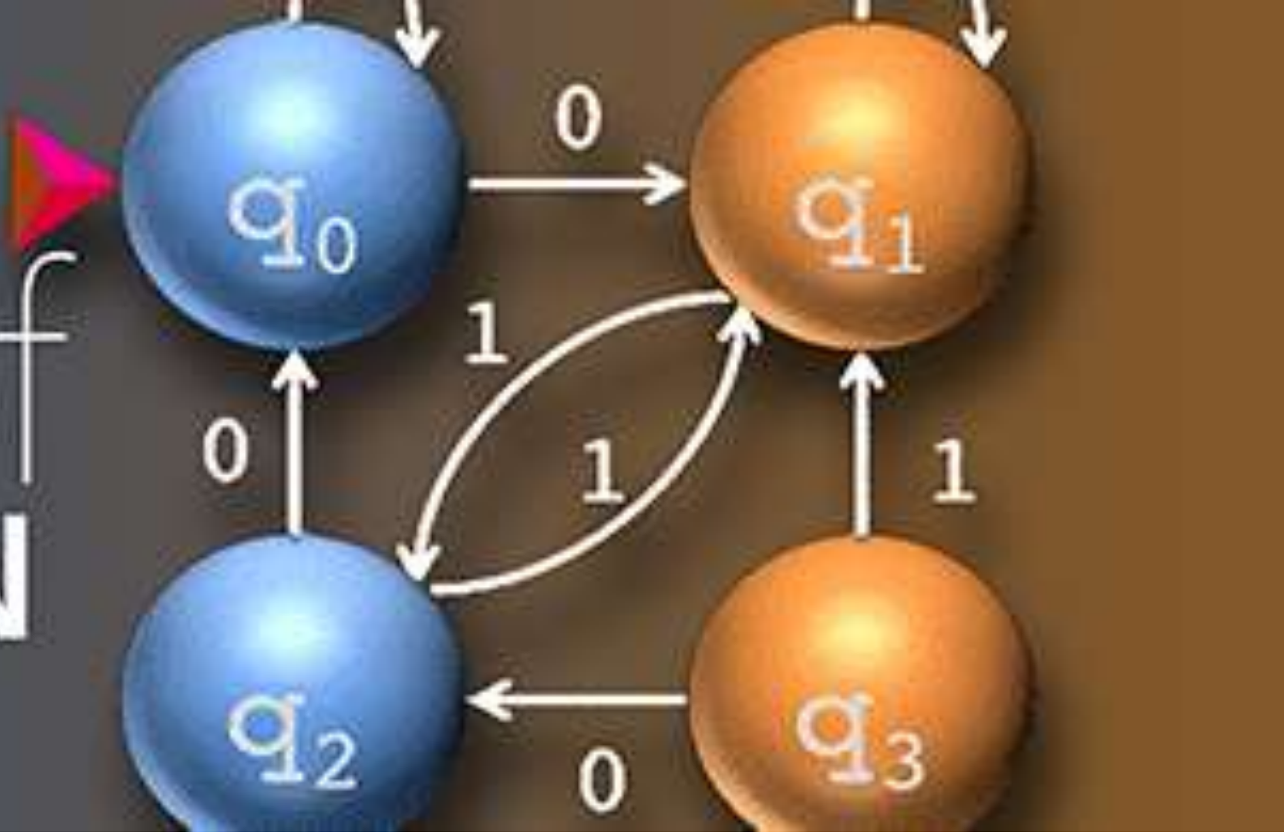
CSE 305

Theory of COMPUTATION



Lecture 22

Regular Expressions (3)

