Theoretische Informatik

Reguläre Sprachen
und Automaten

- Reguläre Sprachen
  - Reguläre Ausdrücke und Sprachen
  - Gleichungen
  - Syntaxdiagramme
  - Erweiterungen
  - Anwendungen

Reguläre Ausdrücke über Σ

Syntax:

- Konstante Ausdrücke
  - ∅,
  - ε,
  - a für jedes $a \in \Sigma$
- Sind e und f reguläre Ausdrücke, dann auch
  - e + f
  - e f
  - e*
- Ist e ein regulärer Ausdruck, dann auch
  - (e)

Häufig „|“ statt „+“

Beispiele für $\Sigma = \{0,1\}$: 0, ε, 0(10+01)*1, 0(0+1)*1, (0+1)*0(0+1)*, ...

Beispiele für $\Sigma = \{a,\ldots,z\}$: if, (bla)*, ε, java(ε +script)(bla)*, …

Beispiel für $\Sigma = \{0,\ldots,9\}$:

0 + (1+2+3+4+5+6+7+8+9)(0+1+2+3+4+5+6+7+8+9)*

Vereinbarung: * bindet am stärksten, + am schwächsten

Semantik regulärer Ausdrücke

- Jeder reguläre Ausdruck *e* beschreibt eine Sprache L(*e*):
  - $L(\emptyset) := \{\}$
  - $L(\varepsilon) := \{\varepsilon\}$
  - $L(a) := \{a\}$ für jedes $a \in \Sigma$
- und induktiv:
  - $L(e + f) := L(e) \cup L(f)$
  - $L(e\ f) := L(e) \pm L(f)$
  - $L(e^*) := L(e)^*$

- Beispiele:

$L((0+1)^*\ 0\ (1 + (01)^*))$

$= L(0+1)^*\ L(0)\ (L(1) \cup L(01)^*)$

$= (L(0) \cup L(1))^*\ \{0\}\ (\{1\} \cup (\{0\}\ \{1\})^*)$

$= (\{0\} \cup \{1\})^*\ \{0\}\ (\{1\} \cup \{01\}^*)$

$= \{0,1\}^*\ (\{01\} \cup \{0\}\ \{01\}^*)$

$= L((0+1)^*) - L((0+1)^*(11)) - \{1,\varepsilon\}$

$L(e + \emptyset) = L(e) \cup L(\emptyset)$

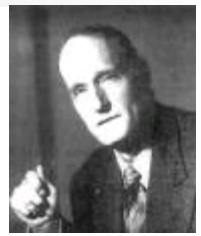
$= L(e) \cup \{\}$

$= L(e)$

$L(e\ \emptyset) = L(e)\ L(\emptyset) = L(e)\{\} = \{\}$
aber
$L(e\ \varepsilon) = L(e)\{\varepsilon\} = L(e)$

Vorsicht: - ist kein regulärer Operator !

Gleichungen



Stephen Kleene

Definition: $r_1 = r_2 \;:\Leftrightarrow\; L(r_1) = L(r_2)$

- Es gelten u.a.

$$\emptyset + e = e$$
$$e + f = f + e$$
$$(e + f) + g = e + (f + g)$$

$$\varepsilon\, e = e = e\, \varepsilon$$
$$(e\, f)\, g = e\, (f\, g)$$

$$e(f + g) = ef + eg$$
$$(e + f)g = eg + fg$$

$$\varepsilon^* = \varepsilon$$
$$(e^*)^* = e^*$$
$$(\varepsilon + e)^* = e^*$$
$$(e^* f^*)^* = (e+f)^*$$
$$(ef)^* e = e\,(fe)^*$$

- aber auch:

