

**Dozent:** Prof. Dr. Michael Eichberg  
**Kontakt:** michael.eichberg@dhbw.de, Raum 149B  
**Version:** 1.1.1

**Quelle:** Die Folien sind teilweise inspiriert von oder basierend auf Lehrmaterial von Prof. Dr. Ritterbusch und Theoretische Informatik - kurzgefasst von Prof. Dr. Uwe Schöning.

---

**Folien:** [https://delors.github.io/theo-algo-formale\\_sprachen/folien.de.rst.html](https://delors.github.io/theo-algo-formale_sprachen/folien.de.rst.html)  
[https://delors.github.io/theo-algo-formale\\_sprachen/folien.de.rst.html.pdf](https://delors.github.io/theo-algo-formale_sprachen/folien.de.rst.html.pdf)

**Fehler melden:** <https://github.com/Delors/delors.github.io/issues>

# 1. Einführung

# Alphabete und Sprachen

Formale Sprachen sind ein zentraler Aspekt der theoretischen Informatik.

- Nutzungsinterface zwischen Computer und Mensch
- Grundlage für Programmiersprachen

Es gibt unterschiedliche Klassen und Modelle formaler Sprachen:

- Erkennbarkeit und Ausdruckskraft
- Anforderungen an Computermodele zur Erkennbarkeit
- Komplexität von Verfahren zur Erkennung

# Alphabete

## Definition

Ein Alphabet  $\Sigma = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$  ist eine endliche Menge von Zeichen / Symbolen.

## Beispiel

### Abzählbare Mengen

- $\Sigma_{lat} = \{a, b, c, \dots, z\}$
- $\Sigma_{ziffer} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $\Sigma_{unicode} = \{x | x \text{ ist ein Unicode-Zeichen}\}$
- $\Sigma_{logik} = \{0, 1, (, ), \wedge, \vee, \neg, (, )\} \cup \Sigma_{lat}$

# Kartesisches Produkt

## Definition

Ein kartesisches Produkt wie  $A \times B$  oder  $A^n$  für  $n \in \mathbb{N}$  von Mengen oder Alphabeten bezeichnet die Menge der Tupel  $(a, b)$  oder  $(a_1, \dots, a_n)$  von Elementen der Mengen:

$$\begin{aligned} A \times B &:= \{(a, b) | a \in A, b \in B\} \\ A^n &:= \underbrace{A \times \dots \times A}_{n \text{ Faktoren}} = \{(a_1, \dots, a_n) | a_1, \dots, a_n \in A\} \end{aligned}$$

## Beispiel

■  $\Sigma_{lat} \times \Sigma_{lat} = \{(a, a), (a, b), \dots, (z, z)\}$

■  $\Sigma_{lat}^3 = \{(a, a, a), (a, a, b), \dots, (z, z, z)\}$

# Kleene-Abschluss

## Definition

Ein Wort  $\omega$  ist ein endliches — ggf. leeres — Tupel  $(w_1, w_2, \dots, w_n) \in \Sigma^n$  von Zeichen  $w_k \in \Sigma$  eines Alphabets mit Länge  $|\omega| = n$  der Anzahl der Zeichen.

- Wörter werden meist ohne Klammern geschrieben; d. h.  $\omega = w_1 w_2 \dots w_n$ .
- Das leere Wort (das Wort ohne Zeichen) wird mit  $\varepsilon$  bezeichnet.
- Besondere Wortmengen:
  - $\Sigma^0 = \{\varepsilon\}$
  - $\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$
  - $\Sigma^+ = \bigcup_{n=1}^{\infty} \Sigma^n$

Die Operationen  $M^*$  und  $M^+$  auf einer Menge  $M$  werden als

- Kleene-\*-Abschluss oder
- Kleene-+-Abschluss bezeichnet.

## Beispiel

- $\Sigma_{lat}^* = \{\varepsilon, a, b, \dots, z, aa, ab, \dots, zz, aaa, \dots\}$
- $\Sigma_{lat}^+ = \{a, b, \dots, z, aa, ab, \dots, zz, aaa, \dots\}$

## Beispiel

Sei  $M = \{01, 2\}$ , so ergeben sich u.a. diese Wortmengen:

$$\begin{aligned} M^0 &= \{\varepsilon\} \\ M^1 &= \{01, 2\} \\ M^2 &= \{0101, 012, 201, 22\} \\ M^3 &= \{010101, 01012, 01201, 0122, 20101, 2012, 2201, 222\} \\ &\dots \\ M^+ &= M^1 \cup M^2 \cup \dots = \{01, 2, 0101, 012, 201, 22, 010101, 01012, \dots\} \\ M^* &= M^0 \cup M^+ = \{\varepsilon, 01, 2, 0101, 012, 201, 22, 010101, 01012, \dots\} \end{aligned}$$



## Beobachtung

Die Wortlänge  $|\omega|$  für ein  $\omega \in M^*$  hängt von der Definition des Alphabets ab. So ist in diesem Beispiel  $|222| = 3$  während  $|0101| = 2$  ist.

# Produkt und Konkatination

## Definition

Die Konkatination von zwei Wörtern  $\omega = (\omega_1, \dots, \omega_n)$  und  $v = (v_1, \dots, v_m)$  ist definiert als das Wort, das durch ein aneinanderreihen der beiden Wörter entsteht:

$$\omega \cdot v = \omega v = (\omega_1, \dots, \omega_n) \cdot (v_1, \dots, v_m) = \omega_1 \dots \omega_n v_1 \dots v_m$$

Das leere Wort ist  $\omega^0 = \varepsilon$  und die n-te Potenz von  $\omega$  ist:

$$\omega^n = \underbrace{\omega \cdot \dots \cdot \omega}_{n \text{ Faktoren}} \text{ für } n > 0$$

## Beispiel

Sei  $\Sigma = \{a, e, n, r\}$ , sowie  $\omega = na \in \Sigma^*$  und  $v = er \in \Sigma^*$ .

$\omega^2 = nana$ ,  $v\omega = erna$  und  $v\omega^2v = ernanaer$

# Abschluss-Eigenschaften

## Bemerkung

Der Begriff *Abschluss in obiger Definition* bedeutet:

Auf einer Menge mit einer Verknüpfung liefert jede Anwendung der Operation mit Elementen wieder ein Element aus der Menge.

## Beispiel

- die Subtraktion ist auf den natürlichen Zahlen nicht abgeschlossen,
- der Abschluss der natürlichen Zahlen bezüglich der Subtraktion sind die ganzen Zahlen.

Die Kleene-Abschlüsse und Multiplikationen werden später in regulären Ausdrücken auf Wörtern verwendet, damit ist dann der Abschluss oder das kartesische Produkt der Menge mit genau diesem Wort gemeint.

## Beispiel

$$\begin{aligned}(ab)^+ &= \{ab\}^+ &= \{ab, abab, ababab, \dots\} \\ cd^*e &= \{c\} \times \{d\}^* \times \{e\} &= \{ce, cde, cdde, cddde, \dots\}\end{aligned}$$



# Übung

1.1. Alphabet  $\Sigma = \{a, el, en, g, l, ste\}$

Gegeben sei das Alphabet  $\Sigma = \{a, el, en, g, l, ste\}$ . Welche der folgenden Worte liegen in  $\Sigma^4$ ?

$\omega_1 = galgen$ ,  $\omega_2 = stelle$ ,  $\omega_3 = sagen$ ,  $\omega_4 = lagen$ ,  $\omega_5 = allen$ ,  $\omega_6 = aalen$

---

**12.** Alphabet  $\Sigma = \{e, en, in, r, t, u\}$

Gegeben sei das Alphabet  $\Sigma = \{e, en, in, r, t, u\}$ . Welche der folgenden Worte liegen in  $\Sigma^5$ ?

$\omega_1 = \text{reiner}$ ,  $\omega_2 = \text{teurer}$ ,  $\omega_3 = \text{treuer}$ ,  $\omega_4 = \text{teuren}$ ,  $\omega_5 = \text{retten}$ ,  $\omega_6 = \text{teuer}$

---

# Übung

1.3. Alphabet  $\Sigma = \{e, g, in, l, s, ter\}$

Gegeben sei das Alphabet  $\Sigma = \{e, g, in, l, s, ter\}$ . Welche der folgenden Worte liegen in  $\Sigma^*$ ?

$\omega_1 = \text{tester}$ ,  $\omega_2 = \text{seile}$ ,  $\omega_3 = \text{lines}$ ,  $\omega_4 = \text{segel}$ ,  $\omega_5 = \text{seinen}$ ,  $\omega_6 = \text{erster}$

# Formale Sprachen

## Definition

Jede Teilmenge  $L \subseteq \Sigma^*$  ist eine formale Sprache über dem Alphabet  $\Sigma$ .

## Beispiel

Sei  $\Sigma = \{0, 1, 2\}$ , dann ist  $\Sigma^*$  die Menge oder Sprache von Wörtern aus den Ziffern 0, 1 oder 2 beliebiger Länge wie 101 oder auch 0001.

Die Menge  $M_3 \subset \Sigma^*$  der binären Zahlen ohne führende Nullen:

$$M_3 = \{0\} \cup \{1\} \times \{0, 1\}^* = \{0, 1, 10, 11, 100, 101, 110, 111, 1000, \dots\}$$

Die Menge  $M_2 \subset \Sigma^*$  von einer gleichen Anzahl von 0 und 1 in dieser Reihenfolge:

$$M_2 = \{0^n 1^n | n \in \mathbb{N}\} = \{01, 0011, 000111, 00001111, 0000011111, \dots\}$$

Die Wörter  $M_1 \subset \Sigma^*$  mit gleicher Anzahl von 0, 1 und 2 in dieser Reihenfolge:

$$M_1 = \{0^n 1^n 2^n | n \in \mathbb{N}\} = \{012, 001122, 000111222, 000011112222, \dots\}$$

Die Menge  $M_0 \subset \Sigma^*$  mit Wörtern der Länge von Zweierpotenzen:

$$M_0 = \{w \in \Sigma^* | |w| = 2^n, n \in \mathbb{N}_0\} = \{0, 1, 2, 00, 01, \dots, 21, 22, 0000, \dots\}$$

# Übung

## 1.4 Wörter bestimmen

Bestimmen Sie die Wörter der folgenden Sprache:

$$L = \{acx^m(zq)^n \mid n \in \{0, 1\}, m \in \{1, 2\}\}$$

---

### 15. Wörter bestimmen

Bestimmen Sie die Wörter der folgenden Sprache:

$$L = \{(b^m a)^l z a \mid m \in \{0, 1\}, l \in \{1, 2, 3\}\}$$

## 2. Abzählbarkeit und Gödelnummern

# Abzählbar (unendlich)

## Beobachtung

Selbst mit endlichen Alphabeten können formale Sprachen unendlich groß sein.

## Definition

Eine Menge  $M$  ist *abzählbar*, wenn die einzelnen Elemente abzählbar sind, es also eine bijektive Funktion  $f : N \rightarrow M$  von den natürlichen Zahlen  $N = \mathbb{N}$  oder einer Teilmenge der natürlichen Zahlen  $N \subset \mathbb{N}$  auf  $M$  gibt.

Wenn es keine solche Funktion geben kann, so ist die Menge *überabzählbar unendlich*.