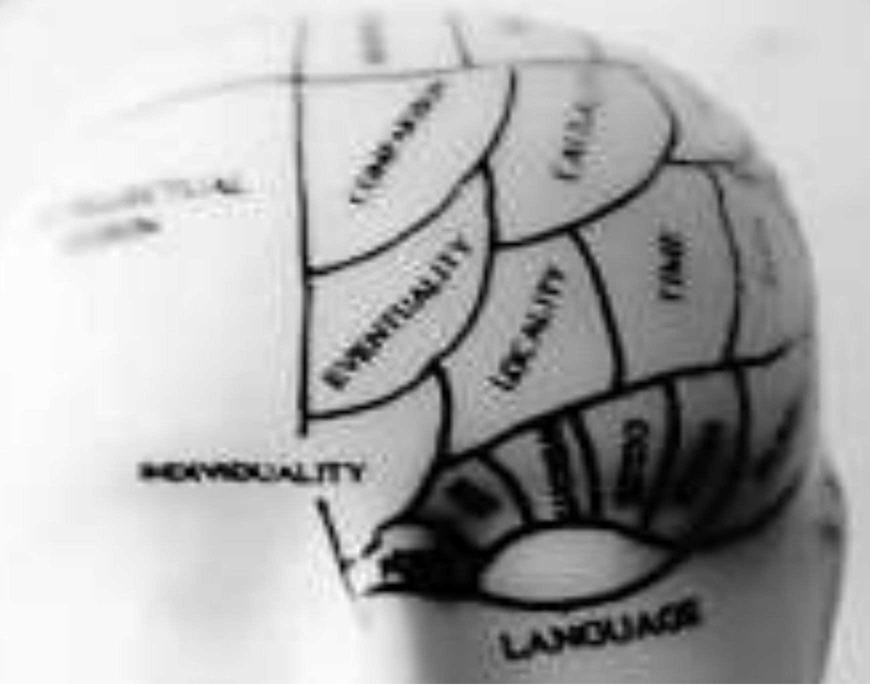
Md. Mijanur Rahman, Prof. Dr.

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

www.mijanrahman.com

Contents

Regular Expressions



- Regular Expressions
- Regular Languages
- Operation on Regular Languages
- Extensions of Regular Expressions
- Regular Sets and Properties of Regular Sets
- Identities Related to Regular Expressions
- Examples: Regular Expressions
- Conversion of Regular Expressions into Finite Automata
- Conversion of Finite Automata into Regular Expressions
- **Pumping Lemma for Regular Languages**
- **Closure Properties of Regular Languages**
- **Relationship with other Computation Models**

Pumping Lemma for Regular Languages

- We know that regular languages and finite automata are closely related; however, finite automata have a finite memory (number of states), **while regular languages can be infinite.**
- **We are now presenting a theorem to determine whether a given language is regular or not.**
- **This theorem uses the pigeonhole principle and is known as 'pumping lemma for regular languages'**, because some part of the string is pumped to the same state.

Pumping Lemma for Regular Languages

- We know that the language accepted by the finite automata is called **Regular Language**.
- If we are given a language **L** and asked whether it is regular or not? So, to prove a given Language L is not regular we use a method called **Pumping Lemma.**
- The term **Pumping Lemma** is made up of two words:
  - **Pumping:** The word pumping refers to generate many input strings by pushing a symbol in an input string again and again.
  - **Lemma:** The word Lemma refers to intermediate theorem in a proof.

Pumping Lemma for Regular Languages

- **Pumping Lemma** is used to prove that given language is not regular. So, first of all we need to know when a language is called regular. A language is called regular if:
  - Language is accepted by finite automata.
  - A regular grammar can be constructed to exactly generate the strings in a language.
  - A regular expression can be constructed to exactly generate the strings in a language.

Principle of Pumping Lemma:

- **The pumping lemma states that all the regular languages have some special properties. If we can prove that the given language does not have those properties, then we can say that it is not a regular language.**

Theorem: Pumping Lemma for Regular Languages

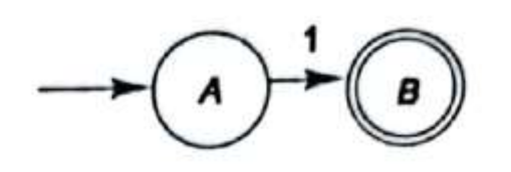
- **Theorem:**
  - If L is an infinite regular language then there exists some positive integer n (pumping length) such that any string $w \in L$ has length greater than or equal to n. i.e. $|\mathbf{w}| >= \mathbf{n}$, then string can be divided into three parts, w=xyz satisfying the following condition:
    - $|y| > 0$
    - $|xy| <= n$
    - For all $i \geq 0$, the string xy^iz is also in L.
  - $|\mathbf{w}|$ represents the length of string w and $\mathbf{y^i}$ means that i copies of y are concatenated together, $\mathbf{y^0 = \varepsilon}$.
  - **In simple terms, this means that if a string y is 'pumped', i.e., if y is inserted any number of times, the resultant string still remains in L.**