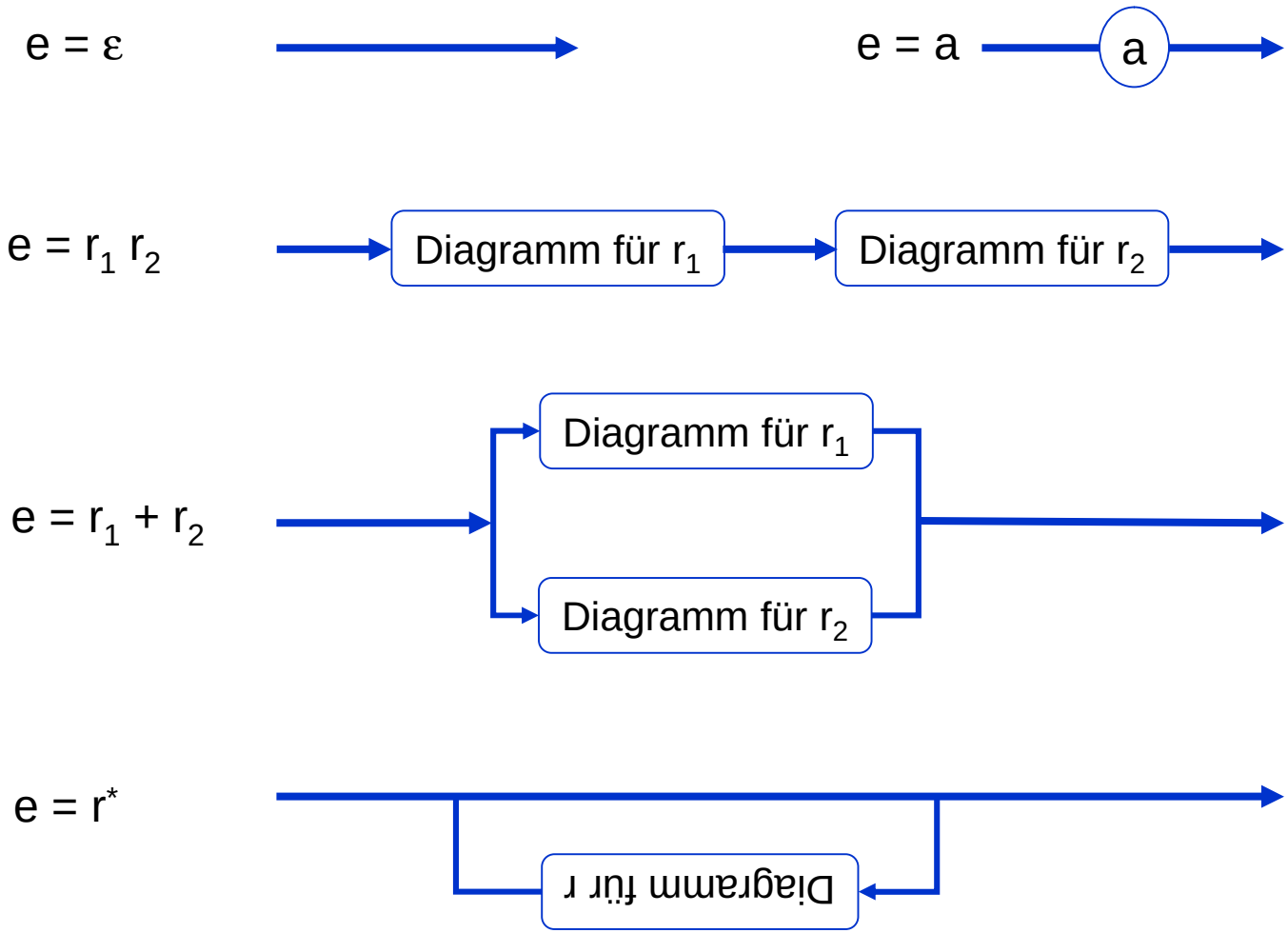
$$(e\, f + e)^*\, e = e\, (f\, e + e)^*$$

Herleitung

$$
\begin{aligned}
(e\,f + e)^*\,e &= (e\,f + e\,\varepsilon)^*e \\
&= (e\,(f + \varepsilon))^*e \\
&= e\,((f + \varepsilon)\,e)^* \\
&= e\,(f\,e + \varepsilon\,e)^* \\
&= e\,(f\,e + e)^*
\end{aligned}
$$

- Geht sowas auch automatisch ?
- Gibt es einen Algorithmus zu entscheiden, ob $r_1 = r_2$ gilt ?
  - Input: reguläre Ausdrücke r_1, r_2
  - Ausgabe: true, falls $L(r_1) = L(r_2)$, false, sonst … ?

