UNIT-III
REGULAR LANGUAGES

REGULAR EXPRESSIONS

- Regular expressions are useful for representing certain sets of strings in an algebraic fashion.
- In arithmetic we can use the operations + and x to build up expressions such as (5 + 3) x 4.
- Similarly we can use the regular operations to build up expressions describing languages, which are called regular expressions. An example is (0 ∪ 1)0*.
- The value of the arithmetic expression is the number 32. The value of the regular expression is a language.
- In this case the language consisting of all strings starting with a 0 or a 1 followed by any number of 0s.
- We get this result by dissecting the expression into parts. First the symbol 0 and 1 are shorthand for the sets {0} and {1}.
- So (0 ∪ 1) means ({0} ∪ {1}). The value of this part is the language {0, 1}. The part 0* means {0}* and its value is the language consisting of all strings containing any number of 0s.
- Second, like the x symbol in algebra, the concatenation symbol ∘ often is implicit in regular expressions. Thus (0 ∪ 1)0* actually is shorthand for (0 ∪ 1) ∘ 0*.
- The concatenation attaches the strings from the two parts to obtain the value of the entire expression.
- Regular expressions have an important role in computer science applications. In applications involving text, users may want to search for strings that satisfy certain patterns.
- Utilities like AWK and GREP in UNIX, modern languages like PERL and text editors all provide mechanisms for the description of patterns by using regular expressions.
- Another example of regular expression is (0 ∪ 1)*.
    - It starts with language (0 ∪ 1) and applies the * operation. The value of this expression is the language consisting of all possible strings of 0s and 1s if Σ = {0, 1}, we can write Σ as shorthand for the regular expression (0 ∪ 1).
    - More generally if Σ is any alphabet, the regular expression Σ describes the language consisting of all the strings over this alphabet, and Σ* describes the language consisting of all strings over that alphabet.
    - Similarly Σ*1 is the language that contains all the strings that end in a 1.
    - The language (0Σ*) ∪ (Σ*1) consists of all strings that either start with a 0 or end with a 1.
    - Like arithmetic star operation is done first followed by concatenation and finally union, unless parenthesis are used to change the usual order.
- Regular expressions are closely related to Nondeterministic Finite Automata and can be thought of as a "user – friendly" alternative to the NFA notation for describing software components.

OPERATORS OF REGULAR EXPRESSIONS

- Regular expressions denote languages. For a simple example, the regular expression 01* + 10* denotes the language consisting of all strings that are either a single 0 followed by any number of 1s or a single 1 followed by any number of 0s.
- ***Union*** of two languages L and M, denoted L ∪ M, is the set of strings that are either in L or M or both.
  - For example if L ={001, 10, 111} and M = {ε, 001} then L ∪ M = { ε, 10, 001, 111}.
- ***Concatenation*** of two languages L and M is the set of strings that can be formed by taking any string in L and concatenating it with any string in M.
  - For example if L ={001, 10, 111} and M = {ε, 001} then L . M, or just LM, is {001, 10, 111, 001001, 10001, 111001}.
  - The first three strings in LM are the strings in L concatenated with ε. Since ε is the identity for concatenation, the resulting strings are the same as the strings of L.
  - The last three strings in LM are formed by taking each string in L concatenating it with the second string in M, which is 001. For example 10 is concatenated with 001 result in 10001
- ***Closure*** or ***star*** or ***Kleene closure*** of a language L is denoted L* and represents the set of those strings that can be formed by taking any number of strings from L, possibly with repetitions (i.e. the same string may be selected more than once) and concatenating all of them.
- For example if L ={0, 1}, then L* is all strings of 0's, 1's. if L={0, 11}, then L* consists of those strings of 0's and 1's such that the 1's come in pairs e.g., 011, 11110, and ε, but not 01011 or 101.

FORMAL DEFINITION OF A REGULAR EXPRESSION

- We give the formal recursive definition of regular expressions over Σ as follows: (say R is a regular expression) if R is
  1. *a* for some *a* in the alphabet Σ,
  2. ε,
  3. ϕ,
  4. (R_1 U R_2) where R_1 and R_2 are regular expressions.
  5. ($R_1 \circ R_2$) where R_1 and R_2 are regular expressions (or)
  6. (R_1*) where R_1 is a regular expression.
- In items 1 and 2 the regular expressions *a* and ε represent the language {*a*} and {ε}, respectively.
- In item 3 the regular expression ϕ represents the empty language.
- In items 4, 5 and 6 the expressions represents the language obtained by taking the Union or concatenation of the languages R_1 and R_2, or the star of the language R_1, respectively.

