Regular Expressions

Unit 3

Regular Expression

Regular expressions are algebraic expressions used for representing **regular languages**; the languages accepted by finite automaton

Regular expressions offer a declarative way to express the strings we want to accept

Many systems use regular expression as input language; for eg, lexical analyzer generators such as LEX and FLEX

Regular Expression

A regular expression is built out of simpler regular expressions using a set of defining rules

Each regular expression ‘r’ denotes a language L(r)

The defining rules specify how L(r) is formed by combining, in various ways, the languages denoted by the sub expressions of ‘r’

Regular Expression

Let Σ be an alphabet; the regular expressions over the alphabet Σ are defined inductively as follows:

Basis steps:

- Ø is a regular expression representing empty language

- ε is a regular expression representing the language of empty strings. i.e. {ε}

- if ‘a’ is a symbol in Σ, then ‘a’ is a regular expression representing the language {a}

Regular Expression

Now the following operations over basic regular expressions define complex regular expressions as:

If ‘r’ and ‘s’ are the regular expressions representing the language L(r) and L(s) then

- r U s is a regular expression denoting the language L(r) U L(s)

- r . s is a regular expression denoting the language L(r) . L(s)

- r^* is a regular expression denoting the language $(L(r))^*$

- (r) is a regular expression denoting the language (L(r))

Regular Operators

There are three operators that are used to generate the languages that are regular

(1) Union (U / | /+): If L1 and L2 are two regular languages then,

$$L1 \cup L2 = \{s \mid s \in L1 \text{ or } s \in L2\}$$

eg:

if L1 = {00, 11} and L2 = {ε, 10}

then L1 U L2 = {ε, 00, 11, 10}

Regular Operators

(2) Concatenation (.): If L1 and L2 are two regular languages then,

$$L1 \,.\, L2 = \{l1 \,.\, l2 \mid l1 \in L1 \text{ and } l2 \in L2\}$$

eg:

if L1 = {00, 11} and L2 = {ε, 10}

then L1 . L2 = {00, 11, 0010, 1110}

and L2 . L1 = {00, 11, 1000, 1011} So, L1 . L2 !=L2 . L1

Regular Operators

(3) Kleene Closure (*): If L is a regular language then,

$$L^* = \bigcup_{i \geq 0} L^i = L^0 \cup L^1 \cup L^2 \cup \ldots\ldots$$

eg:

if L = {0, 11}

then $L^* = \{\varepsilon\} \cup \{0, 11\} \cup \{00, 011, 110, 1111\} \cup \ldots\ldots$

Regular Operators

We also define **positive closure** as $L^+ = L^* - L^0 = L^1 \cup L^2 \cup L^3 \cup \ldots\ldots$

Precedence of regular operators:

(1) Highest - Kleene Closure(*)

(2) Second highest - Concatenation(.)

(3) Lowest - Union(+)

Regular Expression

eg: Write a RE for the set of string that consists of alternating 0's and 1's over Σ = {0,1}

First part: Generate the language {01, 0101, 0101,……}

Second part: Generate the language {10, 1010, 101010,……}

Start with the basic regular expressions 0 and 1 that represent the language {0} and {1} respectively

Regular Expression

If we concatenate these two REs, we get the RE 0.1 that represents the language {01}

Then, to generate the language of zero or more occurrences of 01, we take Kleene closure i.e. the RE (01)* represents the language {01, 0101, ……}

Similarly, the RE for second part is (10)*

Finally, we take union of the above two to get the required RE i.e. the RE (01)*+(10)* represents the given language

Regular Language

Let Σ be an alphabet, the class of regular language over Σ is defined inductively as:

- Ø is a regular language representing empty language

- {ε} is a regular language representing language of empty strings

- For each a ε Σ, {a} is a regular language

- If L1, L2, ……, Ln are regular languages, then so is L1 U L2 U ……U Ln

- If L1, L2, ……, Ln are regular languages, then so is L1 . L2 . ……. . Ln

- If L is a regular language, then so is L*

Application of Regular Languages

Validation: Determining that a string complies with a set of formatting constraints. Like email address validation, password validation, etc.

Search and Selection: Identifying a subset of items from a larger set on the basis of a pattern match.

Tokenization: Converting a sequence of characters into words, tokens (like keywords, identifiers), for interpretation later.

Algebraic Rules for Regular Expression

(1) Commutativity

The union of regular expressions is commutative but concatenation is not

If r and s are regular expressions representing like languages L(r) and L(s) then,

$$r + s = s + r \quad \text{but} \quad r \,.\, s \neq s \,.\, r$$

Algebraic Rules for Regular Expression

(2) Associativity

The union as well as concatenation of regular expressions are associative

If t, r, s are regular expressions representing regular languages L(t), L(r) and L(s) then,

$$t + (r + s) = (t + r) + s$$

and, $$t \,.\, (r \,.\, s) = (t \,.\, r) \,.\, s$$

Algebraic Rules for Regular Expression

(3) Distributive Law

If r, s, t are regular expression representing regular languages L(r), L(s) and L(t) then,

$$r(s + t) = rs + rt \quad \rightarrow \text{left distribution}$$

$$(s + t)r = sr + tr \quad \rightarrow \text{right distribution}$$

Algebraic Rules for Regular Expression

(4) Identity Law

Ø is identity for union i.e. for any regular expression r representing regular language L(r),

$$r + Ø = Ø + r = r$$

ε is identity for concatenation

$$ε \,.\, r = r = r \,.\, ε$$

Algebraic Rules for Regular Expression

(5) Annihilator

An annihilator for an operator is a value such that when the operator is applied to the annihilator and some other value, the result is the annihilator

Ø is annihilator for concatenation

$$\text{i.e. } \emptyset \cdot r = r \cdot \emptyset = \emptyset$$

Algebraic Rules for Regular Expression

(6) Idempotent Law of Union

For any regular expression r representing the regular language L(r),

$$r + r = r$$

This is the idempotent law of union

Algebraic Rules for Regular Expression

(7) Law of Closure

For any regular expression r representing the regular language L(r),

- $(r^*)^* = r^*$
- Closure of $\emptyset = \emptyset^* = \varepsilon$
- Closure of $\varepsilon = \varepsilon^* = \varepsilon$
- Positive closure of r, $r^+ = rr^*$

Practice Questions

(Q) Consider Σ = {0, 1}.

(a) Write the RE that denotes the language of all string that begins with a ‘1’ and ends with a ‘0’.

(b) Write the RE that denotes the language of all strings that ends with 00 (binary multiples of 4).

(c) Write the RE that denotes the set of all strings that have alternating 1s and 0s.

Practice Questions

(d) Write the RE that denotes strings having exactly three 1's.

(e) Write RE tha denotes strings having at most two 0's.

(f) Write the RE that denotes strings having substring "001".

(g) Write the RE that denotes the language {w|w contains a single 1}.

(h) Write the RE for {w|w is a string of even length}.

(i) Write the RE for {w|w is a string containing even number of 0's}.

Practice Questions

Ans (a): $1(1 + 0)^*0$

Ans (b): $(1 + 0)^*00$

Ans (c): $(01)^* + (10)^*$

Ans (d): $0^*10^*10^*10^*$

Ans (e): $1^*(0 + \varepsilon)1^*(0 + \varepsilon)1^*$

Ans (f): $(1 + 0)^*001(1 + 0)^*$

Ans (g): 0^*10^*

Ans (h): $((0+1).(0+1))^*$

Ans (i): $(1^*01^*01^*)^*$

Equivalence of RE and Finite Automata

Regular expression approach to describing languages is fundamentally different from the finite automaton approach

However, these two notations turn out to represent exactly the same set of languages i.e. **regular languages**

We have seen that DFA, NFA and ε-NFA accept the same class of languages

Equivalence of RE and Finite Automata

In order to show that the regular expressions define the same class of language as finite automata, we show that:

(1) Every language defined by one of these automata is also defined by a regular expression (We will convert DFA to RE)

(2) Every language defined by a regular expression is defined by one of these automata (We will convert RE to ε-NFA)

Equivalence of RE and Finite Automata

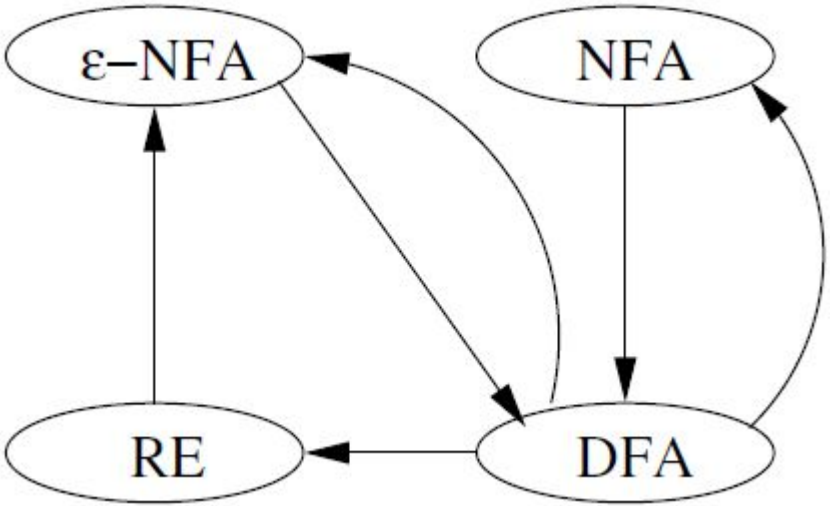


fig: Plan for showing equivalence of four different notations for regular languages

