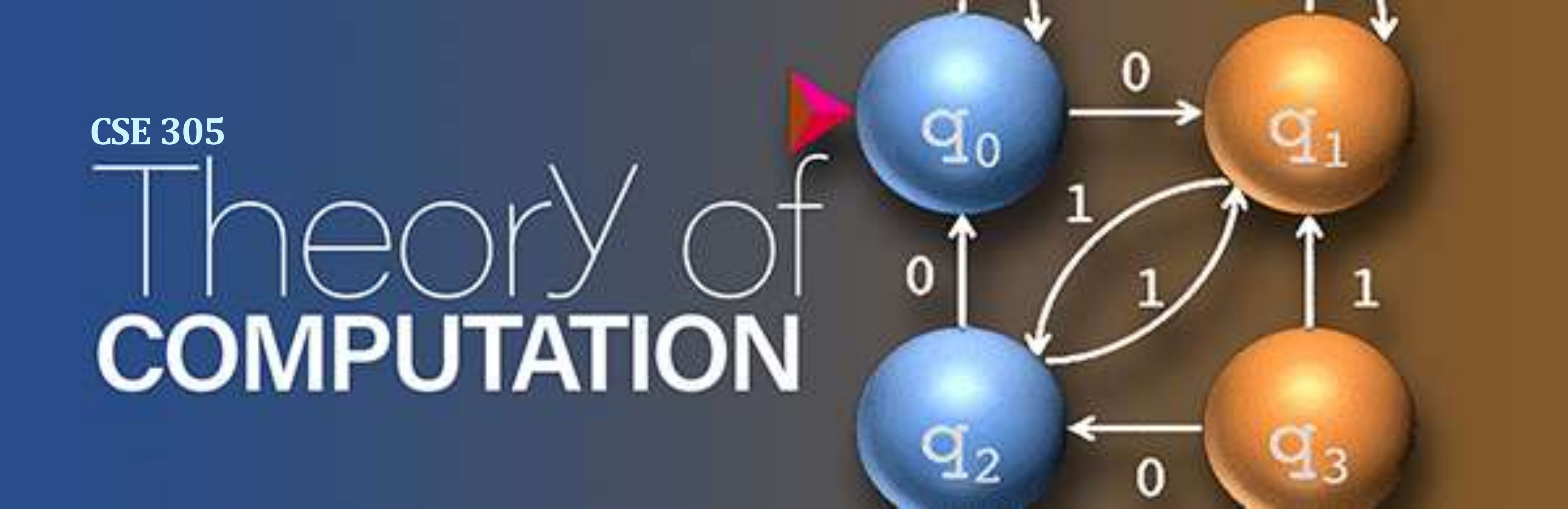


Lecture 21
Regular Expressions (2)

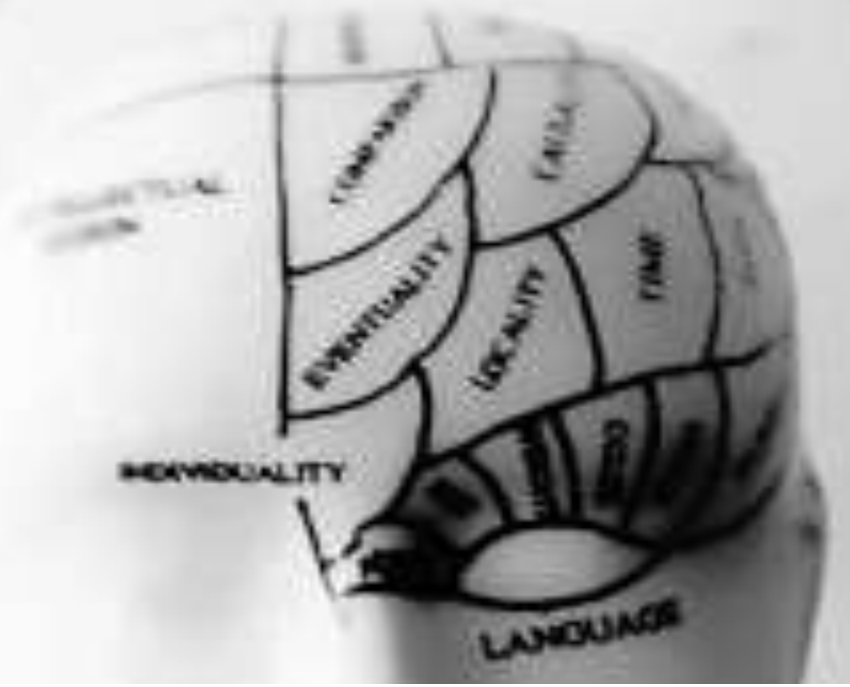
Md. Mijanur Rahman, Prof. Dr.

