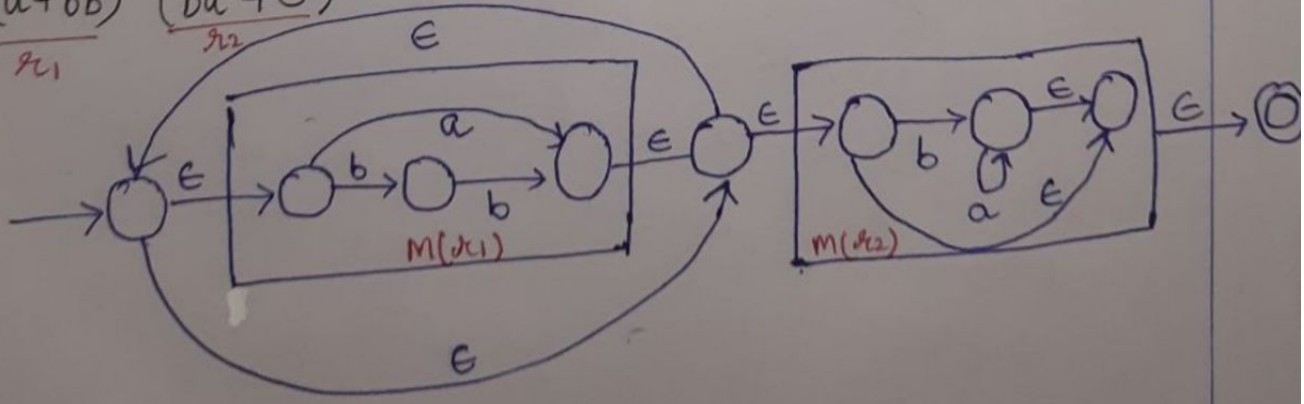


$(ba^* + \epsilon)$



$(a+bb)^*$ $(ba^*+\epsilon)$

r_1 r_2



Equivalence of Two finite automata

Steps to identify equivalence (comparison Method)

(1) For any pair of states $\{q_i, q_j\}$, the transition for input $a \in \Sigma$ is defined by $\{q_a, q_b\}$

where $\delta(q_i, a) = q_a$

$\delta(q_j, a) = q_b$

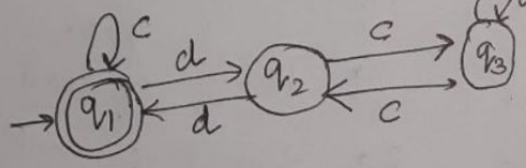
The two automata are equivalent if for a pair $\{q_a, q_b\}$,

- both the states q_a, q_b are non-final.
- both the states q_a, q_b are final.

(2) If initial state is final state of one automaton, then in second automaton it should be also final state.

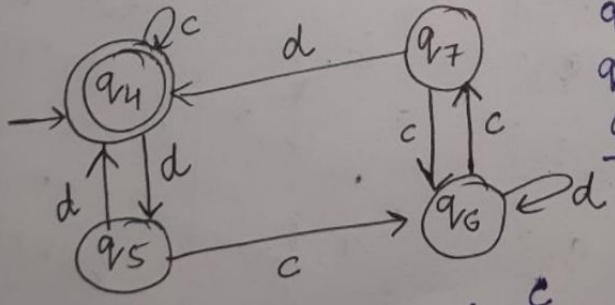
Example

M1 DFA



	c	d
→ q_1	q_1	q_2
⇒ q_2	q_3	q_1
q_3	q_2	q_3

M2 DFA



	c	d
→ q_4	q_4	q_5
q_5	q_6	q_4
q_6	q_7	q_6
q_7	q_6	q_4

Comparison method

Draw transition Table

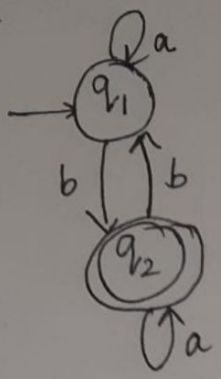
or comparison Table

	c	d
→ $\{q_1, q_4\}$ F F	$\{q_1, q_4\}$	$\{q_2, q_5\}$
$\{q_2, q_5\}$ NF NF	$\{q_3, q_6\}$	$\{q_1, q_4\}$
$\{q_3, q_6\}$ NF NF	$\{q_2, q_7\}$	$\{q_3, q_6\}$
$\{q_2, q_7\}$ NF NF	$\{q_3, q_6\}$	$\{q_1, q_4\}$

