Reguläre Ausdrücke

Theorie

Version vom 19. Februar 2021

1 Einleitung

Ein regulärer Ausdruck (RegExp) ist eine Folge von Zeichen zur Beschreibung von Textstücken (Zeichenketten, Strings).

Reguläre Ausdrücke bilden eine formale Sprache, um Zeichenketten zu beschreiben. Sie sind Bestandteil vieler Textverarbeitungsprogramme und Programmiersprachen wie C++, Java, JavaScript, Perl, Python, ...

Wo werden reguläre Ausdrücke eingesetzt?

Parsing

Identifikation und Extraktion von Textstücken, die gewissen Kriterien genügen.

Beispiel: Aus einer Adressdatei sollen alle Telefonnummern erkannt und in eine separate Datei geschrieben werden.

Suchen

Teilstrings lokalisieren, die mehr als eine Form besitzen.

Beispiel: Suche nach Wörtern, die mit `ge` beginnen *und* die Endung `en` haben. Also:

- `gehen`
- `gelingen`

aber nicht

- `gerne`
- `grunzen`

Suchen und Ersetzen

Die im oben erwähnten Suchvorgang gefundenen Textstücke flexibel durch andere ersetzen.

Zeichenketten auftrennen (splitten)

Eine Zeichenkette an bestimmten Stellen aufspalten.

Beispiel: Einen längeren Text in seine einzelnen Sätze aufteilen.

Gültigkeitsprüfung (Validierung)

Testen, ob eine Zeichenkette gewissen Kriterien genügt.

Beispiele:

- Ist eine E-Mail-Adresse richtig geformt?
- Ist ein Computerprogramm syntaktisch korrekt?

2 Zeichen und Zeichenketten

Normale Zeichen

In erster Linie steht ein Zeichen für sich selbst. Wenn ein regulärer Ausdruck `und` lautet, so „matcht“ er:

- `Hund`
- `undemokratisch`
- `und`
- `Holunder`

aber nicht:

- `Underberg`
- `Gudrun die Grosse`

Reguläre Ausdrücke sind normalerweise *case sensitive*.

Sonderzeichen

Einige Zeichen haben im Rahmen der regulären Ausdrücke jedoch eine besondere Bedeutung und müssen mit einem Backslash (\) maskiert werden, um sie als Literale zu verwenden. Es sind dies:

\	.	^	$	?	+	*
{	}	[	]	(	)	|

Um also einen Punkt oder einen Backslash in einen regulären Ausdruck aufzunehmen, muss man `\.` bzw. `\\` eingeben.

Die Meta-Bedeutung dieser Zeichen werden wir später noch kennen lernen.

Beispiel: Der reguläre Ausdruck für `US$` lautet `US\$`.

3 Zeichenklassen

In vielen Fällen möchte man, dass ein regulärer Ausdruck an einer Stelle nicht auf ein besonderes Zeichen sondern auf eine ganze Menge von Zeichen passt. Dies kann durch eine *Zeichenklasse* bewirkt werden, die aus einer Folge von Einzelzeichen besteht, die von eckigen Klammern (`[...]`) eingeschlossen ist.

Beispiel: Der reguläre Ausdruck `s[ae]gen` erkennt

- `sagen`
- `segen`

aber nicht

- `saegen` (pro Zeichenklasse genau ein Zeichen auswählen)
- `Gösgen`

Übung 1

Wie müsste ein regulärer Ausdruck für eine zweistellige Zahl lauten?

Lösung:

Das ist etwas mühsam! Deshalb besteht die Möglichkeit, bestimmte zusammenhängende Zeichenklassen mit einem Bindestrich etwas kompakter zu schreiben.

Der obige reguläre Ausdruck kann daher etwas kürzer in der Form

`[1-9][0-9]`

dargestellt werden. Entsprechend gibt es für Buchstaben die Abkürzungen `[a-z]` und `[A-Z]`. Aber auch zusammenhängende Teilbereiche wie `[a-f]` oder `[P-T]` sind möglich.

