

Reguläre Ausdrücke

Handreichung zur Präsentation

Michael Dienert

19. März 2019

Inhaltsverzeichnis

1	Mustersuche in Texten	1
1.1	Reguläre Ausdrücken	1
1.2	Syntax von regulären Ausdrücken	1
1.3	Literale und Metazeichen	1
2	Syntax von Regulären Ausdrücken	2
2.1	Zeichengruppen	2
2.2	Positionsanker	2
2.3	Multiplizität	3
2.4	Gruppierung und Rückreferenz	3
2.5	Character-Classes	3

1 Mustersuche in Texten

1.1 Reguläre Ausdrücke

- Mit regulären Ausdrücken lassen sich Texte nach bestimmten *Mustern* durchsuchen.
- Der reguläre Ausdruck ist selbst eine Zeichenkette, die einer bestimmten Syntax genügen muss.
- Mit einem reguläre Ausdruck kann genau eine bestimmte oder auch eine Menge von Suchmustern beschrieben werden.
- Beispiel: `\?aus` (oder auch `.\?aus`, bei extended-regex s.u.) passt auf 'maus', 'haus', 'Haus', 'raus', aber nicht 'klaus'.
- Die meisten Editoren, Programmiersprachen und viele spezielle Dienstprogramme können reguläre Ausdrücke auswerten.
- Wichtigstes Kommando zum Absuchen von Textdateien: `grep`

1.2 Syntax von regulären Ausdrücken

- Es gibt mehrere Standards für die Syntax von regulären Ausdrücken. Die beiden wichtigsten sind der **Perl**- und der **POSIX**-Standard.
- 1992 hat man den POSIX.2-Standard für reguläre Ausdrücke festgelegt.
- Der POSIX.2-Standard zerfällt wiederum in zwei Teile:
 - BRE: Basic Regular Expression
 - ERE: Extended Regular Expression
- zusammen mit der Programmiersprache `Perl` hat sich seit 1980 der Perl-Regex-Standard entwickelt. Perl-Regexes sind mächtiger und leichter zu lesen als die POSIX-Ausdrücke.
- Perl-Regexes werden als *PCRE - Perl Compatible Regular Expressions* bezeichnet.
- Java verwendet die PCRE (mit ein paar Ausnahmeregelungen).

1.3 Literale und Metazeichen

- Ein regulärer Ausdruck ist, wie schon erwähnt, eine Zeichenkette (String), die *Pattern* genannt wird.
- Patterns bestehen aus zwei Arten von Zeichen:
 - Normale Textzeichen (Literele).

- Metazeichen (Metazeichen). Metazeichen haben eine spezielle Bedeutung, die auch vom verwendeten Standard abhängt.
- Mit vorangestellten \-Zeichen kann man die Sonderbedeutung eines Metazeichens *aus-schalten* und die Metazeichen in Literale umwandeln.
- Z.B. sind die runden Klammern () Metazeichens. Mit \() kann man in einer Zeichenkette nach einer (-Klammer suchen.
- Da das \-Zeichen in Java selbst ein Sonderzeichen ist, muss man ihm in einem Java-Regex-Pattern selbst ein \-Zeichen voranstellen. Alle \-Zeichen im Regex werden in Java **verdoppelt!**

2 Syntax von Regulären Ausdrücken

2.1 Zeichengruppen

[abc] genau *eines* der angegebenen Zeichen.

[a-z] genau *eines* der angegebenen Zeichen, diesmal mit *Bereichsausdruck*.

[a-fu-z] genau *eines* der angegebenen Zeichen

[^abc] genau *ein* Zeichen, das *nicht* 'a', 'b' oder 'c' ist

. ein beliebiges Zeichen ausser *Newline*.

2.2 Positionsanker

^ passt auf die leere Zeichenkette am Anfang einer Zeile. Das ^-Zeichen ist nur ein Metazeichen, wenn es am Anfang eines Regex steht.

\$ passt auf die leere Zeichenkette am Ende einer Zeile. Das \$-Zeichen ist nur ein Metazeichen, wenn es am Ende eines Regex steht.

\< passt auf die leere Zeichenkette am Anfang eines Worts.

\> passt auf die leere Zeichenkette am Ende eines Worts.

Beispiel: alle Wörter *aus* oder *Aus* suchen: \<[aA]us\>

Alternativen: \b passt auf Anfang oder Ende; \s passt auf *whitespace*

2.3 Multiplizität

? Das vorangestellte Objekt tritt 0 oder 1 mal auf.

* Das vorangestellte Objekt tritt 0 mal oder beliebig oft auf (0,1,2,..n).

+ Das vorangestellte Objekt tritt mindestens 1 mal auf (1,2,..n).

{n} Das vorangestellte Objekt tritt genau n mal auf.

{n,} Das vorangestellte Objekt tritt mindestens n mal auf.

{n,m} Das vorangestellte Objekt tritt mindestens n und höchstens m mal auf.

| Passt auf Regex vor oder nach dem |-Zeichen (Ex-OR).

2.4 Gruppierung und Rückreferenz

(**regex**) Passt auf *regex*, jedoch kann *regex* als ganzes modifiziert werden (z.B. mit den Multiplizitätsoperatoren).

\n Rückreferenz auf die vorangehende n-te Gruppe. n muss eine einzelne Ziffer sein (1 - 9).

2.5 Character-Classes

\w Gross- und Kleinbuchstaben, Ziffern und Unterstrich (Wort)

\W **keine** Gross- und Kleinbuchstaben, Ziffern, Unterstrich (Wort)

\d Ziffer (Digit)

\D **keine** Ziffer

\s Whitespace

\S **kein** Whitespace

\d Anfang **oder** Ende

\D nicht Anfang und nicht Ende

\n Neue Zeile