## Satz

Jede endliche Menge ist abzählbar.

## Beweis

Eine endliche Menge  $M$  hat eine endliche Anzahl  $n = |M|$  von Elementen.

Wird nun beginnend von  $M_0 = M$  und  $k = 1$  in  $n$  Schritten jeweils ein Element  $m_k$  der Menge  $M_{k-1}$  entnommen mit  $M_k = M_{k-1} \setminus \{m_k\}$ , so ist induktiv  $|M_k| = |M_{k-1}| - 1 = n - k$  und es ist  $M_n = \emptyset$ .

Die Bijektion lautet dann  $f : N \rightarrow M$  mit  $f(k) = m_k$  mit  $N = \{1, \dots, n\}$ . ■

## Satz

Jede Teilmenge  $M \subseteq N$  einer abzählbaren Menge  $N = \{n_1, n_2, \dots\}$  ist abzählbar.

## Beweis

Sei  $f(k) = n_k$  die Abzählung der Menge  $N$ . Sei  $R = \{k \in \mathbb{N} | n_k \in M\}$ ; d. h. die Menge der Indizes der Elemente aus  $N$ , die in  $M$  sind. Dann ist die Einschränkung  $f|_R : R \rightarrow M$  von  $f$  genau die Abzählung, die die Abzählbarkeit von  $M$  beweist. ■

## Beispiel

Abzählbar unendliche Mengen sind — zum Beispiel:

- die geraden Zahlen  $\{2n | n \in \mathbb{N}\}$
- die Quadratzahlen  $\{n^2 | n \in \mathbb{N}\}$
- die Menge der Fakultäten  $\{n! | n \in \mathbb{N}\}$
- die ganzen Zahlen  $\mathbb{Z}$  mit der Funktion:

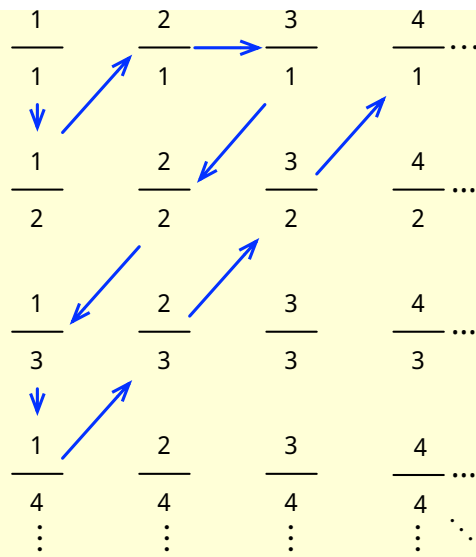
$$f(n) = \begin{cases} n/2 & \text{für } n \text{ gerade} \\ -(n-1)/2 & \text{für } n \text{ ungerade} \end{cases}$$

$$f(1) = 0, f(2) = 1, f(3) = -1, f(4) = 2, f(5) = -2, \dots$$

## Beispiel

Die rationalen Zahlen  $\mathbb{Q}$  sind abzählbar unendlich.





Rationale Zahlen können als Brüche dargestellt werden und mit Hilfe des Diagonalisierungsverfahren von Cantor (auch: Cantors erstes Diagonalargument) in eine Bijektion zu den natürlichen Zahlen gebracht werden.

Die 0 und alle negativen Brüche können wie zuvor eingeschoben werden. Auch alle rationalen Vektoren  $\mathbb{Q}^n$  in beliebiger Dimension  $n \in \mathbb{N}$  sind so abzählbar.

#### Satz

Für jede endliche Menge oder Alphabet  $\Sigma$  ist deren Kleene-Abschluss  $\Sigma^*$  abzählbar.

#### Beweis

Ist das Alphabet  $\Sigma$  leer, so ist auch  $\Sigma^*$  leer, und damit für  $N = \emptyset$  trivial abzählbar.

Ist  $\Sigma$  nicht leer, dann besitzt  $\Sigma$  mit Größe  $n = |\Sigma|$  eine Aufzählung  $m_k$  mit  $k = 1, \dots, n$ .

Jedes Wort  $w = m_{k_1} m_{k_2} \dots m_{k_l}$  kann dann im Stellenwertsystem zur Basis  $n + 1$  dargestellt werden:

$$1 + k_1 \cdot (n + 1)^{l-1} + k_2(n + 1)^{l-2} + \dots + k_l(n + 1)^0$$

und somit der Zahl  $1 + (k_1 k_2 \dots k_l)_{(n+1)}[1]$  zugeordnet werden.

#### Beispiel

Wort = e r n st  
Länge:  $l = 3$

Basis:

$$|\Sigma| + 1 = 5$$

höchstwertigste  
Stelle

2  
 $m_1 = e$

$$1 \cdot 5^2$$

Stelle

1  
 $m_3 = rn$

$$3 \cdot 5^1$$

niedwertigste  
Stelle

0  
 $m_4 = st$

$$4 \cdot 5^0$$

#### Legende

$\Sigma = \{e, i, rn, st\}$

$m_1 = e$

$m_2 = i$

$m_3 = rn$

$m_4 = st$

Die Abbildung  $f : N \rightarrow \Sigma^*$  mit  $N \subseteq \mathbb{N}$  ergibt sich für  $f(x)$  aus der Stellenwertdarstellung von  $x - 1 > 0$  zur Basis  $n + 1$  beginnend mit der höchstwertigen Ziffer  $k_1$  bis zur letzten Stelle  $k_l$ .

#### Wiederholung

Umrechnung einer Dezimalzahl in eine Zahl zur Basis  $n$ , erfolgt durch Division mit Rest durch  $n$  und die Reihenfolge der Reste ist dann die Stellenwertdarstellung, beginnend mit dem letzten Rest. D. h. der erste Rest ist die letzte Ziffer der Stellenwertdarstellung.

#### Beispiel

Umrechnung von  $5 = 5_{10}$  zur Basis 5:

1.  $5 / 5 = 1$  Rest 0 (letzte Ziffer/niederwertigste Stelle)
2.  $1 / 5 = 0$  Rest 1 (erste Ziffer/höchstwertige Stelle)

Die Stellenwertdarstellung ist dann  $10_5$ .

Gegenprobe:  $1 \cdot 5^1 + 0 \cdot 5^0 = 5$ .

### Beispiel

Umrechnung von  $7 = 7_{10}$  zur Basis 3:

1.  $7 / 3 = 2$  Rest 1
2.  $2 / 3 = 0$  Rest 2

Die Stellenwertdarstellung ist dann  $21_3$ .

Gegenprobe:  $2 \cdot 3^1 + 1 \cdot 3^0 = 7$

### Hinweis

Wenn an einer Zahl keine spezifische Basis angegeben ist, oder aus dem Kontext unmittelbar eine andere Basis anzunehmen ist (z. B. 2 oder 16), so ist die Basis 10 anzunehmen. D. h. die Dezimaldarstellung ist die Standarddarstellung und 34 wäre zum Beispiel Äquivalent zu  $34_{10}$ .

Das Bild  $f(x)$  ist dann das Wort  $m_{k_1}m_{k_2} \dots m_{k_l}$ .

Das leere Wort  $\epsilon$  wird von 1 abgebildet und entsprechend ist  $f(1) = \epsilon$ . ■

### Beispiel

Sei  $\Sigma = \{e, i, rn, st\}$  mit Aufzählung  $m_1 = e$ ,  $m_2 = i$ ,  $m_3 = rn$ , und  $m_4 = st$ , dann haben die folgenden Wörter diese Abzählung nach Stellenwert:

$x$	1	2	3	4	5
	1	1 + 1 $1_5 + 1_5$	1 + 2 $1_5 + 2_5$	1 + 3 $1_5 + 3_5$	1 + 4 $1_5 + 4_5$
		[2]			
Wort	$\epsilon$	e	i	rn	st
$f(x)$	$f(1) = \epsilon$	$f(2) = e$	$f(3) = i$	$f(4) = rn$	$f(5) = st$ (Anm.: $k$ ist 4 für $st$ )

$x$	...	$7 = 1 + 6$ $12_5 = 1_5 + 11_5$	$8 = 1 + 7$ $13_5 = 1_5 + 12_5$	...	$45 = 1 + 44$ $140_5 = 1_5 + 134_5$	...
	...	1 + $1 \cdot 5 + 1$ 1 + $11_5$	1 + $1 \cdot 5 + 2$ 1 + $12_5$	...	1 + $1 \cdot 25 + 3 \cdot 5 + 4$ 1 + $134_5$	...
Wort	...	ee	ei	...	ernst	...

Unbesetzt bleibt, wo eine 0 in der Stellenwertdarstellung vorliegt. Zum Beispiel ist  $f(6) = 1 + 1 \cdot 5^1 + 0 \cdot 5^0 = 1_5 + 10_5$ .

### Satz

Jede formale Sprache ist abzählbar.

### Beweis

Da jede formale Sprache  $L$  über einem endlichen Alphabet  $\Sigma$  definiert ist, ist das eine direkte Folge aus vorherigem Satz, dass  $\Sigma^*$  abzählbar ist, und wie zuvor gezeigt damit auch die Teilmenge  $L \subseteq \Sigma^*$  abzählbar ist. ■

## Abzählen mit Hilfe von Gödelnummern

Gödelnummern unterstützen abzählbare un-/endliche Mengen. Letzteres (abzählbar unendlich) ist mit einem einfachen Stellenwertsystem zur Basis der Anzahl der Elemente und des somit (zwangsweise) unendlichen Alphabets nicht möglich.

### Definition

Sei  $(p_n)$  die Folge der Primzahlen:

$$p_1 = 2, p_2 = 3, p_3 = 5, p_4 = 7, p_5 = 11, p_6 = 13, \dots$$

Für eine abzählbare Menge  $M = m_1, m_2, \dots$  ist die Gödelnummer  $c_M : M^* \rightarrow \mathbb{N}$  des Tupels  $w = (m_{k_1}, m_{k_2}, \dots, m_{k_l})$  gegeben durch

$$c_M(w) = p_1^{k_1} \cdot p_2^{k_2} \cdot \dots \cdot p_l^{k_l} = \prod_{i=1}^l p_i^{k_i}$$

### Wiederholung

**Fundamentalsatz der Arithmetik:** Jede natürliche Zahl  $n > 1$  kann eindeutig als ein Produkt von Primzahlen geschrieben werden, wobei die Reihenfolge der Primfaktoren ignoriert wird. D. h. die Gödelnummer  $c_M(w)$  ist eineindeutig für jedes Wort  $w \in M^*$ .