Regular Expression to ε-NFA

Theorem: Every language defined by a regular expression is also defined by a finite automaton

For any regular expression r, there is an ε-NFA that accepts the same language as r

This say that both RE and ε-NFA are equivalent in terms of language representation

Regular Expression to ε-NFA

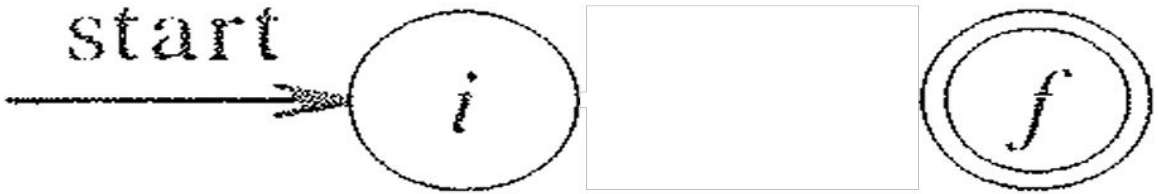
Proof: Let L(r) be the language for regular expression r, now we have to show there is an ε-NFA E such that L(E) =L(r)

The proof can be done through structural induction on r, following the recursive definition of regular expressions

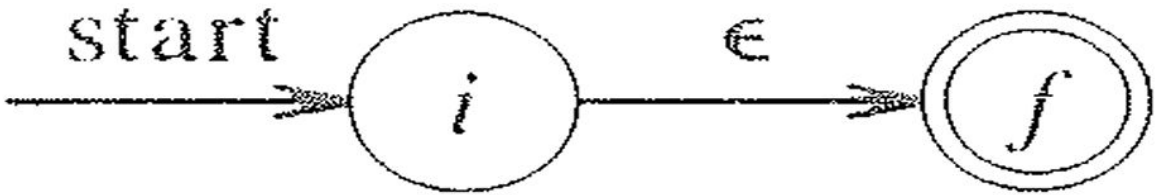
We know, Ø, ε, ‘a’ are the regular expressions representing an empty language, language for empty strings and the language {a}, respectively

Basis: The ε-NFA accepting these languages can be constructed as:

(i) For r = Ø

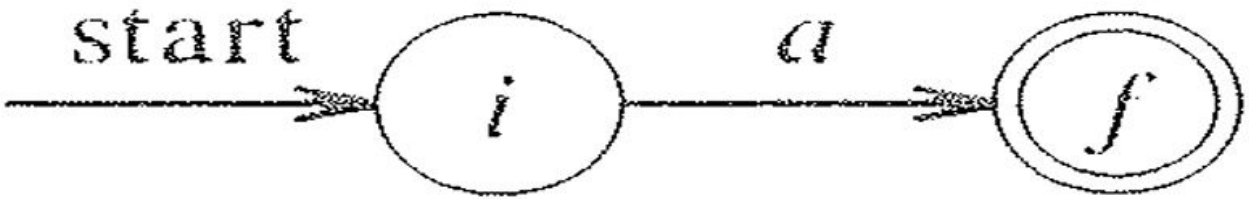


(ii) For r = ε



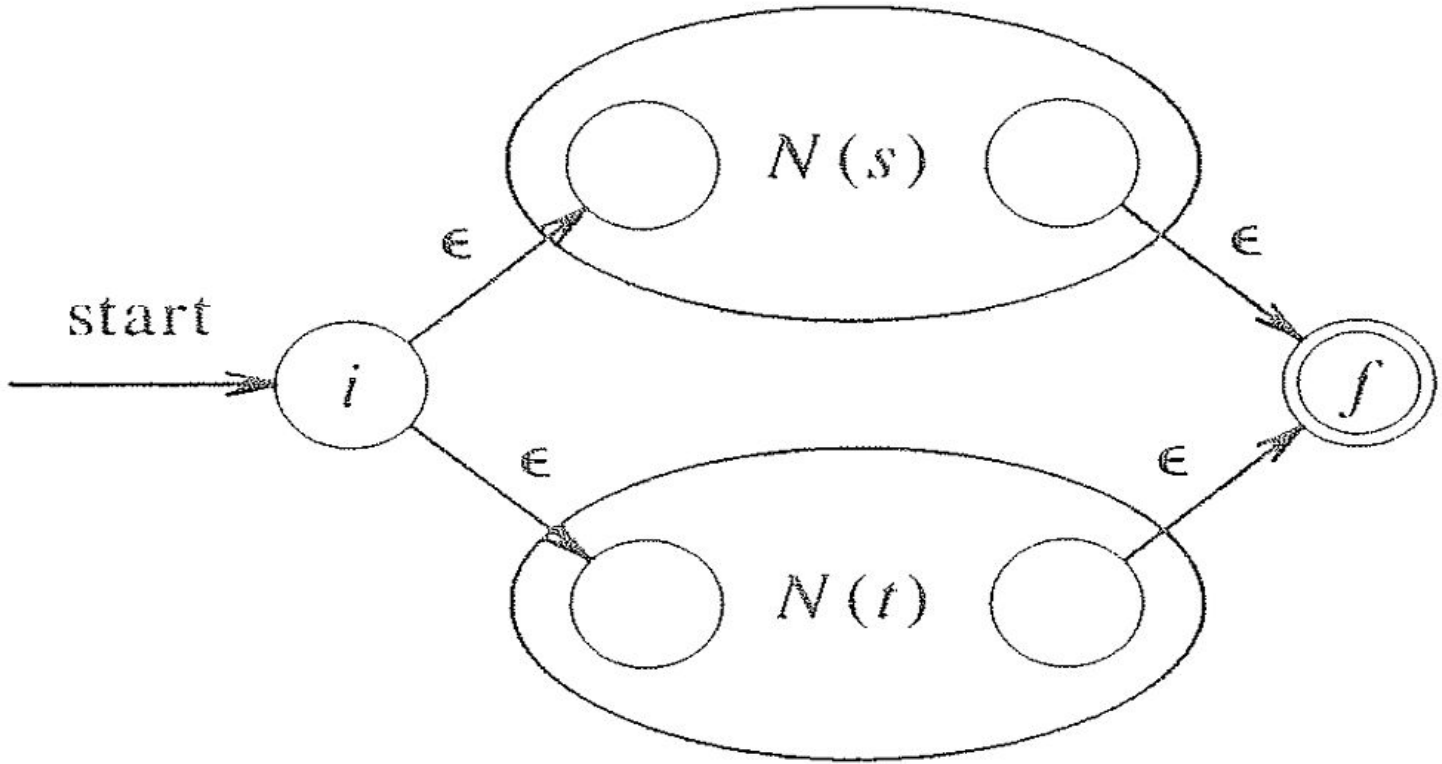
Regular Expression to ε-NFA

(iii) for r = a



Induction: Suppose N(s) and N (t) are ε-NFAs for RE s and t, respectively

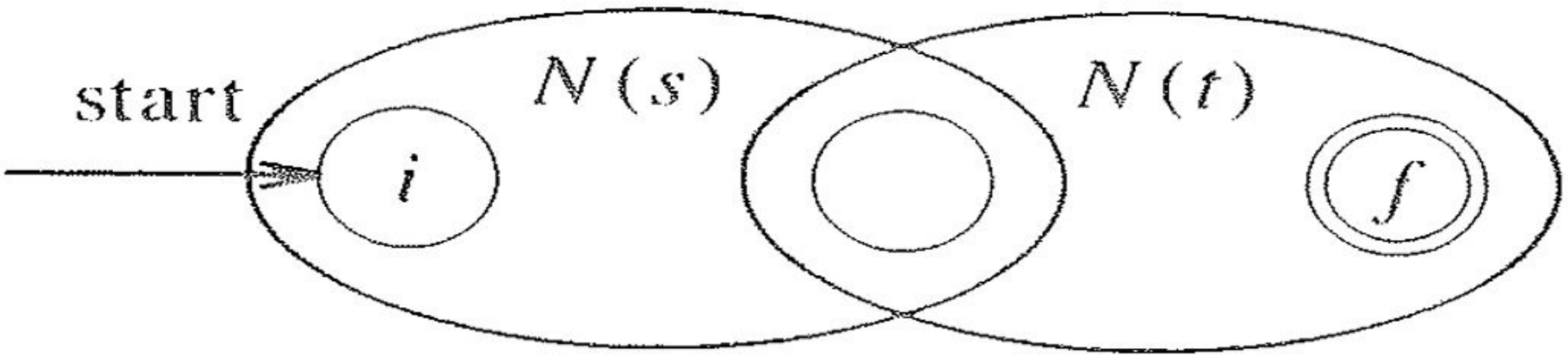
(i) For the regular expression s U t, [here, L(s U t) = L(N(s) U N(t))]



