Lektion 7

In dieser Lektion werden reguläre Ausdrücke eingeführt, eine Methode, mit der man Sprachen kurz und elegant definieren kann. Sie lernen zunächst, wie reguläre Ausdrücke syntaktisch aufgebaut werden können und dann, wie die Semantik regulärer Ausdrücke definiert ist, d.h. welche Sprache durch einen Ausdruck beschrieben wird. Aus der Semantik ergibt sich auch, wie reguläre Ausdrücke äquivalent umgeformt und vereinfacht werden können, so dass deren Bedeutung gleich bleibt.

Kap. 5 Reguläre Ausdrücke

Inhalt

- Syntax und Semantik regulärer Ausdrücke
- Definition von Sprachen mittels regulärer Ausdrücke
- Äquivalenz und Vereinfachung regulärer Ausdrücke
- Ausblick: Anwendungen regulärer Ausdrücke

5.1 Motivation

Prüfung von Textformaten

Nehmen Sie an, Sie hätten die Aufgabe eine Methode zu implementieren, die prüft, ob ein String eine zulässig aufgebaute Datumsangabe ist.

- Folgende Formen für Datumsangaben seien zugelassen (hier jeweils an einem Beispiel gezeigt):

  9. Januar 2006
  09. Januar 06
  09.01.2006
  9.1.2006
  09.01.06
  9.1.06
  2006/01/09

- Da die Methode sehr oft verwendet wird, soll sich auch möglichst effizient implementiert sein.

Wie schwierig und zeitaufwendig wäre es für Sie, so eine Methode zu implementieren - eine halbe Stunde, eine Stunde, zwei Stunden, vier Stunden , …?

Vermutlich wäre Ihre Antwort, dass es im Prinzip nicht wirklich schwierig, aber doch etwas zeitaufwendig wäre. Im ersten Semester haben Sie in Programmieren den elementaren Umgang mit Zeichenketten geübt (ich nehme an, reguläre Ausdrücke haben Sie dort noch nicht kennengelernt). Da es relativ viele unterschiedliche Formate sind, würde das Programm mit den elementaren Zeichenkettenbefehlen aber wohl recht umfangreich werden (100, 200 Zeilen oder mehr?), und da dann einige Zeilen Programmcode zusammenkommen mit vielen Prüfungen und Fallunterscheidungen, würde es vermutlich einige Zeit dauern, bis alle Fehler in der Implementierung beseitigt sind und alles korrekt funktioniert.

Gibt es elegantere Möglichkeiten, so eine Aufgabe zu lösen? Im Prinzip ja, indem man auf einer höheren, abstrakteren Beschreibungsebene arbeitet und dann passende Bibliotheken oder Werkzeuge einsetzt. Sog. *reguläre Ausdrücke* sind eine formale Beschreibungsmethode, um Sprachen elegant und übersichtlich zu definieren. Mit geeigneten Methoden und Werkzeugen/Bibliotheken kann dann zu einem regulären Ausdruck sehr einfach geprüft werden, ob ein String zur Sprache des Ausdrucks gehört. D.h. der regulärer Ausdruck kann als eine Art Textmuster betrachtet werden und die Sprache des regulären Ausdrucks sind dann alle Wörter, die zu dem Textmuster passen (in Java gibt's z.B. eine Methode `matches` dafür).

Datumsformat-Beschreibung mit regulären Ausdrücken

Die Beschreibung möglicher Datumsangabe auf der Basis regulärer Ausdrücke könnte beispielsweise so aussehen (Die Notation hier ist an ein Compilerbau-Tool angelehnt):

Tag2 = ["0"-"3"] ["0"-"9"]
Tag12 = (["1"-"3"])? ["0"-"9"]

Monat = "Januar" | "Februar" | ... | "November" | "Dezember"
Monat1 = "0"["1"-"9"] | "1"["0"-"2"]
Monat12 = ["1"-"9"] | "1"["0"-"2"]