### Bemerkung

Die Primfaktorzerlegung einer Zahl  $x$  kann wie folgt erfolgen, wenn wir die Liste der Primzahlen  $p = [2, 3, 5, 7, 11, 13, 17, \dots]$  haben:

```
1 # Um die Primzahlen zu erzeugen, kann z. B. das Sieb des Eratosthenes
2 # verwendet werden. Die Bestimmung der Primzahlen ist hier
3 # jedoch nicht Gegenstand.
4 p = [2, 3, 5, 7, 11, 13, 17, 19]
5
6 def primfaktorzerlegung(x, i = 0):
7     c = 0 # Häufigkeit des Primfaktors
8     while x % p[i] == 0:
9         c += 1
10        x = x // p[i]
11    factor = str(p[i]) + "^" + str(c)
12    if x == 1:
13        return factor
14    else:
15        return factor + " * " + primfaktorzerlegung(x, i+1)
```

### Beispiel

Primfaktorzerlegung von 10:

■  $X = 10, p_1 = 2$

■  $10 / 2 = 5$  Rest 0  $\Rightarrow$  **2 ist ein Primfaktor**

■  $5 / 2 = 2$  Rest 1  $\Rightarrow$  2 ist kein weiterer Primfaktor; um die nächsten Primfaktoren zu bestimmen setzen wir  $X = 5$

■  $X = 5, p_2 = 3$

■  $5 / 3 = 1$  Rest  $2 \Rightarrow 3$  ist *kein* Primfaktor, da wir einen Rest haben;  $X$  hat unverändert den Wert 5

■  $X = 5, p_3 = 5$

■  $5 / 5 = 1$  Rest  $0 \Rightarrow$  **5 ist ein Primfaktor**

$$10 = 2^1 \cdot 5^1$$

### Beispiel

Primfaktorzerlegung von 12:

■  $X = 12, p_1 = 2$

■  $12 / 2 = 6$  Rest  $0 \Rightarrow$  **2 ist ein Primfaktor**

■  $6 / 2 = 3$  Rest  $0 \Rightarrow$  **2 ist noch einmal ein Primfaktor**

■  $3 / 2 = 1$  Rest  $1 \Rightarrow X' = 3$

■  $X = 3, p_2 = 3$

■  $3 / 3 = 1$  Rest  $0 \Rightarrow$  **3 ist ein Primfaktor**

$$12 = 2^2 \cdot 3^1$$

### Beispiel

Sei  $\Sigma = \{e, i, rn, st\}$  mit Aufzählung  $m_1 = e, m_2 = i, m_3 = rn$  und  $m_4 = st$ , dann haben die folgenden Wörter diese Gödelnummern:

Wort	$\epsilon$	$e$	$i$	$rn$	$st$	$e rn st$
$c_M(w)$	$2^0 = 1$	$2^1$	$2^2$	$2^3$	$2^4$	$2^1 \cdot 3^3 \cdot 5^4 = 33750$
		$p_1^{k_1=1}$	$p_1^{k_2=2}$			

### Beobachtung

Unbesetzt bleibt, wo bis zum höchsten Primzahlfaktor davor eine Primzahlpotenz 0 ist.

Z. B. ist die Primzahlzerlegung von  $10 = 2^1 \cdot 3^0 \cdot 5^1$ . Somit gäbe es an der zweiten Stelle *kein Zeichen* was unsinnig ist.

### Satz

Die Menge von endlichen Folgen  $P = \{p = (w_1, w_2, \dots, w_n) | w_k \in L, n \in \mathbb{N}\}$  aus Wörtern einer formalen Sprache  $L \subseteq \Sigma^*$  (also Programmen) über einem Alphabet  $\Sigma$  ist abzählbar.

### Beweis

Jede formale Sprache  $L \subseteq \Sigma^*$  ist abzählbar. Damit kann nach Definition für jede Folge  $p \in P$  injektiv eine Gödelnummer  $c_L(p)$  über  $L$  bestimmt werden.

Auf der Menge  $N = \{x = c_L(p) | p \in P\}$  kann die Umkehrung  $f : N \rightarrow P$  von  $c_L$  auf  $P$  eingeschränkten bijektiven Funktion  $c_{L|P} : P \rightarrow N$  bestimmt werden, und damit ist  $P$  abzählbar. ■

[2] Wir haben immer  $1 + \dots$ , da wir noch das leere Wort  $\epsilon$  haben.

[1] Die Darstellung  $(k_1 k_2 \dots k_l)_{(n+1)}$  ist die Stellenwertdarstellung zur Basis  $n + 1$  des Wortes  $w$ .



# Überabzählbar unendlich

## Satz

Die Menge der reellen Zahlen  $r \in (0, 1) \subset \mathbb{R}$  ist überabzählbar unendlich.

## Beweis

### Cantor's (zweites) Diagonalargument

Angenommen die reellen Zahlen sind als Binärbrüche wie folgt abzählbar:

$$\begin{array}{rcl} r_1 & = & 0, x_{11} x_{12} x_{13} x_{14} x_{15} \dots \\ r_2 & = & 0, x_{21} x_{22} x_{23} x_{24} x_{25} \dots \\ r_3 & = & 0, x_{31} x_{32} x_{33} x_{34} x_{35} \dots \\ r_4 & = & 0, x_{41} x_{42} x_{43} x_{44} x_{45} \dots \\ & \vdots & \vdots \end{array}$$

Sei jetzt  $r = 0, \overline{x_{11}} \overline{x_{22}} \overline{x_{33}} \overline{x_{44}} \overline{x_{55}} \dots \in (0, 1)$ , dann ist  $r$  nicht in der Abzählung und es liegt ein Widerspruch zur Annahme vor.  $\mathbb{R}$  ist also überabzählbar unendlich.

-----  
 $\bar{x}$  ist das einfache Komplement von  $x$ . Das bedeutet, dass 0 durch 1 und 1 durch 0 ersetzt wird.

Beachte, dass  $r$  über die gesamte (unendliche) Diagonale definiert ist und dadurch zu jeder bestehenden Zahl unterschiedlich sein muss; d. h.  $r$  ist nicht gleich zu  $r_1$  in der ersten Stelle, nicht gleich zu  $r_2$  in der zweiten Stelle, nicht gleich zu  $r_3$  in der dritten Stelle, ... und nicht gleich  $r_n$  in der  $n$ -ten Stelle.

Die Kardinalität (bereits) der Menge der reellen Zahlen im Bereich  $(0, 1)$  ist also größer als die der natürlichen Zahlen. ■

# Schlussfolgerungen aus der Überabzählbarkeit

Angenommen:

- jedes in einer formalen Sprache geschriebenes Programm löst ein Problem
- wir interpretieren dies als Berechnung einer Lösung

So sind dies verschwindend wenige lösbare Probleme verglichen schon mit der Reichhaltigkeit der reellen Zahlen im Intervall  $(0, 1)$ .

## Schlussfolgerung

Soweit davon auszugehen ist, dass die Teilmenge der in der Realität tatsächlich relevanten reellen Zahlen tatsächlich auch überabzählbar ist, wird es nie möglich sein, für alle Fragestellungen über solche Zahlen Lösungen in der Form von Programmen über einer gegebenen formalen Sprache zu formulieren.

## Schlussfolgerung

Gleichzeitig ist aber auch die Anzahl der formalen Sprachen sehr groß.

## Beweis

Für jede reelle Zahl  $x \in \mathbb{R}$  mit Nachkommastellen  $r_1 r_2 \dots$  gibt es eine formale Sprache  $L_x$  über  $\Sigma_{\text{Zahl}}$ :

$$L_x = \{r_1 r_2 \dots r_n \in \Sigma_{\text{Zahl}}^* \mid x \text{ hat die ersten } n \text{ Nachkommastellen } r_1 \dots r_n\}$$

Beispielsweise ist  $L_\pi = \{1, 14, 141, 1415, 14159, 141592, 1415926, \dots\}$ . Damit ist die Anzahl der formalen Sprachen mindestens so groß, wie die Anzahl reeller Zahlen im Intervall  $(0, 1)$ , also aller möglichen Nachkommastellen in  $\mathbb{R}$ , zuzüglich der 0, und damit nach vorherigem Satz überabzählbar unendlich. ■

# Übung

## 2.1. Stellenwerte I

Gegeben sei das Alphabet  $\Sigma = \{a, gen, i, re\}$  mit Aufzählung in dieser Reihenfolge. Bestimmen Sie die Zahlen  $n$  nach Stellenwert mit Bild  $f(n)$  der Wörter *regen*, *aare* und die Worte mit Stellenwert 15, 118.



---

## 2.2. Stellenwerte II

Gegeben sei das Alphabet  $\Sigma = \{e, h, r, ste\}$  mit Aufzählung in dieser Reihenfolge. Bestimmen Sie die Zahlen  $n$  nach Stellenwert mit Bild  $f(n)$  der Wörter *steh*, *rehe* und die Worte mit Stellenwert 45, 1417.

---

# Übung

## 2.3. Gödelnummern I

Gegeben sei das Alphabet  $\Sigma = \{e, l, ste, te\}$  mit Aufzählung in dieser Reihenfolge. Bestimmen Sie die Gödelnummer  $c(w)$  der Wörter *este*, *elle* und die Worte mit Gödelnummer 720, 12600.

---

## 2.4. Gödelnummern II

Gegeben sei das Alphabet  $\Sigma = \{h, he, re, ste\}$  mit Aufzählung in dieser Reihenfolge. Bestimmen Sie die Gödelnummer  $c(w)$  der Wörter *steh*, *reste* und die Worte mit Gödelnummer 144, 1500.

---

# Übung

## 2.5. Gödelnummern und ChatGPT

Eine Befragung von ChatGPT zum Thema Gödelnummern ergab, dass ChatGPT vorgeschlagen hat allen Zeichen  $a \in \Sigma$  eine Primzahl zuzuordnen und dann für das Vorkommen eines Zeichens  $a$  an Stelle  $i$  den aktuellen Wert mit der Primzahl des Zeichens hoch  $i$  zu multiplizieren.

### Beispiel

Sei  $\Sigma = \{a, b, c, d\}$

Zuweisung von Primzahlen an Symbole:  $a \rightarrow 2, b \rightarrow 3, c \rightarrow 5, d \rightarrow 7$

Für das Wort:  $abac$  wäre nach dem von ChatGPT vorgeschlagenen Verfahren die Gödelnummer  $c(abac) = 2^1 \cdot 3^2 \cdot 2^3 \cdot 5^4 = 90\,000$ .

Bewerten Sie diesen Vorschlag.

### 3. Verknüpfungen und Entscheidbarkeit

---

# Verknüpfungen von formalen Sprachen

## Satz

Sind  $L_1$  und  $L_2$  zwei formale Sprachen über den Alphabeten  $\Sigma_1$  und  $\Sigma_2$ , so gilt:

1. Die Vereinigung  $L_{\cup} = L_1 \cup L_2$  ist eine formale Sprache über dem Alphabet  $\Sigma_1 \cup \Sigma_2$ .
2. Der Schnitt  $L_{\cap} = L_1 \cap L_2$  ist eine formale Sprache über dem Alphabet  $\Sigma_1 \cup \Sigma_2$ .

Die Wörter aus  $L_{\cap}$  sind aber natürlich in  $(\Sigma_1 \cap \Sigma_2)^*$  enthalten.

