Theory of Computation (CT-502)

Course Instructor
ANUJ GHIMIRE

DISCLAIMER

- *This document does not claim any originality and cannot be used as a substitute for prescribed textbooks.*
- *The information presented here is merely a collection from various sources as well as freely available material from internet and textbooks.*
- *The ownership of the information lies with the respective authors or institutions.*

Regular Expression and Finite Automata

- A language denoted by ***L*** over an alphabet ∑ is subset of all the strings that can be formed out of ∑.
- Simply, language is subset of kleen closure over an alphabet ∑ :L ⊆ ∑* (set of strings chosen from ∑* defines language).
- Regular Expression is a formula for representing a language in terms of a form combined using 3 operations union, concatenation and kleen closure.

Regular Expression and Finite Automata

- A regular expression is a rule that describe a whole set of strings belongs to certain language.
- Any regular expression is recursively defined as:
  - Empty state (ϕ), empty string(ε) symbol of input alphabet ($\sum$) are regular expression
  - Let R1 and R_2 be regular expression then union of R1 and R2 denoted by (R1 + R2) is also Regular Expression.
  - Let R1 and R_2 be regular expression then concatenation of R1 and R2 denoted by (R1 . R2) is also Regular Expression.
  - Let R be regular expression then kleen closure of R denoted by R* is also Regular Expression.

Regular Expression and Finite Automata

- Regular expression are the language generator and finite automata are the language acceptor.
- The regular expression approach for describing regular language is fundamentally different from the finite automata approach.
- However, these two notations represent exactly the same set of languages, called regular languages.
- A language is said to be regular language if and only if a finite automata recognizes it.

Conversion of Regular Expression To Finite Automata

- We can convert the finite automata to its equivalent regular expression and vice versa.
- We have already discussed that DFA and two kinds of NFA with & without ε-transition which accept the same class of languages.
- In order to show that the regular expressions define the same class, we must show that:
  - Every language defined by one of these automata is also defined by a regular expression. For this proof, we can assume the language is accepted by some DFA.

Conversion of Regular Expression To Finite Automata

- Every language defined by a regular expression is defined by one of these automata. For this part of proof, the easiest is to show that there is an NFA with ε-transitions accepting the same language.

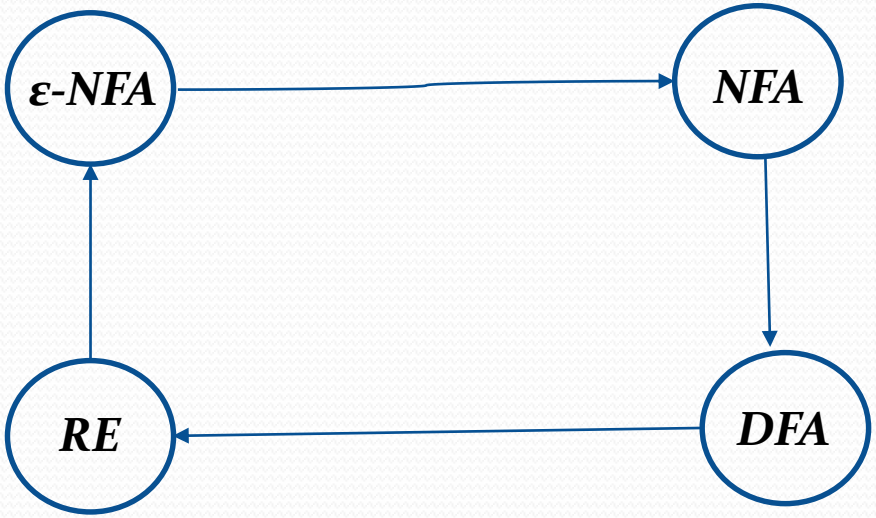
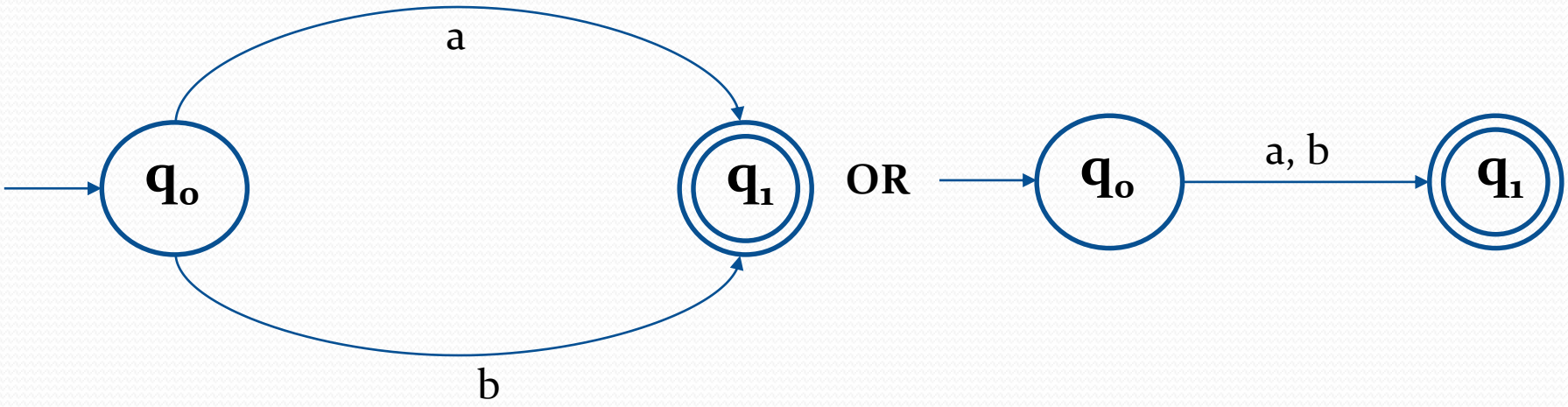


Fig: Plan for showing the equivalence of four different notations for regular language.

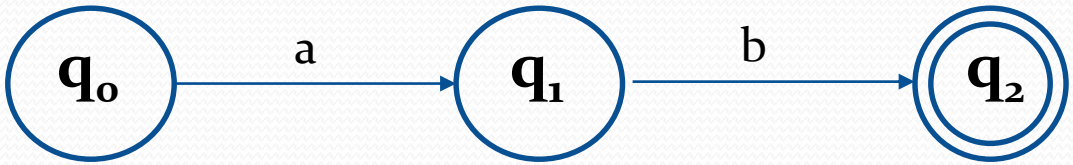
Conversion of Regular Expression To Finite Automata

- For the conversion of regular expression to finite automata we have some important basic rules as follows:
  - When we have the expression like ***(a+b)*** or ***(a ∪ b)*** or ***(a|b)***

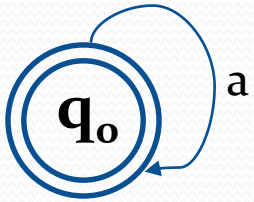


Conversion of Regular Expression To Finite Automata

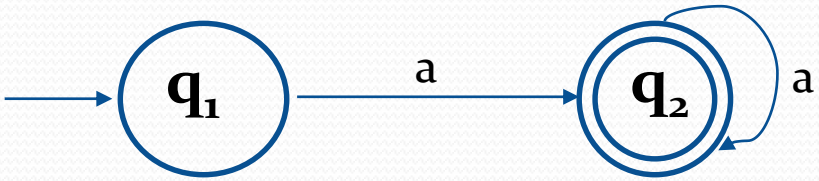
- When we have the expression of the form ***a.b*** then



- If we have the expression of the form a* then:

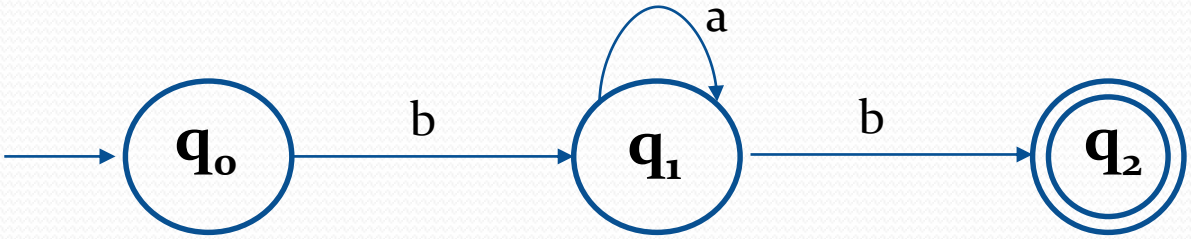


- If we have the expression of the form a^+ then:

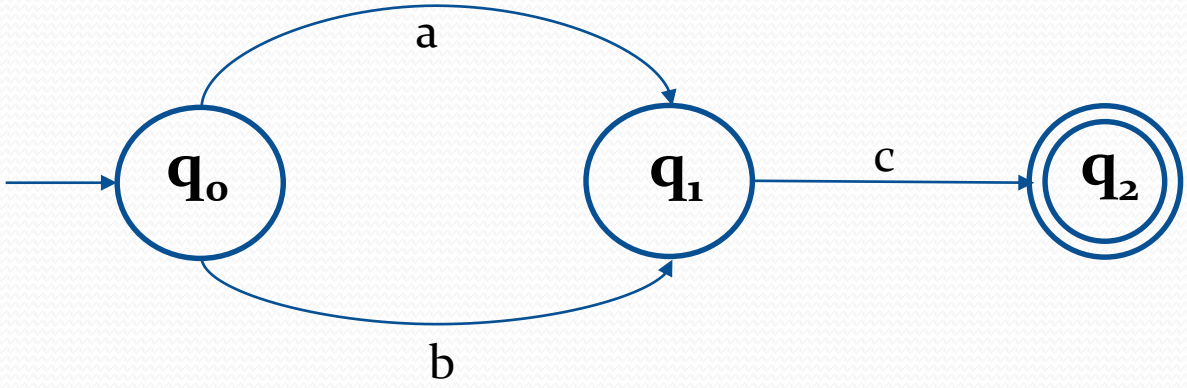


Conversion of Regular Expression To Finite Automata_Example

- Convert the given RE into FA:
  - ba*b



  - (a+b)c

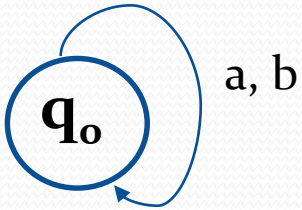


Conversion of Regular Expression To Finite Automata_Example_1

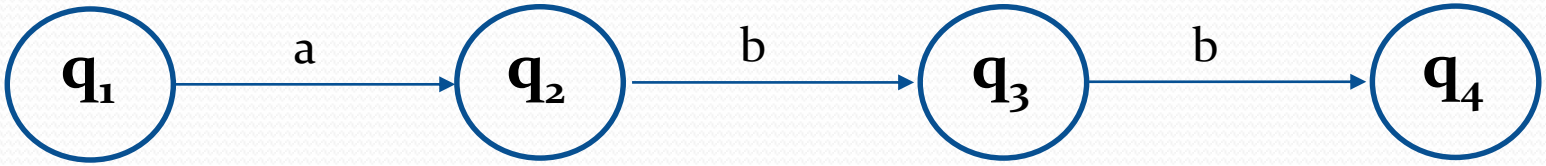
- Convert the given RE into FA:
  - $(a|b)^*(abb|a^+b)$

To solve this problem we first break down the given RE, construct the respective FA and then combine them.

