Nadar Saraswathi College of Engineering & Technology

NSCET E-LEARNING PRESENTATION

LISTEN … LEARN… LEAD…

COMPUTER SCIENCE AND ENGINEERING

III YEAR / V SEMESTER

CS8501 – Theory of Computation

P.MAHALAKSHMI,M.E,MISTE

ASSISTANT PROFESSOR

Nadar Saraswathi College of Engineering & Technology,

Vadapudupatti, Annanji (po), Theni – 625531.

UNIT II

Regular Expressions and Languages

Introduction

Regular Expression

✓The language accepted by finite automata can be easily described by simple expressions called Regular Expressions. It is the most effective way to represent any language.

✓The languages accepted by some regular expression are referred to as Regular languages.

✓A regular expression can also be described as a sequence of pattern that defines a string.

✓Regular expressions are used to match character combinations in strings. String searching algorithm used this pattern to find the operations on a string.

A **Regular Expression** can be recursively defined as follows −

✓**ε** is a Regular Expression indicates the language containing an empty string.

L (ε) = {ε}

✓**φ** is a Regular Expression denoting an empty set.

L (φ) = { }

- ✓ x is a Regular Expression where L = {x}

If **X** is a Regular Expression denoting the language **L(X)** and **Y** is a Regular Expression denoting the language **L(Y)**, then

- ✓ **X + Y** is a Regular Expression corresponding to the language

  L(X) ∪ L(Y) where **L(X+Y) = L(X) ∪ L(Y)**.

- ✓ **X . Y** is a Regular Expression corresponding to the language

  L(X) . L(Y) where **L(X.Y) = L(X) . L(Y)**

- ✓ **R*** is a Regular Expression corresponding to the language

  L(R*)where **L(R*) = (L(R))***

- ➢ x* - Kleen Closure

  - Means zero or more occurrence of x. It can generate {ε, x, xx, xxx, xxxx, ...}

- ➢ x^+ - Positive Closure

  - Means one or more occurrence of x. It can generate {x, xx, xxx, xxxx, ...}

Example 1:

Write the regular expression for the language accepting all combinations of a's over the set ∑ = {a}

Solution

L={ ε, a, aa, aaa, aaa, ……..} **RE = a***

Example 2:

Write the regular expression for the language accepting all combinations of a's except the null string over the set ∑ = {a}

Solution

L={ a, aa, aaa, aaa, ……..} **RE = a^+**

Example 2:

Write the regular expression for the language accepting all the string containing any number of a's and b's

Solution

L={ε, a, b, abb,baa,aba,bab,…..}

RE = (a+b)*

Example 1:

Write the regular expression for the language accepting all the string which are starting with 1 and ending with 0, over ∑ = {0, 1}.

Solution

L={10,100,110,11010, …..}

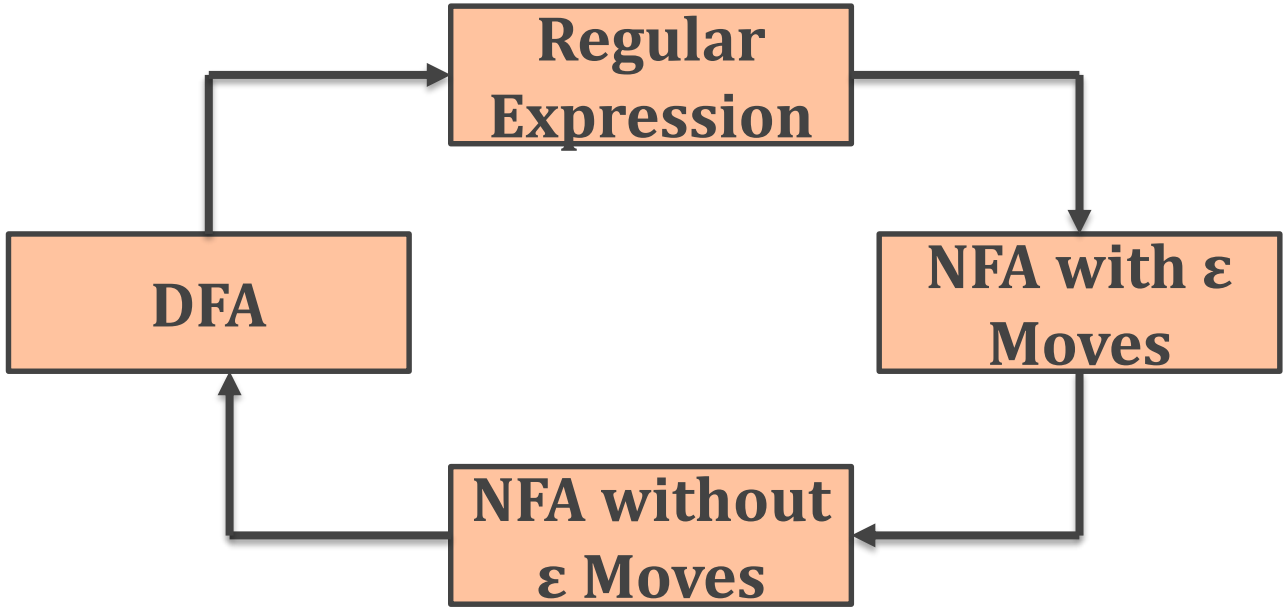
RE= 1 (0+1)* 0

Topic

Finite Automata and Regular Expressions

Convert Regular Expression into Finite Automata

- ✓ Close relationship between a finite automata and a regular expression is shown in figure.



Conversion of RE into FA (NFA)

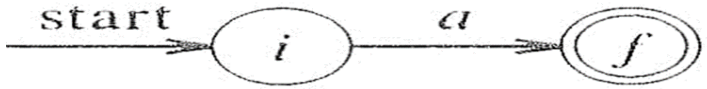
Thompson's Construction Method

- ✓ Thompson's Construction used to find out a Finite Automaton(NFA) from a Regular Expression.