## Beweis

Die Vereinigung der Alphabete  $\Sigma = \Sigma_1 \cup \Sigma_2$ , also zweier endlicher Mengen, ist wieder eine endliche Menge und damit ein Alphabet. Da sowohl  $\Sigma_k \subseteq \Sigma$  für  $k = 1, 2$ , sind  $L_1$  und  $L_2$  auch Sprachen über  $\Sigma$  und es gilt  $L_k \subseteq \Sigma_k^* \subseteq \Sigma^*$ , da die Teilmengenbeziehung in jeder Mengengenpotenz und damit auch in deren Vereinigung gilt. Damit sind auch  $L_1 \cup L_2 \subseteq \Sigma^*$  und  $L_1 \cap L_2 \subseteq \Sigma^*$  und damit Sprachen über  $\Sigma = \Sigma_1 \cup \Sigma_2$ . ■

## Satz

Sind  $L_1$  und  $L_2$  zwei formale Sprachen über den Alphabeten  $\Sigma_1$  und  $\Sigma_2$ , so gilt:

3. Das Komplement  $\overline{L_k} = \Sigma_k^* \setminus L_k, k = 1, 2$  ist formale Sprache über Alphabet  $\Sigma_k$ .

## Beweis

Nach Definition der Mengendifferenz gilt  $\Sigma_k^* \setminus L_k \subseteq \Sigma_k^*$ . Somit ist  $\overline{L_k} \subseteq \Sigma_k^*$  und somit eine Sprache über  $\Sigma_k$ . ■

## Satz

Sind  $L_1$  und  $L_2$  zwei formale Sprachen über den Alphabeten  $\Sigma_1$  und  $\Sigma_2$ , so gilt:

4. Das Produkt  $L_1 L_2 = \{w_1 w_2 | w_1 \in L_1, w_2 \in L_2\}$  ist eine formale Sprache über  $\Sigma_1 \cup \Sigma_2$ .

## Beweis

Für  $L = L_1 \cup L_2$  ist  $L_1 \subseteq L$  und  $L_2 \subseteq L$ .

$L$  ist somit eine Sprache über  $\Sigma = \Sigma_1 \cup \Sigma_2$  nach Satz 1. Damit ist jedes Wort  $w \in L \subseteq \Sigma^*$  in einem  $w \in \Sigma^k$  für ein bestimmtes  $k$  enthalten. Ebenso ist damit  $w_1 w_2 \in \Sigma^{k_1} \Sigma^{k_2} = \Sigma^{k_1+k_2} \subseteq \Sigma^*$ . Damit ist  $LL \subseteq \Sigma^*$  und damit  $L_1 L_2 \subseteq LL \subseteq \Sigma^*$  Sprache über  $\Sigma$ . ■

## Satz

Sind  $L_1$  und  $L_2$  zwei formale Sprachen über den Alphabeten  $\Sigma_1$  und  $\Sigma_2$ , so gilt:

5. Kleensche Abschlüsse  $L_k^*$  und  $L_k^+, k = 1, 2$  sind formale Sprachen über  $\Sigma_k$ .

## Beweis



### Beobachtung

Zunächst ist  $\varepsilon \in \Sigma_k^*$ , somit reicht es für  $L_k^+$  zu argumentieren.

- Jedes Wort  $w \in L_k^+$  ist in  $w \in L_k^n$  für ein bestimmtes  $n$ .
- Damit gibt es Teilworte  $m_1 m_2 \dots m_n = w$  mit  $m_i \in L_k$
- Da  $L_k \subseteq \Sigma_k^*$  gibt es  $p_i$ , so dass  $m_i \in \Sigma_k^{p_i}$
- Damit ergibt sich, dass  $m_1 \dots m_n \in \Sigma_k^{\sum p_i}$  liegt und damit in  $\Sigma_k^{\sum p_i} \subseteq \Sigma_k^*$
- Damit dies für alle Worte in  $L_k^+$  gilt, ist  $L_k^+ \subseteq \Sigma_k^*$  und damit eine Sprache über  $\Sigma_k$  ■

## Beispiel

### Komplement einer Sprache

#### Gegeben

Alphabet:  $\Sigma_k = \{a, b\}$

Sprache:  $L_k$  Alle Wörter, die mit dem Symbol  $a$  beginnen.

$$L_k = \{a, aa, ab, aaa, aab, \dots\}$$

#### Komplement der Sprache:

Das Komplement  $\overline{L_k}$  enthält alle Wörter aus  $\Sigma_k^*$ , die *nicht* mit  $a$  beginnen. Das bedeutet:

$$\overline{L_k} = \{\epsilon, b, bb, ba, bba, bbb, \dots\}$$

# Existenz der Abzählbarkeit

## Wiederholung

Sind  $L_1$  und  $L_2$  abzählbar, so sind mit entsprechenden Anpassungen auch

- Vereinigung,
- Schnitt und
- Produkt

abzählbar.

## Beobachtung

Die Abzählbarkeit des Komplements kann nicht so einfach beantwortet werden!

Dies ist jedoch kein Problem, da jede formale Sprache abzählbar ist und damit auch ihr Komplement.

## ? Frage

Kann mit dem Wissen der Existenz auch die tatsächliche Abzählung angegeben werden?

## Zusammenfassung

Wir unterscheiden deswegen die einfache und nicht konstruktivistische Erkenntnis einer Abzählbarkeit von einer konstruktiven und praktischen Aufzählbarkeit.

## Definition

Eine Menge oder Sprache  $M$  ist **aufzählbar** oder **rekursiv aufzählbar**, wenn eine surjektive Abbildung  $f : \mathbb{N} \rightarrow M$  bekannt ist, die nach endlichen Schritten für jedes  $n \in \mathbb{N}$  die Berechnung von  $f(n)$  ermöglicht, falls  $M \neq \emptyset$ . Daraus ergibt sich eine Aufzählung von  $M$  durch die Folge  $(f(1), f(2), \dots)$ .

## Bemerkung

Die Bedeutung der „Berechenbarkeit“ wird später im Sinne eines „Programms“ erklärt.

---

„aufzählbar“: bezieht sich auf die Existenz der Aufzählung als „berechenbare Funktion“,

„rekursiv aufzählbar“: bezieht sich auf die Existenz eines „Programms“, was aber hier äquivalent ist.



# Aufzählbarkeit

## Satz

Sei  $\Sigma$  ein Alphabet, dann ist  $\Sigma^*$  aufzählbar.

## Beweis

Die Konstruktion aus dem früheren Satz zur Abzählbarkeit von  $\Sigma^*$  ist schon eine konstruktive Aufzählung von  $\Sigma^*$ .

Die nicht zugeordneten natürlichen Zahlen werden beispielsweise auf das jeweils zuletzt zugeordneten Wort abgebildet. ■

Zwischen den Bezeichnungen „aufzählbar“ zu „abzählbar“ ist der relevante Unterschied in der konstruktiven Kenntnis der Aufzählbarkeit im Gegensatz von der nicht konstruktiven Gewissheit der Abzählbarkeit.

### Achtung!

Es ist aber kein Verfahren bekannt, wie aus einer allgemeinen Aufzählung einer Sprache konstruktiv eine Aufzählung des Komplements abgeleitet werden kann. Das Gleiche gilt bei zwei aufgezählten Sprachen für deren Schnitt.

---

Die Übertragung der Eigenschaft der Aufzählbarkeit muss mit Angabe eines ausführbaren Algorithmus erfolgen.

So kann - wie bei der Aufzählung von  $\mathbb{Z}$  - bei der Vereinigung abwechselnd die eine oder die andere Aufzählung verwendet werden. Die Aufzählung der rationalen Zahlen kann nach dem vorgestellten Verfahren von Cantor erfolgen. Die gilt ggf. auch für das Produkt.

# Entscheidungsproblem

## Definition

Das *Entscheidungsproblem* bezeichnet die Frage, ob für ein Problem ein ausführbares Verfahren angegeben werden kann, mit dem in endlich vielen Schritten eine Entscheidung für das Problem bestimmt wird.

Ein Problem ist ...

**entscheidbar:** wenn ein solches Verfahren existiert

**nicht-entscheidbar:** wenn es ein solches Verfahren nicht geben kann

**semi-entscheidbar:** wenn ein Verfahren existiert, das nach endlich vielen Schritten die Entscheidung für eine Klasse von möglichen Antworten bestimmt

# Wortproblem

(Ein Beispiel für ein Entscheidbarkeitsproblem.)

## Definition

Sei  $L$  eine Sprache über  $\Sigma$  und  $w \in \Sigma^*$ . Das Wortproblem bezeichnet die Frage, ob  $w$  Teil der Sprache ist, also entweder  $w \in L$  oder  $w \notin L$  gilt.

## Satz

Sind  $L$  und  $\bar{L}$  aufzählbare Sprachen über dem Alphabet  $\Sigma$ , so ist das Wortproblem  $w \stackrel{?}{\in} L$  für ein  $w \in \Sigma^*$  entscheidbar.

Dann werden  $L$  und  $\bar{L}$  als *entscheidbare Sprachen* oder *rekursive Sprachen* bezeichnet.

## Beweis

Es seien  $f_L : \mathbb{N} \rightarrow L$  und  $f_{\bar{L}} : \mathbb{N} \rightarrow \bar{L}$  die Aufzählungen von  $L$  und  $\bar{L}$ .

Abwechselnd wird aufsteigend — beginnend bei  $k = 1$  — das Wort  $w$  mit  $f_L(k)$  und  $f_{\bar{L}}(k)$  verglichen.

Nach endlicher Anzahl von Schritten ist  $f_L(k) = w$ , dann ist  $w \in L$ , oder  $f_{\bar{L}}(k) = w$ , dann ist  $w \notin L$ . ■

Es ist wichtig, dass der Vergleich von  $w$  abwechselnd mit  $L$  und  $\bar{L}$  (aufsteigend) erfolgt, da wir sonst nicht nach einer endlichen Anzahl von Schritten garantiert zu einem Ergebnis kommen.

Ist  $L$  aufzählbar, doch  $\bar{L}$  nicht, so endet das Verfahren, genau dann wenn  $w \in L$  ist. Daher ist Wortproblem aufzählbarer Sprachen semi-entscheidbar.

## Satz

Jede entscheidbare Sprache ist aufzählbar.

## Beweis

Jede formale Sprache  $L$  basiert auf einem Alphabet  $\Sigma_L$ . Damit ist der Abschluss  $\Sigma_L^*$  mit  $f_{\Sigma^*}$  aufzählbar und  $L \subseteq \Sigma_L^*$ . Entweder ist die Sprache  $L$  leer, oder es gibt ein Wort  $w_0 \in L$ .

Wenn  $L$  entscheidbar ist, so kann für jedes  $n \in \mathbb{N}$  in endlichen Schritten bestimmt werden, ob  $f_{\Sigma_L^*}(n) \in L$  ist. Wenn ja, so ist  $f_L(n) = f_{\Sigma_L^*}(n)$ , und sonst  $f_L(n) = w_0$ . ■

Damit gilt:

rekursive bzw. entscheidbare Sprache  $\Rightarrow$  rekursiv aufzählbare Sprache

semi-entscheidbare Sprache  $\Leftarrow$  rekursiv aufzählbare Sprache

## Beobachtung

Eine rekursive Aufzählung kann die Sprache völlig durcheinander aufzählen.

Es ist nie sicher, ob frühe Lücken zur Stellenwertaufzählung später aufgefüllt werden.

# Das Collatz-Problem

## Definition

Die Collatz-Funktion  $f : \mathbb{N} \rightarrow \mathbb{N}$  ist definiert als:

$$f(n) = \begin{cases} n/2 & \text{für gerade } n \\ 3n + 1 & \text{für ungerade } n \end{cases}$$

Das Collatz-Problem besteht darin, für ein gegebenes  $n$  die Folge  $f(n), f(f(n)), f(f(f(n))), \dots$  zu betrachten und zu entscheiden, ob die Folge irgendwann den Wert 1 erreicht.

