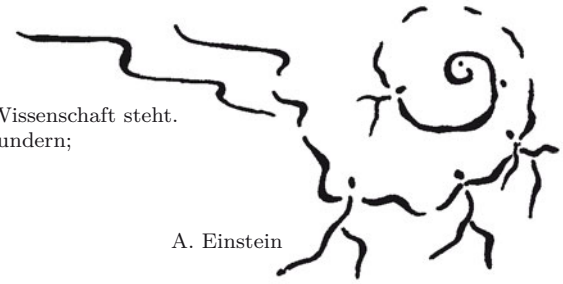
Das Tiefste, das wir erfahren können,
sind die Offenbarungen der Mystik.
Sie sind das fundamentalste Gefühl,
das an der Wiege aller wahren Kunst und Wissenschaft steht.
Wer es nicht kennt, kann sich nicht mehr wundern;
er erlebt das tiefe Staunen nicht mehr:
Er ist so gut wie tot ...
Wie eine erloschene Kerze ...



A. Einstein

2 Alphabete, Wörter, Sprachen und die Darstellung von Problemen

2.1 Zielsetzung

Die Rechner arbeiten im Prinzip mit Texten, die nichts anderes sind als Folgen von Symbolen aus einem bestimmten Alphabet. Die Programme sind Texte über dem Alphabet der Rechnertastatur, alle Informationen sind im Rechner als Folgen von Nullen und Einsen gespeichert, Eingaben und Ausgaben sind im Wesentlichen auch Texte (oder können zumindest als Texte dargestellt werden) über einem geeignet gewählten Alphabet. Aus dieser Sicht realisiert jedes Programm eine Transformation von Eingabetexten in Ausgabetexte.

Das erste Ziel von Kapitel 2 ist, den Formalismus für den Umgang mit Texten als Informationsträger einzuführen. Dieser liefert die notwendige Grundlage, um überhaupt formal die Grundbegriffe der Informatik wie algorithmisches Problem (Aufgabe), Algorithmus (Programm), Rechner, Berechnung, Eingabe, Ausgabe usw. definieren zu können. Die Grundbegriffe, die hier eingeführt werden, sind **Alphabet**, **Wort** und **Sprache**. Ein Teil unserer Aufgabe hier ist es, auch den Umgang mit diesen Begriffen zu üben und so einige grundlegende Operationen auf Texten zu erlernen.

Das zweite Ziel dieses Kapitels ist zu lernen, wie der eingeführte Formalismus zur formalen Darstellung algorithmischer Aufgaben genutzt werden kann. Dabei betrachten wir überwiegend zwei Klassen von Aufgaben, **Entscheidungsprobleme** und **Optimierungsprobleme**.

Das dritte und letzte Ziel dieses Kapitels ist, sich mit der Komprimierbarkeit von Texten zu beschäftigen. Wir führen hier den Begriff der **Kolmogorov-Komplexität** ein. Dank diesem können wir nicht nur über die kürzeste Darstellung von Objekten (Texten) und die Komprimierbarkeit von Darstellungen sprechen, sondern auch den Informationsinhalt von Texten messen und eine sinnvolle Definition des Attributs **zufällig** für Texte geben. Dies ist ein Beitrag der Informatik auf der philosophischen Ebene, weil er sinnvoll erklärt, wann ein Objekt oder eine Erscheinung als zufällig eingeordnet werden kann. Ein anderer

J. Hromkovič, *Theoretische Informatik,*
DOI 10.1007/978-3-658-06433-4_2, © Springer Fachmedien Wiesbaden 2014

wichtiger Punkt ist, dass die Kolmogorov-Komplexität ein wertvolles Instrument zur Untersuchung von Berechnungen ist, was auch an mehreren Stellen in diesem Buch deutlich gemacht wird.

2.2 Alphabete, Wörter und Sprachen

Bei der algorithmischen Datenverarbeitung repräsentieren wir die Daten und betrachteten Objekte durch Folgen von Symbolen. Genau wie bei der Entwicklung natürlicher Sprachen fangen wir mit der Festlegung von Symbolen an, die wir zur Darstellung der Daten verwenden wollen. Im Folgenden bezeichnet $\mathbb{N} = \{0, 1, 2, \ldots\}$ die Menge der natürlichen Zahlen.

Definition 2.1. *Eine endliche nichtleere Menge Σ heißt **Alphabet**. Die Elemente eines Alphabets werden **Buchstaben** (**Zeichen**, **Symbole**) genannt.*

Die Bedeutung ist die gleiche wie bei natürlichen Sprachen: Das Alphabet wird verwendet, um eine schriftliche Darstellung einer Sprache zu erzeugen. Für uns ist nur wichtig zu wissen, dass wir uns beliebige, aber nur endlich viele Symbole aussuchen dürfen, um eine Darstellung untersuchter Objekte zu realisieren.