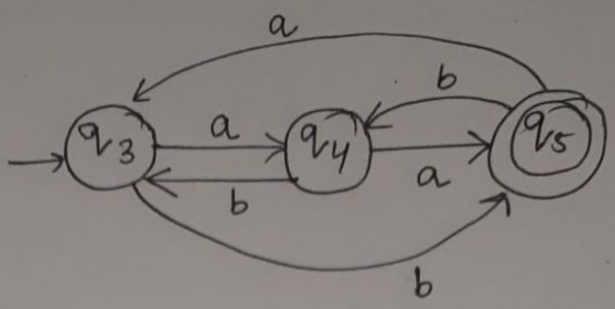
All state pair contains either both final states or both non-final states

Hence M_1 and M_2 are equivalent.

Example 2



M1 DFA



M2
DFA

Transition Table M_1

	a	b
→ q_1	q_1	q_2
(q_2)	q_2	q_1

Transition Table M_2

	a	b
→ q_3	q_4	q_5
q_4	q_5	q_3
(q_5)	q_3	q_4

Comparison Table

initial state ⇒ $\{q_1, q_3\}$

	a	b
→ $\{q_1, q_3\}$	$\{q_1, q_4\}$	$\{q_2, q_5\}$
$\{q_1, q_4\}$	$\{q_1, q_5\}$	$\{q_2, q_3\}$
$\{q_2, q_5\}$	$\{q_2, q_3\}$	$\{q_1, q_4\}$
$\{q_1, q_5\}$	$\{q_1, q_3\}$	$\{q_2\ q_4\}$
F NF $\{q_2, q_3\}$	$\{q_2\ q_4\}$	$\{q_1\ q_5\}$
$\{q_2, q_4\}$ F NF	$\{q_2\ q_5\}$	$\{q_1\ q_3\}$

There exist pairs of states where one state is final and another one is non-final.

Hence DFA M_1 and M_2 are not equivalent.

Definition of a transition graph

A transition graph is defined by 5-tuples

Q A finite set of states

Σ a finite set of input symbols

$S \subseteq Q$ a set of starting states $S \subseteq Q$ (non-empty set)

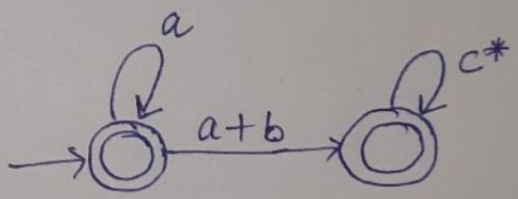
$F \subseteq Q$ a set of final or accepting states

δ A finite set of transitions, (directed edge labels) (q_i, a, q_j) where $q_i, q_j \in Q$, $a \in \Sigma$

- The transition graph is a directed labelled graph.

Generalized transition graph

A generalized transition graph is a transition graph whose edges are labeled with regular expressions; otherwise it is same as the usual transition graph.

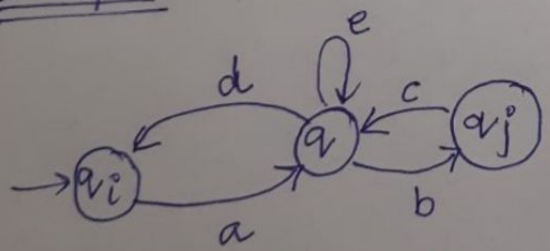


Example

Generatized graph for language

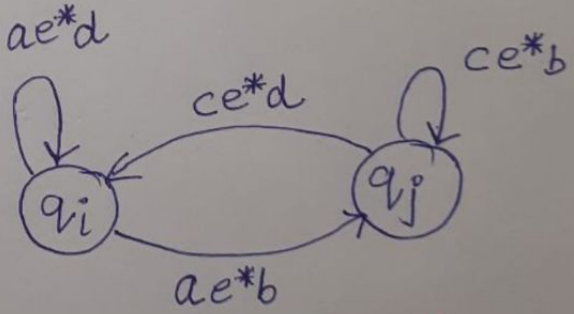
$L(a^* + a^*(a+b)c^*)$

Example



Transition graph

removal of q



generalized transition graph

Prof. Minakshi

Pigeonhole Principle

if we put n objects (pigeons) into m boxes (pigeonholes), and if $n > m$ then at least one box must have more than one item in it.