## Beispiel

$$\begin{aligned} f(6) &= 3, f(3) = 10, f(10) = 5, f(5) = 16, \\ f(16) &= 8, f(8) = 4, f(4) = 2, f(2) = 1, \dots \end{aligned}$$

Die Folge erreicht für  $n = 6$  den Wert 1 nach 8 Schritten.

## Satz

Das Collatz-Problem ist semi-entscheidbar.

## Beweis

Die Collatz-Folge kann für ein gegebenes  $n$  in endlich vielen Schritten berechnet werden.

Wenn die Folge den Wert 1 erreicht, so ist das Problem entschieden.

Wenn die Folge nicht den Wert 1 erreicht, so ist das Problem nicht entschieden, aber es ist auch nicht sicher, ob die Folge den Wert 1 nicht doch noch erreicht. ■

Das Collatz-Problem kann direkt in eine Collatz-Sprache über  $\Sigma_{\text{Zahl}}$  übertragen werden:

$$L_{\text{Collatz}} = \{n \in \mathbb{N} \mid \exists k \in \mathbb{N}_0 : f^k(n) = 1\}$$

Das Wortproblem auf dieser Sprache ist damit — hier nach Definition des Problems statt einer Aufzählung — ebenso mindestens semi-entscheidbar.

Ob das Problem auch entscheidbar ist, konnte bisher niemand beantworten. Die naive Methode des Ausprobierens, ob es überhaupt ein  $w \in \mathbb{N}$  mit  $w \notin L_{\text{Collatz}}$  gibt, hat trotz intensiver Suche bisher nicht geendet.

# Das Halteproblem

## Definition

Das Halteproblem ist die Fragestellung, ob die Ausführung eines Programms  $p$  bei gegebenen Eingabedaten  $x$  nach endlichen Schritten terminiert.

Das Halteproblem ist die verallgemeinerte Fragestellung zum Collatz-Problem. Entsprechend ist die äquivalente Sprache:

$L_{\text{Halteproblem}} =$

$$\{(p, x) \in \Sigma_{\text{Unicode}}^* \times \Sigma_{\text{Unicode}}^* \mid p(x) \text{ terminiert nach endlichen Schritten} \}$$

nur semi-entscheidbar, da durch Ausführung des Programms nur  $(p, x) \in L_{\text{Halteproblem}}$  gezeigt werden kann.

## Bemerkung

Alan Turing konnte beweisen, dass es keinen Algorithmus gibt, der die Entscheidung  $(p, x) \notin L_{\text{Halteproblem}}$  für beliebige  $p$  und  $x$  in endlicher Zeit beantworten kann.

# Übung

## 3.1. Collatz-Funktion

Die parametrisierte Collatz-Funktion  $f_{\alpha,\beta}(n) : \mathbb{N} \rightarrow \mathbb{N}$  laute für  $\alpha, \beta \in \mathbb{N}$ :

$$f_{\alpha,\beta}(n) = \begin{cases} n/2 & \text{für } n \text{ gerade} \\ \alpha \cdot n + \beta & \text{sonst} \end{cases}$$

1. Bestimmen Sie mit einem Programm das kleinste  $k \in \mathbb{N}$  für das  $f_{3,1}^k(27) = 1$  ist.
2. Sei die Sprache  $L_{\text{Collatz}_{3,7}} = \{n \in \mathbb{N} \mid \exists k \in \mathbb{N}_0 : f_{3,7}^k(n) = 1\}$ .

Bestimmen Sie mit einem Programm die Menge  $M = \bar{L}_{3,7} \cap [1, 20]$ .

# Übung

## 3.2. Rekursive Sprachen

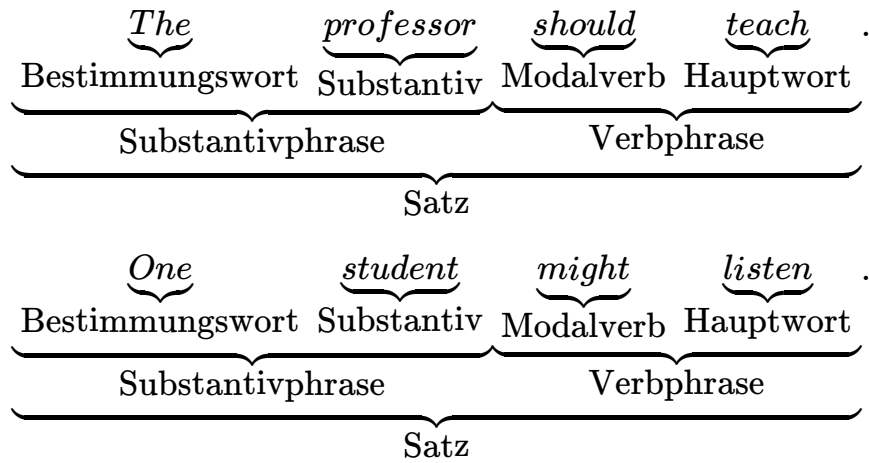
Es seien  $L_1$  und  $L_2$  rekursive Sprachen über dem Alphabet  $\Sigma$ . Sei  $L = L_1 \setminus L_2$ .

Zeigen oder widerlegen Sie, dass  $L$  eine rekursive Sprache über  $\Sigma$  sei.



## 4. Grammatiken

# Englische Grammatik (Beispielhaft)



Ein Satz  $S$  wird mit diesen Regeln  $R$  gebildet:

- Ein Satz besteht aus einer Substantivphrase und einer Verbphrase.
- Eine Substantivphrase hat ein optionales Bestimmungswort und ein Substantiv.
- Eine Verbphrase besteht aus optionalem Modalverb und einem Hauptverb.
- Ein Bestimmungswort ist *The* oder *One*.
- Ein Substantiv ist *student* oder *professor*.
- Ein Modalverb ist *should* oder *might*.
- Ein Hauptverb ist *listen* oder *teach*.

Darin wurden diese Variablen  $V$  und Symbole  $T$  verwendet:

- $V$ : {Satz, Substantivphrase, Verbphrase, Bestimmungswort, Substantiv, Modalverb, Hauptverb}
- $T$ : {The, One, student, professor, should, might, listen, teach}.

# Grammatiken

## Definition

Eine Grammatik ist ein Tupel  $G = (V, T, R, S)$ , wo

$V$ : das Alphabet der Variablen,

$T$ : das Alphabet der Terminalen Symbole mit  $V \cap T = \emptyset$ ,

$R = r_1, \dots, r_n$ : die endliche Menge der Regeln  
 $r_k: (V \cup T)^* \setminus T^* \rightarrow (V \cup T)^*$

$S \in V$ : das Startsymbol ist.

Die Regeln von Grammatiken werden auch Produktionen genannt

# Ableitungen

## Definition

Sei  $G = (V, T, R, S)$  eine Grammatik. Eine **Ableitung** ist die Anwendung einer Regel  $r \in R$  mit  $a \mapsto b$  auf das Wort  $w_1 \in (V \cup T)^*$  zum Wort  $w_2 \in (V \cup T)^*$ , geschrieben  $w_1 \xRightarrow{r} w_2$ , wenn es  $x, y \in (V \cup T)^*$  gibt, so dass:

$$\begin{array}{ccccc} w_1 & = & x & a & y \\ \Downarrow_r & & & \mapsto_r & \\ w_2 & = & x & b & y \end{array}$$

## Definition

Eine **transitive Ableitung**  $w_1 \xRightarrow{*} w_n$  ist die Anwendung keiner oder beliebig vieler Regeln  $r \in R$ , um von  $w_1$  auf  $w_n$  zu schließen. Die Sprache einer Grammatik  $L(G)$  ist die Menge aller möglichen Wörter, die durch die Regeln der Grammatik transitiv aus dem Startsymbol  $S$  abgeleitet werden können:

$$L(G) := \{w \in T^* \mid S \xRightarrow{*} w\}$$

## Zusammenfassung

Ableitungen aus einer Grammatik definieren eine Sprache.

# Eine Grammatik für boolesche Ausdrücke

Eine Grammatik für boolesche Terme ist  $G_{\text{Logik}} = (V, T, R, S)$  mit

$$\begin{aligned} V &= \{\text{Term, Literal, Variable}\} \\ T &= \Sigma_{\text{Logik}} = \{\vee, \wedge, \neg, (, ), 0, 1, a, \dots, z\} \\ R &= \{r_1, r_2, r_3, r_4\}, wo \\ &\quad r_1 : \text{Term} \mapsto \text{Literal} | \text{Variable} | \neg \text{Term} | ( \text{Term} ) \\ &\quad r_2 : \text{Term} \mapsto \text{Term} \vee \text{Term} | \text{Term} \wedge \text{Term} \\ &\quad r_3 : \text{Literal} \mapsto 0 | 1 \\ &\quad r_4 : \text{Variable} \mapsto a | \dots | z \\ S &= \text{Term} \end{aligned}$$

Bemerkung

$r_2 : \text{Term} \mapsto \text{Term} \vee \text{Term} | \text{Term} \wedge \text{Term}$  ist zu interpretieren als:
 
$$\begin{cases} r_{2.1} : \text{Term} \mapsto \text{Term} \vee \text{Term} \\ r_{2.2} : \text{Term} \mapsto \text{Term} \wedge \text{Term} \end{cases}$$

Eine Ableitung des Terms  $S \xRightarrow{*} (a \wedge b) \vee c \in L(G_{\text{Logik}})$  kann dann so ablaufen:

Regel		S = Term			
r2.1				↓	
		Term		∨	Term
r1.4,r1.2		↓			↓
		(Term)		∨	Variable
r2.2,r4		↓			↓
	( Term	∧	Term )	∨	c
r1.2,r1.2	↓		↓		
	( Variable	∧	Variable )	∨	c
r4,r4	↓		↓		
	( a	∧	b )	∨	c

# Übung

## 4.1. Sprache bestimmen: ersw

Bestimmen Sie die Sprache  $L(G)$  für  $G = (V, T, R, S)$ :

$$V = \{A, B, C\}$$

$$T = \{e, r, s, w\}$$

$$R = \{r_1, r_2, r_3\},$$

$$r_1 : A \mapsto Bw|wsC$$

$$r_2 : B \mapsto Cr$$

$$r_3 : C \mapsto e|s$$

$$S = A$$

# Übung

## 4.2. Sprache bestimmen: kot

Bestimmen Sie die Sprache  $L(G)$  für  $G = (V, T, R, S)$ :

$$V = \{A, B, C\}$$

$$T = \{k, o, t\}$$

$$R = \{r_1, r_2, r_3, r_4\},$$

$$r_1 : A \mapsto Bt|Co$$

$$r_2 : B \mapsto Ct$$

$$r_3 : C \mapsto k|o$$

$$r_4 : Ctt \mapsto o|ok$$

$$S = A$$

---

Wenn auf der linken Seite einer Regel ein komplexer Ausdruck steht, dann erfolgt die Ersetzung für den Ausdruck als Ganzes.

