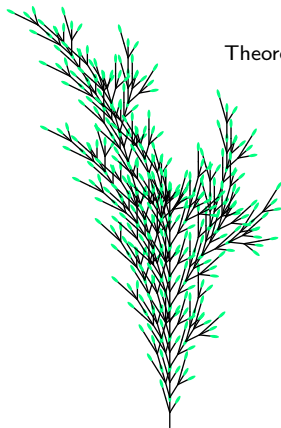


Formale Systeme, Automaten, Prozesse

Peter Rossmanith

Theoretische Informatik, RWTH Aachen

7. Juli 2020



Termine

Vorlesung über Zoom und Videoaufzeichnungen

- Mittwoch, 10:30 - 12:00 Uhr
- Freitag, 10:30 - 12:00 Uh

Fragestunden gegen Ende der Vorlesungen

Tutorübungen

- Montag, 8:30 - 10:00 Uhr
- Dienstag, 8:30 - 10:00 Uhr
- Dienstag, 10:30 - 12:00 Uhr

Globalübung

- Dienstag, 14:30 - 16:00 Uhr

Homepage: <https://tcs.rwth-aachen.de/lehre/FSAP/SS2020>

Anmeldungen über RWTH Online

Tutorübungen

Ablauf einer Doppelstunde

- Ausgabe der Übungsblätter am Montag 8 Uhr in Moodle
- Abgabe der Hausaufgaben bis Montag 8 Uhr in Moodle
- Gemeinsames Bearbeiten der Tutoraufgaben in Zoom
- Miniprüfung (15 Minuten), an Tutor schicken
- Rückgabe der korrigierten Hausaufgaben in Moodle

Anmeldungen über RWTH Online

Erste reguläre Tutorübung am **20. April**

Weitere Angebote

Peter Rossmanith

Es wird eine Sprechstunde über Videokonferenz geben

Henri Lotze, Daniel Mock, Tim Hartmann

Sprechzeiten: Donnerstag, 10:30 - 11:30 Uhr und nach
Vereinbarung

Globalübung

Dienstag 14:30 - 16:00

Zur Vorlesung wird es ein einfaches Skript geben, das sich eng an
die Vorlesung hält

Email: tcs-teaching@cs.rwth-aachen.de

Verwenden Sie nur diese Email-Adresse!

Prüfungen

1. Klausur

Montag, 17. August, 8:30 – 10:30 Uhr

2. Klausur

Mittwoch, 16. September, 10:30 – 12:30 Uhr

Teilnahmevoraussetzungen (BSc. Informatik)

- Regelmäßige Teilnahme an Tutorübungen und Hausaufgaben
- 50% der Punkte bei den Hausaufgaben
- 50% der Punkte bei den Miniprüfungen

Einleitendes Beispiel

Betrachte folgendes Problem:

Eingabe: Ein String w aus 0en und 1en

Frage: Sind diese beiden Eigenschaften erfüllt?

- Es kommt 11 nicht als Unterwort in w vor.
- Als Binärzahl ist w durch drei teilbar.

Beispiele: 0101, 1001, 00110, 0101010

Gesucht:

Ein Programm, das w bekommt und 0 oder 1 zurückgibt.

Eine mögliche Lösung:

```
int F[] = { 1, 0, 0, 0, 1, 0, 0};
```

```
int delta[][2] =
```

```
{{ 0, 1}, { 3, 6}, { 3, 4}, { 2, 5}, { 0, 6}, { 2, 6}, { 6, 6}};
```

```
int drei_not_11(char * w)
```

```
{
```

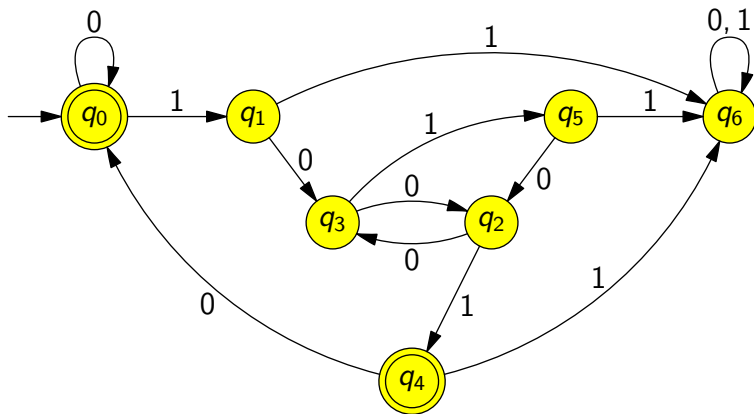
```
    int q = 0;
```

```
    while(*w) q = delta[q][*w++ - '0'];
```

```
    return F[q];
```

