Automata Theory

Regular Expressions & Regular Languages

Regular Expressions (REs) & Regular Languages (RLs)

- We used Finite Automata to describe **regular languages**
- We can also use **regular expressions** to **describe regular languages**
- **Regular Expressions are an algebraic way to describe languages**
- Regular Expressions describe exactly the regular languages
- If E is a RE, then L(E) is the regular language that it defines
- For each regular expression E, we can create a DFA A such that L(E) = L(A)
- For each DFA A, we can create a regular expression E such that L(A)=L(E)
- A regular expression is built up of simpler regular expressions using defining rules

For a language to be regular, there must be an automaton that accepts that language.

REs & RLs | Definition

- R is a regular expression if it is:
  - **a** for some a in the alphabet $\sum$, standing for the language {**a**}
  - ε, standing for the language {ε}
  - **Ø**, standing for the **empty language**
  - R1+R2 where R1 and R2 are regular expressions, and + signifies union (sometimes | is used)
  - R1R2 where R1 and R2 are regular expressions and this signifies concatenation
  - R* where R is a regular expression and signifies closure
  - (R) where R is a regular expression, then a parenthesized R is also a regular expression

- This definition may seem circular, but 1-3 form the basis.
- Precedence: Parentheses have the highest precedence, followed by *, concatenation, and then union.
- In simple terms, a regex is a set of strings built from the three **regular operations**, union, concatenation and star

REs & RLs | Definition

Regular expressions over alphabet Σ

	Reg. Expr. E	Language it denotes L(E)
Basis 1:	ϕ	$\{\}$
Basis 2:	ε	$\{\varepsilon\}$
Basis 3:	$a \in \Sigma$	$\{a\}$

$L = \emptyset$	RE is $\emptyset$
$L = \{\lambda\}$	RE = λ
$L = \{a\}$	RE = a
$L = L_1 \cup L_2$	RE = $(r_1 + r_2)$
$L = L_1 L_2$	RE = $(r_1 r_2)$
$L = L_1^*$	RE = (r_1^*)

Note:

{a} is the language containing one string, and that string is of length 1.

REs & RLs | Definition | Regular Operations

Induction 1 – or (union): If $\mathbf{E_1}$ and $\mathbf{E_2}$ are regular expressions, then $\mathbf{E_1+E_2}$ is a regular expression, and $\mathbf{L(E_1+E_2) = L(E_1) \cup L(E_2)}$.

- Sipser,s book use union symbol ∪ to represent **or** operator instead of +. Some people also use bar symbol | to represent **or** operator.

Induction 2 – concatenation: If $\mathbf{E_1}$ and $\mathbf{E_2}$ are regular expressions, then $\mathbf{E_1E_2}$ is a regular expression, and $\mathbf{L(E_1E_2) = L(E_1).L(E_2)}$ where $L(E_1).L(E_2)$ is the set of strings wx such that w is in $L(E_1)$ and x is in $L(E_2)$.

REs & RLs | Definition | Regular Operations

Induction 3 – Kleene Closure: If **E** is a regular expression, then **E*** is a regular expression, and **L(E*) = (L(E))***.

Induction 4 – Parentheses: If **E** is a regular expression, then **(E)** is a regular expression, and **L((E)) = L(E)**.

Operator	*Precedence*	*Associativity*
*	highest	
concatenation	next	left associative
+	lowest	left associative

ab^*+c means **$(a((b)^*))+(c)$**

Parentheses may be used wherever needed to influence the grouping of operators.
We may remove parentheses by using precedence and associativity rules.

REs & RLs | Operations on Languages

- **Remember:** A language is a set of strings
- We can perform operations on languages

Union: $L \cup M = \{ w : w \in L \text{ or } w \in M \}$

Concatenation: $L.M = \{ w : w=xy, x \in L, y \in M \}$

Powers: $L^0 = \{ \varepsilon \}, \quad L^1 = L, \quad L^{k+1} = L.L^k$

Kleene Closure: $L^* = \bigcup_{i=0}^{\infty} L^i$

REs & RLs | Operations on Languages | Examples

$L = \{00,11\}$ $M = \{1,01,11\}$

$L \cup M = \{00,11,1,01\}$

$L.M = \{001,0001,0011,111,1101,1111\}$

$L^0 = \{\varepsilon\}$ $L^1 = L = \{00,11\}$ $L^2 = \{0000,0011,1100,1111\}$

$L^* = \{\varepsilon, 00, 11, 0000, 0011, 1100, 1111, 000000, 000011, ...\}$

Kleene closures of all languages (except two of them) are infinite.

1. $\phi^* = \{\}^* = \{\varepsilon\}$
2. $\{\varepsilon\}^* = \{\varepsilon\}$

REs & RLs | Examples

Regular Expressions	Regular Set
(0 + 10*)	L = { 0, 1, 10, 100, 1000, 10000, ... }
(0*10*)	L = {1, 01, 10, 010, 0010, ...}
(0 + ε)(1 + ε)	L = {ε, 0, 1, 01}
(a+b)*	Set of strings of a's and b's of any length including the null string. So L = { ε, a, b, aa , ab , bb , ba, aaa.....}
(a+b)*abb	Set of strings of a's and b's ending with the string abb. So L = {abb, aabb, babb, aaabb, ababb, ...}
(11)*	Set consisting of even number of 1's including empty string, So L= {ε, 11, 1111, 111111, ...}
(aa)*(bb)*b	Set of strings consisting of even number of a's followed by odd number of b's , so L = {b, aab, aabbb, aabbbbb, aaaab, aaaabbb, ...}
(aa + ab + ba + bb)*	String of a's and b's of even length can be obtained by concatenating any combination of the strings aa, ab, ba and bb including null, so L = {aa, ab, ba, bb, aaab, aaba, ...}

Any set that represents the value of the RE at hand is called a **Regular Set.**

REs & RLs | Examples

- $01^* = \{0, 01, 011, 0111, \ldots.\}$
- $0^* + 1 = \{\varepsilon, 0, 1, 00, 000, 0000, \ldots\}$
- $(01^*)(01) = \{001, 0101, 01101, 011101, \ldots\}$
- $(0+1)^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, \ldots\} \rightarrow$ *all possible strings of 0 and 1*
- $(0+1)^*00(0+1)^* = \{00, 100, 1001, 000, 0001, \ldots\} \rightarrow$ *all strings of 0 and 1 including substring 00*
- $(0+1)^*00(0+1)^* + (0+1)^*11(0+1)^* \rightarrow$ *all strings of 0 and 1 containing substring 00 or 11*
- $(1+10)^* \rightarrow$ *all strings starting with {ε} or 1 but not containing substring 00*
- $(0+1)^*011 \rightarrow$ *all strings ending in 011*
- $0^*1^* = \{\varepsilon, 0, 00, 000, \ldots, 1, 11, 111, \ldots, 01, 001, 011 \ldots\} \rightarrow$ *all strings not containing 1 followed by 0*

REs & RLs | RL for Given RE | Examples

Alphabet $\Sigma = \{0,1\}$

Regular Expression: **01**

- $L(\mathbf{01}) = \{01\}$

$L(\mathbf{01}) = L(\mathbf{0})\, L(\mathbf{1}) = \{0\}\{1\} = \{01\}$

Regular Expression: **01+0**

- $L(\mathbf{01+0}) = \{01, 0\}$

$L(\mathbf{01+0}) = L(\mathbf{01}) \cup L(\mathbf{0}) = (L(\mathbf{0})\, L(\mathbf{1})) \cup L(\mathbf{0})$

$= (\{0\}\{1\}) \cup \{0\} = \{01\} \cup \{0\} = \{01,0\}$

Regular Expression: **0(1+0)**

- $L(\mathbf{0(1+0)}) = \{01, 00\}$

$L(\mathbf{0(1+0)}) = L(\mathbf{0})\, L(\mathbf{1+0}) = L(\mathbf{0})\, (L(\mathbf{1}) \cup L(\mathbf{0}))$

$= \{0\}\, (\{1\} \cup \{0\}) = \{0\}\, \{1,0\} = \{01,00\}$

- Note order of precedence of operators.

REs & RLs | RL for Given RE | Examples

Regular Expression: **0***

- L(**0***) = {ε, 0, 00, 000,… } = all strings of 0's, including the empty string

Regular Expression: **(0+10)*(ε+1)**

- L((**0**+**10**)*(ε+**1**)) = all strings of 0's and 1's without two consecutive 1's.

Regular Expression: **(0+1)(0+1)**

- L(**(0+1)(0+1)**) = {00,01,10,11} = all strings of 0's and 1's of length 2.

Regular Expression: **(0+1)***

- L(**(0+1)***) = all strings with 0 and 1, including the empty string

REs & RLs | RE for Given RL | Examples

Language: All strings of 0's and 1's starting with 0 and ending with 1

$$\mathbf{0(0+1)^*1}$$

Language: All strings of 0's and 1's with at least two consecutive 0's

$$\mathbf{(0+1)^*00\ (0+1)^*}$$

Language: All strings of 0's and 1's without two consecutive 0's

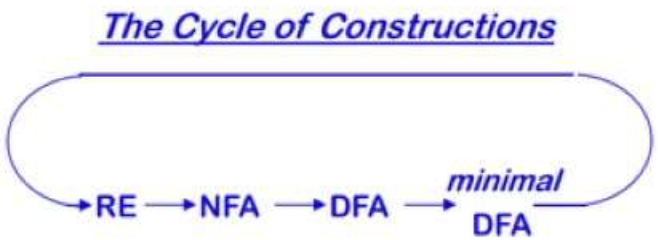
$$((\mathbf{1+01})^*(\varepsilon+\mathbf{0}))$$

Language: All strings of 0's and 1's with even number of 0's

$$\mathbf{1^*(01^*01^*)^*}$$

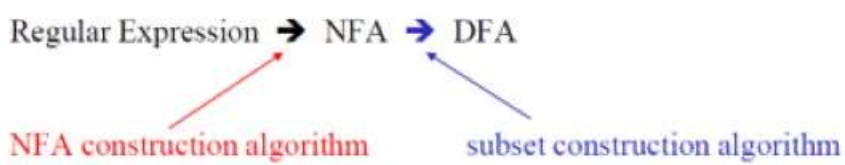
REs & RLs | Relationship with Finite Automata

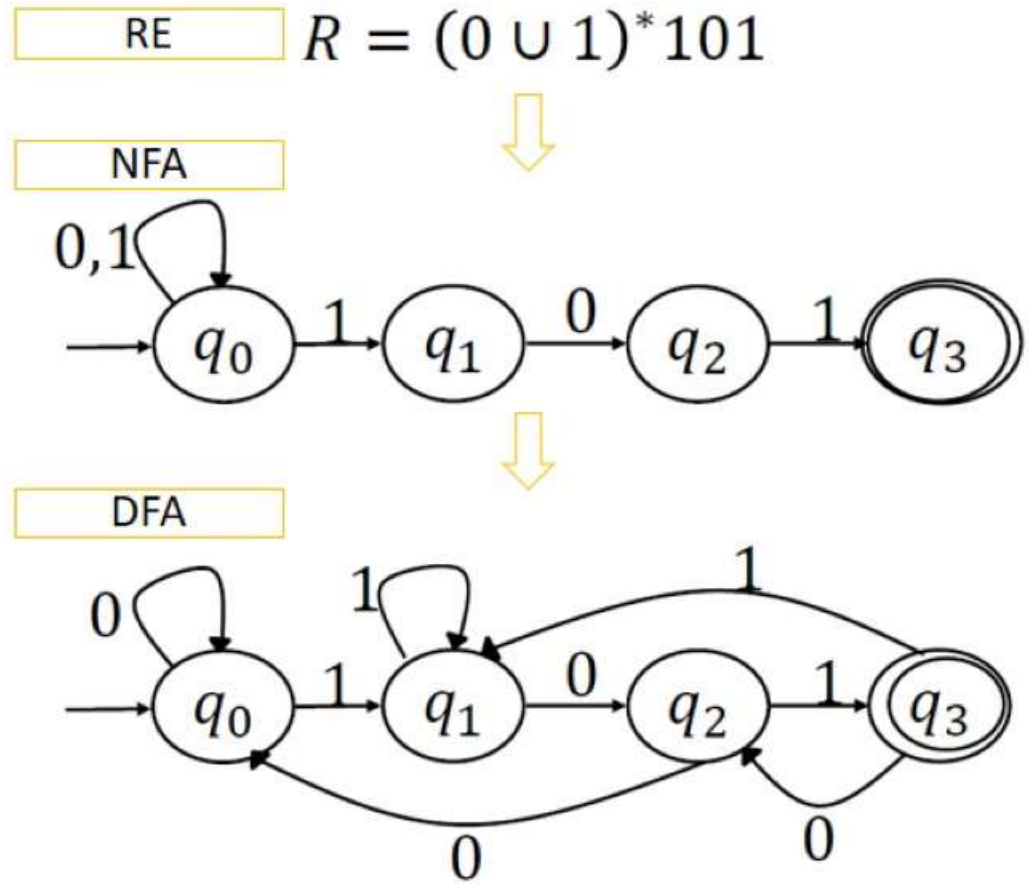
- The last piece of the puzzle is to show that **regular expressions are equivalent to automata**.
- Regular expressions and finite state automata are really **two different ways of expressing the same thing.**
- In some cases you may find it easier to start with one and move to the other
  - E.g., the language of an even number of one's is typically easier to design as a NFA or DFA and then convert it to a RE
- For every RE E, there exists an **ε-NFA** A such that L(E)=L(A).
- For every **DFA** A, there exists a RE E such that L(A)=L(E).



REs & RLs | Convert RE to NFA

- For every RE there is a finite automaton.
- We will give an algorithm which converts a given RE to a NFA.
- We have already discussed how to convert a NFA to a DFA using **subset construction**.
  - Thus, there is an NFA for each RE and their languages are equivalent.
  - And, there is a DFA for each RE and their languages are equivalent.





REs & RLs | Convert RE to NFA

- **Theorem:** Every language defined by a **RE** is also defined by a **FA**.
  - This theorem says that every language represented by a RE is a regular language
    - i.e. there is a DFA which recognizes that language
  - In the proof of this theorem, we will create a NFA which recognizes the language of a given RE.
  - **This means that any language represented by an RE can be recognized by an NFA.**
    - Previously, we show how to create an equivalent DFA for a given NFA.
    - This means that any language recognized by a NFA can be recognized by a DFA.

Regular Expressions ⟹ NFA ⟹ DFA ⟹ Regular Languages

Regular Expressions ⟹ Regular Languages

REs & RLs | Convert RE to NFA

- **Theorem:** Every language defined by an **RE** is also defined by an **FA**.
- **Proof:** Suppose that L(R) is the language of a RE R
- An ε-NFA construction for a RE: We show that an ε-NFA A whose language L(A) accepts L(R)
- i.e., L(A) is equal to L(R) and this FA has the following properties:
  - NFA has exactly one accepting state
  - No arcs into the initial state
  - No arcs out of the accepting state
- The proof is by structural induction on R following the recursive definition of REs

REs & RLs | Convert RE to NFA

- These rules are called to be the algorithm of **Thompson's construction** that have the following superficial steps:
  - **Step 1:** Construct an NFA with Null moves from the given regular expression.
  - **Step 2:** Remove Null transition from the NFA and convert it into its equivalent DFA.
- Thompson's construction algorithm, also called the McNaughton–Yamada–Thompson algorithm, is a method of transforming a RE into an equivalent NFA
  - An NFA can be made deterministic by the **powerset construction** and then be minimized to get an optimal automaton corresponding to the given regular expression. However, an NFA may also be interpreted directly.
  - To decide whether two given REs describe the same language, each can be converted into an equivalent minimal DFA via Thompson's construction, powerset construction, and DFA minimization.
  - If, and only if, the resulting automata agree up to renaming of states, the REs' languages agree.

Thompson's construction

- Core insight: inductively build up NFA using **basis rules**
- Core concept: use null transitions to build NFA quickly

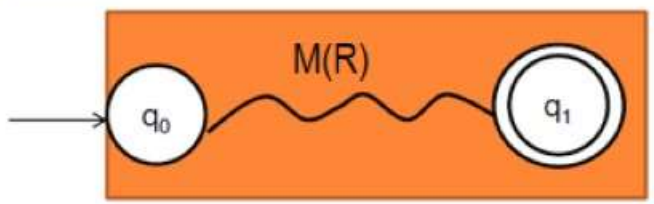
REs & RLs | Convert RE to NFA

- **Thompson's construction:** Begin by parsing any given RE R into its constituent subexpressions.
- The **rules** for constructing an NFA consist of
  - **basis rules** for handling subexpressions with no operators, and
  - inductive rules for constructing larger NFA's from the NFA's for the immediate subexpressions of a given RE.