Dept. of Computer Science and Engineering, Jatiya Kabi Kazi Nazrul Islam University, Bangladesh.

www.mijanrahman.com

Contents

Regular Expressions



Conversion of Regular expression into Finite Automata

- As the regular expressions can be constructed from Finite Automata using the State Elimination Method, the reverse method, **state decomposition method can be used to construct Finite Automata from the given regular expressions.**
- **Note:** This method will construct NFA (with or without ε-transitions, depending on the expression) for the given regular expression, which can be further converted to DFA using NFA to DFA conversion.

- **Thus, the method includes the following steps:**

  Step 1 − Construct a Transition diagram for a given RE by using Non-deterministic finite automata (NFA) with ε moves.

  Step 2 − Convert NFA with ε to NFA without ε.

  Step 3 − Convert the NFA to the equivalent Deterministic Finite Automata (DFA).

Conversion of Regular expression into Finite Automata

State Decomposition Method:

- **Theorem:** Every language defined by a regular expression is also defined by a Finite Automata.
- **Proof:** Let's assume L = L(R) for a regular expression R. We prove that L = L(M) for some ε-NFA M with:
  1) Exactly one accepting state.
  2) No incoming edges at the initial state.
  3) No outgoing edges at the accepting state.
- The proof is done by **structural induction** on R by following the steps below:
- **Step 1:** Create a starting state, say q_1, and a final state, say q_2. Label the transition q_1 to q_2 as the given regular expression, R, as in Fig 1. But, if R is $(Q)^*$, Kleene's closure of another regular expression Q, then create a single initial state, which will also be the final state, as in Fig 2.

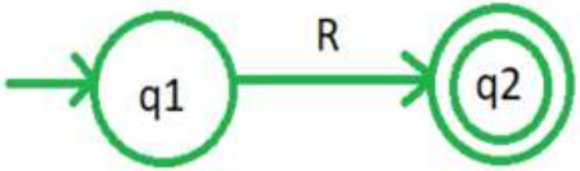


Fig 1

Fig 2

Conversion of Regular expression into Finite Automata

- **Step 2:** Repeat the following rules (state decomposition method) by considering the least precedency regular expression operator first until no operator is left in the expression. Precedence of operators in regular expressions is defined as **Union < Concatenation < Kleene's Closure**.

- **Union operator (+)** can be eliminated by introducing parallel edges between the two states, as shown in Fig.3.

- **The concatenation operator ('.' or *no operator at all*)** can be eliminated by introducing a new state between the states, as shown in Fig.4.

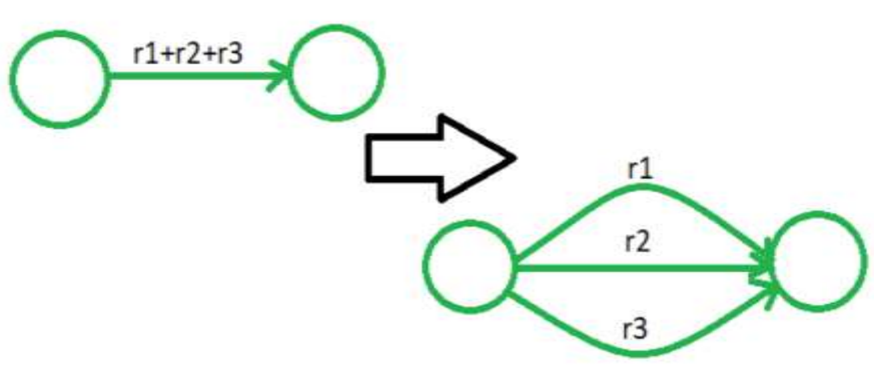


Fig 3: Removal of Union Operator

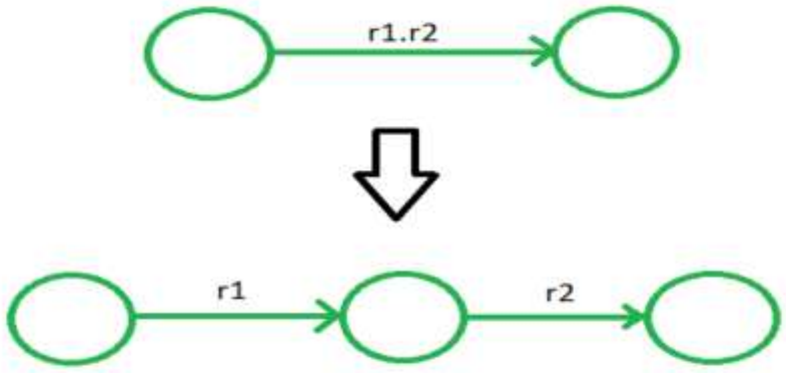


Fig 4: Removal of Concatenation Operator

Conversion of Regular expression into Finite Automata

- **Kleene's Closure (*)** can be eliminated by introducing self-loops on states based on the following conditions:
  1. If there is only one outgoing edge at the left-most state, i.e., A in transition A -> B, then introduce self-loop on state A and label edge A to B as an ε-transition, as shown in Fig 5.
  2. Else if there is only one incoming edge at the right-most state, i.e., B in transition A -> B, then introduce self-loop on state B and label edge A to B as an ε-transition, as shown in Fig 6.
  3. Else introduce a new state between two states having self-loop labeled as the expression. The new state will have ε-transitions with the previous states as follows, as shown in Fig 7.