D. h. Sei das aktuelle Wort  $w = Ctt$ , dann wird  $w \xrightarrow{r_4} o|ok$ .

# Übung

## 4.3. Ableitung finden: ewtiewet

Wie wird das Wort *ewtiewet* aus der Grammatik  $G = (V, T, R, S)$  abgeleitet?

$$V = \{P, Q, R, S\}$$

$$T = \{e, i, t, w\}$$

$$R = \{r_1, r_2, r_3, r_4, r_5\},$$

$$r_1 : P \mapsto i|wQ$$

$$r_2 : Q \mapsto et|we|wit$$

$$r_3 : R \mapsto Qwt|tieP$$

$$r_4 : S \mapsto Pe|ewR|i|wQwe$$

$$r_5 : wtieP \mapsto wtietie$$

$$S = S$$



# Übung

## 4.4. Ableitung finden: etrrtse

Wie wird das Wort *etrrtse* aus der Grammatik  $G = (V, T, R, S)$  abgeleitet?

$$V = \{X, Y, Z\}$$

$$T = \{e, r, s, t\}$$

$$R = \{r_1, r_2, r_3\},$$

$$r_1 : X \mapsto rts$$

$$r_2 : Y \mapsto etZ|reX$$

$$r_3 : Z \mapsto rXe|srt|tse$$

$$S = Y$$

# Grammatiken für die vorhergehenden Beispiele

$$M_3 = \{0\} \cup \{1\} \times \{0, 1\}^* = \{0, 1, 10, 11, 100, 101, 110, 111, \dots\} = L(G):$$

$$\begin{aligned} G &= (V, T, R, S) \\ V &= \{\text{Start}, A\} \\ T &= \{0, 1\} \\ R &= \{r_1, r_2\}, \\ &\quad r_1 : \text{Start} \mapsto 0|1|1A \\ &\quad r_2 : A \mapsto 0|1|0A|1A \\ S &= \text{Start} \end{aligned}$$

$$M_2 = \{0^n 1^n | n \in \mathbb{N}\} = \{01, 0011, 000111, \dots\} = L(G):$$

$$\begin{aligned} G &= (V, T, R, S) \\ V &= \{S\} \\ T &= \{0, 1\} \\ R &= \{r_1\}, \\ &\quad r_1 : S \mapsto 0S1|01 \\ S &= S \end{aligned}$$

$$M_1 = \{0^n 1^n 2^n | n \in \mathbb{N}\} = \{012, 001122, 000111222, \dots\} = L(G):$$

$$\begin{aligned} G &= (V, T, R, S) \\ V &= \{S, B, C\} \\ T &= \{0, 1, 2\} \\ R &= \{r_1, r_2, r_3, r_4, r_5, r_6\} \quad , \quad \begin{aligned} r_1 : S &\mapsto 0SBC|0BC \\ r_2 : CB &\mapsto BC \\ r_3 : 0B &\mapsto 01 \\ r_3 : 1B &\mapsto 11 \\ r_3 : 1C &\mapsto 12 \\ r_3 : 2C &\mapsto 22 \end{aligned} \\ S &= S \end{aligned}$$

## 5. Chomsky-Hierarchie

# Aufbau der Chomsky-Hierarchie

## Definition

Unterteilung der formalen Grammatiken  $G = (V, T, R, S)$  in vier Klassen:

**Typ-0:** In einer allgemeinen Chomsky-Grammatik oder Typ-0 Grammatik sind alle Regeln zugelassen.

$$rk : (V \cup T)^* \setminus T^* \mapsto (V \cup T)^*$$

**Typ-1:** In einer **kontextsensitiven Grammatik** oder Typ-1 Grammatik müssen die Regeln Prefix und Postfix vor und nach der Ersetzung erhalten, und die Länge des Wortes erhalten oder wachsen lassen, also

$$rk : uAv \mapsto uww \text{ mit } u, v \in (V \cup T)^*, A \in V \text{ und } w \in (V \cup T)^+.$$

Einmalig ist die Regel  $S \mapsto \varepsilon$  erlaubt, dann darf aber  $S$  auf keiner rechten Seite einer anderen Regel auftreten.

**Typ-2:** In einer **kontextfreien Grammatik** oder Typ-2 Grammatik dürfen Regeln links nur aus einer Variablen bestehen, also

$$rk : A \mapsto w \text{ mit } A \in V \text{ und } w \in (V \cup T)^+.$$

Einmalig ist die Regel  $S \mapsto \varepsilon$  erlaubt, dann darf aber  $S$  auf keiner rechten Seite einer anderen Regel auftreten.

**Typ-3:** In einer **regulären Grammatik** oder Typ-3 Grammatik dürfen Regeln links nur aus einer Variablen bestehen, und auf der rechten Seite aus einem terminalen Symbol und optional einer Variable, die bei allen Regeln nur links für *links-lineare Grammatiken* oder nur rechts für *rechts-lineare Grammatiken* stehen darf:

$$rk : A \mapsto aB \text{ (rechts-linear) oder } A \mapsto Ba \text{ (links-linear) oder } A \mapsto a \text{ mit } A, B \in V, a \in T.$$

Einmalig ist die Regel  $S \mapsto \varepsilon$  erlaubt, dann darf aber  $S$  auf keiner rechten Seite einer anderen Regel auftreten.

# Chomsky-Typ einer Sprache

## Beobachtung

Regeln von Grammatiken mit höherem Typ erfüllen immer auch „tiefere“ Bedingungen.

Eine relevante Frage ist: Welches ist der höchste Grammatik-Typ einer erzeugten Sprache?

## Definition

Eine formale Sprache  $L$  ist von einem bestimmten *Chomsky-Typ* und entsprechend kontextsensitiv, kontextfrei oder regulär, wenn es eine Grammatik  $G$  gibt, die die Sprache  $L = L(G)$  erzeugt.

## Zusammenfassung

Da Sprachen höheren Typs auch die Kriterien tieferen Typs erfüllen, sind somit reguläre Sprachen auch kontextfrei, sowie kontextfreie Sprachen auch kontextsensitiv.

# Einordnung von Grammatiken in die Chomsky-Hierarchie

## ? Frage

Welchen Typ hat die folgende Grammatik  $G = (V, T, R, S)$ ?

$$\begin{aligned} V &= \{\text{Start}, A\} \\ T &= \{0, 1\} \\ R &= \{r_1, r_2\}, \\ &\quad r_1 : \text{Start} \mapsto 0|1|1A \\ &\quad r_2 : A \mapsto 0|1|0A|1A \\ S &= \text{Start} \end{aligned}$$

## ? Frage

Welchen Typ hat die folgende Grammatik  $G = (V, T, R, S)$ ?

$$\begin{aligned} V &= \{S\} \\ T &= \{0, 1\} \\ R &= \{r_1\}, \\ &\quad r_1 : S \mapsto 0S1|01 \\ S &= S \end{aligned}$$

## ? Frage

Welchen Typ hat die folgende Grammatik  $G = (V, T, R, S)$ ?

$$\begin{aligned} V &= \{S, B, C\} \quad , \quad S = S \quad , \quad T = \{0, 1, 2\} \\ R &= \{r_1, r_2, r_3, r_4, r_5, r_6\} \quad , \quad r_1 : S \mapsto 0SBC|0BC \\ &\quad r_2 : CB \mapsto BC \\ &\quad r_3 : 0B \mapsto 01 \\ &\quad r_3 : 1B \mapsto 11 \\ &\quad r_3 : 1C \mapsto 12 \\ &\quad r_3 : 2C \mapsto 22 \end{aligned}$$

Können wir die Grammatik umformulieren, damit dies eine Type 1 Grammatik wird?

Umformulierung einer allgemeinen Regel zur Vertauschung von zwei Variablen in kontextsensitive Regeln (der Kontext ist hierbei nicht explizit definiert kann aber natürlich ergänzt werden):

Gegeben sei die Regel  $r_2 : CB \mapsto BC$ .

Umformulierung in kontextsensitive Regeln:

$$\begin{aligned} r_{2'.1} &: CB \mapsto CX \\ r_{2'.2} &: CX \mapsto YX \\ r_{2'.3} &: YX \mapsto YC \\ r_{2'.4} &: YC \mapsto BC \end{aligned}$$

In jeder Regel wird nur eine Variable ersetzt!

## ? Frage

Welchen Typ hat die folgende Grammatik  $G = (V, T, R, S)$ ?

$$\begin{aligned} V &= \{Start, o, >, <, \#, *\} \quad , \quad T = \{0, 1, 2\} \quad , \quad S = Start \\ r_1 : \quad Start &\mapsto \# < o \# \\ r_2 : \quad \# < &\mapsto \# > \mid * \\ r_3 : \quad > o &\mapsto oo > \\ R = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7\} \quad , \quad r_4 : \quad > \# &\mapsto < \# \\ r_5 : \quad o < &\mapsto < o \\ r_6 : \quad *o &\mapsto 0 * \mid 1 * \mid 2 * \\ r_7 : \quad * \# &\mapsto \varepsilon \end{aligned}$$

---

Die Grammatik erzeugt die Sprache:

$$\begin{aligned} M_0 &= \{w \in \Sigma^* \mid |w| = 2^n, n \in \mathbb{N}\} \\ &= \{0, 1, 2, 00, 01, \dots, 21, 22, 0000, 0001, \dots\} \\ &= L(G) \end{aligned}$$

# Übung

## 5.1. Chomsky-Typ: ikos

Bestimmen Sie den Chomsky-Typ der Grammatik  $G = (V, T, R, S)$  und geben Sie eine Ableitung für das Wort *okoik* an.

$$\begin{aligned} V &= \{X, Y, Z\} \\ T &= \{i, k, o, s\} \\ R &= \{r_1, r_2, r_3, r_4, r_5\} \quad , \quad \begin{aligned} r_1 &: X \mapsto io|isk|ok \\ r_2 &: Xo \mapsto ikso|ko|okio|oso \\ r_3 &: Y \mapsto Xoik|k|o|s \\ r_4 &: Z \mapsto oY \\ r_5 &: oXo \mapsto oko|osioo \end{aligned} \\ S &= Z \end{aligned}$$



# Übung

## 5.2. Chomsky-Typ: ru

Bestimmen Sie den Chomsky-Typ von  $G = (V, T, R, S)$  und die Sprache  $L(G)$ :

$$V = \{A, B, C\}$$

$$T = \{r, u\}$$

$$R = \{r_1, r_2, r_3, r_4, r_5\}$$

$$r_1 : A \mapsto uB$$

$$r_2 : B \mapsto r$$

$$r_3 : Bir \mapsto ru|u|ur$$

$$r_4 : C \mapsto AiB|r|rB|u$$

$$r_5 : riB \mapsto u$$

$$S = C$$

# Übung

## 5.3. Chomsky-Typ: iosu

Bestimmen Sie den Chomsky-Typ von  $G = (V, T, R, S)$  und die Sprache  $L(G)$ :

$$V = \{A, B, C, D\}$$

$$T = \{i, o, s, u\}$$

$$R = \{r_1, r_2, r_3, r_4\}$$

$$r_1 : A \mapsto Co|o$$

$$r_2 : B \mapsto iCu|iDu|uA$$

$$r_3 : C \mapsto is$$

$$r_4 : D \mapsto us oA$$

$$S = B$$

## 6. Typ-0 und Typ-1 Grammatiken

# Allgemeine Chomsky Typ-0 Grammatiken

Zur Erinnerung: Entscheidbare Sprachen sind aufzählbar.