REs & RLs | Convert RE to NFA | Basis

- The conversion is by structural induction on R
- The **basis** of the construction of FSA from regular expressions:

  - $R = \varepsilon$
  - $R = \emptyset$
  - $R = a \in \sum$



All these automata satisfies the three initial conditions

Regular Expression R = ε — **L(ε) = {ε}**

NFA A:

L(A) = {ε}

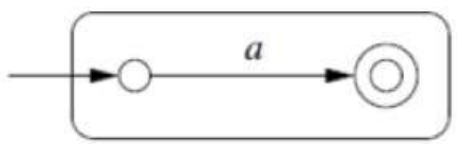
Regular Expression R = ϕ — **L(ϕ) = {}**

NFA A:

L(A) = {}

Regular Expression R = a ∈ Σ — **L(a) = {a}**

NFA A:



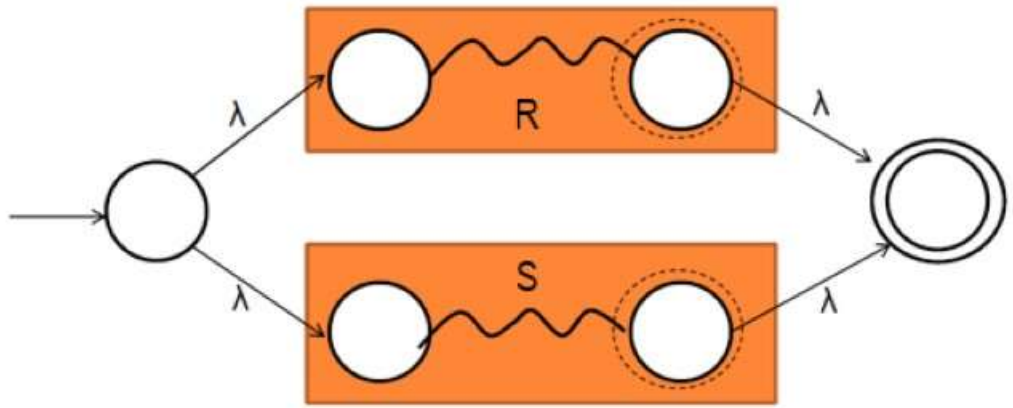
L(A) = {a}

REs & RLs | Convert RE to NFA | Proof by Induction

- **Inductive Hypothesis**
  - We assume that the statement of the theorem is true for immediate subexpressions of a given regular expression
  - i.e. the languages of these subexpressions are also the languages of NFAs with a single accepting state.
- **Induction**
  - There are four cases for the induction:
    - R + S
    - RS
    - R*
    - (R)

REs & RLs | Convert RE to NFA | Induction Case: R + S

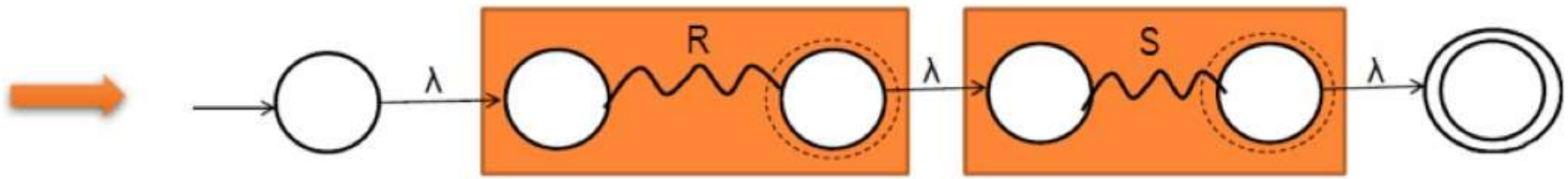
Regular Expression: R + S
L(R + S) = L(R) ∪ L(S)
Epsilon NFA A



- By IH, we have automaton R for regular expression R, and automaton S for regular expression S, and a new automaton for R+S is constructed as given on the left
- Starting at new start state, we can go to start states of automatons R or S
- For some string in L(R) or L(S), we can reach accepting state of R or S
- From there, we can reach accepting state of the new automaton by ε transition.
- **Thus, L(A) = L(R) ∪L(S)**

REs & RLs | Convert RE to NFA | Induction Case: RS

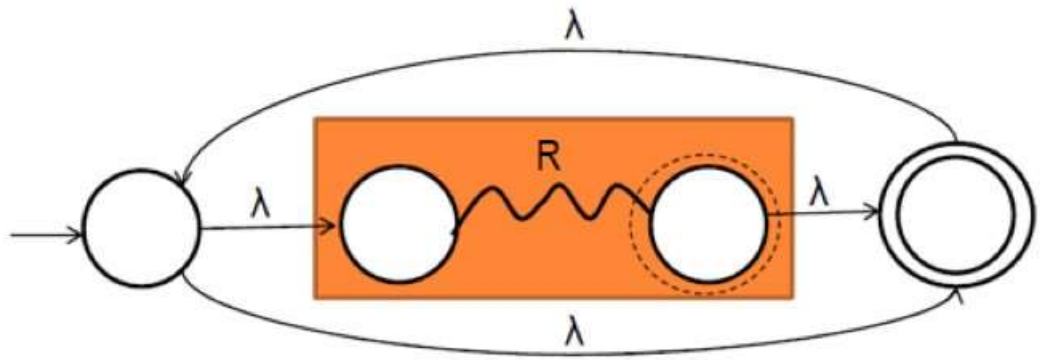
Regular Expression: RS
L(RS) = L(R)L(S)
Epsilon NFA A

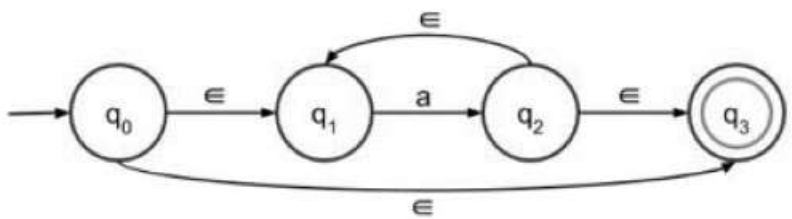


- A new automaton for RS is constructed as given on the left
- Starting at starting state of R, we can reach accepting state of R by recognizing a string in L(R).
- From accepting state of R, we can reach starting state of S by ε-transition.
- From starting state of S, we can reach accepting state of S by recognizing a string in L(S).
- The accepting state of S is also the accepting state of the new automaton A.
- **Thus, L(A) = L(R)L(S)**

REs & RLs | Convert RE to NFA | Induction Case: R*

Regular Expression: R*
L(R*) = (L(R))*
Epsilon NFA A



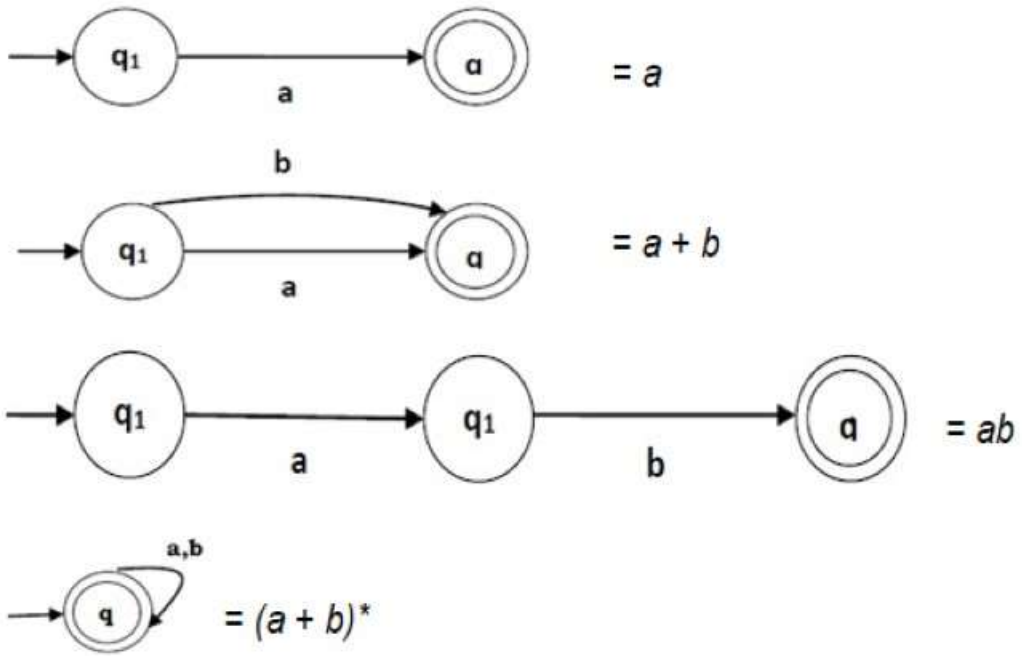


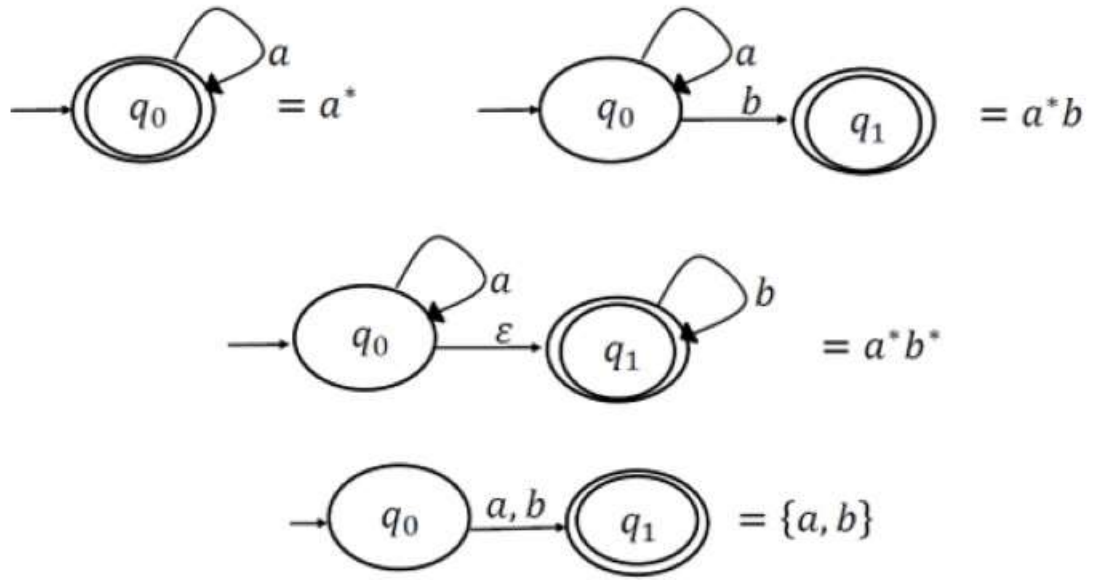
- A new automaton for R* is constructed as given on the top-right
- Starting at new starting state, we can reach new accepting state.
  - ε is in (L(R))*
- Starting at new starting state, we can reach starting state of R.
- From starting state of R, we can reach accepting state of R recognizing a string in L(R).
  - We can repeat this one or more times by recognizing strings in L(R), L(R)L(R),….
- **Thus, L(A) = (L(R))***

REs & RLs | Convert RE to NFA | Induction Case: (R)

- By IH, we have automaton R for regular expression R, and a new automaton for (R) is same as the automaton of R
- The automaton for R also serves as the automaton for (R) since the parentheses **do not change the language** defined by the expression.

REs & RLs | Convert RE to NFA | Examples

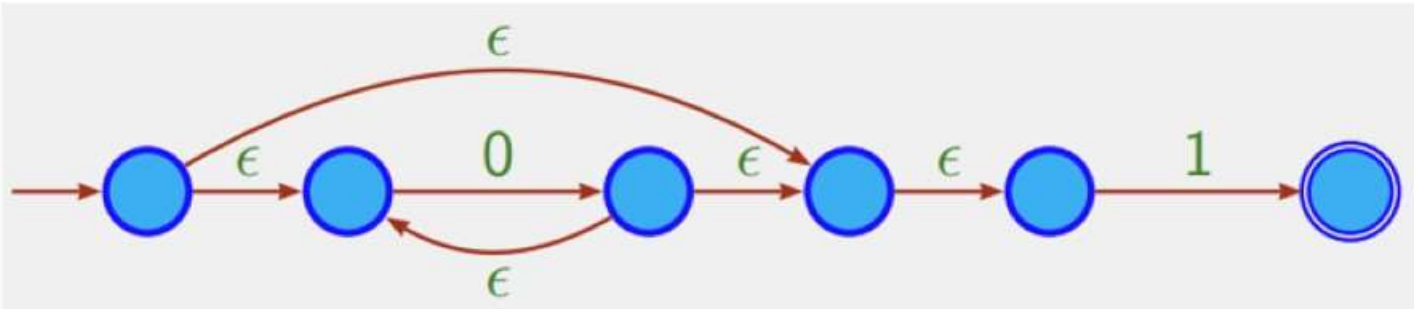




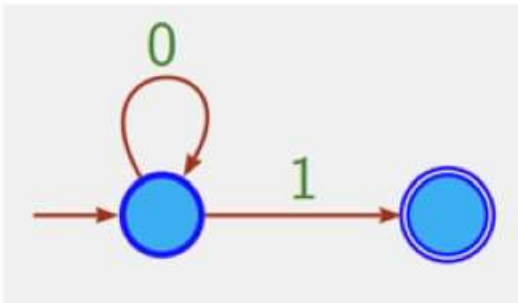
We can reduce the regular expression into smallest regular expressions and converting these to NFA and finally to DFA.

REs & RLs | Convert RE to NFA | Examples

- Let us follow the previous method to construct an epsilon-NFA for RE **0*1**

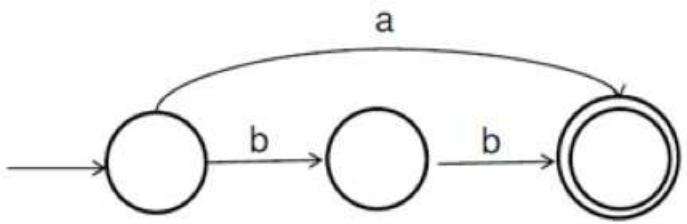


- Compare this machine with the following FA for the same language:



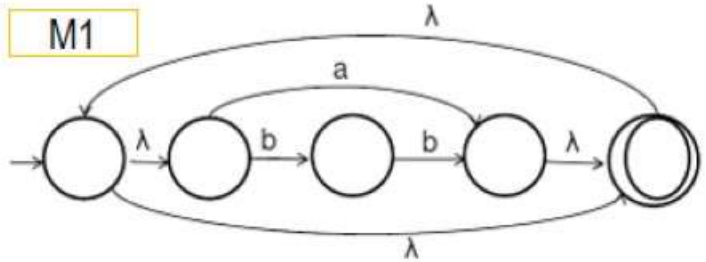
REs & RLs | Convert RE to NFA | Examples

- Convert **(a+bb)*(ba*+λ)** to NFA



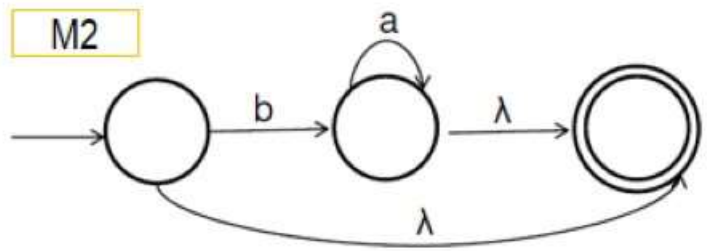
Automata for **L(a+bb)**

M1

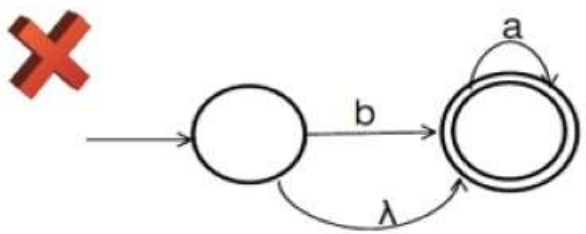


Automata for **L(a+bb)***

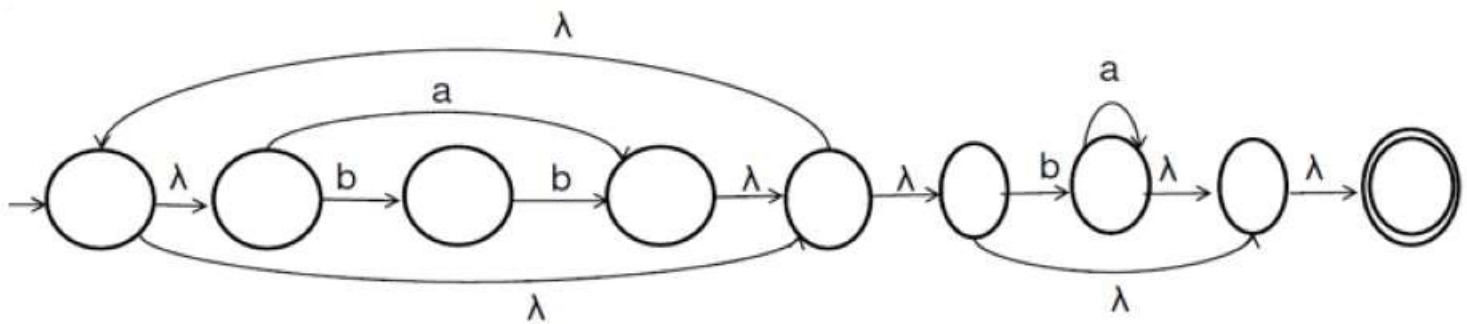
M2



Automata for **L(ba* + λ)**



This machine can not be **M2** since it also accepts the language of a*

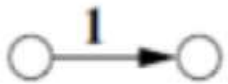


Combine M1 with M2 → Automata for **L((a+bb)*(ba*+λ))**

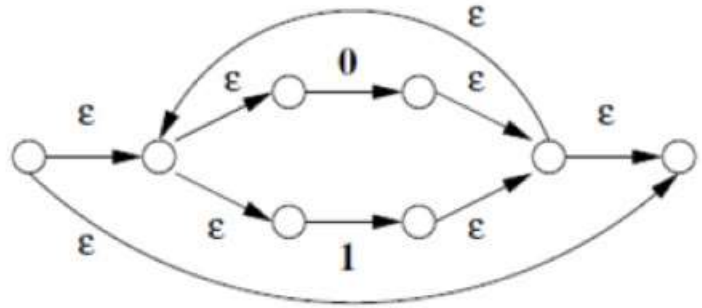
REs & RLs | Convert RE to NFA | Examples

