## Some Terminologies

**Alphabets :**

An alphabet is a finite, nonempty set of symbols. The alphabet of a language is normally denoted by Σ .

Example :

$$\Sigma = \{0,\ 1\}$$

$$\Sigma = \{a,\ b,\ c\}$$

$$\Sigma = \{a,\ b,\ c,\ \&,\ z\}$$

$$\Sigma = \{\#,\ \nabla,\ \spadesuit,\ \beta\}$$

## Strings or Words over Alphabet :

A string or word over an alphabet Σ is a finite sequence of concatenated symbols of Σ. Denoted by Σ*

**Example :** 0110, 11, 001 are three strings over the binary alphabet { 0, 1 } .
aab, abcb, b, cc are four strings over the alphabet { a, b, c }.

It is not the case that a string over some alphabet should contain all the symbols from the alphabet. For example, the string cc over the alphabet { a, b, c } does not contain the symbols a and b. Hence, it is true that a string over an alphabet is also a string over any superset of that alphabet.

**Length of a string :**
The number of symbols in a string w is called its length, denoted by |w|.

**Example :** | 011 | = 4, |11| = 2, | b | = 1

It is convenient to introduce a notation e for the empty string, which contains no symbols at all. The length of the empty string e is zero, i.e., | e | = 0.

**Empty string:**
The string of zero length is empty string. It is denoted by e or $\lambda$.

**Reverse string:**
IF $w= w_1w_2\ldots.w_n$ where each $w_i \in \Sigma$ , the reverse of w is $w_nw_{n-1}\ldots\ldots w_1$

**Substring:**
Z is a substring of w if z appears consecutively within w .
e.g " deck" is a substring of abcdeckabcjkl.

**Concatenation**
Two strings over the same alphabet can be combined to form a third by operation of concatenation. The concatenation of strings x and y written xoy or simply xy is the string x followed by the string y.
Formally w=xy if and only if |w| =|x| + |y|, w(j)= x(j) for j=1 … , |x| and w(|x|+j)=y(j) for j=1 , |y|.
e.g 01o 001=01001
woe=eow =w for any string w . concatenation is associative i.e (wx)y= w(xy) for any string w, x,y.

**Suffix:**
If w=xy for some x, then y is a suffix of w.
**Prefix:**
If w=xy for some y, then x is a prefix of w.
Note: for each string w and each natural number i, the string $w^i$ is defined as
$w^0 = e$
$w^{i+1} = w^i w$
$w^1=w$
$do^2 = dodo$

**Language**

Language L over the alphabet ( $\Sigma$) is the subset of $\Sigma^*$ . Any set of strings over an alphabet $\Sigma$ i.e any subset of $\Sigma^*$ will be called a language. Thus $\Sigma^*$ , ø and $\Sigma$ are languages.

Example

For $\Sigma= \{ 0, 1\}$

$L= \{ x : x \in \{ 0,1 \} ^* \mid x$ has even number of Zero's $\}$

$011011100 \in L$ , $1111 \in L$ , $1011011000 \notin L$

The infinite language L are denoted as

$L= \{ w \in \Sigma^* : w$ has property $P\}$

**1.5 Operation on Languages**

1. **Concatenation operation**

   If $L_1$ and $L_2$ are languages over $\Sigma$ , their concatenation is L = L1.L2 or L1L2 where

   $L= \{ w \in \Sigma^* : w = xy$ for some $x \in L1$ and $y \in L2 \}$

   **e.g** $\Sigma= \{ 0,1\}$

   $L_1= \{ w \in \Sigma^* : w$ has an even number of 0's$\}$

   $L_2= \{ w \in \Sigma^* : w$ starts with a 0 and the rest of the symbols are 1's $\}$

   then

   $L_1L_2= \{ w \in \Sigma^* : w$ has an odd number of 0's $\}$
2. **Kleene star**

   Denoted by L* . It is the set of all strings obtained by concatenating zero or more strings from L.

   $L^*= \{ w \in \Sigma^* : w = w_1 o\ w_2\ o \ldots\ldots o\ w_k$ for some $K \geq 0$ and some $w_1 o\ w_2\ o \ldots\ldots o\ w_k \in L \}$

   We write $L^+$ for the language $LL^*$.

   Equivalently $L^+$ is the language $L^+ = \{ w \in \Sigma^* : w = w_1 o\ w_2\ o \ldots\ldots o\ w_k$ for some $K \geq 1$ and some $w_1 o\ w_2\ o \ldots\ldots o\ w_k \in L \}$
3. **Union**

## 1.6 A regular expression

A regular expression is a string that describe a whole set of strings, according to certain syntax rules. These expressions are used by many text editor and utilities especially in the unix operation System to search bodies of text for certain patterns and for example, replace the forward strings with a certain other string.