Jahr4 = ("1"|"2")["0"-"9"]["0"-"9"]["0"-"9"]
Jahr2 = ["0"-"9"]["0"-"9"]

Datum = *Tag12* ". " *Monat* " " *Jahr4*
| *Tag2* "." *Monat* " " *Jahr2*
| *Tag2* "." *Monat2* "." (*Jahr4* | *Jahr2*)
| *Tag12* "." *Monat12* "." (*Jahr4* | *Jahr2*)
| *Jahr4* "/" *Monat2* "/" *Tag2*

Auch wenn Sie sich im Moment noch nicht mit regulären Ausdrücken auskennen, können Sie vielleicht schon grob erahnen, was die Beschreibung bedeuten soll.

Mit dieser 12-zeiligen Beschreibung und ein paar Zeilen Rahmenprogramm dazu

zum Aufruf einer passenden Prüfmethode bzw. des daraus automatisch generierten Programmcodes ist dann die Aufgabe erledigt. Da die Beschreibung viel kürzer und übersichtlicher ist als die von Hand programmierte Lösung, ist die Wahrscheinlichkeit viel geringer, dabei Fehler zu machen. Und die Implementierung ist vermutlich auch deutlich effizienten, als wenn Sie im selbst geschriebenen Code viele Einzelfälle von Hand durchprobieren.

Die Beschreibung von Textmustern oder Textformaten durch reguläre Ausdrücke wird in der Informatik an vielen Stellen verwendet (weitere Beispiele dafür werden am Ende des Kapitels vorgestellt).

Wir betrachte hier zunächst die elementaren Grundlagen, die aus theoretischer Sicht die Basis bilden. In der Praxis einsetzbare Werkzeuge bieten dann noch deutlich flexiblere Beschreibungsmöglichkeiten. Wir gehen dabei in zwei Schritten vor:

(1) Wir definieren zunächst die **Syntax** regulärer Ausdrücke, d.h. wie können sie formuliert werden.

(2) Aufbauend auf der Syntax wird dann die **Semantik** der Ausdrücke definiert. Jeder Ausdruck beschreibt eine Sprache, d.h. eine Menge von "passenden" Wörtern.

5.2 Syntax regulärer Ausdrücke

Definition 5.1 - Syntax regulärer Ausdrücke

Die Menge der **regulären Ausdrücke** über einem Alphabet Σ ist rekursiv so definiert:

Folgendes sind *elementare* reguläre Ausdrücke:

(1)	$\varnothing$	
(2)	ε	
(3)	**a**	für jedes Zeichen $a \in \Sigma$

Sind *R* und *S* beliebige reguläre Ausdrücke, dann sind auch folgendes (*zusammengesetzte*) reguläre Ausdrücke:

(4)	$R \cdot S$
(5)	$R \mid S$
(6)	R^*
(7)	**(** *R* **)**

Es gibt also drei Formen von elementaren regulären Ausdrücken ($\varnothing$, ε, a) und mit Hilfe von Operatoren ($\cdot$ und | sind zweistellige Operatoren, * ist ein einstelliger Postfixoperator) und Klammern können zusammengesetzt Ausdrücke gebildet werden.

Beispiel 5.2 - Syntax regulärer Ausdrücken

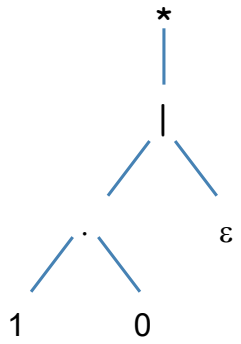
Ist beispielsweise $\Sigma = \{0, 1\}$ als Alphabet gegeben, dann ist

$(1 \cdot 0 \mid \varepsilon)^*$

ein syntaktisch korrekter regulärer Ausdruck, denn er ist nach obigen Regeln aufgebaut:

- 1 und 0 sind elementare reguläre Ausdrücke, siehe (3).
- Daraus kann nach (4) der Ausdruck $1 \cdot 0$ gebildet werden.
- Da ε ein regulärer Ausdruck ist, siehe (2), kann nach (5) der Ausdruck $1 \cdot 0 \mid \varepsilon$ gebildet werden
- (7) erlaubt es, Ausdrücke zu klammern, also erhält man $(1 \cdot 0 \mid \varepsilon)$
- nach (6) ist dann $(1 \cdot 0 \mid \varepsilon)^*$ ein regulärer Ausdruck.

