3 Regular Expressions and Languages

'Every Regular expression is associated with some language.'

Introduction

The power of computers is estimated with regard to their (a) speed (b) storage capacity and (c) amplification capability of codes. Suppose, we have a directory called public - html containing a number of personal world wide web files like gif, midi, html etc., without any subdirectories. Then, the reorganisation of the directory, by creating subdirectories for each type of file could be done by using a *pattern-matching mechanism*. In other words, a pattern is used to extract (or filter) out a subset of files from a larger collection of files. For example, *. midi might generate the following subset of files:

Karn.midi, Chen.midi, Andh.midi, ...

In other words, one command works whether there are 3 or 3000 .midi files. Our goal is to develop a way of specifying a filter, that extracts out a desired subset from a larger set by using the language operations discussed in chapter 2.

We shall show that the basic set of operations on languages provides us with a mathematical way of specifying a *filter*.

Suppose Σ represents the alphabet of all possible characters that appear in file names. Then obviously, all the file names, that begin with 1, can be given by the expression:

$$\{1\} \circ \Sigma^*.$$

Accordingly, all the files that end with .midi can be given by the expression:

$$\Sigma^* \circ \{.\text{midi}\}.$$

Then, clearly

$$L_{\text{filter}} = \{1\} \circ \Sigma^* \circ \{.\text{midi}\}$$

represents the set of strings that start with 1 and end in .midi.

3.1 Regular Languages

Regular Languages are those that can be constructed from the simple set of operations of *Union*, *Concatenation* and *Kleene star*. Every regular language can be expressed as an expression (or formula) by using the set operations of union, concatenation and kleene star.

3.1.1 *Definition*

Suppose Σ is an alphabet. Then, the class of *regular languages* over Σ is inductively defined in the following manner:

(i) ϕ is a regular language.
(ii) For each $a \in \Sigma$, $\{a\}$ is a regular language.
(iii) If $L_1, L_2, \ldots, L_n$ are regular languages (where n is any natural number), then so is $\bigcup_{i=1}^{n} L_n$.
(iv) If $L_1, L_2, \ldots, L_n$ are regular languages (where n is any natural number), then so is $L_1 \circ L_2 \circ \ldots \circ L_n$.
(v) If L is a regular language, then so is L^*.
(vi) Nothing else is a regular language, unless it is constructed using points (i) ~ (v).

Each of the above formulae is called a *regular construction* and *regular languages* are those that are generated by regular constructions. While writing the above formulae in a compact form, the following convention is adapted:

a. the shorthand for $\{a\}$ is a
b. the shorthand for $x \circ y$ is xy
c. only when it is necessary to override the normal precedence of $*$, over $\circ$ and over $\cup$, parentheses are to be used.

EXAMPLE 3.1.1: (Regular languages)

a. Let $\Gamma = \{\alpha_i, \ldots, \alpha_R\}$ be an alphabet. Then $\{\alpha_i\}$ is a regular language (by rule (ii)) for $1 \leq i \leq R$. By applying rule (iii) to these singleton sets, we see that Γ is a regular language.
b. Let $L = \{b, bc\}$ be a language over $\Sigma = \{b, c\}$. Then by rule (ii), both $\{b\}$ and $\{c\}$ are regular languages and by rule (iv), $\{b\} \circ \{c\} = \{bc\}$ is a regular language. Again by using rule (iii), $\{b\} \cup \{bc\} = L$ is a regular language.
c. $\{\epsilon\}$ is a regular language and ϕ is also a regular language (by rule (i)). Then by applying rule (v), we observe that $\phi^* = \{\epsilon\}$ is a regular language.
d. Let $L = \{\epsilon, 0^2, 0^4, 0^6, \ldots\}$. Then, by rules (i) and (ii), $\{00\}$ is a regular language and so is $\{00\}^*$ (by rule (V)). Further, L is a regular language, since L equals $\{00\}^*$.
e. The language $\{\epsilon, 0, 1, 00, \cdots\} \cup \{\epsilon, 1, 11, \cdots\} = \{0 \cup 1\}^*$ or can be constructed by $\{0 \cup 1\}^* \cup 1^*$.
f. The language $\{\epsilon, b, bb, \cdots\}$ is constructed by b^*.

g. Observe that the language over the alphabet $\Sigma = \{0, 1\}$, that consists of all strings having the pattern of 01 as a substring, can be constructed easily by

$$(0 \cup 1)^* 01 (0 \cup 1)^*.$$

h. Observe that the language over the alphabet $\Sigma = \{0, 1\}$, whose strings contain an even number of 0s, can be constructed by

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

or simply $1^*(01^*01^*)^*$, where ϵ is included in this language.

3.1.2 *Standard Representations of Regular Languages:*

- Regular Expression
- DFAs
- NFAs
- Regular Grammars

Note–1:
We discuss the properties of regular languages in chapter 7, next we prove that certain languages are not regular using *pumping lemma* in chapter 12 and finally discuss the regular language and its relationship with formal grammar in chapter 15.

3.2 Regular Expressions

In this section, an easy-to-use notation for describing the construction of sets of strings, (using the basic language operations) is discussed together with its grammar (or syntax) and interpretation (semantics).

3.2.1 *Definition (Regular expression)*

Certain sets of strings or languages can be represented in an algebraic fashion, and these algebraic expressions of languages are called *regular expressions* (R.E).

A regular expression is a string that describes the whole set of strings according to a certain syntax rule. These expressions are used by text editors and utilities (in particular, UNIX OS), to search bodies of text for certain patterns.