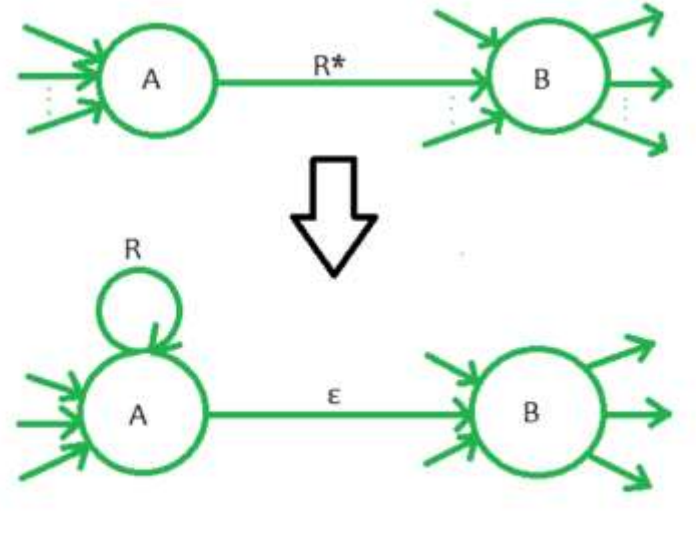


Fig 5

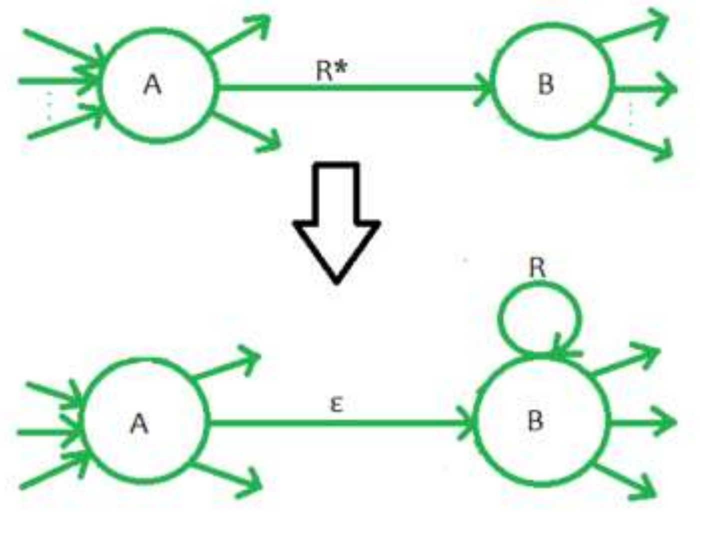


Fig 6

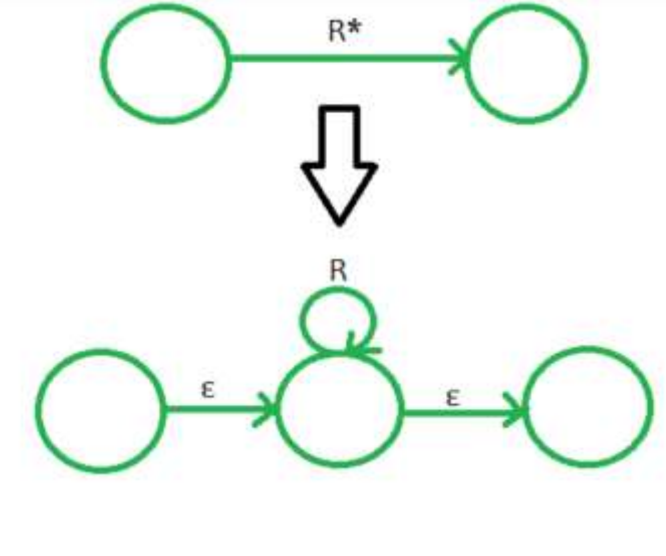


Fig 7

Conversion of Regular expression into Finite Automata

- **Example:** Construct Finite Automata for the regular expression, R = $(ab + ba)^*$

Solution:

- **Step 1:** As the given expression, R, is of the form $(Q)^*$, so we will create a single initial state that will also be the final state, having self-loop labeled (ab + ba), as shown in Fig 8.