Theorem: Pumping Lemma for Regular Languages

Proof:

- If L is a regular language, then there must be a corresponding DFA that recognizes it Let us assume that DFA $M = (Q, \Sigma, \delta, q_0, F\}$, where Q is a finite sequence of states, that is, $Q = \{q_o, q_1, q_2, \ldots, q_n\}$.
- Let us consider a string x (over the set of alphabets, Σ) of length more than n. if we traverse x starting from state q_o (starting state) and if it is accepted by the finite automata, then the traversal will end at some final state.
- Let us assume that the sequence of states is given by $q_o\ q_1\ q_i\ q_j\ q_f$. However, this sequence will contain |w|+1 states as in order to accept one character input we require two states if loops are not allowed.
- For example, to accept 1 we need one initial state A and one final state *B* as shown below.

Fig.1: **FA to accept an input symbol 1.**



Theorem: Pumping Lemma for Regular Languages

Proof:

- Thus, to accept |w| characters we need |w|+1 states if loops are not included. However, we have only |w| states. Therefore, according to the pigeonhole principle, at least one state must be repeated. The sequence of states may thus be assumed to be the following (say w = xyz):

$$\underbrace{q_0q_iq_j\cdots\cdots q_m}\underbrace{\cdots\cdots\ \cdots q_m}\underbrace{\cdots\cdots\cdots\cdots q_f}$$

That is $\quad (q_0, x) = q_m$

$$\delta^*(q_m, y) = q_m$$

$$\delta^*(q_m, z) = q_f$$

- **Such a sequence can be represented as the finite automata given below.**

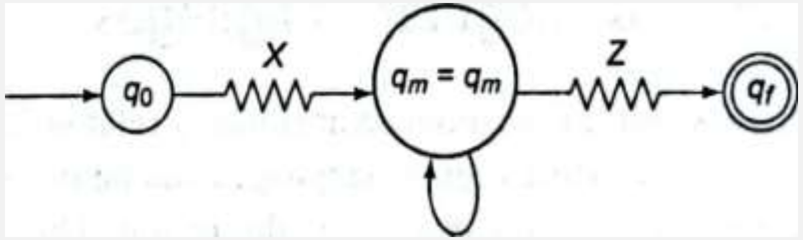
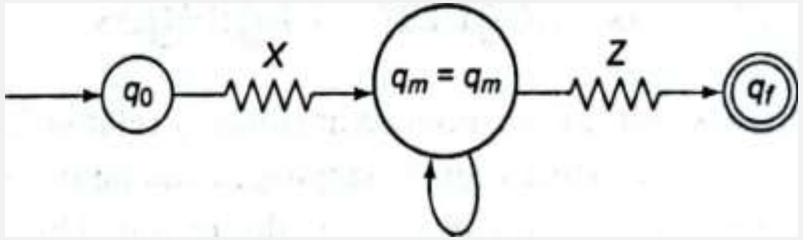


Fig.2: **Finite automata to accept a sequence of characters.**

Theorem: Pumping Lemma…



Proof:

- In the finite automata shown in Fig.2, we start from the initial state q_o and reach the state q_m after reading x. We remain in the same state till we read y and then on encountering z, we reach the state q_f, which is the final state.

- In this manner, the string is accepted. Here, we should note that the length of |y| should be equal to or more than 1, so that the loop is used. Therefore, every string of a language can be divided into three substrings x, y, and z, such that ly| = 1 and |xy| = n.