Regular Expression to ε-NFA

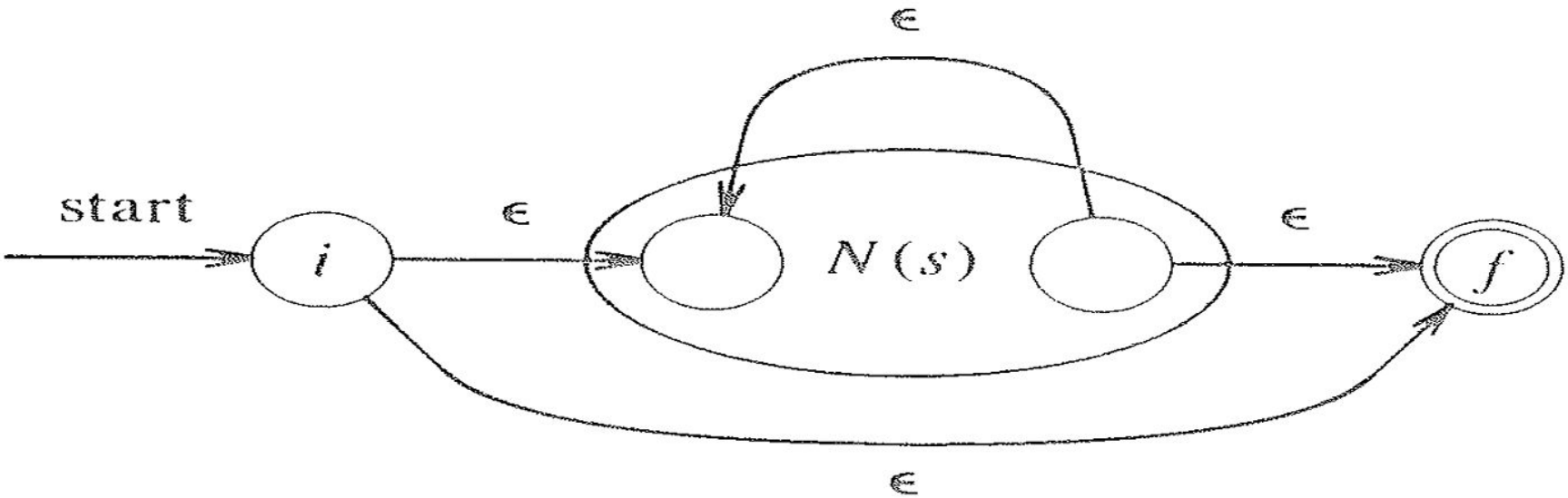
(ii) For the regular expression st, [here, L(st) = L(N(s) . N(t))]

Here, there is an ε-transition from old final state of N(s) to old start state of N(t)



Regular Expression to ε-NFA

(iii) For the regular expression s*, [here, L(s*) = L((N(s))*)]



Regular Expression to ε-NFA

(iv) Finally, for regular expression (r), the automaton for r also serves as the automaton for (r), since the parentheses do not change the language defined by the expression

This completes the proof

Regular Expression to ε-NFA

Thompson's Construction is the method used for converting RE to ε-NFA

It guarantees that the resulting NFA will have exactly one final state and one start state

Regular Expression to ε-NFA

Steps:

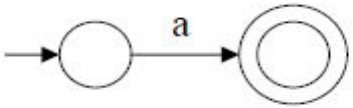
(1) First, parse the regular expression into sub expressions

(2) Construct ε-NFAs for each of the basic symbols in regular expression (using basis from the above theorem)

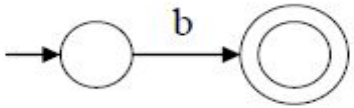
(3) Finally, combine all ε-NFAs of sub expressions to get required ε-NFA of given regular expression (using induction of the theorem)

Example:- NFA construction of RE **(a+b)** * **a**

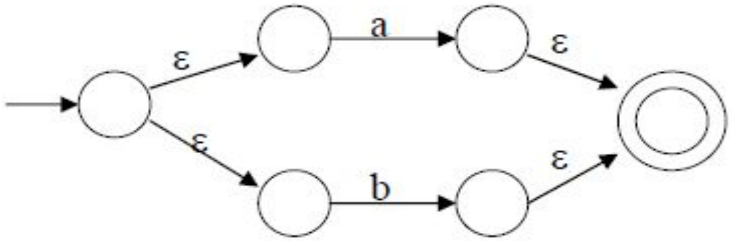
a:



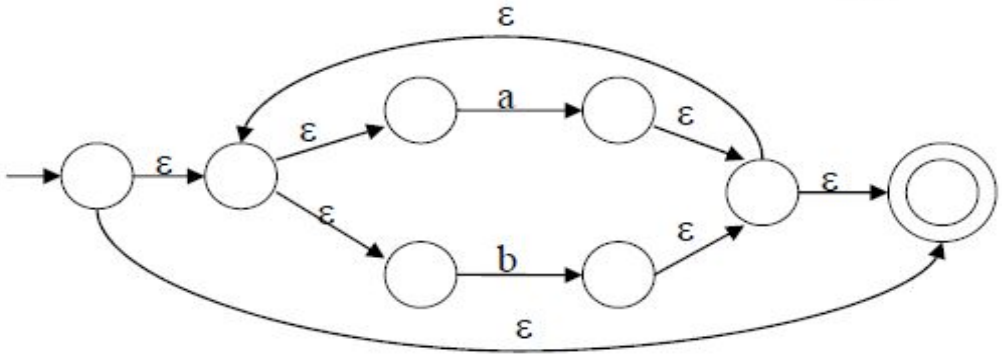
b:



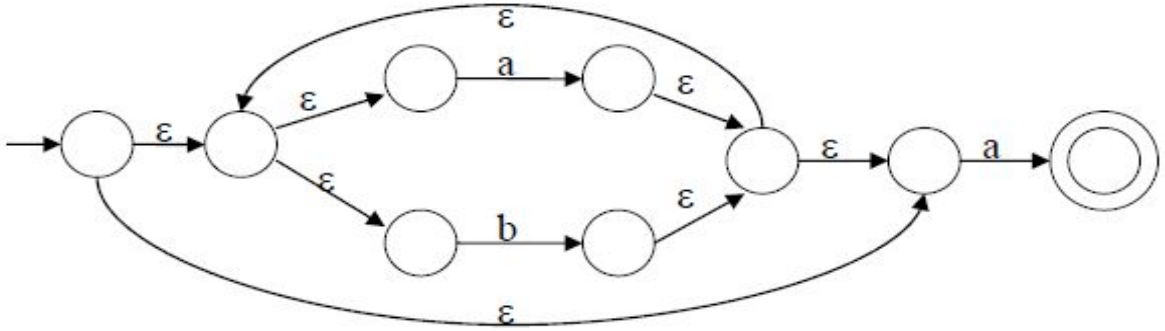
(a + b)



(a + b) *



(a+b) * a



Practice Questions

(Q) Convert the following RE to equivalent finite automata:

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

(b) $(00 + \varepsilon)^*10$

DFA to Regular Expression: Arden's Theorem

Let p and q be the regular expressions over the alphabet Σ, if p does not contain an empty string then

$$r = q + rp$$

has a unique solution

$$r = qp^*$$

DFA to Regular Expression: Arden's Theorem

Proof: Here, $r = q + rp$ .....(i)

Putting the value of $r = q + rp$ on the right hand side (i),

$$r = q + (q + rp)p$$

$$r = q + qp + rp^2 \text{.....(ii)}$$

Again, repeating the process on (ii),

$$r = q + qp + (q + rp)p^2$$

$$r = q + qp + qp^2 + rp^3$$

DFA to Regular Expression: Arden's Theorem

Continuing in the same way, we will get

$$r = q + qp + qp^2 + qp^3 \ldots\ldots$$

$$r = q(\varepsilon + p + p^2 + p^3 + \ldots\ldots)$$

$$r = qp^*$$

Hence, proved

Use of Arden's rule to find RE for DFA

To convert the given DFA into a RE, there are some assumptions regarding the transition system:

- The transition diagram should not have ε-transitions

- There must only be one initial state

- The vertices i.e. states in the DFA are as

$$q_1, q_2, \ldots\ldots, q_n \text{ (Any } q_i \text{ is final state)}$$

w_{ij} denotes the regular expression representing the set of labels of the edges from q_i to q_j

Then we write a system of equations as:

$$q_1 = q_1 w_{11} + q_2 w_{21} + q_3 w_{31} + \ldots\ldots + q_n w_{n1} + \varepsilon$$