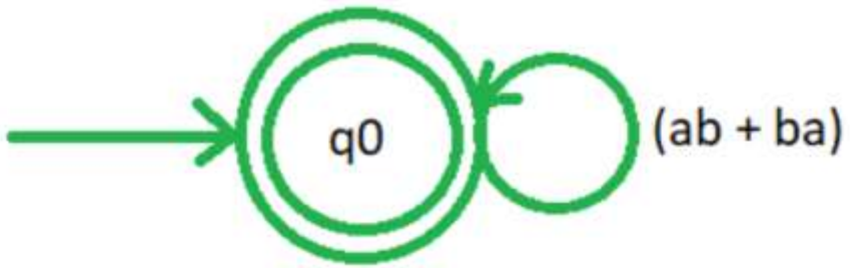


Fig 8

Conversion of Regular expression into Finite Automata

Step 2:

A. As the least precedency operator in the expression is a union(+). So we will introduce parallel edges (parallel self-loops here) for ‘ab’ and ‘ba’, as shown in Fig 9.

B. Now we have two labels with concatenation operators (no operator mentioned between two variables is concatenation), so we remove them one by one by introducing new states, q_1 and q_2 as shown in Fig 10 and Fig 11. (Refer Fig 4 above)

Step 3: As no operators are left, we can say that Fig 11 is the required finite automata (NFA).

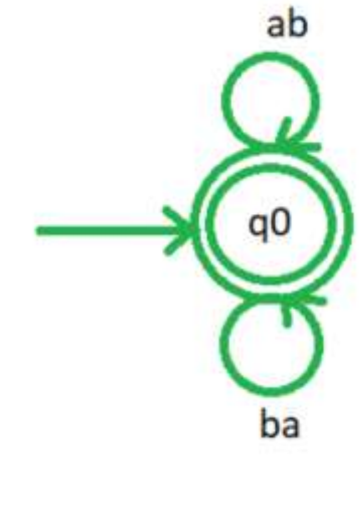


Fig 9

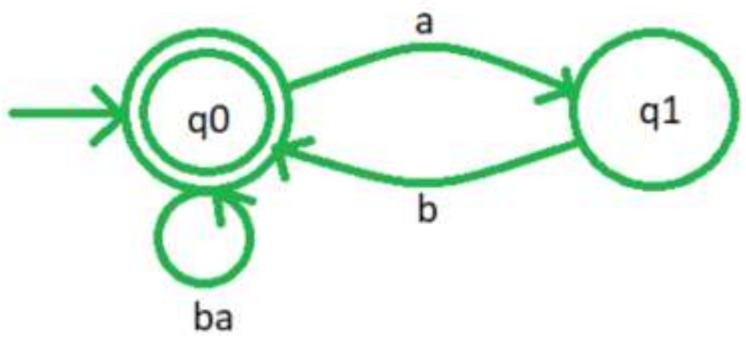
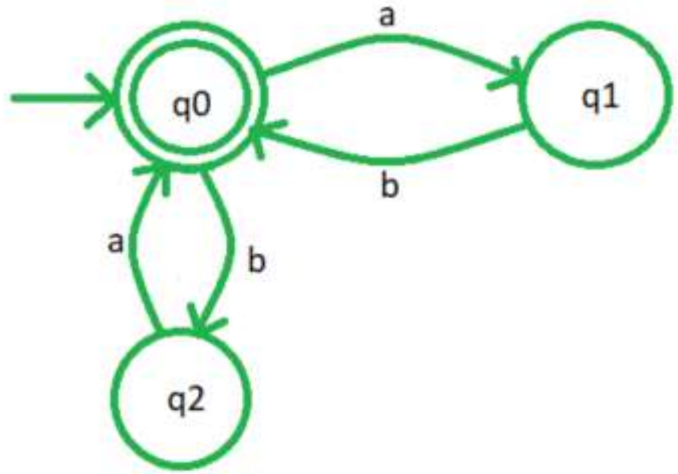


Fig 10

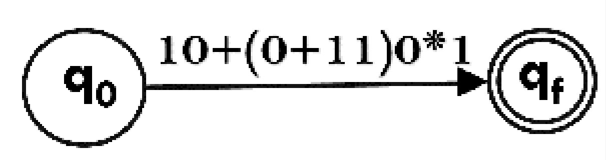


Conversion of Regular expression into Finite Automata

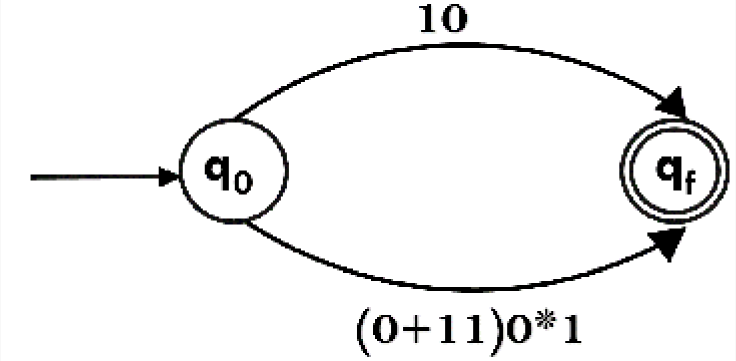
- **Example : Design a FA from given regular expression 10 + (0 + 11)0* 1.**

- **Solution:** First we will construct the transition diagram for a given regular expression.