## Satz

Die Sprache einer allgemeinen, also Typ-0, Grammatik ist (rekursiv) aufzählbar.

## Beweis

Sei  $r$  die Anzahl Regeln,  $m$  die maximale Verlängerung durch die Anwendung einer Regel und  $k$  die Anzahl Ableitungen.

Die  $k$ -te Anwendung einer Regel  $\varrho \leq r$  an Stelle  $\mu \leq 1 + (k - 1) \cdot m$  wird kodiert als:

$$\nu_k = \varrho + \mu \cdot (r + 1)$$

Durch die Konstruktion von  $\nu_k$  wird sichergestellt, dass jede Ableitung eindeutig kodiert ist. Aus  $\nu_k$  lässt sich die angewandte Regel und die Stelle der Anwendung durch einfache Division durch  $r + 1$  ablesen. Der ganzzahlige Anteil ist die Position und der Rest die angewandte Regel.

- $\varrho$  ist der griechische Buchstabe Rho,
- $\mu$  ist der griechische Buchstabe My,
- $\nu$  ist der griechische Buchstabe Ny.

Die Gödelnummer eines Wortes nach  $s$  Ableitungen ist mit  $(p_k)$  Primzahlfolge:

$$n = \prod_{k=1}^s p_k^{\nu_k}$$

■

## Beispiel

Gegeben sei  $G = (V, T, R, S)$ :

$$\begin{aligned} V &= \{T\} \\ T &= \{0, 1, +\} \\ R &= \{r_1, r_2, r_3\} \\ r_1 : T &\mapsto T + T \\ r_2 : T &\mapsto 0 \\ r_3 : T &\mapsto 1 \\ S &= T \end{aligned}$$

Ableitung von  $0 + 0$ :

$$1. \quad T \xRightarrow{r_1, \mu=1} T + T \xRightarrow{r_2, \mu=1} 0 + T \xRightarrow{r_2, \mu=3} 0 + 0$$

$$n = 2^{\nu_1} \cdot 3^{\nu_2} \cdot 5^{\nu_3}$$

$$\nu_1 = 1 + 1 \cdot 4 = 5$$

$$\nu_2 = 2 + 1 \cdot 4 = 6$$

$$\nu_3 = 2 + 3 \cdot 4 = 14 \quad \left(\frac{14}{4} = 3 \text{ Rest } 2\right)$$

$$2. \quad T \xrightarrow{r_1, \mu=1} T + T \xrightarrow{r_2, \mu=3} T + 0 \xrightarrow{r_2, \mu=1} 0 + 0$$

# Chomsky Typ-0 Grammatiken - Schlussfolgerungen und Beobachtungen

- Ist eine formale Sprache rekursiv aufzählbar, so wird sich daraus auch eine Typ-0 Grammatik erzeugen lassen.
- (Aber) nicht jede Typ-0 Grammatik ist entscheidbar (d.h. rekursiv)!
  - Für eine Typ-0 Sprache des Halteproblems ist nur das positive entscheidbar.
  - Eine Endlosschleife endet - per Definition - nie...
- Es muss auch sehr viele formale Sprachen geben, die nicht Typ-0 sind:
  - Typ-0 Sprachen sind durch Turingmaschinen erzeugbar, also aufzählbar.
  - Die Menge der formalen Sprachen ist überabzählbar...

# Chomsky Typ-1 - kontextsensitive Grammatiken

## Satz

Die Sprache einer kontextsensitiven, also Typ-1, Grammatik ist entscheidbar.

## Beweis

Erzeugte Wörter aus Produktionen sind in der Länge monoton wachsend!

Sei  $G = (V, T, R, S)$  und  $w \in T^*$  mit  $n = |w|$  und  $M$  Produkte, die auf Worte der Länge  $n$  abgebildet werden können:

$M = \{(V \cup T)^m \mid 0 < m \leq n\}$  ist durch  $|M| = \sum_{m=1}^n (|V| + |T|)^m$  beschränkt! Nach spätestens  $|M|$  Ableitungen sind alle möglichen Quellen, bzw. maximal  $n \cdot |M|$  Stellen für Ableitungen, durchsucht.

Damit ist bei einer Suche unter allen Worten bis Länge  $n$  nach endlicher Suche durch und kann  $w \in L(G)$  oder  $w \notin L(G)$  entschieden werden. ■

## Achtung!

Die Umkehrung gilt nicht: Nicht jede entscheidbare Sprache ist kontextsensitiv! Es kann eine entscheidbare Typ-0 Sprache konstruiert werden.

Sind entscheidbare Sprache damit eine gute Wahl für Programmiersprachen?

- Entscheidbarkeit sagt nichts über die Komplexität der Entscheidung aus.
- Der Aufwand zur Analyse von Typ-1 Sprachen ist bereits sehr hoch.
- Trotzdem haben viele Programmiersprachen Anteile, die kontextsensitiv sind:
  - der wesentliche Teil (insbesondere die Syntaxanalyse) ist jedoch kontextfrei
  - Sonderfälle (zum Beispiel Typprüfungen) werden gesondert verarbeitet

# Die Sprachhiarchie und die Chomsky-Typen

## Satz

Seien

- $L$  die Menge der formalen Sprachen,
- $L_k$  die Menge der Sprachen vom Chomsky-Typ  $k$ ,
- $L_{\text{aufzählbar}}$  die Menge der aufzählbaren formalen Sprachen und
- $L_{\text{entscheidbar}}$  die Menge der entscheidbaren formalen Sprachen,

dann gilt:

$$\underbrace{L_3}_{\text{regulär}} \subset \underbrace{L_2}_{\text{kontextfrei}} \subset \underbrace{L_1}_{\text{kontextsensitiv}} \subset \underbrace{L_{\text{entscheidbar}}}_{\text{rekursiv}} \subset \underbrace{L_0 = L_{\text{aufzählbar}}}_{\text{allg. Chomsky-Grammatik}} \subset \underbrace{L}_{\text{formale Sprache}}$$

# Übung

## 6.1. Aufzählung einer Sprache

Gegeben sei

$G = (V, T, R, S)$ :

$V = \{T\}$

$T = \{0, 1, \cdot, (, )\}$

$R = \{r_1, r_2, r_3\}$

$r_1 : T \mapsto 1$

$r_2 : T \mapsto (T)$

$r_3 : T \mapsto T \cdot T$

$S = T$

1. Gegeben Sei folgende Ableitung:

$T \mapsto T \cdot T \mapsto (T) \cdot T \mapsto (1) \cdot T$

Bestimmen Sie die Gödelnummer.

2. Bestimmen Sie die Ableitung/das Wort für die Gödelnummer

$n = 37\,968\,750\,000\,000$ .



## 7. Grammatiken kontextfreier Sprachen

# Chomsky Typ-2: Kontextfreie Grammatiken

Grammatiken für die wichtige Klasse der kontextfreien Sprachen sind nicht eindeutig:

Zwei Grammatiken für Terme wie  $1 + 2 * 3 \in L(G1) = L(G2)$ :

$$\begin{aligned} G_1 &= (V_1, T_1, R_1, S_1) \\ V_1 &= \{Term\} \\ T_1 &= \{0, 1, \dots, 9, +, *\} \\ R_1 &= \{r_1, r_2, r_3\} \\ r_1 : Term &\mapsto Term + Term \\ r_2 : Term &\mapsto Term * Term \\ r_3 : Term &\mapsto 0|1|\dots|9 \\ S_1 &= T \end{aligned}$$

$$\begin{array}{c} \underbrace{1}_{Term} + \underbrace{2}_{Term} * \underbrace{3}_{Term} \\ \underbrace{\hspace{1.5cm}}_{Term} \end{array}$$

$$\begin{aligned} G_2 &= (V_2, T_2, R_2, S_2) \\ V_2 &= \{Sum, Prod\} \\ T_2 &= \{0, 1, \dots, 9, +, *\} \\ R_2 &= \{r_1, r_2, r_3, r_4\} \\ r_1 : Sum &\mapsto Sum + Prod \\ r_2 : Sum &\mapsto Prod \\ r_3 : Prod &\mapsto Prod * Prod \\ r_4 : Prod &\mapsto 0|1|\dots|9 \\ S_2 &= Sum \end{aligned}$$

$$\begin{array}{c} \underbrace{1}_{Prod} + \underbrace{2}_{Prod} * \underbrace{3}_{Prod} \\ \underbrace{\hspace{1.5cm}}_{Sum} \end{array}$$

## ? Frage

In welcher Weise unterscheiden sich die beiden Grammatiken?

Bedenken Sie insbesondere die Rechenregeln für die Auswertung von Termen.

# Formate zur Beschreibung kontextfreier Grammatiken

## (E)BNF (Klassisch)

(a) Backus-Naur-Form (BNF) bzw. (b) erweiterte Backus-Naur-Form (EBNF), die die BNF um praktische Elemente erweitert.

### Beispiel

$G_2$  in (E)BNF:

```
<sum> ::= <sum> "+" <prod> | <prod>
<prod> ::= <prod> "*" <prod> | "0" | "1" | ... | "8" | "9"
```

## PEG (Modern)

### Parsing Expression Grammar (PEG)

### Beispiel

$G_2$  in PEG:

```
start: sum
sum:   sum "+" prod | prod
prod:  prod "*" prod | "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

Eine PEG definiert eine Reihenfolge zur Auflösung des Syntaxbaums. D. h. ein Umstellen der Regeln führt zu einer anderen Sprache. D. h. würde die Regel `sum: sum "+" prod | prod` in `sum: prod | sum "+" prod` geändert, würde sich die Sprache ändern bzw. manche Ausdrücke nicht mehr erkannt werden.

# Domänenspezifische Sprachen

- Nur selten werden erfolgreich neue allgemeine Programmiersprachen entwickelt.
- Häufig(er) werden Domänenspezifische Sprachen (DSLs) entwickelt:
  - DSLs sind oft kontextfrei oder regulär.
  - DSLs befähigen Personen mit Domänenwissen, Programme in Ihrer Sprache zu entwickeln.
  - DSLs sind oft einfacher zu verstehen und zu verwenden als allgemeine Programmiersprachen.
  - DSLs haben oft große Einschränkungen sind dafür aber verständlicher
  - DSLs können oft einfacher optimiert werden, da sie weniger allgemein sind

Wir unterscheiden externe und interne DSLs.

## Externer DSLs:

Externe DSLs sind eigenständige Sprachen unabhängig von anderen Sprachen.

- Zahlreiche Beispiele: SQL, Reguläre Ausdrücke, CSS, ...
- Volle Kontrolle über Grammatik und Mächtigkeit
- (Sehr viel) mehr Entwicklungsaufwand

## Interne DSLs:

Interne DSLs sind in einer anderen Sprache eingebettet und nutzen deren Syntax.

- Prominents Beispiel: JSON
- Es gibt Programmiersprachen, die gut (z. B. Scala) und solche die schlecht (z. B. Java) für die Entwicklung von internen DSLs geeignet sind.

# Entwicklung von DSLs mit LARK

Lark ist ein Python-Parser-Generator für kontextfreie Grammatiken.

- LARK basiert auf EBNF
- LARK unterstützt das Erstellen von Parse-Trees basierend auf der Grammatik.

## Beispiel

### "+" Ausdrücke