[ε is added to start state's equation]

$$q_2 = q_1 w_{12} + q_2 w_{22} + q_3 w_{32} + \ldots\ldots + q_n w_{n2}$$

$$q_3 = q_1 w_{13} + q_2 w_{23} + q_3 w_{33} + \ldots\ldots + q_n w_{n3}$$

……

$$q_n = q_1 w_{1n} + q_2 w_{n2} + q_3 w_{n3} + \ldots\ldots + q_n w_{nn}$$

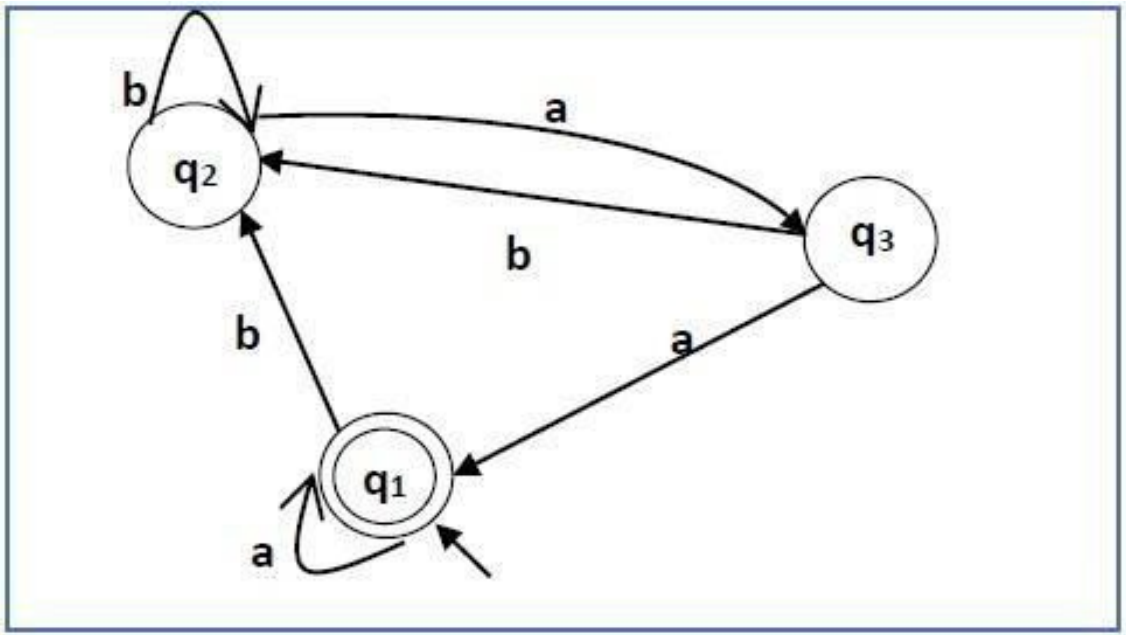
Use of Arden's rule to find RE for DFA

Solving these equations for q_i in terms of w_{ij} gives the regular expression equivalent to given DFA

The required regular expression is obtained from the equations of the final states

Example

(Q) Construct a regular expression for the automata given below.



Example

The equations for the three states q_1, q_2, and q_3 are as follows:

$q_1 = q_1a + q_3a + \varepsilon$ (ε move is because q1 is the initial state.

$q_2 = q_1b + q_2b + q_3b$

$q_3 = q_2a$

Example

Now, we will solve these three equations –

$$q_2 = q_1b + q_2b + q_3b$$

$$= q_1b + q_2b + (q_2a)b \qquad \text{(Substituting value of } q_3\text{)}$$

$$= q_1b + q_2(b + ab)$$

$$= q_1b\,(b + ab)^* \qquad \text{(Applying Arden's Theorem)}$$

$q_1 = q_1a + q_3a + \varepsilon$

$= q_1a + q_2aa + \varepsilon$ (Substituting value of q_3)

$= q_1a + q_1b(b + ab^*)aa + \varepsilon$ (Substituting value of q_2)

$= q_1(a + b(b + ab)^*aa) + \varepsilon$

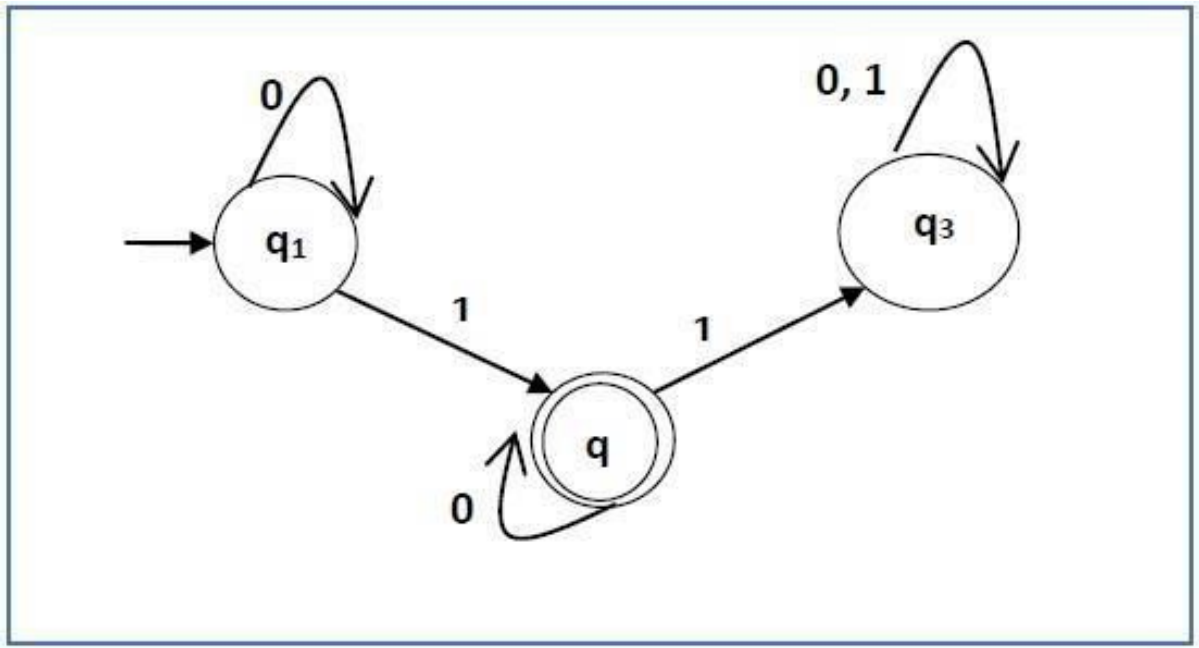
$= \varepsilon\ (a+ b(b + ab)^*aa)^*$

$= (a + b(b + ab)^*aa)^*$

Hence, the regular expression is $(a + b(b + ab)^*aa)^*$.

Practice Question

(Q1) Construct a regular expression for the automata given below.



Solution

Here, q1 is the start state and q2 is the final state

The equations for the states are as follows:

$$q_1 = q_1 0 + \varepsilon$$

$$q_2 = q_1 1 + q_2 0$$

$$q_3 = q_2 1 + q_3 0 + q_3 1$$

Now, we will solve these three equations –

$$q_1 = \varepsilon 0^* \text{ [As, } \varepsilon R = R]$$

So, $q_1 = 0^*$

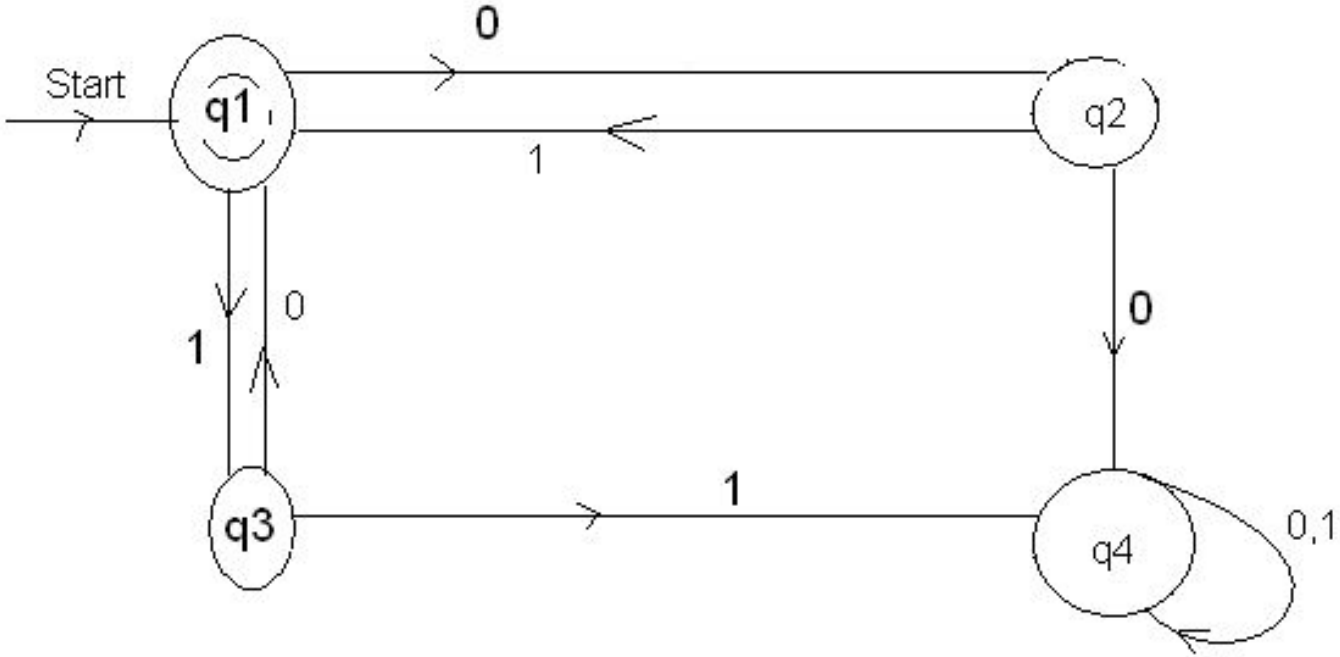
$$q_2 = 0^*1 + q_2 0$$

So, $q_2 = 0^*1(0)^*$ [By Arden's theorem]

Hence, the required regular expression is 0^*10^*

Practice Question

(Q2) Construct a regular expression for the automata given below.