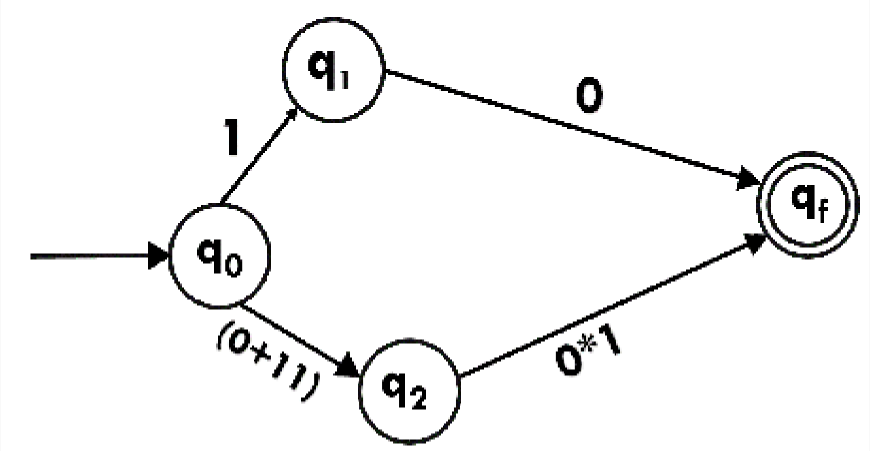
Step-1:



Step-2:



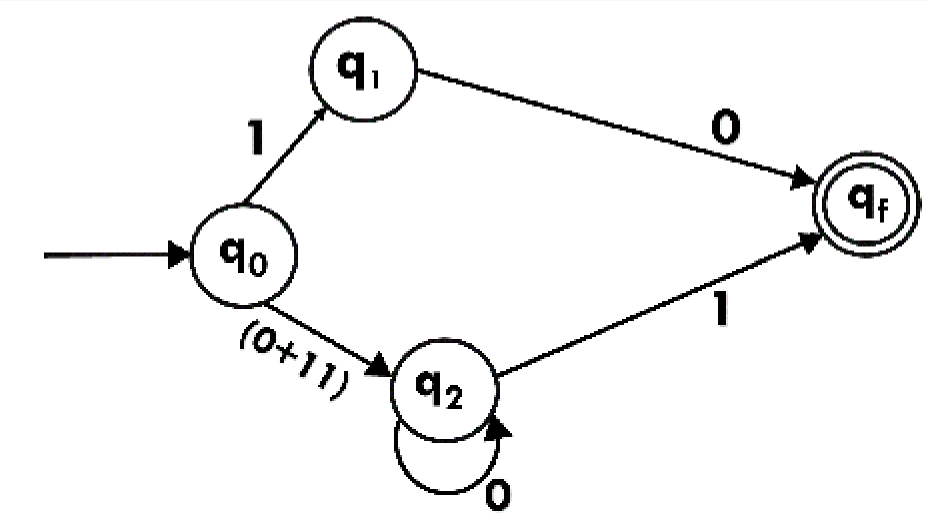
Step-3:



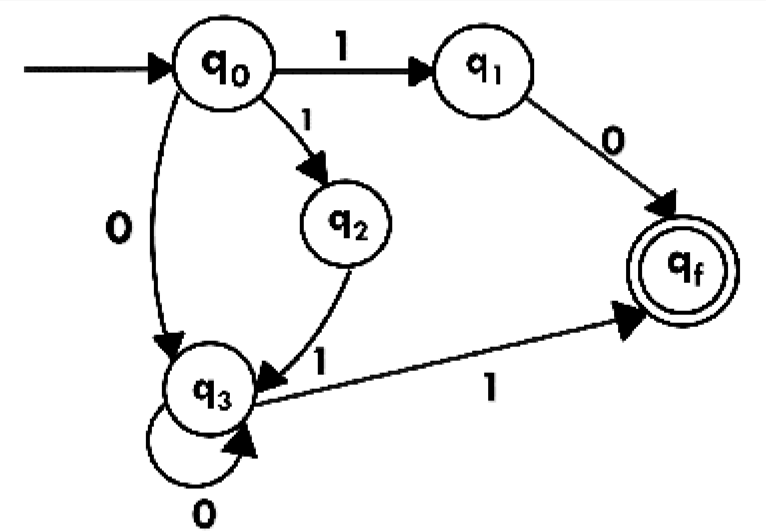
Conversion of Regular expression into Finite Automata

- **Example : Design a FA from given regular expression 10 + (0 + 11)0* 1.**

Step-4:



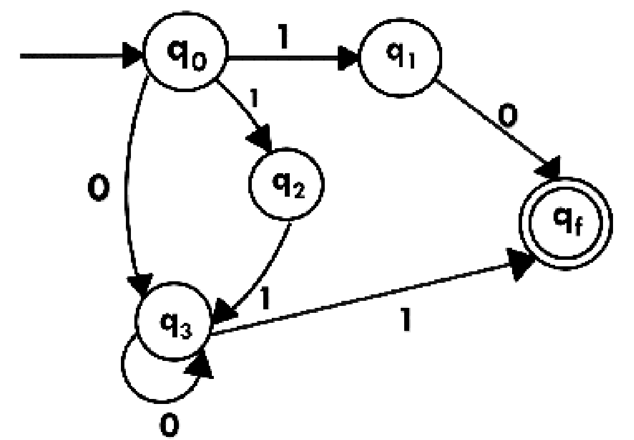
Step-5:



- Now we have got NFA without ε. Now we will convert it into required DFA for that, we will first write a transition table for this NFA.

Conversion of Regular expression into Finite Automata

- **Example : Design a FA from given regular expression 10 + (0 + 11)0* 1.**



Transition table for the NFA:

State	0	1
→q0	q3	{q1, q2}
q1	qf	ϕ
q2	ϕ	q3
q3	q3	qf
*qf	ϕ	ϕ

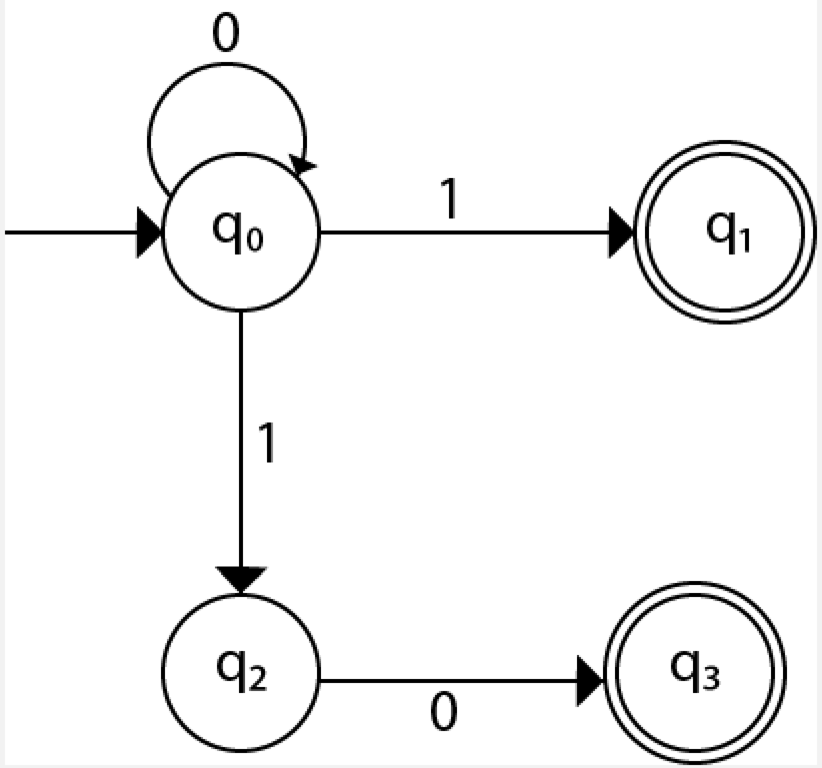
The equivalent DFA will be:

State	0	1
→[q0]	[q3]	[q1, q2]
[q1]	[qf]	ϕ
[q2]	ϕ	[q3]
[q3]	[q3]	[qf]
[q1, q2]	[qf]	[qf]
*[qf]	ϕ	ϕ

Conversion of Regular expression into Finite Automata

- **Example: Construct the FA for regular expression 0*1 + 10.**

NFA:



DFA: ?

Conversion of Finite Automata into Regular Expression

- There are two methods for converting a Finite Automata (FA) to Regular expression (RE).
- These methods are as follows –
    - Arden's Theorem Method.
    - State Elimination Method.

Arden's Theorem Method

- The Arden's Theorem is useful for checking the equivalence of two regular expressions as well as in the conversion of DFA to a regular expression.
- Let us see its use in the conversion of DFA to a regular expression.
- Following algorithm is used to build the regular expression form given DFA.