for $(a|b)^*$

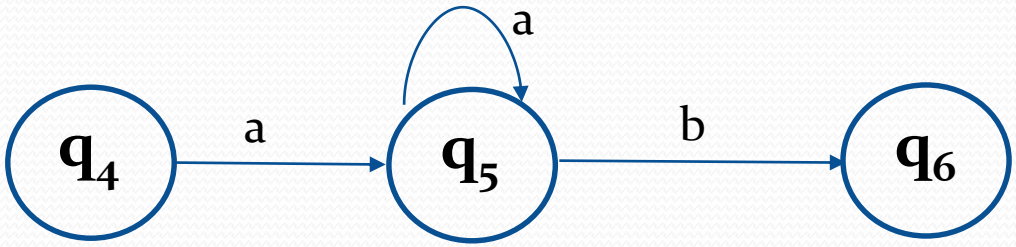


for ***abb***

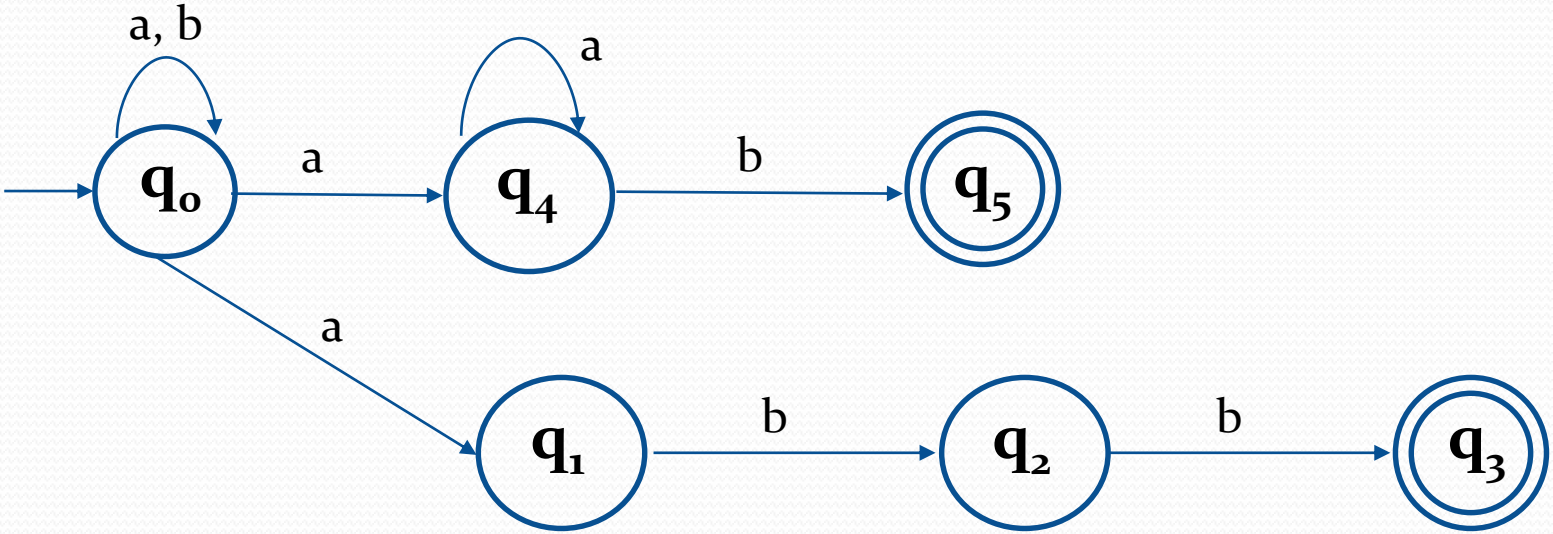


Conversion of Regular Expression To Finite Automata_Example_1

for $a^+ b$



Now we combine all of them into the single FA as:



Conversion of Regular Expression To Finite Automata_Example_2

Convert following Regular Expression to NFA and then to DFA

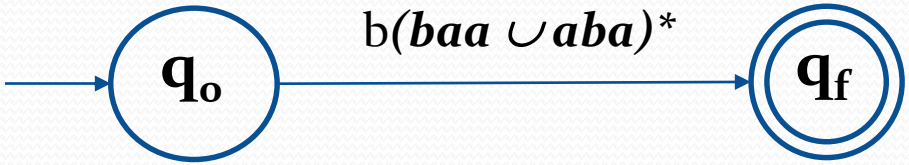
$$L{:}b(baa \cup aba)^*$$

- *To convert the given RE into the required FA we perform the step wise transition of the state for a particular input.*
- *The final FA will accept all the set of string generated by the given RE.*
- *So, we draw the FA step by step which will accept all the string that belongs to that language.*

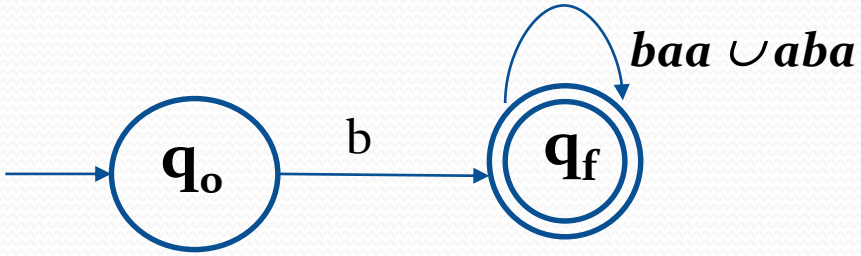
Conversion of Regular Expression To Finite Automata_Example_2

Here, $L: b(baa \cup aba)^*$

We know the FA being constructed will moves to final state after consuming $b(baa \cup aba)^*$

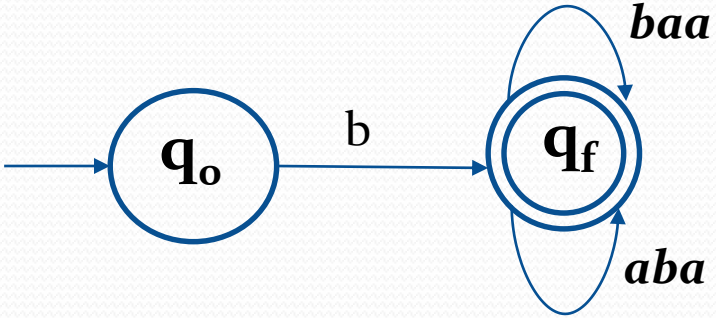


We first transit the automata for ***b*** and then $(baa \cup aba)^*$ which will take the automata to final state.

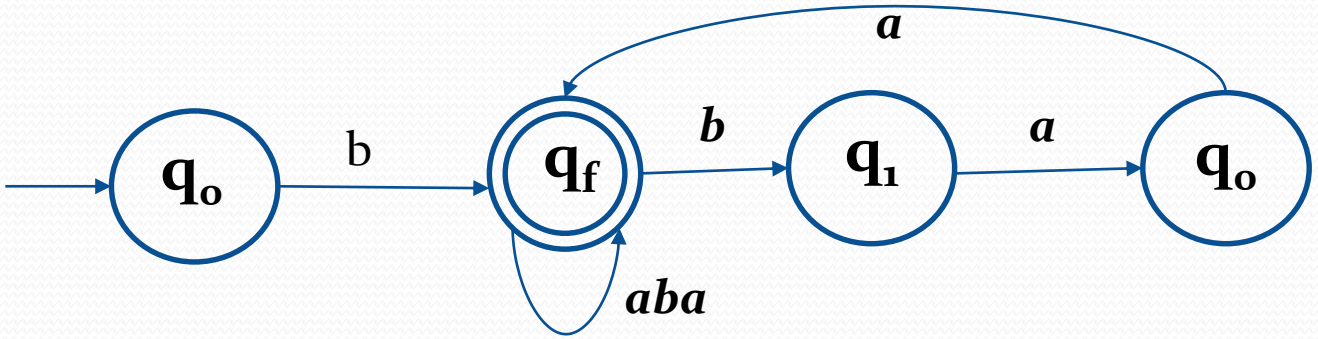


Conversion of Regular Expression To Finite Automata_Example_2

Now to break down the input (baa ∪ aba)* as follows:

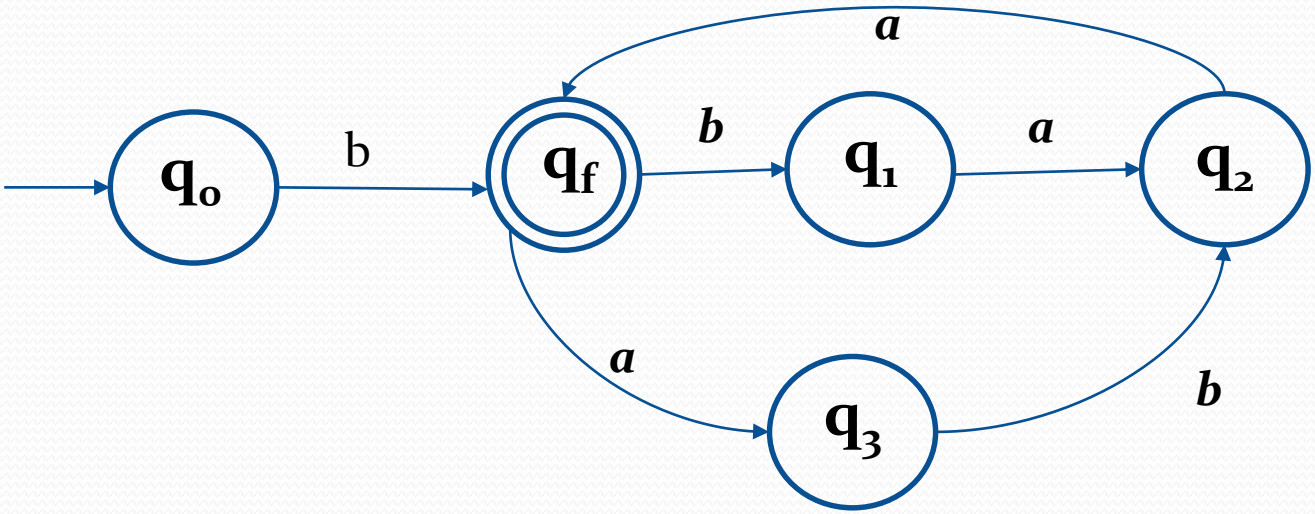


Again we further break down the input ***baa*** as:



Conversion of Regular Expression To Finite Automata_Example_2

Again we further break down the input ***aba*** as:



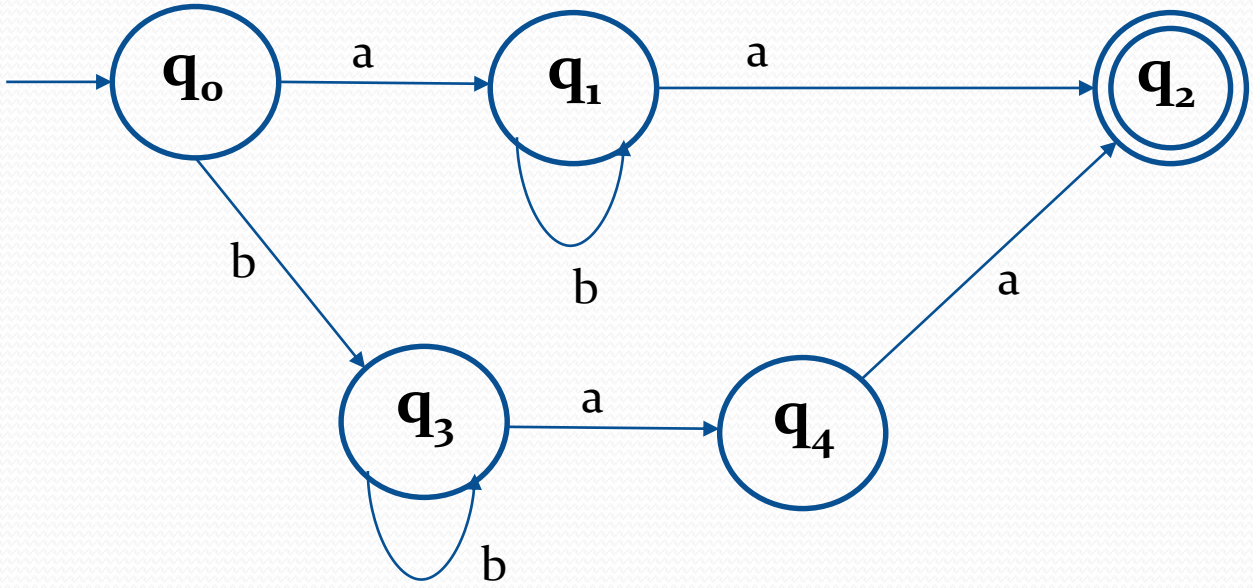
There is no other input that needs to be further broken down so this is the required FA for the given language.

