# What are the identity rules for regular expression?

# Properties of Regular expressions

**Regular expressions:**

* It is a way of representing regular languages.
* The algebraic description for regular languages is done using regular expressions.
* They can define the same language that various forms of finite automata can describe.
* Regular expressions offer something that finite automata do not, i.e. it is a declarative way to express the strings that we want to accept. They act as input for many systems. They are used for string matching in many systems(Java, python etc.)
* For example, Lexical-analyzer generators, such as Lex or Flex.

The widely used operators in regular expressions are Kleene closure(∗) ,concatenation(.) , Union(+).

**Rules for regular expressions:**

* The set of regular expressions is defined by the following rules.
* Every letter of ∑ can be made into a regular expression, null string, ∈ itself is a regular expression.   
  If r1 and r2 are regular expressions, then (r1), r1.r2, r1+r2, r1\*, r1+are also regular expressions.

**Example –**∑ = {a, b} and r is a regular expression of language made using these symbols

| **Regular expr** | **Regular set/language** |
| --- | --- |
| ∅ | { } |
| ∈ | {∈} |
| a\* | {∈, a, aa, aaa …..} |
| a+ b | {a, b} |
| a.b | {ab} |
| a\* + ba | {∈, a, aa, aaa,…… , ba} |

**Operations performed on regular expressions:**

**1. Union:**The union of two regular languages, L1 and L2, which are represented using L1 ∪ L2, is also regular and which represents the set of strings that are either in L1 or L2 or both.

**Example:**

L1 = (1+0).(1+0) = {00 , 10, 11, 01} and   
L2 = {∈ , 100}  
then L1 ∪ L2 = {∈, 00, 10, 11, 01, 100}.

**2. Concatenation:**

The concatenation of two regular languages, L1 and L2, which are represented using L1.L2 is also regular and which represents the set of strings that are formed by taking any string in L1 concatenating it with any string in L2.

**Example:**

L1 = { 0,1 } and L2 = { 00, 11} then L1.L2 = {000, 011, 100, 111}.

**3. Kleene closure:**

If L1 is a regular language, then the Kleene closure i.e. L1\* of L1 is also regular and represents the set of those strings which are formed by taking a number of strings from L1 and the same string can be repeated any number of times and concatenating those strings.

**Example:**

L1 = { 0,1} = {∈, 0, 1, 00, 01, 10, 11 …….} , then L\* is all strings possible with symbols 0 and 1 including a null string**.**

**Algebraic properties of regular expressions:**

Kleene closure is an unary operator and Union(+) and concatenation operator(.) are binary operators.

**1. Closure:**

If r1 and r2 are regular expressions(RE), then

* r1\*  is a RE
* r1+r2 is a RE
* r1.r2 is a RE

**2. Closure laws –**

* (r\*)\* = r\*, closing an expression that is already closed does not change the language.
* ∅\* = ∈, a string formed by concatenating any number of copies of an empty string is empty itself.
* r+ =  r.r\* = r\*r, as r\* = ∈ + r + rr+ rrr …. and r.r\* = r+ rr + rrr ……
* r\* = r\*+ ∈

**3. Associativity –**  
If r1, r2, r3 are RE, then   
**i.) r1+ (r2+r3) = (r1+r2) +r3**

* **For example** : r1 = a , r2 = b , r3 = c, then
* The resultant regular expression in LHS becomes a+(b+ c) and the regular set for the corresponding RE is {a, b, c}.
* for the RE in RHS becomes (a+ b) + c and the regular set for this RE is {a, b, c}, which is same in both cases. Therefore, the associativity property holds for union operator.

**ii.) r1.(r2.r3)  = (r1.r2).r3**

* **For example –** r1 = a , r2 = b , r3 = c
* Then the string accepted by RE a.(b.c) is only abc.
* The string accepted by RE in RHS is (a.b).c is only abc ,which is  same in both cases. Therefore, the**associativity property holds for concatenation operator.**

Associativity property does not hold for Kleene closure(\*) because it is unary operator.

**4. Identity –**  
In the case of union operators  
if r+ x = r  ⇒ x= ∅ as r ∪ ∅= r, therefore ∅ is the identity for +.  
Therefore, ∅ is the identity element for a union operator.  
In the case of  concatenation operator –  
if r.x = r , for x= ∈  
r.∈ = r  ⇒ ∈ is the identity element for concatenation operator(.) .

**5. Annihilator –**

* If r+ x = r  ⇒  r ∪ x= r , there is no annihilator for +
* In the case of a concatenation operator, r.x = x, when x = ∅, then r.∅ = ∅, therefore ∅ is the annihilator for the (.) operator. For example {a, aa, ab}.{ } = { }

**6. Commutative property –**  
If r1, r2 are RE, then

* r1+r2 = r2+r1. For example, for r1 =a and r2 =b, then RE a+ b and b+ a are equal.
* r1.r2 ≠ r2.r1. For example, for r1 = a and r2 = b, then  RE a.b is not equal to b.a.

**7. Distributed property –**  
**If r1, r2, r3** are regular expressions, then

* (r1+r2).r3 = r1.r3 + r2.r3  i.e. Right distribution
* r1.(r2+ r3) = r1.r2 + r1.r3  i.e. left distribution
* (r1.r2) +r3  ≠ (r1+r3)(r2+r3)

**8. Idempotent law –**

* r1 + r1 = r1  ⇒  r1 ∪ r1 = r1 , therefore the union operator satisfies idempotent property.
* r.r ≠  r ⇒ concatenation operator does not satisfy idempotent property.

**9. Identities for regular expression –**  
The two regular expression’s P and Q are equivalent (denoted as P=Q) if and only if P represents the same set of strings as Q does.

For showing the equivalence of two regular expressions we need to show some identities of regular expressions

Let P, Q and R be the regular expressions then the identity rules are as follows −

* εR=R ε=R
* ε\*= ε ε is null string
* (Φ)\*= ε Φ is empty string
* ΦR=R Φ= Φ
* Φ+R=R
* R+R=R
* RR\*=R\*R=R+
* (R\*)\*=R\*
* Ε+RR\*=R\*
* (P+Q)R=PR+QR
* (P+Q)\*=(P\*Q\*)\*=(P\*+Q\*)\*
* R\*(ε+R)=( ε+R)R\*=R+
* (R+ε)\*=R\*
* Є+R\*=R\*
* (PQ)\*P=P(QP)\*
* R\*R+R=R\*R