4 Zeichenfolgen

1. Reguläre Ausdrücke
2. Backus Naur Form

Literatur, Quellen

- Friedl, J.E.F.: Reguläre Ausdrücke - 3. Aufl., Beijing ; Köln [u.a.] : O'Reilly, 2008.
- Online-Test für reguläre Ausdrücke zum einfachen Überprüfen der Übungsaufgaben
  - www.regexr.com
  - regexpal.com

Reguläre Ausdrücke

- Regeln zur Beschreibung von Zeichenketten
- Unterstützung durch zahlreiche Programmiersprachen, Tools
- Bezeichnung:
  - Englisch: regular expression
  - Abgekürzt: Regex
- Einführung
  - Ein regulärer Ausdruck beschreibt eine Menge von Worten durch Regeln
  - Typ-3 in Chomsky-Hierarchie
  - Wird durch endlichen Automat akzeptiert

Reguläre Ausdrücke

- Definition
  - a definiert eine Menge, die nur das Zeichen a als Wort enthält
  - F G definiert Worte, die jeweils aus einem Wort aus $f \in F$ und $g \in G$ zusammengesetzt sind
  - F|G definiert Worte, die aus F oder G stammen
  - Häufige Erweiterungen:

`Y+`	eine nicht-leere Folge von Y
`Y*`	eine evtl. leere Folge von Y
`Y?`	optional Y
`[abc]`	eines der Zeichen a, b oder c
`Y{m,n}`	eine Folge von Y mit mind. m und höchstens n Elementen
`Y{m,}`	eine Folge von Y mit mind. m Elementen
`Y{n}`	eine Folge von Y mit genau n Elementen

Reguläre Ausdrücke

- Beispiele
  - Ausdruck besteht aus beliebig vielen a oder b
    - (a+|b+)
    - (a|b)+
    - [ab]+
  - Ausdruck fängt mit a an und endet mit b
    - a(a|b)*b
  - Beispiel: Reguläre Ausdrücke für Zahlen
    - D = (0|1|2...9) oder D=[0-9]
    - D+
    - D*.D+
    - D+(.D+)?

Reguläre Ausdrücke in der Programmierung

- Einige vordefinierte Zeichenklassen
  - Nicht von allen Implementierungen in gleicher Weise unterstützt

`.`	Beliebiges Zeichen
`\.`	Punkt
`\\`	Backslash
`\n`	Newline
`\d`	Ziffer, identisch mit `[0-9]`
`\D`	Keine Ziffer, identisch mit `[^0-9]`
`\s`	Leerraumzeichen (z.B. Space, Tabulator)
`\S`	Kein Leerraumzeichen, identisch mit `[^\s]`
`\w`	Alphanumerisches Zeichen, identisch mit `[a-zA-Z_0-9]`
`\W`	Kein alphanumerische Zeichen, identisch mit `[^\w]`
`^`	Beginn des Textes
`$`	Ende des Textes

Reguläre Ausdrücke

- Aufgaben
  - Definieren Sie die Anrede einer Person als regulären Ausdruck
  - Definieren Sie eine Mailadresse als regulären Ausdruck
  - Definieren Sie eine Adresse inkl. Strasse Hausnummer, Postleitzahl, Stadt als regulären Ausdruck

- Verwendung regulärer Ausdrücke
  - C, Java, PHP
  - Unix-Skripte wie egrep, sed
  - Webseiten wie z.B. http://www.regexe.de
  - Java siehe nächste Folie

Reguläre Ausdrücke in der Programmierung

- Verwendung regulärer Ausdrücke in Java
  - Klasse String

```
public String[] split(String regex)
public boolean matches(String regex)
public String replaceAll(String regex,
                         String replacement)
```

  - Klasse Pattern

```
static boolean matches(String regex,
                       CharSequence input)
```

  - Beispiel

```
Pattern.matches(
    "[a-zA-Z]{2,20}\.? [0-9]{1,3}[a-zA-Z]?",
    "Brauneggerstr. 55" )
```

Backus Naur Form (BNF)

- Beschreibung
  - Backus Naur Form (BNF) zur Beschreibung von kontextfreien Grammatiken
  - Ersetzungssysteme
  - Definition der Sprache als Menge von Sätzen, die mit Regeln erzeugt werden können
- Anwendung für
  - Programmiersprachen, wie z.B. Java, C
  - Sprachen als Schnittstelle, z.B. HTML, XML
  - Strukturen von Protokollen

HTML-Beispiel

```
<table>
  <tr>
    <td>Mo</td>
    <td>11-13</td>
    <td>AM</td>
  </tr>
  <tr>
    <td>Fr</td>
    <td>11-13</td>
    <td>AM</td>
  </tr>
</table>
```