Case 1: r = ε

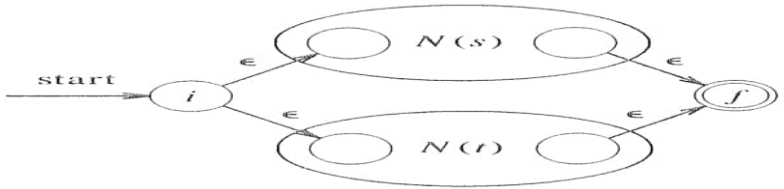
Case 2: r = φ in ∑

Case 3: r = a in ∑



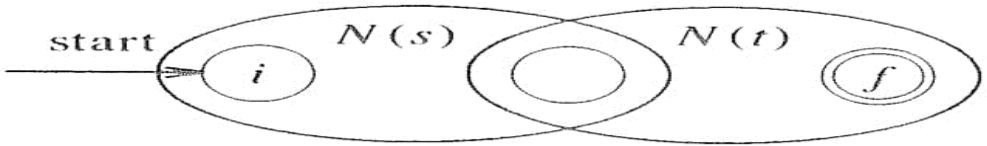
Case 4:Union

✓Suppose N(s) and N (t) are NFA's for regular expressions s and t, respectively. For the regular expression (s|t) (or) (s+t)



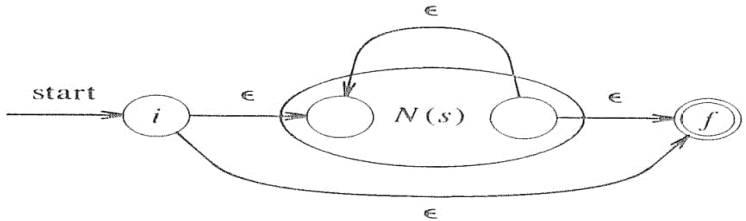
Case 4: Concatenation

✓Suppose N(s) and N (t) are NFA's for regular expressions s and t, respectively. For the regular expression st (or) s.t

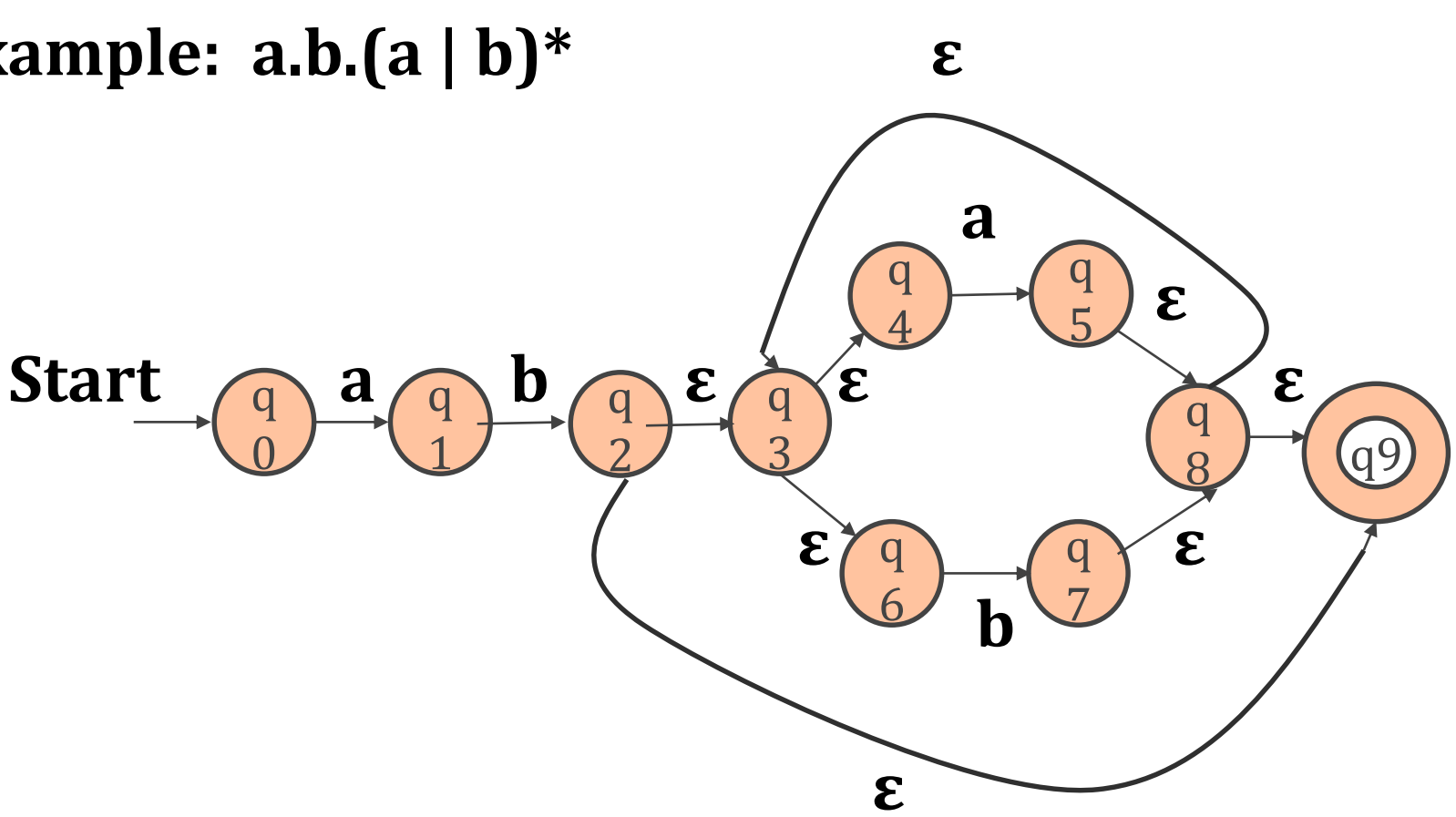


Case 4: Kleen Closure

For the regular expression S*



Example: a.b.(a | b)*



L={ab ε , aba, abb, abaaabbb, abababababa.....}

iii) Proof by Counter Example

- ✓ In order to prove certain statements, we need to see all possible conditions in which that statement remains true.
- ✓ There are some situation in which the statement can't be true.