- From these observations, we can conclude that:

$$\delta^*(q_0, xyz) = \delta(\delta\,(q_0, x), yz) = \delta(q_m, yz) = \delta(\delta\,(q_m, y), z) = \delta(q_m, z) = q_f$$

- Similarly, we can say that:

$$\delta^*(q_0, xy^iz) = \delta(\delta\,(q_0, \overset{x}{\cancel{\delta}}), y^iz) = \delta(q_m, y^iz) = \delta(\delta\,(q_m, y^i), z) = \delta(q_m, z) = q_f$$

- Hence, if any string w is contained in the language L then it can be divided into three substrings, namely, x, y, and z such that w = xyz and xy^iz is also present in L for $i \geq 0$.

Applying Pumping Lemma

- We will use above theorem to prove that given language is not regular. The steps needed to prove that given languages is not regular are given below:
  - **Step1:** Assume L is a regular language in order to obtain a contradiction. Let n be the number of states of corresponding finite automata.
  - **Step2:** Now chose a string w in L that has length n or greater; i.e. $|w| >= n$. Use pumping lemma to write:

    $w = xyz$ with $|xy| <= n$ and $|y| = 0$.
  - **Step3:** Finally demonstrate that we cannot pumped by considering all ways of dividing w into x, y and z, and for each such division find a value of I such that $xy^iz \in L$. This contradicts our assumption; hence L is not regular.
- We prove $xy^iz \in L$ by considering the length of xyz i.e. $|xy^iz|$ or by using the structure of strings in L.

Example: Applying Pumping Lemma

- **Example**

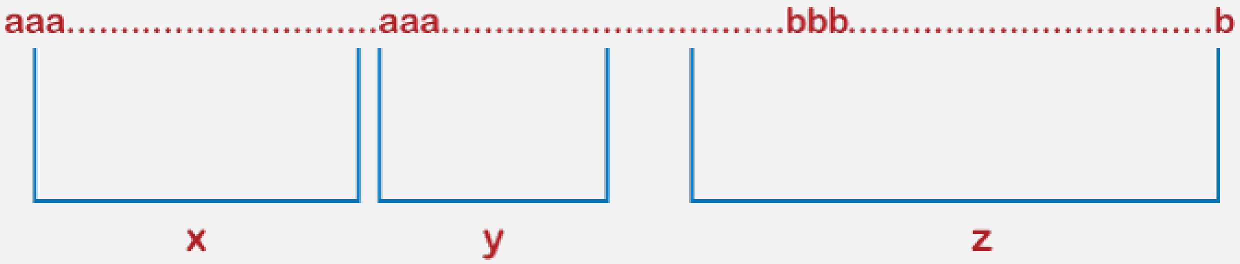
  Let L= { $a^n b^n$ | n ≥ 0 }. By using pumping lemma show that L is not regular language.

- **Solution:**
  - **Step1:** Assume L is a regular language in order to obtain contradiction. Let n be the number of states in finite automata accepting L.
  - **Step2:** Let w = $a^n b^n$, then |w| = 2n > n. Using pumping lemma, we can demonstrate w in three parts of xyz such that w = xyz with |xy| ≤ n and |y| > 0.
  - **Step3:** Now we want to find some *i*, for which $xy^iz \notin L$. This means, a proof of contradiction.

  - **There are three possibilities for y, we will consider all cases one by one and show that given language contains some string not for { $a^n b^n$ | n ≥ 0 }.**

Example: Applying Pumping Lemma

- **Case 1:** The string y consists of only a's i.e. $y = a^k$ $(k \geq 1)$.



We have $w = xyz$

$w = a^n b^n$

- In given language we have equal numbers of a's and b's in $w \in L$; so, it must satisfy this condition. Let us take $i = 0$.