Definition BNF

- Kontextfreie Grammatik G = (T, N, P, S) besteht aus
  - T — Menge der Terminalsymbole (Terminale) entspricht der Elemente der Sprache
  - N — Menge der Nichtterminalsymbole (Nichtterminale) wird in Regeln definiert
  - $S \in N$ — Startsymbol
  - $P \subseteq N \times V^*$ — Menge der Produktionen, $V = T \cup N$ $(A, x) \in P$ mit $A \in N$, $x \in V^*$ statt $(A, x) \in P$ schreibt man $A ::= x$
- Unterscheidung Terminalsymbole und Nichtterminalsymbole
  - Verwendung von spitzen Klammern für Nichtterminalsymbole
  - Oder: Anführungsstriche für Terminalsymbole

Definition BNF

- Beispiele
  - <DeutscherSatz> ::= <Subjekt> <Prädikat> <Objekt>
    <Subjekt> ::= <Artikel> <Substandiv>
    <Objekt> ::= <Artikel> <Substantiv>
    <Substantiv> ::= "Mann" | "Buch"
    <Artikel> ::= "der" | "die" | "das"
    <Prädikat> ::= "liest"
  - ZifferAußerNull ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9“
  - Ziffer ::= "0" | ZifferAußerNull
  - Klammerung ::= "(" Liste ")"
    Liste ::= Klammerung Liste
    Liste ::=

Ableitungsregel

- Definition Ableitung
  - Sei G=(T, N, P, S) eine kontextfreie Grammatik. Dann kann man mit der Produktion A::=x∈P das Nichtterminal A∈N durch die rechte Seite x ersetzen. Das ist eine **Ableitungsregel**.
  - Mehrere Ableitungsregeln nacheinander angewandt heißen **Ableitung**

- Beispiel
  - Klammerung ::= "(" Liste ")"
    Liste ::= Klammerung Liste
    Liste ::=

Klammerung
⇒(Liste)
⇒(Klammerung Liste)
⇒(Klammerung Klammerung Liste)
⇒(Klammerung (Liste) Liste)
⇒((Liste) (Liste) Liste)
⇒(() (Liste) Liste)
⇒(() () Liste)
⇒(() ())

Beispiele

- Boolesche Term-Bäume
  - T = { true, false, (,), ∧, ∨, ¬, Variable}
    N = BOOL
    S = BOOL
    P = { BOOL ::= Variable
    BOOL ::= true
    BOOL ::= false
    BOOL ::= (BOOL ∧ BOOL)
    BOOL ::= (BOOL ∨ BOOL)
    BOOL ::= ¬BOOL }

Erweiterte BNF

- EBNF - Erweiterung der Backus Naur Form
  - Erweiterung zur besseren Lesbarkeit
- Notation
  - Alternative |
  - Gruppierung ( )
  - Optionaler Inhalt [ ]
  - Beliebige Wiederholung { }
- BNF ohne Erweiterung
  - Zahl ::= PositiveZahl | "-" Positive Zahl
    PositiveZahl ::= (Ziffer PositiveZahl) | Ziffer
- BNF mit Erweiterung
  - Zahl ::= ["-"] { Ziffer }

Beispiele

- HTML-Tabellen
  - Table ::= <table> Rows </table>
    Rows ::= { Row }
    Row ::= <tr> Cells </tr>
    Cells ::= { Cell }
    Cell ::= <td> Text </td>
    Cell ::= <td> Table </td>

- Satz der Tabellen-Grammatik
  - <table>
    <tr>
    <td>Mo</td>
    <td>11-13</td>
    <td>AM</td>
    </tr>
    <tr>
    <td>Fr</td>
    <td>11-13</td>
    <td>AM</td>
    </tr>
    </table>

Erweiterte BNF

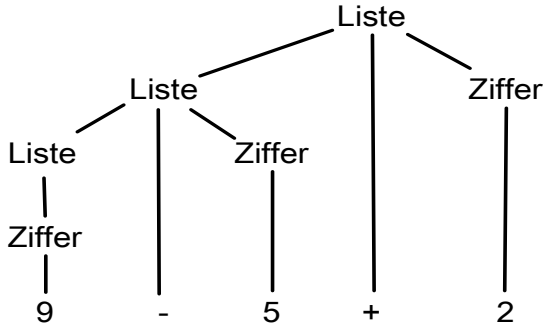
- Beispiel

```
start → expr
expr → expr + term | term
term  →term * fact |  fact
fact  → ID | INT | (expr)
```

Syntaxanalyse und Syntaxbäume

- Parsing: Zeichenkette → Struktur
- Beispiel-Produktionen
  - Liste → Liste + Ziffer
    Liste → Liste – Ziffer
    Liste → Ziffer
    Ziffer → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Parsebaum
(konkreter Syntaxbaum)



Syntaxbaum
(abstrakter Syntaxbaum)