Solution

The equations are:

$$q_1 = q_2 1 + q_3 0 + \varepsilon \ldots\ldots(i)$$

$$q_2 = q_1 0 \ldots\ldots(ii)$$

$$q_3 = q_1 1 \ldots\ldots(iii)$$

$$q_4 = q_2 0 + q_3 1 + q_4 0 + q_4 1 \ldots\ldots(iv)$$

Solution

Putting the values of q_2 and q_3 in (i)

$$q_1 = q_1 01 + q_1 10 + \varepsilon$$

i.e. $q_1 = q_1(01+10) + \varepsilon$

i.e. $q_1 = \varepsilon + q_1(01+10)$ [here, r = q+rp]

i.e. $q_1 = \varepsilon(01+10)^*$ [using Arden's rule]

Since q_1 is final state, the final regular expression for the DFA is

$$\varepsilon(01+10)^* = (01+10)^*$$

Closure Properties of Regular Languages

(1) Closure under Union

If L1 and L2 are two regular languages, their union L1 U L2 will also be regular

Let L1 and L2 be the languages of RE R and S respectively, then R+S is a regular expression whose language is L1 U L2

eg: L1 = {an | n > 0} and L2 = {bn | n > 0}

L3 = L1 U L2 = {an U bn | n > 0} is also regular

Closure Properties of Regular Languages

(2) Closure under Intersection

If L1 and L2 are two regular languages, their intersection L1 ∩ L2 will also be regular

Let L1 and L2 be the languages of RE R and S respectively, then R ∩ S is a regular expression whose language is L1 ∩ L2

eg: L1 = {ambn | n > 0 & m > 0} and L2= {ambn U bnam | n > 0 & m > 0}

L3 = L1 ∩ L2 = {ambn | n > 0 and m > 0} is also regular

Closure Properties of Regular Languages

(3) Closure under Concatenation

If L1 and L2 are two regular languages, their concatenation L1.L2 will also be regular

Let L1 and L2 be the languages of RE R and S respectively, then R.S is a regular expression whose language is L1.L2

eg: L1 = {an | n > 0} and L2 = {bn | n > 0}

L3 = L1.L2 = {am . bn | m > 0 and n > 0} is also regular

Closure Properties of Regular Languages

(4) Closure under Kleene Closure

If L is a regular language, its kleene closure L^* will also be regular

Let L be the language of RE R, then R^* is a regular expression whose language is L^*

Closure Properties of Regular Languages

(5) Closure under Complement

The complement of a language L (with respect to an alphabet Σ such that Σ^* contains L) is $\Sigma^* - L$

Since Σ^* is surely regular by the property of closure under kleene closure, the complement of a regular language is always regular

To complement a FA, we make the accepting states non-accepting and vice versa

Minimization of DFA: Table Filling Algorithm

Given a DFA M that accepts a language L(M), minimization of M is the process of finding an equivalent DFA M' that has the minimum number of states

The process of minimization involves identifying **equivalent states** and **distinguishable states**

Minimization of DFA: Table Filling Algorithm

Equivalent states: Two states p & q are called equivalent states, denoted by p ≡ q, if and only if for each input string x, $\hat{\delta}(p, x)$ is a final state if and only if $\hat{\delta}(q, x)$ is a final state

Distinguishable states: Two states p & q are said to be distinguishable states if there exists at least one string x, such that if $\hat{\delta}(p, x)$ is a final state, $\hat{\delta}(q, x)$ is not a final state or vice-versa

For minimization, the **table filling algorithm** is used

The steps of the algorithm are:

(1) Draw a table for all pairs of states (Qi, Qj) of the given DFA[all are unmarked initially].

(2) Consider every state pair (Qi, Qj) in the DFA where Qi ∈ F and Qj ∉ F or vice versa and mark them[here, F = set of final states].

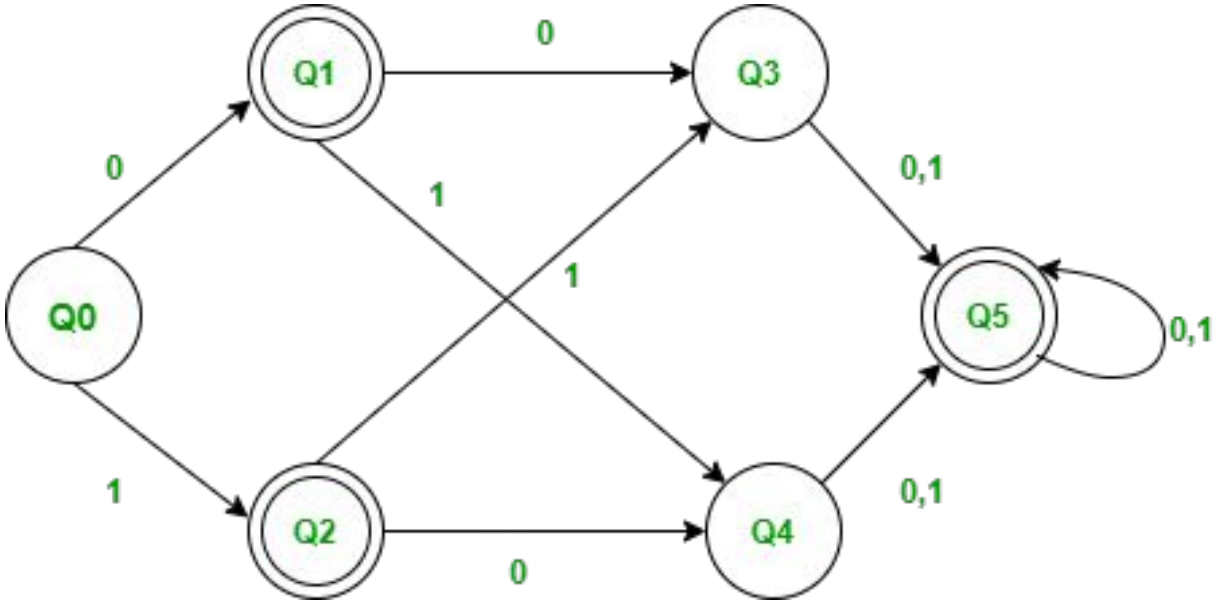
(3) Repeat this step until we cannot mark any more states:

If there is an unmarked pair (Qi, Qj), mark it if the pair {δ(Qi, a), δ(Qj, a)} is marked for some input alphabet a.

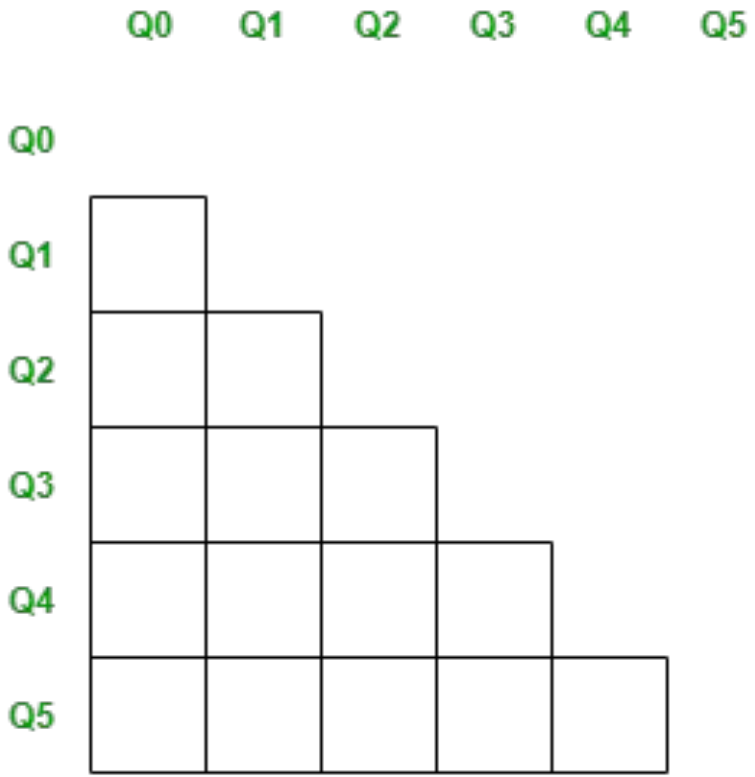
(4) Combine all unmarked pairs (Qi, Qj) and make them a single state in the reduced DFA.