Pumping Lemma for regular languages (or regular sets)

- pumping lemma for regular languages uses pigeonhole principle The proof is based on the observation that in a transition graph with n states any walk of length n or longer than n must repeat some vertex, that is contain a cycle.

Theorem: Let L be an infinite regular language. Then there exists some positive integer m such that any $w \in L$ with $|w| \geq m$ can be decomposed as

$$w = xyz \quad \text{with } |xy| \leq m \text{ and } |y| \geq 1$$

such that $w_i = xy^iz$ is also in L for all $i = 0, 1, 2, \ldots$

It means "Every sufficiently long string in L can be broken into three parts in such a way that any arbitrary number of repetitions of the middle part (i.e y) yields another string in L. we say that the middle string is pumped Hence the pumping lemma term used.

Proof
- Let L is regular, there exists a dfa that recognizes it Let such a dfa have n states
- Now take a string w in L such that $|w| \geq m$ if $m \geq n$ then there exists x, y, z such that $w = xyz$, $y \neq \epsilon$ and $xy^iz \in L$ for each $i \geq 0$.

Let $w = a_1 a_2 a_3 ---- a_m$ $\quad m > n$

$|w| \geq m$ so it w contains at least m symbols.

$\delta(q_0, a_1 a_2 a_3 ---- a_i) = q_i$ for $i = 1, 2, 3 --- m$

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

Q_1 is sequence of states in the path with value $w = a_1 a_2 -- a_m$

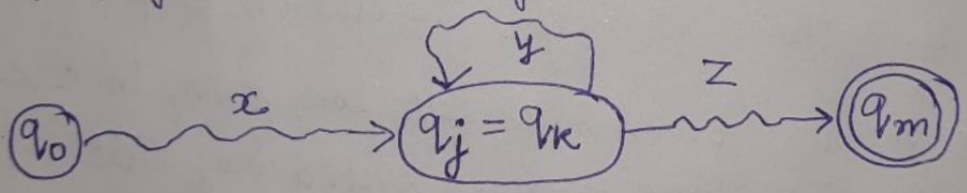
As there are n distinct states, at least two states in Q_1 must coincide.

- Among various pairs of repeated states, we take the first pair.
- Let take q_j and q_k $(q_j = q_k)$
- Then j and k satisfy the condition $0 \leq j < k \leq n$

w can be decomposed into three substrings

$\underbrace{a_1 a_2 --- a_j}_{x}$, $\underbrace{a_{j+1} ---- a_k}_{y}$, $\underbrace{a_{k+1} ----- a_m}_{z}$

As $k \leq n$, $|xy| \leq n$ and $w = xyz$



As $k \leq n$, $|xy| \leq n$ and $w = xyz$. The path with path value w shown

The automaton M starts from the initial state q_0. On applying string x, it reaches q_j $(= q_k)$.

On applying the string y, it comes back to q_j $(= q_k)$. So after application of y^i for each $i \geq 0$ the automaton is in same state q_j

On applying z, it reaches final state q_m.

Hence $xy^iz \in L$. As every state in Q_1 is obtained by applying an input symbol, $y \neq \epsilon$.

Pumping lemma

- gives necessary conditions for an input string to belong to a regular set.
- pumping lemma gives a method of pumping (generating) many strings from a given string.
- Pumping lemma gives a necessary condition, that can be used to show that certain language is not regular.
- It is a negative test.

Application of pumping lemma

- Pumping lemma is to be applied to show that certain languages are not regular.
- If a language is regular, it satisfies pumping lemma.
- If L does not satisfy pumping lemma, then it is not regular.

Method to prove that a languag L is not regular.

- First, assume that L is regular, it has pumping length p
- So pumping lemma should hold for L.
- Use the pumping lemma to obtain contradiction.
  - select w such that $|w| \geq p$
  - select y such that $|y| > 1$
  - select x such that $|xy| \leq p$
  - assign the remaining string to z.
  - select i such that the resulting string is not in L.

Hence L is not regular.

if L is a regular language then L has a pumping length p such that any string w where $|w| \geq p$ may be divided into 3 parts $w = xyz$ such that following conditions must be true

① $|y| > 0$

② $|xy| \leq p$

③ $xy^iz \in L$ for all i

Ques: Show that $L = \{a^n b^n \mid n \geq 0\}$ is not a regular language.