As $xyz = a^n b^n$

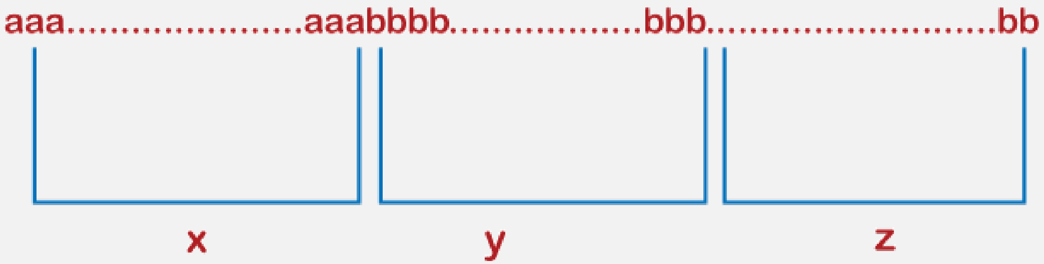
$xz = a^{n-k} b^n$

$n-k \neq n$

- So $xz \notin L$. This case is a contradiction.

Example: Applying Pumping Lemma

- **Case 2:** The string y consists of only b's i.e. $y = b^m$ $(m \geq 1)$.



We have $w = xyz$

$w = a^n b^n$

- In given language, we have equal number of a's and b's in $w \in L$, so it must satisfy this condition. Let us take i=0.

As $xyz = a^n b^n$

$xz = a^n b^{n-m}$

Where $n \neq n-m$

- So $xz \notin L$. This case also gives contradiction.

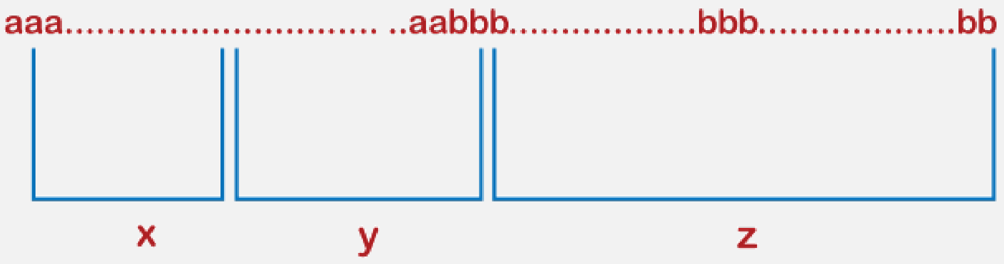
Example: Applying Pumping Lemma

- **Case 3:** The string y consists of both a's and b's i.e. $y = a^k b^m$ $(k \geq 1; m \geq 1)$.

We have w = xyz

$w = a^n b^n$

$w = a^{n-k}\, a^k b^m\, b^{n-m}$



- In given language we have equal number of a's and b's; so it must satisfy this condition. Let us take i=2.

$xy^2z = xyyz$

$= a^{n-k}\, a^k b^m\, a^k b^m\, b^{n-m}$ here, $x = a^{n-k}$; $y = a^k b^m$; $z = b^{n-m}$.

- In this case, the string xyyz must have equal number of a's and b's; **but they are out of order with some b's before a's**. **Hence it is not a member of L, which contradicts our assumption.**
- Thus, in all cases we get a contradiction. Therefore, **L is not regular.**

Closure Properties of Regular Languages

- **A set is closed under an operation if applying that operation to any members of the set always yields a member of the set.** For example, the positive integers are closed under addition and multiplication, but not division.
- Thus, we use the term "**Closure**" when we talk about **sets of things**. If we have two regular languages L1 and L2, and L is obtained by applying certain operations on L1, L2 then L is also regular.
- **Closure Properties used in Regular Languages are as follows:**
  - **Union**
  - **Concatenation**
  - **Complementation**
  - **Intersection**
  - **Reversal**
  - **Difference**
  - **Homomorphism**
  - **Inverse Homomorphism**

Closure Properties of Regular Languages

- **Kleene Closure**

Let R is regular expression whose language is L. Now apply the Kleene closure on given regular expression and language. So, R* is a regular expression whose language will become L*.

Example:

Suppose R= (a) then its language will be L= {a}. Now apply Kleene Closure on given regular expression and language,

if R* = (a)* then its language will be L* = {e, a, aa, aaa, aaaa….}

So, L* is still a regular language. Thus Kleene clouser is satisfied.