```
1 from lark import Lark
2
3 GRAMMAR = """
4     s: term
5     term: term "+" term → add
6         | NUMBER → no
7
8     %import common.NUMBER
9     """
10
11 l = Lark(GRAMMAR, start="s")
12 print(l.parse("1+2"))
```

### Resultierender Parse Tree

```
Tree(
  Token("RULE", "start"),
  [
    Tree(
      "add",
      [
        Tree("no", [Token("NUMBER", "1")]),
        Tree("no", [Token("NUMBER", "2")]),
      ],
    ),
  ],
)
```

## 8. Grammatiken regulärer Sprachen

# Chomsky Typ-3: Reguläre Grammatiken

Ein regulärer Ausdruck ist eine effiziente Darstellung von Sprachen.

## Definition

Die Menge  $\mathcal{R}(\Sigma)$  umfasst alle regulären Ausdrücke über einem Alphabet  $\Sigma$ . Es sind  $\emptyset, \varepsilon \in \mathcal{R}(\Sigma)$  und  $\Sigma \subset \mathcal{R}(\Sigma)$ . Sind  $\alpha, \beta \in \mathcal{R}(\Sigma)$ , so sind auch  $\alpha\beta, (\alpha|\beta), (\alpha)^* \in \mathcal{R}(\Sigma)$ .

Die Sprache  $L(\alpha)$  eines regulären Ausdrucks  $\alpha \in \mathcal{R}(\Sigma)$  sei definiert durch  $L(\emptyset) = \emptyset, L(\varepsilon) = \varepsilon$  und  $L(a) = a$  für ein  $a \in \Sigma$ , sowie

- $L(\alpha\beta) = L(\alpha)L(\beta),$
- $L(\alpha|\beta) = L(\alpha) \cup L(\beta),$
- $L(\alpha^*) = (L(\alpha))^*$

für reguläre Ausdrücke  $\alpha, \beta \in \mathcal{R}(\Sigma)$ .

## Beispiel

Auf dem Alphabet  $\Sigma = \{a, b\}$  ist  $\alpha = a(ab)^*(a|bb) \in \mathcal{R}(\Sigma)$  ein regulärer Ausdruck, der diese Sprache beschreibt:

$$L(\alpha) = \{aa, abb, aaba, aabbb, aababa, aababbb, aabababa, aabababbb, \dots\}$$

## Satz

Die Menge der Sprachen regulärer Ausdrücke beschreibt genau die Menge der regulären Sprachen auf einem Alphabet.

## Beispiel

Gegeben sei:

$$M_3 = \{0\} \cup \{1\} \times \{0, 1\}^* = \{0, 1, 10, 11, 100, 101, 110, 111, 1000, \dots\}$$

mit Grammatik:

$$\begin{aligned} G &= (\{\text{Start}, A\}, \{0, 1\}, R, \text{Start}) \\ R &= \{r_1, r_2\}, \\ r_1 &: \text{Start} \mapsto 0|1|1A \\ r_2 &: A \mapsto 0|1|0A|1A \end{aligned}$$

Der regulären Ausdruck  $\alpha = 0|1(0|1)^*$  beschreibt die Sprache.

Die Darstellung ist aber nicht eindeutig:  $\beta = 0|1|1(0|1)^*$  ist eine äquivalenter Ausdruck.

## Satz

Alle endlichen formalen Sprachen sind reguläre Sprachen.

## Beweis

Eine endliche formale Sprache  $L$  besteht aus endlich vielen Worten  $L = \{w_1, \dots, w_n\}$ . Diese Sprache wird auch durch den regulären Ausdruck  $w_1| \dots |w_n$  erzeugt und daher ist die Sprache regulär. ■

# Grammatiken regulärer Sprachen

## Abfolgen und Alternativen

XY	X nach Y
X Y	X oder Y
(X)	Klammerung und Abfrage

## Zeichen

x	Das Zeichen x
\.	Der Punkt .
\t	Tabulator
\\	Backslash \

## Positionen


^	Anfang der Zeile
\$	Ende der Zeile
\b	Wortgrenze
\B	Nicht-Wortgrenze

## Mengen

.	Beliebiges Zeichen
[abc]	Ein Zeichen aus Liste
[^abc]	Ein Zeichen außer aus Liste
[a-r]	Ein Zeichen aus Bereich
\d	Eine Ziffer [0–9]
\D	Keine Ziffer [^0–9]
\w	Wortzeichen [a-zA-Z0–9_]

## Anzahl

X?	einmal oder kein mal
X*	kein mal oder beliebig oft
X+	einmal oder beliebig oft
X{n}	exakt n mal
X{n,}	mindestens n mal
X{n,m}	von n bis m mal
X*?	mit ? nicht gierig

-  *Regular Expressions* werden in fast allen Programmiersprachen und IDEs zur Textanalyse und Transformation verwendet.
- Es gibt leicht unterschiedliche Syntaxvarianten.
- Fast alle Implementierungen bieten Erweiterungen, die über die klassischen regulären Sprachen hinausgehen. (z. B. Lookahead, Lookbehind, Charakterklassen...)



# Übung

## 8.1. Bestimmung eines regulären Ausdrucks

Bestimmen Sie einen möglichst kurzen regulären Ausdruck  $\alpha \in \mathcal{R}(T)$ , für den  $L(\alpha) = L(G)$  für die Grammatik  $G = (V, T, R, S)$  gilt:

$$V = \{A, B, C\}$$

$$T = \{o, r, s\}$$

$$R = \{r_1, r_2, r_3\}$$

$$r_1 : A \rightarrow rC|sB$$

$$r_2 : B \rightarrow rC|oB$$

$$r_3 : C \rightarrow o|s$$

$$S = A$$

# Übung

## 8.2. Rechts-lineare Grammatiken

Bestimmen Sie eine rechts-lineare Typ-3 Grammatik  $G = (V, T, R, S)$  für das Alphabet  $T = (e, r, s, t)$ , für die  $L(G) = L(\alpha)$  mit  $\alpha = r((s|t)^*|e)^*$  gilt.

# Übung

## 8.3. Links-lineare Grammatiken

Bestimmen Sie eine links-lineare Typ-3 Grammatik  $G = (V, T, R, S)$  für das Alphabet  $T = (a, b, k, n)$ , für die  $L(G) = L(\alpha)$  mit  $\alpha = b^*an(k|a)^*$  gilt.

# Übung

## 8.4. Regulären Ausdruck Vereinfachen

Vereinfachen Sie den regulären Ausdruck  $\alpha = (a^*b^*a^*|aba)^*(a|ac^*|aba)$  zu einem äquivalenten kürzeren Ausdruck  $\beta$  mit  $L(\alpha) = L(\beta)$ .

# Übung

## 8.5. Konvertierung einer einfachen Markup Sprache mittels RegExps

Konvertieren Sie den verlinkten Text ([https://delors.github.io/theo-algo-formale\\_sprachen/code/SomeText.txt](https://delors.github.io/theo-algo-formale_sprachen/code/SomeText.txt)) mittels mehrere regulärer Ausdrücke in HTML. Das HTML soll dann dem verlinkten Ergebnis entsprechen ([https://delors.github.io/theo-algo-formale\\_sprachen/code/SomeText.html](https://delors.github.io/theo-algo-formale_sprachen/code/SomeText.html)).

Beachten Sie, dass ggf. die Reihenfolge in der Sie die regulären Ausdrücke auswerten relevant sein kann.

Nutzen Sie ein Diff Tool Ihrer Wahl (zum Beispiel VS Code oder einfach `diff`), um zu überprüfen ob Ihr Ergebnis den Erwartungen entspricht.

Nutzen Sie entweder `sed` zur Auswertung Ihrer regulären Ausdrücke oder VS Code.

---

### SED 101

- `sed` ist ein Stream-Editor, der einzelne Textzeilen bearbeiten kann.
- Um reguläre Ausdrücke zu verwenden, muss `sed` mit dem Flag `-E` gestartet werden. Z. B.: `sed -E -f SomeTextToHTML.sed SomeText.txt > SomeText.html`.
- `\s` steht für alle "whitespace characters" (funktioniert aber ggf. nur unter bestimmten Versionen; z. B. nicht auf dem Mac); `[ ]` oder `[:space:]` sind eine Alternative.
- Ein Ausdruck in `sed` hat die Form:  
`s/regexp/replacement/flags`
  - Dabei wird der durch den regexp erkannte Text durch replacement ersetzt. Flags sind optional. Das `g` Flag (global) ermöglicht es alle Übereinstimmungen zu ersetzen und nicht nur die Erste.
  - Ein `&` im replacement bezieht sich auf den gesamten gefundenen Text.
  - Ein `&` im replacement kann durch ein Backslash escaped werden: `\&`.
  - `\1`, `\2`, `\3` bezieht sich auf die gefundenen Gruppen (in Klammern) im regexp.
- Ein `sed` Script ist eine Liste von `sed` Befehlen, die in einer Datei gespeichert werden und dann Zeile für Zeile auf den Input angewendet werden.
- `sed` ist immer greedy und versucht längst-mögliche Übereinstimmungen zu finden. Ggf. ist es notwendig eine Formulierung zu finden, die verhindert, dass zu viel Text erfasst wird.

#### Hinweis

Es kann notwendig sein Hilfsttransformationen durchzuführen, um die eigentlich gewünschte Transformation zu erreichen.

#### Beispiele

```
1 $ echo "START aa B aa C aa ENDE START aa D aa" | sed -E 's/aa[^E]*aa/MATCH/'
2 START MATCH ENDE START aa D aa
3
4 $ echo "START aa B aa C aa ENDE START aa D aa" | sed -E 's/aa[^E]*aa/MATCH/g'
5 START MATCH ENDE START MATCH
```

# Übung

## 8.6. DSL entwerfen mit LARK

Sie wollen eine DSL für Ihre eigene Markupsprache entwickeln. Ihr Ziel ist es Texte folgender Art zu parsen, um diese danach weiterzuverarbeiten.

+ Wer bin ich?

Ich bin **\*Prof.\*** an der DHBW [link: [www.dhbw-mannheim.de](http://www.dhbw-mannheim.de)].

Meine Homepage finden sie hier: [link: [www.michael-eichberg.de](http://www.michael-eichberg.de)].

Ein „+“ am Anfang einer Zeile kennzeichnet eine Überschrift. Text in „\*“ soll fett dargestellt werden. URLs stehen in Blöcken, die mit „[link:“ anfangen und mit „]“ enden.

Definieren Sie eine Grammatik in LARK. Wenn Sie reguläre Ausdrücke verwenden wollen – zum Beispiel zum Parsen von URLs – können Sie dies in der Grammatik direkt angeben (siehe `WORD` Regel). Der angehängte Code dient als Grundlage.

```
1 from lark import Lark
2 GRAMMAR = """
3     S: ...
4
5     WORD: /[a-zA-Z]+/
6     %import common.WS_INLINE
7     %ignore WS_INLINE
8 """
9
10 l = Lark(GRAMMAR, start="S")
11 print(l.parse("""+ Wer bin ich?"""))
```