Wir präsentieren jetzt einige der hier am häufigsten benutzten Alphabete.

- $\Sigma_{\text{bool}} = \{0, 1\}$ ist das Boole'sche Alphabet, mit dem die Rechner arbeiten.
- $\Sigma_{\text{lat}} = \{a, b, c, \ldots, z\}$ ist das lateinische Alphabet.
- $\Sigma_{\text{Tastatur}} = \Sigma_{\text{lat}} \cup \{A, B, \ldots, Z, \sqcup, >, <, (,), \ldots, !\}$ ist das Alphabet aller Symbole der Rechnertastatur, wobei $\sqcup$ das Leersymbol ist.
- $\Sigma_m = \{0, 1, 2, \ldots, m-1\}$ für jedes $m \geq 1$ ist ein Alphabet für die m-adische Darstellung von Zahlen.
- $\Sigma_{\text{logic}} = \{0, 1, x, (,), \wedge, \vee, \neg\}$ ist ein Alphabet, in dem man Boole'sche Formeln gut darstellen kann.

Im Folgenden definieren wir Wörter als Folgen von Buchstaben. Man bemerke, dass der Begriff **Wort** in der Fachsprache der Informatik einem beliebigen Text entspricht und nicht nur der Bedeutung des Begriffs Wort in natürlichen Sprachen.

Definition 2.2. *Sei Σ ein Alphabet. Ein **Wort** über Σ ist eine endliche (eventuell leere) Folge von Buchstaben aus Σ. Das **leere Wort λ** ist die leere Buchstabenfolge. (Manchmal benutzt man ε statt λ.)*

*Die **Länge $|w|$** eines Wortes w ist die Länge des Wortes als Folge, d. h. die Anzahl der Vorkommen von Buchstaben in w.*

***Σ^*** ist die Menge aller Wörter über Σ, **Σ^+**$= \Sigma^* - \{\lambda\}$.*

Die Folge $0, 1, 0, 0, 1, 1$ ist ein Wort über Σ_{bool} und über Σ_{Tastatur}, $|0, 1, 0, 0, 1, 1| = 6$. Das leere Wort λ ist ein Wort über jedem Alphabet, $|\lambda| = 0$.

Verabredung. Wir werden Wörter ohne Komma schreiben, das heißt, statt der Folge $x_1, x_2, \ldots, x_n$ schreiben wir $x_1x_2 \ldots x_n$. Statt $0, 1, 0, 0, 1, 1$ benutzen wir also im Folgenden die Darstellung 010011.

Das Leersymbol $\sqcup$ über Σ_{Tastatur} ist unterschiedlich von λ, es gilt $|\sqcup| = 1$. Somit kann der Inhalt eines Buches oder ein Programm als ein Wort über Σ_{Tastatur} betrachtet werden.

Es gilt

$$\begin{aligned}(\Sigma_{\text{bool}})^* &= \{\lambda, 0, 1, 00, 01, 10, 11, 000, 001, 010, 100, 011, \ldots\} \\ &= \{\lambda\} \cup \{x_1x_2 \ldots x_i \mid i \in \mathbb{N},\ x_j \in \Sigma_{\text{bool}} \text{ für } j = 1, \ldots, i\}.\end{aligned}$$

Wir sehen an diesem Beispiel, dass eine Möglichkeit, alle Wörter über einem Alphabet aufzuzählen, darin besteht, alle Wörter der Länge $i = 0, 1, 2, \ldots$ hintereinander zu schreiben.

Aufgabe 2.1. Bestimmen Sie für jedes $i \in \mathbb{N}$, wie viele Wörter der Länge i über einem Alphabet Σ existieren.

Aufgabe 2.2. Gegeben sei das Alphabet $\Sigma = \{0, 1, \#\}$. Seien k, n positive ganze Zahlen mit $k \leq n$.

(a) Bestimmen Sie die Anzahl der verschiedenen Wörter der Länge n mit genau k Vorkommen des Symbols $\#$.

(b) Bestimmen Sie die Anzahl der verschiedenen Wörter der Länge n mit höchstens k Vorkommen des Symbols $\#$.