Der syntaktische Aufbau des Ausdrucks kann auch durch folgenden abstrakten Syntaxbaum dargestellt werden:



Für jeden korrekt aufgebauten regulären Ausdruck lässt sich so ein abstrakter Syntaxbaum angeben (und umgekehrt).

Anmerkung

- in der Literatur wird teilweise auch *R*+*S* statt *R*|*S* verwendet.

Damit eindeutig klar ist, welcher strukturelle Aufbau mit einem regulären Ausdruck gemeint ist, gibt es Vorrangregeln für die Operatoren, ähnlich wie in der Arithmetik:

*Notationskonvention 5.3 - Vorrang der Operatoren *, ·, |*

- ❒ Operator * bindet stärker als Operator ·
- ❒ Operator · bindet stärker als Operator |

- Es gilt also auch hier "Punkt- vor Strichrechnung" und der einstellige Operator * hat höchste Priorität.

- Ähnlich wie in der Arithmetik wird der Operator $\cdot$ meist weggelassen. Es wird üblicherweise

  RS

  statt

  $R \cdot S$

  notiert.

Beispiel/Aufgabe 5.4 - Syntax regulärer Ausdrücke

Folgendes sind reguläre Ausdrücke über Alphabet $\Sigma = \{0,1\}$:

(1) 01

(2) 01*

(3) (01)*

(4) 01| ε

(5) (01)*(01| ε)

Stellen Sie jeweils den syntaktischen Aufbau der Ausdrücke als abstrakten Syntaxbaum dar.

5.3 Semantik regulärer Ausdrücke

Syntax-orientierte Semantikdefinition

Der syntaktische Aufbau der regulären Ausdrücke ist wichtig, denn die Definition dafür, welche Sprache durch einen regulären Ausdruck angegeben wird, d.h. die Semantik der Ausdrücke, basiert auf der syntaktischen Struktur:

- Wir definieren erst, was die Bedeutung (= Sprache) elementarer regulärer Ausdrücke ist.
- Für die zusammengesetzten Ausdrücke definieren wir, wie sich die Sprache des ganzen Ausdrucks aus der Sprache der Teilausdrücke ergibt.

Betrachtet man den Ausdruck als Syntaxbaum, dann ergibt sich damit eine "Bottom-Up"-Vorgehensweise: Man fängt bei den Blättern an und kann dann nach und nach die Bedeutung der darüber liegenden Teile bestimmten, bis man am Ende bei der Wurzel ist und die Bedeutung des ganzen Ausdrucks hat. Diese typische Vorgehensweise nennt man *syntax-orientierte Semantikdefinition*.

Ein regulärer Ausdruck R beschreibt eine Menge von "passenden" Wörtern $L(R)$. Ein Ausdruck kann also als ein Muster für mögliche Zeichenfolgen betrachtet werden. Die Bedeutung wird folgendermaßen rekursiv definiert (da auch der syntaktische Aufbau rekursiv definiert wurde).

Definition 5.5 - Semantik regulärer Ausdrücke

Die von einem **regulären Ausdruck *R* dargestellte Sprache *L*(*R*)** ist folgendermaßen definiert:

(1)	$L(\emptyset) = \{\}$	// leere Sprache, enthält gar kein Wort
(2)	$L(\varepsilon) = \{\varepsilon\}$	// Sprache, die nur das leere Wort enthält
(3)	für alle Zeichen $a \in \Sigma$:	
	$L(a) = \{a\}$	// Sprache, die nur das Wort aus dem einzelnen Zeichen a enthält
(4)	Ist $L(R) = L_1$	
	und $L(S) = L_2$	
	dann ist $L(R \cdot S) = L_1 \cdot L_2$	// Konkatenation der Sprachen
(5)	Ist $L(R) = L_1$	
	und $L(S) = L_2$	
	dann ist $L(R \mid S) = L_1 \cup L_2$	// Mengenvereinigung der Sprachen
(6)	Ist $L(R) = L_1$	
	dann ist $L(R^*) = L_1^*$	// Kleene-Abschluss der Sprache
(7)	Ist $L(R) = L_1$	
	dann ist $L((R)) = L_1$	// Klammern haben keine extra Bedeutung, sind nur zur syntaktischen Strukturierung

Der regulär Ausdruck ist hier jeweils blau angegeben, die Sprache des Ausdrucks rot.

Anmerkungen

- Bei den Operatoren · und * ist im ersten Moment vielleicht verwirrend, dass sie in den Definitionen links (blau) und rechts (rot) vorkommen. Bei der blauen Version links handelt es sich jeweils um das Operatorsymbol, mit dem rein syntaktisch zusammengesetzte Ausdrücke gebildet werden können. Bei den roten Versionen rechts vom Gleichheitszeichen handelt es sich um die Operatoren auf Sprachen, die im vorigen Kapitel eingeführt wurden.
- Die Bedeutung zusammengesetzter Ausdrücke kann auch so erklärt werden:
  - ***R* | *S*** ist die Menge aller Wörter, die in der Sprache von *R* oder in der Sprache von *S* enthalten sind (oder in beiden). Der Strich-Operator ist also eine Alternative: kann wie *R* oder wie *S* sein. Die Sprechweise ist deshalb "*R* oder *S*".
  - ***RS*** ist die Menge aller Wörter, so dass der erste Teil des Worts in der Sprache von *R* liegt und und der Rest des Worts in der Sprache von *S*.
  - ***R**** ist die Menge aller Wörter $w = u_1 u_2 \ldots u_n$, die sich aus beliebig vielen Teilwörter u_1 bis u_n aus der Sprache von *R* zusammensetzen (für ein $n \geq 0$).

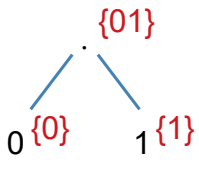
Jedes Wort aus der Sprache von *R* kann dabei beliebig oft verwendet werden. *n* = 0 ist möglich, d.h. das leeres Wort ε ist immer mit dabei.

Beispiel 5.6 - Semantik regulärer Ausdrücke

Welche Sprachen werden durch die regulären Ausdrücke jeweils dargestellt?

(1) 01, d.h. 0·1 :

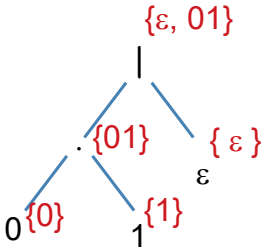
L(**0**) = {0}
L(**1**) = {1}
L(**0·1**) = {0} · {1} = { 01 }



Der Ausdruck beschreibt also die Sprache, die nur das Wort 01 enthält.

(2) 01| ε:

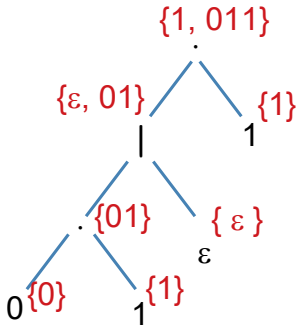
L(**01**) = { 01 } (s.o.)
L(**ε**) = { **ε** }
L(**01|ε**) = { 01 } ∪ { **ε** } = { **ε**, 01 }



Der Ausdruck 01| ε beschreibt die Sprache, die aus den beiden Wörtern 01 und ε besteht.