- Convert **(0+1)*1** to NFA

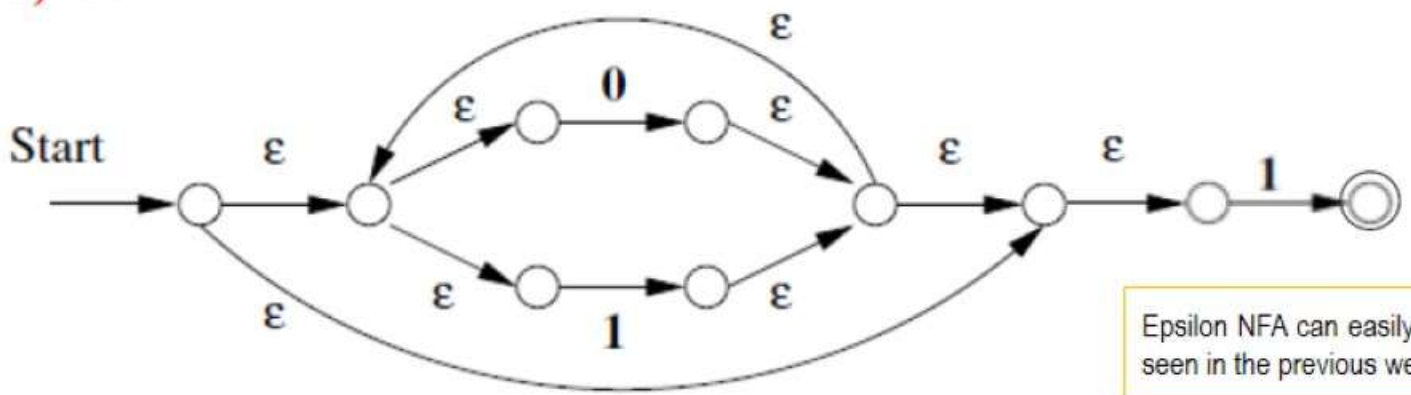
Automaton for 1:



Automaton for (0+1)*:



Automaton for (0+1)*1:



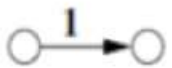
Epsilon NFA can easily be converted to DFA as we seen in the previous weeks

REs & RLs | Convert RE to NFA | Examples

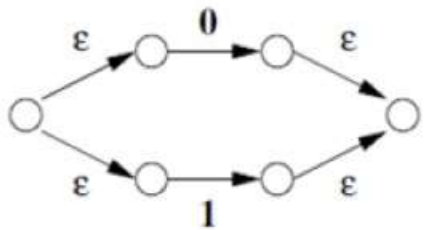
- Convert **(0+1)*1(0+1)** to NFA

Automaton for 0:

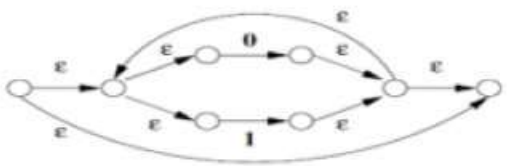
Automaton for 1:



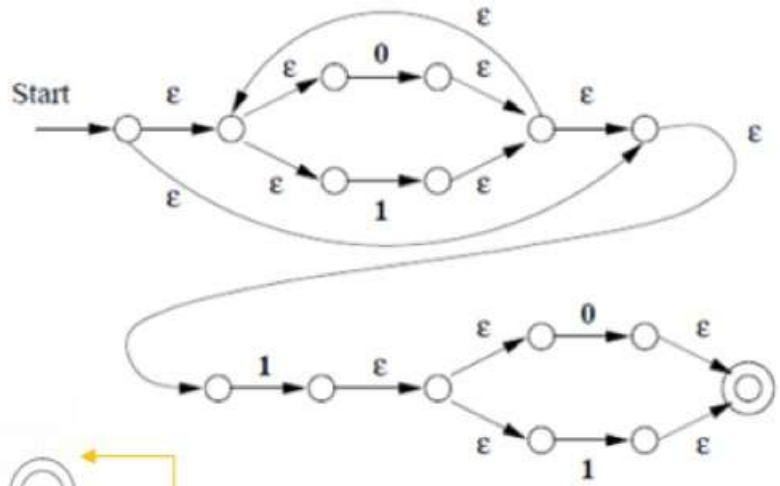
Automaton for 0+1:

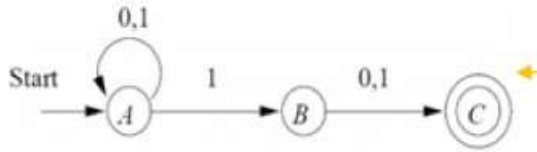


Automaton for (0+1)*:



Automaton for (0+1)*1(0+1) :





- Convert it to equivalent DFA or just remove redundant ε transitions!
- Epsilon NFA can easily be converted to DFA as we seen in the previous weeks

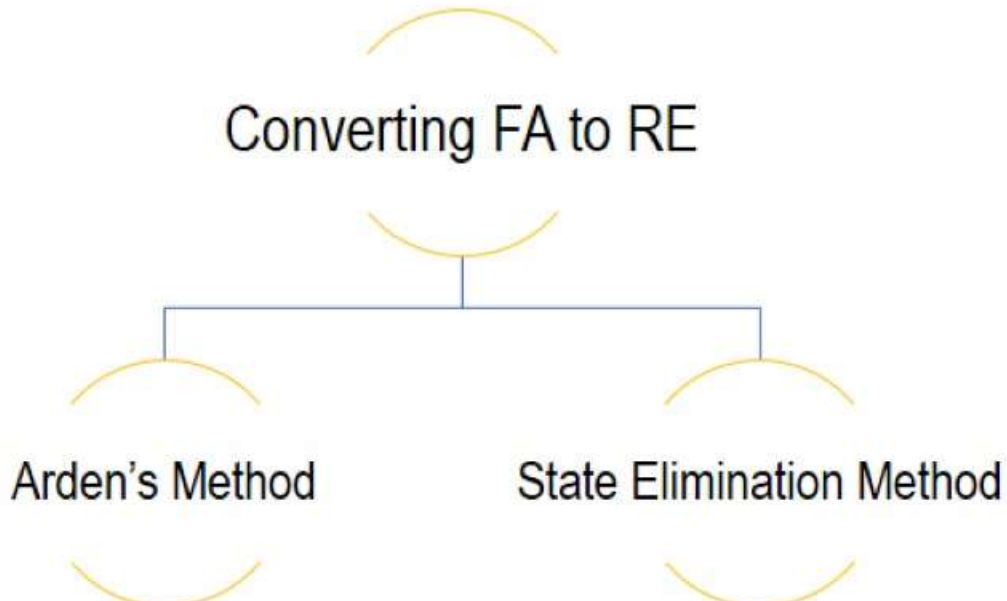
REs & RLs | Convert FA to RE

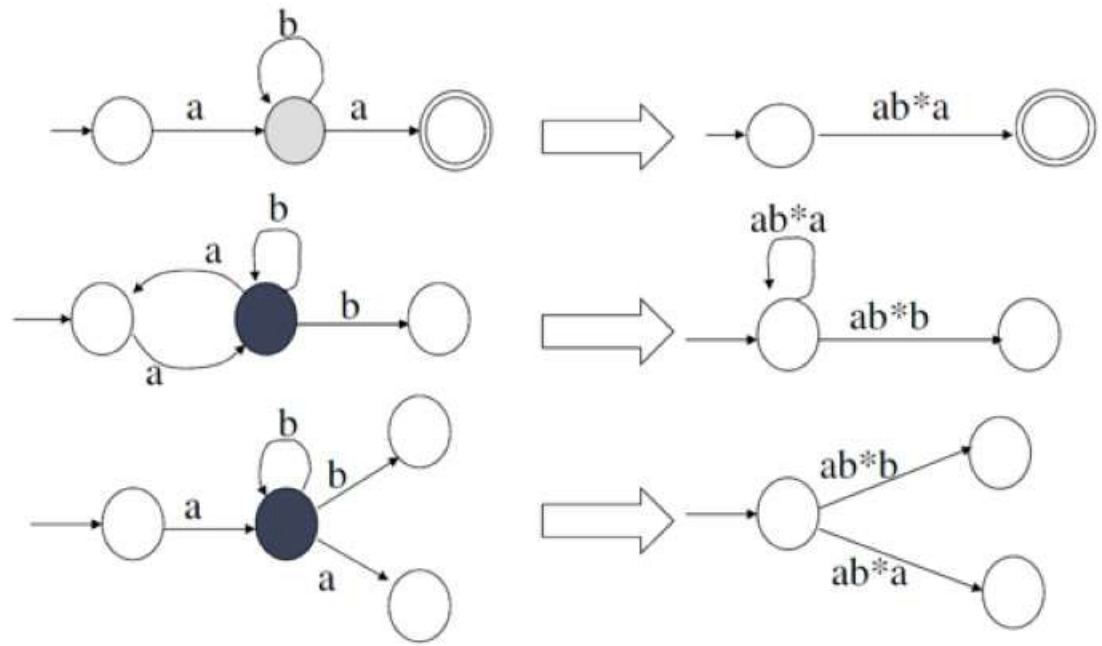
- **Theorem:** If a language is regular, then it is described by a regular expression
- In order to prove this theorem, we will create a regular expression for any given DFA and the language of this regular expression is equivalent to the language of that DFA
  - Since a regular language is described by a DFA, a regular language is also described by a regular expression

Regular Languages ⟹ DFA ⟹ Regular Expressions

Regular Languages ⟹ Regular Expressions

REs & RLs | Convert FA to RE | Methods



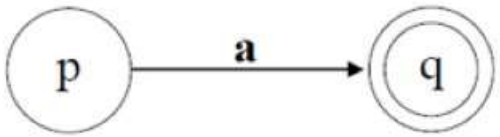


REs & RLs | Convert FA to RE | State Elimination

- To create a regular expression which describes the language of the given DFA:
  - First we create a **Generalized NFA** (GNFA) from the given DFA
  - A GNFA has **generalized transitions** and a **generalized transition** is a transition whose label is a regular expression
  - Then, we will iteratively **eliminate states** of the GNFA one by one, until two states (i.e., start state and accepting state) and a single generalized transition is left
  - The label of this single transition (a regular expression) will be the regular expression that describes the language of the given DFA

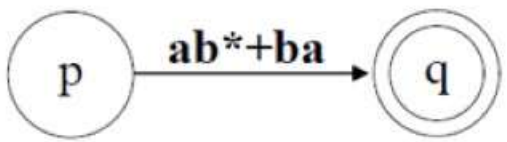
REs & RLs | Convert FA to RE | State Elimination

- When a DFA has single symbols as transition labels:



  - If we are in state **p** and the next input symbol matches **a**, go to state **q.**

- Now , look at a **generalized transition**:

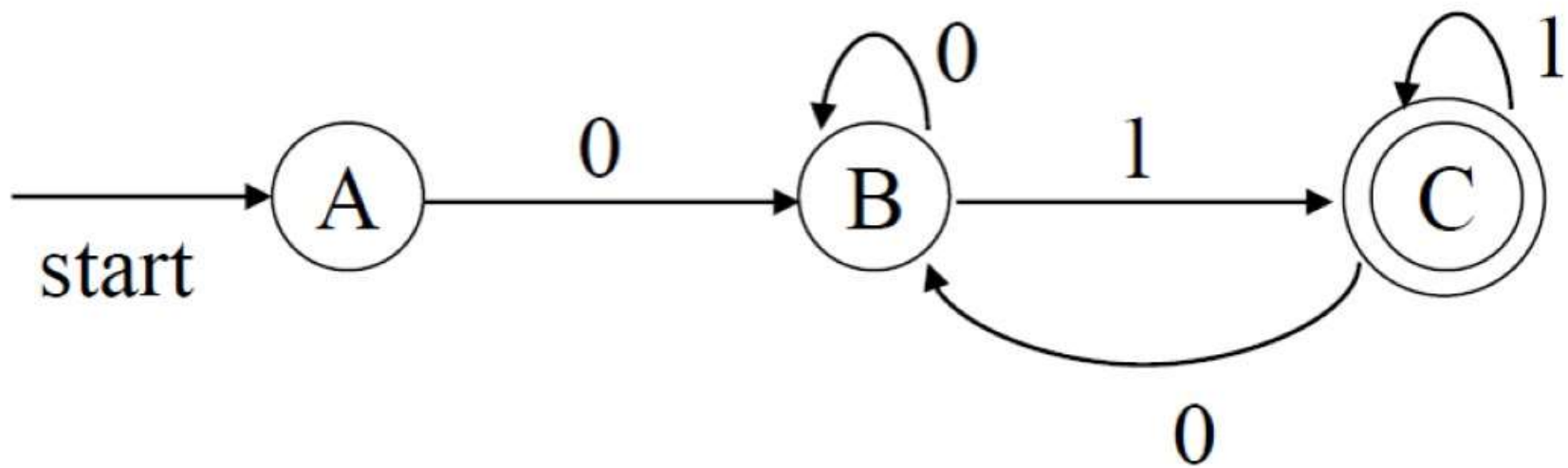


  - If we are in state p and **a prefix of the remaining input** matches the regular expression **ab*+ba** then go to state q.

  - A **generalization transition** is a transition **whose label is a regular expression**.

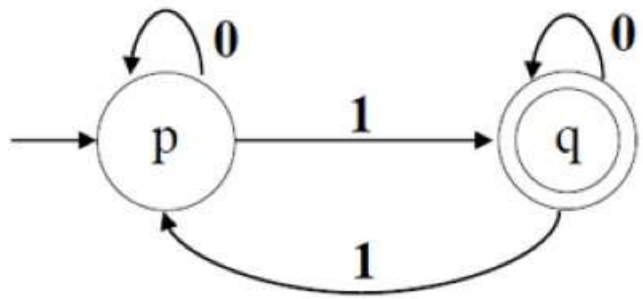
REs & RLs | Convert FA to RE | State Elimination

- A **Generalized NFA** (GNFA) is an NFA with generalized transitions
- In fact, all standard DFA transitions with single symbols are generalized transitions with regular expressions of a single symbol!

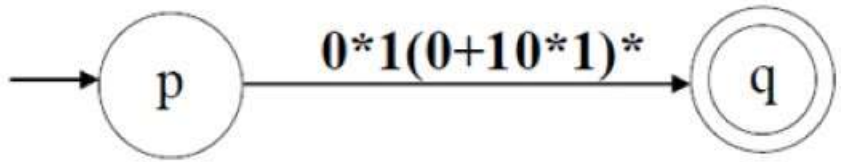


REs & RLs | Convert FA to RE | State Elimination

- Consider the following DFA.



- What will be the corresponding GNFA with two states (start state and an accepting state) with a single generalized transition.
  - **0*1** takes the DFA from state p to q
  - **(0+10*1)*** takes the DFA from q back to q
  - So, **0*1(0+10*1)*** represents all strings take the DFA from state p to q.

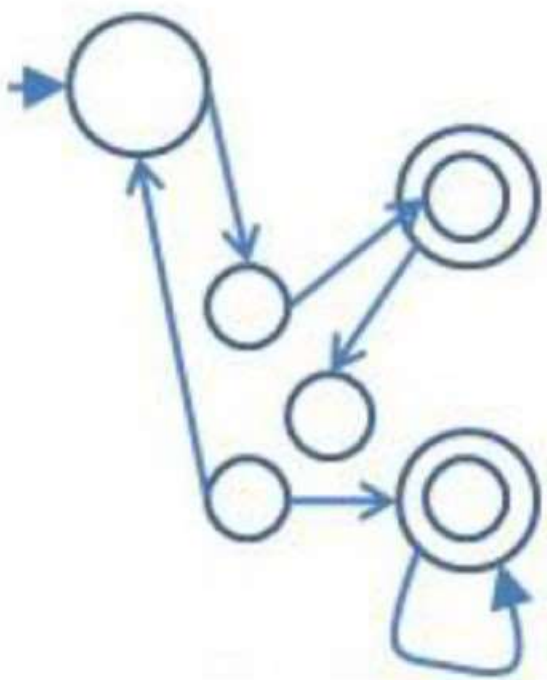


REs & RLs | Convert FA to RE | State Elimination

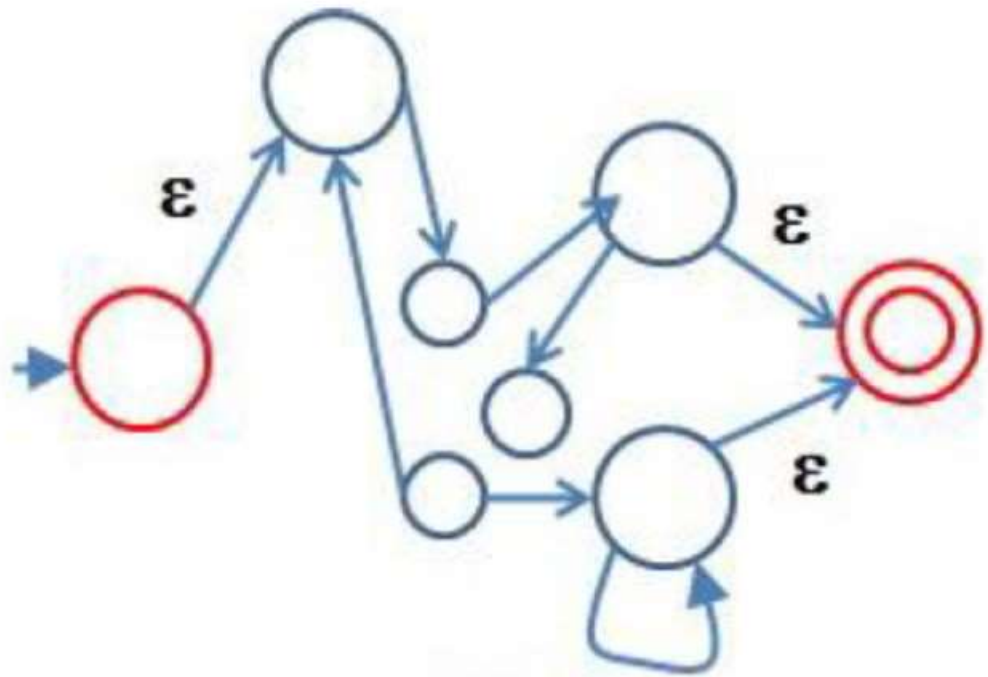
- We will convert the given DFA to a GNFA in a **special form**. We will **add two new states** to a DFA:
  - A **new start state** with an ε-transition to the original start state, but there will be **no other transitions from any other state to this new start state**
  - A **new final state** with an ε-transition from all the original final states, but there will be **no other transitions from this new final state to any other state**
- If the label of the DFA is a single symbol, the corresponding label of the GNFA will be that single symbol: **0** $\longrightarrow$ **0**
- If there are more than one symbol on the label of the DFA , the corresponding label of the GNFA will be **union** (OR) of those symbols: **0, 1** $\longrightarrow$ **0 + 1**
- The previous start and final states will be non accepting states in this GNFA.

