Computergrundlagen
Reguläre Ausdrücke

Olaf Lenz

Institut für Computerphysik
Universität Stuttgart

Wintersemester 2011/12

Motivation

Wozu dienen *reguläre Ausdrücke*?

- Finden von *Mustern* in Dateien
- Komplexeres Suchen und Ersetzen in Dateien (z.B. im Texteditor)

```
"This License" refers to version 3 ...

                    ⇓

\emph{This License} refers to version 3 ...
```

Reguläre Ausdrücke – Definition

- Ein *regulärer Ausdruck* beschreibt ein *Muster*, das auf eine Zeichenfolge *passen* kann (*„matchen“*), oder nicht.
- Formale Definition (Syntax)
  - Die leere Menge ε ist ein regulärer Ausdruck.
  - Jedes Zeichen ist ein regulärer Ausdruck.
  - Wenn x und y reguläre Ausdrücke sind, dann ist auch
    - (xy) ein regulärer Ausdruck (*Verkettung*)
    - $(x|y)$ ein regulärer Ausdruck (*Alternative*)
    - $(x*)$ ein regulärer Ausdruck (*„Kleenesche Hülle“*)
- Semantik
  - a passt auf „a“
  - (ab) passt auf „ab“
  - $((ab)|c)$ passt auf „ab“ oder „c“
  - $((ab)*)$ passt auf „ab“, „abab“, aber auch auf „“
  - $(((ab)|c)*)$ kann als $(ab|c)*$ abgekürzt werden (Operatorpräzendez)

Reguläre Ausdrücke in Python

- `re.match(pattern, string)`
  Testet, ob der reguläre Ausdruck *pattern* am Anfang der Zeichenkette *string* passt
- `re.search(pattern, string)`
  Sucht, ob *pattern* irgendwo in *string* passt
- Reguläre Ausdrücke in Python sind selbst Zeichenketten
- „Metazeichen“ müssen „gequotet“ werden
  `\ ( ) [ ] { } * + . ^ $`

```
>>> import re
>>> if re.match('(ab|c)*', 'abab'): print 'Passt!'
Passt!
>>> if re.search(r'\(1\*2\)', '7+(1*2)'): print 'Passt auch!'
Passt auch!
```

Erweiterte reguläre Ausdrücke

- `'.'` passt auf jedes Zeichen
  `'H.se'` passt auf `'Hase'` oder `'Hose'`
- `'a+'` $\equiv$ `'aa*'` (mindestens einmal)
  `'Hallo+'` passt auf `'Hallo'` oder `'Hallooo'`, aber nicht auf `'Hall'`
- `'a?'` $\equiv$ `'(a|)'` (ein- oder keinmal)
  `'b?engel'` passt auf `'bengel'` oder `'engel'`
- `'a{2,3}'`: zwei- oder dreimal
  `'Hallo{2,3}'` passt auf `'Halloo'` oder `'Hallooo'`
- `'^'` / `'$'`: Anfang / Ende der Zeichenfolge

Zeichenklassen

- `'[abc]'` ≡ `'a|b|c'`
  `'[HM]aus'` passt auf `'Haus'` oder `'Maus'`
- `'[^abc]'`: alle Zeichen *außer* a, b oder c
- `'[a-z]'`: alle Zeichen zwischen a und z
- Spezielle Zeichenklassen:
  - `'\w'`: Alphanumerisches Zeichen („**w**ord-character“)
  - `'\W'`: Alle Zeichen außer alphanumerischen Zeichen
  - `'\s'` / `'\S'`: (alles außer) Leerzeichen („**s**pace“)
  - `'\d'` / `'\D'`: (alles außer) Ziffern („**d**igit“)
  - `'\b'` / `'\B'`: Anfang / Ende eines Wortes

```
>>> floatpattern = r'[+-]?\d*\.\d*(e[+-]?\d+)?'
>>> if re.match(floatpattern, '-1.3e-17'): print 'Float'
Float
>>> if re.match(floatpattern, '17'): print 'Float'
>>> if re.match(floatpattern, '.3'): print 'Float'
Float
```

Suchen und Ersetzen

- `sub(`*`pattern, repl, string`*`)`
  Ersetzt alle Auftreten von *pattern* in *string* durch *repl*
- Klammern („(“ und „)“) in regulären Ausdruck erzeugen *Gruppen*
- Gruppe 0 ist der Treffer des gesamten Musters
- Andere Gruppennummern nach Reihenfolge im Ausdruck
- *repl* kann Referenzen auf Gruppen des reguläre Ausdrucks enthalten (`\1`, `\2`, `...`)

```
>>> import re
>>> s='"This License" refers to version 3 ...'
>>> print re.sub('"(.*)"', r'\emph{\1}', s)
\emph{This License} refers to version 3 ...
```

Rückgabewerte

- `re.match` und `re.search` geben *Match-Objekte* zurück
- Diese ermöglichen, mehr über den Treffer herauszufinden
  - `groups()` ergibt die Treffer der einzelnen Gruppen
  - `group(`*id*`)` ergibt den Treffer von Gruppe *id*
  - `span(`*id*`)` ergibt Tupel mit Anfangs- und Endposition
  - `start(`*id*`)` und `end(`*id*`)`

```
>>> m = re.search('(ab|bc)(c|d)+', 'xxxxabcxxxx')
>>> print m.groups()
('ab', 'c')
>>> print m.group(0), m.group(1), m.group(2)
abc ab c
>>> print m.span(0), m.span(1)
(4, 7) (4, 6)
>>> print m.start(0), m.end(0)
4 7
```

„Greediness“

```
>>> import re
>>> s='"Freedom" and "software" ...'
>>> print re.sub('"(.*)"', r'\emph{\1}', s)
\emph{Freedom" and "software} ...
```

- Muster matchen immer so viel wie möglich („greedy“)
- Zusätzliches ? hinter dem Muster verhindert das

```
>>> import re
>>> s='"Freedom" and "software" ...'
>>> print re.sub('"(.*?)"', r'\emph{\1}', s)
\emph{Freedom} and \emph{software} ...
```

Reguläre Ausdrücke in anderen Programmen

- Alle Programmiersprachen besitzen Implementationen
- Emacs
  - Edit $\rightarrow$ Replace $\rightarrow$ Replace Regexp
  - Tastenkombination `M-C-%`
  - Eingabe von regulären Ausdrücken
  - *fast* so wie in Python
    - `(ab)` $\Rightarrow$ `\(ab\)`
    - `\d` $\Rightarrow$ `[0-9]`
- egrep!