Finally convert this NFA to its equivalent DFA

Conversion of Regular Expression To Finite Automata_Example_3

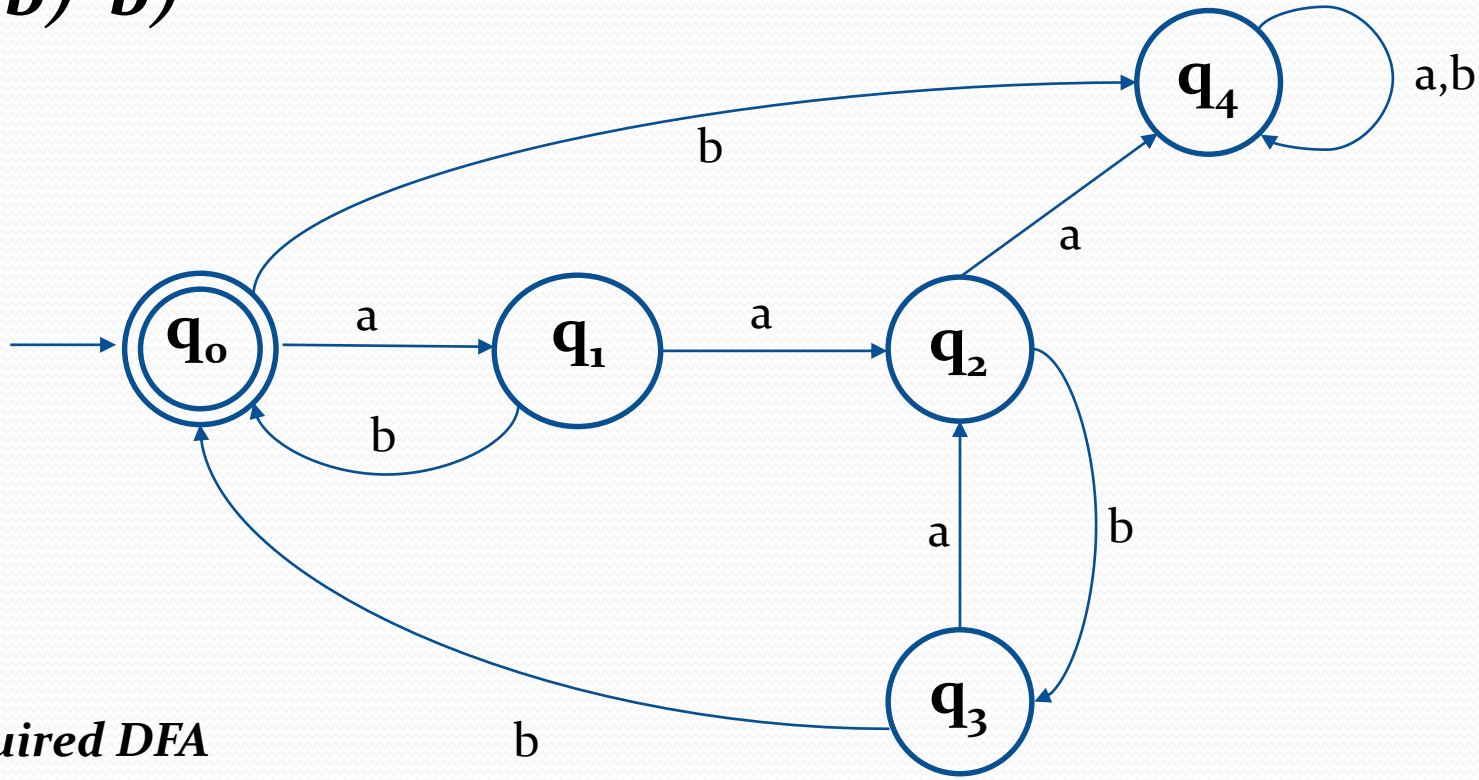
Construct a NFA for the language

$$L: (ab^*a \cup b^*aa)$$



Conversion of Regular Expression To Finite Automata_Example_4

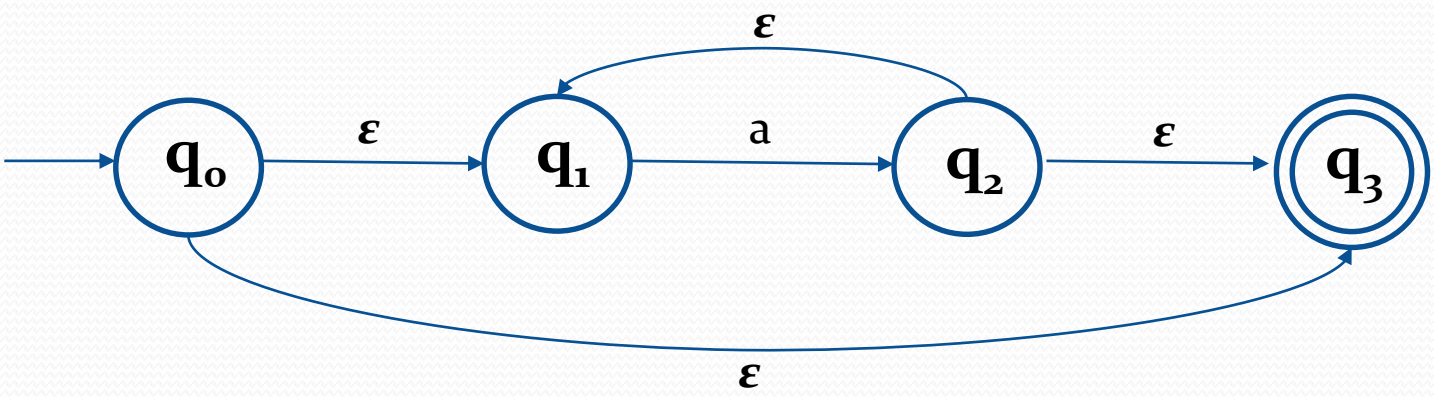
Design a Deterministic Finite Automata (DFA) for the RE:***(a(ab)*b)****



Required DFA

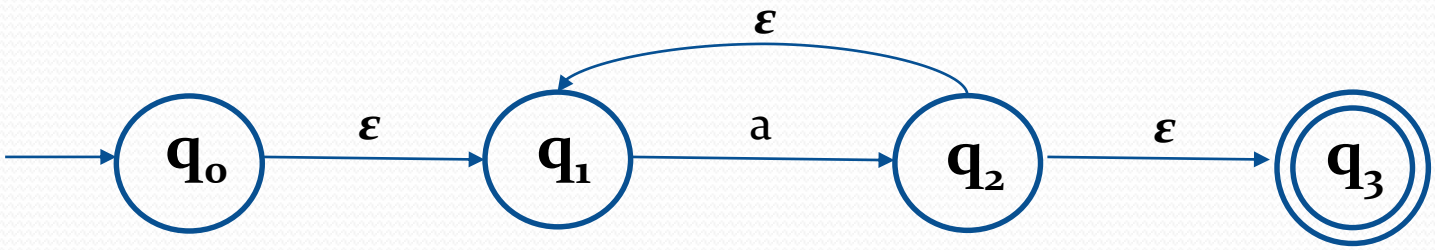
Conversion of Regular Expression To ε-NFA

- For the conversion of the regular expression to its equivalent ***ε-NFA,*** we have some important basic rules as we used before (***including now ε***) which is as follows:
  - When we have the expression like ***a****

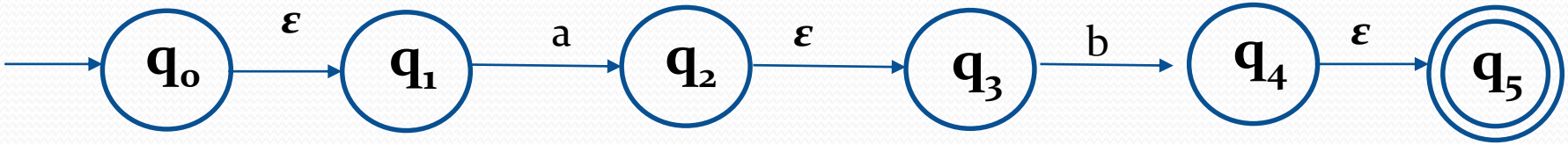


Conversion of Regular Expression To ε-NFA

- When we have the expression like a^+

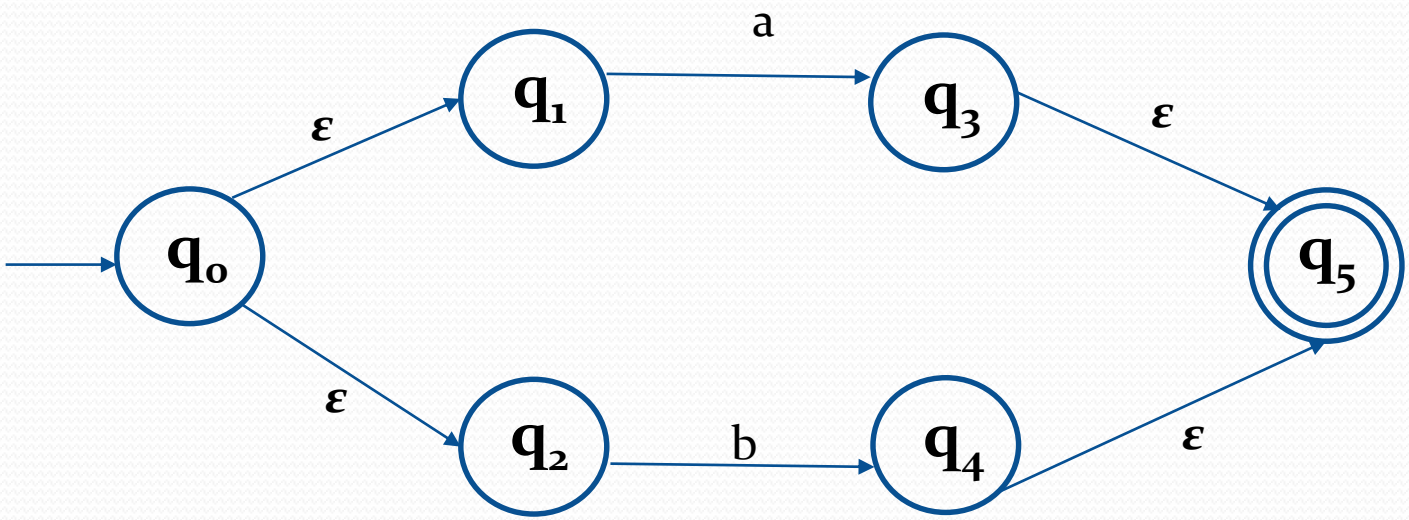


- When we have the expression like (a.b)



Conversion of Regular Expression To ε-NFA

- When we have the expression like ***(a+b)*** or ***(a ∪ b)*** or ***(a|b)***



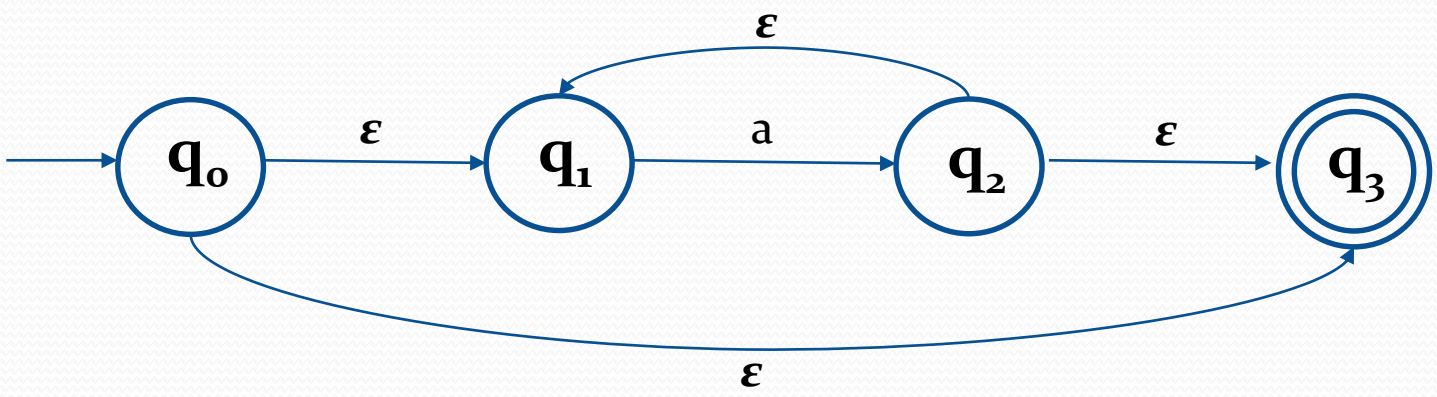
Conversion of Regular Expression To ε-NFA_Example_1

- Convert the given regular expression into ε-NFA

RE: (a+b)*

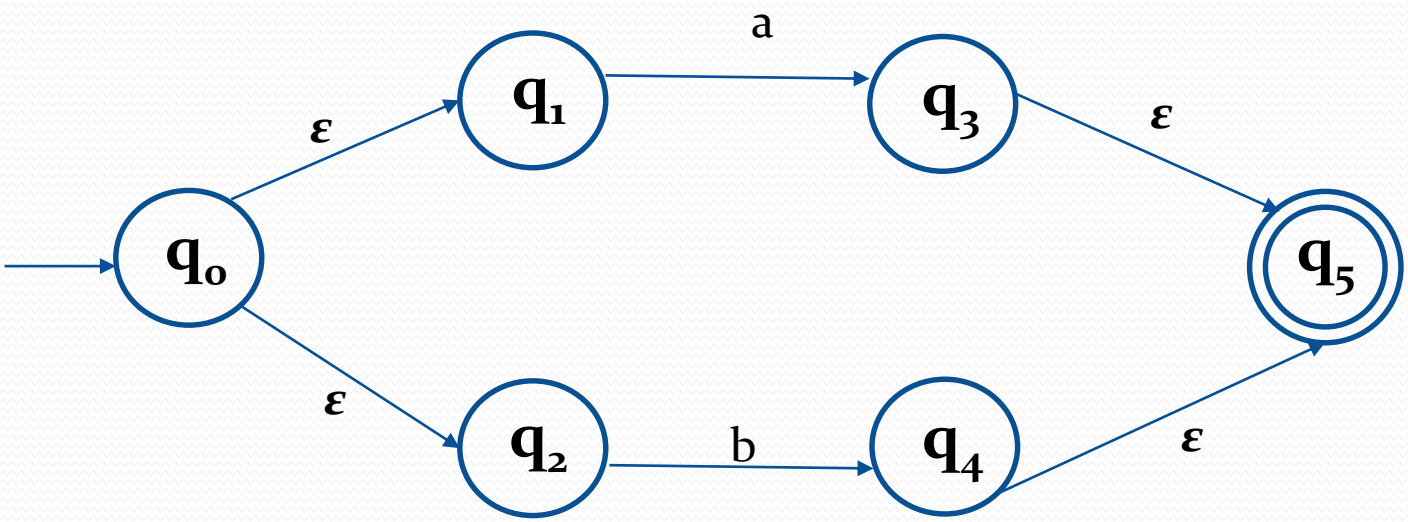
Solution,

Let us think (a+b)* whole as a* then draw the ε-NFA as per the rule of ***a**** as:



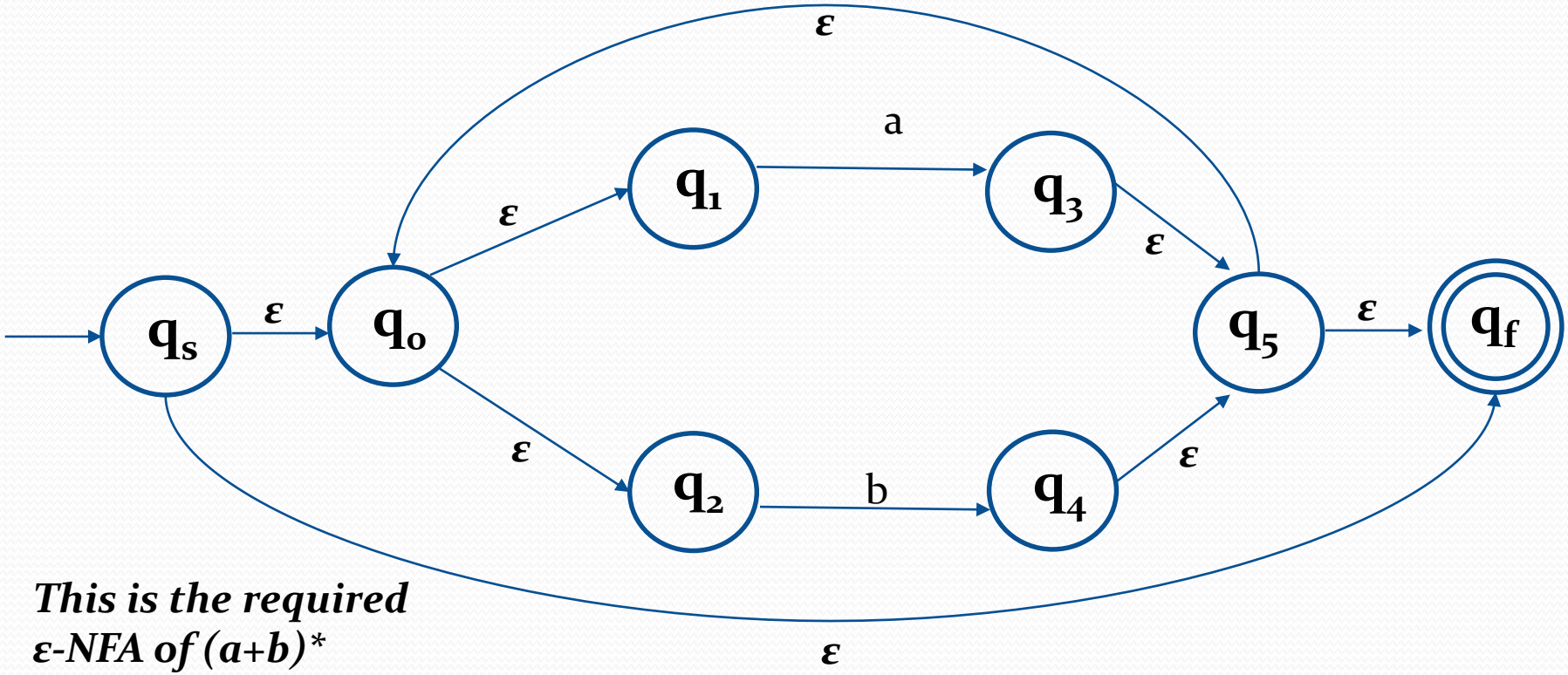
Conversion of Regular Expression To ε-NFA_Example_1

Now substitute ***a+b*** 2nd diagram in place of ***a*** in 1st diagram as



Conversion of Regular Expression To ε-NFA_Example_1

But we have ***(a+b)*** instead of ***a*** so, ***a+b*** form can be formed from the rule of ***ε*** as:



This is the required ε-NFA of (a+b)*

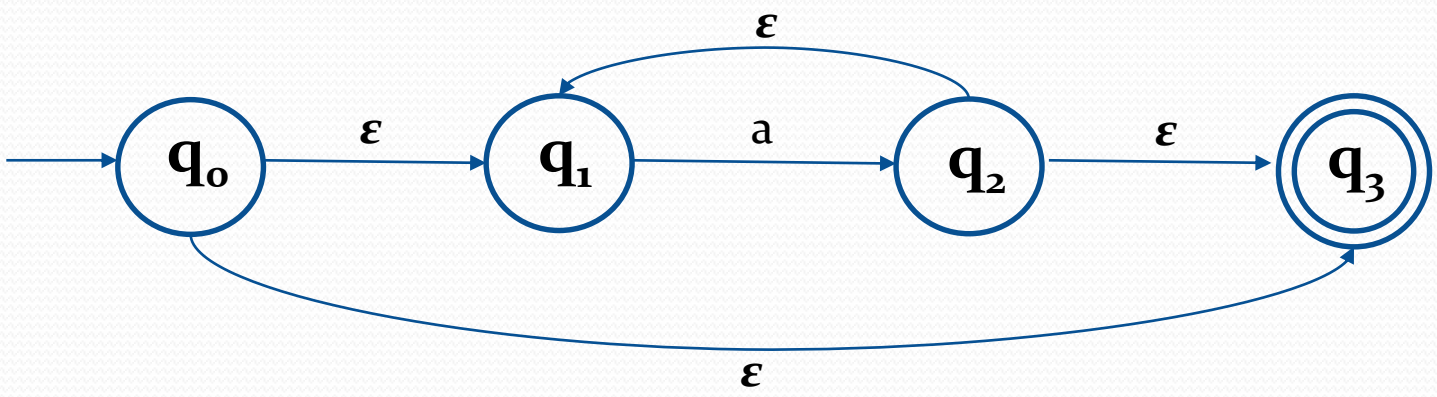
Conversion of Regular Expression To ε-NFA_Example_2

- Convert the given regular expression into ε-NFA

RE: (00+11)*

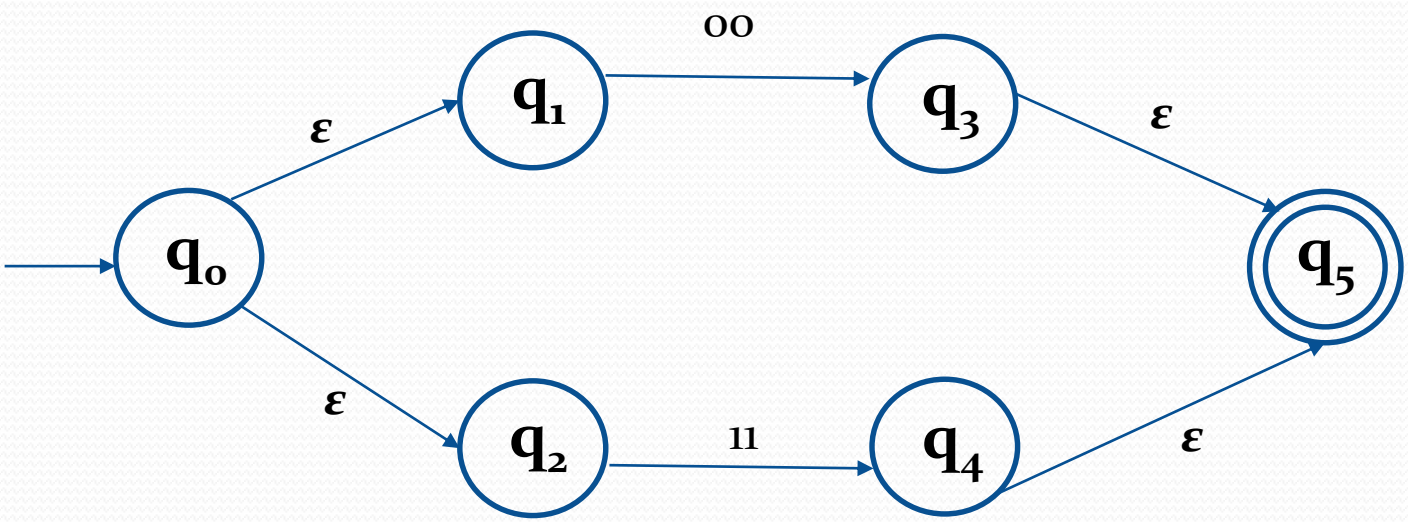
Solution,

Let us think (00+11)* whole as a* then draw the ε-NFA as per the rule of ***a**** as:



Conversion of Regular Expression To ε-NFA_Example_2

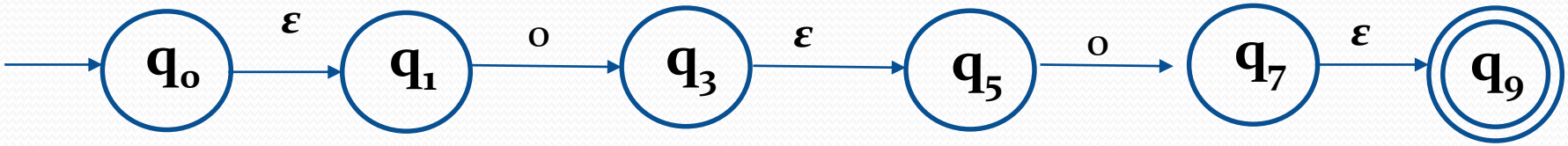
Now substitute ***00+11*** in place of ***a*** in 1st diagram as

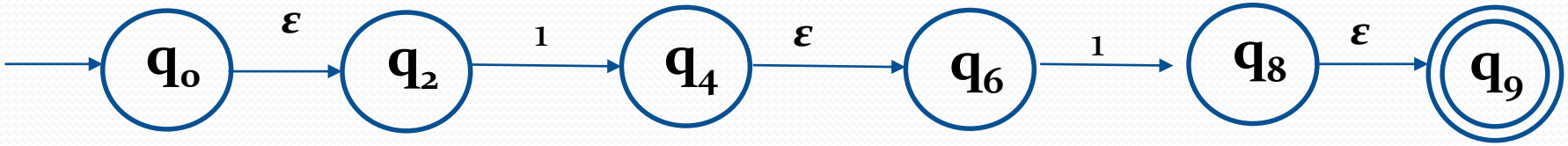


Conversion of Regular Expression To ε-NFA_Example_2

But instead of ***a*** we have ***00+11***. Again two inputs 00 and 11 cannot remain at same place in diagram so we subdivide 00 and 11 and draw for each separately as follows;