- Assume L is a regular language, it has a pumping length p
- any string such that $|string| \geq p$ can be pumped.
- choose a string w such that $w = a^p b^p$
- Now divide w into xyz such that

  $|y| > 0$

  $|xy| \geq p$
- Assume $p = 7$
- $w =$ aaaaaaabbbbbbb

Case 1 y has only a's

$\underbrace{aa}_{x}\underbrace{aaaaa}_{y}\underbrace{bbbbbbb}_{z}$ Here $|y| > 0$ and $|xy| \leq p$

$6 \leq 7$

$i=2$ $\underbrace{aa}_{}\underbrace{aaaa}_{y}\underbrace{aaaa}_{y}\underbrace{abbbbbbb}_{z} \notin L$ condition ③ fails.

Case 2 y has b part only.

$\underbrace{aaaaaaa}_{x}\underbrace{bbbbbbb}_{y}$ Here $|y| > 0$ and $|xy| \not\leq p$

$9+4 \not\leq 7$

$i=2$

$\underbrace{aaaaaaa}_{x}\underbrace{bb}_{y}\underbrace{bbbb}_{y}\underbrace{bbbb}_{y}\underbrace{b}_{z} \notin L$

condition ②
condition ③ failed

Case 3 y is in a and b part

$\underbrace{aaaaaa}_{x}\underbrace{a}_{y}\underbrace{bbbbbbb}_{z}$

Here $|y|>0$ $|xy|\leq p$
$9 \not\leq 7$

$i=2$

$\underbrace{aaaaa}_{x}\underbrace{aabb}_{y}\underbrace{aabb}_{y}\underbrace{bbbbbb}_{z} \notin L$

condition ②
condition ③ failed

Condition $xy^iz \notin L$
This is the contradiction
Hence L is not regular.

Ques Show that $L=\{ww \in \{a,b\}^*\}$ is not regular using pumping lemma.

- Assume L is regular and it has pumping length p.
- any string such that |string| $\geq p$ can be pumped.
- choose $w = a^p b a^p b$
- Assume $p=7$
- Divide w into xyz

$\underbrace{aa}_{x}\underbrace{aaaa}_{y}\underbrace{a\,b}_{z}\underbrace{aaaaaaa\,b}_{z}$

$|y|>0 \Rightarrow$ True.
$|xy| \leq p$
$6 \leq 7 \Rightarrow$ True

$i=2$ $aa\underbrace{aaaa}_{y}\underbrace{aaaa}_{y}\underbrace{ab\,aaaaaaab}_{z} \notin L$

$xy^2z \notin L$.

$xy^iz \notin L$
condition ③ failed.

we get a contradiction, hence,
$L = \{ ww \mid w \in \{a,b\}^* \}$ is not a regular language.

Prob: $L = \{ w \in \{a,b\}^* \mid w$ has an equal number of a's and b's $\}$ is not regular.

- Assume L is regular.
- We know that $L_1 = a^*b^*$ is regular
- Hence the $L_1 \cap L$ must be regular because regular language is closed under intersection.
- $L_1 \cap L \Rightarrow L_2 = \{ a^n b^n \}$
- It has been proved by pumping lemma, $\{a^n b^n : n \in N\}$ is not regular
- This is the contradiction to the closure property of regular languages under intersection.

Hence $L = \{ w \in \{a,b\}^* \mid w$ has equal number of a's and b's $\}$ is not regular.

Example

Show that $L = \{a^p \mid p \text{ is a prime number}\}$ is not a regular language.

(1) Assume L is regular language. Let n be the number of states in the finite automata accepting L.

(2) Let p be a prime number greater than n. Let $w = a^p$

(3) By pumping lemma, w can be written as $w = xyz$ with $|xy| \leq n$ and $|y| > 0$.

x, y, z are simply strings of a's

So $y = a^m$ for some $m \geq 1$ and $m \leq n$

(3) Let $i = p+1$

Then $|xy^iz| = |xyz| + |y^{i-1}|$

$= p + (i-1)m$

$= p + im - m$

$= p + (p+1)m - m$

$= p + pm + m - m$

$= p + pm$

$= p(1+m)$

Since $w = a^p$, $|w| = p$

$w = xyz$, $|xyz| = p$

$y = a^m$

$y^{i-1} \Rightarrow (a^m)^{i-1}$

$|y^{i-1}| \Rightarrow (i-1)m$

$i = p+1$

→ $p(1+m)$ is not a prime number

→ $|xy^iz|$ is not a prime number

→ $xy^iz \notin L$

This is a contradiction. Therefore, L is not a regular language.