(3) (01| ε)1:

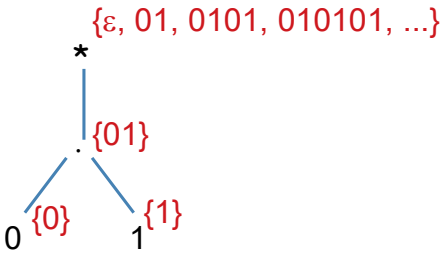
L(01| ε) = { **ε**, 01 } (s.o.)
L((01| ε)) = L(01| ε) = { **ε**, 01 }
L(1) = { 1 }
L((01| ε)1) = { ε, 01 } · { 1 } = {1, 011}



D.h. (01| ε)1 bedeutet: Wort fängt mit 01 oder nichts an, danach kommt am Ende auf jeden Fall eine 1.

(4) (01)*:

L((01)) = L(01) = { 01 }
L((01)*) = { 01 }*
= {ε, 01, 0101, 010101, 01010101, ...}

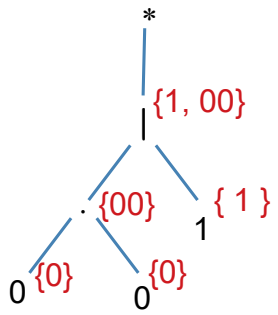


(01)* beschreibt also alle Wörter, die man bilden kann, indem man 01 beliebig oft zusammensetzt (auch nullmal)

(5) (00| 1)*:

L((00)) = { 00 }
L(1) = { 1 }
L((00|1)) = {00} ∪ {1}
= {1, 00}
L((00|1)*) = { 1, 00 }*
= {ε, 1 , 00, 100, 001, 11, 0000, …}

{ε, 1 , 00, 11, 100, 001, 0000, …}



(00| 1)* beschreibt also die Menge aller Wörter, die man durch beliebige Kombinationen der Wörter 00 und 1 bilden kann (einschließlich leeres Wort ε).

Aufgabe 5.7 - Semantik regulärer Ausdrücke

Welche Sprachen werden dargestellt?

(1) (ab | ba)

(2) (a | bb)(cc | aa)

(3) a* | b

(4) ab*

Folgende zusätzliche Operatoren für reguläre Ausdrücke werden oft verwendet, um mit regulären Ausdrücken bequemer arbeiten zu können.

Definition 5.8 - R^+ und R?

- **R^+** ist Abkürzung für **R R***
- **R?** ist Abkürzung für **R | ε**

- R^+ bedeutet also: einmal das, was durch R beschrieben wird und dann noch beliebig oft etwas, was durch R beschrieben wird, d.h. also "mindestens einmal, beliebig oft R".
- R? steht für R | ε, d.h. entweder R oder das leere Wort. Anders ausgedrückt: "R ist optional".

Beispiel 5.9 - regulärer Ausdruck

Gesucht ist ein regulärer Ausdruck für alle Wörter über Alphabet Σ = {a, b, c}, die *genau ein* b enthalten (Zeichen a und c können beliebig oft vorkommen).

- Lösung: Das heißt, vor dem b kann eine beliebig Folge von Zeichen a und c stehen, danach kann auch eine beliebig Folge von a und c stehen. Eine beliebige

Folge von a und c kann durch (a | c)* angegeben werden.

Somit ergibt sich:

(a | c)* b (a | c)*

Da bei * immer die leere Folge mit dabei ist, kann das b also auch ganz am Anfang oder ganz am Ende stehen

Aufgabe 5.10 - reguläre Ausdrücke

- Geben sie einen regulären Ausdruck für alle Wörter über Alphabet Σ = {a, b, c}, die *mindestens* ein b enthalten

Folgendes Beispiel zeigt nun auch eine sinnvollere Anwendung für reguläre Ausdrücke.

Beispiel 5.11 - Regulärer Ausdruck für natürliche Zahlen

Alle Dezimalzahlen (ohne Vorzeichen) sollen durch einen regulären Ausdruck beschrieben werden, z.B.

0
3
42
100

Dezimalzahlen Version 1:

- Dezimalzahlen sind (nicht leere) Folgen der Ziffern 0 bis 9, die sich so beschreiben lassen:

  D = (0 | 1 | … | 9)$^+$

  D ist hier als Abkürzung für den Ausdruck angegeben, um später darauf Bezug nehmen zu können. Hier muss ()$^+$ statt ()* verwendet werden, da es mindestens eine Ziffer geben muss.