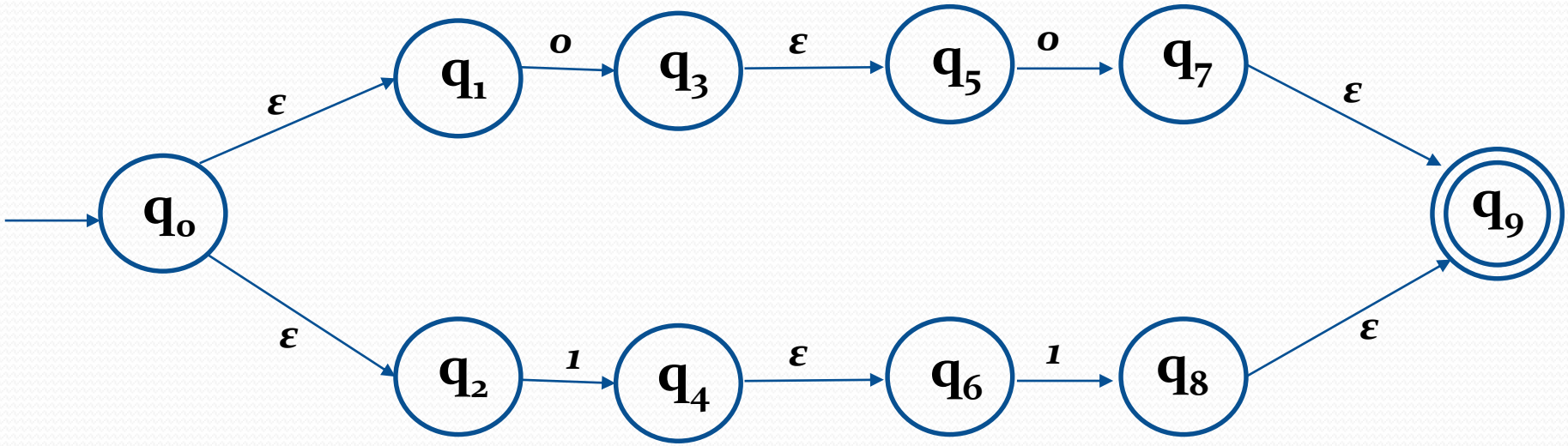
for 00 and 11





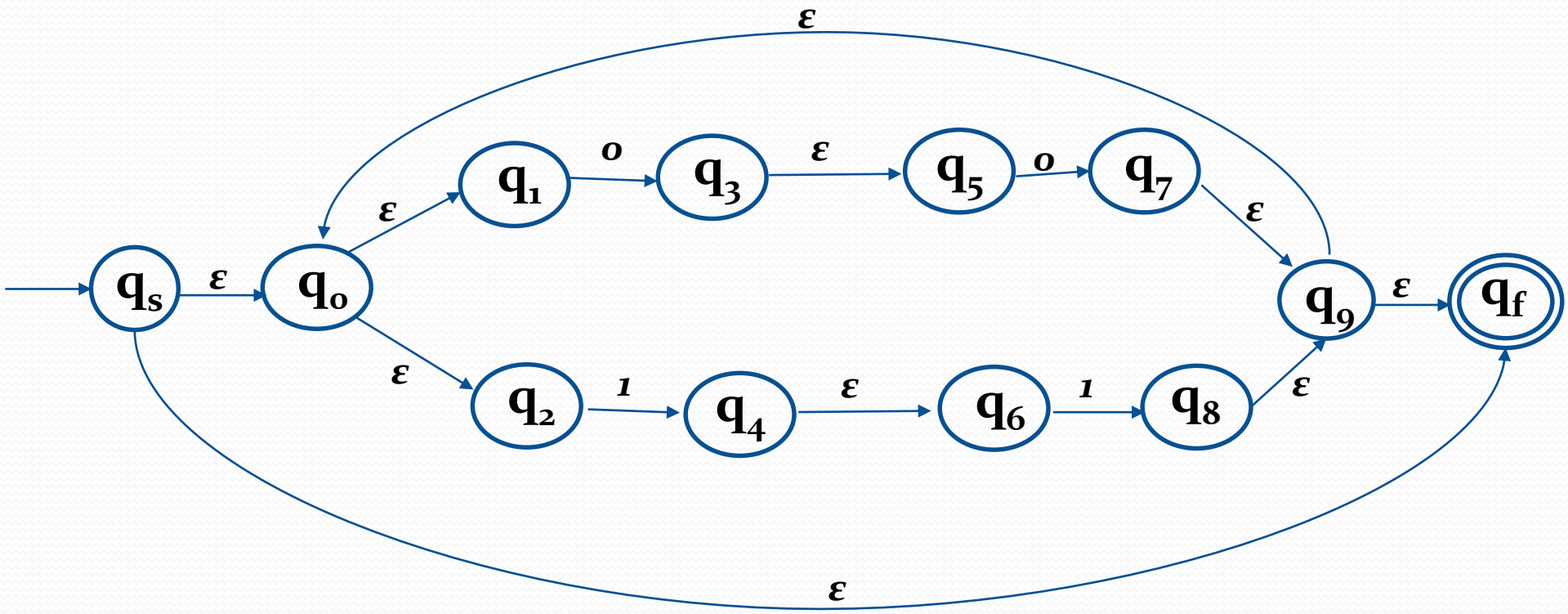
Conversion of Regular Expression To ε-NFA_Example_2

Again ***00 and 11*** can be converted in ε-NFA as:



Conversion of Regular Expression To ε-NFA_Example_2

Now finally its closure is drawn as:



This is the required ε-NFA of (00+11)*

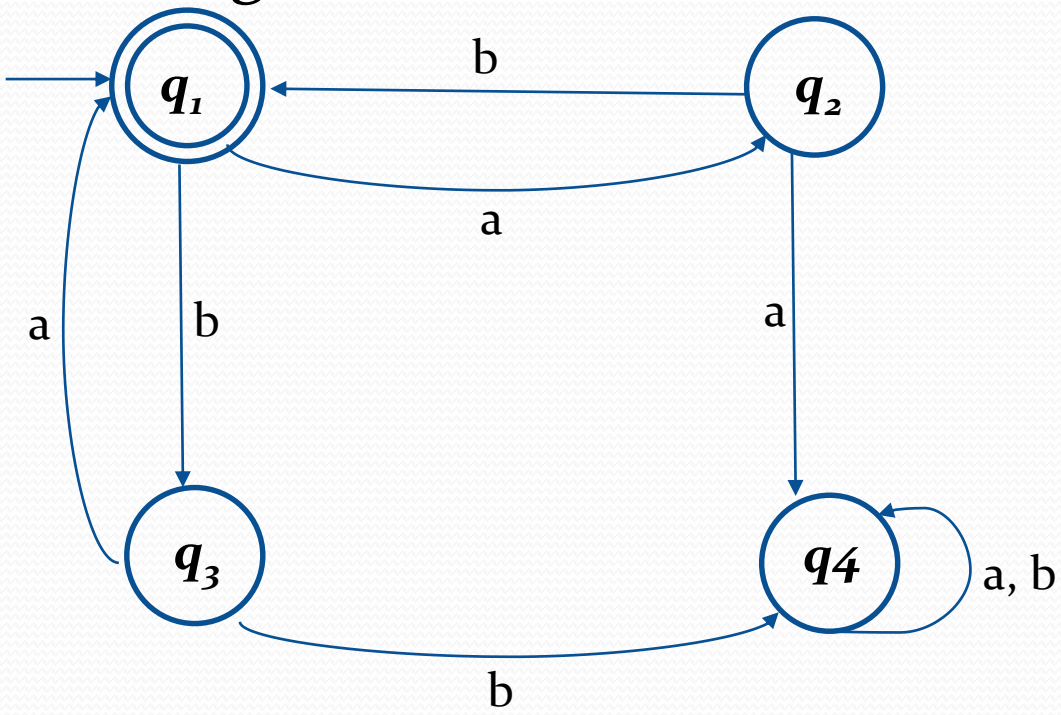
Conversion of FA to Regular Expression

- We use Arden's Theorem to convert the FA to RE.
- Let P and Q be the two regular expression such that P does not contain the empty string (ε) then regular expression R of the form $R=Q+RP$ has the unique solution as:

$$R=QP^*$$

Conversion of FA to Regular Expression

- Find RE for the given DFA



Conversion of FA to Regular Expression

Here the equation for each in terms of incoming transitions are:

$q_1=\varepsilon+q_2b+q_3a$----------(i) *{incoming coming from no where so ε}*

$q_2=q_1a$----------(ii)

$q_3=q_1b$----------(iii)

$q_4=q_2a+q_3b+q_4a+q_4b$----(iv)

Conversion of FA to Regular Expression

Taking equation (i) since it is final state

$q_1=\varepsilon+q_2b+q_3a$----------(i)

Substituting value of q_2 andq_3 from (ii) and (iii)

$q_1=\varepsilon+q_1ab+q_1ba$

$q_1=\varepsilon+q_1(ab+ba)$---------(v)

*Equation (v) is of the form **R=Q+RP**, so we can write it in the form **R=QP*** **according to Arden's Theorem.***

i.e. $q_1=\varepsilon(ab+ba)^*$

$q_1=(ab+ba)^*$, [according to algebraic laws of regular expression $\varepsilon.R=R$]

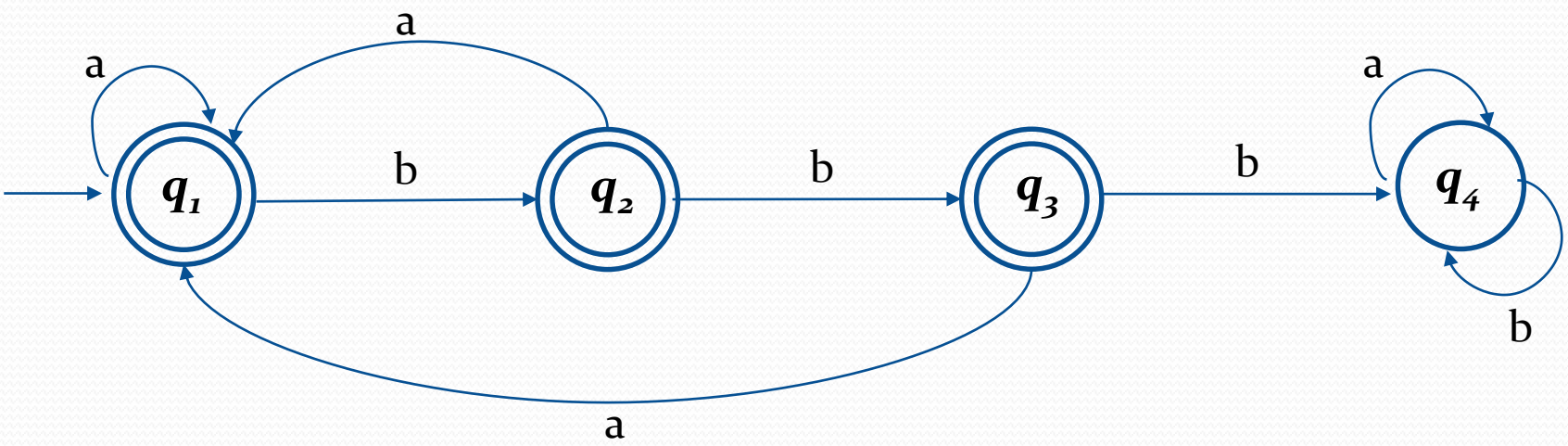
Conversion of FA to Regular Expression

Since q_1 is the final state and we have derived the equation of q_1 hence $q_1=(ab+ba)^*$ is the required regular expression.

so required ***RE=(ab+ba)****

Conversion of FA to Regular Expression

Obtain the RE for the given FA



Conversion of FA to Regular Expression

Here the equation for each in terms of incoming transitions are:

$q_1=\varepsilon+q_1a+q_2a+q_3a$----------(i)

$q_2=q_1b$----------(ii)

$q_3=q_2b$----------(iii)

$q_4=q_3b+q_4a+q_4b$----(iv)

Conversion of FA to Regular Expression

Taking equation (i) since q_1 it is final state

$q_1 = \varepsilon + q_1a + q_2a + q_3a$

Substituting value of q_2 andq_3 from (ii) and (iii)

$q_1 = \varepsilon + q_1a + q_1ba + q_2ba$

$q_1 = \varepsilon + q_1a + q_1ba + q_1bba$

$q_1 = \varepsilon + q_1(a + ba + bba)$---------(v)

*Equation (v) is of the form **R=Q+RP**, so we can write it in the form **R=QP*** **according to Arden's Theorem.***

i.e. $q_1 = \varepsilon(a + ba + bba)^*$

$q_1 = (a + ba + bba)^*$, [according to algebraic laws of regular expression $\varepsilon.R = R$]

Conversion of FA to Regular Expression

Taking equation (ii) since q_2 is also final state

$q_2=q_1b$

$q_2=(a+ba+bba)^*b$

Taking equation (iii) since q_3 is also final state

$q_3=q_2b$

$q_3=(a+ba+bba)^*bb$

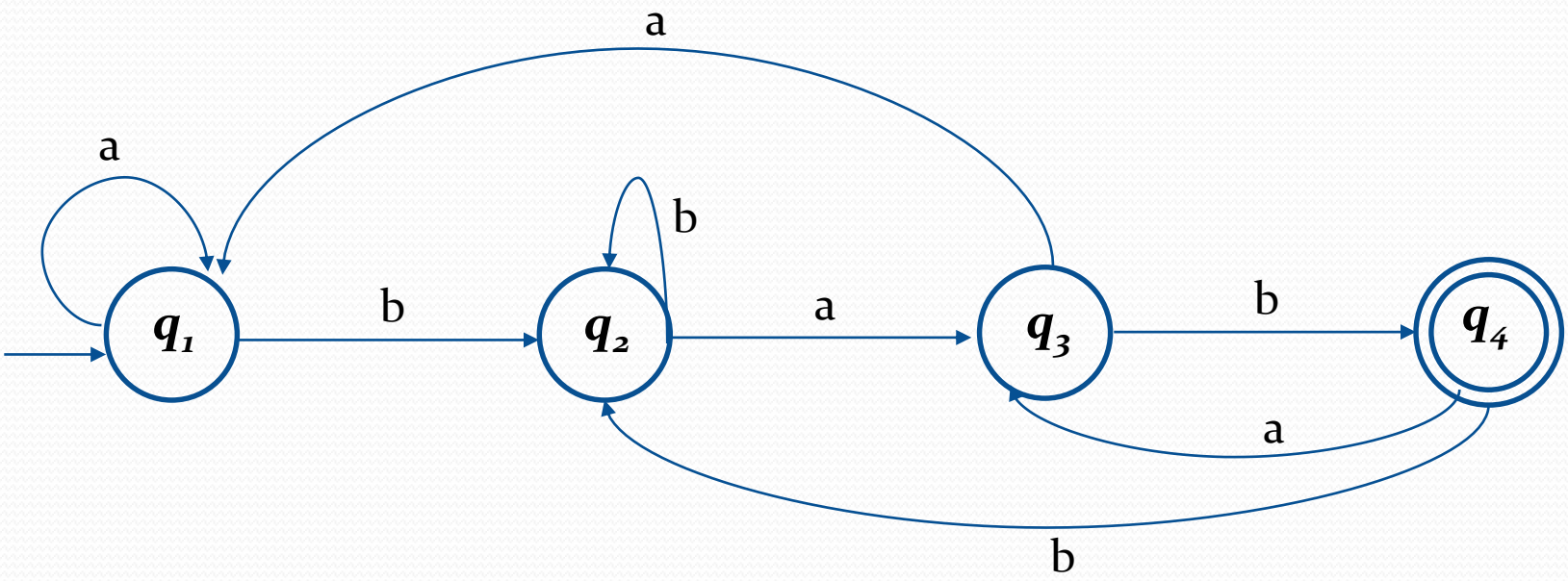
We have derived the expression for all the final states so the regular expression is union of all the obtained expression i.e.