Example Show that $L = \{a^{i^2} \mid i \geq 1$ is not a regular$\}$

① Suppose L is regular. Let p be the number of states in the finite automaton accepting L. L has a pumping length p.

② Let $w = a^{p^2}$. Then $|w| = p^2$, $p^2 > p$ By pumping lemma we can write $w = xyz$ with $|xy| \leq p$ and $|y| > 0$

③ Now consider xy^2z, xy^iz where $i=2$

$|x| + 2|y| + |z| > |x| + |y| + |z|$ as $y > 0$

This means

$|xyz| = p^2 < |xy^2z|$

$|x| + |y| + |z| = p^2 < |x| + 2|y| + |z|$

As per condition $|xy| \leq p$ so we have $|y| \leq p$.

$$\begin{aligned} |xy^2z| &= |x| + 2|y| + |z| \\ &= |x| + |y| + |y| + |z| \\ &\not\equiv \underline{|x| + |y| + |z|} + |y| \\ &\leq p^2 + p \qquad \because |y| \leq p \end{aligned}$$

i.e $|xyz| < |xy^2z|$

$p^2 < |xy^2z| \leq p^2 + p$

$p^2 < |xy^2z| < p^2 + p + p + 1$

$p^2 < |xy^2| < (p+1)^2$

Hence $|xy^2z|$ strictly lies b/w p^2 and $(p+1)^2$ but is not equal to any one of them.

Thus $|xy^2z|$ is not a perfect square and so $xy^2z \notin L$.

This is contradiction. Therefore L is not regular.

Closure Properties of regular sets or regular language

- class of regular sets or regular language is closed under following operations
  - Union
  - Concatenation
  - closure (iteration)
  - Transpose
  - Complementing
  - set intersection

• Transpose

if L is regular, then L^T is also regular.

if L is a regular language then we can create a finite automaton $M = (Q, \Sigma, q_0, \delta, F)$ such that M accepts L.

we can construct a transition system M' by starting with the starting state of M, and reversing the direction of directed edges

initial state ⇒ final state
final states = initial state
reverse the edges

- The set of initial states of M' = set of final states of M
  = F
- The final state of M' = initial state of M
  = q_0

if $w \in L$. we have a path from q_0 to some final state in F with path value w.

By reversing the edge, we get path value in M' ⇒ w^T from initial to final. So $w^T \in L(M')$

⇒ L(M') is regular.

Prof. Minakshi

Closure Properties of regular language:

① If L_1 and L_2 are two regular languages then.
$L_1 \cup$ (union) L_2 will also be regular.

Proof: L_1 and L_2 are two regular languages.
There exists two regular expressions r_1 and r_2 such that
$L_1 = L(r_1)$ ⇒ language represented by r_1
$L_2 = L(r_2)$ ⇒ Language represented (described by r_2)
By regular expression's definition, we know that
$r_1 + r_2$ is a regular expression.

Hence $L_1 \cup L_2$ or $L_1 + L_2$ will also be regular.

② If L_1 and L_2 are two regular languages, then $L_1 \cdot L_2$ (concatenation of L_1 and L_2) will also be regular.

There exists two regular expressions
r_1 and r_2 such that
$L_1 = L(r_1)$
$L_2 = L(r_2)$
According to the regular expression definition $r_1 \cdot r_2$ is also a regular expression the $L_1 \cdot L_2$ is also regular language.

③ If L is a regular language the L^* (Kleene closure of L is also a regular language.
If L is a regular expression, then, there exists a regular expression r such that
$L = L(r)$
By the definition of regular expression r^* (Kleene closure of r) is also regular expression. Hence L is regular language.

Union, Intersection and complements

- Suppose L_1 and L_2 are regular languages over an alphabet Σ.
- There are two FAs M_1 and M_2 accepting L_1 and L_2 respectively

$$L_1 = L(M_1)$$
$$L_2 = L(M_2)$$

- According to closure properties of regular languages
  $L_1 \cup L_2$, $L_1 \cdot L_2$ and L_1^* are also regular languages.