Example 1

All primes are odd. 5/2=2 2%2=1 2%2=1

Disproof :

The integer 2 is prime but 2 is even

Example 2

There is no such pair of integers a & b such that a mod b = b mod a

Disproof: Consider a=2 and b=3 then clearly

2 mod 3 ≠ 3 mod 2

Correct version: To change the statement slightly

a mod b = b mod a when a=b

- ✓ This type of proof is called counter example. Such proof is true only at some specific condition

Convert Finite Automata into Regular Expression

- ✓ Every regular language defined by finite automata is also defined by the regular expression.

Two Types

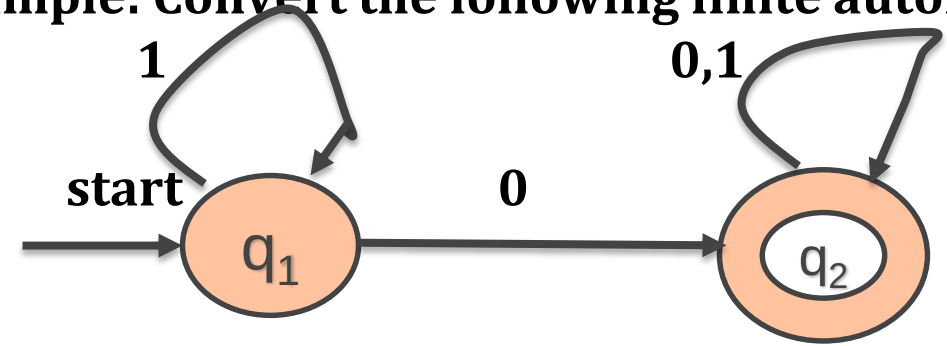
i) Transitive closure method or formula method

- ✓ The regular expression $R_{ij}^{(k)}$ whose language is the set of strings w such that, w is the label of the path from the initial state i to the final state j in finite automata and k is the total number of states.
- ✓ The path from the initial state i to the final state j that has no intermediate states.

$$R_{ij}^{k} = R_{ik}^{k-1}(R_{kk}^{k-1})^{*} R_{kj}^{k-1} + R_{ij}^{k-1}$$

- ✓ It is very clear and implementation is simple. This method is tedious from manual use and it tends to generate much extended regular expressions.

Example: Convert the following finite automata to regular expression



Solution

i -> initial state = 1

j -> final state = 2

k -> total number of states = 2

Apply the values in formula

$$R_{12}^{2} = R_{12}^{(2-1)} . (R_{22}^{(2-1)})^* . R_{22}^{(2-1)} + R_{12}^{(2-1)}$$

$$R_{12}^{2} = R_{12}^{(1)} . (R_{22}^{(1)})^* . R_{22}^{(1)} + R_{12}^{(1)}$$

$R_{12}^{(1)}$= ? $R_{22}^{(1)}$= ?

Find Initial values

$R_{11}^{(0)} = 1$

$R_{12}^{(0)} = 0$

$R_{21}^{(0)} = \phi$

$R_{22}^{(0)} = 0+1$

Find $R_{12}^{(1)}$

i=1 j=2 k=1

$$\mathbf{R_{12}^{1} = R_{11}^{(0)} . (R_{11}^{(0)})^* . R_{12}^{(0)} + R_{12}^{(0)}}$$

$= 1.1^*.0 + 0$

$= 0.[11^* + \varepsilon]$ $[RR^* + \varepsilon = R^*]$

$\mathbf{R_{12}^{1}} = 0.1^*$

Find R22(1)

i=2 j=2 k=1

$R_{22}^{(1)} = R_{21}^{(0)} . (R_{11}^{(0)})^* . R_{12}^{(0)} + R_{22}^{(0)}$

$= \phi . 1^*. 0 + (0+1)$ $[\phi. R= \phi]$

$= \phi + (0+1)$ $[\phi+R=R]$

$R_{22}^{(1)} = (0+1)$ $R_{12}^{1} = 0.1^*$

$R_{12}^{2} = R_{12}^{(1)} . (R_{22}^{(1)})^* . R_{22}^{(1)} + R_{12}^{(1)}$

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

$= 01^* [(0+1)^*. (0+1) + \varepsilon]$ $[RR^*+ \varepsilon=R^*]$

$R_{12}^{2} = 01^*.(0+1)^*$

ANS:

RE=01*.(0+1)*

ii) State removal method or state elimination techniques

To find the regular expression for n state DFA was more complex and large using the formula method. Identifies nodes within the transition diagram and removes states, building up regular expressions along each transition.

Rules

1. Unify all final states into a single final state using ε - transitions
2. Unify all multi-transitions into a single transitions that contains union of inputs.
3. Remove states until there is only the starting and final state.
4. Get the resulting regular expression by direct calculation

Operations on Language

L1 & L2 – Two language then

i) Union of two language → L1 ∪ L2

ii) Concatenation of two language → L1.L2

iii) Intersection of two language → L1 ∩ L2

iv)Difference of two language → L1 – L2

Ex: Consider the string X=110 & Y=0110 then

XY=1100110 YX=0110110

Set

- ✓ Collection of well defined distinct elements without repetition.
- ✓ Elements are enclosed within curly brackets '{' & '}'
- ✓ Every element is separated by comma

Ex: A={a,b,c,d,e}

A is a set with five elements.

