Reguläre Ausdrücke

Michael Jäger

25. April 2019

Zeichenketten und Sprachen

Ein Alphabet ist eine *endliche* Menge von Symbolen.

Beispiele:

1. $\Sigma_1 = \{\mathtt{0}, \mathtt{1}\}$
2. $\Sigma_2 = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}, \mathtt{h}, \mathtt{i}, \mathtt{j}, \mathtt{k}, \mathtt{l}, \mathtt{m}, \mathtt{n}, \mathtt{o}, \mathtt{p}, \mathtt{q}, \mathtt{r}, \mathtt{s}, \mathtt{t}, \mathtt{u}, \mathtt{v}, \mathtt{w}, \mathtt{x}, \mathtt{y}, \mathtt{z}\}$
3. $\Gamma_1 = \{\mathtt{0}, \mathtt{1}, \mathtt{a}, \mathtt{b}, \$\}$
4. $\Gamma_2 = \{IfStmt, AssignStmt, WhileStmt, DoWhileStmt\}$

   Beachte: $IfStmt$ ist **ein** Symbol!

Wörter

- Sei Σ ein Alphabet. Ein Wort über Σ (auch Zeichenkette, String) ist eine endliche Aneinanderreihung (Sequenz) von Symbolen aus Σ.

  Beispiele:

  - $\texttt{0101100}$ ist ein Wort über $\{\texttt{0}, \texttt{1}\}$
  - $\texttt{abrakadabra}$ ist ein Wort über
    $\{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}, \texttt{e}, \texttt{f}, \texttt{g}, \texttt{h}, \texttt{i}, \texttt{j}, \texttt{k}, \texttt{l}, \texttt{m}, \texttt{n}, \texttt{o}, \texttt{p}, \texttt{q}, \texttt{r}, \texttt{s}, \texttt{t}, \texttt{u}, \texttt{v}, \texttt{w}, \texttt{x}, \texttt{y}, \texttt{z}\}$.

- Die Aneinanderreihung heißt Konkatenation und ist assoziativ:

  $(uv)w = u(vw)$ für alle Wörter u, v, w über einem Alphabet.

- Die Länge eines Worts w ist die Anzahl der Symbole von w.

  Notation: $|w|$

Das leere Wort

- Das Wort der Länge 0 heißt leeres Wort und wird durch ε repräsentiert.

  Anm.: nicht mit dem Leerzeichen ($\sqcup$) oder der leeren Menge ($\emptyset$) verwechseln!

- Das leere Wort ist das neutrale Element bzgl. der Konkatenation:

  $w\varepsilon = \varepsilon w = w$

Notation: Für ein Alphabet Σ sei

$$\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$$

Sprachen

- Σ^* ist die Menge der Wörter über Σ:

  $\Sigma^* = \{w = a_1 \dots a_n \mid n \geq 0, a_1, \dots, a_n \in \Sigma\}$

- Σ^+ ist die Menge der *nichtleeren* Wörter über Σ:

  $\Sigma^+ = \{w = a_1 \dots a_n \mid n > 0, a_1, \dots, a_n \in \Sigma\}$

- $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$

- Eine Sprache L über Σ ist eine Menge von Wörtern über Σ, anders ausgedrückt:

  $L \subseteq \Sigma^*$.

Beispiele für Sprachen

1. $L = \{01, 110110, \varepsilon\}$ ist eine Sprache über $\{0, 1\}$

2. $L = \{\varepsilon\}$ ist eine Sprache über jedem Alphabet.
   Achtung: $\{\varepsilon\} \neq \{\}$

3. Die Menge aller Primzahlen in Dezimalschreibweise ist eine Sprache über $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

   Wortproblem: Ist das Wort 17 in der Sprache?

   Man sieht:
   Sprachen haben mit Berechnungen zu tun!