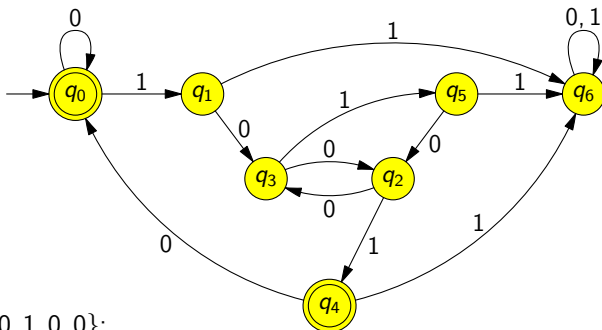
```
}
```

Das Programm simuliert. . .



einen sogenannten endlichen Automaten.

Vergleiche:



```
int F[] = { 1, 0, 0, 0, 1, 0, 0};
```

```
int delta[][2] = {{ 0, 1}, { 3, 6}, { 3, 4}, { 2, 5}, { 0, 6}, { 2, 6}, { 6, 6}};
```

```
int drei_not_11(char * w)
```

```
{
    int q = 0;
    while(*w) q = delta[q][*w++ - '0'];
    return F[q];
}
```

Wie effizient ist dieses Programm?

```
drei_not_11:
movsbl (%rdi), %eax
xorl %edx, %edx
testb %al, %al
je .L2
.L3:
subl $48, %eax
addq $1, %rdi
cltq
```

```
leaq (%rax,%rdx,2), %rax
movslq delta(,%rax,4), %rdx
movsbl (%rdi), %eax
testb %al, %al
jne .L3
.L2:
movl F(,%rdx,4), %eax
ret
```

Etwas besser zu verstehender MIPS-Code (RISC-Prozessor):

```

drei_not_11:
    lb $2,0($4)
    beq $2,$0,.L2
    move $3,$0
    lui $5,%hi(delta)
    addiu $5,$5,%lo(delta)
.L3:
    sll $3,$3,1
    addu $3,$3,$2
    addiu $4,$4,1
    addiu $3,$3,-48
    lb $2,0($4)
    sll $3,$3,2
    addu $3,$5,$3
    bne $2,$0,.L3

                                lw $3,0($3)
.L2:
                                lui $4,%hi(F)
                                addiu $4,$4,%lo(F)
                                sll $3,$3,2
                                addu $3,$3,$4
                                j $31
                                lw $2,0($3)
delta:
                                .word 0, 1, 3, 6, 3
                                .word 4, 2, 5, 0, 6
                                .word 2, 6, 6, 6
F:
                                .word 1, 0, 0, 0, 1
                                .word 0, 0

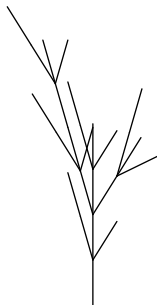
```

Zeichnen von Pflanzen

Zeichenprogramm, das diese Befehle kennt:

- F: Zeichne eine kurze Linie.
- -: Drehe dich ein wenig nach rechts.
- +: Drehe dich ein wenig nach links.
- [: Merke dir die augenblickliche Position und Richtung.
-]: Kehre zur letzten gemerkten Position und Richtung zurück.

```
F [+FF] [--F] F [+F [+FF] [--F] FF [+FF]
[--F] F] [--F [+FF] [--F] F] F [+FF] [--F] F
```