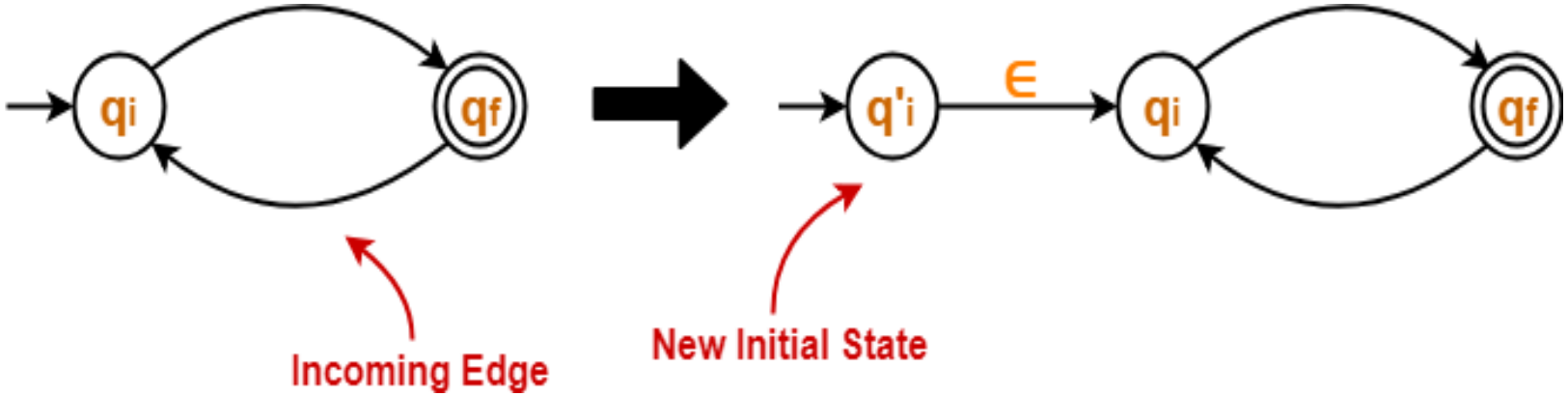
Conversion Steps

- ✓ This method involves the following steps in finding the regular expression for any given DFA.

Step1:

- ✓ The initial state of the DFA must not have any incoming edge.
- ✓ If there exists any incoming edge to the initial state, then create a new initial state having no incoming edge to it.

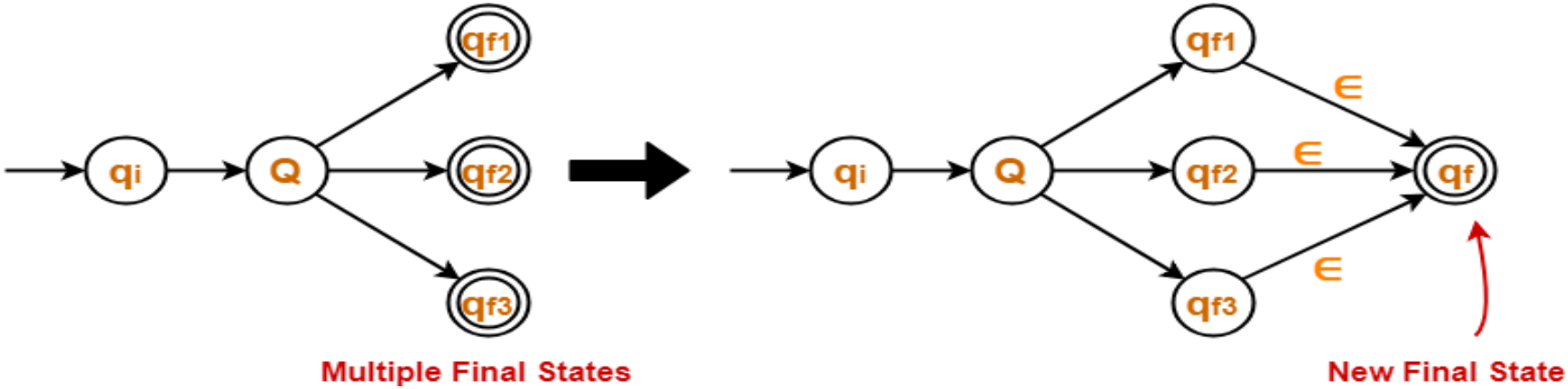
Example



Step 2:

- ✓ There must exist only one final state in the DFA.
- ✓ If there exists multiple final states in the DFA, then convert all the final states into non-final states and create a new single final state.

Example

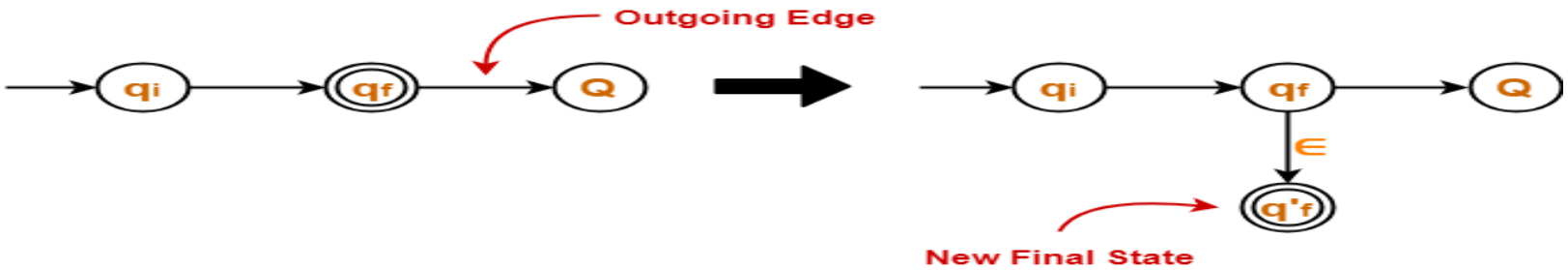


Step 3:

✓The final state of the DFA must not have any outgoing edge.

✓If there exists any outgoing edge from the final state, then create a new final state having no outgoing edge from it.

Example:



Step 4:

✓Eliminate all the intermediate states one by one.

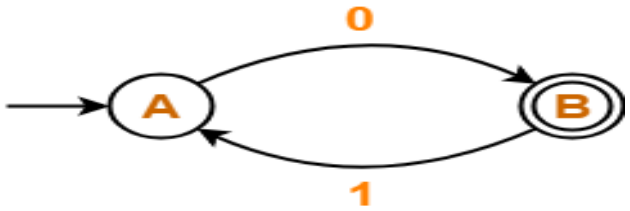
✓These states may be eliminated in any order.

In the end,

✓Only an initial state going to the final state will be left.

✓The cost of this transition is the required regular expression.