$$q_1 \cup q_2 \cup q_3$$

$$RE=(a+ba+bba)^* \cup (a+ba+bba)^*b \cup (a+ba+bba)^*bb$$

Conversion of FA to Regular Expression

Find RE for given DFA



Closure Properties of Regular Language

- Closure refers to some operation on a language, resulting in a new language that is of the same "type" as originally operated on i.e. regular.
- Those operations on regular languages that are certain to result in regular language are known as closure qualities.
- The class of language accepted by finite automata is closed under:
  - Union
  - Concatenation
  - Kleen Closure
  - Complement
  - Intersection

Closure Properties of Regular Language (Union)

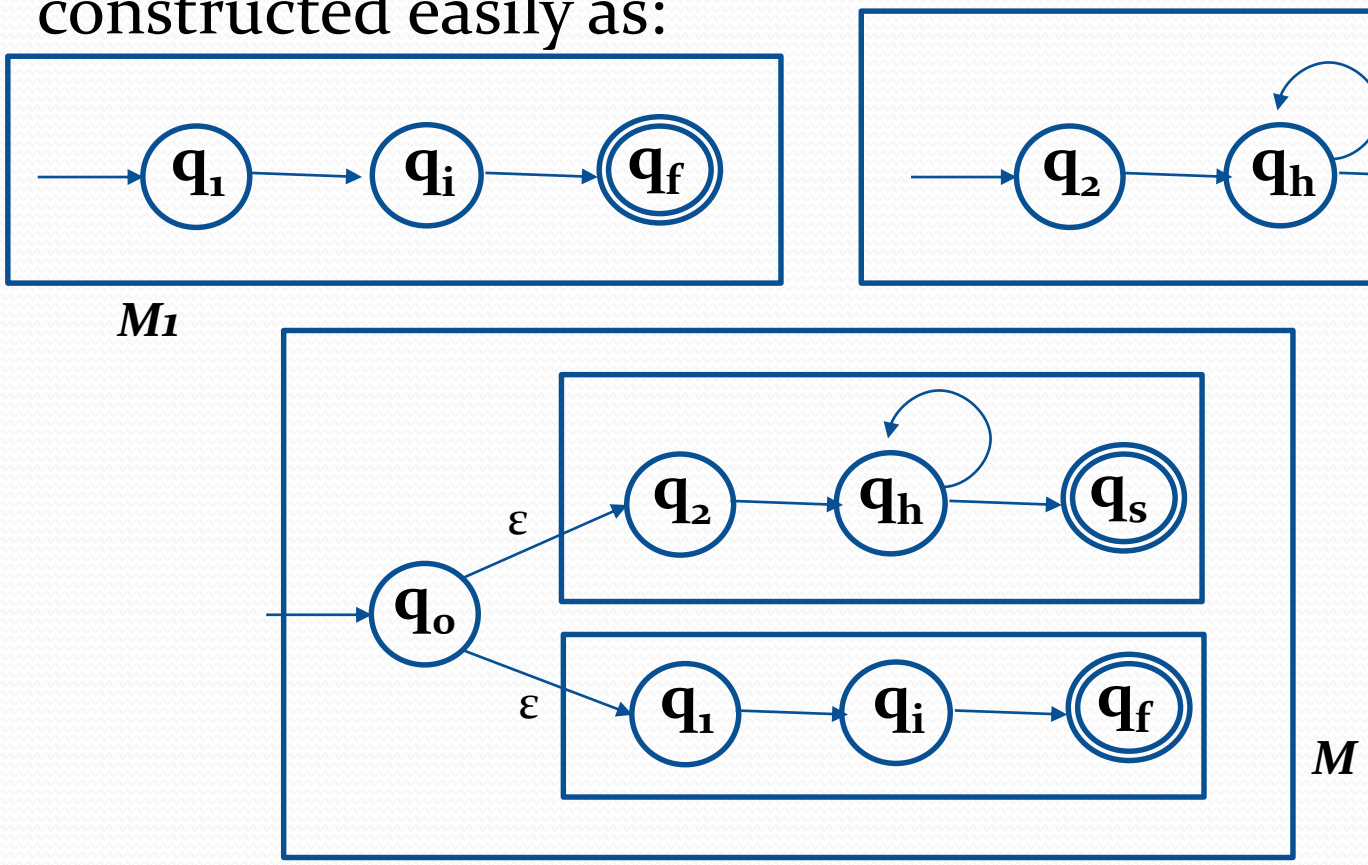
- Let R1 and R2 are the regular language then the union of R1 and R2 denoted by R1+R2 or R1 ∪ R2 is also regular.
- To show that R1+R2 are regular language we need to design a non deterministic automata which can recognize the language (R1+R2).
- For this let us consider the non deterministic automata M1 which can recognize R1 i.e. string generated by R1 will be accepted by M1.
- M1 can be defined as $M1 = (Q1, \Sigma1, \delta1, q_1, F1)$ *such that R1=L(M1)*

Closure Properties of Regular Language (Union)

- Also consider another non deterministic automata M2 which can recognize R2 i.e. string generated by R2 will be accepted by M2.
- M2 can be defined as: $M2 = (Q2, \Sigma 2, \delta 2, q_2, F2)$ *such that* $R2 = L(M2)$
- We now construct a non deterministic automata such that L(M)=L(M1)+L(M2).

Closure Properties of Regular Language (Union)

- A simple non deterministic automata can be constructed easily as:



Closure Properties of Regular Language (Union)

- Simply newly constructed automata M uses non-determinism to guess whether input is in L(M1) or in L(M2).
- The automata M then processes the string exactly as the corresponding automata either M1 or M2 so it follows L(M)=L(M1)+L(M2).
- The formal proof can be given as:

From these two NFAs, we will construct an NFA

M = (Q, ∑, δ, qo, F), such that L(M) = L(M1) + L(M2)

Closure Properties of Regular Language (Union)

- Different tuple of M can be defined as:

$Q: Q_1 \cup Q_2 \cup q_0$

$\sum : \sum_1 \cup \sum_2 \cup \{\varepsilon\}$

$q_0 : q_0$

F: Set of Final States ($F \subseteq Q$)

$\delta : \delta_1 \cup \delta_2 \cup \delta(q_0, \varepsilon) \rightarrow q_1 \cup \delta(q_0, \varepsilon) \rightarrow q_2$

Closure Properties of Regular Language (Union)

- That is M beings any computation by non deterministically choosing to enter either initial sate of M1, or the initial state of M_2.
- M then reproduces either M1, or M2, thereafter.
- Hence M accepts the string ***w*** if and only if M1, accepts ***w*** or M2 accepts ***w*** and L(M)=L(M1)+L(M2).
- Hence, regular properties are closed under union.
- **Steps in Union of R1 and R2**
  - *Create a new start state.*
  - *Make a ε-transition from the new start state to each of the original start states.*

Closure Properties of Regular Language (Concatenation)

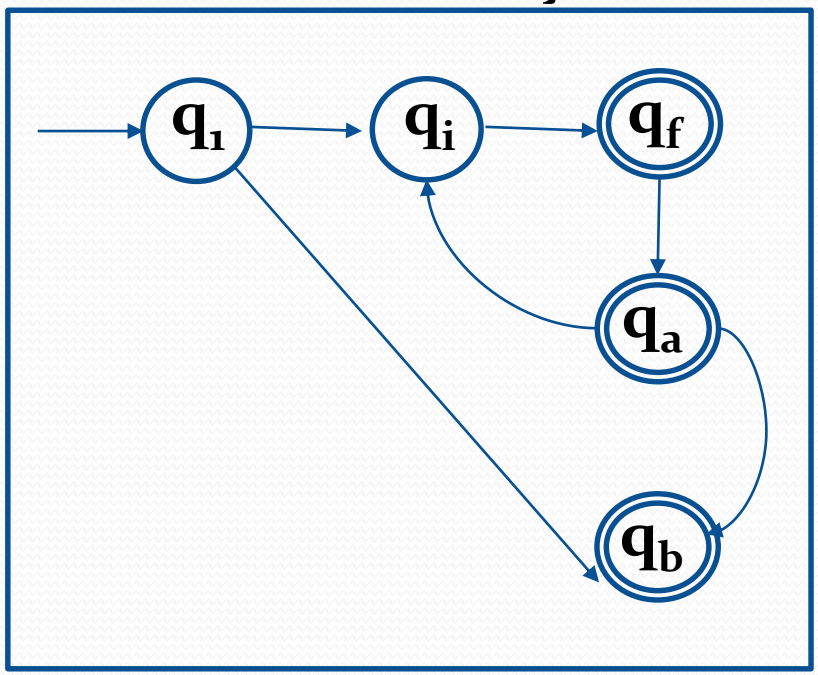
- Let R1 and R2 are the regular language then the concatenation of R1 and R2 denoted by R1.R2 is also regular.
- To show that R1.R2 are regular language we need to design a non deterministic automata which can recognize the language (R1.R2).
- For this let us consider the non deterministic automata M1 which can recognize R1 i.e. string generated by R1 will be accepted by M1.
- M1 can be defined as $M1 = (Q1, \Sigma1, \delta1, q_1, F1)$ *such that R1=L(M1)*

Closure Properties of Regular Language (Concatenation)

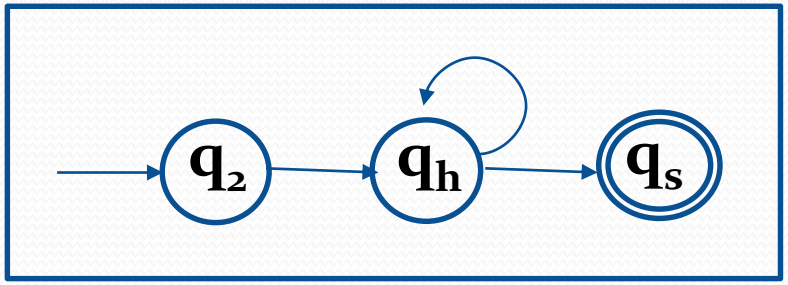
- Also consider another non deterministic automata M2 which can recognize R2 i.e. string generated by R2 will be accepted by M2.
- M2 can be defined as: $M2 = (Q2, \Sigma 2, \delta 2, q_2, F2)$ *such that* $R2 = L(M2)$
- We now construct a non deterministic automata such that L(M)=L(M1).L(M2).

Closure Properties of Regular Language (Concatenation)

- A simple non deterministic automata can be constructed easily as:

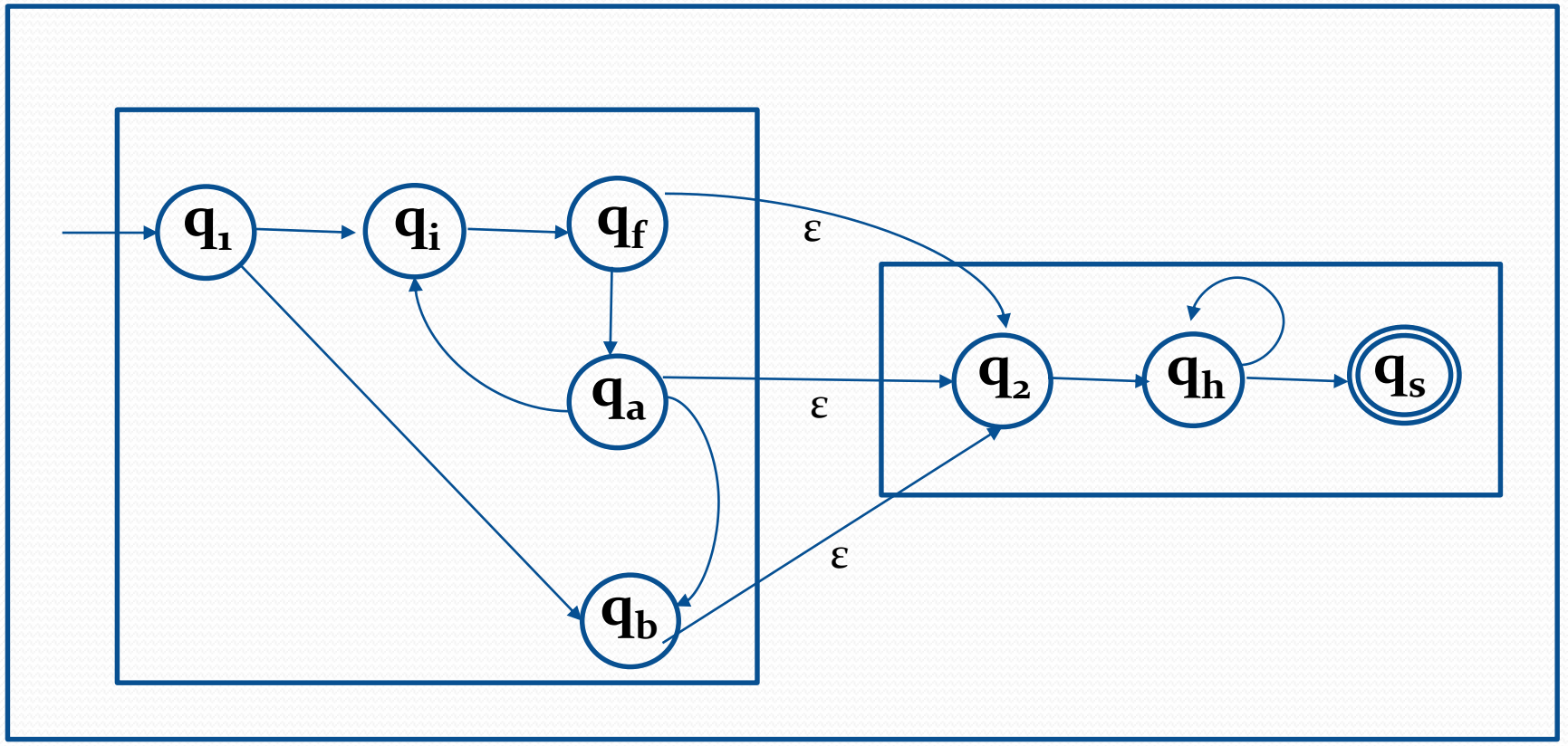


M1



M2

Closure Properties of Regular Language (Concatenation)