4. Die Menge aller korrekten Java-(Quelltext-)Programme.

Operationen für Sprachen: Konkatenation zweier Sprachen

Seien L_1 und L_2 Sprachen, dann ist die Konkatenation von L_1 und L_2

$$L_1 L_2 = \{w \mid w = uv, u \in L_1, v \in L_2\}$$

- Beispiel:

  $L_1 = \{ja, aber\}, L_2 = \{ja, NICHT, DOCH\}$

  $L_1 L_2 = \{jaja, jaNICHT, jaDOCH, aberja, aberNICHT, aberDOCH\}$

- Die Konkatenation ist **assoziativ:** $(L_1 L_2)L_3 = L_1(L_2 L_3)$

- Die Sprache $\{\varepsilon\}$ ist neutrales Element bezüglich der Sprachkonkatenation.

Potenzen einer Sprache

Für $i = 0, 1, \ldots$ ist die $i-te$ *Potenz* einer Sprache L

$$L^i = \{w_1 w_2 \ldots w_i \mid w_1, w_2, \ldots, w_i \varepsilon L\}$$

Genauer:

$L^0 = \{\varepsilon\}$

$L^i = LL^{i-1}$, für alle $i > 0$

Beispiel:

Gegeben seien $L = \{ja, aber\}$.

$L^2 = \{jaja, jaaber, aberja, aberaber\}$

$L^3 = \{jajaja, jajaaber, jaaberja, jaaberaber, aberjaja, aberjaaber,$
$aberaberja, aberaberaber\}$

Mengenoperationen für Sprachen

Seien L_1 und L_2 Sprachen über Σ. Die Definitionen der Mengenoperationen Vereinigung, Schnittmenge und Komplementmengen sind auch für Sprachen sinnvoll:

1. Vereinigung

   $L_1 \cup L_2 = \{w \mid w \in L_1 \text{ oder } w \in L_2\}$

2. Schnittmenge

   $L_1 \cap L_2 = \{w \mid w \in L_1 \text{ und } w \in L_2\}$

3. Komplementmenge

   $\overline{L} = \Sigma^* \setminus L$

Kleenesche Hülle

- Kleenesche Hülle von L (Konkateniere 0 oder mehr beliebige Wörter aus L)

  $L^* = \bigcup_{i=0}^{\infty} L^i$

- positive Hülle (Konkateniere 1 oder mehr beliebige Wörter aus L)

  $L^+ = \bigcup_{i=1}^{\infty} L^i$

Anmerkungen:

- L^* besteht aus $\in$ und den Wörtern, die sich in Bestandteile aus L zerlegen lassen.
- Die Kleenesche Hülle nennen wir auch einfach *-Operation, die positive Hülle $+$-Operation
- Zu L^+ gehört $\in$ nur dann, wenn es auch zu L gehört.
- Es gilt: $L^* = L^+ \cup \{\varepsilon\}$

Beispiele

1. $\{0,1\}^+ =$ Menge aller nichtleeren Binärzahlen

2. $\{00,01,10,11\}^* =$ Menge aller Binärzahlen mit einer geraden Ziffernzahl

Reguläre Ausdrücke

Reguläre Ausdrücke sind eine kompakte Schreibweise für die oben definierten Operationen zur Verknüpfung von Sprachen.

Beispiele:

	Ausdruck	**Sprache**
1	xyz	$\{xyz\}$
2	$011\|100$	$\{011\} \cup \{100\} = \{011, 100\}$
3	0^*10^*	$\{0\}^*\{1\}\{0\}^*$
4	$(01)^*$	$(\{0\}\{1\})^* = \{01\}^*$
5	$ja\|nein\|doch$	$\{ja\} \cup \{nein\} \cup \{doch\} = \{ja, nein, doch\}$
6	$(a\|b)^*abb$	$(\{a\} \cup \{b\})^*\{a\}\{b\}\{b\}$
7	$(-\|\varepsilon)(0\|1\|2\|3\|4\|5\|6\|7\|8\|9)^+$	$\{-, \varepsilon\}\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^+$
8	$0(0\|1)^*0 \mid 1(0\|1)^*1 \mid 0 \mid 1$	$\{w \in \{0,1\}^* \mid w$ beginnt und endet mit dem selben Z