Suppose $M_1 = (Q_1, \Sigma, q_1, \delta_1, F_1)$ $\{q_1, q_2 \Rightarrow$ initial states
$M_2 = (Q_2, \Sigma, q_2, \delta_2, F_2)$ $\{F_1, F_2 \Rightarrow$ Final states

Let M be an FA defined by $M = (Q, \Sigma, q_0, \delta, F)$

- $Q = Q_1 \times Q_2$
- $q_0 = \{q_1, q_2\}$
- transition function δ defined by
  $\delta(\{p, q\}, a) = \{\delta_1(p, a), \delta_2(q, a)\}$
  for any $p \in Q_1$, $q \in Q_2$ and $a \in \Sigma$

(Union) • If $F = \{(p, q) \mid p \in F_1$ Or $q \in F_2\}$ M accepts $L_1 \cup L_2$

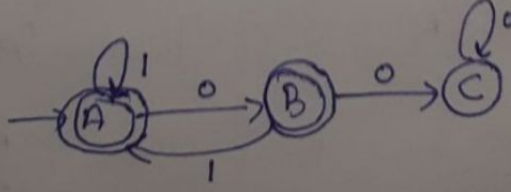
(Intersect) • If $F = \{(p, q) \mid p \in F_1$ and $q \in F_2\}$ M accepts $L_1 \cap L_2$

(Difference) • If $F = \{(p, q) \mid p \in F_1$ and $q \notin F_2\}$ M accepts $L_1 - L_2$

Example suppose $L_1 = \{x \mid 00$ is not a substring of $x\}$
$L_2 = \{x \mid x$ ends with $01\}$
$\Sigma = \{0, 1\}$

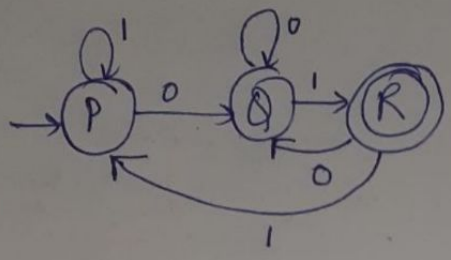
$M_1 \Rightarrow L_1$
$M_2 \Rightarrow L_2$

m_1



$Q_1 = \{A, B, C\}$
$F_1 = \{A, B\}$

M_2



$Q_2 = \{P, Q, R\}$

$F_2 = \{R\}$

- Transition Table of M_1

	0	1
→ (A)	B	A
(B)	C	A
C	C	C

Transition Table of M_2

	0	1
→ P	Q	P
Q	Q	R
(R)	Q	P

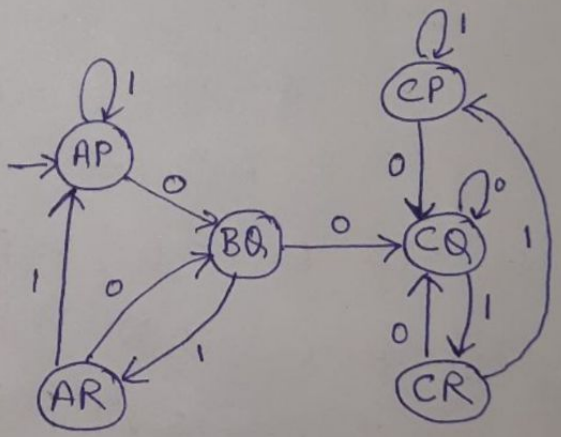
$M \Rightarrow (Q, \Sigma, \delta, q_0, F)$

$Q = Q_1 \times Q_2$

$q_0 \Rightarrow (A, P)$ → starting state of M_1, → starting state of M_2

- Transition Table for M

	0	1
→[A,P]	[B,Q]	[A,P]
[B,Q]	[C,Q]	[A,R]
[C,Q]	[C,Q]	[C,R]
[A,R]	[B,Q]	[A,P]
[C,R]	[C,Q]	[C,P]
[C,P]	[C,Q]	[C,P]



For union of M_1 and M_2 — Final states AP, BQ, AR, CR

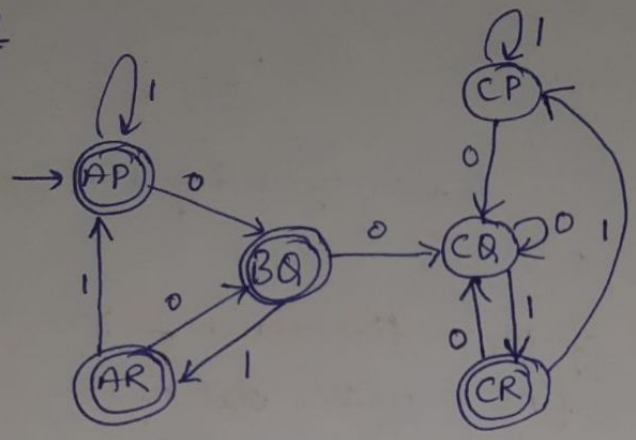
For intersection of M_1 and M_2 — Final states ⇒ (AR) $A \in F_1$ and $R \in F_2$