Darüber hinaus sind die Bereiche kombinierbar: `[a-zA-Z]` usw.

Soll ein Zeichen *nicht* aus einer Zeichenklasse kommen, verwendet man als *erstes* Zeichen in der Zeichenklasse den Zirkumflex (`^`).

Beispiel: Der reguläre Ausdruck `an[^tz]en` passt *nicht* auf

- `Tanten`
- `tanzen`

aber auf

- `danken`
- `Tannenbaum`

Der Zirkumflex verneint nur dann, wenn er das erste Zeichen der Zeichenklasse ist. Ab der zweiten Position verliert er seine besondere Wirkung und steht für sich selbst.

Beispiel: `[2-9][^^][2-9]` findet `3*4` aber nicht `3^4`

Für häufig vorkommende Zeichenklassen gibt es Abkürzungen:

Abk.	**passt auf . . .**
.	ein beliebiges (auch leeres) Zeichen
\d	eine Ziffer (digit) [0-9]
\D	keine Ziffer [^\d]*
\s	auf ein Zwischenraumzeichen (space) [␣\t\n\r\v\f]
\S	auf kein Zwischenraumzeichen [^\s]
\w	ein Wortzeichen [a-zA-Z0-9_]*
\W	kein Wortzeichen [^\w]

* Abhängig vom verwendeten Zeichensatz können jeweils weitere Ziffern oder Wortzeichen (z. B. Umlaute) hinzukommen.

Die Zeichenklassen-Abkürzungen können innerhalb oder ausserhalb von Zeichenklassen stehen:

- `[A\d]` erkennt ein Zeichen, wenn es `A` oder eine Ziffer ist.
- `\d-\d` erkennt eine Schulnote der Form `5-6`.

Der Punkt hat jedoch eine unterschiedliche Bedeutung, abhängig davon, ob er innerhalb oder ausserhalb einer Zeichenklasse steht:

- Ausserhalb einer Zeichenklasse steht der Punkt für ein beliebiges Zeichen *ausser* dem Newline-Zeichen (`\n`).
- Innerhalb einer Zeichenklasse steht der Punkt für sich selbst.

Beispiele:

- `\d[.,]\d` erkennt `5.0` oder `0,1`
- `\d.\d` erkennt zusätzlich `7x7` oder `007` usw.

Zur Erinnernung: Möchte man, dass ein regulärer Ausdruck an einer bestimmten Stelle einen Punkt enthält, so kann man diesen auch mit einem Backslash (\) maskieren:

`\d\.\d` erkennt ebenfalls `5.0` oder `3.1`

Übung 2

Damit Binärdaten (01001001) von Menschen „besser gelesen“ werden können, werden sie im 16er-System (Hexadezimalsystem) dargestellt. Da man für das 16er-System auch 16 Ziffern braucht, unser Dezimalsystem aber nur 10 Ziffern kennt, verwendet man die ersten 6 Buchstaben des Alphabets dafür. Gross und Kleinschreibung ist egal.

Dezimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...
Hexadezimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	...

Um eine Hexdezimalzahl von einer Dezimalzahl zu unterscheiden, versieht man sie mit dem Präfix `0x`. *Beispiel:* `0x4A`

Aufgabe: Schreibe einen regulären Ausdruck für eine zweistellige Hexadezimalzahl.

Lösung:

4 Quantoren

Um die Syntax der regulären Ausdrücke kompakter zu machen, können *Quantoren* (Wiederholungsfaktoren) verwendet werden.

Ein Quantor stehen unmittelbar nach einem regulären Ausdruck und hat die allgemeine Form `{m,n}`. Dabei bezeichnet `m` die minimale und `n` die maximale Anzahl von Wiederholungen des regulären Ausdrucks. Wird keine maximale Anzahl von Wiederholungen vorgegeben, so sind „beliebig viele“ Wiederholungen erlaubt. Bei Zeichenklassen kann bei jeder Wiederholung ein beliebiges Zeichen der Zeichenklasse passen.