REs & RLs | Convert FA to RE | State Elimination

DFA

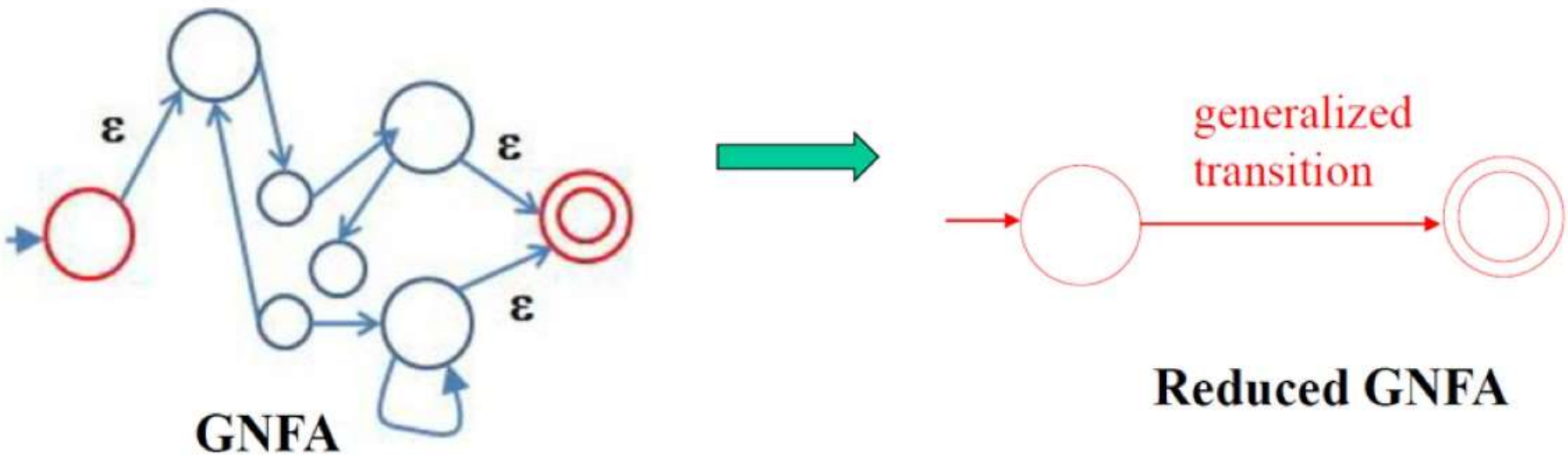


GNFA in a special form



REs & RLs | Convert FA to RE | State Elimination

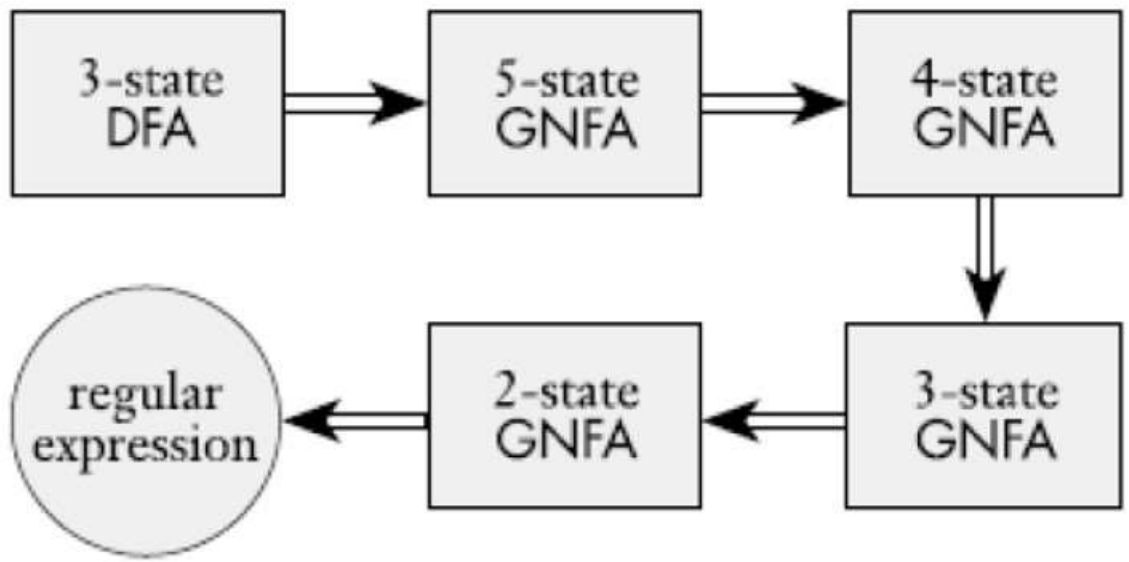
- We eliminate all states of the GNFA **one by one** leaving only the start state and the final state



- When the GNFA is fully converted, **the label of the only generalize transition is the regular expression** for the language accepted by the original DFA.

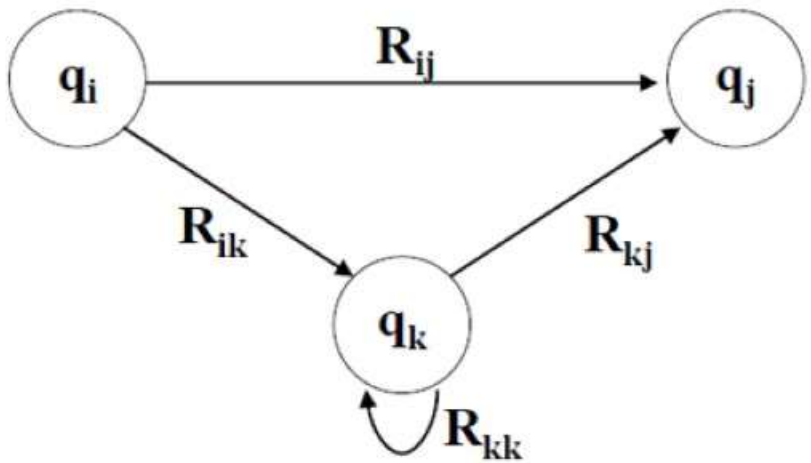
REs & RLs | Convert FA to RE | State Elimination

- Assume that our DFA has 3 states.
  - Create a GNFA with 5 states in a special form.
  - Eliminate a state on by one until we obtain a GNFA with two states (start state and final state).
  - Label on the arc is the regular expression describing the language of the DFA.

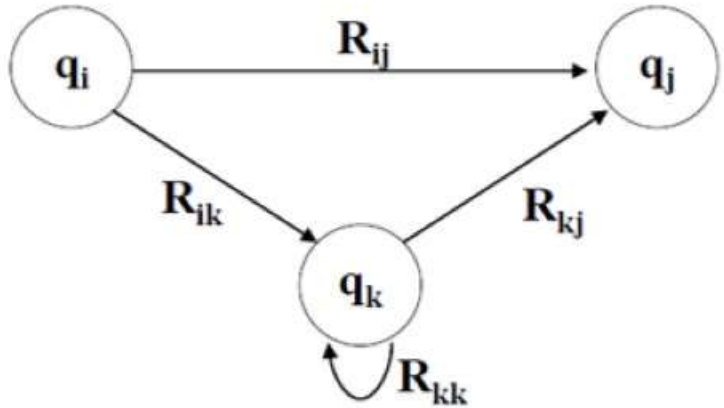


REs & RLs | Convert FA to RE | State Elimination

- Suppose we want to eliminate state $\mathbf{q_k}$, and $\mathbf{q_i}$ and $\mathbf{q_j}$ are two of the remaining states (i=j is possible; i.e. $\mathbf{q_i}$ can be equal to $\mathbf{q_j}$).



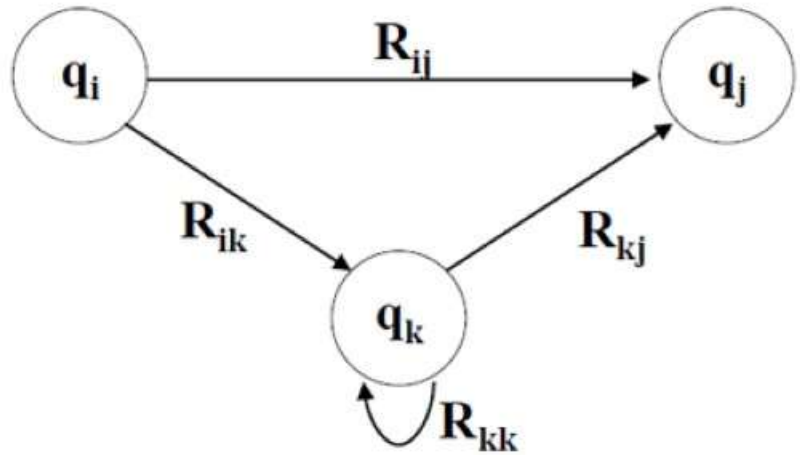
REs & RLs | Convert FA to RE | State Elimination



- How can we modify the transition label between $\mathbf{q_i}$ and $\mathbf{q_j}$ to reflect the fact that $\mathbf{q_k}$ will no longer be there?
  - There are two paths between $\mathbf{q_i}$ and $\mathbf{q_j}$
    - Direct path with regular expression $\mathbf{R_{ij}}$
    - Path via $\mathbf{q_k}$ with the regular expression $\mathbf{(R_{ik})\ (R_{kk})^*\ (R_{kj})}$

REs & RLs | Convert FA to RE | State Elimination

- There are two paths between $\mathbf{q_i}$ and $\mathbf{q_j}$
  - Direct path with regular expression $\mathbf{R_{ij}}$
  - Path via $\mathbf{q_k}$ with the regular expression $\mathbf{(R_{ik})\ (R_{kk})^*\ (R_{kj})}$

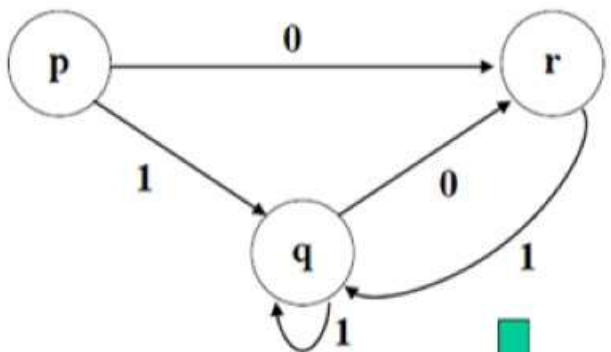


- After removing $\mathbf{q_k}$,the new label would be

$$\textbf{new } (R_{ij}) = (R_{ij}) + (R_{ik})\ (R_{kk})^*\ (R_{kj})$$

REs & RLs | Convert FA to RE | State Elimination

- When we are eliminating a state **q**, we have to update labels of state pairs **p** and **r** such that there is a transition from **p** to **q** and there is a transition from **q** to **r**.
  - **p** and **r** can be same state.
- Missing arc labels are **ϕ**

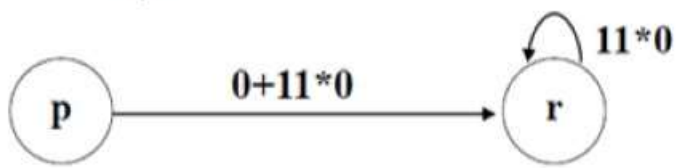


$R_{pp} = R_{pp} + R_{pq} (R_{qq})^* R_{qp} = \phi + 1(1)^* \phi = \mathbf{\phi}$

$R_{pr} = R_{pr} + R_{pq} (R_{qq})^* R_{qr} = 0 + 1(1)^*0 = \mathbf{0+11^*0}$

$R_{rr} = R_{rr} + R_{rq} (R_{qq})^* R_{qr} = \phi + 1(1)^*0 = \mathbf{11^*0}$

$R_{rp} = R_{rp} + R_{rq} (R_{qq})^* R_{qp} = \phi + 1(1)^* \phi = \mathbf{\phi}$



REs & RLs | Convert FA to RE | State Elimination

- **Some simplification rules for REs**

$\phi^* = \varepsilon$

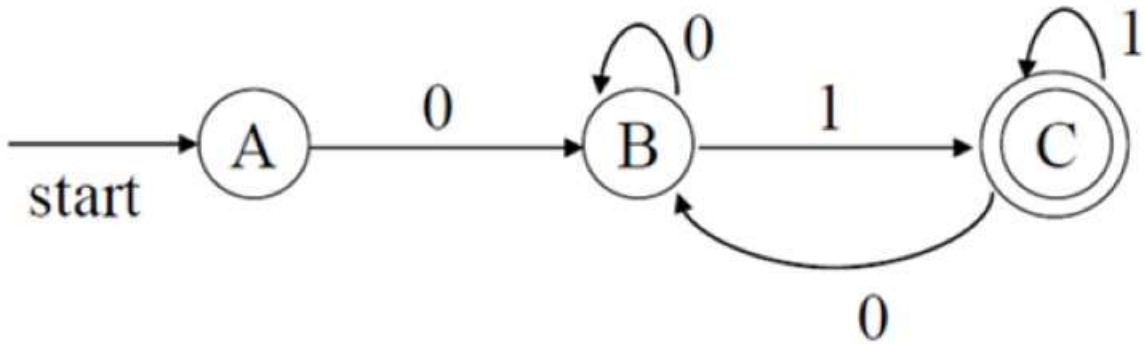
$\varepsilon^* = \varepsilon$

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

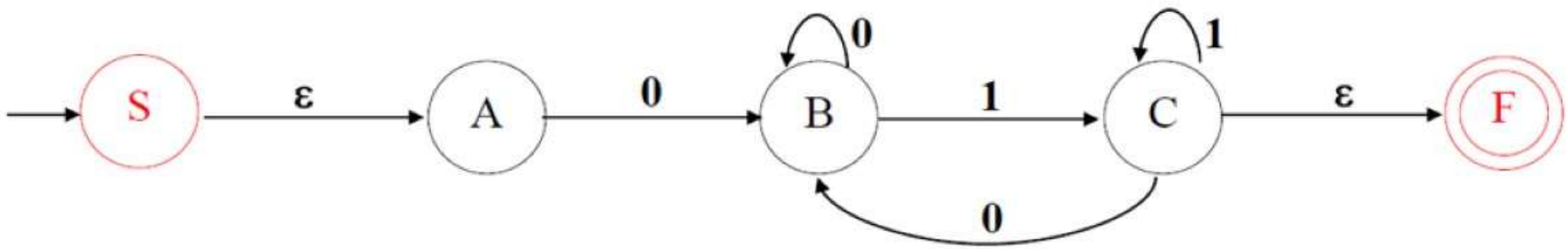
$\varepsilon R = R\varepsilon = R$	ε is the identity for concatenation.
$\phi R = R\phi = \phi$	ϕ is an annihilator for concatenation.
$\phi+R = R+\phi = R$	ϕ is the identity for union.