1. Let q_1 be the initial state.
2. There are q_2, q_3, q_4 ....q_n number of states. The final state may be some q_j where $j<= n$.
3. Let α_{ji} represents the transition from q_j to q_i.
4. Calculate q_i such that

$$q_i = q_j \alpha_{ji}$$

If q_j is a start state then we have:

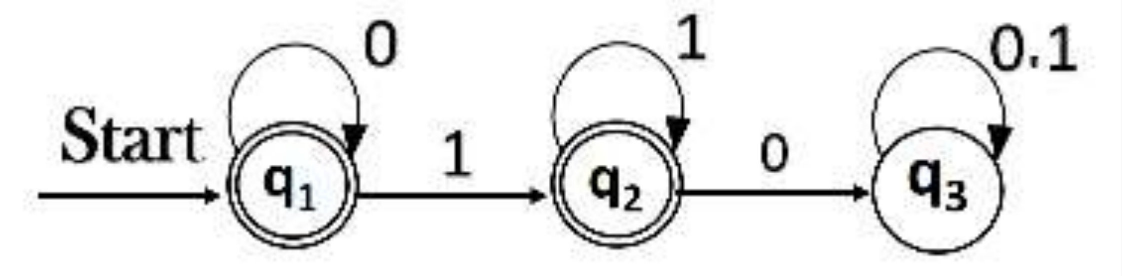
$$q_i = q_j \alpha_{ji} + \varepsilon$$

5. Similarly, compute the final state which ultimately gives the regular expression 'r'.

Arden's Theorem Method: FA to RE

Example:

- Construct the regular expression for the given DFA



Solution:

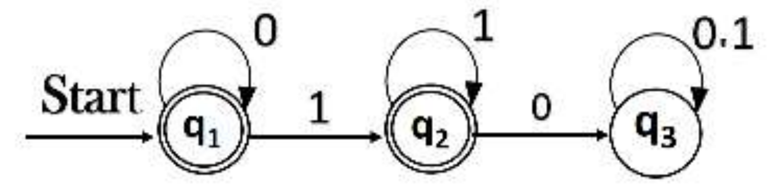
- Let us write down the equations

$$q_1 = q_1 0 + \varepsilon$$

- Since q_1 is the start state, so ε will be added, and the input 0 is coming to q_1 from q_1 hence we write-

State = source state of input × input coming to it

Arden's Theorem Method: FA to RE



- Similarly,

  $q_2 = q_1 1 + q_2 1$

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

- **Since the final states are q_1 and q_2, we are interested in solving q_1 and q_2 only.**
- Let us see q_1 first

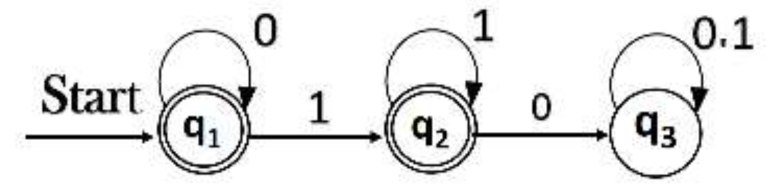
  $q_1 = q_1 0 + \varepsilon$

- We can re-write it as

  $q_1 = \varepsilon + q_1 0$

- Which is similar to R = Q + RP, and gets reduced to R = Q P*.
- Assuming R = q1, Q = ε, P = 0

Arden's Theorem Method: FA to RE



$r = 0^* + 0^* 1^+$

- Assuming $R = q_1$, $Q = \varepsilon$, $P = 0$
- We get

  $q_1 = \varepsilon.(0)^*$

  $= 0^*$ Applying ($\varepsilon.R^* = R^*$)
- Substituting the value into q_2, we will get

  $q_2 = q_1\ 1 + q_2\ 1$

  $= 0^*\ 1 + q2\ 1$

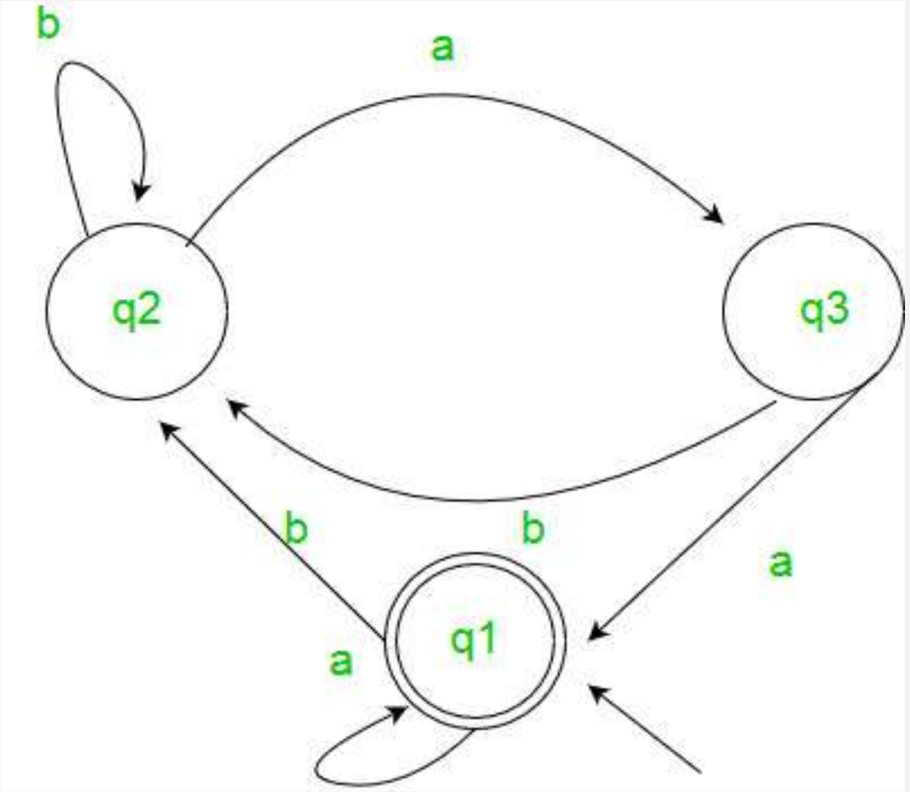
  $= 0^*\ 1\ (1)^*$ Applying ($R = Q + RP \rightarrow Q\ P^*$)
- The regular expression (for final states) is given by