Wörter können wir benutzen, um unterschiedliche Objekte wie zum Beispiel Zahlen, Formeln, Graphen und Programme darzustellen. Ein Wort $x = x_1x_2 \ldots x_n \in (\Sigma_{\text{bool}})^*$, $x_i \in \Sigma_{\text{bool}}$ für $i = 1, \ldots, n$, kann als die binäre Darstellung der Zahl

$$\text{Nummer}(x) = \sum_{i=1}^{n} x_i \cdot 2^{n-i}$$

betrachtet werden. Für eine natürliche Zahl $m \in \mathbb{N} - \{0\}$ bezeichnen wir mit $\text{Bin}(m) \in (\Sigma_{\text{bool}})^*$ die kürzeste[1] binäre Darstellung von m, also $\text{Nummer}(\text{Bin}(m)) = m$. Wir definieren $\text{Bin}(0) = 0$.

Aufgabe 2.3. Eine binäre Darstellung jeder positiven Zahl beginnt mit einer 1. Wie lang ist $\text{Bin}(m)$ bei einer gegebenen Zahl m?

Aufgabe 2.4. Sei $x \in \Sigma_m^*$ für ein $m \geq 1$. Betrachten Sie x als m-adische Darstellung einer Zahl $\text{Nummer}_m(x)$. Wie berechnet man $\text{Nummer}_m(x)$?

Eine Zahlenfolge $a_1, a_2, \ldots, a_m$, $m \in \mathbb{N}$, $a_i \in \mathbb{N}$ für $i = 1, \ldots, m$, kann man als

$$\text{Bin}(a_1)\#\text{Bin}(a_2)\# \cdots \#\text{Bin}(a_m) \in \{0, 1, \#\}^*$$

darstellen.

[1] Dies bedeutet lediglich, dass das erste Symbol von $\text{Bin}(m)$ eine 1 ist.

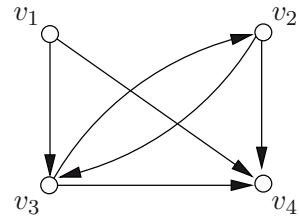


Abbildung 2.1

Sei $G = (V, E)$ ein gerichteter Graph mit der Knotenmenge V und der Kantenmenge $E \subseteq \{(u, v) \mid u, v \in V,\ u \neq v\}$. Sei $n = |V|$ die Kardinalität von V. Wir wissen, dass wir G durch eine Adjazenzmatrix M_G repräsentieren können. $M_G = [a_{ij}]$ hat die Größe $n \times n$ und

$$a_{ij} = 1 \iff (v_i, v_j) \in E.$$

Daher bedeutet $a_{ij} = 1$, dass die Kante (v_i, v_j) in G vorhanden ist, und $a_{ij} = 0$ bedeutet, dass die Kante (v_i, v_j) in G nicht vorhanden ist. Eine Matrix können wir als ein Wort über dem Alphabet $\Sigma = \{0, 1, \#\}$ repräsentieren. Wir schreiben einfach die Zeilen von M_G hintereinander und das Symbol $\#$ benutzen wir, um das Ende einer Zeile zu markieren.

Für den Graphen in Abbildung 2.1 ist die entsprechende Adjazenzmatrix

$$\begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Die vorgeschlagene Kodierung als Wort über $\{0, 1, \#\}$ ist

$$0011\#0011\#0101\#0000\#.$$

Es ist klar, dass diese Darstellung eindeutig ist, was bedeutet, dass man aus der gegebenen Darstellung den Graphen eindeutig bestimmen kann.

Aufgabe 2.5. Die vorgeschlagene Darstellung eines Graphen als Wort über $\{0, 1, \#\}$ hat die Länge $n(n + 1)$ für einen Graphen mit n Knoten. Überlegen Sie sich eine kürzere eindeutige Darstellung von Graphen über dem Alphabet $\{0, 1, \#\}$.

Aufgabe 2.6. Entwerfen Sie eine Darstellung für Graphen über dem Alphabet Σ_{bool}.

Bei algorithmischen Aufgaben sind die Eingaben oft gewichtete Graphen $G = (V, E, h)$, wobei h eine Funktion von E nach $\mathbb{N} - \{0\}$ ist. Informell bedeutet dies, dass jeder Kante $e \in E$ ein Gewicht (manchmal auch Kosten genannt) $h(e)$ zugeordnet ist. Wir wissen, dass auch solche Graphen durch Adjazenzmatrizen darstellbar sind. Auch in diesem Fall

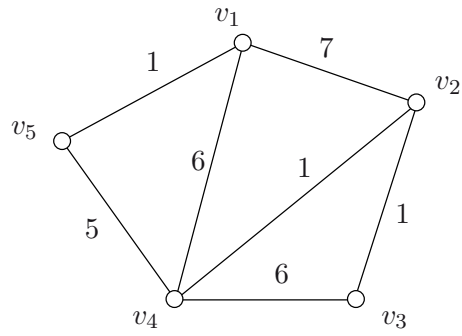


Abbildung 2.2

bedeutet $a_{ij} = 0$, dass die Kante[2] $\{v_i, v_j\}$ nicht vorhanden ist. Falls $\{v_i, v_j\} \in E$, dann ist $a_{ij} = h(\{v_i, v_j\})$ das Gewicht der Kante $\{v_i, v_j\}$.