REs & RLs | Convert FA to RE | State Elimination | Example

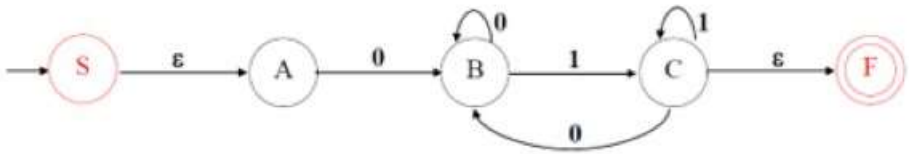
A DFA



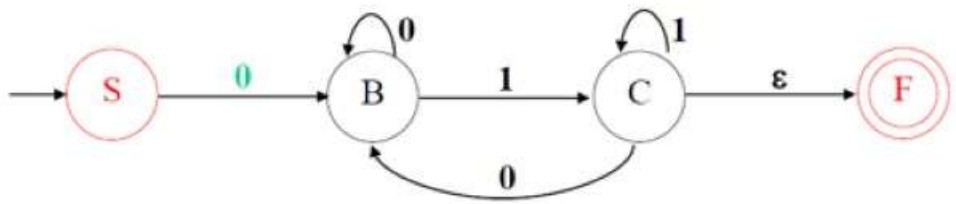
A GNFA in a special form:



REs & RLs | Convert FA to RE | State Elimination | Example

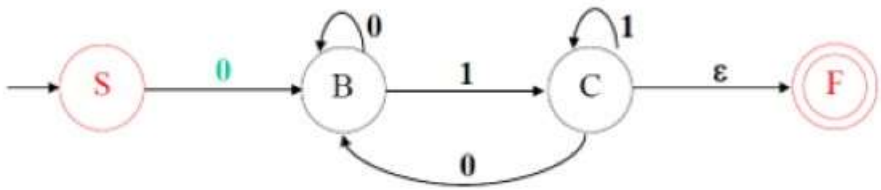


new $R_{SB} = R_{SB} + R_{SA} (R_{AA})^* R_{AB} = \phi + \varepsilon (\phi)^* 0 = 0$



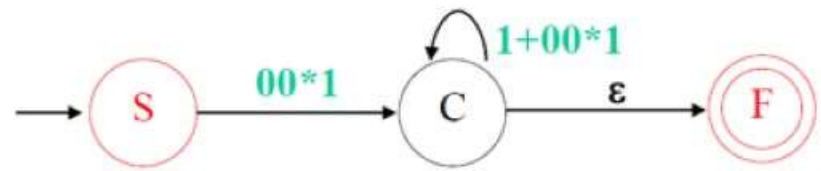
Eliminate State A

REs & RLs | Convert FA to RE | State Elimination | Example



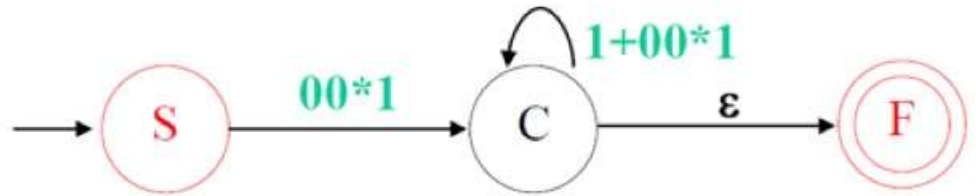
new $R_{SC} = R_{SC} + R_{SB} (R_{BB})^* R_{BC} = \phi + 0 (0)^* 1 = 00^*1$

new $R_{CC} = R_{CC} + R_{CB} (R_{BB})^* R_{BC} = 1 + 0 (0)^* 1 = 1+00^*1$



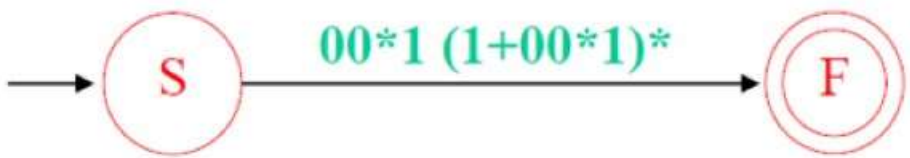
Eliminate State B

REs & RLs | Convert FA to RE | State Elimination | Example

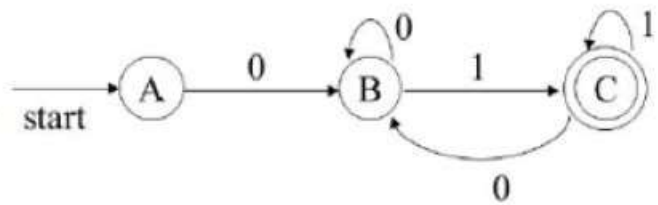


Eliminate State C

$$\text{new } R_{SF} = R_{SF} + R_{SC}\,(R_{CC})^*\,R_{CF} = \phi + 00^*1\,(1+00^*1)^*\,\varepsilon = 00^*1\,(1+00^*1)^*$$

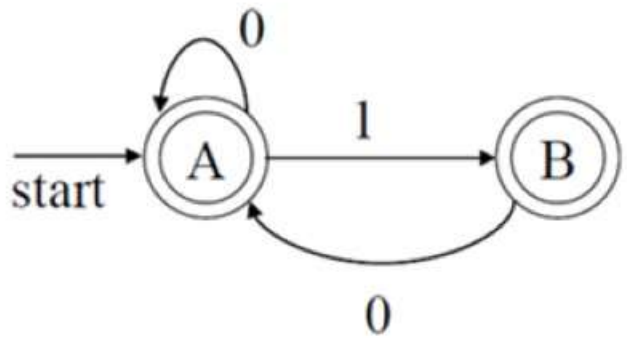


Thus, the regular expression is: **00*1 (1+00*1)***

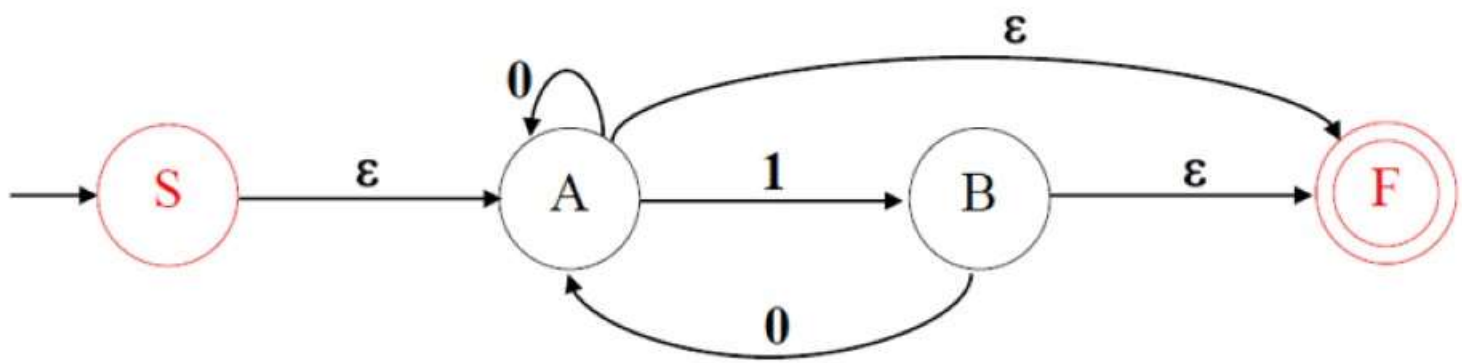


REs & RLs | Convert FA to RE | State Elimination | Example

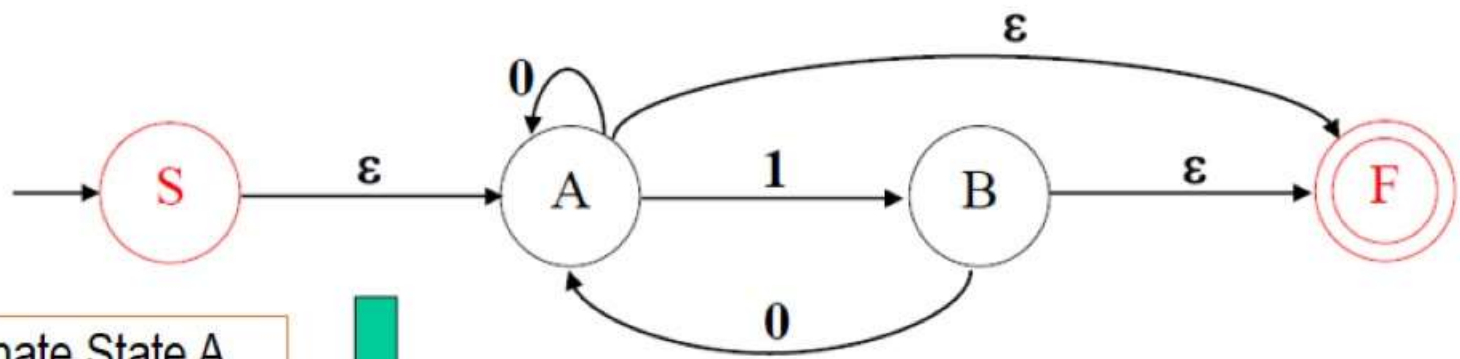
- **A DFA**



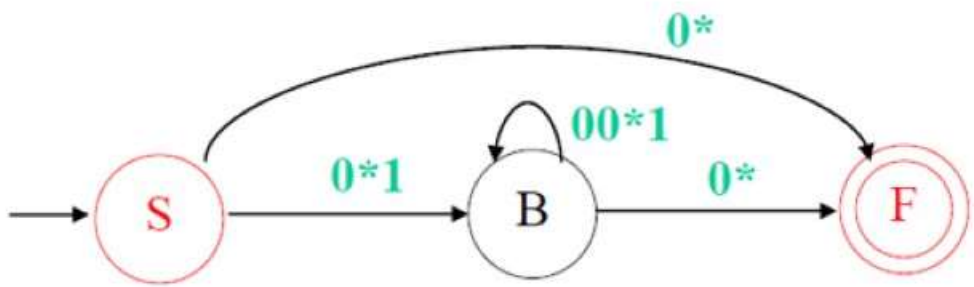
- **A GNFA in a special form:**



REs & RLs | Convert FA to RE | State Elimination | Example



Eliminate State A



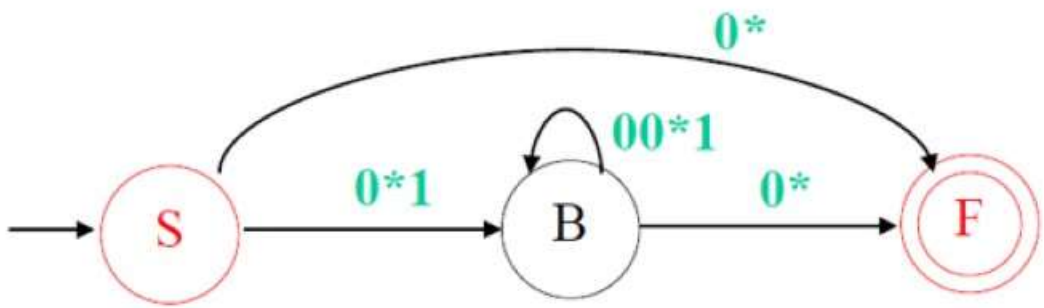
$R_{SF} = R_{SF} + R_{SA} (R_{AA})^* R_{AF} = \phi + \varepsilon (0)^* \varepsilon = 0^*$

$R_{SB} = R_{SB} + R_{SA} (R_{AA})^* R_{AB} = \phi + \varepsilon (0)^* 1 = 0^*1$

$R_{BB} = R_{BB} + R_{BA} (R_{AA})^* R_{AB} = \phi + 0 (0)^* 1 = 00^*1$

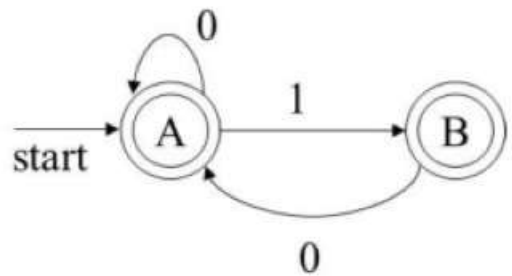
$R_{BF} = R_{BF} + R_{BA} (R_{AA})^* R_{AF} = \varepsilon + 0 (0)^* \varepsilon = \varepsilon + 00^* = 0^*$

REs & RLs | Convert FA to RE | State Elimination | Example

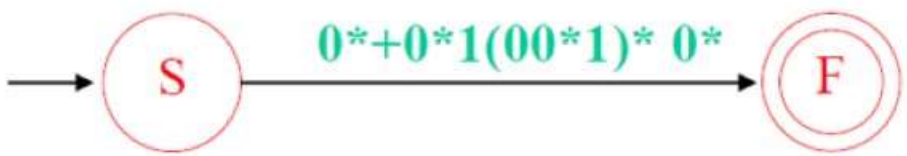


Eliminate State B

Thus, the regular expression is: 0*+0*1(00*1)*0*



$R_{SF} = R_{SF} + R_{SB} (R_{BB})^* R_{BF} = 0^* + 0^*1 (00^*1)^* 0^* = 0^*+0^*1(00^*1)^* 0^*$

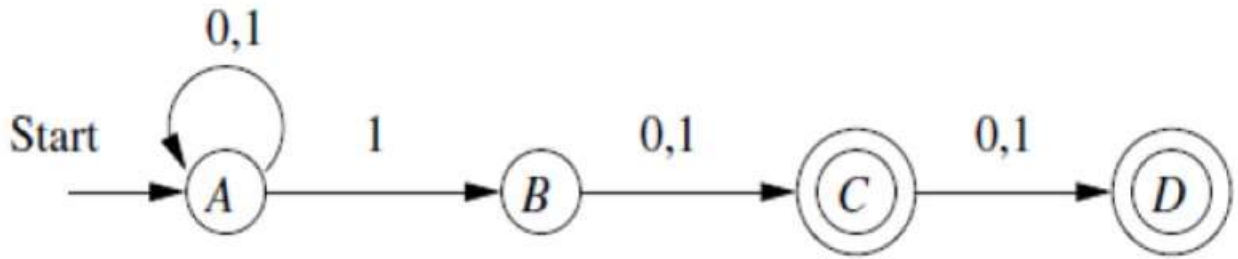


REs & RLs | Convert FA to RE | State Elimination | Example

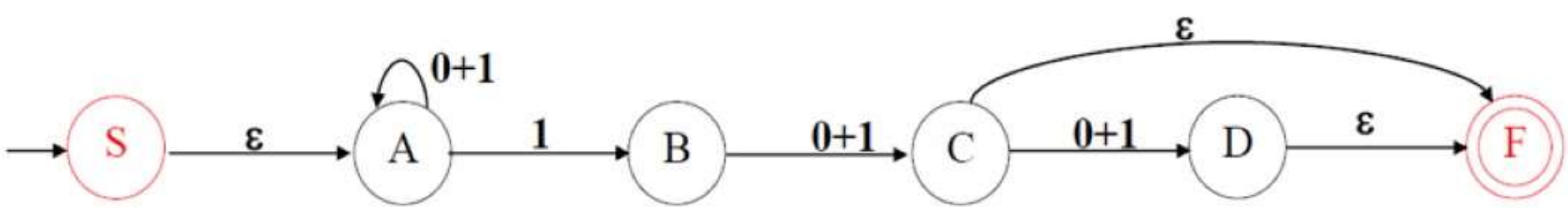
- We can use the conversion by state elimination algorithm for NFA too.
- First, we have to represent the given NFA as a GNFA.
  - If the label is a single symbol, the label of the generalized automaton will be that single symbol.
    - 0 ➔ 0 ε ➔ ε
  - If there are more than one symbol, the label will be union (OR) of those symbols.
    - 0,1 ➔ 0+1 0,1,ε ➔ **0+1+ε**

REs & RLs | Convert FA to RE | State Elimination | Example

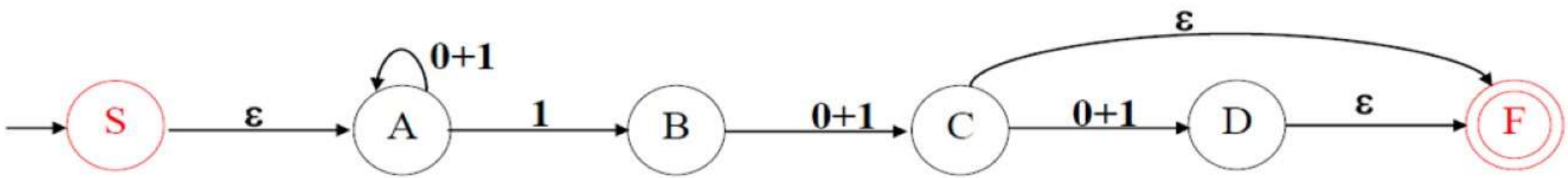
Convert a NFA to a regular expression



Convert a NFA to a GNFA in a special form.