Examples

- In the following instances we assume that the alphabet Σ is {0, 1}.
  - 0*10* = {w | w contains a single 1}.
  - Σ*1Σ* = {w | w has at least one 1}.
  - Σ*001Σ* = {w | w contains the string 001 as a substring}.
  - (01*)* = {w | every 0 in w is followed by at least one 1}.
  - (ΣΣ)* = {w | w is a string of even length}.
  - (ΣΣΣ)* = {w | the length of w is a multiple of three}.
  - 01 ∪ 10 = {01, 10}.
  - 0Σ*0 ∪1Σ*1∪ 0 ∪ 1 = {w | w starts and ends with the same symbol}.
  - $(0 \cup \varepsilon)1^* = 01^* \cup 1^*$.
    The expression 0 ∪ ε describes the language {0, ε}, so the concatenation operation adds either 0 or ε before every string in 1*.
  - $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$.
  - 1*Ø = Ø.
    Concatenating the empty set to any set yields the empty set.
  - Ø*= {ε}.
    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.

REGULAR SET

- Any set represented by the regular expression is called the regular set. If for example let **a, b** ∈ Σ then
  - **a** denotes the set**{a}**
  - **a**∪**b** denotes the set **{a, b}**
  - **ab** denotes the set **{ab}**
  - **a*** denotes the set **{*ε*, a, aa, aaa, …}** and
  - **(a**∪**b)*** denotes the set **{a, b}***.

IDENTITY RULES

- If we let R be any regular expression, we have the following identities.
  - $R \cup \emptyset = R$.
    Adding the empty language to any other language will not change it.
  - $R \circ \varepsilon = R$.
    Joining the empty string to any string will not change it.
- However exchanging Ø and ε in the preceding identities may cause the equalities to fail.

- $R \cup \varepsilon$ may not equal R.
  For example if R =0 then L(R) = {0} but $L(R \cup \varepsilon) = \{\emptyset, \varepsilon\}$.
- $R \circ \emptyset$ may not equal R.
  For example if R = 0 then L(R) = {0} but $L(R \circ \emptyset) = \emptyset$.

- Regular expressions are useful tools in the design of compilers for programming languages.
- Elemental objects in a programming language, called tokens such as the variable names and constants, may be described with regular expressions.
- For example a numerical constant that may include a fractional part and/or a sign may be described as a member of the language

$$(+ \cup - \cup \varepsilon)\ (D^+ \cup D^+.\ D^* \cup D^*.\ D^+)$$

- Where D = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} is the alphabet of decimal digits. Example of generated strings are: 72, 3.14159, +7., and -.01.
- Once the syntax of the tokens of the programming language has be described with regular expressions, automatic systems can generate the lexical analyzer, the part of the compiler that initially processes the input program.
- Some more identity rules are as given below:

I_1: $\phi + R = R$ I_4: $\varepsilon^* = \varepsilon$ and $\phi^* = \varepsilon$ I_7: $RR^* = R^*R$

I_2: $\phi R = R\phi = \phi$ I_5: $R + R = R$ I_8: $(R^*)^* = R^*$

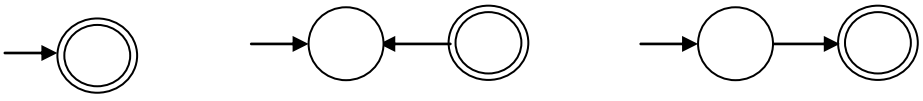
I_3: $\varepsilon R = R\varepsilon = R$ I_6: $R^*R^* = R^*$ I_9: $\varepsilon + RR^* = R^* = \varepsilon + R^*R$

I_{10}: $(PQ)^*P = P(QP)^*$ I_{11}: $(P + Q)^* = (P^*Q^*) = (P^* + Q^*)^*$ I_{12}: $(P + Q)R = PR + QR$ and $R(P + Q) = RP + RQ$

CONSTRUCTING FINITE AUTOMATA FOR A GIVEN REGULAR EXPRESSION

TRANSITION SYSTEM AND REGULAR EXPRESSIONS

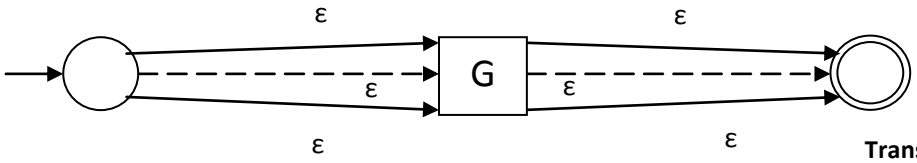
- The following theorem describes the relation between transition systems and regular expressions.
- According to the theorem every regular expression R can be recognized by a transition system, i.e. for every string w in the set R there exists a path from the initial state to a final state with path value w.
- The proof is by the principle of induction on the total number of characters in R. By character we mean elements of Σ, ε, ϕ, * and U. for example if R = ε U 10*11*0, the characters are ε, U, 1, 0, *, 1, 1, *, 0 and the number of characters is 9.
- Let the number of characters in R be 1. Then R = ε or R = ϕ or $R = a_i$, $a_i \in \Sigma$. The transition systems given in figure below will recognise these regular expressions.



