



Regular Expression

Dr. Nadeem Akhtar

Assistant Professor

Department of Computer Science & IT

The Islamia University of Bahawalpur

PhD – Formal methods in Software engineering

IRISA – University of South Brittany – FRANCE.

Strings and Languages

An **alphabet** is any finite set of symbols.

Examples of symbols: letters, digits, and punctuation.

The set $\{0, 1\}$ is the binary alphabet

ASCII (example of alphabet)

Unicode (100,000 characters from alphabets of different languages around the world).

Strings and Languages

String

A string over an alphabet is a finite sequence of symbols drawn from that alphabet.

“sentence” and “word” synonyms for “string”.

Strings and Languages

Language:

Any countable set of strings over some fixed alphabet

Examples:

- Set of well-defined C programs
- Set of all grammatically correct English sentences

Also **abstract languages** like \emptyset , the empty set or $\{\epsilon\}$, the set containing only the empty string, are languages under this definition.

Strings and Languages

Concatenation:

x and y are strings, concatenation of x and y denoted xy , is the string formed by appending y to x .

Example: $x=\text{dog}$, $y=\text{house}$ then $xy=\text{doghouse}$

Empty String is the identity under concatenation that is for any string s

$$\epsilon s = s\epsilon = s$$

Strings and Languages

Exponentiation

S^0 to be ϵ

For all $i > 0$, S^i is $S^{i-1}S$

since $\epsilon s = s$, it follows that $S^1 = S$

then $S^2 = SS$, and $S^3 = SSS$, and so on

Strings and Languages

prefix: of string S is any string obtained by removing zero or more symbols from the end of S .

Example: ban , banana , and ϵ are prefixes of banana .

suffix: of String S is any String obtained by removing zero or more symbols from the beginning of S .

Example: nana , banana , and ϵ are suffixes of banana .

Substring: of string S is obtained by deleting any prefix and any suffix from S .

Example: banana , nan , and ϵ are substrings of banana .

Strings and Languages

Proper prefixes, suffixes and substrings of a string S are those, prefixes, suffixes, and substrings of S that are not ϵ or not equal to S itself.

Subsequence of S is any string formed by deleting zero or more not necessarily consecutive positions of S . Example: baan is a substring of banana

Operations on Languages

Union, Concatenation, Closure

Union of L and M

$$L \cup M = \{s \mid s \text{ is in } L \text{ or } s \text{ is in } M\}$$

Concatenation of L and M

$$LM = \{st \mid s \text{ is in } L \text{ and } t \text{ is in } M\}$$

Kleene closure of L

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

Set of strings you get by concatenating L zero or more times

L^0 “Concatenation of L zero times” and L^i is $L^{i-1}L$

Operations on Languages

Positive closure of L

$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

(same as Kleene closure without the term L^0)

ϵ will not be in L^+ unless it is L itself

Operations on Languages

Example

$$L = \{A, B, \dots, Z, a, b, \dots, z\} \quad (52)$$

$$D = \{0, 1, \dots, 9\} \quad (10)$$

- (1) LUD is the set of letters and digits – Language of 62 strings of length one, each of which strings is either one letter or one digit.
- (2) LD is the set of 520 strings of length two, each consists of one letter followed by one digit.
- (3) L^4 is the set of all 4-letter strings.
- (4) L^* is the set of all strings of letters, including ϵ , the empty string.
- (5) $L(LUD)^*$ is the set of all strings of letters and digits beginning with a letter.
- (6) D^+ is the set of all strings of one or more digits.

Regular Expression

An algebraic notation that is compact and easy for humans to use and understand.

Regular expressions that describe simple sets of strings can be combined to form regular expressions that describe more complex sets of Strings.

e.g. (*r*, *s*, and *t*) in italics used for regular expression.

Regular expressions built recursively out of smaller regular expressions. Each regular expression *r* denotes a language $L(r)$, which is also defined recursively from the languages denoted by *r*'s sub-expressions.

Regular Expression

BASIS

- (1) ϵ is a regex, and $L(\epsilon)$ is $\{\epsilon\}$, that is, the languages whose sole member is the empty string.**
- (2) If a is the symbol in Σ , then a is a regex and $L(a) = \{a\}$, that is, the language with one string, of length one, with a in its one position.**

Formal definition of Regular Expression

Say that R is a **regular expression** if R is

1. a for some a in the alphabet Σ ,
2. ϵ ,
3. ϕ ,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions,
6. (R_1^*) , where R_1 is a regular expression.

In items 1 and 2, the regular expressions a and ϵ represent the languages $\{a\}$ and $\{\epsilon\}$, respectively.