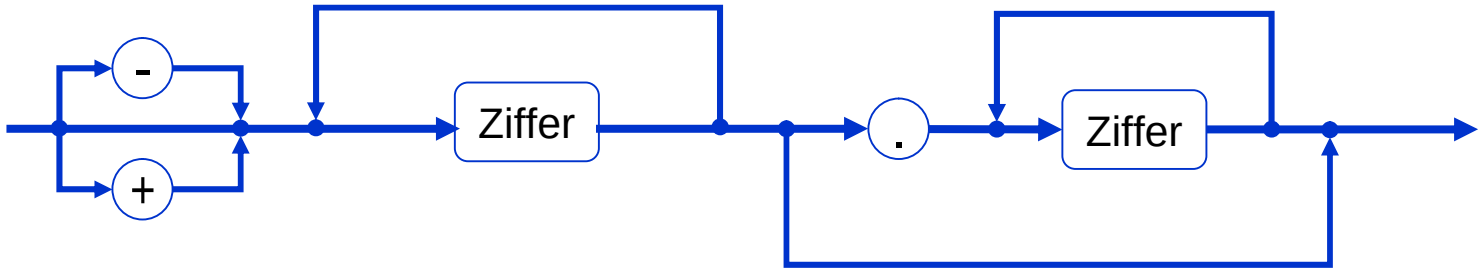
Syntaxdiagramme für reg. Ausdrücke



Keine Rekursion erlaubt !!!

Beispiel: Syntaxdiagramm

- Real Zahlen
  - als Syntaxdiagramm



  - als regulärer Ausdruck:

$$(+ \mid - \mid \varepsilon)\ \text{Ziffer}^{+}\ (.\ \text{Ziffer}^{+})?$$

wobei

$$\text{Ziffer} = (0 \mid 1 \mid \ldots \mid 9)$$

Erweitert reguläre Ausdrücke

? und + für optionale und zu wiederholende Teile

- e? := (e + ε)
- e^+ := ee*

[ ] für Teilmengen des Alphabets:

- [abc …] := a+b+c+ …
- [a-z] := a+b+…+z
- [a-zäüöß] := [a-z] + [äüöß]

:= für Abkürzungen (Keine Rekursion erlaubt !!!)

- digit := [0-9]
- digits := digit*
- nat := 0 + [1-9]digits
- sign := [+,-]?

Beispiel: Gleitkommazahlen in Java:

[\+,-]? (0+[1-9]digit*) (. [0-9]* (E[+,-]?digit digit)?)?

wobei

digit := [0-9]

Variationen und Zusätze

- Oft benutzt man | statt +
- [^...] Negation auf Zeichenmengen
  - [^aeiou] = [bcdfghj-np-tv-z]
  - [^x-z] = [a-w]
- Vorgeschriebene Wiederholungen {n,m}
  - e{2,4} = ee + eee + eeee
  - e{3,*} = eeee*
- Escaping: Zeichen, verlieren Sonder-Bedeutung
  - [\(,\)]* eine Folge von Klammern
  - [\n] neue Zeile
  - ** eine Folge von Sternen
  - \\ ein backslash

Programmiersprachen

- Teile von Programmiersprachen beschreibt man durch reguläre Ausdrücke
  - Ziffern
    - digit := [0-9] = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
  - Bezeichner
    - id := [a-zA-Z_$] [a-zA-Z_$0-9]*
  - HexZiffer
    - hex := <digit> | [A-F]
  - Unicode Zeichen
    - unic := \u<hex>
  - Gleitpunkt-Literale
    - dez := [\+|-]? [0-9]* (.[0-9]+)? (e[\+|-]?[0-9]+)? (d|f)?