Example

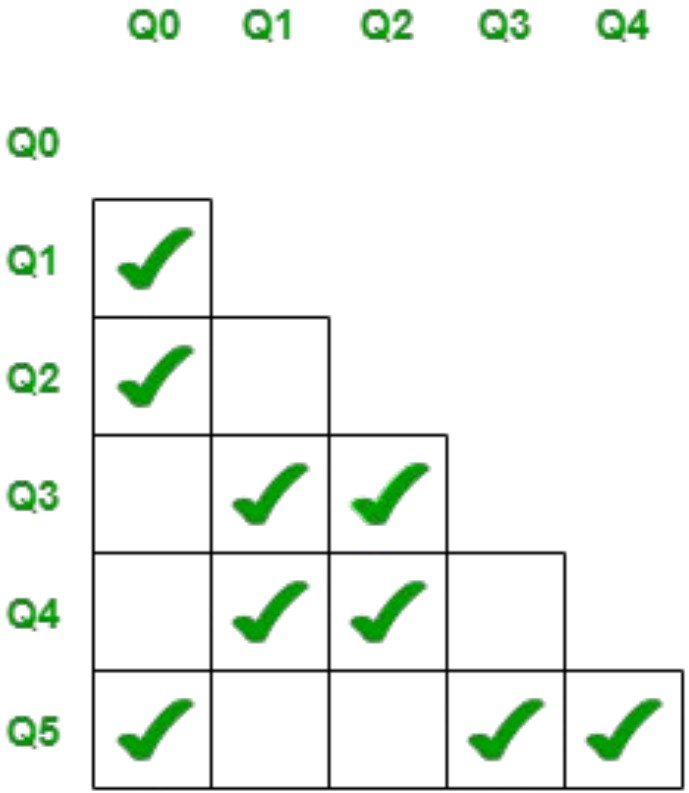
(Q) Minimize the following DFA using table filling algorithm:



(1) Create the pairs of all the states involved in DFA



(2) Mark all the pairs (Qa, Qb) such that Qa is a final state and Qb is a non-final State or vice-versa



Example

(3) For all unmarked pairs, check transition on input symbols to see if they can be marked

(i) Check for the unmarked pair (Q2, Q1)

$\delta(Q2, 0) = Q4$ and $\delta(Q1, 0) = Q3 \Rightarrow (Q4, Q3)$ - unmarked

$\delta(Q2, 1) = Q3$ and $\delta(Q1, 1) = Q4 \Rightarrow (Q4, Q3)$ - unmarked

(ii) Check for the unmarked pair (Q3, Q0)

$\delta(Q3, 0) = Q5$ and $\delta(Q0, 0) = Q1 \Rightarrow (Q5, Q1)$ - unmarked

$\delta(Q3, 1) = Q5$ and $\delta(Q0, 1) = Q2 \Rightarrow (Q5, Q2)$ - unmarked

Example

(iii) Check for the unmarked pair (Q4, Q0)

δ(Q4, 0) = Q5 and δ(Q0, 0) = Q1 ⇒ (Q5, Q1) - unmarked

δ(Q4, 1) = Q5 and δ(Q0, 1) = Q2 ⇒ (Q5, Q2) - unmarked

(iv) Check for the unmarked pair (Q4, Q3)

δ(Q4, 0) = Q5 and δ(Q3, 0) = Q5 ⇒ (Q5, Q5) - doesn't exist

δ(Q4, 1) = Q5 and δ(Q3, 1) = Q5 ⇒ (Q5, Q5) - doesn't exist

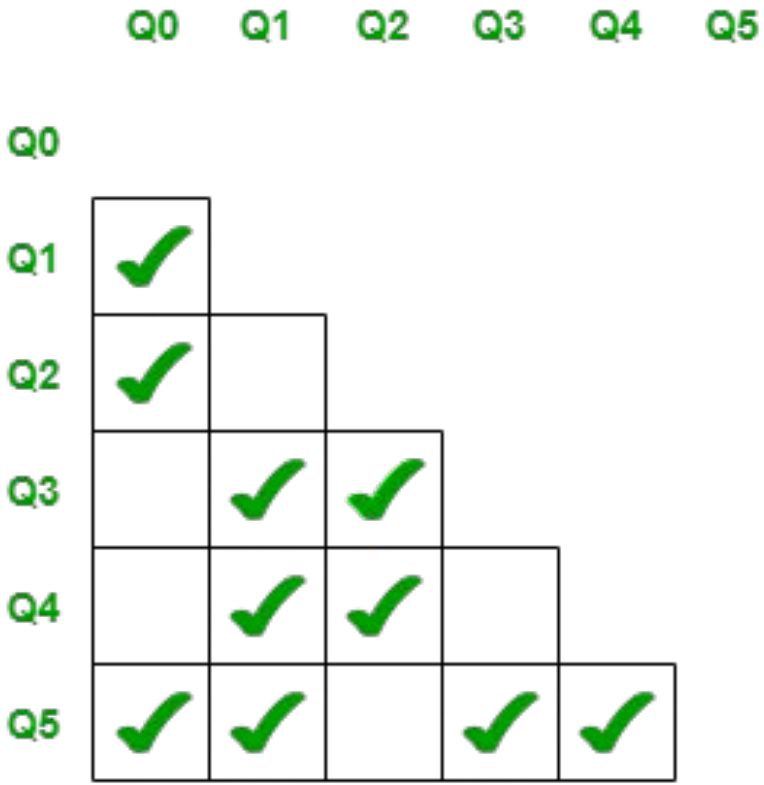
Example

(v) Check for the unmarked pair (Q5, Q1)

$\delta(Q5, 0) = Q5$ and $\delta(Q1, 0) = Q3 \Rightarrow (Q5, Q3)$ - marked

Hence, we can mark pair (Q5, Q1) i.e. they are distinguishable

Example



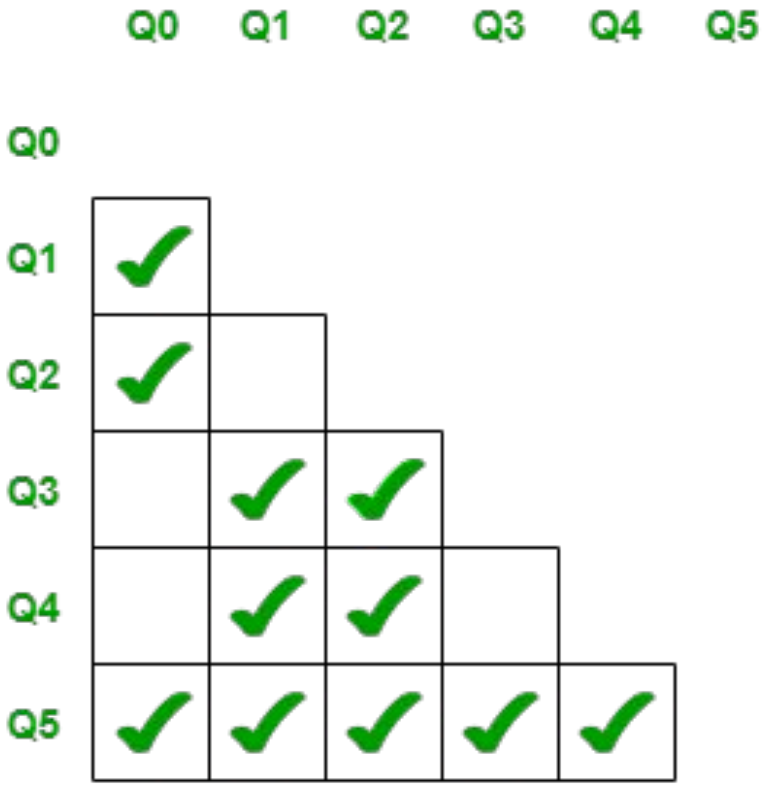
Example

(vi) Check for the unmarked pair (Q5, Q2)

$\delta(Q5, 0) = Q5$ and $\delta(Q2, 0) = Q4 \Rightarrow (Q5, Q4)$ - marked

Hence, we can mark pair (Q5, Q2)

Example



Example

All unmarked pairs have been checked but since new markings were made, we repeat the process until no more markings can be made

(vii) Check for the unmarked pair (Q2, Q1)

$\delta(Q2, 0) = Q4$ and $\delta(Q1, 0) = Q3 \Rightarrow (Q4, Q3)$ - unmarked

$\delta(Q2, 1) = Q3$ and $\delta(Q1, 1) = Q4 \Rightarrow (Q4, Q3)$ - unmarked

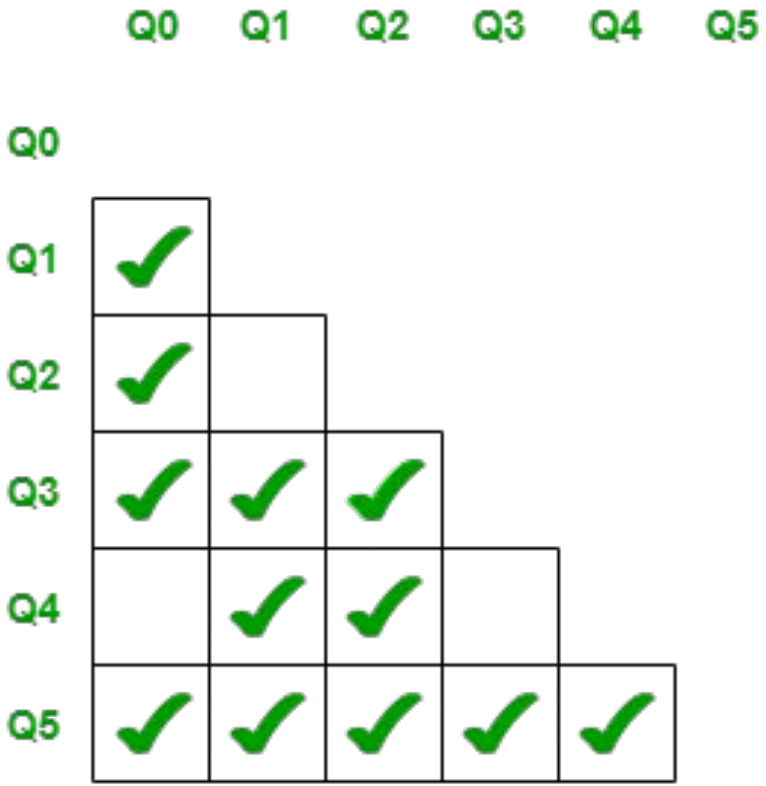
Example

(viii) Check for the unmarked pair (Q3, Q0)