In item 3, the regular expression ϕ represents the empty language.

In items 4, 5, and 6, the expressions represent the languages obtained by taking the union or concatenation of the languages R_1 and R_2 , or the star of the language R_1 , respectively.

Regular Expression

INDUCTION: Suppose r and s are regular expressions denoting languages $L(r)$ and $L(s)$ respectively

- (1) $(r) \mid (s)$ is a regex denoting the languages $L(r) \cup L(s)$.**
- (2) $(r)(s)$ is a regex denoting the language $L(r) L(s)$.**
- (3) $(r)^*$ is a regex denoting $(L(r))^*$.**
- (4) (r) is a regex denoting $L(r)$.**

We can add additional pairs of parenthesis around the expressions without changing the language they denote.

Regular Expression

PRECEDENCE:

- (1) The unary operator * and + has the highest precedence and is left associative**
- (2) Concatenation has second highest precedence and is left associative**
- (3) + (Union) has lowest precedence and is left associative**

Regular Expression

Example:

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

- (1) The regex $a|b$ denotes the language $\{a, b\}$
- (2) $(a|b)(a|b)$ denotes $\{aa, ab, ba, bb\}$
 $aa | ab | ba | bb$ regex of the same language.
- (3) a^* denotes the language consisting of all strings of zero or more a 's, that is $\{\epsilon, a, aa, aaa, aaaa, \dots\}$
- (4) $(a|b)^*$ denotes the set of all strings consisting of zero or more instances of a or b , that is, all strings of a 's and b 's: $\{\epsilon, a, b, aa, ab, ba, bb, aaa, \dots\}$
- (5) $a|a^*b$ denotes the language $\{a, b, ab, aab, aaab, \dots\}$ that is, the string a and all strings consisting of zero or more a 's and ending in b .

Regular Expression

Regular expression	Language (set of strings)	Informal description
a	$\{“a”\}$	The set consisting of the one-letter string “a”.
ϵ	$\{“”\}$	The set containing the empty string.
s/t	$L(s) \cup L(t)$	Strings from both languages
st	$\{vw \mid v \in L(s), w \in L(t)\}$	Strings constructed by concatenating a string from the first language with a string from the second language. Note: In set-formulas, “ \mid ” reads as “where”.
s^*	$\{“”\} \cup \{vw \mid v \in L(s), w \in L(s^*)\}$	Each string in the language is a concatenation of any number of strings in the language of s .

Regular Expression

st (pronounced “s t”) describes the concatenation of the languages $L(s)$ and $L(t)$, i.e., the sets of strings obtained by taking a string from $L(s)$ and putting this in front of a string from $L(t)$. For example, if $L(s)$ is {“a”, “b”} and $L(t)$ is {“c”, “d”}, then $L(st)$ is the set {“ac”, “ad”, “bc”, “bd”}.

Regular Expression

The language for s^* (pronounced “s star”) is described recursively: It consists of the empty string plus whatever can be obtained by concatenating a string from $L(s)$ to a string from $L(s^*)$. This is equivalent to saying that $L(s^*)$ consists of strings that can be obtained by concatenating zero or more (possibly different) strings from $L(s)$. If, for example, $L(s)$ is $\{“a”, “b”\}$ then $L(s^*)$ is $\{“”, “a”, “b”, “aa”, “ab”, “ba”, “bb”, “aaa”, \dots\}$, i.e., any string (including the empty) that consists entirely of as and bs.

Regular Expression

Note that while we use the same notation for concrete strings and regular expressions denoting one-string languages, the context will make it clear which is meant.

We will often show strings and sets of strings without using quotation marks, e.g., write $\{a, bb\}$ instead of $\{“a”, “bb”\}$. When doing so, we will use ε to denote the empty string, so the example from $L(s^*)$ above is written as $\{\varepsilon, a, b, aa, ab, ba, bb, aaa, \dots\}$.

The letters u , v and w in italics will be used to denote unspecified single strings, i.e., members of some language.

As an example, abw denotes any string starting with ab .