Einführung

Künstliche Pflanzen

```

F[+FF] [--F]F[+F[+FF] [--F]FF[+FF] [--F]F] [--F[+FF] [--F]F[+FF] [--F]F[+FF] [--F]F[+FF] [--F]F[+FF] [--F]F[+FF]
]F[+F[+FF] [--F]FF[+FF] [--F]F] [--F[+FF] [--F]F]F[+FF] [--F]FF[+FF] [--F]FF[+FF] [--F]FF[+FF] [--F]FF[+FF]
--F]FF[+FF] [--F]F] [--F[+FF] [--F]F]F[+FF] [--F]F] [--F[+FF] [--F]F[+F[+FF] [--F]FF[+FF] [--F]FF[+FF]
F] [--F]F] [--F[+FF] [--F]F]F[+FF] [--F]F]F[+FF] [--F]F[+F[+FF] [--F]FF[+FF] [--F]FF[+FF] [--F]FF[+FF] [--F]FF[+FF]
--F[+FF] [--F]F]F[+FF] [--F]F[+F[+FF] [--F]F[+F[+FF] [--F]FF[+FF] [--F]F] [--F[+FF] [--F]F[+FF] [--F]F[+FF]
F]F[+FF] [--F]F[+F[+FF] [--F]F[+F[+FF] [--F]FF[+FF] [--F]F] [--F[+FF] [--F]F[+FF] [--F]F[+FF] [--F]F[+FF]
[+FF] [--F]F[+F[+FF] [--F]FF[+FF] [--F]F] [--F[+FF] [--F]F]F[+FF] [--F]F]F[+FF] [--F]F]F[+FF] [--F]F]F[+FF]
+F[+FF] [--F]FF[+FF] [--F]F] [--F[+FF] [--F]F]F[+FF] [--F]FF[+FF] [--F]F[+F[+FF] [--F]FF[+FF] [--F]F[+F[+FF]
FF[+FF] [--F]F] [--F[+FF] [--F]F]F[+FF] [--F]F[+F[+FF] [--F]F[+F[+FF] [--F]F[+F[+FF] [--F]FF[+FF] [--F]FF[+FF]
F]F] [--F[+FF] [--F]F]F[+FF] [--F]FF[+FF] [--F]F[+F[+FF] [--F]FF[+FF] [--F]F] [--F[+FF] [--F]F]F[+FF]
] [--F]F]F[+FF] [--F]F] [--F[+FF] [--F]F[+F[+FF] [--F]FF[+FF] [--F]F] [--F[+FF] [--F]F]F[+FF]
F[+FF] [--F]F]F[+FF] [--F]F[+F[+FF] [--F]FF[+FF] [--F]F] [--F[+FF] [--F]F]F[+FF] [--F]F]F[+FF]
F] [--F[+FF] [--F]F[+F[+FF] [--F]FF[+FF] [--F]F] [--F[+FF] [--F]F]F[+FF] [--F]F[+F[+FF]
] [--F]F]F[+F[+FF] [--F]FF[+FF] [--F]F] [--F[+FF] [--F]F]F[+FF] [--F]FF[+FF] [--F]F]F[+F[+FF]
FF] [--F]FF[+FF] [--F]F] [--F[+FF] [--F]F]F[+FF] [--F]F]F[+FF] [--F[+FF] [--F]F]F[+F[+FF]
FF[+FF] [--F]F] [--F[+FF] [--F]F]F[+FF] [--F]F]F[+FF] [--F]F[+F[+FF] [--F]FF[+FF] [--F]FF[+FF]
]F] [--F[+FF] [--F]F]F[+FF] [--F]F]F[+FF] [--F]F[+F[+FF] [--F]FF[+FF] [--F]F] [--F[+FF]
] [--F]F]F[+FF] [--F]F[+F[+FF] [--F]F[+F[+FF] [--F]FF[+FF] [--F]F] [--F[+FF] [--F]F]F[+F[+FF]
+FF] [--F]FF[+FF] [--F]F[+F[+FF] [--F]FF[+FF] [--F]F] [--F[+FF] [--F]F]F[+FF] [--F]F]F[+FF]
--F[+FF] [--F]F[+F[+FF] [--F]FF[+FF] [--F]F] [--F[+FF] [--F]F]F[+FF] [--F]F]F[+FF] [--F]F[+F[+FF]
F]F[+F[+FF] [--F]FF[+FF] [--F]F] [--F[+FF] [--F]F]F[+FF] [--F]F]

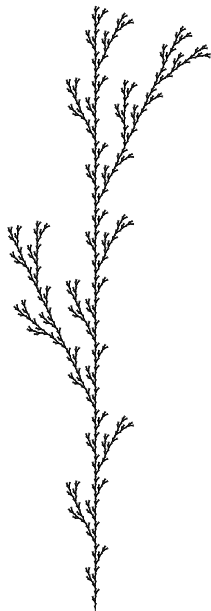
```

Starte mit F und wende $F \mapsto F[+FF][--F]F$ an!

