INDEX

Regular Expressions :

Regular expressions are simple algebraic expressions that are used to represent a set of strings.

Recursive definition of regular expression:

For given input alphabet ∑, regular expression is defined by following rules:

1. Every character belonging to ∑, is a regular expression.
2. Null string ε is a regular expression.
3. If R_1 and R_2 are two regular expressions, then $R_1+ R_2$ is also a regular expression.
4. If R_1 and R_2 are two regular expressions, then $R_1. R_2$ is also a regular expression.
5. If R is a regular expression, then (R) is also a regular expression.
6. If R is a regular expression, then R^* is also a regular expression.
7. Any combination of above rules is also a regular expression.

Regular Set or Regular Language :

Set of strings generated by regular expressions is known as regular set (or) regular language. It is the language accepted by Finite Automata.

Recursive definition of regular language from regular expression:

For given input alphabet ∑, regular expression is defined by following rules:

1. If Null string ε is a regular expression, then the regular language is L={ε} which contains the empty string.
2. For every input symbol $a \in \Sigma$, a is a regular expression, then the regular set is L={a}.
3. Let R_1 = a and R_2 = b and $R_1+ R_2$ is a regular expression, then regular expression is a+b (means a or b) and regular set is L={a,b}.
4. Let R_1 = (a+b) and R_2 = c and $R_1. R_2$ is a regular expression, then regular expression is (a+b).c (means starting with a or b followed by c) and regular set is L={ac,bc}.
5. Let R={a} and R^* is a regular expression (means any number of repetitions of a from 0 to ∞) then regular set is L={ ε,a,aa,aaa...}.
6. If ϕ is a regular expression, then regular language is L={}= ϕ which denotes the empty set.

Properties of Regular Expressions:

If R_1 and R_2 are two regular expressions, then language generated by R_1 is $L(R_1)$ and language generated by R_2 is $L(R_2)$. The properties of regular expressions are

1. **Union:** It is a string from $L(R_1)$ or $L(R_2)$. It is denoted by $L(R_1+ R_2) = L(R_1) \cup L(R_2)$.
   Example: let $R_1 = ab+c \Longrightarrow L(R_1) =\{ab,c\}$
   $R_2 = d+ef \Longrightarrow L(R_2) =\{d,ef\}$
   Then $L(R_1+ R_2) = L(R_1) \cup L(R_2) = \{ab,c\} \cup \{d,ef\} = \{ab,c,d,ef\}$
2. **Concatenation:** It is a string from $L(R_1)$ followed by $L(R_2)$. It is denoted by $L(R_1. R_2) = L(R_1) . L(R_2)$.
   Example: let $R_1 = ab+c \Longrightarrow L(R_1) =\{ab,c\}$
   $R_2 = d+ef \Longrightarrow L(R_2) =\{d,ef\}$
   Then $L(R_1. R_2) = L(R_1) . L(R_2) = \{ab,c\} . \{d,ef\} = \{abd,abef,cd,cef\}$
3. **Kleene Closure:** It is a string obtained by concatenating zero or more strings with repetitions. It is denoted by $L(R^*) = (L(R))^*$.
   Example: let $R = ab+c \Longrightarrow L(R) =\{ab,c\}$
   Then $L(R^*) = (L(R))^* = \{ab,c\}^* = \{\varepsilon, ab,c,abab,abc,cab,cc,ababc\ldots\}$
4. L((R))=L(R) denotes same language as R.
5. L(R).ε = ε .L(R) = L(R).