Example 1: Find regular expression for the following DFA

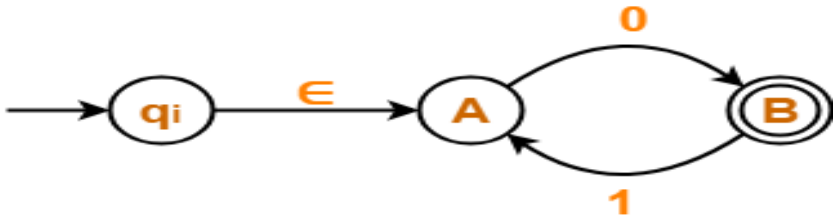


Solution

Step 1:

Initial state A has an incoming edge.

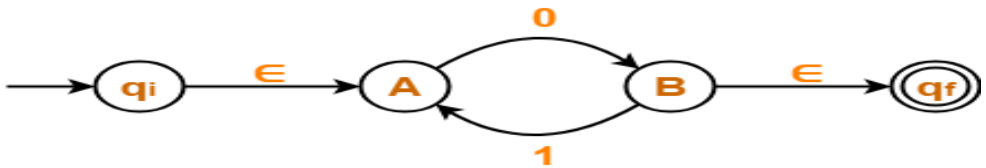
So, we create a new initial state q_i. The resulting DFA is



Step 2:

Final state B has an outgoing edge.

So, we create a new final state q_f. The resulting DFA is-



Step 3:

Now, we start eliminating the intermediate states.

✓First, let us eliminate state A.

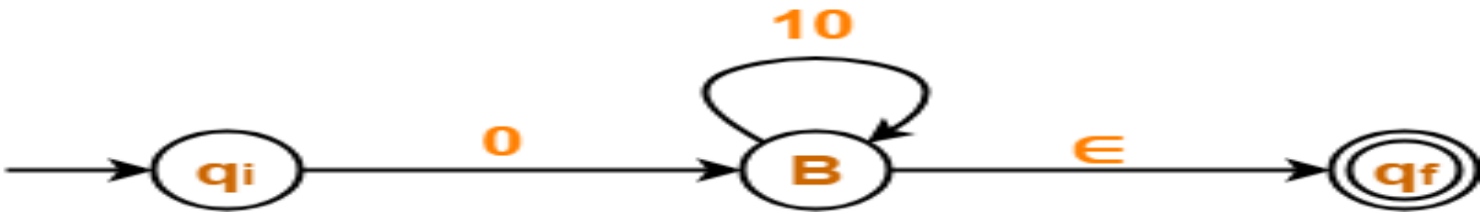
✓There is a path going from state q_i to state B via state A.

✓So, after eliminating state A, we put a direct path from state q_i to state B having cost $\in.0 = 0$

✓There is a loop on state B using state A.

✓So, after eliminating state A, we put a direct loop on state B having cost 1.0 = 10.

Eliminating state A, we get-



Step-04:

Now, let us eliminate state B.

✓There is a path going from state q_i to state q_f via state B.

✓So, after eliminating state B, we put a direct path from state q_i to state q_f having cost $0.(10)^*.\in = 0(10)^*$

Eliminating state B, we get

Ans:

Regular Expression = 0(10)*

Topic

Proving languages not to be Regular

Introduction

The properties of the regular languages are,

1.Pumping Lemma 2. Closure Properties

Pumping Lemma

It is a way to prove that certain languages are not regular.

Closure Properties

It allows to build the recognizers for languages that are constructed from other languages by certain properties.

Pumping Lemma

Generally the language denoted by the regular expression is regular language or regular set.

The regular languages are accepted by following,

i) DFA ii) NFA iii) NFA – ε iv) RE

Applications of pumping lemma

Pumping lemma is used to prove that the language is not regular.

Steps

i) Assume that given language is a regular language.

ii) Calculate the number of states in DFA

iii) Take one string 'w' and calculate the length of the string

iv) If the length of the string |w| ≥n, then break the string into 3 strings.

v) Consider w=xyz

a) $|xy| \leq n$

b) $y \neq \varepsilon$ (or) $|y|>0$ or $|y|\geq 1$

c) For all $k\geq 0$, the string xy^kz is also in 'L'.

Example

Show that the language $L=\{ a^nb^n \mid n\geq 1 \}$ is not a regular.

Solution

a) Assume that given language 'L' is a regular language

b) The numbers of states of $L=\{ a^nb^n \mid n\geq 1 \}$ is

$n+n=2n$

c) Take one string 'w' and calculate the length of the string

$w= a^nb^n$

$w = n+n=2n$

d) If the length of the string $|w| \geq n$

$|w| \geq n$

$2n \geq n$

So break w into 3 strings

e) w=xyz

Let w= $a^i b^i$

a^i

$xy = a^m$ **$y=a^j$** **$z=a^{i-m}$**

To check the assumption

$xyz= a^m a^{i-m} b^i$

$= a^{m+i-m} b^i$

$= a^i b^i$

So the assumption is correct

i) $|xy| \leq n$

$|a^m| \leq n \rightarrow m \leq n$

ii) $y \neq \varepsilon$ (or) $|y|>0$ or $|y| \geq 1$

$a^j \neq \varepsilon$

iii) For all k≥0, the string xy^kz is also in 'L'.

$$xy^kz = x\ y\ y^{k-1}\ z$$
$$= \mathbf{a^m\ (a^j)^{k-1}\ a^{i-m}\ b^i}$$
$$= a^{m + j(k-1)+ i-m}\ b^i$$
$$\mathbf{xy^kz = a^{i+j(k-1)}\ b^i}$$

Put k = 0

$xy^0z = a^{i + j(0-1)}\ b^i$

$= a^{i-j}\ b^i \neq a^i\ b^i \neq L$

k=1

$xy^1z = a^{i + j(1-1)}\ b^i$

$= a^{i + 0}\ b^i = a^i\ b^i = L$

k=2

$xy^1z = a^{i + j(2-1)}\ b^i$

$= a^{i + j}\ b^i \neq a^i\ b^i \neq L$

For k=0,1,2 the string that does not belong to the language L

So the language L={ a^nb^n | n≥1 } is not a regular

Topic

Closure Properties of Regular Languages

Introduction

A closure property is a statement that a certain operation on languages, when applied to languages in a class (e.g., the regular languages), produces a result that is also in that class. For regular languages, we can use any of its representations to prove a closure property.