3.2.2 Standard Regular expression

Suppose Σ is an alphabet. Then, the *standard regular expressions*, over the alphabet Σ, are defined (inductively) as follows:

a. The string ( ) is a regular expression.
b. For each $\sigma \in \Sigma$, the string symbol σ is a regular expression.
c. If R is a regular expression, then (R) and R^* are regular expressions.
d. If R_1 and R_2 are regular expressions, then $(R_1|R_2)$ and (R_1R_2) are regular expressions.
e. Regular expressions are formed only through rules (a) through (d).

Mathematically, each regular expression is a string over the alphabet $\Sigma \cup \{(,),|,*\}$. From the foregoing discussion, it is observed that each regular expression is merely a string of symbols assembled using the syntactic rules of definition–(a) to (e). A grammatical variable indicates a point, where the definition is expanded in a recursive manner.

The meaning of a regular expression is the particular *regular construction* that it describes. For example, $(a|c)^*$ describes the construction $(\{a\} \cup \{c\})^*$. But in a broader sense, the meaning of a regular expression is the *language* it constructs.

3.3 Components of Regular Expressions

The following are the notations used in regular expressions:

Sometimes '+', '$\circ$' and '$*$' are used to represent union, concatenation and kleene star closure respectively. A formal *definition* to regular expressions can be given as:

a. $\phi, \in$ and a belonging to Σ, are all regular expressions and are called the *primitive regular expressions*.
b. If R_1 and R_2 are regular expressions, then $R_1 + R_2, R_1 \circ R_2, R_1^*$ and (R_1) are all regular expressions.
c. A string is a regular expression, if and only if, it can be derived from primitive regular expressions by a finite number of applications of rule (b). Thus, by the repeated application of the rules, regular expressions are constructed.

EXAMPLE 3.3.1: (Regular expressions)

a. If $\Sigma = \{a, b, c\}$ is some alphabet, the regular expression can be built from the symbols as
$$(a + bc)^* \circ (c + \phi).$$
b. If $R_1 = c$ and $R_2 = \phi$, then $c + \phi$ i.e. $R_1 + R_2$ is also a regular expression.

c. $\circ(a + b+)$ is not a regular expression, since there is no way by which a regular expression can be constructed from the primitive regular expression.
d. If $\Sigma = \{0, 1\}$, then the following are the regular expressions that can be built from it:
   - 01 means, a zero followed by one concatenation (0 and 1).
   - $0 + 1$ means, either a zero or a one (union)
   - 0^* means, $\in + 0 + 00 + 000 + \cdots$ (star kleene)
   - 1^+ means, $1 + 11 + 111 + \cdots$ (positive closure)
e. If $\Sigma = \{0, 1\}$, then using the parentheses, the following regular expressions could be constructed:
   - $(0 + 1)^*$ means, set of all strings over 0 and 1.
   - $0^*10^*10^*$ means, strings containing exactly two ones.
   - $(0 + 1)^*11$ means, strings which end with two ones.

EXAMPLE 3.3.2: (Describing regular expressions in English)

Regular Expressions	*Meaning*
O^*	Set of strings of zeros of any length including $\in$
$O+$	Set of strings of zeros of any length excluding $\in$
$(a + b)^*$	Set of strings of a's and b's of any length including $\in$
$(a + b)^+$	Set of strings of a's and b's of any length excluding $\in$
$(a + b)^*abb$	Set of strings of a's and b's ending with abb
$ab(a + b)^*$	Set of strings of a's and b's starting with ab
$O^*\|^*2^*$	Set of strings of any number of zeros, followed by any number of 1's, followed by any number of 2's.
$O^+1^+2^+$	Set of strings of any number of zeros, followed by any number of 1's, followed by any number of 2's excluding $\in$.
$OO^*11^*22^*$	Set of strings of 0's, 1's and 2's with atleast one zero, followed by atleast one 1, followed by atleast one 2.
$(a + b)^*aa(a + b)^*$	Set of strings of a's and b's of any length having a substring aa.
$a^*b(a^*ba^*b)^*a^*$	A string which starts and ends with a or b, and has atleast one b, after the first b. Also, all b's in the string appear in pairs. Any number of a's can appear in any place in the string. Thus it is the set of strings over the alphabet $\{a, b\}$, that contains an odd number of b's.

$((a+b)^3)^*(\in +a+b)-((a+b)^3)$	Represents the strings of length 3. Here, $((a+b)^{3*})$ represents the strings of length, which is multiple of 3. Since $((a+b)^3)^*(a+b)$ represents the strings of length $3n+1$, where n is a natural number, the given regular expression represents the string of length $3n+1$.
$(b+ab)^*(a+ab)^*-(b+ab)^*$	$(b+ab)^*$ represents strings, which do not contain any substring aa and strings which end in b. $(a+ab)^*$ represents strings, which do not contain any substring bb. Hence, altogether it represents any string containing a substring with no aa followed by one b, followed by a substring with no bb.