Token für reguläre Ausdrücke

- Reguläre Ausdrücke spezifizieren Token

  - `[ _a-zA-Z][a-zA-Z0-9]*` `return BEZEICHNER;`
  - `[-+]?{dig}+\.?([eE][-+]?{dig}+)?` `return NUM;`
  - `if|else|while` `return keyWord;`

- Für jeden dieser Ausdrücke braucht man ein Programm, das diesen erkennt

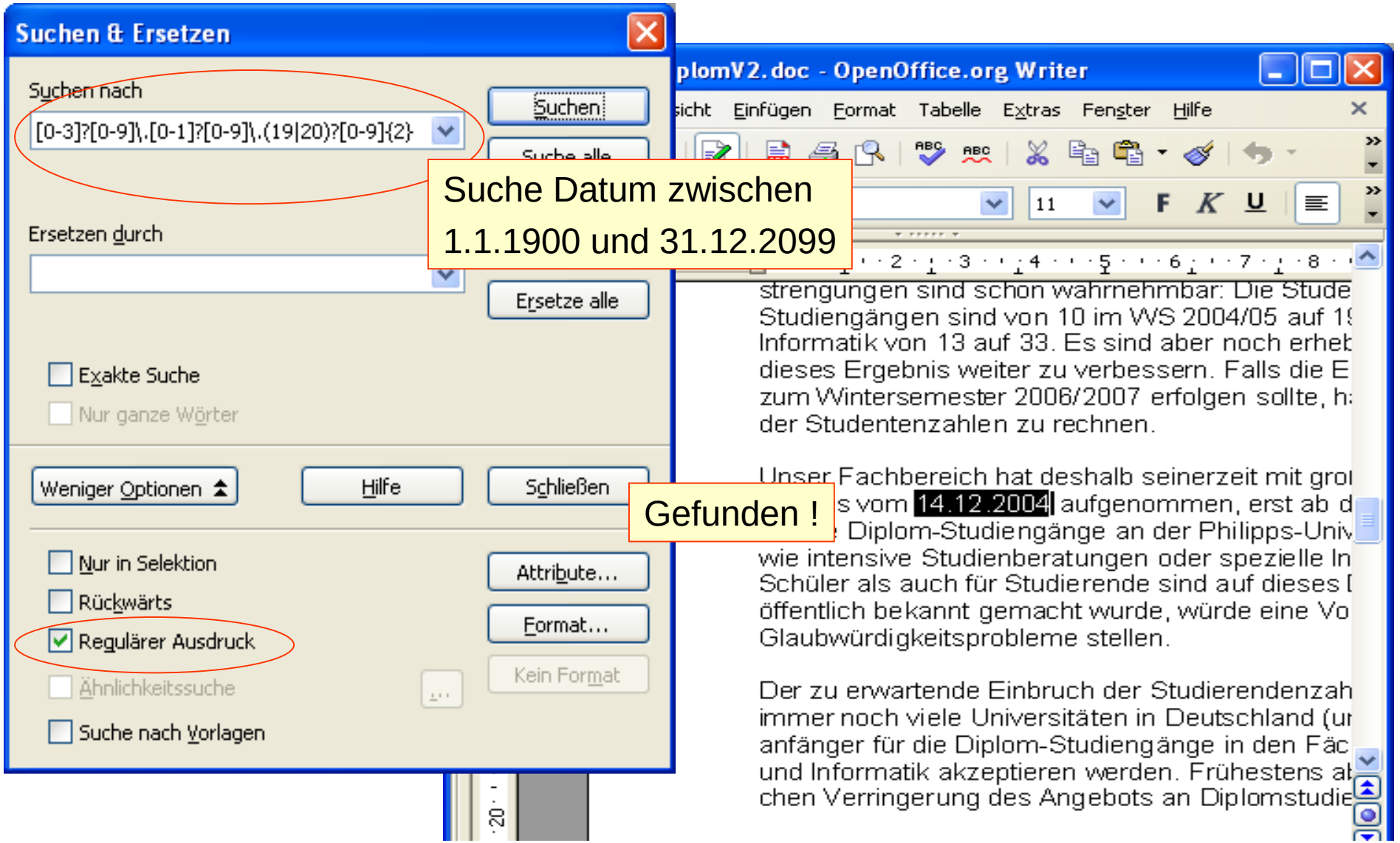
  - oder ein Programm, das mehrere Ausdrücke erkennt und jeweils das entsprechende Token zurückgibt