Formale Definition regulärer Ausdrücke

Definition 1: Sei Σ ein Alphabet. Die Menge der regulären Ausdrücke über Σ ist induktiv definiert. Jeder reguläre Ausdruck *r* repräsentiert eine Sprache $L(r) \subseteq \Sigma^*$.

1. Für jedes Symbol $a \in \Sigma$ ist a ein regulärer Ausdruck. $L(a) = \{a\}$
2. ϵ ist ein regulärer Ausdruck. $L(\epsilon) = \{\epsilon\}$
3. $\emptyset$ ist ein regulärer Ausdruck. $L(\emptyset) = \emptyset$.
4. Seien r_1 und r_2 reguläre Ausdrücke (über Σ). Dann ist $(r_1 r_2)$ ein regulärer Ausdruck. $L((r_1 r_2)) = L(r_1)L(r_2)$
5. Seien r_1 und r_2 reguläre Ausdrücke. Dann ist $(r_1 \mid r_2)$ ein regulärer Ausdruck. $L((r_1 \mid r_2)) = L(r_1) \cup L(r_2)$
6. Seien r ein regulärer Ausdruck. Dann ist r^* ein regulärer Ausdruck. $L(r^*) = L(r)^*$

Definition 2: Seien *r* und *s* reguläre Ausdrücke über Σ. *r* und *s* heißen **äquivalent** (Notation: *r*=*s*), genau dann, wenn $L(r) = L(s)$.

Anmerkungen

- Den $|$-Operator nennen wir auch ODER-Operator oder Vereinigungs-Operator. (Manchmal auch als $\cup$ notiert.)

- Wie bei den zugrunde liegenden Sprachoperationen definieren wir auch einen $+$-Operator als Schreibabkürzung:

$$r^+ = rr^*$$

Operator-Präzedenz für reguläre Ausdrücke

Unter den Operatoren wird eine Präzedenzfolge festgelegt, ein Operator höherer Präzedenz bindet stärker.

Die höchste Präzedenz haben die Operatoren $*$ und $+$, danach kommt die Konkatenation und schließlich mit der niedrigsten Präzedenz der $|$-Operator.

Für die regulären Ausdrücke p, q, r gilt also

$$\begin{array}{lcl} p \mid qr & = & (p \mid (qr)) \\ pq \mid r & = & ((pq) \mid r) \\ pq^* & = & (p(q^*)) \\ p \mid q^* & = & (p \mid (q^*)) \\ pq^+ & = & (p(q^+)) \\ p \mid q^+ & = & (p \mid (q^+)) \end{array}$$

Beispiel

Welche Sprache repräsentiert der folgende reguläre Ausdruck?

$$r = (a \mid b)^* abb$$

Wir betrachten dazu einige Teilausdrücke:

$L(a \mid b) = L(a) \cup L(b) = \{a\} \cup \{b\} = \{a, b\}$

$L((a \mid b)^*) = (L(a \mid b))^* = \{a, b\}^* =$
$\{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, baa, abb, bab, bba, bbb, aaaa \dots\}$

$L(((((a \mid b)^*)a)b)b) = \{a, b\}^*\{a\}\{b\}\{b\} = \{a, b\}^*\{abb\}$

Die Sprache besteht aus allen Wörtern über $\{a, b\}$, die mit *abb* enden.

Exkurs: UNIX-Notation für reguläre Ausdrücke

- Für den praktischen Umgang mit Programmen, die reguläre Ausdrücke verarbeiten, sind weitere Operatoren nützlich, die eine kompaktere Notation ermöglichen.