Identity rules of Regular Expressions:

Two regular expressions R_1 and R_2 are equivalent if $L(R_1) = L(R_2)$. The identity rules of regular expressions are

1. ε + R = R + ε
2. ε . R = R . ε = R
3. R + R = R
4. Φ + R = R + Φ =R
5. Φ . R = R . Φ = ϕ
6. ε* = ε = ϕ*
7. R* . R* = R*
8. R . R* = R* . R
9. ε + R . R* = ε + R* . R = R*
10. (R*)* = R*

PROOFS			
	LHS	RHS	

ε + R = R + ε	L(ε + R) = L(ε) ∪ L(R)
	= {ε} ∪ L(R)
		L(R + ε) = L(R)∪ L(ε)
		= L(R) ∪ {ε}
			Since L(ε + R)= L(R + ε), therefore ε + R = R + ε

ε . R = R . ε = R	L(ε . R) = L(ε) . L(R)
	= {ε} . L(R)
	= L(R)
		L(R + ε) = L(R). L(ε)
		= L(R) . {ε}
		= L(R)
			Since L(ε . R)= L(R . ε)= L(R), therefore ε . R = R . ε = R

R+R=R	L(R + R) = L(R) ∪ L(R) = L(R)		Since L(R+R) = L(R), therefore R+R = R

Φ + R = R + Φ =R	L(Φ + R) =L(Φ) ∪ L(R)
	= {} ∪ L(R)
	= L(R)
		L(R+Φ) = L(R)∪ L(Φ)
		= L(R) ∪ {}
		= L(R)
			Since L(Φ+R)= L(R+Φ) = L(R), therefore Φ+R= R + Φ=R

Φ . R = R . Φ = ϕ	L(Φ . R) =L(Φ) . L(R)
	= {} . L(R)
	= {}
	= Φ
		L(R.Φ) = L(R). L(Φ)
		= L(R) . {}
		= {}
		= Φ
			Since L(Φ.R)= L(R.Φ) = Φ , therefore Φ .R = R . Φ = ϕ

ε* = ε = ϕ*	L(ε*)=(L(ε))*
	={ ε}*
	={ ε, εε, εεε, …}
	={ ε, ε, ε, …}
	={ ε, εε, εεε, …}
	={ ε}
	= L(ε)
		L(ϕ *)=(L(ϕ))*
		={ }*
		={ ε}
		= L(ε)
			Since L(ε*) =L(ϕ*)=L(ε) therefore ε* = ε = ϕ*

R* . R* = R*	L(R*.R*)=L(R*).L(R*) =(L(R))*.(L(R))* =(L(R))*.(ε.ε) =(L(R))*.(ε) =(L(R))*		Since L(R*.R*)=(L(R))* Therefore R*.R*=R*

R . R* = R* . R	L(R.R*)=L(R).L(R*)
	=L(R).(L(R))*
	=(R).{ε,R,RR,…}
	={R,RR,RRR,…}
		L(R*.R)=L(R*).L(R)
		=(L(R))*.L(R)
		={ε,R,RR,..}(R)
		={R,RR,RRR,..}
			Since L(R.R*)=L(R*.R) Therefore R.R*=R*.R

ε+R.R*=ε+R*.R=R*	L(ε+R.R*)=L(ε)∪L(R.R*)
	= {ε} ∪ {R,RR,RRR,…}
		L(ε+R*.R)=L(ε)∪L(R*.R)
		= {ε} ∪ {R,RR,RRR,…}
			Since L(ε+R.R*)=L(ε+R*.R)=

	= L(R*)	= L(R*)	L(R*) Therefore ε+R.R*=ε+R*.R=R*

(R*)* = R*	L((R*)*)= (L(R*))*
	={(L(R)*}*
	=(R*)*
	={ ε,R,RR,…}*
	={ ε,R,RR,…}
	=R*
		L(R*)=(L(R))*
		=R*
			Since L((R*)*)= L(R*)
			Therefore (R*)* = R*

Algebraic laws of Regular Expressions:

1. Commutative law:	R + S = S + R
2. Associative law:	(R+S)+T = R+(S+T)
	(R.S).T = R.(S.T)
3. Distributive law:	(R+S)T = RT+ST
	(S+T)R = SR+TR
4. Demorgan's law:	(R+S)* = (R*.S*)* = (R*+S*)*
	(R.S)* = (R*.S*)* = (R*+S*)*
5. Other laws:	(RS)*R = R(SR)*

Equivalence of two regular expressions:

For every regular expression, there is an equivalent finite automata. Two regular expressions are said to be equivalent, if same finite automata is constructed for both of the regular expressions.