Regular expressions were designed to represent regular languages with a mathematical tool, a tool built from a set of primitives and operations. This representation involves a combination of strings of symbols from some alphabet $\sum$, parantheses and the operators +, ×, and *.

***Formal Definition***

*The regular expressions over an alphabet ∑* are all strings over the alphabet Σ ∪ {( , ), ø, ∪,*} that can be obtained as follows.*

- *ø and each member of Σ is a regular expression.*
- *If α and β are regular expressions, then so is (α β).*
- *If α and β are regular expressions, then so is (α ∪β).*
- *If α is a regular expressions, then so is α* .*
- *Nothing is a regular expression unless it follows from (1) through (4).*

*Note: Every regular expression is associated with some language.*

Example of regular expression

Assume that Σ = {a, b, c}

- *Zero or more: a** means "zero or more *a*'s",
- To say "zero or more *ab*'s," i.e., {e, *ab, abab,* …..} you need to say (*ab*)*.
- *One or more:* Since *a** means "zero or more *a's*", you can use *aa** (or equivalently *a***a*) to mean "one or more *a*'s". Similarly to describe 'one or more *ab*'s", that is {*ab*, *abab*, *ababab*, KK}, you can use *ab* (*ab*)*.
- *Zero or one:* It can be described as an optional '*a*' with (*a* + e).
- *Any string at all:* To describe any string at all (with Σ= { *a*, *b*, *c*} you can use (*a* + *b* + *c*)*.
- *Any nonempty string:* This is written any character from Σ= {*a*, *b*, *c*} followed by any string at all: (*a* + *b* + *c*) (*a* + *b* + *c*)*
- *Any string not containing* ..........: To describe any string at all that does not contain an '*a*' (with Σ={*a*, *b*, *c*}), you can use (*b* + *c*)*.
- *Any string containing exactly one ........:* To describe any string that contains exactly one '*a*' put "any string not containing an *a*", on either side of the '*a*' like: (*b* + *c*)* *a*(*b* + *c*)* .

**<u>The Operators of Regular Expressions:</u>**

+, **.** , * (Union, concatenation , kleene star)

**<u>Precedence of Regular-Expression Operators</u>**

- The star Operator is of highest precedence.
- Next in precedence comes the concatenation or dot operator.
- Finally, unions are grouped with their operands.

For example:

The expression 10*+0 is grouped (1(0)*)+0.

**Difference between L$^*$ and L$^+$**

L* denotes Kleene closure and is given by $L^* = \bigcup_{i=0}^{\infty} L^i$
Example : 0*= {e, 0,00,000,…. }
Language includes empty words also.

L$^+$ denotes Positive closure and is given by $L^+ = \bigcup_{i=1}^{\infty} L^i$
$L^+ = LL^*$

**Regular Languages**

The regular languages are those languages that can be constructed from the “big three” set operations viz., (a) Union (b) Concatenation (c) Kleene star.
A regular language is defined as follows.
***Definition*****:** Let Σ be an alphabet. The class of “regular languages” over ∑ is defined inductively as follows**:**

(a) $\varnothing$ is a regular language
(b) For each $\sigma \in \Sigma$, $\{\sigma\}$ is a regular language
(c) For any natural number $n \geq 2$ if $L_1, L_2, \ldots\ldots L_n$ are regular languages, then so is $L_1 \cup L_2 \cup \ldots \ \ldots \cup L_n$.
(d) For any natural number $n \geq 2$, if $L_1, L_2, \ldots\ldots L_n$ are regular languages, then so is $L_1 \circ L_2 \circ \ldots \ \ldots \circ L_n$.
(e) If $L$ is a regular language, then so is $L^*$.
(f) Nothing else is a regular language unless its construction follows from rules (a) to (e).

*Examples*:

(i) $\varnothing$ is a regular language (by rule (a))
(ii) $L = \{a, ab\}$ is a language over $\Sigma = \{a, b\}$ because, both $\{a\}$ and $\{b\}$ are regular languages by rule (b). By rule (d) it follows that $\{a\} \circ \{b\} = \{ab\}$ is a regular language. Using rule (c), we see that $\{a\} \cup \{ab\} = L$ is a regular language.
(iii) The language over the alphabet {0,1} where strings contain an even number of 0’s can be constructed by

$$(1^*((01^*)(01^*))^*)$$

or simply $1^*(01^*\ 01^*)^*$.

Formally, the relation between regular expressions and the languages they represent is established by a function L, such that if a is any regular expression, then L(a) is the language represented by a. That is, L is a function from strings to languages. The function £ is defined as follows.