Einführung

Künstliche Pflanzen

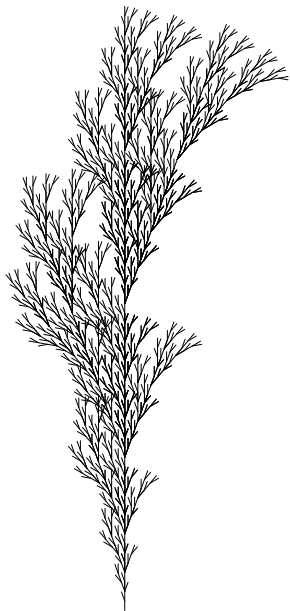




$$n = 5$$

$$\delta = 25.7$$

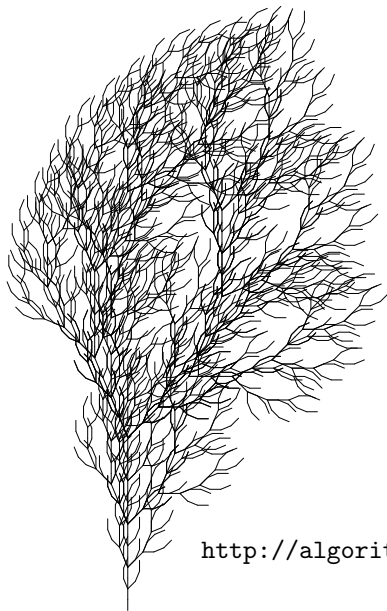
$$F \mapsto F[+F]F[-F]F$$



$$n = 5$$

$$\delta = 20$$

$$F \mapsto F[+F]F[-F][F]$$



$$n = 4$$

$$\delta = 22.5$$

$$F \mapsto$$

$$FF - [-F + F + F] + [+F - F - F]$$

Buch:

P. Prusinkiewicz

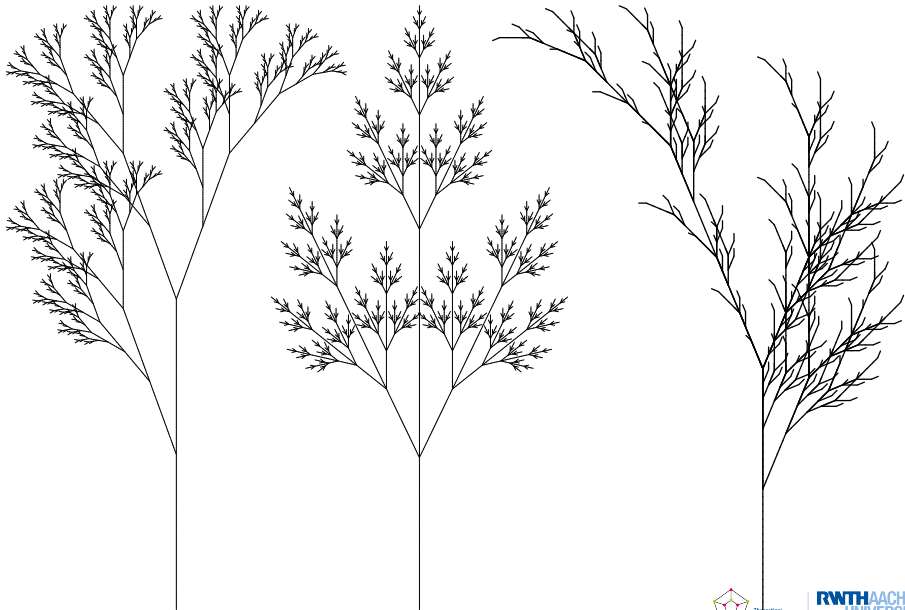
A. Lindenmayer

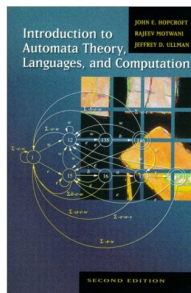
The Algorithmic Beauty of
Plants

<http://algorithmicbotany.org/papers/abop/abop.pdf>

Einführung

Künstliche Pflanzen



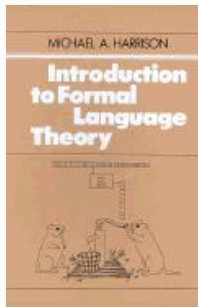


Introduction to Automata Theory, Languages, and Computation (2nd Edition)

by John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman

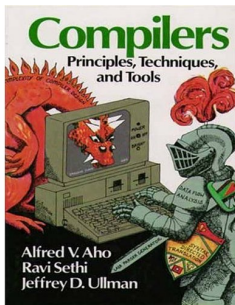
„Einzelplatzlizenz“ im RWTH-Netz:

<https://ebookcentral.proquest.com/lib/rwthaachen-ebooks/detail.action?docID=5832060>



Introduction to Formal Language Theory

by Michael A. Harrison



Compilers: Principles, Techniques, and Tools

by Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman

Wörter und Sprachen

Was ist ein Wort, was ist eine Sprache?

Informelle Antwort:

- 1 Ein Wort ist eine Aneinanderkettung von Symbolen aus einem Alphabet.
- 2 Eine Sprache ist eine Menge von Wörtern.

Beispiele:

01, 101001, ϵ sind Wörter über dem Alphabet $\{0, 1\}$

$\{0, 1, 101, 1001\}$ und $\{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots\}$ sind Sprachen über dem Alphabet $\{0, 1\}$.

Wie sieht eine formale, mathematisch korrekte Formalisierung dieser Begriffe aus?

Definition

- 1 Eine *Halbgruppe* (H, \circ) besteht aus einer Menge H und einer assoziativen Verknüpfung $\circ : H \times H \rightarrow H$.
- 2 Ein *Monoid* ist eine Halbgruppe mit einem neutralen Element.
- 3 Sei (M, \circ) ein Monoid und $E \subseteq M$.
 E ist ein *Erzeugendensystem* von (M, \circ) , falls jedes $m \in M$ als $m = e_1 \circ \dots \circ e_n$ mit $e_i \in E$ dargestellt werden kann.

Ein neutrales Element e ist links- und rechtsneutral. Für jedes x gilt $e \circ x = x \circ e = x$.

Frage: Ist das neutrale Element in einem Monoid eindeutig?

Ja, denn $e_1 \circ e_2 = e_1$ und $e_1 \circ e_2 = e_2$.

Beispiele

- $(\mathbf{Z}, +)$ ist ein Monoid.
 $\{-1, 1\}$ ein Erzeugendensystem.
- $(\mathbf{N}_0, +)$ ist ein Monoid.
 $\{1\}$ ein Erzeugendensystem.
- (\mathbf{Z}_8, \cdot) ist ein Monoid.
 $\{2, 3, 5\}$ ein Erzeugendensystem.

Frage:

Ist $\{-16, 17, 18\}$ ein Erzeugendensystem für $(\mathbf{Z}, +)$?

Ist $\{3, 5, 7\}$ ein Erzeugendensystem für (\mathbf{Z}_8, \cdot) ?

Freie Erzeugendensysteme

Definition

Ein Erzeugendensystem E für ein Monoid (M, \circ) ist *frei*, falls jedes $m \in M$ auf nur eine Art als $m = e_1 \circ \cdots \circ e_n$ mit $e_i \in E$ dargestellt werden kann.

Falls E ein freies Erzeugendensystem für (M, \circ) ist, dann sagen wir, daß (M, \circ) das von E *frei erzeugte Monoid* ist.

Frage:

Ist *das* korrekt?

Beispiele

$(\mathbf{Z}, +)$ ist von $\{-1, 1\}$ nicht frei erzeugt:

- $2 = 1 + 1 = 1 + 1 + (-1) + 1$
- $0 = (-1) + 1 = 1 + (-1)$

$(\mathbf{N}_0, +)$ ist von $\{1\}$ frei erzeugt.

Frage: Wie kann das neutrale Element erzeugt werden?

Frage: $(\mathbf{N}_0, +)$ von $\{1\}$ frei erzeugt. Wie wird 0 erzeugt?

Isomorphismen zwischen Monoiden

Definition

Zwei Monoide (M_1, \bullet) und (M_2, \circ) sind isomorph, falls es eine Abbildung $h: M_1 \rightarrow M_2$ gibt mit

- 1 h ist bijektiv.
- 2 h ist ein Homomorphismus, d.h. $h(u \bullet v) = h(u) \circ h(v)$ für alle $u, v \in M_1$.

Wir nennen h einen *Isomorphismus*.

Theorem

Es sei Σ ein Alphabet. Dann ist das von Σ frei erzeugte Monoid bis auf Isomorphismus eindeutig.