  $r = q1 + q2$

  $= 0^* + 0^*\ 1.1^*$

  $= 0^* + 0^*\ 1^+$ Applying ($1.1^* = 1^+$)
- **Thus, the regular expression is $0^* + 0^*\ 1^+$**

Arden's Theorem Method: FA to RE



Example:

- Construct the regular expression for the given FA.

Solution:

- Here the initial state and the final state is q_1. Other states are q_2 and q_3.
- The equations for the three states q_1, q_2, and q_3 are as follows:

$q_1 = q_1a + q_3a + €$ (€ move is because q_1 is the initial state)
$q_2 = q_1b + q_2b + q_3b$
$q_3 = q_2a$

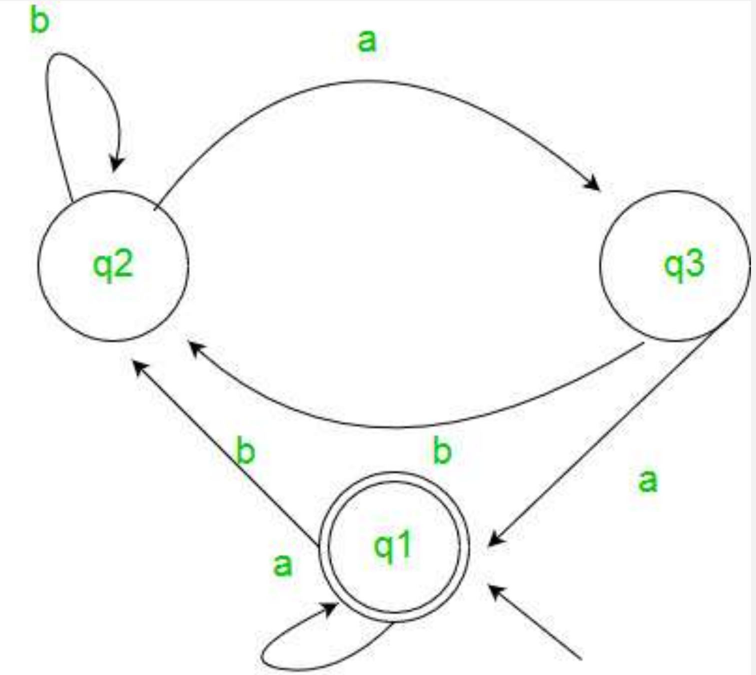
Arden's Theorem Method: FA to RE

- Now, we will solve these three equations.

  $q_2 = q_1b + q_2b + q_3b$
  $= q_1b + q_2b + (q_2a)b$ (Substituting value of q_3)
  $= q_1b + q_2(b + ab)$
  $= q_1b\ (b + ab)^*$ (Applying Arden's Theorem)

  $q_1 = q_1a + q_3a + €$
  $= q_1a + q_2aa + €$ (Substituting value of q_3)
  $= q_1a + q_1b(b + ab^*)aa + €$ (Substituting value of q_2)
  $= q_1(a + b(b + ab)^*aa) + €$
  $= €\ (a+ b(b + ab)^*aa)^*$
  $= (a + b(b + ab)^*aa)^*$

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



(a + b(b + ab)*aa)*

State Elimination Method

- **State elimination method to convert FA to regular expression:**
  - **Step 1:** If the start state is an accepting state or has transitions in, add a new non-accepting start state and add an €-transition between the new start state and the former start state.
  - **Step 2:** If there is more than one accepting state or if the single accepting state has transitions out, add a new accepting state, make all other states non-accepting, and add an €-transition from each former accepting state to the new accepting state.
  - **Step 3:** For each non-start non-accepting state in turn, eliminate the state and update transitions accordingly.

Assignment

- State Elimination Method
  - **State elimination method to convert FA to regular expression.**
  - **Example: Converting FA to RE**

? THE END