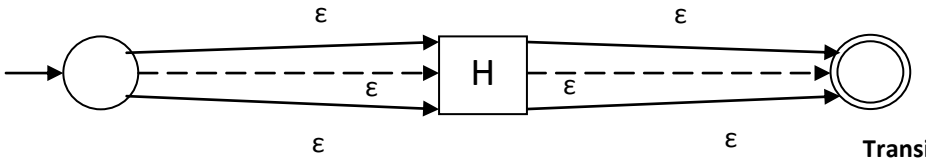
R = ε **R = ϕ** **R = a_i**

- Assume the theorem is true for regular expressions with n characters or less. We must prove that it is also true for n + 1 characters.
- Let R have n + 1 characters. Then,

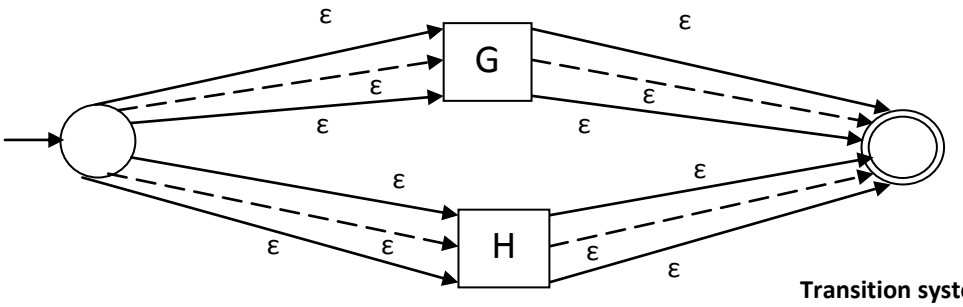
  R = P U Q or R = PQ or R = P*
- Where P and Q are regular expressions, each having n characters or less. By induction hypothesis P and Q can be recognized by transition systems G and H , respectively, as shown in figure as follows:



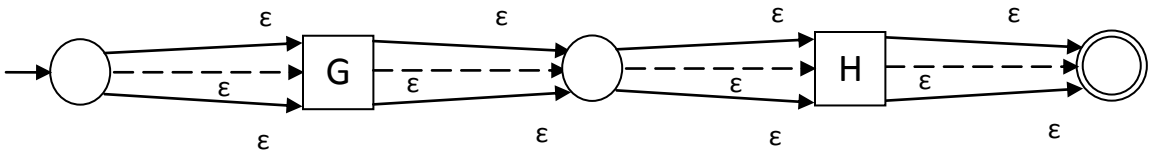
Transition system recognizing P



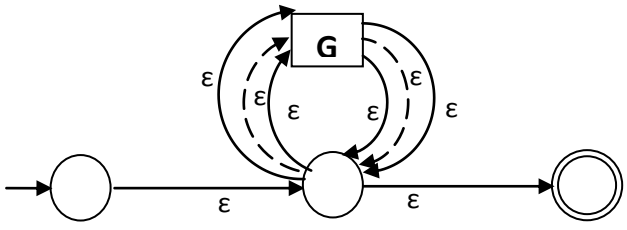
Transition system recognizing Q



Transition system recognizing P U Q



Transition System recognizing PQ



Transition System recognizing P*

- The method we are going to give for constructing a finite automata equivalent to the given regular expression is called the subset method which involves two steps:

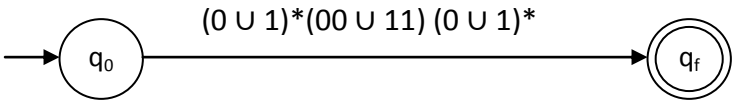
- Construct a transition diagram equivalent to the given regular expression using ε moves.
- Construct the transition table for the transition diagram obtained in step 1. And now construct the equivalent DFA. Also we can reduce number of states if possible.

Example:

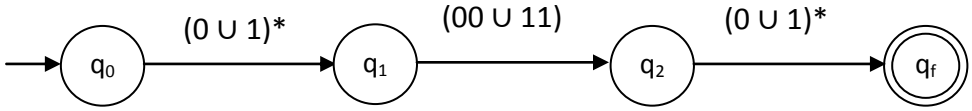
- Construct a finite automata to the regular expression (0 ∪ 1)*(00 ∪ 11)(0 ∪ 1)*

  Step-1 (construction of transition diagram)

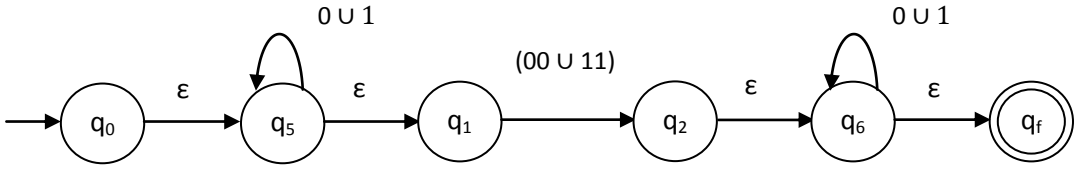
  - First we construct a transition graph with ε moves and then we eliminate ε moves.
  - We start with constructing the automata for the given regular expression as given below:



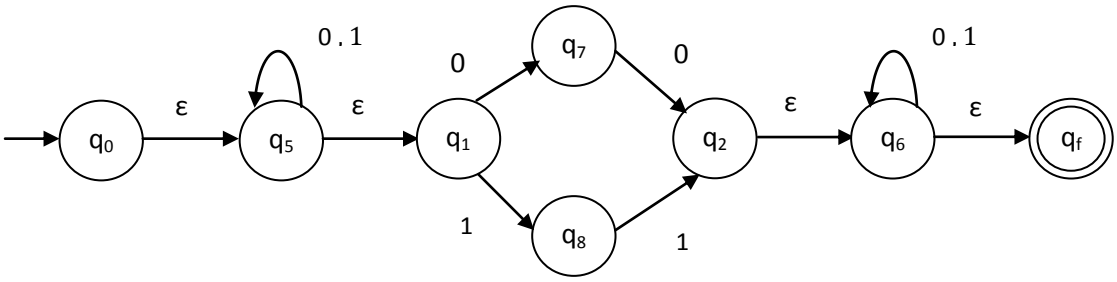
  - Now we eliminate the concatenations in the given regular expression by introducing new states q_1 and q_2 as given below:



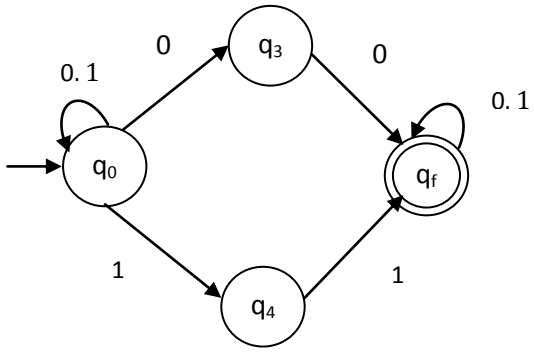
  - Now we eliminate the closure operations (* operations) by introducing the new states q_5 and q_6 and ε moves as given below:



  - Now we eliminate concatenations and union from the above diagram as given below:



  - Now we eliminate the ε moves in above diagram which results in as given below:



Step-2 (construction of DFA)

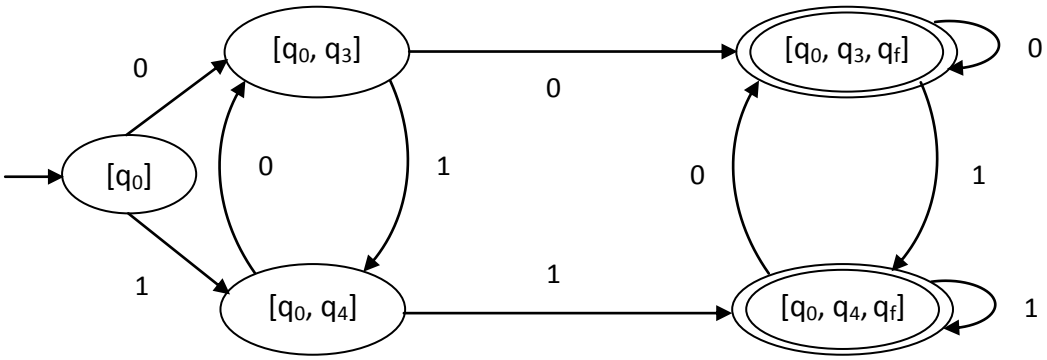
- We now construct the transition table for the NFA as follows:

State	Input	
	0	1
→q_0	q_0, q_3	q_0, q_4
q_3	q_f	ϕ
q_4	ϕ	q_f
*q_f	q_f	q_f

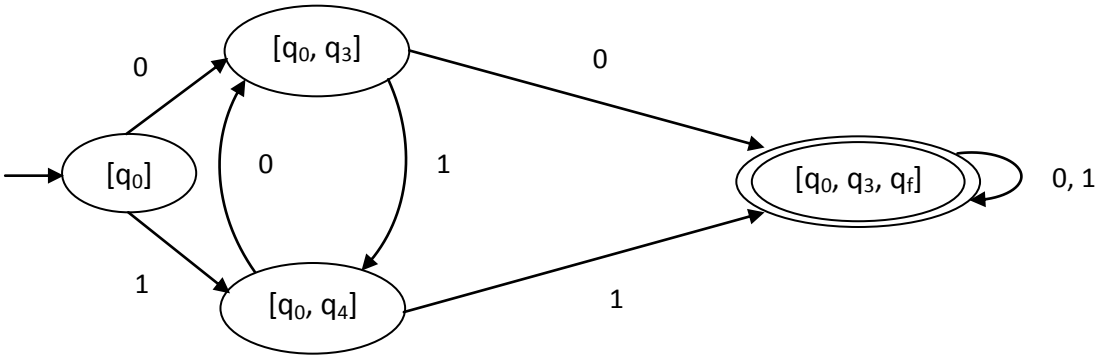
- The *successor table* is constructed and given in the table as shown below:

State	Input	
	0	1
→**[q_0]**	[q_0, q_3]	[q_0, q_4]
[q_0, q_3]	[q_0, q_3, q_f]	[q_0, q_4]
[q_0, q_4]	[q_0, q_3]	[q_0, q_4, q_f]
***[q_0, q_3, q_f]**	[q_0, q_3, q_f]	[q_0, q_4, q_f]
***[q_0, q_4, q_f]**	[q_0, q_3, q_f]	[q_0, q_4, q_f]

- q_f is the only final state for NFA and [q_0, q_3,q_f] and [q_0, q_4,q_f] are the final states for DFA.
- The state diagram for the *successor table* is the required finite automat (DFA) as indicated in the diagram given as follows:



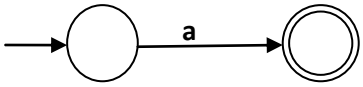
- Finally we try to reduce the number of states. This is possible when two states rows are identical in the *successor table*.
- As the rows corresponding to [q_0, q_3, q_f] and [q_0, q_4, q_f] are identical, the state diagram with reduced number of states for the equivalent automata is given as below:



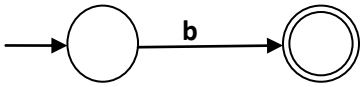
Example

- Construct the nondeterministic finite automata for the regular expression (ab U a)*

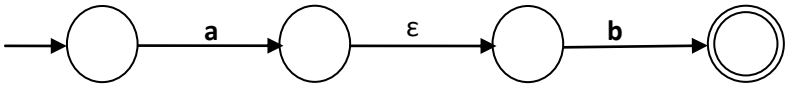
First construct transition diagram for **a**.



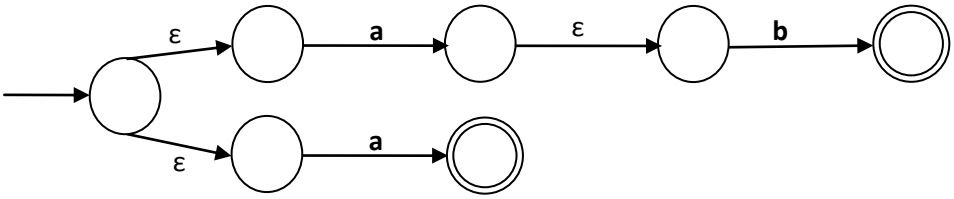
Then construct the transition diagram for **b**.



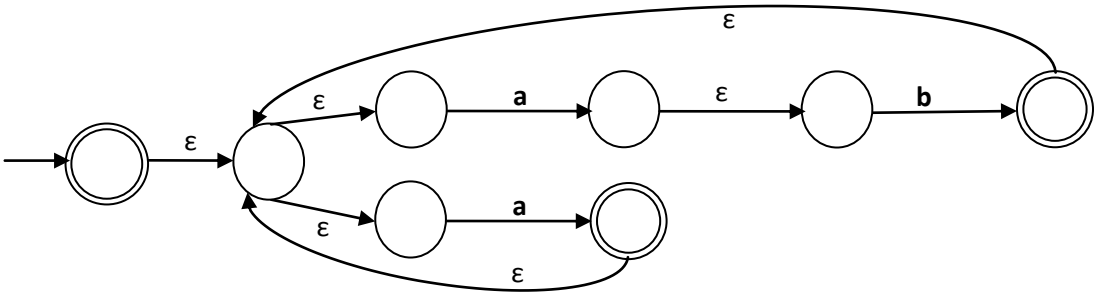
Now construct for **ab**



Now construct for **ab U a**



Now construct for **(ab U a)***



ARDEN'S THEOREM:

- Let **P** and **Q** be two regular expressions over an alphabet Σ. If **P** does not contain **ε** then the following equation in **R**, viz.

$$R = Q + RP \qquad (3.1)$$

- Has a unique solution (i.e. one and only one solution) given by **R** = **QP***.

Proof:

Given the equation R = Q + RP

Now apply R = QP* in R = Q + RP, we get

➔R = Q + (QP*)P

➔R = Q(ε + P*P)

But as per the identity rule I_9: ε + R*R = R* we get

➔R = QP*

Hence is satisfied when **R** = **QP*** . This means **R** = **QP*** is a solution of **R** = **Q** + **RP**.

