# AUTOMATA FORMAL LANGUAGES AND LOGIC


## Lecture notes

## Regular Expression in Practice

**Prepared by:**

**Kavitha K N**

**Assistant Professor**

**Department of Computer Science & Engineering**

**PES UNIVERSITY**

**(Established under Karnataka Act No.16 of 2013)**

**100-ft Ring Road, BSK III Stage, Bangalore – 560 085**

## Table of Contents:

## Regular Expression in practice

### 1. Introduction

Regular expressions focus on solving a simple problem that is defining strings to match other strings. In simple terms a regular expression is equivalent to locating some substring within a document . For Example,expression is a valid regular expression that would match the word “expression ” in this sentence.Despite that the ability of regular expressions are further away than matching a single word.

Regular expressions can be used to match complex patterns of symbols or validate the format of input data for example your PAN card number, Adhaar, Credit card number and so on. The Programmers use regex all the time to find out what variable they are using and in what part of the program. Various programming languages provide the module to help write efficient code for example re module in python and Java String Basically regular expressions are used in search tools. For example To search a file in unix we use grep command and so on.

### 2. Special Characters

Since the regular expression can do more than just a search of a literal piece of a text we have few special characters reserved for the purpose of special use. Following are the meta characters that we will be discussing in detail :

| # | Metacharacter | Short Description |
|---|---|---|
| 1 | . | Matches any single character except newline |
| 2 | * | Repeats preceding pattern 0 or more no. of times |
| 3 | + | Repeats preceding pattern 1 or more no. of times |
| 4 | ? | Makes preceding pattern optional |
| 5 | [ ] | Matches a single character from the one’s specified in [ ] |
| 6 | ^ | Anchor for the start of the String or Negation symbol if used in [ ] |
| 7 | $ | Marks end of the String |
| 8 | { } | Specifies no. of repetitions to be made |
| 9 | \d | Matches a single digit from [0-9] |

| | | |
|---|---|---|
| 10 | \s | Matches space character |

**Explanation of Metacharacters using Examples:**

a) . (dot )

In regular expressions, the dot or period is one of the most commonly used metacharacters. The dot matches a single character, without caring what that character is. The only exception are newline characters

b) * (star)

It Causes the resulting RE to match 0 or more repetitions of the preceding RE

Example: a* matches { $\lambda$ , a, aa,,aaa, ….}

c) + (plus)

Is a metacharacter that means to match 1 or more repetitions of the preceding RE

Example a+ matches 1 or more no. of a's

d) ? (question mark)

It matches a preceding regex at most once, in effect making it optional.

Example: ab?c will match either the strings "abc" or "ac" because the b is considered optional.

e) [ ] (square braces)

Define a character class to match a single character.it matches a single character. For example , if we have "cat" in a square bracket then it will either match c or a or t. Only a single character.

Example:

-> "[a-c]" is the same as "[abc]" and matches "a", "b" or "c"

-> "[A-Z]" matches any upper-case characters in the alphabet.

f) ^ (caret)

Caret is a special character used in regular expression, represented as cap symbol.caret is the anchor for the start of the string, or the negation symbol if used in a character class.

Example: "^abc" matches "a" at the start of the string.

When a caret appears as the first character inside square brackets, it negates the pattern. Here This pattern matches any character except a or b or c.

g) $ (dollar)

The dollar sign is the anchor for the end of the string.

Example:

-> regex$ : Finds regex that must match at the end of the line

-> "xya$" matches a "a" at the end of a line.

-> "^$" matches the empty string.

h) { } (Curly braces) {m,n} : m to n repetitions:

This allows you to specify how many times a token can be repeated.

The syntax is {min,max},

where min is a positive integer number indicating the minimum number of matches, and max is an integer equal to or greater than min indicating the maximum number of matches.If the comma is present but max is omitted, the maximum number of matches is infinite.

So a{0, } is the same as a*, and

A{1, } is the same as a+. Omitting both the comma and max tells the engine to repeat the token exactly min times.

Example:

"a{2,3}" matches "aa" or "aaa".

a{3} will match a character exactly three times.

**3. Examples of Regular Expressions in Practice**

In this section we will see how to write regular expressions to validate the following:

a) PAN Card
b) AADHAR Number
c) Indian Mobile Number
d) Dates in 2020
e) Email Address

a) **PAN CARD:**

The valid PAN Card number must satisfy the following conditions:

- It should be 10 characters long.
- The first five characters should be any upper case alphabets.
- The next four-characters should be any number from 0 to 9.
- The last(tenth) character should be any upper case alphabet.
- It should not contain any white spaces.

PAN CARD Regex - ^[A-Z]{5}[0-9]{4}[A-Z]$

Input: str = "BNZAA2318J"
Output: true
Explanation:
The given string satisfies all the above mentioned conditions.

Input: str = "23ZAABN18J"
Output: false
Explanation:
The given string does not start with upper case alphabets, therefore it is not a valid PAN Card number.

b) **AADHAR NUMBER**

The valid Aadhar number must satisfy the following conditions:

- It should have 12 digits.
- It should not start with 0 and 1.
- It should not contain any alphabet and special characters.
- It should have white space after every 4 digits.

The regular expression is:
^[2-9]{1}[0-9]{3}\\s[0-9]{4}\\s[0-9]{4}$

It says

- The Aadhar number should start with a digit between 2 to 9 followed any 3 digits between 0 to 9 followed by \s, since \ is having a special meaning in RE, its special meaning is escaped by escape sequence,
- Followed by four digits between 0 to 9
- Followed by \s

- Followed by four digits between 0 to 9 $ says it should end with a digit
- So this is one of the regular expression to validate the Aadhar number

Input: str = “3675 9834 6012”
Output: true
Explanation: The given string satisfies all the above mentioned conditions. Therefore it is a valid Aadhar number.