Finite Automata, and Regular Expressions:

Finite Automata to Regular Expressions:

Arden's theorem: It is used to construct regular expression from finite automata.

Statement: If P, Q are two Res, and if P does not contain epsilon ε, then equation R=Q+RP has a unique solution given by R=QP*.

Steps to convert finite automata to regular expression:

1. Write the equation for each and every state using the following notation
   Name of state = (state from which input is coming) (input symbol on the transition)
2. While writing the equation for starting state, include ε.
3. Make substitutions and apply ardens theorem wherever possible to obtain a regular expression.

Regular Expression to Finite Automata:

There are 2 ways to convert regular expression to finite automata. They are

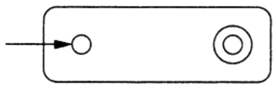
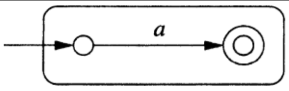
1. Top-down approach
2. Bottom-up approach

Top-down approach: FA construction starts from RE and ends by reaching a single element on transition.

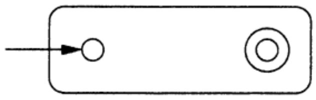
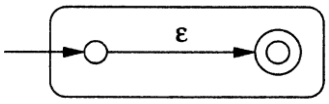
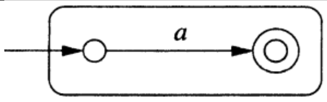
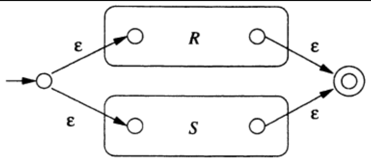
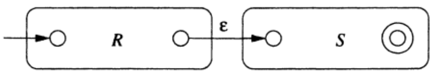
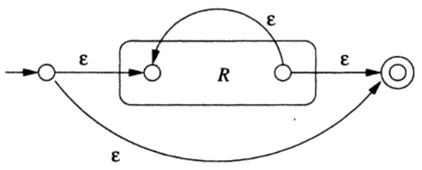
Bottom-up approach: FA construction starts from basic element and ends on reaching RE.

Steps to covert regular expression to finite automata:

Top-down approach:

1	Take 2 states – start state q_0 and final state q_f.	
2	Make transition between these two states and place the given regular expression on the transition.	
3	If regular expression is in the form of $RE=R_1+R_2$, then finite automata is represented as	
4	If regular expression is in the form of $RE=R_1.R_2$, then finite automata is represented as	
5	If regular expression is in the form of RE=R*, then finite automata is represented	
6	Make use of the above steps until the finite automata contains only single element on each transition.	

Bottom-up approach:

1	Divide the given regular expression into number of parts (Each part must be in the form of a RE.) and construct finite automata for each and every part separately using steps 2 to 7.	
2	Take 2 states – start state q_0 and final state q_f. Make transition between these two states and place the regular expression on the transition.	
3	If regular expression is in the form of RE=ε, then finite automata is represented as	
4	If regular expression is in the form of RE=a, then finite automata is represented as	
5	If regular expression is in the form of RE=R+S, then finite automata is represented as	
6	If regular expression is in the form of RE= R.S, then finite automata is represented as	
7	If regular expression is in the form of RE=R*, then finite automata is represented	
8	connect all the finite automata with an epsilon transition to obtain resultant finite automata.	

Pumping Lemma for regular languages:

Every language is not regular. To show that L is not regular, pumping lemma is used. Pumping means "generating". Pumping lemma means "it is a method of generating many input strings from a given string." i.e., it is used to break a given long input string into several substrings."