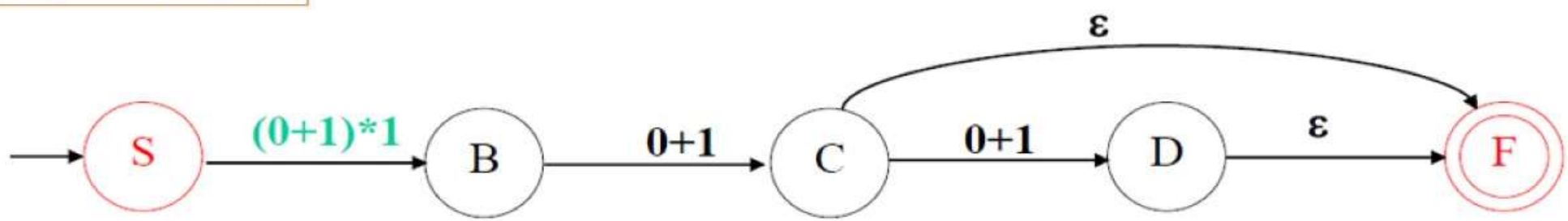


REs & RLs | Convert FA to RE | State Elimination | Example

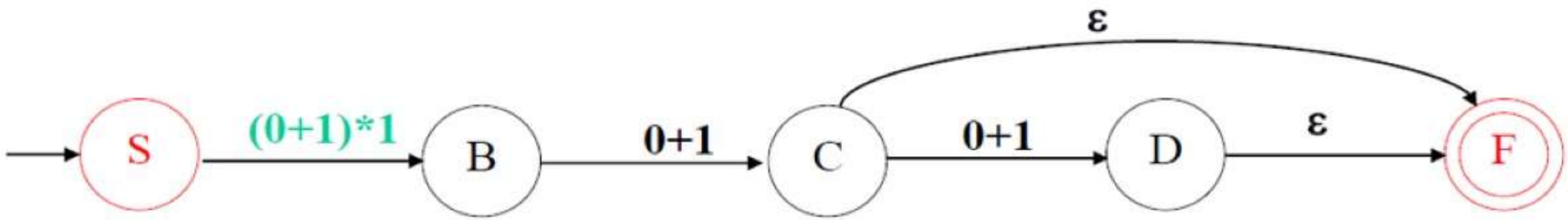


$$R_{SB} = R_{SB} + R_{SA} (R_{AA})^* R_{AB} = \phi + \varepsilon (0+1)^* 1 = (0+1)^*1$$

Eliminate State A

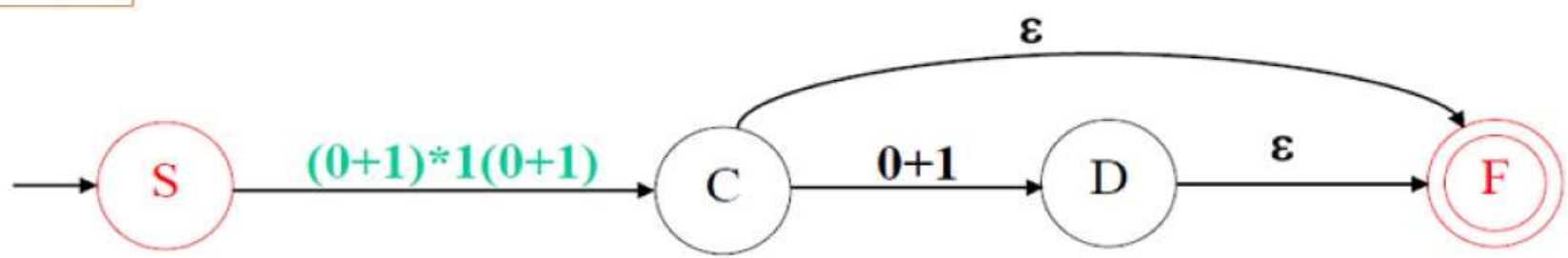


REs & RLs | Convert FA to RE | State Elimination | Example

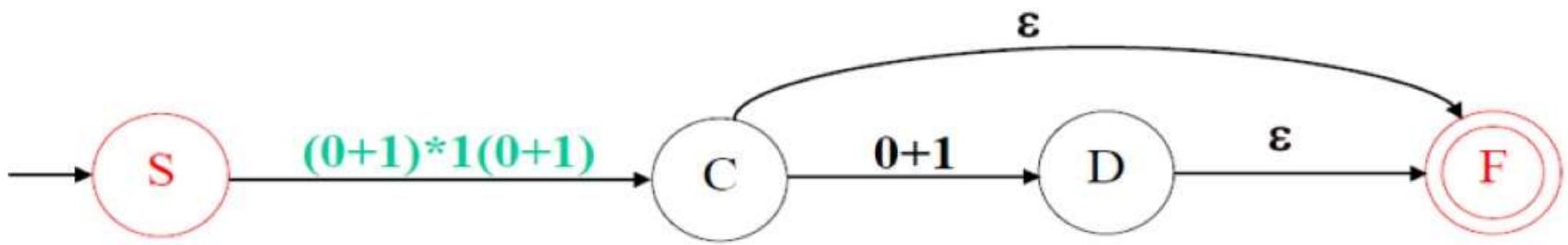


$$R_{SC} = R_{SC} + R_{SB} (R_{BB})^* R_{BC} = \phi + (0+1)^*1 (\phi)^* (0+1) = (0+1)^*1(0+1)$$

Eliminate State B



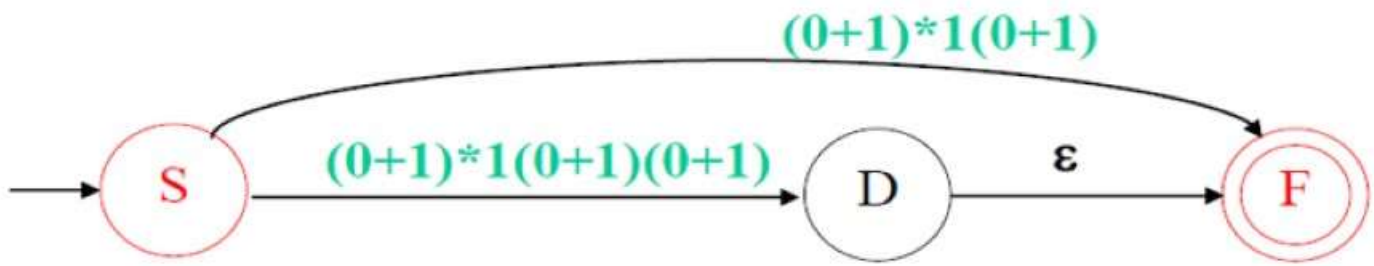
REs & RLs | Convert FA to RE | State Elimination | Example



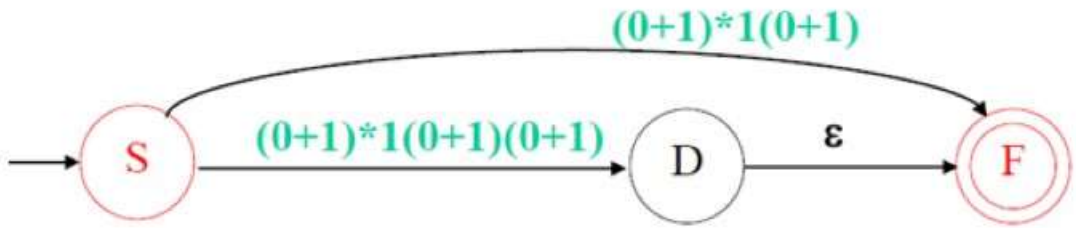
$R_{SD} = R_{SD} + R_{SC} (R_{CC})^* R_{CD} = \phi + (0+1)^*1(0+1) (\phi)^* (0+1) = (0+1)^*1(0+1)(0+1)$

$R_{SF} = R_{SF} + R_{SC} (R_{CC})^* R_{CF} = \phi + (0+1)^*1(0+1) (\phi)^* \varepsilon = (0+1)^*1(0+1)$

Eliminate State C



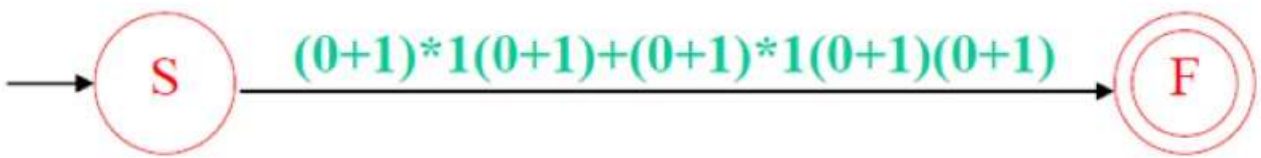
REs & RLs | Convert FA to RE | State Elimination | Example

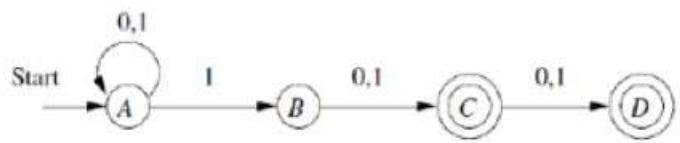


$$R_{SF} = R_{SF} + R_{SD}\,(R_{DD})^*\,R_{DF} = (0+1)^*1(0+1) + (0+1)^*1(0+1)(0+1)\,(\phi)^*\,\varepsilon$$

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

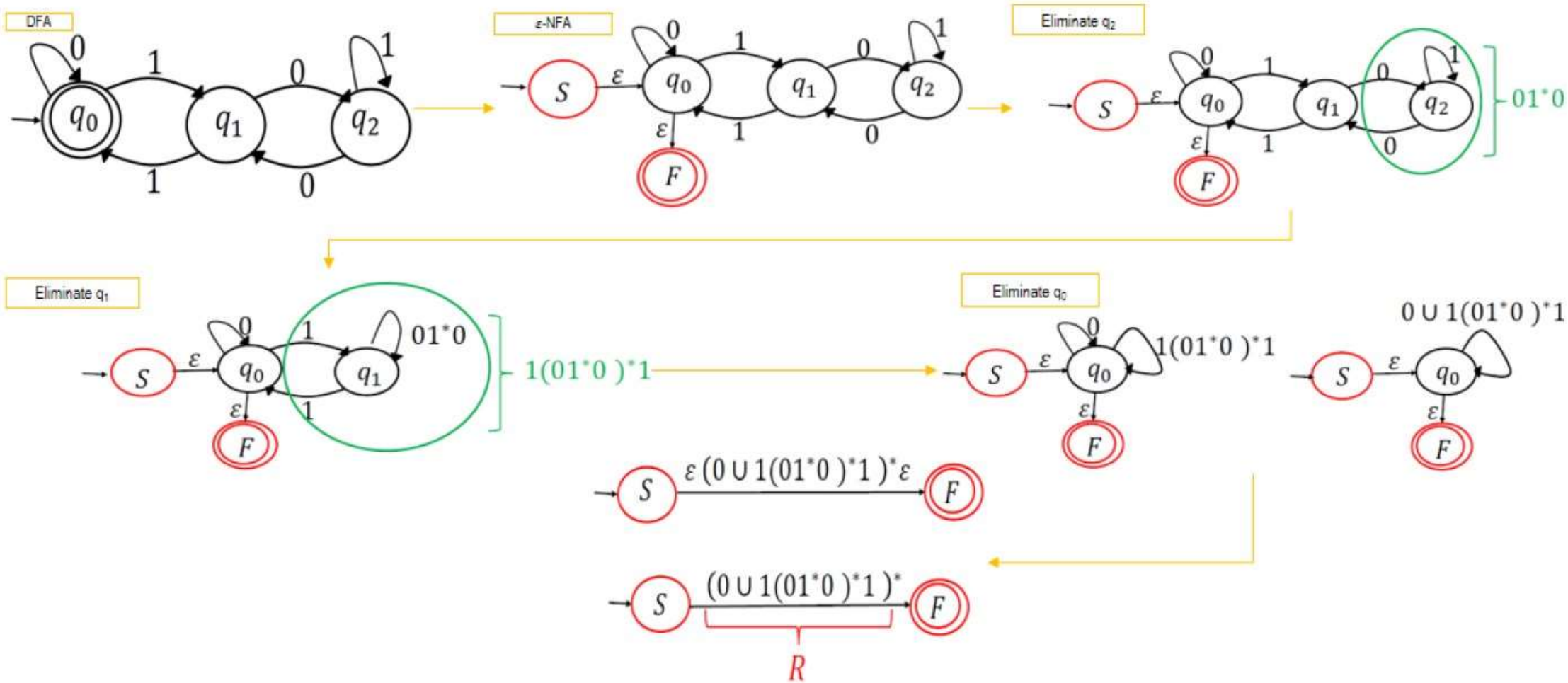
Eliminate State D





Thus, the regular expression is: $(0+1)^*1(0+1)+(0+1)^*1(0+1)(0+1)$

REs & RLs | Convert FA to RE | State Elimination | Example



REs & RLs | Convert FA to RE | Arden's Theorem

- Arden's Theorem is a fundamental result in formal language theory and automata theory, particularly in the context of regular languages and finite Automata.
- It provides a method for finding the solutions to certain types of linear equations over regular languages.
- The theorem is named after **Richard Arden**, who introduced it in the context of solving equations involving regular expressions.
- According to Arden's rule, the set **A*.B** is the smallest language that is a solution for **X** in the linear equation **X = A·X ∪ B**, where **X**, **A**, and **B** are sets of strings.
- Furthermore, this solution is unique if set A does not contain the empty word.

REs & RLs | Convert FA to RE | Arden's Theorem

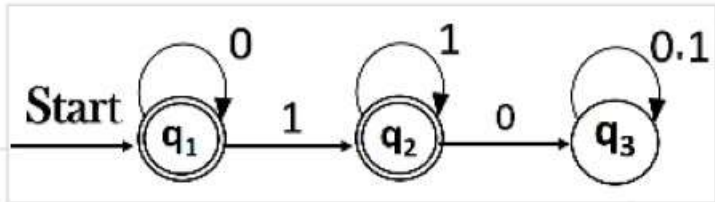
- **Proof:**
- $R = Q + (Q + RP)P$ [Putting the value $R = Q + RP$]
- $= Q + QP + RPP$
- We get the following equation if we put the value of R recursively, again and again:
- $R = Q + QP + QP^2 + QP^3 \ldots..$
- $R = Q (\varepsilon + P + P^2 + P^3 + \ldots.)$
- $R = QP^*$ [P^* represents $(\varepsilon + P + P^2 + P^3 + \ldots.)$]
- **Hence, proved.**

REs & RLs | Convert FA to RE | Arden's Theorem

- Assumptions for Applying Arden's Theorem
  - The transition diagram must not have NULL transitions
  - It must have only one initial state
- Method
  - **Step 1 –** Create equations as the following form for all the states of the DFA having n states with initial state q_1.
    - $q_1 = q_1R_{11} + q_2R_{21} + \ldots + q_nR_{n1} + \varepsilon$
    - $q_2 = q_2R_{12} + q_2R_{22} + \ldots + q_nR_{n2}$
    - .....
    - .....
    - .....
    - .....
    - $q_n = q_nR_{1n} + q_2R_{2n} + \ldots + q_nR_{nn}$
  - R_{ij} represents the set of labels of edges from q_i to q_j, if no such edge exists, then $R_{ij} = \emptyset$
  - **Step 2 –** Solve these equations to get the equation for the final state in terms of R_{ij}

REs & RLs | Convert FA to RE | Arden's Theorem | Example

- Construct the regular expression for the given DFA



1. Let us write down the equations

```
q1 = q1 0 + ε
```

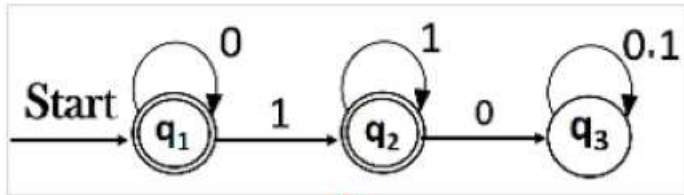
Since q1 is the start state, so ε will be added, and the input 0 is coming to q1 from q1 hence we write

State = source state of input × input coming to it

Similarly,

```
q2 = q1 1 + q2 1
q3 = q2 0 + q3 (0+1)
```

REs & RLs | Convert FA to RE | Arden's Theorem | Example



Since the final states are q1 and q2, we are interested in solving q1 and q2 only. Let us see q1 first

2

$$q1 = q1\ 0 + \varepsilon$$

We can re-write it as

$$q1 = \varepsilon + q1\ 0$$

Which is similar to R = Q + RP, and gets reduced to R = OP*.

Assuming R = q1, Q = ε, P = 0

We get

3

$$q1 = \varepsilon.(0)^*$$
$$q1 = 0^* \quad (\varepsilon.R^* = R^*)$$

Substituting the value into q2, we will get

4

$$q2 = 0^*\ 1 + q2\ 1$$
$$q2 = 0^*\ 1\ (1)^* \quad (R = Q + RP \rightarrow Q\ P^*)$$

The regular expression is given by

$$r = q1 + q2$$
$$= 0^* + 0^*\ 1.1^*$$
$$r = 0^* + 0^*\ 1^+ \quad (1.1^* = 1^+)$$

$$R^+ = RR^* = R^*R$$

REs & RLs | Convert FA to RE | Arden's Theorem | Example

Solution – 1

Here the initial state and final state is q_1.

The equations for the three states q1, q2, and q3 are as follows –

$q_1 = q_1a + q_3a + \varepsilon$ (ε move is because q1 is the initial state0

$q_2 = q_1b + q_2b + q_3b$

$q_3 = q_2a$

$q_2 = q_1b + q_2b + q_3b$ 2

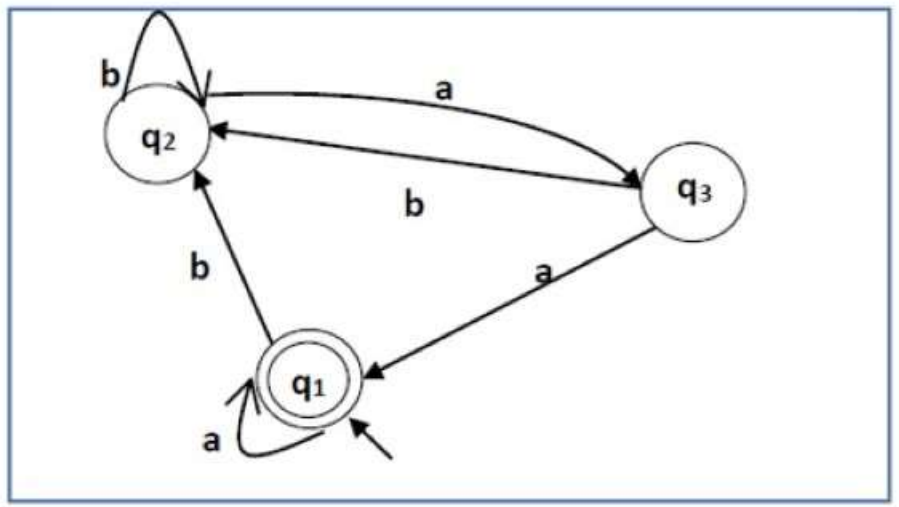
$= q_1b + q_2b + (q_2a)b$ (Substituting value of q_3)

$= q_1b + q_2(b + ab)$

$= q_1b\ (b + ab)^*$ (Applying Arden's Theorem)

$R = Q + RP = QP^*$
$Q = q_1b$
$R = q_2$
$P = (b + ab)$

- Construct a regular expression corresponding to the automata given below –



We will interested in solving q_1 only as it is the final state

REs & RLs | Convert FA to RE | Arden's Theorem | Example

Solution – 1

Here the initial state and final state is $\mathbf{q_1}$.