To prove the uniqueness, consider the following:

➔ Q + RP = Q + (Q +RP)P

➔ Q + RP = Q + QP + RPP

➔ Q + RP = Q + QP + RP^2

➔ Q + RP = Q + QP + $(Q + RP)P^2$

➔ Q + RP = Q + QP + QP^2 + … + QP^i + RP^{i+1}

➔ Q + RP = Q(ε + P + P^2 + … + P^i)+ RP^{i+1}

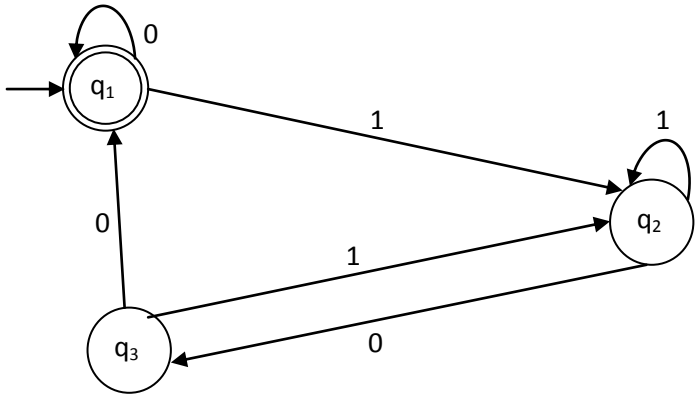
From (3.1),

$$R = Q(\varepsilon + P + P^2 + \ldots + P^i) + RP^{i+1} \text{ for } i \geq 0 \qquad \textbf{(3.2)}$$

- We now show that any solution of (3.1) is equivalent to QP*. Suppose R satisfies (3.1) then it satisfies (3.2).
- Let w be a string of length **i** in the set R. Then w belongs to the set Q(ε + P + P^2 + … + P^i)+ RP^{i+1}. As P does not contain ε, **RP^{i+1}** have no string of length less than i+1 and so w is not in the set **RP^{i+1}**. This means w belongs to the set Q(ε + P + P^2 + … + P^i), hence to **QP***.
- Consider a string w in the set **QP***. Then w is in the set QP^k for some k ≥ 0 and hence in Q(ε + P + P^2 + … + P^k). So w is on the R.H.S of (3.2). Therefore w is in R (L.H.S of (3.1)).
- Thus R and QP* represent the same set. This proves the uniqueness of the solution (3.1).

CONVERSION OF FINITE AUTOMATA TO REGULAR EXPRESSION

- Construct the regular expression corresponding to the state diagram given as follows:



Solution

- In the given transition diagram, there is only one initial state. Also there is no ε – moves.
- The equations are:

$q_1 = q_1 0 + q_3 0 + \varepsilon$, $q_2 = q_1 1 + q_2 1 + q_3 1$, $q_3 = q_2 0$

So

$$q_2 = q_1 1 + q_2 1 + (q_2 0)1 = q_1 1 + q_2(1 + 01)$$

It is of the form R = Q + RP so we get the following:

$$q_2 = q_1 1(1 + 01)^*$$

Also,

$$q_1 = q_1 0 + q_3 0 + \varepsilon = q_1 0 + q_2 00 + \varepsilon$$

$$q_1 = q_1 0 + (q_1 1(1 + 01)^*)00 + \varepsilon$$

$$q_1 = q_1(0 + 1(1 + 01)^*)00) + \varepsilon$$

Now by applying the Arden's theorem we get

$$q_1 = \varepsilon(0 + 1(1 + 01)^*00)^* = (0 + 1(1 + 01)^*00)^*$$

As q_1 is the only final state, the regular expression corresponding to the given diagram is

$$(0 + 1(1 + 01)^*00)^*$$

PUMPING LEMMA FOR REGULAR SETS

NONREGULAR LANGUAGES

- To understand the power of finite automata one must understand their limitations. Here we can show how to prove that certain languages cannot be recognized by any finite automata.

- Let's take a language B = $\{0^n1^n \mid n \geq 0\}$. If we attempt to find a DFA that recognizes B, we discover that the machine seems to need to remember how many 0s have been seen so far as it reads the input.
- Because the number of 0s is not limited, the machine will have to keep track of an unlimited number of possibilities. But it cannot do so with any finite number of states.
- Next we present a method for proving that languages such as B are not regular. Doesn't the argument already given prove nonregularity because the number of 0s is unlimited? It does not. Just because the language appears to require unbounded memory doesn't mean that it is necessarily so.
- It does happen to be true for the language B, but other languages seem to require an unlimited number of possibilities, yet actually they are regular.
- For example consider two languages over the alphabet Σ = {0, 1}:
  C = {w | w has an equal number of 0s and 1s}, and
  D = {w | w has an equal number of occurrences of 01 and 10 as substrings}.
- As expected C is not regular, but surprisingly D is regular. Thus our intuition can sometimes lead us astray, which is why we need mathematical proofs for certainty.