Beweis.

(M_1, \bullet) , (M_2, \circ) von Σ frei erzeugte Monoide.

$$h: M_1 \rightarrow M_2, u = u_1 \bullet \cdots \bullet u_n \mapsto u_1 \circ \cdots \circ u_n$$

$$g: M_2 \rightarrow M_1, v = v_1 \circ \cdots \circ v_m \mapsto v_1 \bullet \cdots \bullet v_m$$

mit $w_1, \dots, w_n \in \Sigma$.

$h(g(w)) = w$, also h bijektiv.

$$\begin{aligned} h(u \bullet v) &= h(u_1 \bullet \cdots \bullet u_n \bullet v_1 \bullet \cdots \bullet v_m) = \\ &u_1 \circ \cdots \circ u_n \circ v_1 \circ \cdots \circ v_m = h(u) \circ h(v), \end{aligned}$$

also ist h ein Homomorphismus. □

Definition

Es sei Σ ein Alphabet.

Dann ist (Σ^*, \cdot) das von Σ frei erzeugte Monoid.

Die Elemente von Σ^* nennen wir *Wörter* (über Σ).

Falls $L \subseteq \Sigma^*$, dann nennen wir L eine *Sprache* (über Σ).

Falls $u, v \in \Sigma^*$, dann schreiben wir auch uv statt $u \cdot v$.

Das neutrale Element von (Σ^*, \cdot) bezeichnen wir mit ϵ .

Theorem

Es seien Σ und Γ Alphabete. Jede Abbildung $\Sigma \rightarrow \Gamma^$ läßt sich eindeutig auf einen Homomorphismus $\Sigma^* \rightarrow \Gamma^*$ erweitern.*

Beweis.

Es sei $h: \Sigma^* \rightarrow \Gamma^*$ ein Homomorphismus. Dann ist $h(w) = h(w_1 \dots w_n)$ mit $w_1, \dots, w_n \in \Sigma = h(w_1) \dots h(w_n)$ weil h ein Homomorphismus ist. □

Wenn wir einen Homomorphismus definieren wollen, genügt es, seine Wirkung auf Symbole zu beschreiben.

Reguläre Ausdrücke

Definition

Es sei Σ ein Alphabet.

- 1 \emptyset ist ein regulärer Ausdruck.
- 2 ϵ ist ein regulärer Ausdruck.
- 3 a ist ein regulärer Ausdruck, falls $a \in \Sigma$.
- 4 rs ist ein regulärer Ausdruck, falls r und s reguläre Ausdrücke sind.
- 5 $r + s$ ist ein regulärer Ausdruck, falls r und s reguläre Ausdrücke sind.
- 6 r^* ist ein regulärer Ausdruck, falls r ein regulärer Ausdruck ist.

Definition

Es seien $A, B \subseteq \Sigma^*$ und $w \in \Sigma^*$.

- $AB := \{ uv \mid u \in A \text{ und } v \in B \}$

- $wA := \{w\}A$ und $Aw := A\{w\}$

- $A^i := \begin{cases} \{\epsilon\} & \text{falls } i = 0 \\ A & \text{falls } i = 1 \\ AA^{i-1} & \text{falls } i > 1 \end{cases}$

- $A^* := \bigcup_{n \geq 0} A^n$

- $A^+ := \bigcup_{n \geq 1} A^n$

Die Sprache eines regulären Ausdrucks

Definition

Wir ordnen jedem regulärer Ausdruck r seine *Sprache* $L(r)$ zu:

- 1 $L(\emptyset) = \emptyset$
- 2 $L(\epsilon) = \{\epsilon\}$
- 3 $L(a) = \{a\}$
- 4 $L(rs) = L(r)L(s)$
- 5 $L(r + s) = L(r) \cup L(s)$
- 6 $L(r^*) = L(r)^*$

Beispiele

- Der reguläre Ausdruck $0^*(10^*10^*)^*$ bezeichnet die Sprache aller Wörter über $\{0, 1\}$, die eine gerade Anzahl von 1en enthalten.
- 1^*0^* sind die Wörter über $\{0, 1\}$ die nicht 01 als Unterwort enthalten.
- $(0 + (11)^* + (10(1 + (00)^*)^*01))^*(0 + 11 + (10(1 + 00)^*01))^*$ sind die durch drei teilbaren Binärzahlen.