The equations for the three states q1, q2, and q3 are as follows –

$q_1 = q_1a + q_3a + \varepsilon$ (ε move is because q1 is the initial state0

$q_2 = q_1b + q_2b + q_3b$

$q_3 = q_2a$

$q_2 = q_1b + q_2b + q_3b$ 2

$= q_1b + q_2b + (q_2a)b$ (Substituting value of q_3)

$= q_1b + q_2(b + ab)$

$= q_1b\ (b + ab)^*$ (Applying Arden's Theorem)

R = Q + RP = QP*
Q = q1b
R = q2
P = (b + ab)

3

$q_1 = q_1a + q_3a + \varepsilon$

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

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

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

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

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

Rewrite statement as
q1 = ε + q1(a + b(b + ab)*aa)
R = Q + RP = QP*
R = q1
Q = ε
P = (a + b(b + ab)*aa)
QP* = ε(a + b(b + ab)*aa)*

ε.R* = R*

REs & RLs | Convert FA to RE | Arden's Theorem | Example

Solution – 1

Here the initial state is q_1 and the final state is q_2

Now we write down the equations –

$q_1 = q_1 0 + \varepsilon$

$q_2 = q_1 1 + q_2 0$

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

Now, we will solve these three equations –

2 $q_1 = \varepsilon 0^*$ [As, $\varepsilon R = R$]

So, $q_1 = 0^*$

$q_2 = 0^*1 + q_2 0$

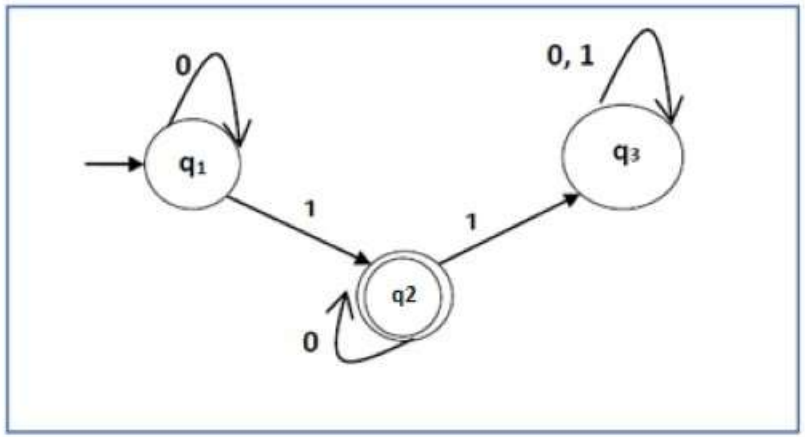
So, $q_2 = 0^*1(0)^*$ [By Arden's theorem]

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

$q_1 = \varepsilon + q_1 0$
$R = Q + RP = QP^*$
$R = q_1$
$Q = \varepsilon$
$P = 0$
$R = \varepsilon 0^* = 0^*$

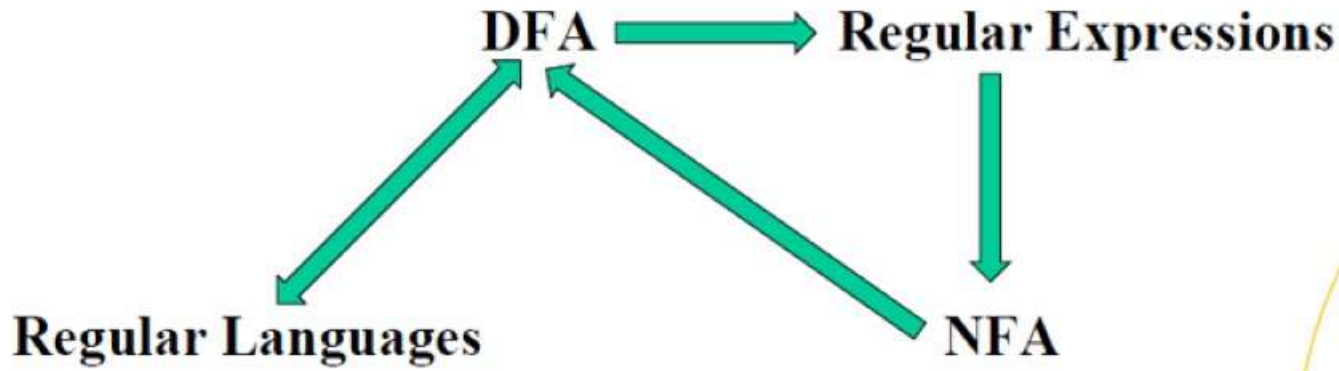
$q_2 = 0^*1 + q_2 0$
$R = q_2$
$Q = 0^*1$
$P = 0$
$R = Q + RP = QP^*$
$= 0^*1(0)^*$

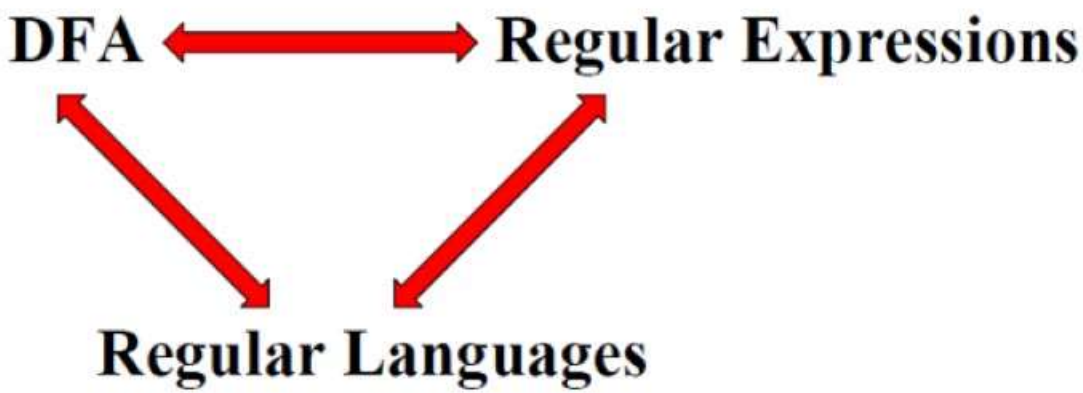
- Construct a regular expression corresponding to the automata given below –



- We will add ε to q_1 since it is initial state
- We will try to solve the state q_2 since it is final state

REs & RLs | Summary





- Both concepts are basically the same. For every regular language there is a regular expression, and for every regular expression there is a regular language.
- If R is a regular expression, L(R) is also a regular language belonging to this expression. In order for a language to be regular, it must be accepted by any DFA.
- From the equivalence of NFA and DFAs, we can say the same for NFA.

REs & RLs | Algebraic Laws

- Two regular expressions were equivalent **iff** they define the same language
- **Algebraic laws** that bring to a higher level the issue of when two regular expressions are equivalent.
- Instead of examining specific regular expressions, we consider pairs of regular expressions with variables as arguments.
- Two regular expressions with variables are equivalent if whatever languages we substitute for the variables, the results of the two expressions are the same language.

REs & RLs | Algebraic Laws

- **Commutativity** is the property of an operator that says we can switch the order of its operands and get the same result.
- **Associativity** is the property of an operator that allows us to regroup the operands when the operator is applied twice.

Commutative Law for Union: $M \cup N = N \cup M$

- we may take the union of two languages in either order.

Associative Law for Union: $(M \cup N) \cup R = M \cup (N \cup R)$

- we may take the union of three languages either by taking the union of the first two initially or taking the union of the last two initially.

Associative Law for Concatenation: $(M\,N)\,R = M\,(N\,R)$

- we can concatenate three languages by concatenating either first two or last two initially.

Concatenation is NOT commutative: $MN \neq NM$

REs & RLs | Algebraic Laws

- An **identity** for an operator is a value such that when the operator is applied to the identity and some other value, the result is the other value
- 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.
  - **Φ is identity for union:** $\Phi \cup N = N \cup \Phi = N$
  - **$\{\varepsilon\}$ is left and right identity for concatenation:** $\{\varepsilon\}\, N = N\, \{\varepsilon\} = N$
  - **Φ is left and right annihilator for concatenation:** $\Phi\, N = N\, \Phi = \Phi$

In other words, **A = B** is an identity if **A** and **B** define the same functions, and an identity is an equality between functions that are differently defined.
For example, $(a + b)^2 = a^2 + 2ab + b^2$ is an identity

REs & RLs | Algebraic Laws

- A **distributive law** involves two operators, and asserts that one operator can be pushed down to be applied to each argument of the other operator individually.
- **Concatenation is left and right distributive over union:**

$$R(M \cup N) = RM \cup RN$$

$$(M \cup N)R = MR \cup NR$$

However, RM U RN ≠ MR U NR
Concatenation is not commutative.

- An operator is said to be **idempotent** if the result of applying it to two of the same values as arguments is that value.
- **Union is idempotent:** $M \cup M = M$

REs & RLs | Algebraic Laws | Closure Laws

Languages	Regular Expressions
$\Phi^* = \{\varepsilon\}$	$\Phi^* = \varepsilon$
$\{\varepsilon\}^* = \{\varepsilon\}$	$\varepsilon^* = \varepsilon$
$L^+ = LL^* = L^*L$	$R^+ = RR^* = R^*R$
$L^* = L^+ \cup \{\varepsilon\}$	$R^* = R^+ + \varepsilon$
$L? = L \cup \{\varepsilon\}$	$R? = R + \varepsilon$
$(L^*)^* = L^*$	$(R^*)^* = R^*$

Given R, P, L, Q as regular expressions, the following identities hold −

- $\emptyset^* = \varepsilon$
- $\varepsilon^* = \varepsilon$
- $RR^* = R^*R$
- $R^*R^* = R^*$
- $(R^*)^* = R^*$
- $RR^* = R^*R$
- $(PQ)^*P = P(QP)^*$
- $(a+b)^* = (a^*b^*)^* = (a^*+b^*)^* = (a+b^*)^* = a^*(ba^*)^*$
- $R + \emptyset = \emptyset + R = R$ (The identity for union)
- $R\,\varepsilon = \varepsilon\,R = R$ (The identity for concatenation)
- $\emptyset\,L = L\,\emptyset = \emptyset$ (The annihilator for concatenation)
- $R + R = R$ (Idempotent law)
- $L\,(M + N) = LM + LN$ (Left distributive law)
- $(M + N)\,L = ML + NL$ (Right distributive law)
- $\varepsilon + RR^* = \varepsilon + R^*R = R^*$

REs & RLs | Checking A Law

- There is an infinite variety of algebraic laws about regular expressions that might be proposed.
- **Methodology**: Exp1 = Exp2
  - Replace each ***variable*** in the law (in Exp1 and Exp2) with ***unique symbols*** to create concrete regular expressions, RE1 and RE2.
  - Check ***the equality of the languages of RE1 and RE2***, ie. **L(RE1) = L(RE2)**
  - **Two regular languages are equal if their DFAs are equal.**

REs & RLs | Checking A Law | Example

Law: $\mathbf{R(M+N) = RM + RN}$

Replace R with a, M with b, and N with c.

➔ $\mathbf{a(b+c) = ab + ac}$

Then, check whether **L(a(b+c))** is equal to **L(ab+bc)**

If their languages are equal, the law is TRUE.

Since, **L(a(b+c))** is equal to **L(ab+bc)**

➔ $\mathbf{R(M+N) = RM + RN}$ is a **true algebraic law**

REs & RLs | Checking A Law | Example

Law: $(M+N)^* = (M^*N^*)^*$

Replace M with a, and N with b.

➔ $(a+b)^* = (a^*b^*)^*$

Then, check whether $L((a+b)^*)$ is equal to $L((a^*b^*)^*)$

Since, $L((a+b)^*)$ is equal to $L((a^*b^*)^*)$

➔ $(M+N)^* = (M^*N^*)^*$ is **a true law**

Lets call it a day!

- Thanks for listening!
- Next week
  - Regular Language Properties!
    - Non-regular languages & Pumping Lemma!

Appendix | REs & RLs | Example

- In the following examples, we assume that the alphabet is Σ=0,1

1. $0^*10^* = \{w \mid w \text{ contains a single } 1\}$.

2. $\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$.

3. $\Sigma^*001\Sigma^* = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$.

4. $1^*(01^+)^* = \{w \mid \text{every } 0 \text{ in } w \text{ is followed by at least one } 1\}$.

5. $(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$.[5]

6. $(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of } 3\}$.

7. $01 \cup 10 = \{01, 10\}$.

8. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 = \{w \mid w \text{ starts and ends with the same symbol}\}$.

9. $(0 \cup \varepsilon)1^* = 01^* \cup 1^*$.
The expression $0 \cup \varepsilon$ describes the language $\{0, \varepsilon\}$, so the concatenation operation adds either 0 or ε before every string in 1^*.

10. $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$.

11. $1^*\emptyset = \emptyset$.
Concatenating the empty set to any set yields the empty set.

12. $\emptyset^* = \{\varepsilon\}$.
The star operation puts together any number of strings from the language to get a string in the result. If the language is empty, the star operation can put together 0 strings, giving only the empty string.

Appendix | REs & RLs | Example

- Write the RE for the language accepting all combinations of a's, over the set $\sum$ = {a}
- Solution:
  - All combinations of a's means a may be zero, single, double and so on.
  - If a is appearing zero times, that means a null string.
  - That is we expect the set of {ε, a, aa, aaa, ...}.
  - So we give a regular expression for this as:
  - R = a*
  - That is Kleene closure of a.

- Write the RE for the language accepting all combinations of a's except the null string, over the set $\sum$ = {a}
- Solution:
  - The regular expression has to be built for the language
  - L = {a, aa, aaa, ...}
  - This set indicates that there is no null string.
  - So we can denote regular expression as:
  - R = a+

Appendix | REs & RLs | Example

- Write the RE for the language accepting all the string which are starting with 1 and ending with 0, over $\sum = \{0, 1\}$.
- Solution:
  - In a regular expression, the first symbol should be 1, and the last symbol should be 0.
  - The RE is as follows:
  - R = 1 (0+1)* 0

- Write the RE for the language starting and ending with a and having any having any combination of b's in between, over the set $\sum = \{a, b\}$
- Solution:
  - The RE will be:
  - R = a b* a

Appendix | REs & RLs | Example

- Write the RE for the language starting with a but not having consecutive b's, over the set $\sum$ = {a, b}
- Solution:
  - The RE has to be built for the language: L = {a, aba, aab, aba, aaa, abab, ...}
  - The RE for the above language is:
  - R = {a + ab}*

- Write the RE for the language accepting all the string in which any number of a's is followed by any number of b's is followed by any number of c's, over the set $\sum$ = {a, b, c}
- Solution:
  - As we know, any number of a's means a* any number of b's means b*, any number of c's means c*.
  - Since as given in problem statement, b's appear after a's and c's appear after b's.
  - So the regular expression could be:
  - R = a* b* c*

Appendix | REs & RLs | Example

- Describe the language denoted by following regular expression

$$RE = (b^*(aaa)^*b^*)^*$$

- Solution:
  - The language can be predicted from the regular expression by finding the meaning of it. We will first split the regular expression as:
  - RE = (any combination of b's) (aaa)* (any combination of b's)
  - L = {The language consists of the string in which a's appear triples, there is no restriction on the number of b's}

Appendix | REs & RLs | Example

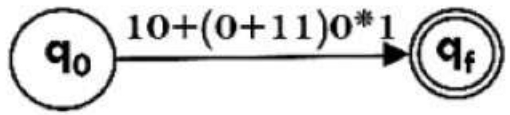
- Write the RE for the language containing the string over {0, 1} in which there are at least two occurrences of 1's between any two occurrences of 1's between any two occurrences of 0's.
- Solution:
  - At least two 1's between two occurrences of 0's can be denoted by (0111*0)*.
  - Similarly, if there is no occurrence of 0's, then any number of 1's are also allowed.
  - Hence the RE for required language is:
  - R = (1 + (0111*0))*

- Write the RE for the language containing the string in which every 0 is immediately followed by 11, over an alphabet of {0, 1}.
- Solution:
  - The RE will be:
  - R = (011 + 1)*

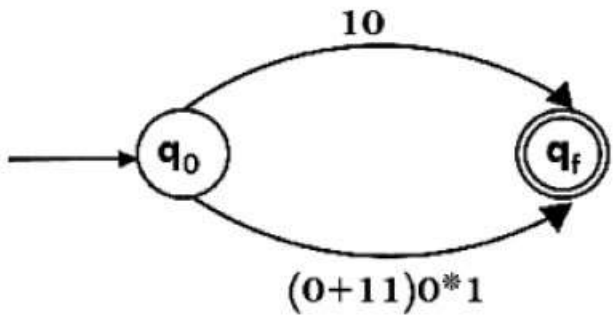
Appendix | REs & RLs | RE to NFA | Example

- Design a FA from given regular expression 10 + (0 + 11)0* 1. (without epsilon moves)