PUMPING LEMMA FOR REGULAR LANGUAGES

- Our technique for proving nonregularity stems from a theorem about regular languages, traditionally called the ***pumping lemma***. This theorem states that all the regular languages have a special property. If we can show that a language does not have this property, we are guaranteed that it is not regular.
- This property states that all strings in the language can be "pumped" if they are at least as long as a certain special value called the ***pumping length***. That means each such string contains a section that can be repeated any number of times with the resulting string remaining in the language.

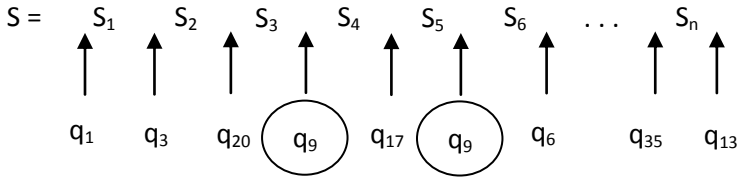
PUMPING LEMMA

- If A is a regular language then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, s = xyz, satisfying the following conditions:
  - For each $i \geq 0$, $xy^iz \in A$,
  - $|y| > 0$, and
  - $|xy| \leq p$.
- The notation where |s| represents the length of the string s, y^i means that i copies of y are concatenated together and y^0 equal ε.
- When s is divided into xyz, either x or z may be ε, but condition 2 says that $y \neq \varepsilon$. Observe that without condition 2 the theorem would be trivially true.

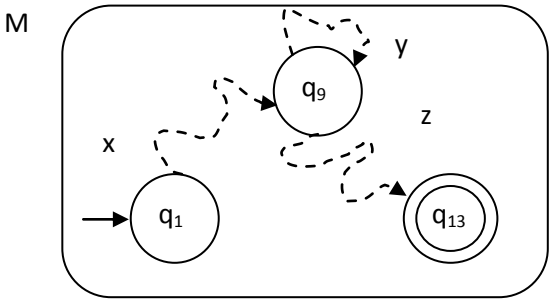
- Condition 3 states that the pieces x and y together have length at most p.

PROOF IDEA:

- Let M = {Q, Σ, δ, q_1, F} be a DFA that recognizes A. We assign the pumping length p to be the number of states of M.
- Now we show that s in A of length at least p may be broken into three pieces xyz satisfying our three conditions.
- If s in A has length at least p, consider the sequence of states that M goes through when computing with input s. It starts with q_1 the start state, then goes to say q_3, then, say q_{20}, then q_9 and so on, until it reaches the end of s in state q_{13}. With s in A, we know that M accepts s, so q_{13} is an accept state.
- If we let n be the length of s, the sequence of states q_1, q_3, q_{20}, q_9, … , q_{13} has length n + 1. Because n is at least p, we know that n + 1 is greater than p, the number of states of M.
- Therefore the sequence must contain a repeated state. This result is an example of the pigeonhole principle. According to this principle, if p pigeons are place into fewer than p holes, some hole has to have more than one pigeon in it.
- The following figure shows the string s and the sequence of states that M goes through when processing s. State q_9 is the one that repeats.



- We now divide s into the three pieces x, y and z. Piece x is the part of s appearing before q_9, piece y is the part between the two appearances of q_9 and piece z is the remaining part of s, coming after the second occurrence of q_9.
- So x takes M from the state q_1 to q_9, y takes M from q_9 back to q_9 and z takes M from q_9 to the accepting state q_{13} as shown in the figure below:



CLOSURE PROPERTIES OF REGULAR SETS

- Let A and B be the languages. We define the regular operations Union, concatenation, and start as follows.
  - Union: **A U B = {w | w Є A or w Є B}.**
  - Concatenation: **AB = {xy| x Є A and y Є B}.**
  - Star: **$A^* = \{w_1 w_2 \ldots w_k \mid k \geq 0$ and each w_i Є A}.**
- Let the alphabet Σ be the standard 26 letters {a, b, . . ., z}. if A = {good, bad} and B = {boy, girl} then
  - A U B = {good, bad, boy, girl}
  - AB = {goodboy, goodgirl, badboy, badgirl}
  - A* = {ε, good, bad, goodgood, goodbad, badgood, badbad, goodgoodgod, goodgoodbad, goodbadbad, . . . }
- Let N = {1, 2, 3, . . . } be the set of natural numbers. When we say that N is closed under multiplication we mean that for any ***r*** and ***s*** in N the product ***r*** X ***s*** also is in N. In contrast N is not closed under division, as 1 and 2 are in N but 1/2 is not.
- Generally speaking, a collection of objects is closed under some operation if applying that operation to members of the collection returns an object still in the collection.
- We now show that the collection of regular languages is closed under all three of the regular operations.