Statement of pumping lemma:

Let $\boldsymbol{M = (Q, \Sigma, \delta, q_0, F)}$ be a finite automata with 'n' number of states.

1. Assume that the given language L is regular accepted by finite automata M.
2. Choose a string $w\epsilon$ L such that $|w| \geq n$. i.e., length of string ≥ number of states.
3. Break the string into 3 strings x,y,z, i.e., w=xyz such that $|xy| \leq n$ and $y \neq \varepsilon$.
4. By using pumping lemma, pump y to the suitable integer 'i' such that $xy^iz \notin L$ for $i \geq 0$. This contradicts our assumption.
5. Therefore L is not regular.

Closure Properties of Regular Languages:

Closure property: *A set is* closed *under an operation if applying that operation to elements of the set results in an element of the set.*

For example, the set of natural numbers is closed under addition and multiplication but not under subtraction or division because 2-3 is negative and 1/2 is not an integer. The set of integers is closed under addition, subtraction and multiplication but not under division.

Regular sets are closed under the following:
1. Union
2. Concatenation
3. Complementation
4. Intersection
5. Kleene closure
6. Substitution
7. Homomorphism
8. Inverse homomorphism
9. Quotient operation

Union: *The class of regular languages is closed under union.*

Statement: If L1 , L2 are two regular languages, then L1 ∪ L2 is also regular.
Proof: let r1 is a regular expression denoting L1=L(r1) and
let r2 is a regular expression denoting L2=L(r2)

L1 ∪ L2 = L(r1) ∪ L(r2)
=L(r1+r2)
According to recursive definition of regular expression, if r1 and r2 are two regular expressions, then r1+r2 is also a regular expression.
Since r1+r2 is a regular expression and L1 ∪ L2 = L(r1+r2), therefore L1 ∪ L2 is also a regular.

Concatenation: *The class of regular languages is closed under concatenation.*

Statement: If L1 , L2 are two regular languages, then L1 . L2 is also regular.
Proof: let r1 is a regular expression denoting L1=L(r1) and
let r2 is a regular expression denoting L2=L(r2)

L1 . L2 = L(r1) . L(r2)
=L(r1.r2)
According to recursive definition of regular expression, if r1 and r2 are two regular expressions, then r1.r2 is also a regular expression.
Since r1.r2 is a regular expression and L1 . L2 = L(r1.r2), therefore L1 . L2 is also a regular.

Complementation: *The class of regular languages is closed under Complementation.*

Statement: If L(M) is a regular language, then L(M') is also regular ($\bar{l} = \sum^* - L$) .
Proof: Let L(M) is regular and let *M=(Q,∑,δ,q0,F)* be a DFA for *L(M)* and *M'=(Q,∑,δ,q0,Q-F)* be a DFA for L(M'), complement of L.
If we construct a DFA for *L(M)*, then $w \in L$, if *w* leads to final state in M. This implies that in *M'*, *w* leads to the same state but this state is non-accepting state in *M'*. Therefore, *M*' rejects *w*. Similarly,if $w \notin L$, then *M*' accepts *w*.
Therefore, *L(M') = L(M).*
Since L(M) is a regular language, therefore L(M') is also regular

Intersection: *The class of regular languages is closed under Intersection.*

Statement: If L1 , L2 are two regular languages, then L1 ∩ L2 is also regular.
Proof: let r1 is a regular expression denoting L1=L(r1) and
let r2 is a regular expression denoting L2=L(r2)

L1 ∩ L2 = $(\overline{\overline{l1} \cup \overline{l2}})$
According to closure properties of regular languages,
since L1 and L2 are two regular languages, its complement $\overline{l1}$ and $\overline{l2}$ are also regular.
since $\overline{l1}$ and $\overline{l2}$ are two regular languages, its union $\overline{l1} \cup \overline{l2}$ is also regular.
since $\overline{l1} \cup \overline{l2}$ are two regular languages, its complement $\overline{\overline{l1} \cup \overline{l2}}$ is also regular.
Therefore L1 ∩ L2 is also regular.