EXAMPLE 3.3.3: (Forming a regular expression)

Construct a regular expression (RE) for the following:

a. RE containing even number of 0s
Solution:
$$RE = (00)^*.$$

b. RE that generates odd number of 1s
Solution:
Recall that $(11)^*$ indicates RE which generates even number of 1s. Now concatenate even number of 1's with 1 to generate odd number of 1's, i.e.,
$$RE = (11)^*.1$$

c. RE to generate a string with any number of zeros, followed by any number of ones but starts with 00
Solution:
$$RE = 00(0+1)^*.$$

d. RE to generate a string with any number of 0's followed by any 1's and ends with 11
Solution:
$$RE = (0+1)^*011$$

e. RE to generate a string with even number of a's followed by odd number of b's
Solution:
$$RE = (aa)^*(bb)^*b.$$

f. RE to generate a string with any number of 0's and 1's and having atleast one pair of consecutive zeros
Solution:
$$RE = (0+1)^*00(0+1)^*.$$

g. RE to generate a string, containing a substring aba
Solution:
$$RE = (a+b)^*aba(a+b)^*.$$

h. RE to generate a string of symbols, having even number of characters
Solution:
$$RE = (aa+ab+ba+bb)^* \quad \text{or} \quad RE = (ab)^*.$$

i. RE to generate a string, containing odd no of a's and odd number of b's
Solution:
$$RE = a(aa)^* \cdot (bb)^*b$$

EXAMPLE 3.3.4: (Forming a regular expression)

a. Set of all words of the form: one 'a' followed by some number of b's (possibly zero)
$$RE \Rightarrow R = ab^*$$

b. Set of all words of the form: some positive number of a's followed by exactly one b
$$RE \Rightarrow R = aa^*b$$

c. Set of all strings of a's and b's that have atleast two symbols, begin and end with one a and have nothing but b's inside
$$RE \Rightarrow R = ab^*a$$

d. Set of all words over $\Sigma = \{a, b\}$
$$RE \Rightarrow (a+b)^* \text{ or } (a^*b^*)^* \text{ or } (\in +a+b)^* \text{ or } (a+b)^+$$

e. Set of even number of x's (possibly zero)
$$RE \Rightarrow R = (xx)^*$$

f. Set of all positive even number of x's
$$RE \Rightarrow R = (xx)^+$$

g. Set of all odd numbers of x's
$$RE \Rightarrow R = (xx)^*x$$

h. Set of all three-lettered words, starting with b over $\Sigma = \{a, b\}$

$$RE \Rightarrow R = baa + bab + bba + bbb.$$

i. Set of all words starting with a and ending with b

$$RE \Rightarrow R = a(a + b)^*b$$

j. Set of all words starting and ending with b

$$RE \Rightarrow (b + b(a + b)^*b)$$

k. Set of all words with exactly two b's

$$RE \Rightarrow R = a^*ba^*ba^*$$

l. Set of all words with atleast two b's

$$RE \Rightarrow R = (a + b)^*b(a + b)^*b(a + b)^*$$

m. Set of all words with atleast one a and atleast one b

$$RE \Rightarrow R = (a + b)^*a(a + b)^*b(a + b)^* + bb^*aa^*$$

n. Set of all words consisting of either all a's or b's, followed by a non-negative number of a's

$$RE \Rightarrow R = a^* + ba^*$$

o. Set of all words with no two consecutive a's,

$$RE \Rightarrow R = (b + ab)^*(a + \epsilon)$$

p. All strings with atleast two consecutive zeros over $\Sigma = \{0, 1\}$

$$RE \Rightarrow R = (0 + 1)^*00(0 + 1)^*$$

q. All strings that end in double letters over $\Sigma = \{a, b\}$

$$RE \Rightarrow R = (a + b)^*(aa + bb)$$

r. All strings that do not end in double letter, over $\Sigma = \{a, b\}$

$$RE \Rightarrow R = \epsilon + a + b + (a + b)^*(ab + ba)$$

s. All strings that have exactly one double letter in them, over $\Sigma = \{a, b\}$

$$RE \Rightarrow R = (\epsilon + b)(ab)^*aa(ba)^*(\epsilon + b) + (\epsilon + a)(ba)^*bb(ab)^*(\epsilon + a)$$

t. All strings that do not end with ab, over $\Sigma = \{a, b\}$

$$RE \Rightarrow R = (a + b)^*(a + bb)$$

u. All strings over $\Sigma = \{a, b\}$ that contain, not more than one occurence of the string aa

$$RE \Rightarrow R = (b + ab)^*a(b + ba)^*$$

3.4 Languages Associated with Regular Expressions

A language represented by a regular expression defines a regular language. In other words, a regular language is a language that can be represented by a regular expression. If 'R' is a regular expression, then $L(R)$ denotes the language associated with R. This language is defined formally as follows.

3.4.1 *Definition*

The language $L(R)$ denoted by any regular expression R is defined by the following rules:

a. ϕ is a regular expression denoting the empty set.
b. ϵ is a regular expression denoting $\{\epsilon\}$
c. For every $a \in \Sigma$, a is a regular expression denoting $\{a\}$.