Sind Sie mit dieser Definition für Dezimalzahlen einverstanden? Ist evtl. etwas zu bedenken? Eine Eigenschaft dieser Definition ist die, dass auch führende Nullen erlaubt sind. Beispielsweise ist 007 ein Wort dieser Sprache.

Wenn führend Nullen nicht erlaubt sein sollen, lässt sich der Aufbau der Zahlen dann auch durch einen regulären Ausdruck beschreiben? Ja, kein Problem:

Dezimalzahlen Version 2 (ohne führende Nullen):

- Ohne führende Nullen bedeutet: Die Zahl darf mit jeder Ziffer außer 0 anfangen, danach können noch beliebig viele Ziffern kommen, einschließlich der 0. Eine Ausnahme gibt es allerdings: Die 0 alleine als Zahl soll weiterhin erlaubt sein.

- Die Lösung sieht dann so aus:

  D' = (`1` | `2` | ... | `9`) (`0` | `1` | ... | `9`)* | `0`

  Hier wird * gebraucht, da auch eine einzelne Ziffer möglich sein muss, d.h. nach der ersten Ziffer darf auch nichts mehr kommen.

Aufgabe 5.12 - Regulärer Ausdruck für Zahlen

In Programmiersprachen sind üblicherweise unterschiedliche Formate für die Angabe von Zahlen möglich. Definieren Sie einen regulärer Ausdruck für eine Zahl (ohne Vorzeichen), die beliebig in einer der folgenden Darstellungen gegeben sein kann

- als Dezimalzahl oder
- als Hexadezimalzahl, beginnend mit `0x` (z.B. `0xaffe007`) oder
- als Gleitpunktzahl mit Dezimalpunkt (z.B. `12.345` oder `0.10000`), wobei es mindestens eine Stelle vor und nach dem Dezimalpunkt geben soll.

5.4 Äquivalenz regulärer Ausdrücke

Bei den Lösungsvorschlägen für die Aufgabe "mindestens ein b" in der vorigen Lektion haben Sie gesehen, dass die gleiche Sprache auf unterschiedliche Weise mit regulären Ausdrücken beschrieben werden kann. Wenn zwei Ausdrücke die gleiche Sprache beschreiben, dann nennen wir die Ausdrücke *äquivalent*.

Definition 5.13 - Äquivalenz regulärer Ausdrücke

Zwei reguläre Ausdrücke *R* und *S* heißen **äquivalent,** falls
L(*R*) = L(*S*),
d.h. wenn beide Ausdrücke die gleiche Sprache darstellen.

Wenn zwei Ausdrücke äquivalent sind, kann man den einen Ausdruck gegen den anderen tauschen, z.B. um einen komplizierten Ausdruck durch einen gleichbedeutenden einfacheren, verständlicheren Ausdruck zu ersetzen.

Bei der äquivalenten Umformung von regulären Ausdrücken können folgende Äquivalenzeigenschaften verwendet werden. Diese Eigenschaften sollten Sie sich einprägen.

Eigenschaft 5.14 - Äquivalenz regulärer Ausdrücke

Für beliebige reguläre Ausdrücke R, S und T gilt:

(1) $R \mid S = S \mid R$

Kommutativität der Alternative | , d.h. die Operanden können in der

Reihenfolge vertauscht werden.

(2) $(R \mid S) \mid T = R \mid (S \mid T)$

Assoziativität der Alternative |: Verknüpft man drei Teil mit der Alternative |, dann ist es egal, wie man es klammert. Die Klammern lässt man deshalb üblicherweise ganz weg und schreibt nur R | S | T.

