Discrete Structures for Computing

CSCE 222

Sandeep Kumar

Many slides based on [Lee19], [Rog21], [GK22]

Regex in Practice

- Show use of fd.
- Show use of ugrep.
- Show use of bks.
- Show use of *grep.class*.
  - A line of text that only contains numbers.
  - An HTML hyperlink `<a href="...">...</a>`.
  - A social security number anywhere in a line.
  - The words "credit card" in a line with any number of spaces between the words "credit" and "card".
  - Jack or John.

Regular Expressions

Regular expressions are a concise way to represent *some* sets of strings. These sets are called *regular* languages.

Regular expressions are often used to:

- Validate that some text matches a pattern,
- Find fragments of a text that match some pattern,
- Extract fragments of a text,
- Replace fragments of text with other text.

Regular Expressions (Simplified)

Based on [Fit12].

Char	Meaning
.	matches any single character except newline
*	preceding construct may be repeated ≥ 0 times
+	preceding construct may be repeated ≥ 1 times
?	preceding construct is optional (0 or 1 times)
x	non-meta characters match themselves

Examples:

- `hello` matches *hello*.
- `9+` matches *9*, *99*, *999* etc.
- `99*` matches *9*, *99*, *999* etc.
- `go*gle` matches *ggle*, *gogle*, *google*,...
- `99?` matches *9*, *99*.
- `honou?r` matches *honor*, *honour*.

Regular Expressions (Simplified)...

Char	Meaning
`^`	matches beginning of input (*start of line when multiline*)
`$`	matches end of input (*end of line when multiline*)
`\b`	matches a word boundary
`\B`	matches a non-word boundary
`\A`	matches beginning of string
`\Z`	matches end of string
`X{n}`	$n \times X$

Examples:

- `z{3}` matches *zzz*.
- `\d{5}(-\d{4})?` matches a United States zip code.
- `^dog` begins with *dog*.
- `dog$` ends with *dog*.

Regular Expressions

Character Classes

$$[C_1 C_2 \ldots]$$

where C_i are characters, ranges represented by c-d or character classes.

Char Class	Meaning
`\d`, `\D`	Digits $0 \ldots 9$; its complement
`\w`, `\W`	Word characters $a \ldots z, A \ldots Z, 0 \ldots 9$; its complement
`\s`, `\S`	Spaces ␣`\n\r\t\f\x{B}`; its complement

- `minimi[sz]e` matches minimize & minimise.
- `\d\d\d-\d\d\d-\d\d\d\d` matches 408-243-0836.
- `\d+-\d+-\d+` matches 408-243-0836.
- `[0-9]+-\d+-\d+` matches 408-243-0836.

Regular Expressions

Matching a simplified floating point number

- `[-+]?\d+\.\d+` matches -23.56123.

Regular Expressions

Sequences, Alternatives & Grouping.

Regex	Meaning
$X\ Y$	Any string from X, followed by any string from Y
$X \mid Y$	Any string from X or Y
(X)	Captures the match of X
$(?:X)$	Non-capturing match of X

- `\d+(\s*,\s*\d+)*` matches numbers separated by ",".
- `(abra).*\1` matches *abra* . . . *abra*.
- `\u00f6` matches ?.
- `\u0065` matches e.

Lookarounds

- Lookarounds do not consume anything.
- Even though they have parens, they do not capture.

- **Positive Lookahead**. `Hillary(?=\s+Clinton)` matches Hillary in *Hillary Clinton* but not in *Hillary Makasa*.
- **Positive Lookbehind**. `(?<=http://)\S+` matches URL not including the `http://` part.
- **Negative Lookahead**. `q(?!u)` matches q if not followed by u.
- **Negative Lookbehind**. `(?<![-+\d])(\d+)` matches digits not preceded by a digit, $+$, or $-$.

Java API

- Compile the regular expression with match options.
- Create a `Matcher` object with the string against which the match is done.
- Invoke *matches* or *find* method on the `Matcher` object.

```
String r = "\\d+-\\d+-\\d+";
String s = "408-243-0836";
Pattern regex = Pattern.compile(r, Pattern.CASE_INSENSITIVE);
Matcher m = regex.matcher("408-243-0836");
System.out.printf("'%s' matches %s? %b\n", r, s, m.matches());
m = regex.matcher("foo408-243-0836bar");
if(m.find()) {
    System.out.printf("'%s' matched %s: %s\n", r, s, m.group());
}
```

Splitting String on a Regular Expression

Remember the magic square assignment.

```
String line = "  23 ,   45,67,   78"

line.trim().split("\\s*,\\s*")

Pattern commas = Pattern.compile("\\s*,\\s*")
commas.split(line.trim())
```

Replacing Regular Expression Matches

```
String line = "  23 ,   45,67,   78"

line.trim().replaceAll("\\s*,\\s*", ",")
```

- The replacement string can contain group numbers `$n` or names `${name}`.
- They are replaced with the contents of the corresponding captured group.

```
"3:45".replaceAll("(\\d+):(?<minutes>\\d+)", "HH $1 MM ${minutes}")
```

Regular Expression References

- Java Documentation.
- regular-expressions.info.
- regex101.com.
- Ray Toal's notes.

Bibliography I


Michael Fitzgerald.
Introducing Regular Expressions.
O'Reilly, 2012.


Ashutosh Gupta and S. Krishna.
Cs 228: Logic for computer science 2022.
https://www.cse.iitb.ac.in/ akg/courses/2022-logic/, January 2022.


Hyunyoung Lee.
Discrete structures for computing.
Class slides for TAMU CSCE 222, 2019.


Phillip Rogaway.
Ecs20 fall 2021 lecture notes, Fall 2021.