EXAMPLE 3.4.1: Describe the language defined by the following regular expressions:

a. $R = ab^*a$

Solution:

The language L, defined by regular expression 'R', is the set of all strings of a's and b's that begin and end with a's and have nothing but b's inside

$$\text{i.e } L(R) = \{aa, aba, abba, abbba, abbbba\}.$$

b. $R = a^*b^*$

Solution:

The language L, defined by regular expression 'R' is the set containing the strings of a's and b's, in which all a's (if any) come before all b's (if any)

$$\text{i.e } L(R) = \{\epsilon, a, b, aa, ab, bb, aaa, aab, abb, bbb, aaa\}.$$

3.5 Properties of Regular Expressions

If R_1 and R_2 are regular expressions, then

a. $L(R_1 + R_2) = L(R_1) \cup L(R_2)$

Example:

If $R_1 = ab^*$, $R_2 = ab^*a$ then $L(ab^* + ab^*a) = L(ab^*) \cup L(ab^*a)$

b. $L(R_1 \cdot R_2) = L(R_1) \cdot L(R_2)$

Example:

If $R_1 = a^*$, $R_2 = aba^*$ then $L(a^* \cdot aba^*) = L(a^*) \cdot L(aba^*)$

c. $L(L(R_1)) = L(R_1)$

d. $L(R_1^*) = (L(R_1))^*$

e. $L(R) \cdot \epsilon = \epsilon \cdot L(R) = L(R)$

3.5.1 *Algebra of Regular Expressions*

Let R, S and T be any arbitrary regular expressions. Then, the following properties are true:

a. $(R+S)+T = R+(S+T)$.
b. $R+R = R$.
c. $R+\phi = \phi+R = R$.
d. $R+S = S+R$.
e. $R\phi = \phi R = \phi$.
f. $R.\epsilon = \epsilon.R = R$.
g. $(RS)T = R(ST)$.
h. $R(S+T) = RS+RT$.
i. $(S+T)R = SR+TR$.
j. $\phi^* = \epsilon^* = \epsilon$.
k. $R^* \cdot R^* = R^* = (R^*)^*$.
l. $R \cdot R^* = R^*R = R^* = \epsilon + RR^*$.
m. $(R+S)^* = (R^*S^*)^* = (R^*+S^*)^*$.
n. $(RS)^* = (R^*S^*)^* = (R^*+S^*)^*$.

3.5.2 *Basic operations of Regular Expressions:*

Let R and S be any two regular expressions.

a. *Concatenation* - RS denoting the set $\{xy | x \in R \text{ and } y \in s\}$

Example:

If $R = \{ab, c\}$ and $S = \{d, ef\}$,

then $RS = \{ab, c\} \cdot \{d, ef\}$

$= \{abd, cd, abef, cef\}$.

b. *Union* - The union $R \cup S$ denotes the set union of R and S.

Example:

If $R = \{ab, c\}$ and $s = \{d, ef\}$

then $R \cup S = \{ab, c\} \cup \{d, ef\}$

$= \{ab, c, d, ef\}$

c. *Kleene closure or star closure* - R^* denotes the smallest superset of R that contains ϵ and is closed under string concatenation. This is the set of all strings, that can be made by concatenating zero or more strings in R.

Examples

(i) $\{ab, c\}^* = \{\epsilon, ab, c, abab, abc, cab, cc, ababab, \ldots.\}$

(ii) $\{01, 2\}^* = \{\epsilon, 01, 2, 0101, 012, 201, 22, 010101, 012012, \ldots.\}$.

EXAMPLE 3.5.1: Form the string set for the regular expression given below.

a. **1^*0**

Solution:

$1^*0 = \{\epsilon, 1, 11, 111, \ldots\} \cdot 0$

$= \{0, 10, 110, 1110, \ldots\}$

b. **00^***

Solution:

$00^* = 0 \cdot \{\epsilon, 0, 00, 000\}$

$= \{0, 00, 000, 0000, \ldots\}$; note that $00^* = 0^+$,

c. **10^*1**

Solution:

$10^*1 = 1 \cdot \{\epsilon, 0, 00, 000, \ldots\} \cdot 1$

$= \{1, 10, 100, 1000, \ldots\} \cdot 1$

$= \{11, 101, 1001, 10001, \ldots\}$

d. **$(100^+)^*$**

Solution:

$(100^+)^* = (10 \cdot \{0, 00, 000, \ldots\})^*$

$= \{100, 1000, 10000, \ldots\}^*$

$= \{\epsilon, 100, 100100, 100100100, 1000, 10001000, \ldots\}$

e. $\mathbf{a^*b^*}$
Solution:

$$a^*b^* = \{\epsilon, a, aa, \ldots\} \cdot \{\epsilon, b, bb, \ldots\}$$
$$= \{\epsilon, b, bb, a, ab, abb, aa, aab, aabb, \ldots\} \text{ or}$$
$$= \{\epsilon, a, aa, b, bb, ab, abb, aab, aabb, \ldots\}$$

f. $\mathbf{(0+1)^*}$
Solution:

$$(0+1)^* = 0^* \cup 1^*$$
$$= \{\epsilon, 0, 00, \ldots\} \cup \{\epsilon, 1, 11, \ldots\}$$
$$= \{\epsilon, 0, 00, 1, 11, \ldots\}$$

g. $\mathbf{(a+c)b^*}$
Solution:

$$(a+c)b^* = ab^* \cup cb^*$$
$$= a\{\epsilon, b, bb, \ldots\} \cup c\{\epsilon, b, bb, \ldots\}$$
$$= \{a, ab, abb, \ldots\} \cup \{c, cb, cbb, \ldots\}$$
$$= \{a, c, ab, cb, abb, cbb, \ldots\}$$

h. $\mathbf{(1+10)^*}$
Solution:

$$(1+10)^* = 1^* \cup (10)^*$$
$$= \{1^*, (10)^*\}$$

i. $\mathbf{(0+1)^*011}$
Solution:

$$(0+1)^*011 = (0 \cup 1)^*011$$
$$= 0^*011 \cup 1^*011$$
$$= \{\epsilon, 0, 00, \ldots\}011 \cup \{\epsilon, 1, 11, \ldots\}011$$
$$= \{011, 0011, 00011, \ldots\} \cup \{011, 1011, 11011, \ldots\}$$
$$= \{011, 0011, 00011, 1011, 11011, \ldots\}$$

