

Regular Expressions

Dylan Lane McDonald

CNM STEMulus Center
Web Development with PHP

November 11, 2015

Outline

1 Introduction to Regular Expressions

- Theoretical Foundations
- Finite State Machines

2 Using Regular Expressions

- Common Syntax
- JavaScript
- PHP

What are Regular Expressions?

Definition

A **regular expression** is a pattern that represents all the possible permutations of a string.

What are Regular Expressions?

Definition

A **regular expression** is a pattern that represents all the possible permutations of a string.

Think of a regular expression as a plan for how strings should look. Informally, a regular expression for a US ZIP code would be, “Five digits, optionally followed by a dash and four digits.” The regular expression doesn’t name all the possible ZIP codes, but gives an acceptance test as to whether a given string is a valid ZIP code. For instance:

- 87102: **Pass**
- 87102-3516: **Pass**
- 871023516: **Fail**
- 8710: **Fail**

Formalizing Regular Expressions

Regular expressions fundamentally work with strings, whose atomic unit is a **character**. The set of all possible characters in a string is known as an **alphabet**, denoted by the Greek letter Σ . Regular expressions have three basic operations on alphabets:

- **Separation**: the Boolean **OR**, “either this character or that character”
- **Quantification**: The number of characters expected in this part of the string (e.g., “3 or more digits”)
- **Grouping**: denoted by parentheses, characters in an alphabet are grouped to define the scope and precedence of separation or quantification

A regular expression is simply a sequence of one or more of the preceding operations.

Further Formalizing Regular Expressions

Take an informal regular expression for allowable file names: “Start with an alphabetic character, optionally followed by 0 or more alphanumeric characters, followed by 1 dot, followed by two or three alphabetic characters.”

Further Formalizing Regular Expressions

Take an informal regular expression for allowable file names: “Start with an alphabetic character, optionally followed by 0 or more alphanumeric characters, followed by 1 dot, followed by two or three alphabetic characters.” Notice we have three alphabets:

- 1 $\Sigma_\alpha = \{a, b, c, \dots, z\}$ (alphabetic)
- 2 $\Sigma_\delta = \{0, 1, 2, \dots, 9\}$ (numeric)
- 3 $\Sigma_\cdot = \{.\}$ (dot)

Further Formalizing Regular Expressions

Take an informal regular expression for allowable file names: “Start with an alphabetic character, optionally followed by 0 or more alphanumeric characters, followed by 1 dot, followed by two or three alphabetic characters.” Notice we have three alphabets:

- 1 $\Sigma_\alpha = \{a, b, c, \dots, z\}$ (alphabetic)
- 2 $\Sigma_\delta = \{0, 1, 2, \dots, 9\}$ (numeric)
- 3 $\Sigma_\cdot = \{.\}$ (dot)

Let $*$ denote “0 or more” and $?$ denote “0 or 1.” Now, we can denote this regular expression as:

$$\alpha [\alpha \mid \delta]^* \cdot \alpha^2 \alpha^?$$

Internally, the regular expression engine creates a finite state machine, as depicted in Figure 1.

Finite State Machine

The finite state machine is basically a formal map of how strings are formed. If the string can arrive at the terminal states q_5 or q_6 , we say the string **passes** the regular expression. Otherwise, it gets stuck in the finite state machine and **fails**.

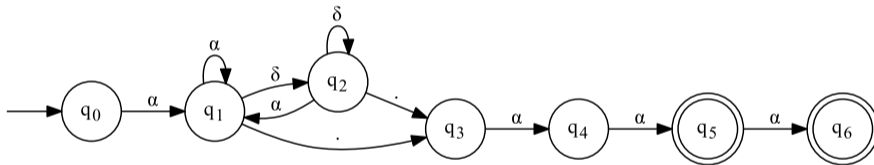


Figure 1: Finite State Machine

All regular expressions can be expressed as finite state machines and all deterministic finite state machines can be expressed as regular expressions.

Character Classes & Operators

Regular expressions come in two major flavors: Perl Compatible Regular Expressions (PCREs) and POSIX. PHP and JavaScript use PCRE syntax.¹ This is a partial list. Use a cheat sheet! [1]

Item	Comment	Item	Comment
.	matches any character	\s	Whitespaces [\t\r\n\v]
\$	end of line	*	Zero or more
^	beginning of line	?	Zero or one
[abc]	Characters a, b, or c	+	One or more
\d	Digits [0-9]	{ <i>m</i> , <i>n</i> }	<i>m</i> or more, no more than <i>n</i>

Table 1: Common Regular Expression Commands

¹PHP has deprecated functions for POSIX regular expressions.

JavaScript Regular Expressions

```
var regex = /^[a-z][\da-z]*\.[a-z]{2}[a-z]?$/;  
var passed = regex.test("foo.js");  
if (passed === true) {  
    console.log("Regular expression passed");  
} else {  
    console.log("Regular expression failed");  
}
```

Listing 1: JavaScript Regular Expression Example

PHP Regular Expressions

```
$regex = "/^[a-z][\da-z]*\.[a-z]{2}[a-z]?$/";  
$passed = preg_match($regex, "foo.php");  
if($passed == 1) {  
    echo "Regular expression passed";  
} else {  
    echo "Regular expression failed";  
}
```

Listing 2: PHP Regular Expression Example

Debugging & Using Regular Expressions

Regular expressions are powerful tools. However, they require a lot of overhead and are inefficient for simple matching and separating. A few more hints to consider when using regular expressions:

- Listings 1 & 2 employ a useful tactic: start the regular expression with a `^` and end it with a `$`. This will strictly ensure what you're trying to match is on a single line and not split on multiple lines.
- Always debug and test your regular expressions thoroughly. [Regex Planet](#) and [PHP Live Regex](#) are useful tools for constructing and testing regular expressions. [2, 3]

Used effectively, regular expressions are the most powerful weapon against malicious and incompetent users.

Cheat Sheet & Tools



Cheatography.

Regular expression cheat sheet.

<http://www.cheatography.com/davechild/cheat-sheets/regular-expressions/pdf/>.



Regex 101.

Regex 101.

<http://www.regex101.com/>.



Philip Bjorge.

Php live regex.

<http://www.phpliveregex.com/>.