

Reguläre Ausdrücke

Jonathan Kleinhellefort <jk@molb.org>

LUG Burghausen

Burghauser Linux-Informationstage 2005

Einführung

Ein Regulärer Ausdruck (**regular expression**, RE) beschreibt eine Menge von Zeichenketten.

Man kann sich einen RE auch als ein **Muster** vorstellen, das auf verschiedene Strings passt.

Beispiele

$a + b +$ *ab, aab, abbb, aaaabbbb, ...*

$[+-]?[0-9]+$ Ganze Zahlen (*-20, 303, +5, ...*)

Geschichte

Das Konzept stammt ursprünglich aus der **theoretischen Informatik**, hat sich allerdings auch in Praxis als nützlich erwiesen.

Eine ähnliche Notation wurde von dem Mathematiker **Stephen Kleene** verwendet, und von **Ken Thompson** für den Unix-Editor **ged** übernommen.

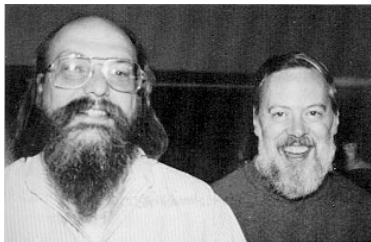


Abbildung: Ken Thompson und Dennis Ritchie

Anwendung

Man verwendet REs, ...

- ▶ ... um Texte nach Mustern zu **durchsuchen** und Informationen herauszufiltern oder zu ersetzen.
- ▶ ... um Dateien einzulesen (**parsen**).
- ▶ ... um das Format von Strings zu **überprüfen**.

Unter Unix nutzen u.a. die Programme Sed, Grep, Awk sowie viele Editoren und Programmiersprachen REs.

Beispiel

Dateien u. Verzeichnisse ausgeben, die mehrere Gigabyte groß sind:

```
du -sh * | grep '[,0-9]*G'
```

Grundregeln

REs bestehen aus zwei Arten von Zeichen: gewöhnliche Zeichen (**Terminale**) und **Metazeichen**.

Terminale

Terminale passen auf sich selbst.

Joker-Zeichen

Der Punkt (.) passt auf jedes Zeichen.

Konkatenation

Schreibt man zwei REs hintereinander, ergibt sich ein RE, der auf die **Konkatenation** von zwei passenden Strings passt.

Auswahl

Man kann auch eine Auswahl an Zeichen angeben:

$[abc]$ eines der Zeichen a , b oder c

$[a-z]$ eines der Zeichen von a bis z

$[\wedge ab]$ nicht a oder b

Achtung: Innerhalb einer Auswahl gibt es nur wenige Metazeichen.

Beispiele

- ▶ Alphanumerisches Zeichen oder Unterstrich:

$[A-Za-z0-9_]$

- ▶ Alle Zeichen außer p , q , r , s (groß oder klein): $[\wedge p-sP-S]$

Alternativen

$ab|cd$ entweder ab oder cd

Achtung: Der senkrechte Strich bindet **schwächer** als die Konkatenation!

Beispiele

- ▶ *Kernighan* oder *Ritchie*: $Kernighan|Ritchie$

Quantoren

Quantoren verändern die Bedeutung des davorstehenden Ausdrucks:

$a?$ macht a optional

$a+$ a muss ein- oder mehrmals vorkommen

a^* a kann null- oder mehrmals vorkommen

$a\{n\}$ a muss n -mal vorkommen

$a\{n, m\}$ a muss n - bis m -mal vorkommen

Beispiel

4-stellige Dezimalzahlen: $[1 - 9][0 - 9]\{3\}$

Vorrang

Wie in der Algebra binden manche Operatoren stärker als andere.

Bindungsstärke von stark nach schwach: Quantoren,
Konkatenation, Alternativen.

Um die Gruppierung zu ändern, kann man Klammern verwenden.

Beispiele

- ▶ $bc+ = b(c+)$
- ▶ $regul(\ddot{a}|ae|a)r = regul\ddot{a}r|regulaer|regular$

Gier

Normalerweise findet die Suche nach einem regulären Ausdruck in einem Text den **ersten** Treffer.

Außerdem findet man den **größten** möglichen Treffer.

Beispiel

Sucht man nach `<.*>` in ein `wenig` html, erhält man `wenig`, und **nicht** ` oder `.

Reguläre Sprachen

Eine **formale Sprache** ist eine Menge von Wörtern über einem **Alphabet**.

Eine **reguläre Sprache** ist eine Sprache, die sich durch einen regulären Ausdruck beschreiben lässt, oder die von einem **deterministischen Automaten** akzeptiert wird.

Beispiel

Alphabet $\Sigma = \{a, b, c\}$

Reguläre Sprache L

$= \{ac, aa, ab, aba, cca, cccc, c\}$ über Σ

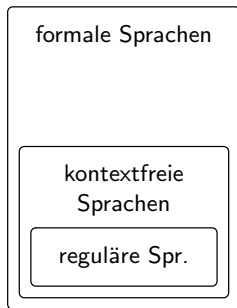


Abbildung: Formale Sprachen

Endliche Automaten

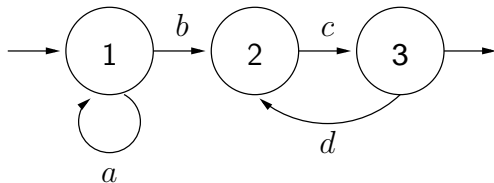


Abbildung: Determinierter endlicher Automat

Ein Endlicher Automat ist mit einem RE gleichmächtig.