j. $\mathbf{(0^* + 1^*)^*}$
Solution:

$$(0^* + 1^*)^* = (0^* \cup 1^*)^*$$
$$= (\{\epsilon, 0, 00, \ldots\} \cup \{\epsilon, 1, 11, \ldots\})^*$$
$$= \{\epsilon, 0, 00, 1, 11, \ldots\}^*$$
$$= \{0^*, (00)^*, 1^*, (11)^*, \ldots\}$$

k. $\mathbf{(0+1)^*\,00\,(0+1)^*}$
Solution:

$$(0+1)^*\,00\,(0+1)^* = 0^* \cup 1^*\,00\,0^* \cup 1^*$$
$$= \{0^*, 1^*\}\,00\,\{0^*, 1^*\}$$
$$= \{0^*00, 1^*00\} \cdot \{0^*, 1^*\}$$
$$= \{0^*000^*, 1^*000^*, 0^*001^*, 1^*001^*\}$$

l. $r = \mathbf{(0+1)} \cdot \mathbf{((0+1)} \cdot \mathbf{(0+1))^*}$
Solution:

$$r = 0\big[(0+1) \cdot (0+1)\big]^* \text{ or } 1\big[(0+1) \cdot (0+1)\big]^*$$
$$= 0\big[0(0+1) \text{ or } 1(0+1)\big]^* \text{ or } 1\big[0(0+1) \text{ or } 1(0+1)\big]^*$$
$$= 0\big[\{00, 01\}, \{10, 11\}\big]^* \text{ or } 1\big[\{00, 01\}, \{10, 11\}\big]^*$$
$$= 0\big[(00)^*, (01)^*, (10)^*, (11)^*\big] \text{ or } 1\big[(00)^*, (01)^*, (10)^*, (11)^*\big]$$
$$= \big\{0(00)^*, 0(01)^*, 0(10)^*, 0(11)^*, 1(00)^*, 1(01)^*, 1(10)^*, 1(11)^*\big\}$$

EXAMPLE 3.5.2: Find the language, for the regular expressions given below:

a. $\mathbf{R = a^*(a+b)}$
Solution:

$$L(R) = L(a^* \cdot (a+b))$$
$$= L(a^*) \cdot L(a+b)$$
$$= (L(a))^* \cdot (L(a) \cup L(b))$$
$$= \{\epsilon, a, aa, aaa, \ldots.\} \cdot \{a, b\}$$
$$L(R) = \{a, aa, aaa, b, ab, aab, aaab, \ldots.\}.$$

b. $\mathbf{R = (a + b)^*(a + bb)}$
Solution:

$$\begin{aligned} L(R) &= L((a+b)^* \cdot (a+bb)) \\ &= L(a+b)^* \cdot L(a+bb) \\ &= (L(a+b))^* \cdot L(a+bb) \\ &= (L(a)^* \cup L(b)^*) \cdot (L(a) \cup L(bb)) \\ &= ((L(a))^* \cup (L(b))^*) \cdot (L(a) \cup L(bb)) \\ &= (\{\epsilon, a, aa, \ldots\} \cup \{\epsilon, b, bb, \ldots\}) \cdot (\{a, bb\}) \\ &= \{\epsilon, a, aa, b, bb, \ldots\} \cdot \{a, bb\} \\ L(R) &= \{a, aa, aaa, bb, abb, bbb, bbbb, \ldots\}. \end{aligned}$$

c. $\mathbf{R = a + b}$
Solution:

$$\begin{aligned} L(R) &= L(a+b) \\ &= L(a) \cup L(b) \\ &= \{a\} \cup \{b\} \\ L(R) &= \{a, b\}. \end{aligned}$$

d. $\mathbf{R = a + b^*}$
Solution:

$$\begin{aligned} L(R) &= L(a + b^*) \\ &= L(a) \cup L(b^*) \\ &= L(a) \cup (L(b))^* \\ &= \{a\} \cup \{\epsilon, b, bb, \ldots\} \\ L(R) &= \{\epsilon, a, b, bb, \ldots\} \end{aligned}$$

e. $\mathbf{R = a^*bc^* + ac}$
Solution:

$$\begin{aligned} L(R) &= L(a^*bc^* + ac) \\ &= L(a^*bc^*) \cup L(ac) \\ &= (L(a^*) \cdot L(b) \cdot L(c^*)) \cup (L(a) \cdot L(c)) \\ &= ((L(a))^* \cdot L(b) \cdot (L(c))^*) \cup (L(a) \cdot L(c)) \end{aligned}$$

$$\begin{aligned} &= (\{\epsilon, a, aa, \ldots\} \cdot \{b\} \cdot \{c\}) \cup (\{a\} \cdot \{c\}) \\ &= (\{b, ab, aab, \ldots\} \cdot \{c\}) \cup (\{ac\}) \\ &= \{bc, abc, aabc, \ldots\} \cup \{ac\} \\ &= \{bc, abc, aabc, \ldots, ac\}. \end{aligned}$$

f. $\mathbf{R = ab^*a}$
Solution:

$$\begin{aligned} L(R) &= L(ab^*a) \\ &= L(a) \cdot L(b^*) \cdot L(a) \\ &= \{a\} \cdot (L(b))^* \cdot \{a\} \\ &= \{a\} \cdot \{\epsilon, b, bb, bbb, \ldots\} \cdot \{a\} \\ &= \{a, ab, abb, \ldots\} \cdot \{a\} \\ &= \{aa, aba, abba, \ldots\} \end{aligned}$$

g. $\mathbf{R = a^*b^*}$
Solution:

$$\begin{aligned} L(R) &= L(a^*b^*) \\ &= L(a^*) \cdot L(b^*) \\ &= (L(a))^* \cdot (L(b))^* \\ &= \{\epsilon, a, aa, \ldots\} \cdot \{\epsilon, b, bb, \ldots\} \\ L(R) &= \{\epsilon, a, aa, b, ab, aab, bb, abb, aabb, \ldots\}. \end{aligned}$$

or

$$L(R) = \{\epsilon, a, b, aa, ab, bb, aaa, aab, abb, bbb, aaa, \ldots\}.$$

EXAMPLE 3.5.3: Give the regular expressions for the following languages:

a. $\mathbf{L(R) = \{a, b, c\}}$
Solution:

$$R = a + b + c \quad \text{because} \quad \begin{aligned} L(R) &= L(a+b+c) \\ &= L(a) \cup L(b) \cup L(c) \\ &= \{a, b, c\}. \end{aligned}$$

b. $\mathbf{L(R) = \{a, b, ab, ba, abb, baa, \ldots\}}$
Solution:

$$R = ab^* + ba^* \quad \text{because} \quad \begin{aligned} L(R) &= L(ab^* + ba^*) \\ &= L(ab^*) \cup L(ba^*) \\ &= \{a\} \cdot \{\epsilon, b, bb\} \cup \{b\} \cdot \{\epsilon, a, aa\} \\ &= \{a, ab, abb\} \cup \{b, ba, baa\} \\ &= \{a, b, ab, ba, abb, baa\} \end{aligned}$$

c. $\mathbf{L(R) = \{\epsilon, a, abb, abbbb, \ldots\}}$
Solution:

$$R = \epsilon + a(bb)^* \quad \text{because} \quad \begin{aligned} L(R) &= L(\epsilon + a(bb)^*) \\ &= L(\epsilon) \cup L(a)(L(bb))^* \\ &= \{\epsilon\} \cup (\{a\} \cdot \{\epsilon, bb, bbbb\}) \\ &= \{\epsilon\} \cup \{a, abb, abbbb\} \\ &= \{\epsilon, a, abb, abbbb\} \end{aligned}$$

d. $\mathbf{L(R) = \{0, 1, 10, 11, 100, 101, 110, 111, \ldots\}}$
Solution:

$$R = 0 + 1(0 + 1)^* \quad \text{because} \quad \begin{aligned} L(R) &= L(0 + 1(0 + 1)^*) \\ &= L(0) \cup (L(1) \cdot L(0 + 1)^*) \\ &= \{0\} \cup (\{1\} \cdot L(0^*) \cup L(1^*)) \\ &= \{0\} \cup (\{1\} \cdot \{\epsilon, 0, 00\} \cup \{\epsilon, 1, 11\}) \\ &= \{0\} \cup (\{1\} \cdot \{\epsilon, 0, 00, 1, 11\}) \\ &= \{0\} \cup \{1, 10, 100, 11, 111\} \\ &= \{0, 1, 10, 100, 11, 111\} \end{aligned}$$

e. $\mathbf{L(R) = \{aa, ab, ba, bb\}}$
Solution:

$$R = aa + ab + ba + bb$$

or

$$R = (a + b)(a + b)$$

f. $\mathbf{L(R) = \{a^{2n}b^{2m+1} | n \geq 0, m \geq 0\}}$.
This language denotes set of all strings with an even number of a's followed by an odd number of b's

$$\Rightarrow R = (aa)^*(bb)^*b.$$

g. If $\Sigma = \{0, 1\}$ and $L(R) = \left\{ w \in \Sigma^* \;\middle|\; \begin{array}{l} w \text{ has atleast one pair of} \\ \text{consecutive zeros} \end{array} \right\}$.

$$\Rightarrow R = (0 + 1)^*00(0 + 1)^*.$$

h. If $\Sigma = \{0, 1\}$ and $L(R) = \left\{ w \in \Sigma^* \;\middle|\; \begin{array}{l} w \text{ has any number of 0's or 1's} \\ \text{and ends with 011} \end{array} \right\}$.

$$\Rightarrow R = (0 + 1)^*011.$$

i. If $\Sigma = \{0, 1, 2\}$ and $L(R) = \left\{ w \in \Sigma^* \;\middle|\; \begin{array}{l} w \text{ has 0's 1's and 2's with atleast} \\ \text{one zero followed by atleast one 1} \\ \text{followed by atleast one 2} \end{array} \right\}$.

$$\Rightarrow R = 00^*11^*22^*.$$

j. If $\Sigma = \{a, b\}$ and $L(R) = \left\{ w \in \Sigma^* \;\middle|\; \begin{array}{l} w \text{ has } a\text{'s and } b\text{'s having} \\ \text{a substring } aa \end{array} \right\}$.

$$\Rightarrow R = (a + b)^*aa(a + b)^*.$$

EXAMPLE 3.5.4:

a. **Find the shortest string that is not in the language, represented by the regular expression $a^*(ab)^*b^*$.**
Solution:

Let $R = a^*(ab)^*b^*$

$$\begin{aligned} L(R) &= L(a)^* \cdot L(ab)^* \cdot b^* \\ &= \{\epsilon, a, aa\} \cdot \{\epsilon, ab, aab, \ldots\} \cdot \{b\} \\ &= \{b, ab, aab, abb, aabb, aaabb, aabb, aaabb, aaaaabb\} \end{aligned}$$

It is seen that string ϵ, a, b are with length, one or less, in the language. The strings with length two are aa, bb, ab. However, ba is not in the language $L(r)$. Hence, 'ba' is shortest string, not in the language.

b. **For the two regular expressions given below,**
   i. **find a string corresponding to r_2 but not to r_1 and**
   ii. **find a string corresponding to both r_1 and r_2.**

$$r_1 = a^* + b^* \qquad r_2 = ab^* + ba^* + b^*a + (a^*b)^*$$

Solution:

i. Any string consisting of only a's or only b's and the empty string are in r_1. So, we need to find strings of r_2 which contain atleast one a and atleast one b. For example, ab and ba are such strings.

ii. A string, corresponding to r_1, consists of only a's or only b's or the empty string. The only strings corresponding to r_2, which consist of only a's or b's are – a, b and the strings consisting of only b's (from $(a^*b)^*$). r_2 does not contain the empty string.

3.6 Uses of Regular Expressions

a. In general, the application of regular expressions can be classified as follows:
   - *Validation:* Checking the correctness of inputs i.e., whether the given string complies with a set of formatted constraints or not.
   - *Search and Selection:* Identification of a subset of items from a larger set, on the basis of a pattern match, i.e., a regular expression (e.g.: use of the grep command) to locate files and lines within files.
   - *Tokenisation:* Conversion of a string of characters into a sequence of words for later interpretation, i.e., the generation of program called *lexical analyser* (generated by the *lex* command) that performs lexical processing of its character input. The first step of compilation called *lexical analysis* is to convert the input from a simple sequence of characters into a list of *tokens* of different kinds–like numerical and string constants, variable identifiers and programming language keywords. Here, the source code for a lex program is a table of regular expressions, coupled with corresponding actions.

b. RE offers a declarative way (algebraic definition) to express set of strings.
   Example: set of strings that consists of single 0 followed by any number of 1s or single 1, followed by any number of 0s i.e. $01^* + 10^*$.

c. RE are used to define languages. They define the languages accepted by finite automata, exactly.

3.6.1 *Regular Expression in Lexical Analysis*

The process of translation by compilers is called *compilation*. This process is quite a complex task and, hence is divided into a number of phases as shown in figure 3.1.

Lexical analysis is the first step of compilation. In this phase, the compiler scans the characters of the source program, one character at a time. When it gets sufficient number of characters to constitute a token of the specified language, it outputs that token.

In order to perform this task, the lexical analyser must know the keywords, identifiers, operators, delimiters and punctuation symbols of the language to be implemented. So, when

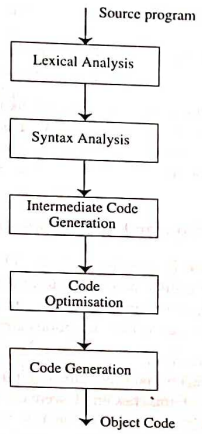

Figure 3.1. *Compilation Process Phases.*

it scans the source program, it will be able to return a suitable token. Therefore, the lexical analyser design must,

a. specify the token of the language, and
b. suitably recognise the tokens.

Therefore, the first thing that is required, is to identify the keywords, identifiers, operators, delimiters and punctuation. These are the tokens of the language. Once identification of the tokens of the language is done, it is important to use suitable notation to specify this tokens. The regular expressions are used to specify this notation. The RE can be used to specify a set of strings and a set of strings that can be specified by using RE notation is called a '*regular set*'. Thus, RE for things like operators, keywords, identifiers etc. of a typical programming language, are as follows:

digit = [0 - 9]
alphabet = [a-z, A-Z]
identifier = [a-z A-Z] [a-z A-Z 0-9]*
keyword = "if" | "else" | "while" | "int" | "float" | "main" | "void"

operators = + | − | ∗ | / | mod | div
boolean = "true" | "false"
comment = " / /" [a-z A-Z]* ("\ r" | "\ n" | "\ r \ n")
whitespace = \ n | \ r | \ t

The advantage of using RE notation for specifying tokens is that, when RE are used, the recogniser for the tokens ends in finite automata (DFA). Thus, designing a lexical analyser becomes a simple process of transforming RE into finite automata and generating the program, for simulating the finite automaton. This process is done by using a software tool called LEX.

3.6.2 *Traditional Unix Regular Expressions (regexps)*

The 'basic' *Unix* regexp syntax has been superceded by *POSIX*, but is still widely used for the purposes of backwards compatibility. Most Unix utilities (grep, sed...) use it by default.

In this syntax, most characters are treated as *literals* - they match only themselves ("*a*" matches "*a*", "*abc*" matches "*abc*", etc). The exceptions are called *metacharacters*:

- `.` Matches any single character
- `[]` Matches a single character that is contained within the brackets - [*abc*], matches "*a*", "*b*", or "*c*". [*a* − *z*], matches any lowercase letter. These can be mixed: [*abcq* − *z*] matches *a*, *b*, *c*, *q*, *r*, *s*, *t*, *u*, *v*, *w*, *x*, *y*, *z*, and so does [*a* − *cq* − *z*].
- `[^]` Matches a single character that is not contained within the brackets - [^*abc*], matches any character other than "*a*", "*b*", or "*c*". [^*a* − *z*], matches any single character that isn't a lowercase letter. As above, these can be mixed.
- `^` Matches the start of the line (or any line, when applied in multiline mode)
- `$` Matches the end of the line (or any line, when applied in multiline mode)
- `\(\)` Marks a part of the expression. The match of enclosed expression can be recalled by *n*, where *n* is a digit from 1 to 9.
- `\n` Matches to the exact string, what the expression enclosed in the *n*'th left parentheses and its pairing right parentheses has been matched to. This construct is theoretically **irregular** and has not been adopted in the extended regular expression syntax.