For difference $(L_1 - L_2)$ — Final states ⇒ AP, BQ

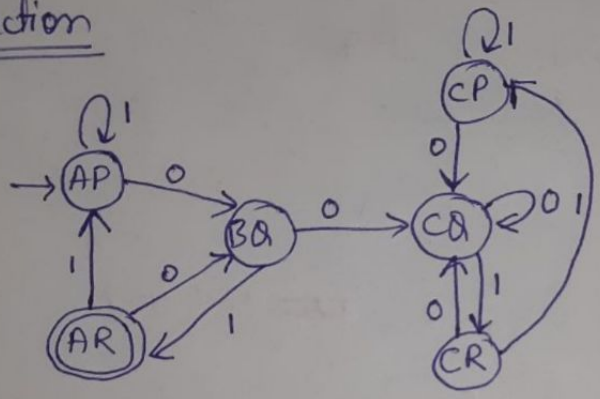
$A \in F_1$, $P \notin F_2$

$B \in F_1$, $Q \notin F_2$

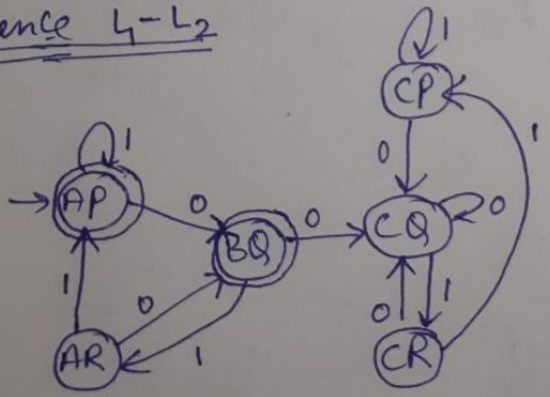
union



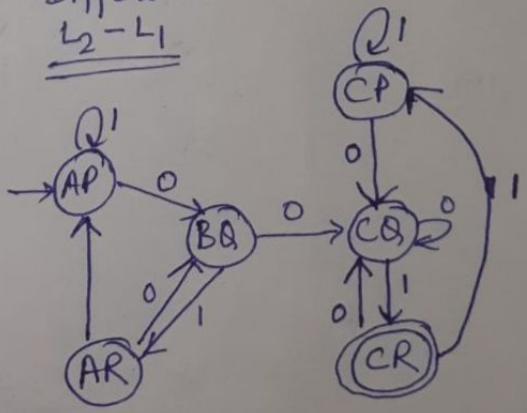
Intersection



Difference $L_1 - L_2$



Difference $L_2 - L_1$



Transpose

- If L is regular the L^T is also regular.

$L \Rightarrow \quad M = (Q, \Sigma, q_0, \delta, F)$ L is accepted by M.

$L^T \Rightarrow \quad M_1 = (Q, \Sigma, q_0', \delta', F')$

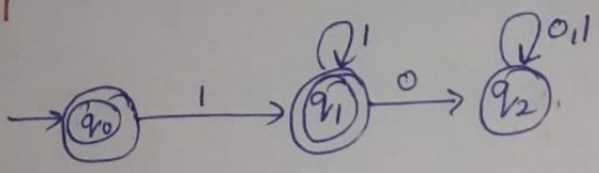
initial state of M_1 = final state of M

Final state of M_2 = initial state of M.