  -) Scanner

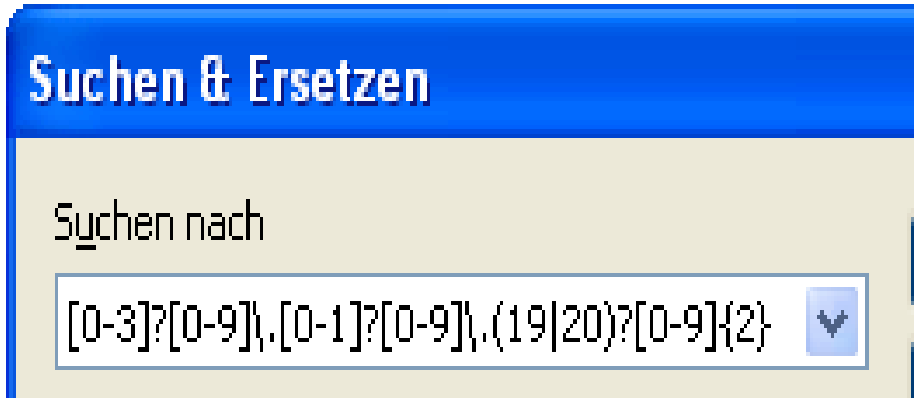
- Scanner besitzt Spezifikation vieler regulärer Sprachen