Kleene Closure: *The class of regular languages is closed under kleene closure.*

Statement: If L1 is a regular language, then L1* is also regular.
Proof: let r1 is a regular expression denoting L1=L(r1) and

L1* = (L(r1))*
=L(r1*)
According to recursive definition of regular expression, if r1 is a regular expression, then r1* is also a regular expression.
Since r1* is a regular expression and L1* = L(r1*), therefore L1* is also a regular.

Substitution: *The class of regular languages is closed under substitution.*

Statement: If L1 is a regular language with some alphabet ∑, then L1 is also regular with a unique replacement of ∑ with $\sum_1$.
Proof: let ∑={a,b}, then a is a regular expression denoting L1={a} and
b is a regular expression denoting L2={b}
therefore L= L1.L2=ab is a regular language over ∑={a,b}.
let $\sum_1$={0,1}, on substituting 'a' with '0' and 'b' with '1'.
L=01 is also regular.

Homomorphism: *The class of regular languages is closed under homomorphism.*

Statement: If L1 is a regular language with some alphabet ∑, and if strings in L1are replaced with other strings, then resultant language is also regular.
Proof: let ∑={a,b}, then aba is a regular expression denoting L1={aba} and
(aa)* is a regular expression denoting L2={(aa)*}
therefore L= L1.L2=aba(aa)* is a regular language over ∑={a,b}.
on substituting 'aba' with '0' and 'aa' with '1'.
h(aba)=0

h(aa)=1
then 01* is also regular.

Inverse Homomorphism: *The class of regular languages is closed under inverse homomorphism.*

Statement: It works by applying homomorphism backwards
Proof: let h(aba)=0
h(aa)=1
then h(01)=aba(aa)
h^{-1}(abaaa)=01 and h^{-1}((abaaa + (aa)*)*)=(01+1*)*

Quotient of Language: *The class of regular languages is closed under quotient of a language.*

Statement: It works by applying homomorphism backwards
Proof: If L1,L2 are two regular languages, then Quotient denoted by L1/L2 is also regular.
L1/L2={x|∃y in L2 and ∋ xy is in L1}
e.g: Let L1=0*10*, L2=0*1, then L1/L2=0*10*/0*1=0*

Finite Automata and regular grammars:

Grammar: Grammar is a set of rules used to define a valid sentence in any language.

Mathematical representation of grammar: Grammar is defined as a 4-tuple G={V,T,P,S} where
V - finite set of non-terminals
T - finite set of terminals
P – finite set of production rules
S – Start symbol

"Non terminals" are represented by capital letters A, B, C,… so on.
"Terminals" are represented by small letters a, b, c,… so on.
"Production rules" are of the form α → β, where
α contains non-terminal symbols or may also contain both terminals and non-terminals with atleast one non-terminal.
β contains terminal, non-terminal symbols or any combination of both terminals and non-terminals.

Types of grammar:

1. Linear grammar-
   Left linear grammar and Right linear grammar
2. Non-Linear grammar
3. Simple grammar
4. Recursive grammar
5. Regular grammar

Linear grammar: A grammar with atmost one variable (non-terminal) at right side of production rule is called linear grammar.

Example:	S→aSb	S→A	S→Ab
	S→ε	A→aB|ε	A→aAb|ε
		B→Ab	

Non linear grammar: A grammar with more than one variable (non terminal) at the right side of the production is called non linear grammar.

Example:	S→SS	S→aA
	S→ε	A→ε
	S→aSb	S→aSAS
	S→bSa	

Simple or S- Grammar: A grammar is said to be S-Grammar if all the productions are of the form A→ax, where A ∈ V (variables)
a ∈ T (terminals)
x ∈ V* (0 or more variables)
and any pair (A,a) occurs atmost once in production p.