➢ ***The class of regular languages is closed under the union operation***

➢ In other words if A_1 and A_2 are regular languages, so A_1 U A_2.

Proof Idea:

➢ We have regular languages A_1 and A_2 and want to show that A_1 U A_2 also is regular. Because A_1 and A_2 are regular, we know that some finite automaton M_1 recognizes A_1 and some finite automaton M_2 recognizes A_2.

➢ To prove that A_1 U A_2 is regular we demonstrate a finite automaton call it M that recognized A_1 U A_2. This is a proof by construction. We construct M from M_1 and M_2. Machine M must accept its input exactly when either M_1 or M_2 would accept it in order to recognize the union language.

PROOF

➢ Let M_1 recognizes A_1, where $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and M_2 recognizes A_2, where $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.

➢ Construct M to recognize A_1 U A_2 where $M = (Q, \Sigma, \delta, q_0, F)$.

- $Q = \{(r_1, r_2) \mid r_1 \in Q_1$ and $r_2 \in Q_2\}$. This is the Cartesian product of sets Q_1 and Q_2 and is written as Q_1 X Q_2. It is the set of all Pairs of states, the first from Q_1 and the second from Q_2.

- Σ, the alphabet, is the same in M_1 and M_2. For simplicity we assume that both M_1 and M_2 have the same input alphabet Σ. If they have different alphabets, Σ_1 and Σ_2, we would then modify the proof to let $\Sigma = \Sigma_1 \cup \Sigma_2$.
- δ, the transition function, is defined as follows. For each $(r_1, r_2) \in Q$ and $a \in \Sigma$, let

$$\delta((r_1, r_2), a) = \delta_1(r_1,a), \delta_2(r_2,a)$$

  Hence δ gets a state of M (which actually is a pair of states from M_1 and M_2), together with an input symbol, and returns M's next state.
- q0 is the pair (q_1, q_2)
- F is the set of pairs in which either member is an accept state of M_1 or M_2. We can write it as:

$$F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$$

- This expression is same as $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$. This concludes the construction of finite automaton M that recognizes the union of A_1 and A_2.
-

➢ ***The class of regular languages is closed under the Concatenation operation***

➢ In other words, if A_1 and A_2 are regular languages then so is A_1A_2.

Proof Idea:

➢ We have regular languages A_1 and A_2 and want to show that A_1A_2 also is regular. Because A_1 and A_2 are regular, we know that some finite automaton M_1 recognizes A_1 and some finite automaton M_2 recognizes A_2.

➢ Construct the automaton M such that it must accept if its input can be broken into two pieces, where M_1 accepts the first piece and M_2 accepts the second piece. But the problem is that M doesn't know where to break its input i.e. where the first part ends and the second begin.

➢ To solve this problem we use the new technique called nondeterminism

Nondeterminism

➢ It is a generalization of determinism, so every deterministic finite automaton is automatically a nondeterministic finite automaton.

➢ Suppose that we are running an NFA on an input string and come to a state with multiple ways to proceed. For example, say that we are in state q_1 in some NFA N_1 and that the next input symbol is a 1. After reading that symbol, the machine splits into multiple copies of it and follows all the possibilities in parallel.

➢ Each copy of the machine takes one of the possible ways to proceed and continues as before. If there are subsequent choices, the machine splits again.

- If a state with a ε symbol on an exiting arrow is encountered, something similar happens. Without reading any input, the machine splits into multiple copies, one following each of the exiting ε-labeled arrows and one staying at the current state. Then the machine proceeds nondeterministically as before.

SOLVE THE FOLLOWING (FOR YOUR PRACTICE)

- *$M = (\{q_1,q_2,q_3\}, \{0, 1\}, \delta, q_1, \{q_3\})$ is a nondeterministic finite automaton, where δ is given by*
  $\delta(q_1, 0) = \{q_2, q_3\}$ $\delta(q_1, 1) = \{q_1\}$
  $\delta(q_2, 0) = \{q_1, q_2\}$ $\delta(q_2, 1) = \varphi$
  $\delta(q_3, 0) = \{q_2\}$ $\delta(q_3, 1) = \{q_1, q_2\}$ Construct an equivalent DFA?
- *Construct an NFA accepting the set of all the strings over {a, b} ending in aba. Use it to construct DFA accepting the same set of all the strings.*
- Construct the DFA with reduced states equivalent to the regular expression **10 + (0 + 11)0*1**.
- Prove that **(a + b)* = a*(ba*)***.
- Construct the transition system corresponding to the regular expression 1) **(ab + c*)*** and 2) **a +bb + bab*a**.
- Find the regular expression representing the following sets:
  - The set of all strings over {0, 1} having atmost one pair of 0's or atmost one pair of 1's.
  - The set of all strings over {*a*, *b*} in which the number of occurrences of *a* is divisible by 3.