  - Wenn er ein Wort einer Sprache erkennt, gibt er das entsprechende Token zurück

Suche in OpenOffice-Dokument



Der verwendete reguläre Ausdruck

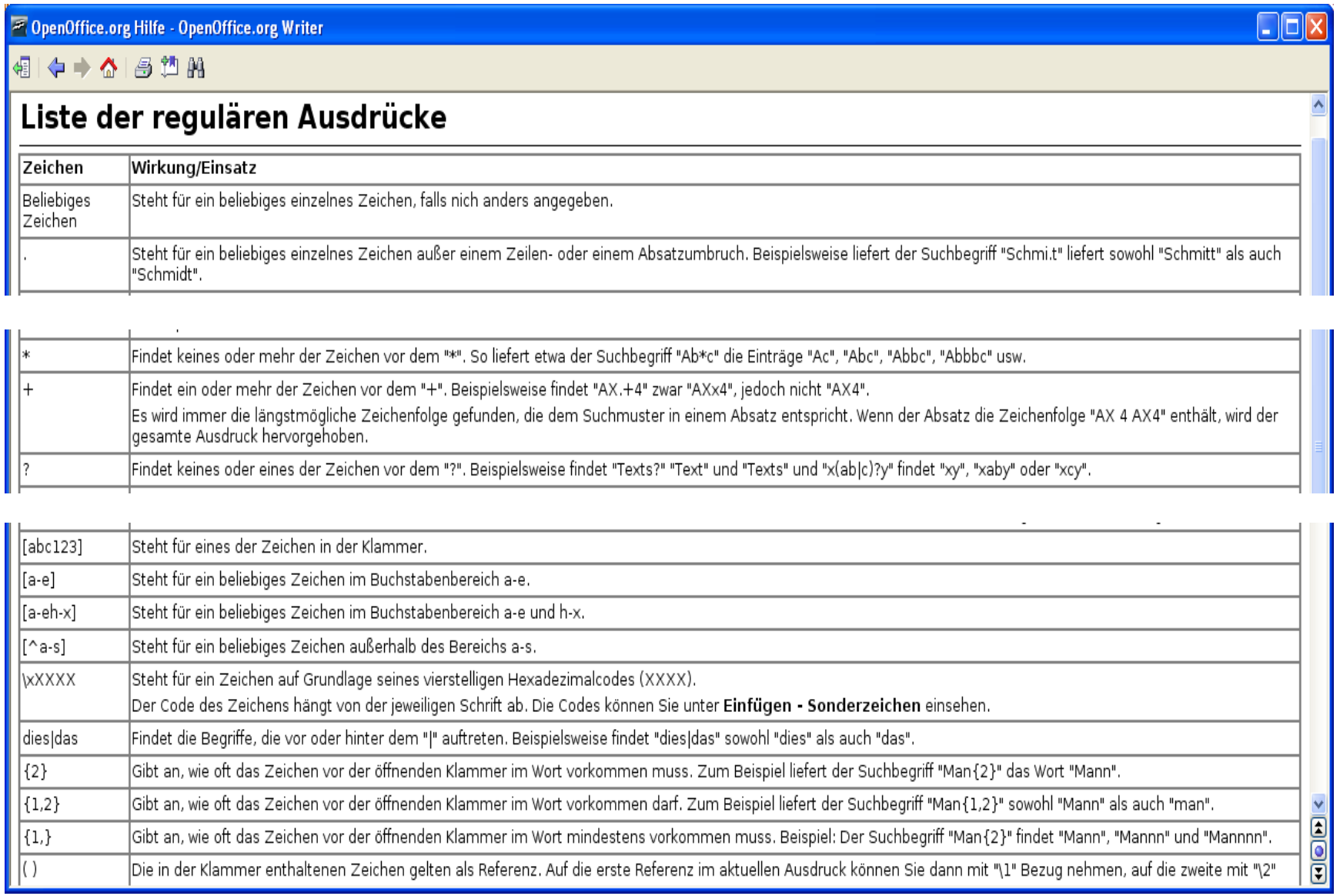


Beispiele !

9.6.06	C
21.07.2007	C
5.22.09	D
32.19.1999	C

- [0-3]?
  - eine Ziffer zwischen 0 und 3 – optional
- [0-9]
  - eine beliebige Ziffer
- \.
  - ein Punkt '.' Escapezeichen '\' notwendig
- (19|20)?
  - 19 oder 20 – optional
- [0-9]{2}
  - 2 beliebige Ziffern

Reguläre Ausdrücke in OpenOffice



Liste der regulären Ausdrücke

Zeichen	Wirkung/Einsatz
Beliebiges Zeichen	Steht für ein beliebiges einzelnes Zeichen, falls nich anders angegeben.
.	Steht für ein beliebiges einzelnes Zeichen außer einem Zeilen- oder einem Absatzumbruch. Beispielsweise liefert der Suchbegriff "Schmi.t" liefert sowohl "Schmitt" als auch "Schmidt".
*	Findet keines oder mehr der Zeichen vor dem "*". So liefert etwa der Suchbegriff "Ab*c" die Einträge "Ac", "Abc", "Abbc", "Abbbc" usw.
+	Findet ein oder mehr der Zeichen vor dem "+". Beispielsweise findet "AX.+4" zwar "AXx4", jedoch nicht "AX4". Es wird immer die längstmögliche Zeichenfolge gefunden, die dem Suchmuster in einem Absatz entspricht. Wenn der Absatz die Zeichenfolge "AX 4 AX4" enthält, wird der gesamte Ausdruck hervorgehoben.
?	Findet keines oder eines der Zeichen vor dem "?". Beispielsweise findet "Texts?" "Text" und "Texts" und "x(ab|c)?y" findet "xy", "xaby" oder "xcy".
[abc123]	Steht für eines der Zeichen in der Klammer.
[a-e]	Steht für ein beliebiges Zeichen im Buchstabenbereich a-e.
[a-eh-x]	Steht für ein beliebiges Zeichen im Buchstabenbereich a-e und h-x.
[^a-s]	Steht für ein beliebiges Zeichen außerhalb des Bereichs a-s.
\xXXXX	Steht für ein Zeichen auf Grundlage seines vierstelligen Hexadezimalcodes (XXXX). Der Code des Zeichens hängt von der jeweiligen Schrift ab. Die Codes können Sie unter **Einfügen - Sonderzeichen** einsehen.
dies|das	Findet die Begriffe, die vor oder hinter dem "|" auftreten. Beispielsweise findet "dies|das" sowohl "dies" als auch "das".
{2}	Gibt an, wie oft das Zeichen vor der öffnenden Klammer im Wort vorkommen muss. Zum Beispiel liefert der Suchbegriff "Man{2}" das Wort "Mann".
{1,2}	Gibt an, wie oft das Zeichen vor der öffnenden Klammer im Wort vorkommen darf. Zum Beispiel liefert der Suchbegriff "Man{1,2}" sowohl "Mann" als auch "man".
{1,}	Gibt an, wie oft das Zeichen vor der öffnenden Klammer im Wort mindestens vorkommen muss. Beispiel: Der Suchbegriff "Man{2}" findet "Mann", "Mannn" und "Mannnn".
( )	Die in der Klammer enthaltenen Zeichen gelten als Referenz. Auf die erste Referenz im aktuellen Ausdruck können Sie dann mit "\1" Bezug nehmen, auf die zweite mit "\2"