Äquivalenter Regulärer Ausdruck: $a * b(cd) * c$

Grenzen von REs

Beispiel

Versuche die Menge aller Wörter zu beschreiben, in der gleich viele b auf a folgen, also ab , $aabb$, $aaabbb$, \dots

Grenzen von REs

Beispiel

Versuche die Menge aller Wörter zu beschreiben, in der gleich viele b auf a folgen, also ab , $aabb$, $aaabbb$, \dots

\implies Das ist unmöglich!

Ausdrücke in Programmiersprachen sind ähnlich rekursiv definiert. Solche Sprachen kann man nicht mit REs beschreiben, dazu benötigt man die mächtigeren **kontextfreien Grammatiken**.

Zum programmieren von etwas komplexeren Parsern sollte man daher keine REs verwenden.

Verschiedene Notationen

Will man ein Metazeichen als Terminal verwenden, so muss man es mit einem Backslash (\) maskieren.

Allerdings wurden in manchen Programmen einige Metazeichen erst *nachträglich eingeführt*. Um Abwärtskompatibel zu bleiben, werden diese genau dann maskiert, wenn man sie als Metazeichen verwendet.

Um diese Inkonsistenz zu beseitigen, wurde ein neuer (POSIX-) Standard für Reguläre Ausdrücke geschaffen.

Daher gibt es nun die „alten“ *Basic Regular Expressions (BREs)*, sowie die neuen POSIX oder *Extended Regular Expressions (EREs)*.

Vergleich BREs/EREs

	Ext. RE	Basic RE
beliebiges Zeichen	.	.
Auswahl	[...]	[...]
Alternative		—
0–∞-malige Wiederholung	*	*
1–∞-malige Wiederholung	+	—
n – m -malige Wiederholung	{ n, m }	\{ n, m \}
Option	?	—
Gruppierung	(...)	\(...\)

Tabelle: Vergleich BREs/EREs

Häufig werden auch die Perl-kompatiblen REs (PCREs) verwendet.

Abkürzende Schreibweisen

POSIX-Zeichenklassen

[*alpha*] alphabetische Zeichen

[*digit*] Ziffern

[*alnum*] Ziffern und Buchstaben

[*space*] Whitespace (Leerzeichen, Tab, ...)

... ..

Viele Programm unterstützen auch noch einige weitere Abkürzungen, z.B. `\w` (alphanumerisches Zeichen) oder Metazeichen (`\<`, `\>` – Wortanfang/Ende).

Maskieren

Man kann Metazeichen auf zwei Arten maskieren:

- ▶ mit einem Backslash
- ▶ in einer Auswahl aus einem Element

Eine Auswahl hat den Vorteil, dass man nicht ausversehen die Bedeutung eines Terminals verändern kann.

Umgebungsquoting

Die Shell und viele Programmiersprachen interpretieren einige der Zeichen in REs selbst. Man sollte daher die REs möglichst stark maskieren (z.B. mit einfachen Anführungszeichen in der Shell).

Metazeichen in einer Auswahl

Da \wedge , $[$, $]$ und $-$ in einer Auswahl eine Sonderbedeutung haben, sollte man folgendes beachten, wenn sie als **Terminal** darin vorkommen sollen:

- ▶ $]$ an den Anfang
- ▶ $[$ und $-$ ans Ende
- ▶ \wedge nicht an den Anfang

Grep

Das Programm `grep(1)` sucht nach einer zu einem regulären Ausdruck passenden Zeile in Dateien oder der Standardeingabe, und gibt diese aus:

```
grep [options] regex [file ...]
```

Eine wichtige Options ist `-E`, mit der man statt nach BREs nach EREs suchen kann.

Beispiele

- ▶ Gigabyte große Verzeichnisse und Dateien finden:
`du -sh * | grep '[,0-9]*G'`
- ▶ In C-Sourcecode nach `FIXME`, `TODD` und `XXX` suchen:
`grep -E 'FIXME|TODD|XXX' *.c`

Sed

Der **Stream Editor** `sed(1)` ist dazu da, Textdaten zu manipulieren.

Sed nimmt Befehle als Argumente oder aus einer Datei entgegen, und führt die angegebenen Kommandos auf einer Datei oder der Standardeingabe aus, und gibt das Ergebnis aus.

Sed benutzt Standardmäßig BREs, kann aber mit der Option `-r` EREs verstehen.

Beispiele

- ▶ Die Mailadresse in einer ganzen Datei ändern:
`sed -e 's/jk@old.org/jk@new.org/g' text >newtext`
- ▶ Gif-Dateien aus Suchresultaten entfernen:
`find . -type f | sed -e '/gif$/d'`

Vi

Der Editor Vi unterstützt wie fast jeder Unix-Editor beim Suchen ebenfalls reguläre Ausdrücke:

```
/⟨regex⟩^M
```

Außerdem gibt es einen Befehl zum Suchen und Ersetzen:

```
:⟨range⟩s/⟨regex⟩/⟨string⟩^M
```

Literatur

- ▶ POSIX Regular Expressions Manual Page: `regex(7)`
- ▶ Helmut Richter: Reguläre Sprache, reguläre Ausdrücke,
`http://www.lrz-muenchen.de/services/schulung/unterlagen/regul/`
- ▶ Obligatory Wikipedia Referenz,
`http://de.wikipedia.org/wiki/Regex`