In diesem Fall können wir die Gewichte $a_{ij} = h(\{v_i, v_j\})$ binär darstellen und durch #-Symbole abtrennen. Um das Ende einer Zeile zu bezeichnen, kann man zwei # hintereinander benutzen. Damit hat der gewichtete Graph in Abbildung 2.2 mit der Adjazenzmatrix

$$\begin{pmatrix} 0 & 7 & 0 & 6 & 1 \\ 7 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 6 & 0 \\ 6 & 1 & 6 & 0 & 5 \\ 1 & 0 & 0 & 5 & 0 \end{pmatrix}$$

die folgende Darstellung über $\{0, 1, \#\}$:

$$0\#111\#0\#110\#1\#\#111\#0\#1\#1\#0\#\#0\#1\#0\#110\#0\#\#110\#$$
$$1\#110\#0\#101\#\#1\#0\#0\#101\#0\#\#.$$

Weil der betrachtete Graph ungerichtet ist, gilt $a_{ij} = a_{ji}$ für alle i, j. In unserer Darstellung bedeutet dies, dass die Information über das Gewicht jeder Kante in dem Wort doppelt vorkommt. Deswegen reicht es aus, nur die Matrixelemente oberhalb der Hauptdiagonalen zu betrachten. Das resultierende Wort über $\{0, 1, \#\}$ ist dann

$$111\#0\#110\#1\#\#1\#1\#0\#\#110\#0\#\#101\#\#.$$

Als letztes Beispiel betrachten wir die Darstellung Boole'scher Formeln, die nur die Boole'schen Operationen Negation ($\neg$), Disjunktion ($\vee$) und Konjunktion ($\wedge$) benutzen. Im Folgenden bezeichnen wir Boole'sche Variablen in Formeln als $x_1, x_2, \ldots$. Die Anzahl der möglichen Variablen ist unendlich und deswegen können wir $x_1, x_2, \ldots$ nicht als Buchstaben unseres Alphabets benutzen. Wir benutzen daher das Alphabet $\Sigma_{\text{logic}} = \{0, 1, x, (,), \wedge, \vee, \neg\}$ und kodieren die Boole'sche Variable x_i als das Wort $x\text{Bin}(i)$ für

[2] Die ungerichtete Kante zwischen u und v bezeichnen wir hier als $\{u, v\}$. Für eine gerichtete Kante von u nach v benutzen wir die übliche Bezeichnung (u, v).

jedes $i \in \mathbb{N}$. Die restlichen Symbole der Formel übernehmen wir eins zu eins. Damit hat die Formel

$$(x_1 \vee x_7) \wedge \neg(x_{12}) \wedge (x_4 \vee x_8 \vee \neg(x_2))$$

die folgende Darstellung

$$(x1 \vee x111) \wedge \neg(x1100) \wedge (x100 \vee x1000 \vee \neg(x10)).$$

Eine nützliche Operation über Wörtern ist die einfache Verkettung zweier Wörter.

Definition 2.3. *Die* ***Verkettung*** (***Konkatenation***) *für ein Alphabet* Σ *ist eine Abbildung* $\mathrm{Kon}\colon \Sigma^* \times \Sigma^* \to \Sigma^*$*, so dass*

$$\mathrm{Kon}(x, y) = x \cdot y = xy$$

für alle $x, y \in \Sigma^*$*.*

Sei $\Sigma = \{0, 1, a, b\}$ und seien $x = 0aa1bb$ und $y = 111b$. Dann ist $\mathrm{Kon}(x, y) = x \cdot y = 0aa1bb111b$.

Bemerkung 2.1. Die Verkettung Kon über Σ ist eine assoziative Operation über Σ^*, weil

$$\mathrm{Kon}(u, \mathrm{Kon}(v, w)) = u \cdot (v \cdot w) = uvw = (u \cdot v) \cdot w = \mathrm{Kon}(\mathrm{Kon}(u, v), w)$$

für alle $u, v, w \in \Sigma^*$. Ferner gilt für jedes $x \in \Sigma^*$:

$$x \cdot \lambda = \lambda \cdot x = x.$$

Also ist (Σ^*, Kon) eine Halbgruppe (Monoid) mit dem neutralen Element λ.