- Reverse the direction of edges
- convert final state of M to initial state
- convert initial state of M to final state

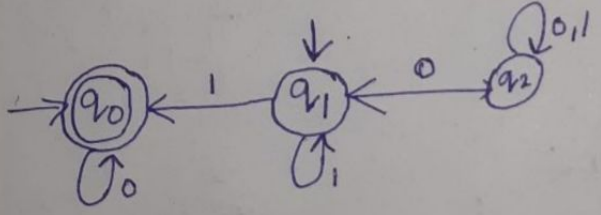
Example 1

M



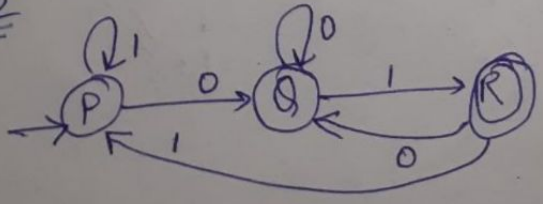
$L = L(M)$

M_1



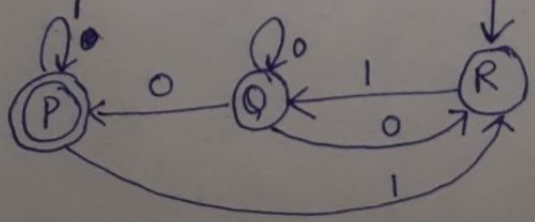
$\Rightarrow L^T \Rightarrow L(M_1)$

Example 2



L

⇓ Reversal or transpose.



L^T

complement with respect to Σ^*

if L is regular then $\Sigma^* - L$ is also regular.

- if L is regular then we can create a DFA $M = (Q, \Sigma, \delta, q_0, F)$ accepting L.

- we can construct another DFA $M_1 = (Q, \Sigma, \delta, q_0, F')$ by defining $F' = Q - F$

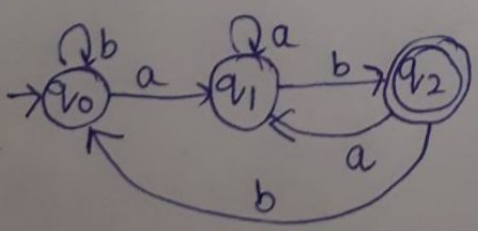
- The final state of M_1 is a non-final state of M and a non-final state of M_1 is a final state of M.

- $w \in L(M_1)$ if and only if

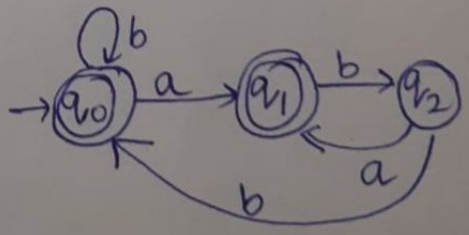
$$\delta'(q_0, w) \in F'$$
$$= Q - F$$
$$= \text{iff } w \notin L$$

Example

DFA for $L = \{ w \in \{a,b\}^* \mid w \text{ ends with sub string } ab \}$



complement $\Rightarrow$



DFA $\Rightarrow$
$L = \{ w \in \{a,b\}^* \mid w \text{ does not end with } ab \}$

Decision Properties of Regular Languages:-

- Emptiness (is L empty?)
- Finiteness (is L finite?)
- Membership (is $w \in L$?)
- Equivalence of two regular Languages ($L1 = L2$?)
- Subset (is $L1 \subseteq L2$?)

Emptiness

- is L empty?
- Checking whether the given language is empty or not.
- Proof by construction FA. if there is not final state in minimized FA. then the language accepted by FA is empty.

Finiteness:

- Checking whether the given language is finite or not
- Proof by construction of FA: if there is no cycle and self loop in minimized FA (excluding reject state, unreachable state or dead states) then the language is finite.

membership:

- checking whether the given string is member of language or belongs to a language.
- Proof by construction of FA: Run the string on DFA accepting language L. if w is the member of L, it will reach to final state otherwise it will reach to non-final state.

Equivalence of two regular languages:

- Checking whether two regular languages equivalent or not.
- Proof by construction of FAs and comparison method

- construct DFAs M_1 and M_2 respectively for L_1 and L_2.
- check the equivalence of DFAs using comparison method.
- if both the DFA's are equivalent, then the language is equivalent.

Subset ($L_1 \subseteq L_2$)

- checking whether the given language $L1$ is subset of given language $L2$ or not.

Proof by construction of FA

construct DFA1 and DFA2 to accept language $L1$ and $L2$ respectively if DFA1 $\cap$ complement(DFA2) accepts an empty language then $L1 \subseteq L2$ otherwise $L1$ is not subset of $L2$.

$L1 \subseteq L2$ if and only if $L1 \cap \text{complement}(L2) = \phi$

$L1 \cap \overline{L2} = \phi$