4- Regular Expressions

Regular expressions are another type of language-defining notation. Regular expressions define exactly the same languages that the various forms of finite automata describe: the regular languages.

Regular expression is a declarative way to express the strings. Thus they serve as input language for many systems that process strings. Examples are

1. Search commands such as UNIX `grep`.
2. Lexical-analyzer generators such as Lex or Flex.

4.1 Operations on Languages

1. **Union:** Union of two languages L and M, denoted L U M, is the set of strings in L or M, or both. For example if L = {0, 11, 001, 101} and M = {ε, 1, 10, 111, 1101}, then

   L U M = {ε, 0, 1, 10, 11, 001, 101, 111, 1101}.
2. **Concatenation:** Concatenation of two languages L and M, denoted LM, is the set of strings formed by taking any string in L and concatenating any string in M. If L = {0, 01, 110} and M = {1, 10, 100} then

   LM = {01, 010, 0100, 011, 0110, 01100, 1101, 11010, 110100}
3. **The Closure (or star or Kleene closure):** The closure of a language L, denoted L*, represents the set of those strings formed by taking any number of strings from L, possibly with repetitions and concatenating all of them. For example if L = {0, 1} then L* is the set of all possible strings of 0's and 1's including ε.

4.2 Building Regular Expressions

- Expressions start with some elementary expressions.
- More expressions are constructed by applying set of operators on elementary and previously constructed expressions.
- Parentheses are used to group operators with their operands.
- Operators for the operations union (U or +), concatenation (dot) and closure (*) are used.

4.2.1 Recursive Definition of Regular expressions

If E is a regular expression then L (E) denotes the language of E.

Basis: The basis consists of three parts:

1. The constant ε is a regular expression, denoting the language {ε} i.e., L (ε) = {ε}
2. The constant ε is a regular expressions, denoting the language ø i.e., L (ø) = ø.
3. If a ∈ Σ, then **a** is a regular expression. The language of **a**, i.e., L (**a**) = {a}.

Induction: If E and F are regular expressions then

1. E + F is a regular expression and L (E+F) = L (E) U L (F).
2. EF is a regular expression and L (EF) = L (E) L (F).
3. E* is a regular expression and L (E*) = (L(E))*
4. (E) is a regular expression and L ((E)) = L (E).

4.2.2 Precedence of regular expression operators

The following is the order of precedence for the operators:

1. The star operator is of highest precedence. It applies only to the smallest sequence of symbols to its left which is a well formed regular expression.
2. Next precedence is for the concatenation or dot operator.
3. Finally all unions (+) are grouped with their operands.

Example 1: The expression **01* + 1** is grouped as

1. The elementary expressions are **0** and **1**.
   L(**0**) = {0} and L(**1**) = {1}
2. The star operator is grouped first. It is applied to **1**, as it is the smallest sequence of symbols to its left and a well formed regular expression giving **1***.
   L(**1***) = (L(**1**))* = ({1})* = {ε, 1, 11, 111, ···} i.e., *zero or more 1's*.
3. Next we group the concatenation between **0** and (**1***) giving **(0(1*)).**
   L(**0(1)***) = L(**0**)L(**1***) = {0} {ε, 1, 11, 111, ···} = {0, 01, 011, 0111,··· } i.e., *zero followed by zero or more 1's*.
4. Finally the union operator + connects **(0(1*))** and **1** i.e., **(0(1*)) + 1**
   L(**(0(1*)) + 1**) = L(**0(1)***) U L(**1**)
   = {0, 01, 011, 0111,···} U {1}
   = {1, 01, 011, 0111,···} i.e., *1 or 0 followed by zero or more 1's*.

Example 2: The expression **(01)* + 1** is grouped as

1. The elementary expressions are **0** and **1**.
   L(**0**) = {0} and L(**1**) = {1}
2. The dot (Concatenation) operator is grouped first, as it is enclosed within parentheses. It is applied to **0** and **1**, giving **01**.
   L(**01**) = L(**0**) L(**1**) = {0} {1} i.e., *0 followed by 1*.
3. The star operator is grouped next giving **(01)*.**
   L((**01**)*) = (L(**01**))* = {ε,01, 0101, 010101, ⋯} i.e., *the pattern 01 repeated zero or more times*.
4. Finally the union operator + connects **(01)*** and **1** i.e., **(01*) + 1**
   L((**01***) **+ 1**) = (L(**01**))* U L(**1**)
   = {ε, 01, 0101, 010101, ⋯} U {1}
   = {ε, 1, 01, 0101, 010101,⋯} i.e., *1 or the pattern 01 repeated zero or times*.

Example 3: Write regular expression to represent the following language.

L = {all possible strings on the alphabet Σ = {a, b}}

Regular expression: (a + b)*

1. The elementary expressions are **a** and **b**.
   L(**a**) = {a} and L(**b**) = {b}
2. The + (Union) operator is grouped first, as it is enclosed within parentheses. It is applied to **a** and **b**, giving **a + b**.
   L(**a + b**) = L(**a**) U L(**b**) = {a, b}.
3. The star operator is grouped next giving **(a + b)*.**
   L((**a + b**)*) = (L(**a + b**))* = {a, b}* i.e., the set {a, b} concatenated zero or more times.

   $\{a, b\}^0$ = {ε}.

   $\{a, b\}^1$ = {a, b}.

   $\{a, b\}^2$ = {a, b}{a, b} = {aa, ab, ba, bb} all strings of length 2.

   $\{a, b\}^3$ = {a, b}{a, b}{a,b} = {aaa, aab, aba, abb, baa, bab, bba, bbb } all strings of length 3.