Closure Under Union

If L and M are regular languages, so is L ∪ M.

Proof:

Let L and M be the languages of regular expressions R and S, respectively. Then R+S is a regular expression whose language is L ∪ M.

Closure Under Concatenation and Kleene Closure

Same idea: RS is a regular expression whose language is LM. R* is a regular expression whose language is L*.

Closure Under Intersection

If L and M are regular languages, then so is L ∩ M.

Proof:

Let A and B be DFA's whose languages are L and M, respectively. Construct C, the product automaton of A and B. Make the final states of C be the pairs consisting of final states of both A and B.

Closure Under Difference

If L and M are regular languages, then so is L – M = strings in L but not M.

Proof:

Let A and B be DFA's whose languages are L and M, respectively. Construct C, the product automaton of A and B. Make the final states of C be the pairs where A-state is final but B-state is not.

Closure Under Complementation

The complement of a language L (with respect to an alphabet Σ such that Σ* contains L) is Σ* – L. Since Σ* is surely regular, the complement of a regular language is always regular.

Closure Under Reversal

Recall example of a DFA that accepted the binary strings that, as integers were divisible by 23. We said that the language of binary strings whose reversal was divisible by 23 was also regular, but the DFA construction was very tricky. Good application of reversal-closure.

Given language L, L R is the set of strings whose reversal is in L.

Example: L = {0, 01, 100}; L R = {0, 10, 001}.

Proof: Let E be a regular expression for L. We show how to reverse E, to provide a regular expression E R for L R.

Reversal of a Regular Expression

Basis: If E is a symbol a, ε, or ∅, then E R = E.

Induction: If E is

F+G, then $E^R = F^R + G^R$.

FG, then $E^R = G^R F^R$

F*, then $E^R = (F^R)^*$.

Homomorphism

A homomorphism on an alphabet is a function that gives a string for each symbol in that alphabet.

Example: h(0) = ab; h(1) = ε.

Extend to strings by h(a 1…a n) = h(a 1)…h(a n).

Example: h(01010) = ababab.

Closure Under Homomorphism

If L is a regular language, and h is a homomorphism on its alphabet, then h(L) = {h(w) | w is in L} is also a regular language.

Proof: Let E be a regular expression for L. Apply h to each symbol in E. Language of resulting RE is h(L).

Example: Closure under Homomorphism

Let h(0) = ab; h(1) = ε. Let L be the language of regular expression 01* + 10*. Then h(L) is the language of regular expression abε* + ε(ab)*. Note: use parentheses to enforce the proper grouping.

Inverse Homomorphism

Let h be a homomorphism and L a language whose alphabet is the output language of h. $h^{-1}(L) = \{w \mid h(w) \text{ is in } L\}$

Example: Inverse Homomorphism

Let h(0) = ab; h(1) = ε. Let L = {abab, baba}. $h^{-1}(L)$ = the language with two 0's and any number of 1's = L(1*01*01*).

Notice: no string maps to baba; any string with exactly two 0's maps to abab.

Closure Proof for Inverse Homomorphism

Start with a DFA A for L. Construct a DFA B for $h^{-1}(L)$ with:

i) The same set of states. ii) The same start state. iii) The same final states.

Input alphabet = the symbols to which homomorphism h applies.

Proof – (2) : The transitions for B are computed by applying h to an input symbol a and seeing where A would go on sequence of input symbols h(a). Formally, $\delta B(q, a) = \delta A(q, h(a))$.

Topic

Equivalence and Minimization of Automata

Conversion of NFA with ε to NFA without ε

Steps

In this method to remove all the ε transitions from given NFA

Step1: Find out all the **ε** transitions for each state from Q. That will be called as ε - closure$\{q_i\}$ where $q_i \in Q$.

Step2: Then δ' transitions can be obtained. The δ' transitions means an ε - closure on δ moves.

Step3: Step2 is repeated for each input symbol and for each state of given NFA.

Step4: Using the resultant states the transition table for equivalent NFA without ε can be built.

Conversion of NFA without ε to DFA

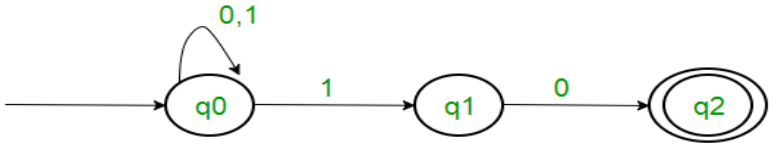
Conversion Steps of NFA to DFA

Step 1:NFA denoted as $M=(Q_N , \sum , \delta_N, q_0, F)$, DFA denoted as $M'=(Q_D, \sum, \delta_D, q_0, F)$ where δ_N is the transition function for NFA and δ_D is the transition function for DFA.

Step 2: Take the initial state in NFA, process the input symbols

Step 3: Get the new state and then process it using the input symbols.

Step 4: Process the NFA using union functions and write it for DA transition.
Step 5: The states in NFA are represented using braces {….} and in DFA using brackets […]
Step 6: Process all the states until there is no repetition of a new state.
Example: Convert the given NDFA into DFA



Solution
The starting state in NFA is {q0}
For DFA,
{q0}=[q0] in DFA -> A (New State)
Step 1: Process the input symbol {0,1}
δ(A,0) = δ(q0,0)
= [q0]
δ(A,0)= [q0] = A
δ(B,0)=δ({q0,q1},0)
= δ(q0,0) ∪ δ(q1,0)
= {q0 ∪ q2}=[**q0,q2]=C**

δ(A,1) = δ(q0,1)
=[q0,q1]
δ(A,1)= [q0,q1] = B (New State)
δ(B,1)=δ({q0,q1},1)
= δ(q0,1) ∪ δ(q1,1)
= {q0,q1} ∪ ϕ
= [q0,q1] =B