Input: str = “3675 9834 6012 8”
Output: false
Explanation: The given string contains 13 digits. Therefore it is not a valid Aadhar number.

**c) Regular expression to validate Indian Mobile Number**

The valid Mobile number must satisfy the following conditions:

- It is a 10 digits number
- The first digit should contain numbers between 6 to 9.
- The remaining 9 digits can contain any number between 0 to 9.
- The mobile number can have 11 digits also by including 0 at the starting.
- The mobile number can be of 13 digits also by including +91 at the starting

MOBILE Number Regex - ^(0|”+91”)?[6-9]\d{9}$

Input: str = “7080059987”
Output: true
Explanation: The given string satisfies all the above mentioned conditions. Therefore it is a valid Indian mobile number.

Input: str = “2457864578”
Output: false
Explanation: The given string contains 2 as first digit. Therefore it is not a valid Indian Mobile number.

**d) The regular expression to validate the date in the format mm/dd/yy**
**Regular expression to validate Dates in the year 2020**

- It is a leap year
- Let the format be DD-MM-YY

- Months with 31 days are : Jan, Mar, May, July, Aug, Oct, Dec
  : [1, 3, 5, 7, 8, 10, 12]
- Months with 30 days are : Apr, June, Sep, Nov
  : [4, 6, 9, 11]
- Month with 29 days : Feb
  : [2]
- Year : 2020

Regular expression for date:
(0[1-9]|[10-31]”-”(0[13578]|1[0|2])|[1-30]”-”(0[469]|11)|[1-29]”-””02”)”-””2020”

Input: str = “4-11-2020”
Output: true
Explanation: The given string satisfies all the above mentioned conditions. Therefore it is a valid date in the year 2020

Input: str = “42-78-2020”
Output: false
Explanation: The given string contains digits which does not satisfy the given condition. Therefore it is not a valid date in the year 2020

**e) Regular expression to validate Email address**

An email is a string (a subset of ASCII characters) separated into two parts by @ symbol. a "personal_info" and a domain, that is personal_info@domain.
The length of the personal_info part may be up to 64 characters long and domain name may be up to 253 characters.

The personal_info part contains the following ASCII characters.

- Uppercase (A-Z) and lowercase (a-z) English letters.
- Digits (0-9).
- Characters ! # $ % & ' * + - / = ? ^ _ ` { | } ~
- Character . ( period, dot or full stop) provided that it is not the first or last character and it will not come one after the other.
- The domain name [for example com, org, net, in, us, info] part contains letters, digits, hyphens, and dots.

Example of valid email id

- mysite@ourearth.com
- my.ownsite@ourearth.org

- mysite@you.net

Example of invalid email id

- mysite.ourearth.com [@ is not present]
- mysite@.com.my [ tld (Top Level domain) can not start with dot "." ]
- @you.me.net [ No character before @ ]

Regular Expression:
^[_a-zA-Z0-9-]+(\.[_a-zA-Z0-9-]+)*@[a-zA-Z0-9-]+(\.[a-zA-Z0-9-]+)*\.(([0-9]{1,3})|([a-zA-Z]{2,3})|(com|edu|info|museum|name))$

**f) Regular expression to validate simple URL**

Regular expression:
^((ht|f)tp(s?))\://([0-9a-zA-Z\-]+\.)+[a-zA-Z]{2,6}(\:[0-9]+)?(/\S*)?$

- The URL can start with ht or f followed by tp
- Followed by zero or one s
- Followed by ":" symbol
- Followed by //
- Followed by any alphanumeric character
- Followed by - or .
- Followed by combination of uppercase or lowercase characters from 2 to 6 times
- Followed by : and digits 0 to nine one or more times

**g) Regular expression in JAVA programming language**

Regular expressions are used in validity checking for example :
Does the given input match the pattern specified using the Regular expression. We should use Java String Library(Use input.matches(regexp) for basic RE matching.
The simple JAVA code snippet shown below specifies how JAVA is used to validate the string.

```
public class validate
{
        public static void main(String[] args)
        {
        String regexp = args[0];
        String Input = args[1];
        StdOut.println(input.matches(regexp));
```

```
        }
    }
```

This code snippet takes 2 command line argument, one is the input string that has to be validated, and the other input is the regular expression

Agrs[0] contains the regular expression and args[1] contains the string to be validated against the regular expression

The method input.matches(regexp) compares the input with the regular expression and prints true if the string matches with the regular expression or False otherwise on to the terminal.
If we compile the code withy the command below:
-> java validate "[$+A-Za-z][A-Za-z0-9]*" ident123

The first argument is the regular expression to validate the Legal JAVA identifier and the second argument is one of the valid JAVA identifiers.
The regular expression says the identifier can start with any combination of uppercase or lowercase letters any number of times followed by combination of anu uppercase, lowercase letters or digits from 0 to 9
The given input string is a valid identifier so if prints TRUE on to the console.

-> java validate "[a-z]+@([a-z]+\.)+(edu/com)" abcd.pes.edu

This example is to validate the simple email address.The regular expression says the email address should start with the combination of any lowercase characters (at least one character should be there)followed by @ character and followed by the combination of any lowercase characters (at least one character should be there) followed by edu or com. The given input string is a valid email id so the code outputs true on to the terminal.