- A single character expression followed by "*" matches to zero or more copies of the expression. For example, "[xyz]*" matches to "", "x", "y", "zx", "zyx", and so on.
- A *n**, where *n* is a digit from 1 to 9, matches to zero or more iterations of the exact string, what the expression enclosed in the *n*'th left parenthesis and its pairing right parenthesis has been matched to. For example, "\(*a*??\)\ 1" matches to "abcbc" and "adede" but not "abcde".

- An expression enclosed in "\(" and "\)", followed by "*" is deemed to be invalid. In some cases (e.g. /usr/bin/xpg4/grep of SunOS 5.8), it matches to zero or more iterations of the same string, which the enclosed expression matches to. In other cases (e.g. /usr/bin/grep of SunOS 5.8), it matches to what the enclosed expression matches to, followed by a literal "*".

`\{x,y\}` Matches the last 'block', atleast *x* and not more than *y* times. - "*a*\ {3,5\}", matches "*aaa*", "*aaaa*" or "*aaaaa*". Note that this is not found in some instances of regex.

EXAMPLE 3.6.1:

".at" matches any three-character strings, ending with "at".
"[hc]at" matches "hat" and "cat".
"[^b]at" matches any three-character strings, ending with "at" and not beginning with '*b*'.
"^[hc]at" matches "hat" and "cat" but only at the beginning of a line.
"[hc]at$" matches "hat" and "cat" but only at the end of a line.

POSIX modern (extended) regexps: The modern "extended" regexp, can often be used with modern Unix utilities, by including the *command line* flag "-E".

POSIX extended regexps are similar in syntax to the traditional Unix regexp, with some exceptions. The following metacharacters are added:

- `+` Match the last 'block' one or more times - "*ba*+" matches "*ba*", "*baa*", "*baaa*" and so on.
- `?` Match the last 'block' zero or one times - "*ba*?" matches "*b*" or "*ba*".
- `|` The choice (or set union) operator: match either the expression before or the expression after the operator - "*abc*|*def*" matches "*abc*" or "*def*".

Also, backslashes are removed: \{...\} becomes {...} and \(...\) becomes (...).

EXAMPLE 3.6.2:

"[hc]+at" matches with "hat", "cat", "hhat", "chat", "hcat", "ccat" etc.
"[hc]?at" matches "hat", "cat" and "at".
"([cC]at | [dD]og)" matches "cat", "Cat", "dog" and "Dog".

Since the characters '(',')', '[',']', '.', '*', '?', '+', '^' and '$' are used as special symbols they have to be 'escaped' somehow, if they are meant literally. This is done by preceding them with '\', which therefore has to be 'escaped' this way, if meant literally.

EXAMPLE 3.6.3:
".\.(\(|\))" matches with the string "*a.*)"

3.7 Exercises

3.7.1 *Regular Languages and Constructions*

1. State whether the following are true or false.
   a. $0^* \cap 1^* = \phi$
   b. $\phi^* = \phi$
   c. $0^*(0^* \cap 1^*) = \{0,1\}^*0^*$
   d. $0^* \cup 1^* = \{0,1\}^*$
2. Prove $((\phi \circ b) \cup (b^* \circ c))$ is a regular language, using the definition 3.2.1.
3. Suppose the current directory has a large (over 80) number of C files (i.e., files ending in the .c extension).
   a. Write a UNIX command to list out all the C files.
   b. Write a UNIX command to copy all the files from the current directory to a subdirectory.
   c. Write a UNIX command to delete all C files from the current directory.

3.7.2 *Regular Expressions*

1. Write a standard regular expression that represents each of the following languages in its shortest form:
   a. ϕ^*
   b. $(0^*1^*)^*$
   c. $(0 \cup 1)^* \circ (0 \cup 1)^*1(0 \cup 1)^*$
   d. $(0 \cup 1)^* \cup 1^* \cup (1 \cup 0)^*$
2. Suppose $\Sigma = \{0,1\}$. Write the shortest regular expression (in terms of number of symbols) for each of the following languages:
   a. The empty language
   b. The language consisting of only the empty string
   c. The language consisting of all strings of length three or more
   d. The language consisting of all strings containing the pattern 010
   e. The language consisting of all strings where 0s may occur only in the even positions

3. Is the set of file names, contained in a given UNIX directory, regular? Justify your answer.

3.7.3 *Applications of Regular expressions*

1. Write an extended *RE* to validate an input that contains (a) exactly four decimal integers, the first of which must be positive and (b) a floating point number possibly with an exponent.
2. Write a *grep* expression to print all the words in the file /usr / dict / words that contain the letters (a) e, l, m and t in alphabetical order and (b) e, t, m, l in any order.
3. How many times does the word "UNIX" appear in the *grep* man page?
4. Write a *grep* expression to search all .html files in the directory public–html for the pattern house.gif.