(3) $RS \mid RT = R(S \mid T)$

Links-Distributivität von | bezüglich Konkatenation: Gemeinsame Anfangsteile von zwei Alternativen können nach links ausgeklammert werden

(4) $RT \mid ST = (R \mid S)T$

Rechts-Distributivität von | bezüglich Konkatenation: Gemeinsame Endteile von zwei Alternativen können nach rechts ausgeklammert werden

(5) $R \mid R = R$

Idempotenz von |: Zweimal das gleiche *R* als Alternative zu haben, ergibt nichts Neues.

(6) $R \mid \varnothing = \varnothing \mid R = R$

$\varnothing$ ist neutrales Element bzgl. |: Da $\varnothing$ die leere Sprache (ohne Wörter) angibt, bringt es nichts zusätzlich, wenn man es als Alternative hat.

(7) $(RS)T = R(ST)$

Assoziativität der Konkatenation: Hängt man drei Teile *R*, *S* und *T* aneinander, ist es egal, wie es geklammert wird. Man lässt deshalb üblicherweise die Klammern weg und schreibt nur *RST.*

(8) $R\varepsilon = \varepsilon R = R$

ε ist neutrales Element bzgl. Konkatenation: Hängt man das leere Wort vorne oder hinten dran, ändert sich nichts.

(9) $R\varnothing = \varnothing R = \varnothing$

$\varnothing$ ist sog. Nullelement bezüglich Konkatenation: Da $\varnothing$ die leere Sprache (ohne Wörter) angibt, kann man keine Wörter bilden, die mit einem Wort daraus anfangen oder enden. Also ergibt sich wieder die leere Sprache.

(10) $R^{**} = R^*$

R^* bedeutet schon, alle Möglichkeiten zu bilden, wie Wörter aus R zusammengesetzt werden können. Wenn man diese Möglichkeiten nochmals beliebig kombiniert, ergibt sich nichts Zusätzliches. .

(11) $R^* = \varepsilon \mid RR^*$ bzw. $R^* = \varepsilon \mid R^+$

R* bedeutet: R beliebig oft, auch mal. ε ist das gleiche wie 0 mal R. Somit kann $\varepsilon \mid RR^*$ als 0 mal R oder mindestens einmal R gelesen werden.

(12) $R^* = \varepsilon \mid R^*$

Da das leere Wort ε immer bei R^* mit dabei ist, ergibt sich nichts Zusätzliches, wenn man es noch explizit als Alternative dazunimmt.

Aufgabe 5.15 - Äquivalenz regulärer Ausdrücke

Gelten folgende Äquivalenzen für beliebige reguläre Ausdrücke *R, S* und *T*?

(1) **R·S = S·R**

(2) **(R | S)* = R* | S***

(3) **R (S | T) R = RSR | RTR**

(4) **R | (ST) = (R | S) (R | T)**

Beispiel 5.16 - Vereinfachung regulärer Ausdrücke

Der reguläre Ausdruck

(ab | b)((ε | a)b)* | ε

kann z.B. in folgender Weise vereinfacht werden, um verstehen zu können, welche Sprache damit dargestellt wird.

(ab | b)(<u>(ε | a)b</u>)* | ε
= (ab | b)(<u>b | ab</u>)* | ε (4)
= (ab | b)(ab | b)* | ε (1)
= (ab | b)* (11)

Der Ausdruck beschreibt also alle Wörter, die durch beliebige Kombination der Wörter ab und b gebildet werden können.

Der Ausdruck könnten auch noch zu ((a | ε) b)* umgeformt werden.

Aufgabe 5.17 - Vereinfachung regulärer Ausdrücke

Vereinfachen Sie folgende reguläre Ausdrücke.

(1) aε (b | c) | ab

(2) ((a(bc) | (ab) d)* | ε) *

Fazit zu Lektion 7

Diese Fragen sollten Sie nun beantworten können

- Wie sind reguläre Ausdrücke syntaktisch aufgebaut?
- Wie ist die Semantik eines regulären Ausdrucks definiert?
- Wie können Sprachen mittels regulärer Ausdrücke beschrieben werden?
- Welche äquivalenten Umformungen sind bei regulären Ausdrücken möglich? Wie können reguläre Ausdrücke vereinfacht werden?