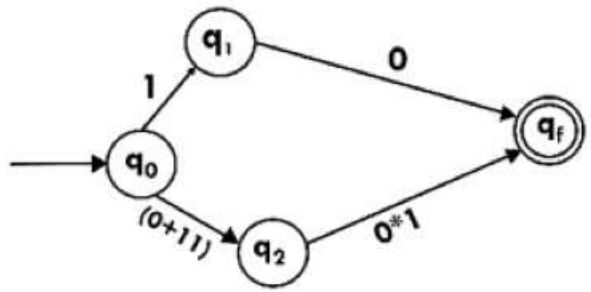
Step 1:



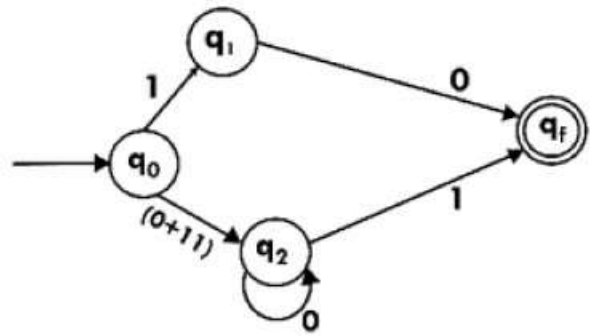
Step 2:



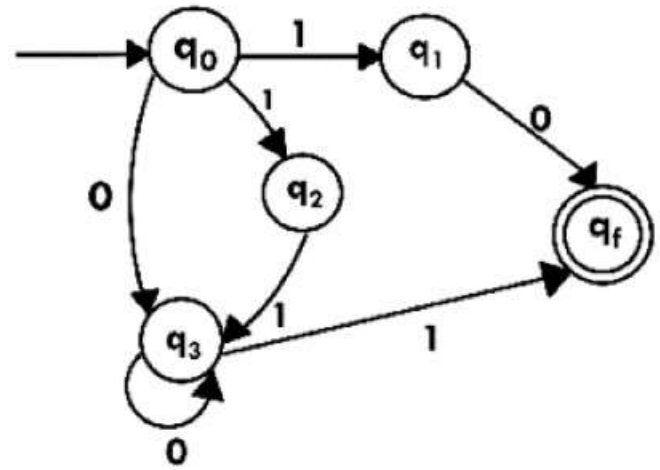
Step 3:



Step 4:



Step 5:



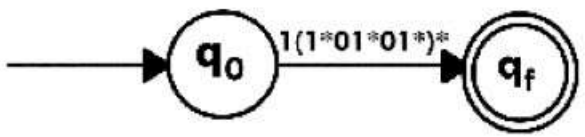
- If we had created NFA with epsilon moves we had to eliminate them at first (but not required for this example)
- After obtaining epsilon-free NFA, we have to convert it to DFA using the way we have seen in the previous weeks

Appendix | REs & RLs | RE to NFA

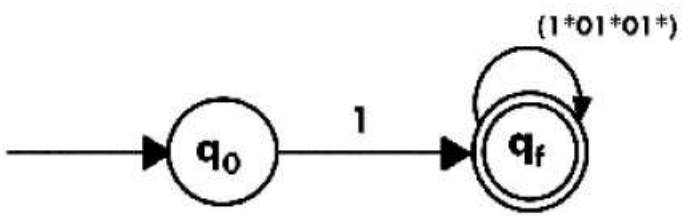
- Design a NFA from given regular expression 1 (1* 01* 01*)*. (without epsilon moves)

Solution: The NFA for the given regular expression is as follows:

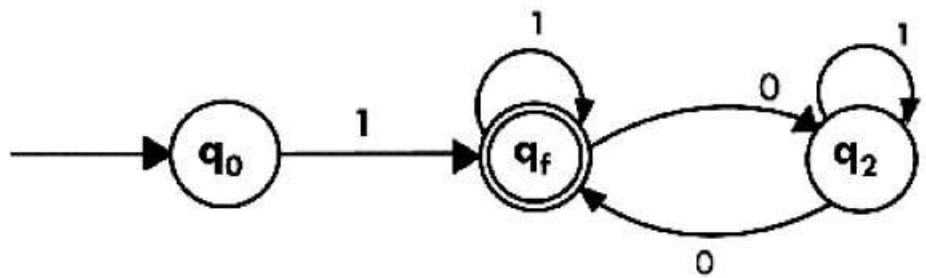
Step 1:



Step 2:



Step 3:



After obtaining epsilon-free NFA, we have to convert it to DFA using the way we have seen in the previous weeks

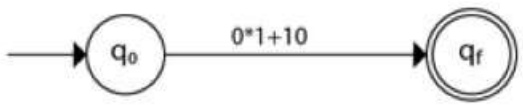
Appendix | REs & RLs | RE to NFA

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

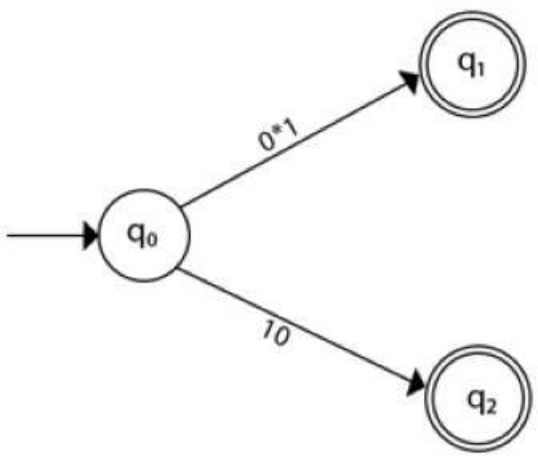
Solution:

We will first construct FA for R = 0*1 + 10 as follows:

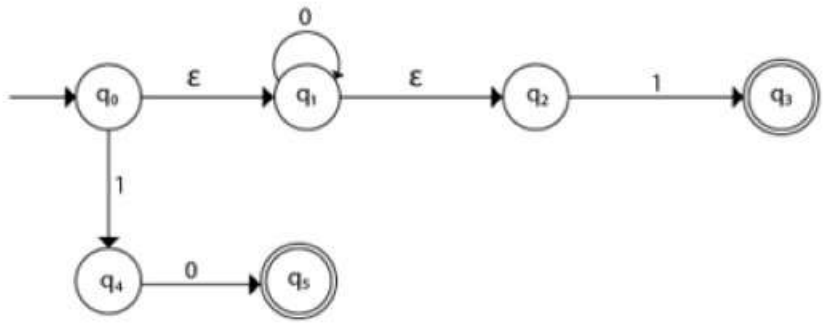
Step 1:



Step 2:



Step 3:

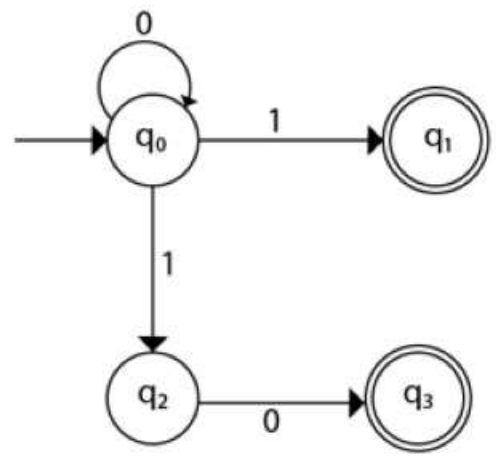


NFA with epsilon moves

After obtaining epsilon-free NFA, we have to convert it to DFA using the way we have seen in the previous weeks

NFA without epsilon moves

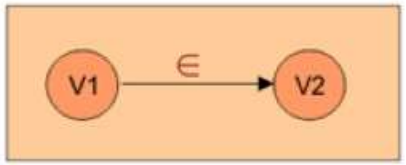
Step 4:



Appendix | REs & RLs | Removal of Epsilon Moves from FA

If the epsilon exist between any two states in automata then the removal of epsilon through some rules is called **elimination of epsilon** move form NFA.

Suppose there are two vertices V1 and V2 and epsilon between them is given below.



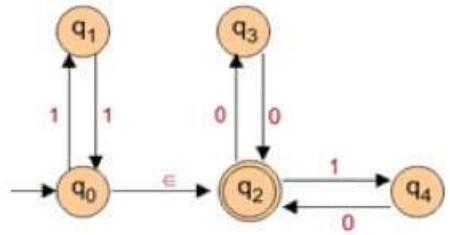
Steps For Elimination Of Epsilon From NFA

- **Step 01:** Find all edges starting from V2
- **Step 02: Remove the epsilon. And** All Finding edges from V2 are Duplicated to V1 without changing edge labels.
- **Step 03:** if V1 is initial state, make V2 initial
- **Step 04:** if V2 is final state, make V1 final
- **Step 05:** Remove all dead states

Note: if more than one epsilon are exist in epsilon-NFA, then first remove the epsilon which is far away from initial state. After Removal of one epsilon remove the others till the all removal of epsilon.

Example of Elimination Of Epsilon From NFA

Consider the following epsilon NFA



Solution

In the above epsilon NFA, suppose q0 as a V1 and q2 as V2 because epsilon exist between these two states. Now apply the steps of epsilon removal.

Step 01: Find all edges starting from V2

- The first input "0" is starting from V2 (q2) and going to q3.
- The second input "1" is starting from V2 (q2) and going to q3

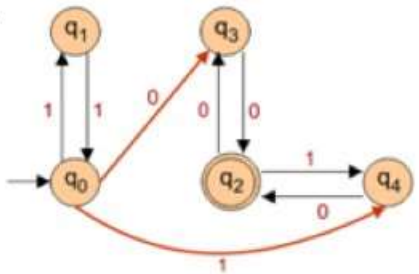
Appendix | REs & RLs | Removal of Epsilon Moves from FA

Step 02: Remove the epsilon. And All Finding edges from V2 are Duplicated to V1 without changing edge labels.

Finding outgoing edges from V2 are "0" and "1".

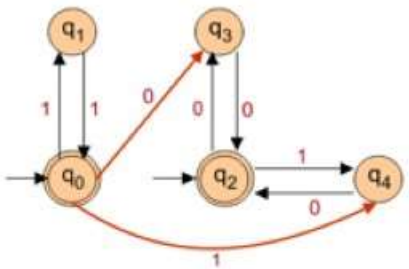
- First Input "0" is going to q3 from V2 (q2). So first duplicate will goes from V1 (q0) to q3.
- Second Input "1" is going to q4 from V2 (q2). So second duplicate will goes from V1 (q0) to q4.

So, duplicated edge with their labels are



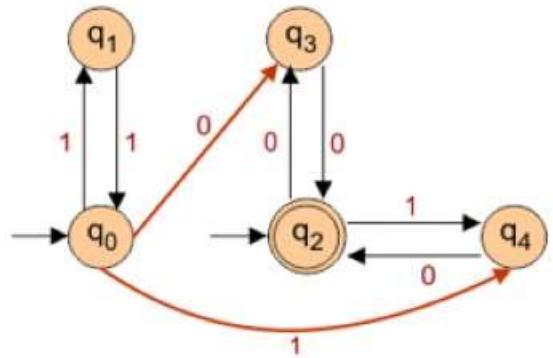
Step 04: if V2 is final state, make V1 final State

As V2 (q2) is a final state. So V1(q0) change to final state. As given below



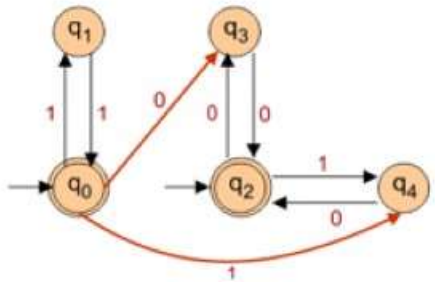
Step 03: If V1 is initial state, make V2 also initial

As V1 (q0) is a initial. So V2(q2) change to initial. As given below



Step 05: Remove all dead states

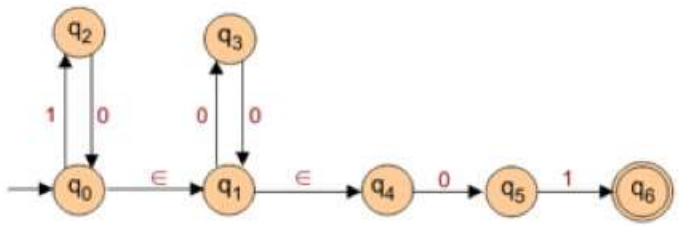
A state which is unreachable is called dead state. In above NFA, there is no dead state. So, final NFA after removal of epsilon is given below



Appendix | REs & RLs | Removal of Epsilon Moves from FA

Example

Consider the following epsilon NFA



Step 01: Find all edges starting from V2

- The only input "0" is starting from V2 (q4) and going to q5.

Step 03: if V1 is initial state, make V2 initial State

As V1 (q1) is not a initial state. So V2(q2) will be unchanged and NFA of Step 2 remain Same.

Step 04: if V2 is final state, make V1 final State

As V2 (q4) is not a final state. So, V1(q1) will unchanged and NFA of Step 2 remain Same.

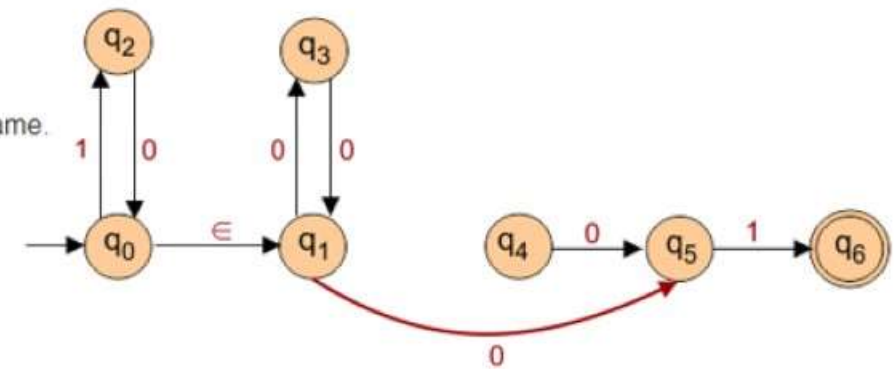
Solution

In the above epsilon NFA, there are two epsilon exist. First we remove the epsilon which is for away from intial state. So, first we remove the epsilon which exist in q1 and q4.

suppose q1 is V1 and q4 is V2 because epsilon exist between these two states. Now apply the steps of epsilon removal.

Step 02: Remove the epsilon. All Finding edges from V2 are Duplicated to V1 without changing edge labels.

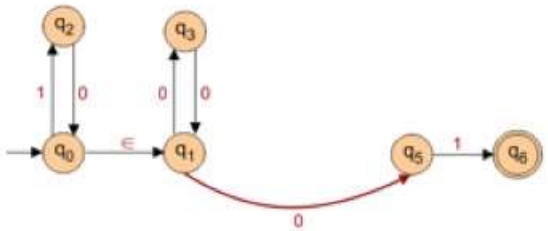
As the Input "0" is going to q5 from V2 (q2). So duplicate will goes from V1 (q1) to q5. So, duplicated edge with their lables are given below



Appendix | REs & RLs | Removal of Epsilon Moves from FA

Step 05: Remove all dead states

A state which is unreachable is called dead state. In above NFA, q4 is dead state. So, NFA after removal of first epsilon and dead state is given below



Now Repeat the Above 5 steps again for elimination of 2nd epsilon.

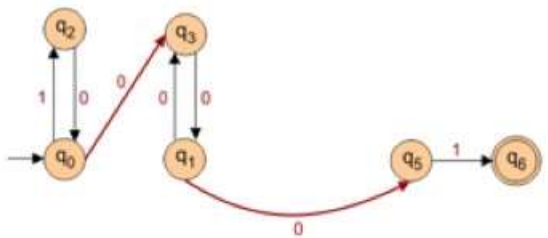
After first 5 steps epsilon- NFA , epsilon exist between q0 and q1. Again suppose q0 is V1 and q2 is V2.

Step 01: Find all edges starting from V2

- The only input "0" is starting from V2 (q2) and going to q3.

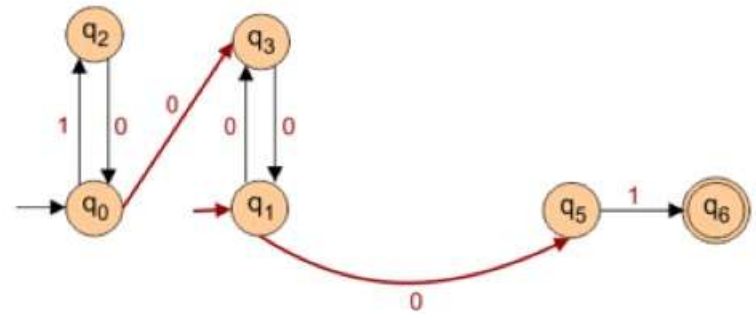
Step 02: Remove the epsilon. All Finding edges from V2 are Duplicated to V1 without changing edge labels.

As the Input "0" is going to q3 from V2 (q1). So duplicate will goes from V1 (q0) to q3. So, duplicated edge with their labels are given below



Step 03: if V1 is initial state, make V2 initial State

As V1 (q0) is a initial state. So V2(q1) will be change to final . As given below



Step 04: if V2 is final state, make V1 final State

As V2 (q1) is not a final state. So, V1(q1) will unchanged and NFA of above step3 will remain the same.

Step 05: Remove all dead states

No dead state found. The above NFA of step 3 will remain the same. So, the final NFA after elimination of both epsilons and dead state is given below