Reguläre Ausdrücke in Unix

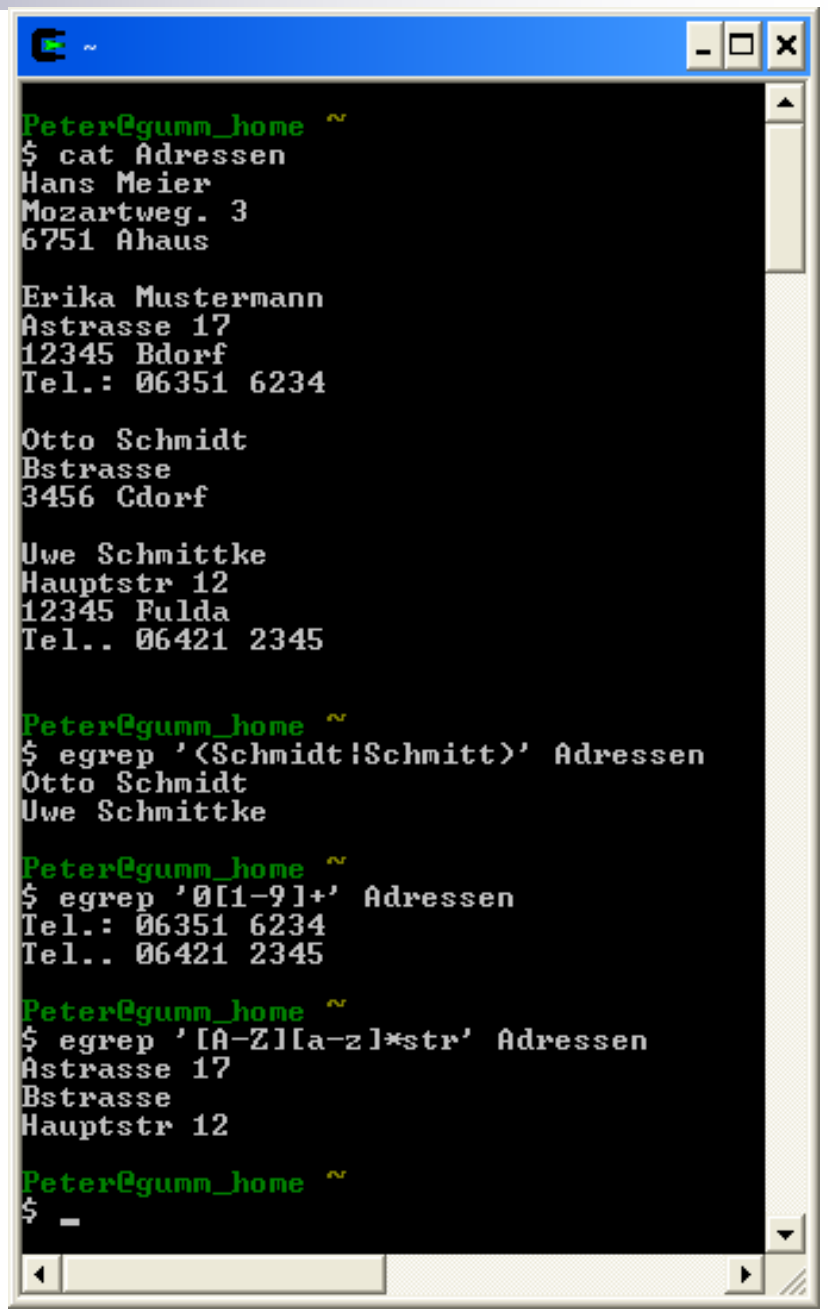
Viele Unix-Kommandos

akzeptieren reguläre Ausdrücke

- Suchkommandos
  - egrep (=grep –E)
- Editoren
  - sed
  - vi
  - emacs

Nicht verwechseln mit *File-Pattern*

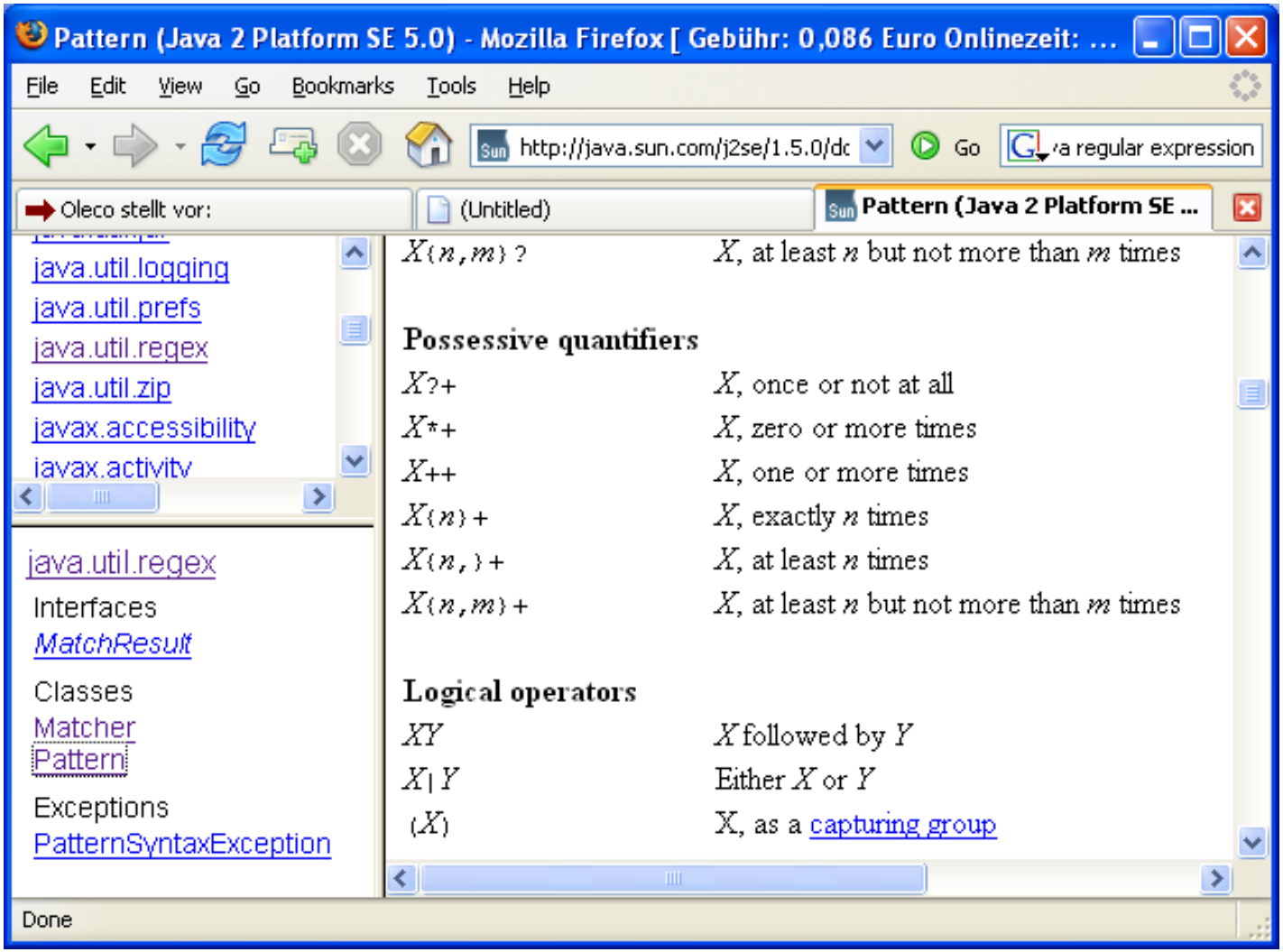
- werden von shell ausgewertet



Java.util.regexp

Java hat ein Paket **java.util.regexp**

Also müssen reguläre Ausdrücke etwas wichtiges sein



Wie funktioniert Suche mit RE ?

- Wie mächtig sind reguläre Ausdrücke
  - Was kann man ausdrücken, was nicht ?
    - Nur „gültige“ Kalenderdaten ?
      - 31.1.2006, 29.2.1996 aber nicht 30.2.04, 31.11.99
- Wie finde ich einen Teilstring, der auf einen regulären Ausdruck passt ?
  - Algorithmus gesucht
  - Wie komplex ist so ein Algorithmus?
  - Effiziente Implementierung