M

Closure Properties of Regular Language (Concatenation)

- Simply newly constructed automata M starts from M1, after consuming all the string generated by R1 it reaches to the one of the final states of M1.
- Then using non-determinism the automata reaches to the starting state of M2.
- The automata M then processes the string exactly as the corresponding automata of M2 making L(M)=L(M1).L(M2).
- The formal proof can be given as:

From these two NFAs, we will construct an NFA

M = (Q, Σ, δ, q0, F), such that L(M) = L(M1).L(M2)

Closure Properties of Regular Language (Concatenation)

- Different tuple of M can be defined as:

  Q: $Q_1 \cup Q_2$

  $\sum : \sum_1 \cup \sum_2 \cup \{\varepsilon\}$

  q_0: q_1

  F: Set of Final States ($F \subseteq Q_2$)

  δ : $\delta_1 \cup \delta_2 \cup \delta(q_f, \varepsilon) \rightarrow q_2$ where $q_f \in F_1$

- That is M beings from starting state of M1 and transits according to transition function of M1 until it reaches the final state of M1.

Closure Properties of Regular Language (Concatenation)

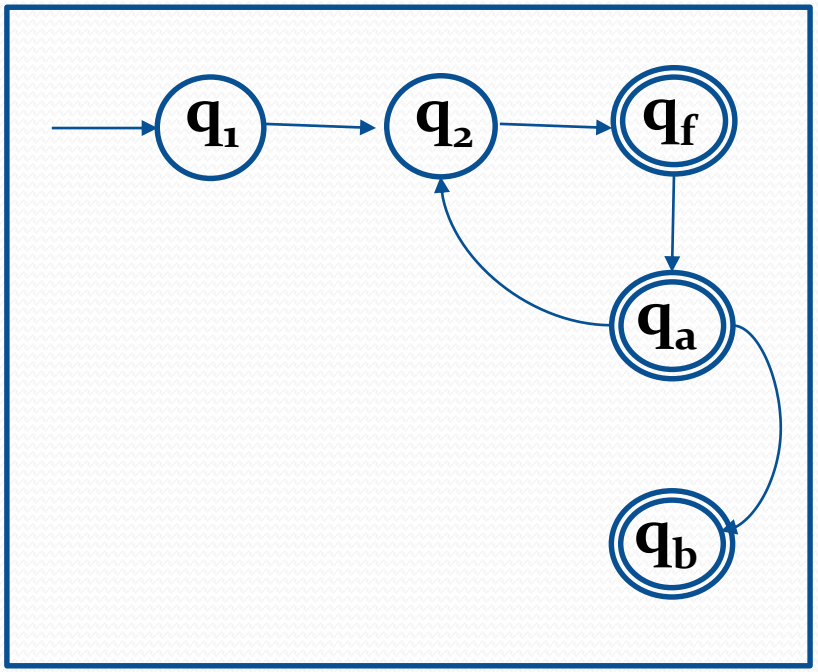
- Then from any of the final state of M1 the automata by non determinism enters to the initial state of M_2.
- M then reproduces M2, thereafter.
- Hence M accepts the string ***w*** if and only if ***M2*** accepts ***w*** and L(M)=L(M1).L(M2).
- Hence, regular properties are closed under concatenation.
- **Steps in Concatenation of R1 and R2**
  - *Put a ε-transition from each final state of M1 to the initial state of M2.*
  - *Make the original final states of M1 as non final.*

Closure Properties of Regular Language (Kleen Closure)

- Let R1 be the regular language then the kleen closure of R1 denoted by R1* also regular.
- To show that R1* is regular language we need to design a non deterministic automata which can recognize the language R1*.
- For this let us consider the non deterministic automata M1 which can recognize R1 i.e. string generated by R1 will be accepted by M1.
- M1 can be defined as $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ *such that* $R_1 = L(M_1)$

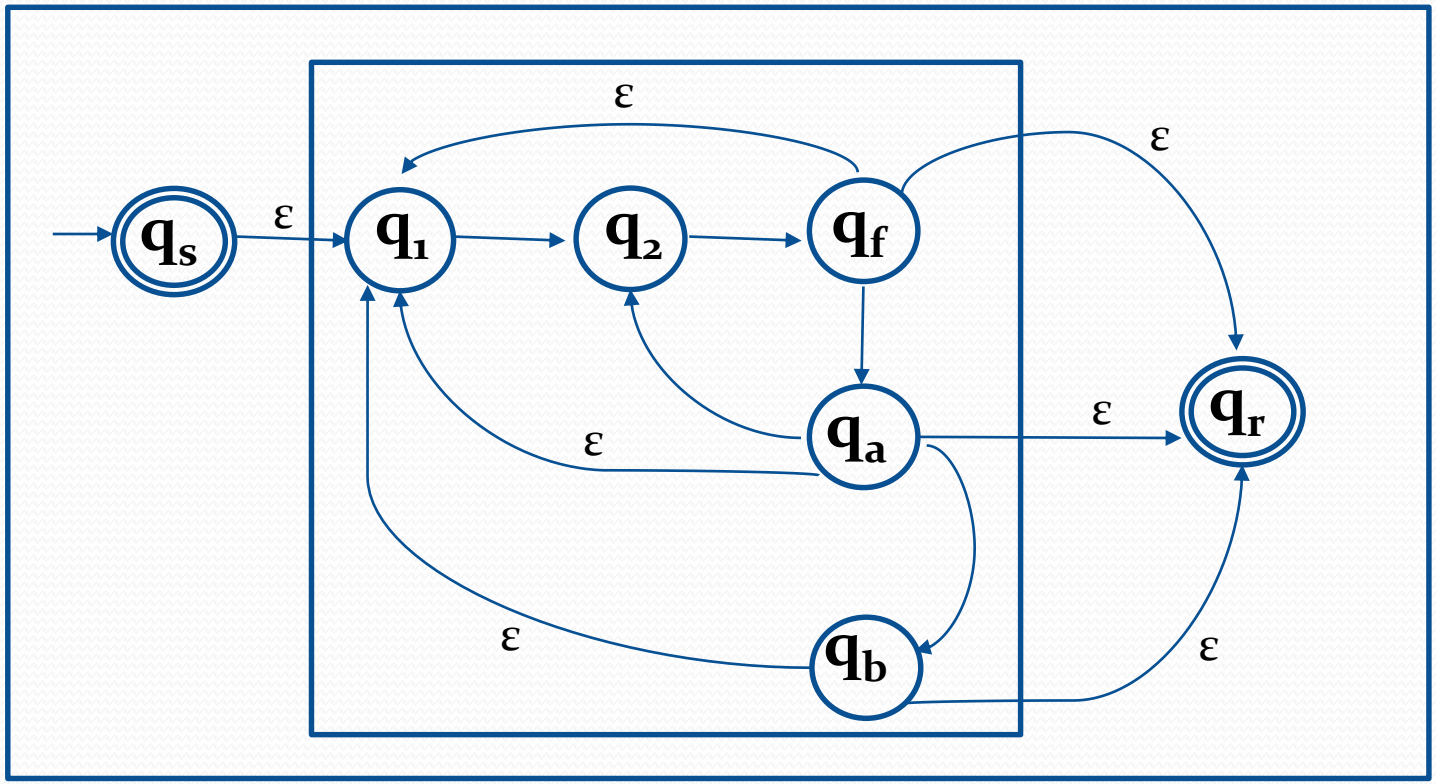
Closure Properties of Regular Language (Kleen Closure)

- We now construct a non deterministic automata such that L(M)=R1*.



M1

Closure Properties of Regular Language (Kleen Closure)



M

Closure Properties of Regular Language (Kleen Closure)

- Simply newly constructed automata M begins from new starting state (which is also the final state as it may accept ε string) and uses non-determinism to reach the starting state of M1.
- The automata M then processes the string exactly as the corresponding automata of M1.
- For the repetition of the string the automata uses the non-determinism from any of the final state of M1 to the initial state of M1 making L(M)=R1*.
- The automata finally uses non-determinism to reach any of the final state of M from the original final state of M1.

Closure Properties of Regular Language (Kleen Closure)

- The formal proof can be given as:

From the original NFA, we will construct new NFA

M = (Q, ∑, δ, qo, F), such that L(M) = R1*

- Different tuple of M can be defined as:

Q: $Q_1 \cup \{q_s\} \cup \{q_r\}$

∑: $\sum_1 \cup \{\varepsilon\}$

q_o: q_s

F: $\{q_s, q_r\}$

δ : $\delta_1 \cup \delta(q_i, \varepsilon) \rightarrow q_1 \cup \delta(q_s, \varepsilon) \rightarrow q_1$ where $q_i \in F_1$

Closure Properties of Regular Language (Kleen Closure)

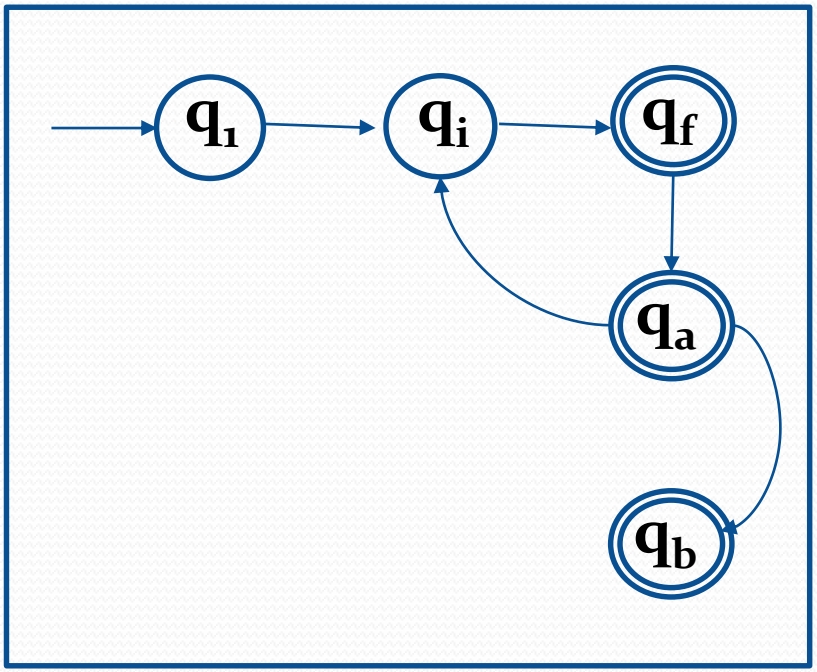
- **Steps in Kleene Closure of R1**
  - Make a new start state (which is also the final state as it may accept ε string); connect it to the original start state with a ε-transition.
  - Make a new final state; connect the original final state (which becomes non final) to it with ε -transitions.
  - For the repetition of the string use the ε -transitions from any of the final state of M1 to the original initial state i.e. of M1.

