Regular Expressions

CSC207 Winter 2015

Computer Science
UNIVERSITY OF TORONTO

The general idea

A **regular expression** is a pattern that a string may or may not match.

Example: `[0-9]+`

`[0-9]` means a character in that range
"+" means one or more of what came before

Strings that match: 9125 4
Strings that don't: abc empty string

Symbols like `[`, `]`, and `+` have special meaning. They are not part of the string that matches. (If we want them to be, we escape them with a backslash.)

Example Uses

Handling white space

A program ought to be able to treat any number of white space characters as a separator.

Identifying blank lines

Most people consider a line with just spaces on it to be blank.

Validating input

To check that input is has the expected format (e.g., a date in DD-MM-YYYY format).

Writing code to examine characters one by one is painful!

Regular expressions make it easy.

Uses

It's much easier to declare a pattern that you want matched than to write code that matches it.

Therefore many languages offer support for this.

Bonus: By having the pattern explicitly declared, rather than implicit in code that matches it, it's much easier to:

understand what the pattern is

modify it

Simple patterns

Pattern	Matches	Explanation
a*	'', 'a', 'aa'	zero or more
b+	'b', 'bb'	one or more
ab?c	'ac', 'abc'	zero or one
[abc]	'a', 'b', 'c'	one from a set
[a-c]	'a', 'b', 'c'	one from a range
[abc]*	'', 'a', 'aa', 'acbccb'	combination

Anchoring

Lets you force the position of match

^ matches the beginning of the line

$ matches the end

Neither consumes any characters.

Pattern	Text	Result
b+	abbc	Matches
^b+	abbc	Fails (no b at start)
^a*$	aabaa	Fails (not all a's)

Escaping

Match actual ^ , $, [, etc. using escape sequences, e.g., \^ and \$ and \[

Also use escapes for other characters:

\t is a tab character

\n is a newline

Predefined Character Classes

Construct	Description
.	any character
\d	a digit [0-9]
\D	a non-digit [^0-9]
\s	a whitespace char [\t\n\x0B\f\r]
\S	a non-whitespace char [^\w]
\w	a word char [0-9a-zA-Z_]
\W	a non-word char [^\W]

Character Classes

Construct	Description
[abc]	a, b, or c (simple class)
[^abc]	any char except a, b, or c (negation)
[a-zA-Z]	a through z or A through Z inclusive (range)
[a-d[m-p]]	a through d or m through p (union)
[a-z&&[def]]	d, e, or f (intersection)
[a-z&&[^bc]]	a through z except for b and c (subtraction)
[a-z&&[^m-p]]	a through z and not m through p (subtraction)

Quantifiers

Construct	Description
X?	0 or 1 times
X*	0 or more times
X+	1 or more times
X{n}	exactly n times
X{n,}	at least n times
X{n.m}	at least n but no more than

Capturing Groups and Backreferences

Capturing groups is a way to treat multiple characters as a single unit.

Use **parentheses** to group.

Capturing groups are **numbered** by counting their opening parentheses from left to right.

((A)(B(C))) has the following groups:

1. ((A)(B(C)))
2. (A)
3. (B(C))
4. (C)

Capturing Groups and Backreferences

The section of the input string matching the capturing group(s) is saved in memory for later recall via **backreference**.

A backreference is specified in the regular expression as a backslash (\) followed by a digit indicating the number of the group to be recalled.

For example,

Pattern	Example matching string
(\d\d)\1	1212
(\w*)\s\1	asdf asdf

Regular expressions in Java

The `java.util.regex` package contains:

- `Pattern`: a compiled regular expression
- `Matcher`: the result of a match

Example: `RegexDemo`