



Theoretische Informatik

Reguläre Sprachen
und Automaten



n Reguläre Sprachen

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



Reguläre Ausdrücke über Σ

Syntax:

- „ Konstante Ausdrücke
 - „ \emptyset ,
 - „ 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 $\Sigma = \{0,1\}$: $0, \epsilon, 0(10+01)^*1, 0(0+1)^*1, (0+1)^*0(0+1)^*, \dots$

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

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

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

Vereinbarung: $*$ bindet am stärksten, $+$ am schwächsten



Semantik regulärer Ausdrücke

n Jeder reguläre Ausdruck e beschreibt eine Sprache $L(e)$:

- .. $L(\emptyset) := \{ \}$
- .. $L(\epsilon) := \{\epsilon\}$
- .. $L(a) := \{a\}$ für jedes $a \in \Sigma$

n und induktiv:

- .. $L(e + f) := L(e) \cup L(f)$
- .. $L(e \cdot f) := L(e) \cdot L(f)$
- .. $L(e^*) := L(e)^*$

n Beispiele:

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

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

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

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

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

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

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

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

$$= L(e)$$

$$L(e \emptyset) = L(e) \quad L(\emptyset) = L(e)\{ \} = \{ \}$$

aber

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

Vorsicht: $-$ ist kein regulärer Operator !



Gleichungen

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



Stephen Kleene

n Es gelten u.a.

$$\begin{aligned}\emptyset + e &= e \\ e + f &= f + e \\ (e + f) + g &= e + (f + g) \\ \\ e \cdot e &= e = e \cdot e \\ (e \cdot f) \cdot g &= e \cdot (f \cdot g)\end{aligned}$$

$$\begin{aligned}e(f + g) &= ef + eg \\ (e + f)g &= eg + fg \\ \\ \varepsilon^* &= \varepsilon \\ (e^*)^* &= e^* \\ (\varepsilon + e)^* &= e^* \\ (e^* f^*)^* &= (e + f)^* \\ (ef)^* e &= e (fe)^*\end{aligned}$$

n aber auch:

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



Herleitung

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

- n Geht sowas auch automatisch ?
- n 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 ... ?

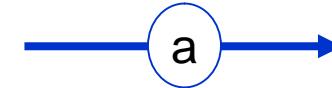


Syntaxdiagramme für reg. Ausdrücke

$e = \epsilon$



$e = a$



$e = r_1 r_2$

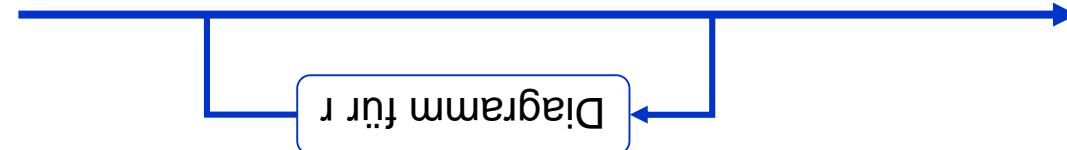


$e = r_1 + r_2$



Keine Rekursion
erlaubt !!!

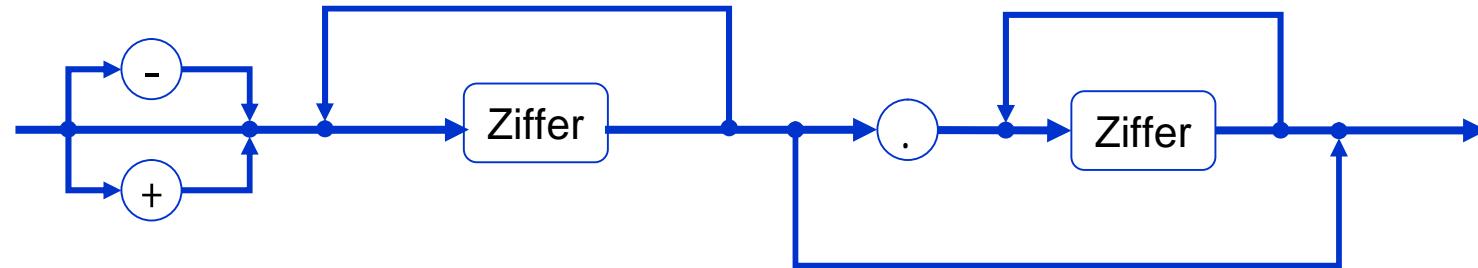
$e = r^*$





Beispiel: Syntaxdiagramm

- n Real Zahlen
 - als Syntaxdiagramm



- als regulärer Ausdruck:
$$(+ \mid - \mid \varepsilon) \text{ Ziffer}^+ (\cdot \text{ Ziffer}^+) ?$$

wobei

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



Erweitert reguläre Ausdrücke

? und + für optionale und zu wiederholende Teile

- .. $e^?$:= $(e + \epsilon)$
- .. e^+ := ee^*

[] für Teilmengen des Alphabets:

- .. $[abc \dots]$:= $a+b+c+\dots$
- .. $[a-z]$:= $a+b+\dots+z$
- .. $[a-zAÖÜß]$:= $[a-z] + [\ddot{a}\ddot{u}\ddot{o}\ddot{\ddot{u}}\ddot{\ddot{o}}\ddot{\ddot{\ddot{u}}}$

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

- .. digit := $[0-9]$
- .. digits := digit^*
- .. nat := $0 + [1-9]\text{digits}$
- .. sign := $[+,-]?$

Beispiel: Gleitkommazahlen in Java:

$[\pm, -]? (0+[1-9]\text{digit}^*) (\cdot [0-9]^* (E[\pm, -]?\text{digit digit})?)?$

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

- n Reguläre Ausdrücke spezifizieren **Token**
 - .. `[_a-zA-Z][a-zA-Z0-9]*` **return BEZEICHNER;**
 - .. `[+-]?\{dig\}+\.\?([eE][+-]?\{dig\}+)?` **return NUM;**
 - .. `if|else|while` **return keyword;**
- n 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**
- n Scanner besitzt Spezifikation **vieler** regulärer Sprachen
 - .. Wenn er ein Wort einer Sprache erkennt, gibt er das entsprechende Token zurück



Suche in OpenOffice-Dokument

The screenshot shows the "Suchen & Ersetzen" (Search & Replace) dialog box in OpenOffice.org Writer. A red oval highlights the search term in the "Suchen nach" field: `[0-3]?[0-9],[0-1]?[0-9],(19|20)?[0-9]{2}`. A yellow box contains the text "Suche Datum zwischen 1.1.1900 und 31.12.2099". Another red oval highlights the "Regulärer Ausdruck" (Regular Expression) checkbox, which is checked. The "Gefunden!" (Found!) message is displayed in a yellow box. The main document window shows text about student numbers and study programs.

Suchen & Ersetzen

Suchen nach: `[0-3]?[0-9],[0-1]?[0-9],(19|20)?[0-9]{2}`

Ersetzen durch:

Exakte Suche
 Nur ganze Wörter

Weniger Optionen

Nur in Selektion
 Rückwärts
 Regulärer Ausdruck
 Ähnlichkeitssuche
 Suche nach Vorlagen

Suche Datum zwischen
1.1.1900 und 31.12.2099

Gefunden!

plomV2.doc - OpenOffice.org Writer

Suche Datum zwischen 1.1.1900 und 31.12.2099

strengungen sind schon wahrnehmbar: Die Studiengänge sind von 10 im WS 2004/05 auf 18 im WS 2005/06 erhöht worden. Informatik von 13 auf 33. Es sind aber noch erhebliche Anstrengungen erforderlich, um dieses Ergebnis weiter zu verbessern. Falls die Erweiterung der Studiengänge zum Wintersemester 2006/2007 erfolgen sollte, hätte dies eine erhebliche Auswirkung auf die Studentenzahlen zu rechnen.

Unser Fachbereich hat deshalb seinerzeit mit großem Aufwand die Studiengänge von 14.12.2004 aufgenommen, erst ab dem Wintersemester 2005/2006. Diplom-Studiengänge an der Philipps-Universität Marburg werden wie intensive Studienberatungen oder spezielle Informationssysteme für Schüler als auch für Studierende sind auf dieses Ergebnis hinzuweisen. Obwohl dies öffentlich bekannt gemacht wurde, würde eine Verteilung der Studiengänge auf diese Weise Glaubwürdigkeitsprobleme stellen.

Der zu erwartende Einbruch der Studierendenzahl ist jedoch nicht so stark wie bei anderen Universitäten in Deutschland (urheberrechtlich geschützt). Anfänger für die Diplom-Studiengänge in den Fächern Informatik und Informatik akzeptieren werden. Frühestens ab dem Wintersemester 2006/2007 kann eine Verringerung des Angebots an Diplomstudienprogrammen erwartet werden.



Der verwendete reguläre Ausdruck

Suchen & Ersetzen

Suchen nach

[0-3]?[0-9]],[0-1]?[0-9]],[19|20]?[0-9]{2}

Beispiele !

9.6.06	C
21.07.2007	C
5.22.09	D
32.19.1999	C

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



Reguläre Ausdrücke in OpenOffice

OpenOffice.org Hilfe - OpenOffice.org Writer

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

```
Peter@gumm_home ~
$ cat Adressen
Hans Meier
Mozartweg. 3
6751 Ahaus

Erika Mustermann
Astrasse 17
12345 Bdorf
Tel.: 06351 6234

Otto Schmidt
Bstrasse
3456 Cdorf

Uwe Schmittke
Hauptstr 12
12345 Fulda
Tel.. 06421 2345

Peter@gumm_home ~
$ egrep '(Schmidt|Schmitt)' Adressen
Otto Schmidt
Uwe Schmittke

Peter@gumm_home ~
$ egrep '0[1-9]+.' Adressen
Tel.: 06351 6234
Tel.. 06421 2345

Peter@gumm_home ~
$ egrep '[A-Z][a-z]*str' Adressen
Astrasse 17
Bstrasse
Hauptstr 12

Peter@gumm_home ~
$ -
```



Java.util.regex

Java hat ein Paket
java.util.regex

Also müssen
reguläre Ausdrücke
etwas wichtiges sein

The screenshot shows a Mozilla Firefox browser window displaying the Java 2 Platform SE 5.0 documentation for the `java.util.regex` package. The URL in the address bar is `http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex.html`. The page content is as follows:

Oleco stellt vor: (Untitled) Sun Pattern (Java 2 Platform SE ...)

Possessive quantifiers

<code>X{n,m}?</code>	X , at least n but not more than m times
<code>X?+</code>	X , once or not at all
<code>X*+</code>	X , zero or more times
<code>X++</code>	X , one or more times
<code>X{n}+</code>	X , exactly n times
<code>X{,n}+</code>	X , at least n times
<code>X{n,m}+</code>	X , at least n but not more than m times

Logical operators

<code>XY</code>	X followed by Y
<code>X Y</code>	Either X or Y
<code>(X)</code>	X , as a capturing group



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