Step 2: Transition Table

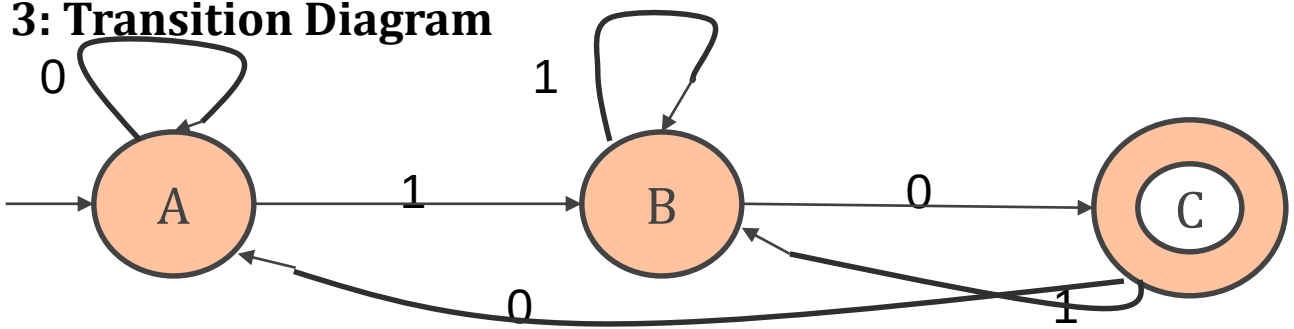
δ(A,0)= [q0] = A δ(A,1)= [q0,q1] = B

δ(B,0)= [q0,q2]= C (New state) δ(C,0)= A

δ(B,1) = B δ(C,1) = B

Input States		0	1
[q0]	->A	A	B
[q0,q1]	B	C	B
[q0,q2]	*C	A	B

Step 3: Transition Diagram



Conversion Steps of NFA with ε to DFA

Step 1: Find out all the **ε** transitions for each state from Q. That will be called as ε - closure{q_i} where $q_i \in Q$.

Step 2: Take the ε-closure for the starting state of NFA as a starting state of DFA.

Step 2: Find the states for each input symbol that can be traversed from the present. That means the union of transition value and their closures for each state of NFA present in the current state of DFA.

Step 3: If we found a new state, take it as current state and repeat step 2.

Step 4: Repeat Step 2 and Step 3 until there is no new state present in the transition table of DFA.

Step 5: Mark the states of DFA as a final state which contains the final state of NFA.

Minimization of DFA

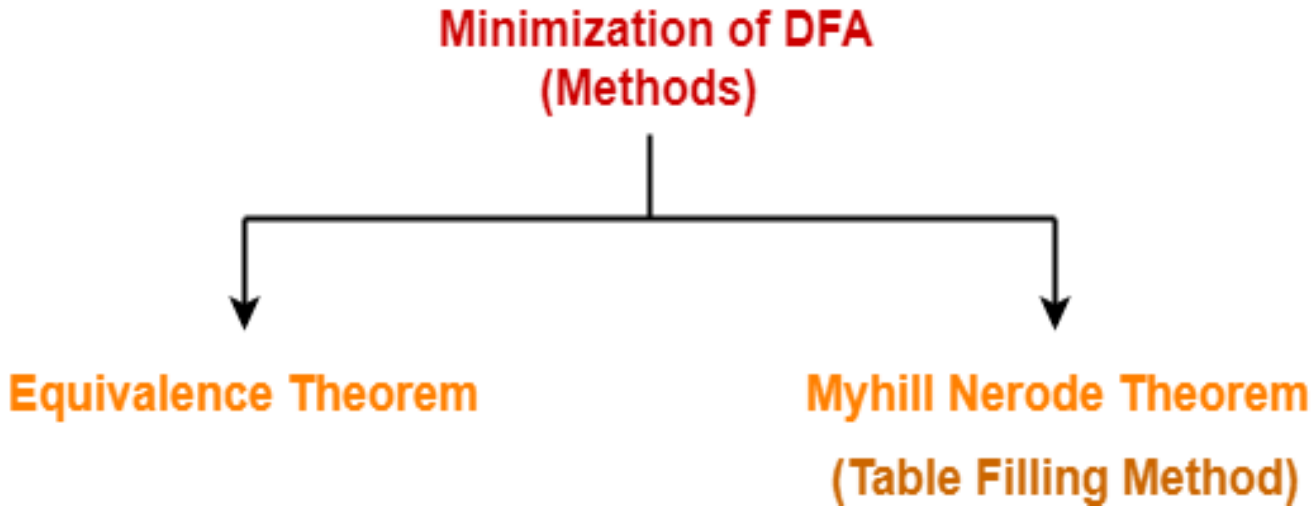
Introduction

✓The process of converting a given DFA into an equivalent DFA with a minimum number of states is considered to be a minimization of DFA.

✓It contains the minimum number of states.

✓The DFA in its minimal form is called as a **Minimal DFA**.

✓The two popular methods for minimizing a DFA are-



Testing Equivalence of states

- ✓ To test equivalence of two states
- ✓ Consider two distinct states p and q that can be replaced by another single state that behaves like both p and q.
- ✓ Say that p and q are equivalent states if and only if they reach the same state for a same input symbol.

Equivalent States

The states p and q are equivalent states

$\delta(p,w) = R$

$\delta(q,w) = R$

- ✓ The states p and q are equivalent if "for all input strings w, $\delta(p,w)$ is an accepting state if and only if $\delta(q,w)$ is also an accepting state.
- ✓ To consider only the transition of the state p and q after processing the input string reaches accepting states or both the states p and q reaches non accepting state.

Distinguishable States

✓If two states are not equivalent, then we say that both states are distinguishable states.

✓That is, the state p is distinguishable from state q if there is at least one string 'w' such that one of δ(p,w) is accepting and it reaches final state and δ(q,w) reaches non accepting state.