(1) $\mathcal{L}(\varnothing) = \emptyset$, and $\mathcal{L}(a) = \{a\}$ for each $a \in \Sigma$.
(2) If $\alpha$ and $\beta$ are regular expressions, then $\mathcal{L}((\alpha\beta)) = \mathcal{L}(\alpha)\mathcal{L}(\beta)$.
(3) If $\alpha$ and $\beta$ are regular expressions, then $\mathcal{L}((\alpha \cup \beta)) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta)$.
(4) If $\alpha$ is a regular expression, then $\mathcal{L}(\alpha^\star) = \mathcal{L}(\alpha)^*$.

Statement 1 defines $\mathcal{L}(\alpha)$ for each regular expression $\alpha$ that consists of a single symbol; then (2) through (4) define $\mathcal{L}(\alpha)$ for regular expressions of some length in terms of $\mathcal{L}(\alpha')$ for one or two regular expressions $\alpha'$ of smaller length. Thus every regular expression is associated in this way with some language.

**Example 1.8.2:** What is $\mathcal{L}(((a \cup b)^\star a))$? We have the following.

$$\begin{aligned}
\mathcal{L}(((a \cup b)^\star a)) &= \mathcal{L}((a \cup b)^\star)\mathcal{L}(a) \text{ by}(2)\\
&= \mathcal{L}((a \cup b)^\star)\{a\} \text{ by (1)}\\
&= \mathcal{L}((a \cup b))^*\{a\} \text{ by (4)}\\
&= (\mathcal{L}(a) \cup \mathcal{L}(b))^*\{a\} \text{ by (3)}\\
&= (\{a\} \cup \{b\})^*\{a\} \text{ by (1) twice}\\
&= \{a, b\}^*\{a\}\\
&= \{w \in \{a, b\}^* : w \text{ ends with an } a\}
\end{aligned}$$

**Languages defined by Regular Expressions**

There is a very simple correspondence between regular expressions and the languages they denote:

| Regular expression | $L$ (Regular Expression) |
|---|---|
| $x$, for each $x \in \Sigma$ | $\{x\}$ |
| $\lambda$ | $\{\lambda\}$ |
| $\varnothing$ | $\{\ \}$ |
| $(r_1)$ | $L(r_1)$ |
| $r_1^*$ | $(L(r_1))^*$ |
| $r_1 r_2$ | $L(r_1)L(r_2)$ |
| $r_1 + r_2$ | $L(r_1) \cup L(r_2)$ |

## Algebraic Laws for RE’s

- Commutative law for union: $L + M = M + L$
- Associative law for union: $(L + M) + N = L + (M + N)$
- Associative law for concatenation: $(LM)N = L(MN)$ , Note that there is no commutative law for
- Concatenation, i.e. $LM \neq ML$
- The identity for union is: $L + \emptyset = \emptyset + L = L$
- The identity for concatenation is: $L.\ e = e.\ L = L$
- The annihilator for concatenation is: $\emptyset L = L\emptyset = \emptyset$
- Left distributive law: $L(M + N) = LM + LN$
- Right distributive law: $(M + N)L = LM + LN$
- Idempotent law: $L + L = L$

**Laws Involving Closure**

- $(L^*)^* = L^*$ – i.e. closing an already closed expression does not change the language
- $\emptyset^* = e$
- $e^* = e$
- $L+ = LL^* = L^*L$ – more of a definition than a law
- $L^* = L^+ + e$
- $L? = e + L$ – more of a definition than a law

**Checking a Law**

Suppose we are told that the law $(R + S)^* = (R^*S^*)^*$ holds for regular expressions. How would we check that this claim is true?

1. Convert the RE’s to DFA’s and minimize the DFA’s to see if they are equivalent
2. We can use the “concretization” test:
   - Think of R and S as if they were single symbols, rather than placeholders for languages, i.e., $R = \{0\}$ and $S = \{1\}$.
   - Test whether the law holds under the concrete symbols. If so, then this is a true law, and if not then the law is false.

## Concretization Test

For our example : $(R + S)^* = (R^*S^*)^*$

We can substitute 0 for R and 1 for S. The left side is clearly any sequence of 0's and 1's. The right side also denotes any string of 0's and 1's, since 0 and 1 are each in $L(0^*1^*)$.

• NOTE: extensions of the test beyond regular expressions may fail.
Consider the “law” $L \cap M \cap N = L \cap M$.
This is clearly false

– Let $L=M=\{a\}$ and $N= \emptyset$. $\{a\} \neq \emptyset$.
– But if $L=\{a\}$ and $M = \{b\}$ and $N=\{c\}$ then
– $L\cap M$ does equal $L\cap M \cap N$ which is empty.
– The test would say this law is true, but it is not because we are applying the test beyond regular expressions.