Beispiele:

- `e{1,2}` passt auf `Nagel` oder `Schnee`
- `[os]{3,}` passt auf `Halloooooo`, `Massstab` oder `grosse`

Für häufig gebrauchte Quantoren gibt es Abkürzungen, die in der folgenden Tabelle zusammengestellt sind:

Abkürzung	**für**	**passt auf ...**
`+`	`{1,}`	mindestens ein Vorkommen
`?`	`{0,1}`	höchstens ein Vorkommen
`*`	`{0,}`	beliebig viele Vorkommen
`{m}`	`{m,m}`	genau `m` Vorkommen

Der `*`-Quantor sollte möglichst vermieden werden, da er oft falsch interpretiert wird. Dazu später mehr.

Übung 3

Schreibe einen regulären Ausdruck, der drei bis sechsstellige ganze Zahlen erkennt, die keine führenden Nullen enthalten.

Lösung:

Gierigkeit

Welchen Zahlstring erkennt der reguläre Ausdruck `2\d+2`, wenn man ihn auf die Ziffernfolge `123512341234` anwendet?

Antwort:

Die Lösung wird verständlich, wenn man weiss, dass Quantoren *gierig* (*greedy*) sind. Dies bedeutet, dass immer die maximal möglich Anzahl an Vorkommen genommen wird.

Der reguläre Ausdruck `\d+` passt auf 1, 2, 3 oder mehr Vorkommen von Ziffern, wählt aber standardmässig so viele, wie er maximal bekommen kann.

Setzt man hinter einen der Quantoren in der obigen Liste ein Fragezeichen (`?`), so wird das gierige Verhalten ins Gegenteil verkehrt und so wenig Vorkommen wie möglich genommen.

Welchen Teilstring erkennt der reguläre Ausdruck `2\d+?2`, wenn man ihn auf die Ziffernfolge `123512341234` anwendet?

Antwort:

Man beachte, dass das zusätzliche Fragezeichen keine Missverständnisse verursacht, da es ja hinter einem Quantor steht und dessen „Gierigkeit“ in „Zurückhaltung“ umwandelt.

5 Alternativen und Gruppen

In der Praxis benötigt man oft reguläre Ausdrücke, die auf eine von mehreren Alternativen passen. Ferner möchte man reguläre Ausdrücke zusammenfassen oder sich in einem regulären Ausdruck auf Teile beziehen, die bereits zuvor erkannt wurden.

Die Gruppenbildung wird mit `(...)` erzielt und Alternativen werden durch das Symbol `|` voneinander getrennt.

Beispiel: `un(fair|tot|ser)`

passt auf:

- `unfair`
- `untot`
- `unser`

Ein gruppierter regulärer Ausdruck ist wieder ein regulärer Ausdruck und kann quantifiziert werden.

Beispiel: `(a|b){1,2}` passt auf

- `a`
- `b`
- `ab`
- `ba`
- `aa`
- `bb`

6 Rückwärtsreferenzen

Wenn das System Rückwärtsreferenzen unterstützt, dann kann man sich mit `\1`, `\2`, `\3`, ... auf die Übereinstimmung in der ersten, zweiten, dritten, ... zurückliegenden Gruppe beziehen. Ein Beispiel soll dies verdeutlichen:

Der reguläre Ausdruck `([1-9])([0-9])\2\1`

erkennt „echte“ vierstellige Zahlenpalindrome:

- 5335
- 1221
- 7007

Er erkennt aber weder `1212` noch `0880`.

7 Weitere Zeichen

Symbol	**passt auf ...**
`^`	einen Zeilenanfang*
`$`	ein Zeilenende
`\b`	eine Zeichenkette am Wortanfang oder Wortende
`\B`	eine Zeichenkette *nicht* am Wortanfang oder Wortende
`\n`	eine Zeilenschaltung im Unix-Format
`\r`	eine Zeilenschaltung im Mac-Format
`\r\n`	eine Zeilenschaltung im Windows-Format

* Nicht mit der Negation `[^...]` von Zeichenklassen verwechseln!

Beispiel: `^$` erkennt eine leere Zeile, die keine Leerzeichen enthält.