Closure Properties of Regular Languages

- **Positive Closure**

R is a regular expression whose language is L.$R^+$is a regular expression whose language is L^+

Example:

Suppose R= (a) then its language will be L= {a} **.** Now apply positive Closure on given regular expression and language,

if $R^+ = (a)^+$ then its language will be $L^+ = \{a, aa, aaa, aaaa….\}$

So, L^+ is still a regular language. Thus positive clouser is satisfied.

Closure Properties of Regular Languages

- **Complement**

The complement of a language L is (Σ* – L). Where sigma (Σ) holds the input symbols use for generating the language. So, complement of a regular language is always regular.

- **Union**

Let L1 and L2 be the languages of regular expressions R1 and R2, respectively. Then R1+R2 (R1 U R2) is ALSO a regular expression whose language is L3 = (R1 U R2). **L3 also belongs to regular language**

- **Concatenation**

Let L1 and L2 be the languages of regular expressions R1 and R2, respectively. Then R1.R2 is ALSO a regular expression whose language is L3 = (R1.R2). **L3 also belongs to regular language**

Closure Properties of Regular Languages

- **Intersection**

Let L1 and L2 be the languages of regular expressions R1 and R2, respectively then the it a regular expression whose language is L1 intersection L2.

- **Set Difference Operator**

Let L1 and L2 be the languages of regular expressions R1 and R2, respectively then it a regular expression whose language is L1 – L2.= strings in L1 but not L2.

- **Reverse Operator**

Given language L, $L^R$is the set of strings whose reversal is in L.
Example: L = {0, 01, 100};
L^R ={0, 10, 001}.
L^R still a regular language

Closure Properties of Regular Languages

- **Homomorphism**

It is use to substitute (replace) the value of sigma with given delta values. Delta is nothing but a symbol. Delta contains some values which are used to replace the sigma values.

Suppose "H" is delta then we can say

$H(L) = \{H(w) \mid w \in L\}$

Example:

If $L = \{00, 101\}$ and $H(0)$ = "aa" and $H(1)$ = "bb" then after substitution $H(L)$ is given below

$H(L) = \{aaaa, bbaabb\}$

Closure Properties of Regular Languages

- **Inverse Homomorphism**

It is the also a substitution technique like homomorphism but functionality of substitution is opposite. It replace the value of Delta with sigma values. It is denoted by power of -1 i.e (H^{-1})

Example:

Suppose “H” is delta then we can say

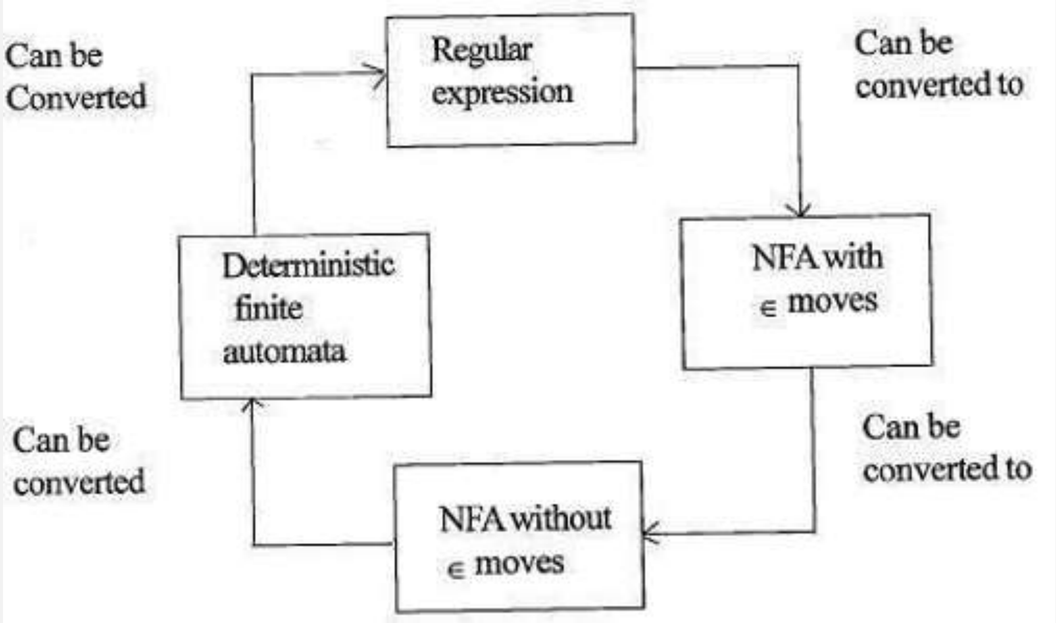
If L = {aabb} and H(0) = “aa” and H(1) = “bb” then after substitution $H^{-1}(L)$ is given below