Example 4: Write regular expression to represent the following language.

L = {Strings on the alphabet Σ = {a, b} that end with ab}

Regular expression: (a + b)*ab

Example 5: Write regular expression to represent the following language.

L = {Strings on the alphabet Σ = {a, b} that start with ab}

Regular expression: ab(a + b)*

Example 6: Write regular expression to represent the following language.

L = {Strings on the alphabet Σ = {a, b} that start with a and end with b}

Regular expression: a(a + b)*b

Example 7: Write regular expression to represent the following language.

L = {Strings on the alphabet Σ = {0, 1} such that the second symbol is 1}

Regular expression:

First symbol can be 0 or 1, the regular expression is (0 + 1).

Second symbol must be 1, followed by any string on the alphabet {0, 1}.

∴ The regular expression is (0 + 1)1(0 + 1)*.

Example 8: Write regular expression to represent the following language.

L = {Strings on the alphabet Σ = {a, b, c} that contain the substring abb}

Regular expression: (a + b + c)*abb(a + b + c)*.

Example 9: Regular expression to represent the strings with any number of a's followed by any number of b's followed by any number of c's.

Regular expression: a*b*c*.

Example 10: Regular expression to represent the strings with one or more a's followed by one or more b's followed by one or more c's.

Regular expression: aa*bb*cc* **OR** a*ab*bc*c.

Example 11: Regular expression to represent the following language

L = {Strings on the alphabet Σ = {0, 1} that end with 0 or 01}.

Regular expression: (0 + 1)*(0 + 01).

Example 12: Regular expression to represent strings with even number of a's followed by odd number of b's.

Regular expression: (aa)*(bb)*b

Example 13: Regular expression to represent the following language

L = {w | w ∈ {a, b}* and $n_a(w)$ mod 3 = 0}.

Regular expression: (b*ab*ab*ab*)*

Example 14: Regular expression to represent the following language

L = {w | w ∈ {a, b}* and $n_a(w)$ mod 3 ≠ 0}.

Regular expression:

$n_a(w)$ mod 3 ≠ 0 i.e., $n_a(w)$ mod 3 = 1 **OR** $n_a(w)$ mod 3 = 2

Regular expression for $n_a(w)$ mod 3 = 1 is (b*ab*)*

Regular expression for $n_a(w)$ mod 3 = 2 is (b*ab*ab*)*

Regular expression for $n_a(w)$ mod 3 ≠ 0 is (b*ab*)* + (b*ab*ab*)*

Example 15: Regular expression to represent the following language

L = {w | w ∈ {a, b}* and w has at least one a}.

Regular expression: (a + b)*a(a + b)*

Example 16: Regular expression to represent the following language

L = {w | w ∈ {a, b}* and w has exactly two a's}.

Regular expression: b*ab*ab*

Example 17: Regular expression to represent the following language

L = {w | w ∈ {a, b}* and w has at most two a's} i.e., number of a's = 0 ***OR*** number of a's = 1 ***OR*** number of a's = 2.

Regular expressions for:

Number of a's = 0, regular expression is b*

Number of a's = 1, regular expression is b*ab*

Number of a's = 2, regular expression is b*ab*ab*

Regular expression for strings with at most two a's:

b* + b*ab* + b*ab*ab*

OR

b*(a + ε)b*(a + ε)b*

Example 18: Regular expression to represent the strings on Σ = {a, b, c} with at least one a and at least one b.

Regular expression:

- Regular expression for strings with one a and one b is ab or ba
  i.e., ab + ba.
- This may be surrounded by any string on the alphabet Σ = {a, b, c}.
- ∴ The regular expression is (a + b + c)*(ab + ba) (a + b + c)*.

Example 19: Regular expression to represent the strings on Σ = {0, 1} with alternating sequence of 0's and 1's.

(1 + Ɛ)(01)*(0 + Ɛ)

Example 20: Regular expression to represent the strings on Σ = {a, b} such that the strings start with or end with ab.

ab(a + b)* + (a + b)*ab

Example 21: Regular expression to represent the strings on Σ = {a, b} such that the string length is odd and ends with a.

((a + b) (a + b))*a

4.3 Algebraic Laws for Regular Expressions:

If R, S, T are regular expressions then

Commutative Law for Union: R + S = S + R

Associative Law:

- For Union: (R + S) + T = R + (S + T)
- For Concatenation: (RS)T = R(ST)

Identities:

(Note: Identity for an operator is a value such that when the operator is applied to the identity and some value x the result is x itself.

For example x + 0 = 0 + x = x zero is the identity for addition.

x * 1 = 1 * x = x one is the identity for multiplication.)

- ∅ + R = R + ∅ = R, ∅ is the identity for union.
- ƐR = RƐ = R, Ɛ is the identity for concatenation.

Annihilators:

(Note: Annihilator for an operator is a value such that when the operator is applied to the annihilator and some value x the result is Annihilator itself.

For example x * 0 = 0 * x = 0, zero is the annihilator for multiplication.)

∅R = R∅ = R, ∅ is the identity for concatenation.

Distributive Laws:

- R(S + T) = RS + RT Left distributive law of concatenation over union.
- (R + S)T = RT + ST Right distributive law of concatenation over union.

Idempotent Law:

R + R = R

Laws Involving Closures:

- $(R^*)^* = R^*$
- $\emptyset^* = \varepsilon$
- $\varepsilon^* = \varepsilon$
- $R^+ = RR^* = R^*R$
- $R^* = R^+ + \varepsilon$
- $R? = R + \varepsilon = \varepsilon + R$