Regular Expression

Shorthands

non-negative integer constants

$(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^*$

- Sequences of letters within square brackets represent the set of these letters. For example, we use $[ab01]$ as a shorthand for $a|b|0|1$.
- Interval notation to abbreviate $[0123456789]$ to $[0-9]$.
- Write $[a-zA-Z]$ to denote all alphabetic letters in both lower and upper case.
- integer constants above, we can write this much shorter as $[0-9][0-9]^*$
- s^* denotes zero or more occurrences of s
- s^+ denotes one or more occurrences of s
- With this notation, we can abbreviate our description of integers to $[0-9]^+$

Regular Expression

It is common that we can have zero or one occurrence of something (e.g., an optional sign to a number)

$s?$ for zero or more occurrence of s (i.e. s/ϵ)

$+$ and $?$ bind with the same precedence as $$.*

Regular Expression

- Algebraic properties of regular expressions

$(r/s)/t = r/s/t = r/(s/t)$	is associative
$s/t = t/s$	is commutative
$s/s = s$	is idempotent
$s? = s/\epsilon$	by definition
$(rs)t = rst = r(st)$	concatenation is associative
$s\epsilon = s = \epsilon s$	ϵ is a neutral element for concatenation
$r(s/t) = rs/rt$	concatenation distributes over
$(r/s)t = rt/st$	concatenation distributes over
$(s^*)^* = s^*$	* is idempotent.
$s^*s^* = s^*$	0 or more twice is still 0 or more
$ss^* = s^+ = s^*s$	by definition

Regular Expression

Example

Variable names. In the programming language C, a variable name consists of letters, digits and the underscore symbol and it must begin with a letter or underscore. This can be described by the regular expression `[a-zA-Z_][a-zA-Z_0-9]*`

Example

An integer constant is an optional sign followed by a non-empty sequence of digits: `[+-]?[0-9]+`

Regular Expression

Example

A floating-point constant can have an optional sign. After this, the mantissa part is described as a sequence of digits followed by a decimal point and then another sequence of digits. Either one (but not both) of the digit sequences can be empty. Finally, there is an optional exponent part, which is the letter e (in upper or lower case) followed by an (optionally signed) integer constant. If there is an exponent part to the constant, the mantissa part can be written as an integer constant i.e., without the decimal point. Some examples:

3.14 -3. .23 3e+4 11.22e-3.

Regular Expression

3.14 -3. .23 3e+4 11.22e-3.

Format can be described by the following regular expression:

```
[+-]? ((([0-9]+. [0-9]*|. [0-9]+)([eE][+-]?[0-9]+)?) | [0-9]+[eE][+-]?[0-9]+)
```

This regular expression is complicated by the fact that the exponent is optional if the mantissa contains a decimal point, but not if it does not (as that would make the number an integer constant). We can make the description simpler if we make the regular expression for floats also include integers, and instead use other means of distinguishing integers from floats. If we do this, the regular expression can be simplified to

```
[+-]?((([0-9]+(. [0-9]*)?)|. [0-9]+)([eE][+-]?[0-9]+)?)
```

Regular Expression

Example:

The set of valid C identifiers.

letter_ stand for any letter or the underscore

digit_ stand for any digit

letter_(letter_ | digit)*

| used means Union

Parenthesis are used to group sub-expressions

Star means “zero or more occurrences of”

Regular Expression

Regular Definitions

If Σ is an alphabet of basic symbols, then a regular definition is a sequence of definitions of the form:

$d_1 \rightarrow r_1$

$d_2 \rightarrow r_2$

$d_3 \rightarrow r_3$

.....

$d_n \rightarrow r_n$

where

(1) Each d_i is a new symbol, not in Σ and not the same as any other of the d 's and

(2) Each r_i is a regex, over the alphabet $\Sigma \cup \{d_1, d_2, d_3, \dots, d_{i-1}\}$.

By restricting r_i to Σ and previously defined d 's, we avoid recursive definitions, we construct a regex over Σ alone for each r_i .

Regular Expression

Example:

C identifiers are strings of letters, digits, and underscores

letter_ $\rightarrow A|B|,...|Z|a|b|,...|z|_$

digit $\rightarrow 0|1|,...|9$

id $\rightarrow \text{letter_}(\text{letter_} | \text{digit})^*$

Regular Expression

Example:

Unsigned numbers (integers or floating point) e.g.
5280, 0.01234, 6.336E4, or 1.89E-4

digit $\rightarrow 0|1|,\dots,|9$

digits $\rightarrow \text{digit digit}^*$

optionalFraction $\rightarrow .\text{digits} | \epsilon$

optionalExponent $\rightarrow (\epsilon(+|-|\epsilon) \text{ digits}) | \epsilon$

number $\rightarrow \text{digits optionalFraction optionalExponent}$

Regular Expression

Describe the languages denoted by the following Regular expression: (where $\Sigma = \{a, b\}$)

(1) $a(a \mid b)^*a$

(2) $((\epsilon \mid a)b^*)^*$

(3) $(a \mid b)^*a(a \mid b)(a \mid b)$

(4) $a^*ba^*ba^*ba^*$

(5) $(aa \mid bb)^*((aa \mid bb)(aa \mid bb)^*(ab \mid ba)(aa \mid bb)^*)^*$