$H^{-1}(L) = \{01\}$

Note: “aa” is replace with “0” and “bb” is replace with “1” in given language.

Relationship between FA and RE

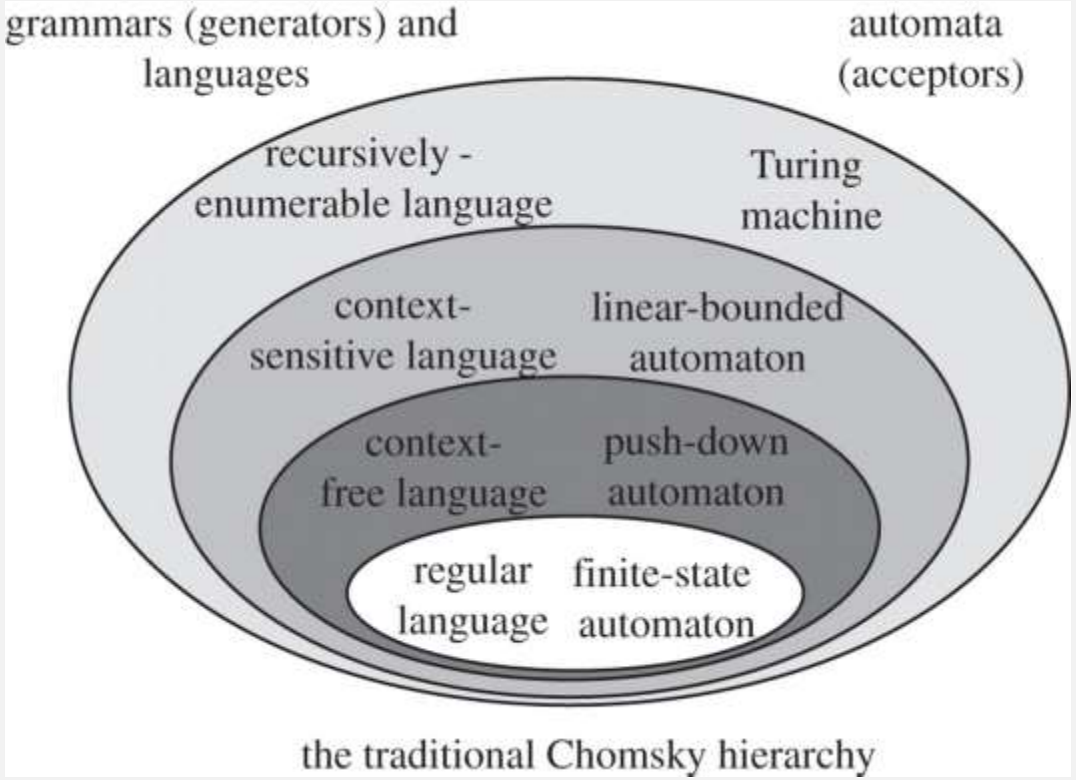
- The relationship between FA and RE is as follows −



- The above figure explains that it is easy to convert
  - RE to Non-deterministic finite automata (NFA) with epsilon moves.
  - NFA with epsilon moves to without epsilon moves.
  - NFA without epsilon moves to Deterministic Finite Automata (DFA).
  - DFA can be converted easily to RE.

Relationship with other Computation Models

- Regular languages are less powerful than context-free languages which are generated by context-free grammars and pushdown automata, and recursively enumerable languages which are generated by Turing machines.



THE END