Minimization of DFA Using Table Filling Algorithm(Myhill Nerode Theorem)

Input: DFA

Output : Minimized DFA

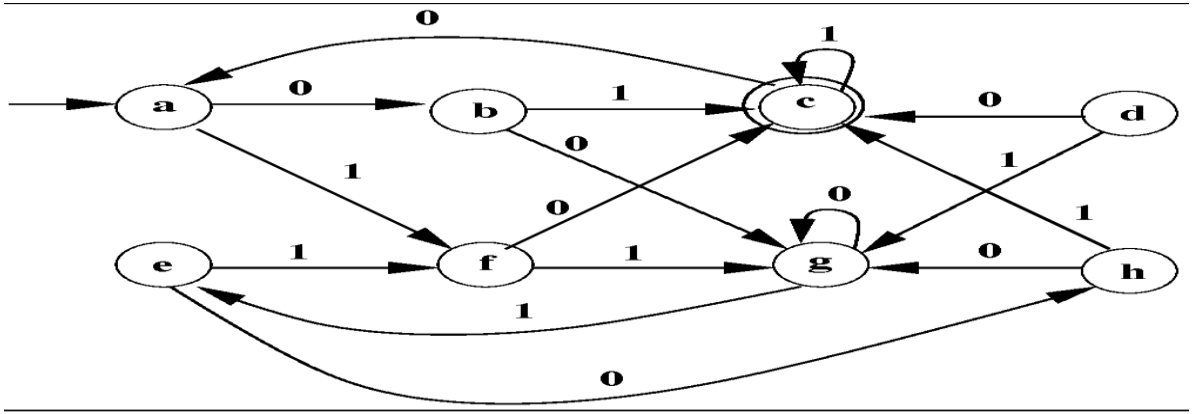
Step 1: Draw a table for all pairs of states (Q_i, Q_j) not necessarily connected directly [All are unmarked initially]

Step 2 : Consider every state pair (Q_i, Q_j) in the DFA where $Q_i \in F$ and $Q_j \notin F$ or vice versa and mark them. [Here F is the set of final states]

Step 3: Repeat this step until we cannot mark anymore states – If there is an unmarked pair (Q_i, Q_j), mark it if the pair {δ (Q_i, A), δ (Q_i, A)} is marked for some input alphabet.

Step 4 : Combine all the unmarked pair (Q_i, Q_j) and make them a single state in the reduced DFA.

Example: Find the Equivalence and Minimization of finite automata for the following FA



Solution

Initial State = a

Final State = c

We can't compare same state itself such as (a,a) (b,b) (c,c) (d,d) (e,e) (f,f) (g,g) (h,h)

Distinguishable States

✓If two states are not equivalent, then we say that both states are distinguishable states.

✓That is, the state p is distinguishable from state q if there is at least one string 'w' such that one of δ(p,w) is accepting and it reaches final state and δ(q,w) reaches non accepting state.

Minimization of DFA Using Table Filling Algorithm(Myhill Nerode Theorem)

Input: DFA

Output : Minimized DFA

Step 1: Draw a table for all pairs of states (Q_i, Q_j) not necessarily connected directly [All are unmarked initially]

Step 2 : Consider every state pair (Q_i, Q_j) in the DFA where $Q_i \in F$ and $Q_j \notin F$ or vice versa and mark them. [Here F is the set of final states]

Step 3: Repeat this step until we cannot mark anymore states – If there is an unmarked pair (Q_i, Q_j), mark it if the pair {δ (Q_i, A), δ (Q_i, A)} is marked for some input alphabet.

Step 4 : Combine all the unmarked pair (Q_i, Q_j) and make them a single state in the reduced DFA.

Step 1: Draw Table for Given States

	a	b	c	d	e	f	g
b	x						
c	x	x					
d	x	x	x				
e		x	x	x			
f	x	x	x		x		
g	x	x	x	x	x	x	
h	x		x	x	x	x	x

Step 2: Compare the accepting state to non accepting state and put X on each pair.
(a,c) (b,c) (d,c) (e,c) (f,c) (g,c) -> Put X

Step 3: Comparing the equivalence of all the remaining pair

(a,b)

δ(a,0)=b δ(a,1)=f

δ(b,0)=g δ(b,1)=c

(a,d)

(a,h)

δ(a,0)=b δ(a,1)=f

δ(h,0)=g δ(h,1)=c

(b,d)

δ(a,0)= b δ(a,1)= f

δ(d,0)= c δ(d,1)= g

(a,e)

δ(a,0)= b δ(a,1)= f

δ(e,0)= h δ(e,1)= f

(a,f)

δ(a,0)= b δ(a,1)= f

δ(f,0)= c δ(f,1)= g

(a,g)

δ(a,0)= b δ(a,1)= f

δ(g,0)= g δ(g,1)= e

δ(b,0)=g δ(b,1)= c

δ(d,0)= c δ(d,1)=g

(b,e)

δ(b,0)=g δ(b,1)= c

δ(e,0)=h δ(e,1)= f

(b,f)

δ(b,0)= g δ(b,1)= c

δ(f,0)= c δ(f,1)= g

(b,g)

δ(b,0)= g δ(b,1)=c

δ(g,0)= g δ(g,1)=e

(b,h)

δ(b,0)= g δ(b,1)=c

δ(h,0)= g δ(h,1)=c

(d,e)

δ(d,0)= c δ(d,1)= g

δ(e,0)= h δ(e,1)= f

(d,f)

δ(d,0)= c δ(d,1)= g

δ(f,0)= c δ(f,1)= g

(d,g)

δ(d,0)= c δ(d,1)= g

δ(g,0)= g δ(g,1)= e

(d,h)

δ(d,0)= c δ(d,1)= g

δ(h,0)= g δ(h,1)= c

(e,f)

δ(e,0)= h δ(e,1)= f

δ(f,0)= c δ(f,1)= g

(e,g)

δ(e,0)= h δ(e,1)= f

δ(g,0)= g δ(g,1)= e

(e,h)

δ(e,0)=h δ(e,1)= f

δ(h,0)= g δ(h,1)= c

(f,g)

δ(f,0)= c δ(f,1)=g

δ(g,0)= g δ(g,1)= e

(f,h)

δ(f,0)= c δ(f,1)= g

δ(h,0)= g δ(h,1)= c

(g,h)

δ(g,0)= g δ(g,1)= e

δ(h,0)= g δ(h,1)= c

Equivalent States are

a ≡ e b ≡ h d ≡ f , c , g

Step 3: DFA Minimization Table

Input / States	0	1
->[a,e]	b	d
[b,h]	g	c
*c	a	c
[d,f]	c	g
g	g	a