

Formal Languages and Compilers Laboratory

Regular Expressions in Practice

Daniele Cattaneo

Material based on slides by Alessandro Barenghi and Michele Scandale

A powerful tool

You have already seen **regular expressions** in the theory classes

However, regular expressions (in short, **regexes**) are **the most useful parsing tool** of them all!

Reasons:

- Widely supported in the industry
 - **Text editors:**
BBEdit, Notepad++, VS Code, JetBrains IDEs, Visual Studio, Xcode, emacs, ...
 - **Programming languages:**
Perl, Python, C++, JavaScript, Rust, Swift, ObjectiveC, Java, ...
 - **Standalone command line tools:**
grep, sed, awk, ...
- Simple to understand, quick to write, reasonably powerful

Quick refresh from the theory

- Regular expression
= **grammar** of a **regular language**
- Regular language
= language recognizable by a **finite state machine**

Chief limitation:

- Finite state machines cannot **count**
- Consequence: cannot parse **grammars with parenthesis** or similar structures
 - Dyck Language
 - $a^n b^n$

Regular Expressions

In the theory classes you have seen the **mathematical foundations** of regular expressions

Mathematical notation for regular expressions is **different** from the syntax accepted by text editors or programming languages

Let's have a look at the standard regular expression syntax used **in practice**

Regular Expressions

Basic character sets:

Syntax	Matches
x	the x character
.	any character except newline
[xyz]	x or y or z
[a-z]	any character between a and z
[^a-z]	any character except those between a and z

Regular Expressions

Composition rules:

Syntax	Matches
R	the R regular expression
RS	concatenation of R and S
$R S$	either R or S
R^*	zero or more occurrences of R
R^+	one or more occurrences of R
$R^?$	zero or one occurrence of R
$R\{m,n\}$	a number of R occurrences ranging from m to n
$R\{n,\}$	n or more occurrences of R
$R\{n\}$	exactly n occurrences of R

Regular Expressions

Regular expression utilities:

Syntax	Matches
(R)	override precedence / capturing group
^R	R at beginning of a line
R\$	R at the end of a line
\t	tab character (just like in C)
\n	newline (just like in C)
\w	a word (same as [a-zA-Z0-9_])
\d	a digit (same as [0-9])
\s	whitespace (same as [\t\r\n])
\W, \D, \S	complement of \w, \d, \s respectively

Examples

	Regex
Number	<code>[0-9]+</code>
Text string*	<code>"[^"]*"</code>
Decimal number	<code>[0-9]+(\.[0-9]+)?</code>
C line comment	<code>//.*\$</code>
Italian Fiscal Code	<code>[A-Z]{6}[A-Z0-9]{9}[A-Z]{1}</code>

Try these at home:

- C-style block comments
(assume there are no * inside)
- YYYY-MM-DD format dates
(with MM between 1 and 12, DD between 1 and 31)

*No quotes allowed in the string!

Regex gone out of control

String with “\” escapes

```
"([^\\""]|\\.)*"
```

C-style block comment with * inside

$$/\backslash^*([^\wedge^*]|\backslash^*+[^{\wedge}/^*])^*\backslash^*+/$$

RFC 5322-compliant e-mail address*

```

\A(?:[a-z0-9!#$%&'*/+=?^_`{|}~ -]+(?:\. [a-z0-9
!#$%&'*/+=?^_`{|}~ -]+)*|"(?:[\\x01-\\x08\\x0b
\\x0c\\x0e-\\x1f\\x21\\x23-\\x5b\\x5d-\\x7f]|\\[\\x01-
\\x09\\x0b\\x0c\\x0e-\\x7f])*")@(?:(?:[a-z0-9](?:
[a-z0-9-]*[a-z0-9])?\\.)+[a-z0-9](?:[a-z0-9-]*
[a-z0-9])?|\\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?
[0-9][0-9]?)\\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?
[0-9][0-9]?|[a-z0-9-]*[a-z0-9]:(?:[\\x01-\\x08
\\x0b\\x0c\\x0e-\\x1f\\x21-\\x5a\\x53-\\x7f]|\\[\\x01-
\\x09\\x0b\\x0c\\x0e-\\x7f]))+\\)]\\z

```

Clean Regular Expressions

Regular expression can describe **simple** concepts:

- Complex structures are typically described by **completely unreadable** regular expression
- If you really need a complex regex, **test it** with one of the tools available online (personally I like www.debuggex.com)

Even with simple concepts is better to keep the regular expression **as clean as possible**

Regular Expressions Are Not Suitable For Input Validation

When using regular expressions the task seems **too complex** or **impossible...**

You need a **real parser!**

Capturing groups

Some editors offer **find and replace** functionality with regular expressions

- 1 Insert parenthesis inside your regular expression (**capturing group**)
- 2 When the regex is matched, the contents of each parenthesis is **captured**
- 3 In the replacement text, you can **insert the captured groups from the original text**

Example:

Change YYYY-MM-DD dates to DD/MM/YYYY

Find: `([0-9]{4})-([0-9]{2})-([0-9]{2})`

Replace: `\3/\2/\1`

Useful UNIX command line tools

grep

Command line: `grep -E <regex>`

Finds all lines in the input that match the specified regex

find

Command line: `find -E . -regex <regex>`

Finds all files under the current directory whose name matches the given regex

sed

Command line:

`sed -Ee s/<regex>/<replacement>/g <filename>`

Reads <filename>, finds all strings matching <regex>, and replaces them with <replacement>. Writes the result to the standard output.

Note: this is just a tiny subset of the capabilities of these tools...
Read the man page!