Example:	S→aS	S→aAS|a	S→aS
	S→bSS	A→SbA	S→bSS
	S→c		S→aSS
			S→c
	(s,a), (s,b) occur only once	(s,a) occur only once	(s,a) occur twice
	∴ s-grammar	∴ s-grammar	∴ not s-grammar

Recursive grammar: A grammar is said to be recursive grammar, if it contains either a recursive production or an indirectly recursive production.

Example:	S→aS	S→aA|b	S→SS
		A→bS|c	S→aSb
			S→ε

Left linear grammar: A grammar G=(V,T,P,S) is said to be left linear grammar if all the productions are of the form A→Bx or A→x, where A,B ∈ V (variables)
x ∈ T* (0 or more terminals)
i.e., RHS non terminal is at left end.

Example:	S→Ab	S→Aab
	S→Sb	A→Aab
	S→ε	A→B
		B→a

Right linear grammar: A grammar is said to be right linear grammar if all the productions are of the form A→xB or A→x, where A,B ∈ V (variables)
x ∈ T* (0 or more variables)
i.e., RHS non terminal is at right end.

Example:	S→abS	S→A
	S→a	A→aA
		A→ε

Regular Grammar: A grammar G=(V,T,P,S) is said to be regular grammar, if it is either right linear grammar or left linear grammar or nothing. i.e., productions are in the form of A→xB or A→Bx or A→x or A→ε, where A,B ∈ V (variables)
x ∈ T* (0 or more terminals)
i.e., LHS contains a single non terminal and
RHS contains only terminals or
single non terminal followed by terminals or
terminals followed by single non terminal or
nothing.

Example:	S→abS	S→Aab
	S→a	A→Aab|B
		B→a

Relation between Regular Grammar and Finite Automata:

Conversion between regular grammar and FA:

1. RLG to FA
2. FA to RLG
3. RLG to LLG
4. LLG to RLG
5. LLG to FA
6. FA to LLG

RLG to FA:

1. Let G=(V,T,P,S) be the RLG and M be the FA.
2. Start symbol of G is the start symbol of M.
3. Each variable in RLG will be a state in FA.
4. If variable has an ε-production A→ε, then that state A will become final state, otherwise a new final state will be introduced.
5. If production in G is of the form A→a_1 a_2 a_3… a_nB, then make transition
6. If production in G is of the form A→a_1 a_2 a_3… a_n, then make transition introducing a new final state.

FA to RLG:

1. Let G=(V,T,P,S) be the RLG and M be the FA.
2. Start symbol of M is the start symbol of G.
3. For each transition , write the production
4. Now combine all the productions to form a RLG.
5. Make substitutions if possible to minimize the grammar.

RLG to LLG:

1. Let L be RLG. Construct FA for given RLG.
2. Replace the start state with final state and vice versa and reverse the direction of all the edges.
3. Construct grammar from step2
4. Reverse the right side of each production of RLG obtained in step3.
5. Resultant grammar is LLG.

LLG to RLG:

1. Reverse the right side of each production of LLG
2. Construct FA for obtained grammar in step1.
3. Replace the start state with final state and vice versa and reverse the direction of all the edges.
4. Construct grammar from FA obtained in step3
5. Resultant grammar is RLG.

LLG to FA:

LLG → RLG → FA

FA to LLG:

FA → RLG → LLG

Regular Expressions and Regular Grammars:

Conversion between regular grammar and regular expression:

1. RG to RE
2. RE to RG

RG to RE:

1. Let G=(V,T,P,S) be the RLG, replace → symbol in the production with '=' symbol.
2. If there is a equation of the form A= aA | b, then convert it into A=a*b
3. By applying substitutions, compute the expression for the starting symbol.

RE to RG:

RE → FA → RLG