$\delta(Q3, 0) = Q5$ and $\delta(Q0, 0) = Q1 \Rightarrow (Q5, Q1)$ - marked

Hence, we can mark pair (Q3, Q0)

Example



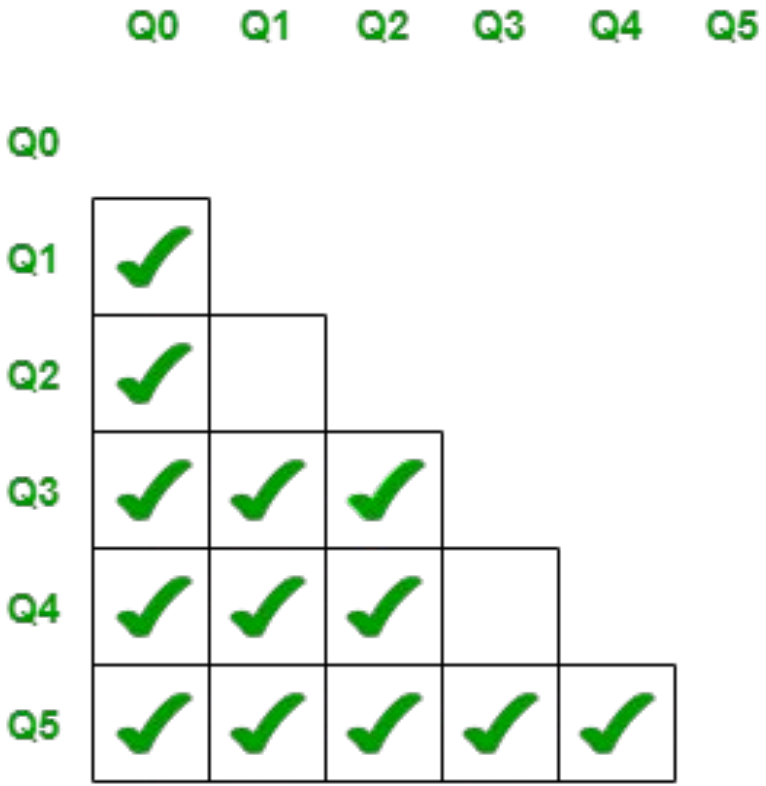
Example

(ix) Check for the unmarked pair (Q4, Q0)

$\delta(Q4, 0) = Q5$ and $\delta(Q0, 0) = Q1 \Rightarrow (Q5, Q1)$ - marked

Hence, we can mark pair (Q4, Q0)

Example



Example

(x) Check for the unmarked pair (Q4, Q3)

$\delta(Q4, 0) = Q5$ and $\delta(Q3, 0) = Q5 \Rightarrow (Q5, Q5)$ - doesn't exist

$\delta(Q4, 1) = Q5$ and $\delta(Q3, 1) = Q5 \Rightarrow (Q5, Q5)$ - doesn't exist

Now even though we repeat the procedure, we cannot mark the pairs (Q2, Q1)[since (Q4, Q3) is not marked] and (Q4, Q3)[since (Q5,Q5) doesn't exist). Hence we stop here.

(4) Combine all unmarked pairs and make them a single state in the minimized DFA

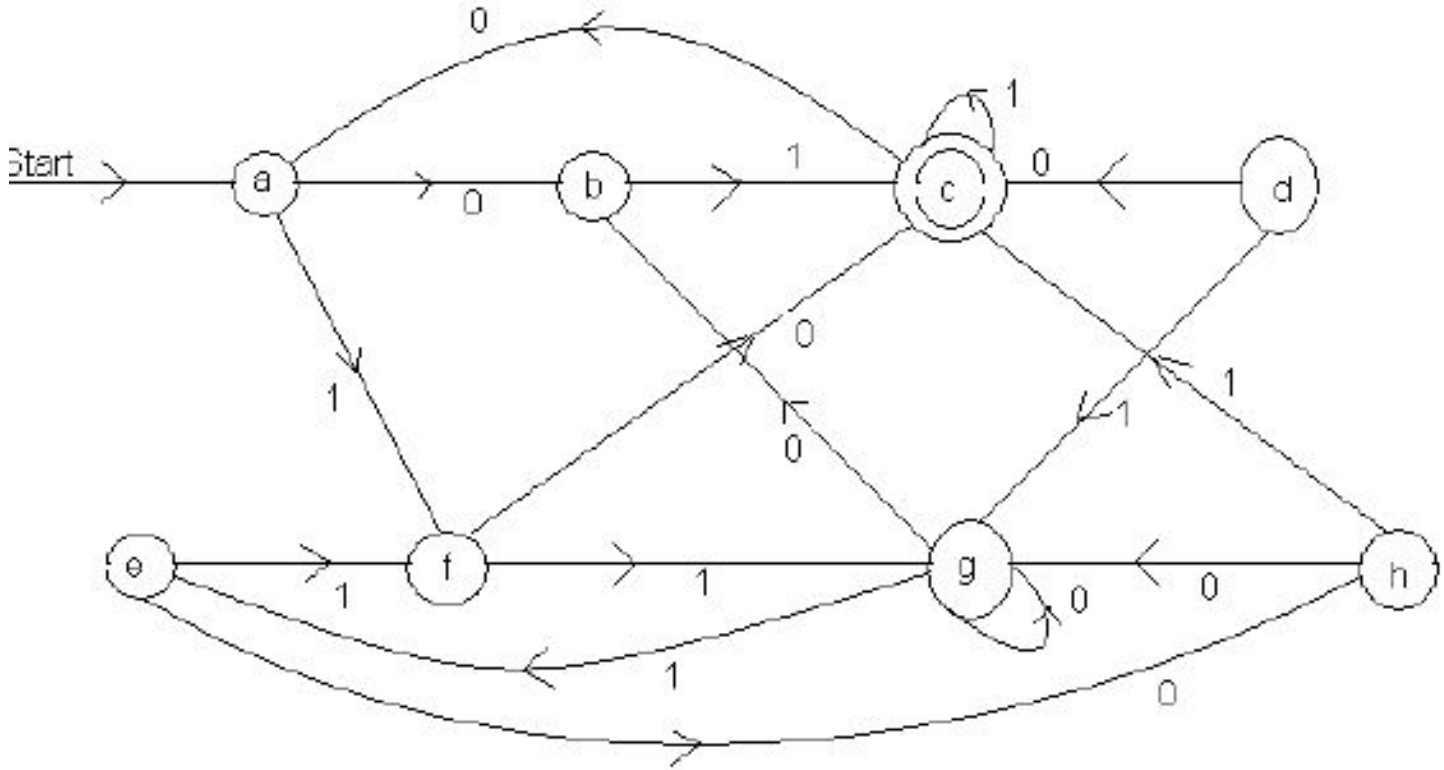
The unmarked pairs are (Q2, Q1) and (Q4, Q3), we combine them

This is the minimized DFA

Q0 remains as starting state, Q5 remains as final state, the pair (Q3,Q4) is a non-final state and (Q1, Q2) is another final state.

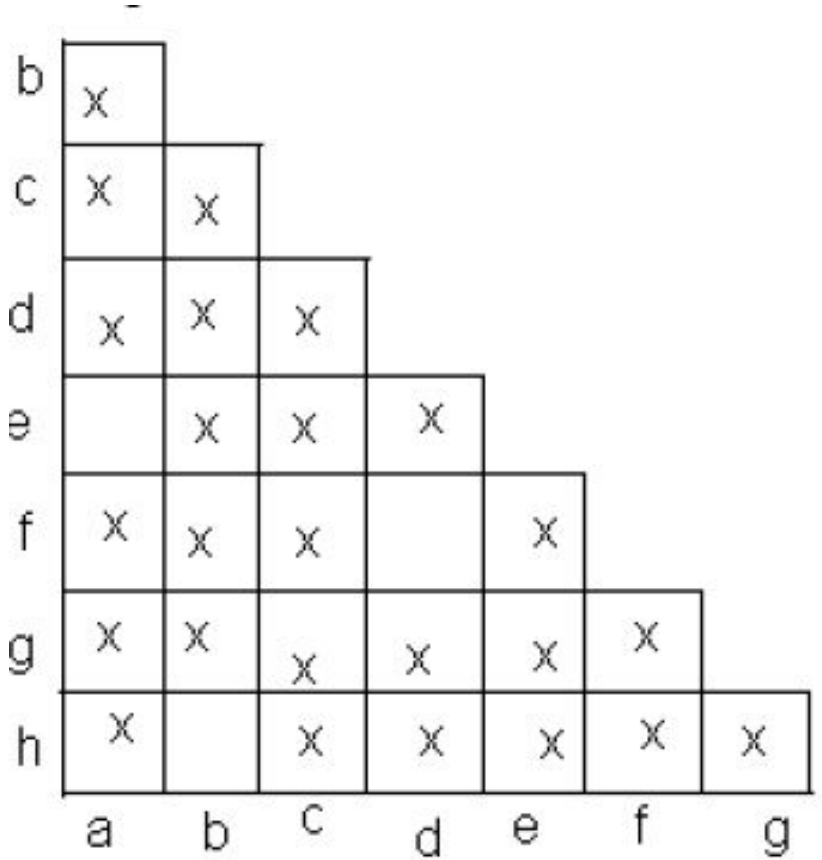
Practice Question

(Q) Minimize this DFA:

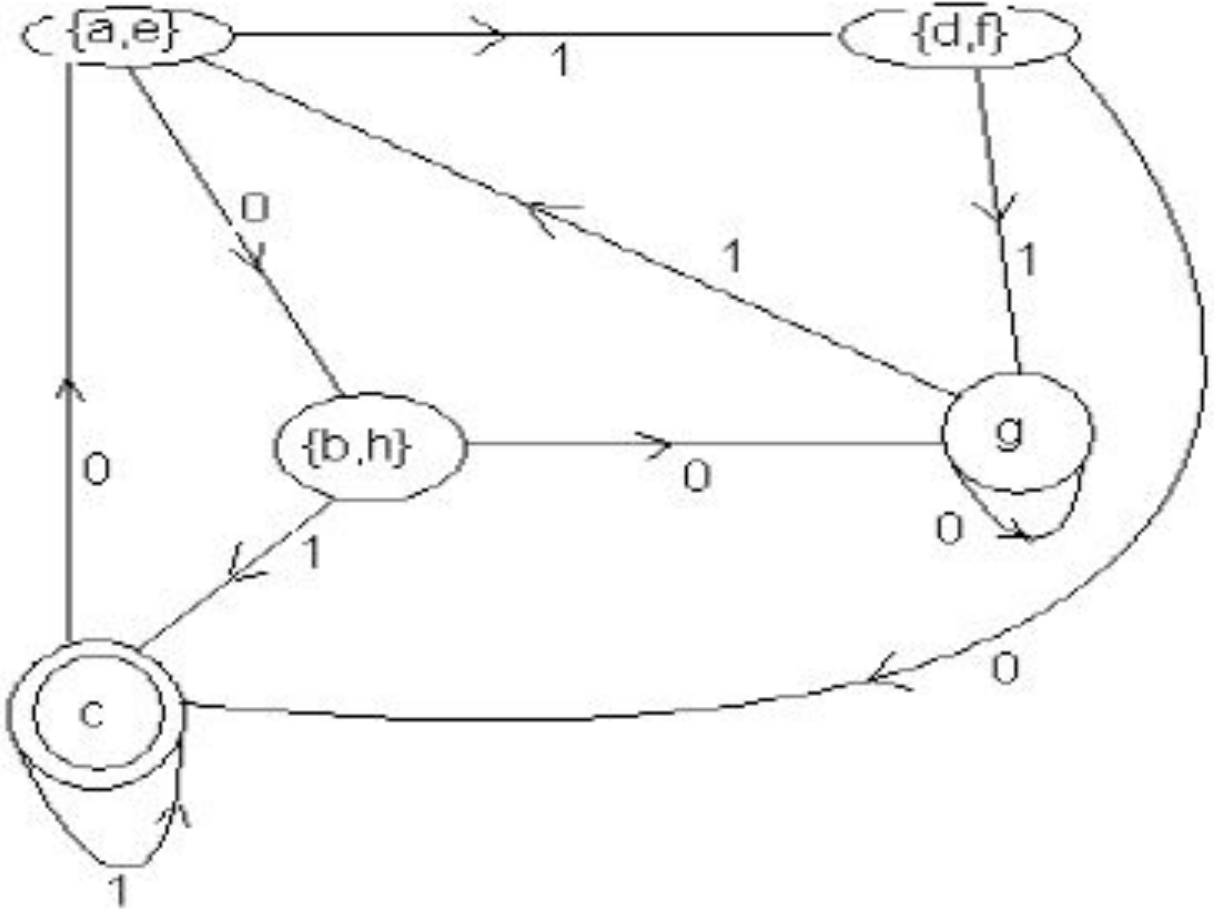


Ans:

Final Table:



Minimized DFA:



Proving Non-regularity of a Language

We know that the class of language known as regular language has at least four different descriptions

They are the language accepted by DFAs, by NFAs, by ε-NFAs, and defined by RE

Not every language is regular; **pumping lemma** is a powerful technique used to show that a language is not regular

Pumping Lemma

Statement: Let L be a regular language. Then, there exists an integer constant n (pumping length) such that for any $x \in L$ with $|x| \geq n$, there are strings u, v, w such that

$$x = uvw; \qquad |uv| \leq n, |v| > 0$$

and,

$$uv^k w \in L \quad \text{for all } k \geq 0.$$

(Here, 'v' is the string that can be pumped i.e repeating 'v' any number of times, or deleting it, keeps the resulting string in the language)

Proof: Suppose L is a regular language, then L is accepted by some DFA M. Let M has n states.

We take some string x ∈ L of length n or greater. Let length of x, |x| = m where m ≥ n.

Suppose $x = a_1a_2a_3\ldots\ldots a_m$ where each $a_i \in \Sigma$ be an input symbol to M. Now, consider q_j be states of M, for j =1,..., n.

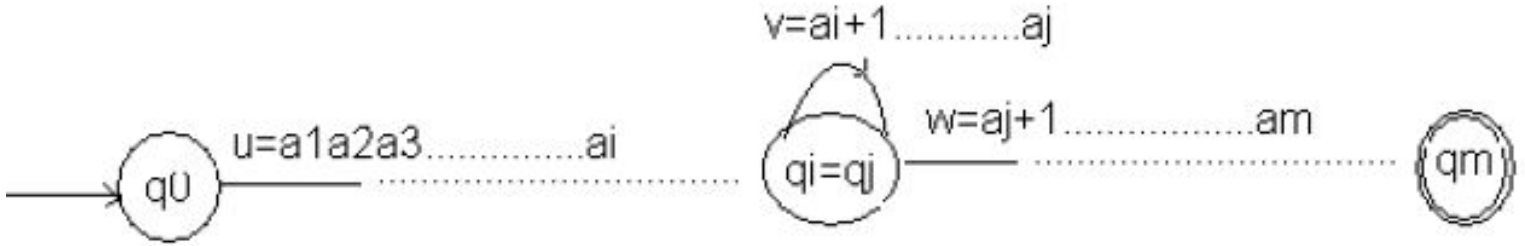
Then, $(q_0, x) = \hat{\delta}(q_0, a_1a_2\ldots a_m)$ [q_0 being start state of M]

$= \hat{\delta}(q_1, a_2\ldots a_m)$

$= \ldots\ldots\ldots\ldots$

$= (q_m, \varepsilon)$ [q_m being final state]

Since m ≥ n, and DFA M has n states only, by pigeonhole principle, there exists some i and j; $0 \le i < j \le m$ such that $q_i = q_j$



Now, we can break x=uvw as

$u = a_1 a_2 \ldots\ldots a_i$

$v = a_{i+1} \ldots\ldots a_j$

$w = a_{j+1} \ldots\ldots a_m$

Pumping Lemma

That is, string $a_{i+1}.....a_j$ takes M from state q_i back to itself since $q_i=q_j$

So we can say M accepts $a_1a_2...a_i(a_{i+1}...a_j)^ka_{j+1}...a_m$ for all $k \geq 0$

Hence, $uv^kw \in L$ for all $k \geq 0$

Application of Pumping Lemma:

To prove any language is not a regular language. Proof by contradiction is used for this purpose.

Pumping Lemma

eg: Show that language, $L=\{0^n 1^n \mid n \geq 0\}$ is not a regular language.

Solution:

Using proof by contradiction, let L is a regular language

Then by pumping lemma, there are strings u, v, w with $v \geq 1$ such that $uv^kw \in L$ for $k \geq 0$

Pumping Lemma

Case 1: Let v contain 0's only

Then, suppose $u = 0^p$, $v = 0^q$, $w = 0^r1^s$

We must have $p+q+r = s$ (as we have 0^n1^n) and $q>0$

Now, $uv^kw = 0^p(0^q)^k0^r1^s = 0^{p+qk+r}1^s$

The strings in $0^{p+qk+r}1^s \in L$ only for $k=1$ otherwise not

Hence, for this case, the language is not regular

Pumping Lemma

Case 2: Let v contain 1's only

Then, suppose $u = 0^p1^q$, $v = 1^r$, $w = 1^s$

We must have $p = q+r+s$ and $r>0$

Now, $uv^kw = 0^p1^q(1^r)^k1^s = 0^p1^{q+rk+s}$

The strings in $0^p1^{q+rk+s} \in L$ only for $k=1$ otherwise not

Hence, for this case too, the language is not regular

Pumping Lemma

Case 3: Let v contain both 0's and 1's only

Then, suppose $u = 0^p$, $v = 0^q1^r$, $w = 1^s$

We must have $p+q = r+s$ and $q+r>0$

Now, $uv^kw = 0^p(0^q1^r)^k1^s = 0^{p+qk}1^{rk+s}$

The strings in $0^{p+qk}1^{rk+s} \in L$ only for $k=1$ otherwise not

Hence, again, the language is not regular

Thus, through contradiction, we have proved that L is not regular