Closure Properties of Regular Language (Complement)

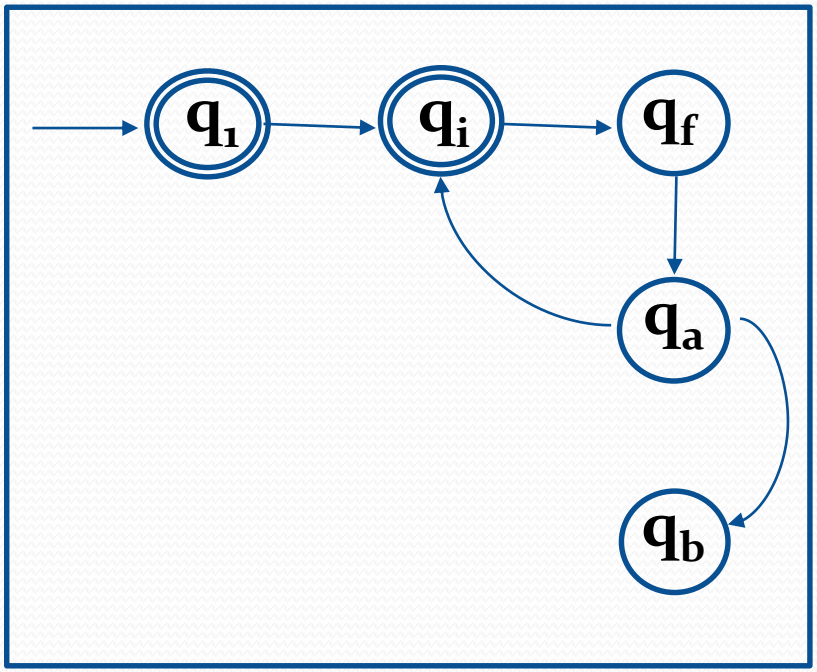
- Let R1 be the regular language then the complement of R1 denoted by $\overline{R1}$ also regular.
- To show that $\overline{R1}$ is regular language we need to design a non deterministic automata which can recognize the language $\overline{R1}$.
- For this let us consider the non deterministic automata M1 which can recognize $\overline{R1}$ i.e. string generated by $\overline{R1}$ will be accepted by M1.
- M1 can be defined as $M1 = (Q1, \Sigma1, \delta1, q_1, F1)$ *such that* $\overline{R1} = L(M1)$

Closure Properties of Regular Language (Complement)

- We now construct a non deterministic automata such that L(M)=$\overline{R1}$.



M1



M

Closure Properties of Regular Language (Complement)

- The formal proof can be given as:

From the original NFA, we will construct new NFA

M = (Q, ∑, δ, qo, F), such that L(M) = $\overline{R1}$

- Different tuple of M can be defined as**:**

Q: Q_1

∑: $\sum_1$

q_0: q_1

F: {Q-F_1}

δ : δ_1

Closure Properties of Regular Language (Complement)

- **Steps in Complement of R1**
  - Make the non final states of original automata as a final state and vice versa.
- ***Intersection Properties of Regular Language***

$$R1 \cap R2 = \overline{\overline{R1} + \overline{R2}}$$ using De-Morgan's Law

  - If R1 and R2 is regular then $\overline{R1}$ and $\overline{R2}$ are also regular using complement properties.
  - If $\overline{R1}$ and $\overline{R2}$ are regular then $\overline{R1} + \overline{R2}$ is also regular by union properties.
  - $\overline{R1} + \overline{R2}$ is regular then its complement $\overline{\overline{R1} + \overline{R2}}$ is also regular from complement properties.
  - $\overline{\overline{R1} + \overline{R2}}$ is equivalent to $R1 \cap R2$ so, $R1 \cap R2$ is also regular.

Pumping Lemma

- We have concluded that the class of regular languages is closed under various operations, and these languages can be described by (deterministic or nondeterministic) finite automata and regular expressions.
- These properties helped in developing techniques for showing that a language is regular.
- Now, we will discuss about the tool that can be used to prove that certain languages are not regular.

Pumping Lemma

- For the regular language
  - The amount of memory that is needed to determine whether or not a given string is in the language is finite and independent of the length of the string.
  - If the language consists of an infinite number of strings, then this language should contain infinite subsets having a fairly repetitive structure.
- Automatically, languages that do not follow above criteria should be non-regular.

Pumping Lemma

- ***Consider the language L: $\{0^n 1^{2n} : n \geq 0\}$.***
  - We can prove this language to be non regular, since DFA has the limited memory and it seems unlikely that a DFA can remember how many 0's it has seen when it has reached the border between the 0's and the 1's.
  - Thus we cannot generate the number of 1 exactly two times the number of 0.
- ***Consider another language L:$\{0^n : n$ is a prime $\}$***
  - The given language is also not regular because the prime numbers do not seems to have any repetitive structure that can be used by a DFA

Pumping Lemma

- These ideas are accurate but not sufficiently precise to be used in formal proofs.
- We will establish a property that all regular languages must possess called the ***Pumping Lemma***.
- If a language does not have this property, then it must be non regular.
- The pumping lemma states that any sufficiently long string in a regular language can be ***pumped***, i.e., there is a section in that string that can be repeated any number of times, so that the resulting strings are all in the language.

Pumping Lemma for Regular Language

- ***Theorem:*** Let L be a regular language. Then there exists an integer p ≥ 1 and p ≥ m(no. of state of FA), called the pumping length, such that any string w in L, with |w| ≥ n, can be decomposed into three parts as ***w = xyz***, such that:
  - $|y| \geq 1,\ y \notin \varepsilon$
  - $|xy| \leq p$
  - $\textit{for all } i \geq 0,\ xy^i z \in L$
- In words, the pumping lemma states that by replacing the portion ***y*** in ***w*** by zero or more copies of it, the resulting string is still in the language L.

Pumping Lemma for Regular Language

- ***Note***
  - ***If any language pass the Pumping Lemma Test then it is undecidable that the language is regular or not.***
  - ***But, if any language fails the Pumping Lemma Test then the language is obvious not regular.***

Pumping Lemma for Regular Language

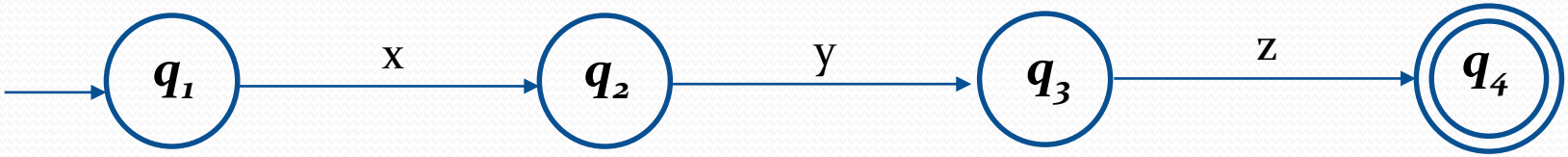
Proof:

Let L be a regular language and $w \in L$.

Let $w = a_1a_2a_3.....a_n$ and the set of states are q_1, q_2, $q_3.....q_m$.

From the theorem of pumping lemma ***w*** can be decomposed into three parts as ***w=xyz***

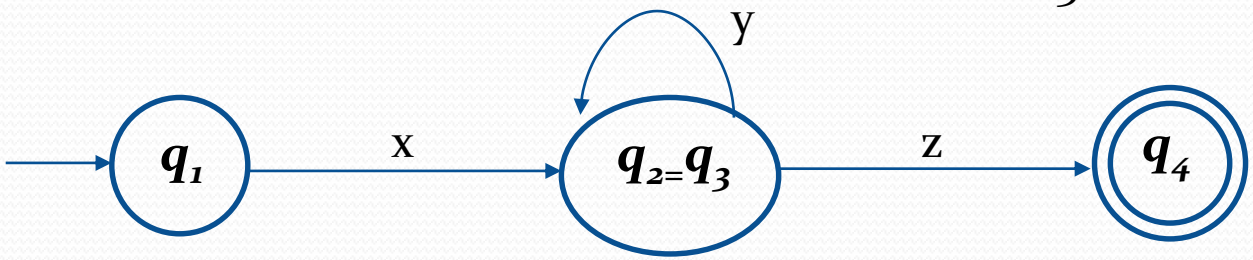
Now its finite automata will be as follows:



Pumping Lemma for Regular Language

Here, |w| = 3, but total states are 4. By pigeon hole principle, any of the two states of FA must coincide.

According to our requirement state q_2 & q_3 gets coincide.



Now in general form,

w = $a_1a_2a_3$.....a_n can be decomposed as:

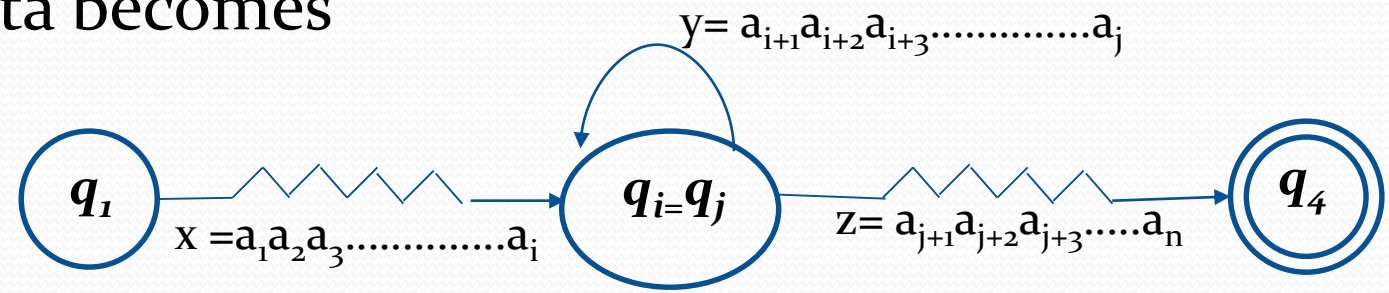
x= $a_1a_2a_3$.....a_i

y= $a_{i+1}a_{i+2}a_{i+3}$.....a_j

z= $a_{j+1}a_{j+2}a_{j+3}$.....a_n

Pumping Lemma for Regular Language

So automata becomes



Now we have to show that $xy^i z \in L$ for i ≥ 0

When i=0, we get $w=xz$ which is processed by FA.

When i=1, we get $w=xyz$, which is also accepted by FA. Similarly, FA processes the strings for all the values of i. So, we say that $xy^i z \in L$ all i ≥ 0.

Hence proved.

Pumping Lemma for Regular Language

- To prove the language L is not Regular using Pumping Lemma make following steps (*prove by contradiction*):
  - Assume that L is regular.
  - It has to have a pumping length (say p)
  - All strings longer than p can be pumped $|w| \geq p$
  - Now find a string 'w' in L such that $|w| \geq p$ and divide w into three parts xyz considering all possible ways such that:

Pumping Lemma for Regular Language

- $|y| \geq 1$, $y \notin \varepsilon$
- $|xy| \leq p$
- Show that $xy^i z \in L$ for some i

- Try to conclude that none can satisfy all the 3 pumping conditions at the same time.
- w cannot be Pumped → CONTRADICTION.

Pumping Lemma for Regular Language_Example_1

Show that the language L: $\{0^n1^n : \text{for } n \geq 0\}$ is not regular.

Solution,

We use the method of proof by contradiction to make the proof. So consider L: $\{0^n 1^n : \text{for } n \geq 0\}$ is regular.

Also, it has a pumping length of ***p***.

So, $w = 0^p1^p$

We now divide ***w*** into three parts ***xyz*** as:

$x = 0^q$

$y = 0^r \; r > 0$

$z = 0^{p-(q+r)}1^p$

Pumping Lemma for Regular Language_Example_1

for pumping factor i=2,

$$xy^2z = 0^q(0^r)^2 0^{p-(q+r)}1^p$$

$$= 0^q 0^{2r} 0^{p-(q+r)}1^p$$

$$= 0^{q+2r+p-q-r}1^p$$

$$= 0^{p+r}1^p$$

Since r >0, p+r > p

Therefore $a^{p+r}\,b^p$ is not of the form $a^p b^p$ *i.e.* $xy^2z \notin L$

Hence the given language is not regular by pumping lemma.

Pumping Lemma for Regular Language_Example_2

Show that the language L: $\{a^n b a^n : \text{for } n \geq 0\}$ is not regular.

Solution,

We use the method of proof by contradiction to make the proof. So consider L: $\{a^n b a^n : \text{for } n \geq 0\}$ is regular.

Also, it has a pumping length of ***p***.

So, ***w***= $a^p b a^p$

Let p=5

$w = a^5 b a^5$

w=aaaaabaaaaa

Pumping Lemma for Regular Language_Example_2

Dividing ***w*** into x, y and z we get,

Case-I: When 'y' contains only 'aa'

w = aa aa abaaaaa [x=aa, y=aa, z=abaaaaa]

Checking for case-I

Let, i=2

w =xy^2z

w = aa aaaa abaaaaa

w = $a^7ba^5 \notin L$

So, case-I fails as the power of a's at the start and end of the string are not equal.

Pumping Lemma for Regular Language_Example_2

Case-II: When 'y' contains 'aaa'

$\boldsymbol{w}$ = aa aaa baaaaa [x=aa, y=aaa, z=baaaaa]

Let, i=2

$\boldsymbol{w} = xy^2z$

$\boldsymbol{w}$ = aa aaaaaa baaaaa

$\boldsymbol{w}$ = aaaaaaaabaaaaa $\notin$ L

So, case-II fails as it doesn't follow the pattern a^nba^n.

Hence, on pumping i.e. $\boldsymbol{xy^i\ z}$ none of the case follow all the three conditions of regular language. So, the given language L: $\{a^nba^n : \text{for } n \geq 0\}$ is not regular.

Pumping Lemma for Regular Language_Example_3

Show that the language L: $\{a^n b^{2n}$: for n ≥ o$\}$ is not regular.