- Daneben gibt es auch noch das Problem mit der fehlenden ε-Taste an der Rechnertastatur!

- Wir benutzen die in UNIX-Systemen übliche Schreibweise, wie sie vom Scannergenerator *lex* und von anderen Programmen (*grep*, *egrep*, *sed*, *emacs* usw.) benutzt wird

- Man beachte, dass in den meisten Programmiersprachen auch eine Programmierschnittstelle zur Verwendung regulärer Ausdrücke verfügbar ist (C/C++: *regcomp*, *regexec*, Java: *Pattern* und *Matcher*)

Man unterscheidet Sonderzeichen und normale Zeichen. Sonderzeichen sind z.B. :
`* + [ ] ? ^ ( ) . $`

- ein „normales“ Zeichen steht für sich selbst
- Ein Punkt `.` steht für ein *beliebiges Zeichen* außer `'\n'`
- Falls ein Sonderzeichen nicht als solches interpretiert werden soll, ist ein „Backslash“ `\` voranzustellen
- Auch innerhalb von in Apostrophe eingeschlossenen Strings werden keine Sonderzeichen interpretiert.
- Klammerung erfolgt durch (und)
- Konkatenation zweier reg. Ausdrücke erfolgt ohne expliziten Operator
- Alternativen werden mittels `|` gebildet
- ein nachgestellter `*` steht für beliebige Wiederholung

- ein nachgestelltes `+` steht für nichtleere Wiederholung
- ein nachgestelltes `?` bezeichnet einen optionalen Anteil
- `^` am Anfang eines regulären Ausdrucks steht für Zeilenanfang
- `$` am Ende eines regulären Ausdrucks steht für Zeilenende
- eine **Zeichenklasse** wird mittel eckiger Klammern `[ ... ]` notiert und steht immer für genau ein Zeichen.

  Sie kann durch **Zeichen-Aufzählung** $x_1x_2 \ldots x_n$ und **Bereichsangaben** $x_1 - x_n$ gebildet werden:

  - $x_1 - x_n$ steht für ein Zeichen aus dem Bereich, z.B. `[0-9]`
  - $x_1x_2 \ldots x_n$ steht für ein Zeichen aus der Menge der angegebenen Zeichen, z.B. `[abcx]`
  - beide Schreibweisen können kombiniert werden z.B. `[0-9a-zA-Z_]`
  - Eine `^`-Zeichen am Anfang einer Zeichenklasse `[^...]` spezifiziert die komplementäre Zeichenmenge, z.B. steht `[^0-9]` für ein beliebiges Zeichen außer einer Ziffer

Beispiele:

a) Alle mit kleinem „a“ beginnenden Zeichenketten: `a.*`

b) Alle nichtleeren Dezimalziffernfolgen: `[0-9]+`

c) Alle Wörter, die aus genau 3 Zeichen bestehen und nicht mit einer Ziffer enden: `..[^0-9]`

d) C-Bezeichner = `[_A-Za-z][_A-Za-z0-9]*`

e) C-Float-Literale = `-?[0-9]+((\.[0-9]+)|((\.[0-9]+)?[eE]-?[0-9]+)`

C-Float-Literale bestehen aus

- einem „Vorkomma-Anteil“ (ggf. mit Minuszeichen) (Syntax: `-?[0-9]+`)
- einem optionalen Nachkomma-Anteil (Syntax: `\.[0-9]+`)
- und einem ebenfalls optionalen Exponenten-Anteil (Syntax: `[eE]-?[0-9]+`).

Dabei ist zu beachten, dass entweder der Nachkomma-Anteil oder der Exponent vorhanden sein muss. Wenn beides fehlt, liegt eine Integer-Konstante vor. Daher ist die folgende Spezifikation nicht korrekt:

```
-?[0-9]+(\.[0-9]+)?([eE]-?[0-9]+)?
```

(Viel komplizierter wird es in der Praxis nur selten !)