Es ist klar, dass die Konkatenation nur für einelementige Alphabete kommutativ ist.

Bemerkung 2.2. Für alle $x, y \in \Sigma^*$ gilt:

$$|xy| = |x \cdot y| = |x| + |y|.$$

Im Folgenden werden wir die einfache Notation xy an Stelle der Notationen $\mathrm{Kon}(x, y)$ und $x \cdot y$ bevorzugen.

Definition 2.4. *Für ein Wort* $a = a_1 a_2 \dots a_n$*, wobei* $a_i \in \Sigma$ *für* $i \in \{1, 2, \dots, n\}$*, bezeichnet* $a^{\mathsf{R}} = a_n a_{n-1} \dots a_1$ *die* ***Umkehrung*** (***Reversal***) *von* a*.*

Aufgabe 2.7. Sei Σ ein Alphabet und seien $u, v \in \Sigma^*$. Beweisen oder widerlegen Sie:

$$(uv)^{\mathsf{R}} = v^{\mathsf{R}} u^{\mathsf{R}}.$$

Definition 2.5. *Sei* Σ *ein Alphabet. Für alle* $x \in \Sigma^*$ *und alle* $i \in \mathbb{N}$ *definieren wir die* i*-te* ***Iteration*** x^i *von* x *als*

$$x^0 = \lambda,\ x^1 = x \text{ und } x^i = xx^{i-1}.$$

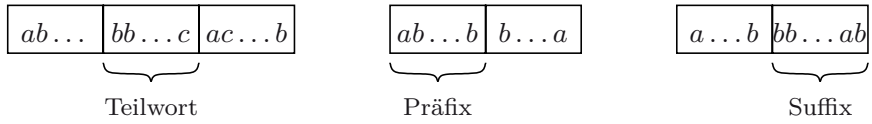


Abbildung 2.3

So ist zum Beispiel $\mathrm{Kon}(aabba, aaaaa) = aabbaaaaaa = a^2b^2a^6 = a^2b^2(aa)^3$. Wir sehen, dass uns die eingeführte Notation eine kürzere Darstellung von Wörtern ermöglicht.

Im Folgenden definieren wir Teilwörter eines Wortes x als zusammenhängende Teile von x (Abbildung 2.3).

Definition 2.6. *Seien $v, w \in \Sigma^*$ für ein Alphabet Σ.*

- *v heißt ein **Teilwort** von $w \iff \exists x, y \in \Sigma^* \colon w = xvy$.*
- *v heißt ein **Präfix** von $w \iff \exists y \in \Sigma^* \colon w = vy$.*
- *v heißt ein **Suffix** von $w \iff \exists x \in \Sigma^* \colon w = xv$.*
- *$v \neq \lambda$ heißt ein **echtes** Teilwort (Präfix, Suffix) von w genau dann, wenn $v \neq w$ und v ein Teilwort (Präfix, Suffix) von w ist.*

Es gilt $(abc)^3 = abcabcabc$, und das Wort abc ist ein echtes Präfix von $(abc)^3$. Das Wort bc ist ein echtes Suffix von $(abc)^3$.

Aufgabe 2.8. Sei Σ ein Alphabet und sei $x \in \Sigma^n$ für ein $n \in \mathbb{N} - \{0\}$. Wie viele unterschiedliche Teilwörter kann x höchstens haben? Zählen Sie alle unterschiedlichen Teilwörter des Wortes $abbcbbab$ auf.

Definition 2.7. *Seien $x \in \Sigma^*$ und $a \in \Sigma$. Dann ist $|x|_a$ definiert als die Anzahl der Vorkommen von a in x.*

Für jede Menge A bezeichnet $\boldsymbol{|A|}$ die Kardinalität von A und $\boldsymbol{\mathcal{P}(A)} = \{S \mid S \subseteq A\}$ die Potenzmenge von A.

Also ist $|abbab|_a = 2$, $|11bb0|_0 = 1$. Für alle $x \in \Sigma^*$ gilt

$$|x| = \sum_{a \in \Sigma} |x|_a.$$

In diesem Buch brauchen wir oft eine feste Ordnung aller Wörter über einem gegebenen Alphabet. Die günstigste Möglichkeit für uns ist, die folgende kanonische Ordnung zu betrachten.

Definition 2.8. *Sei $\Sigma = \{s_1, s_2, \ldots, s_m\}$, $m \geq 1$, ein Alphabet und sei $s_1 < s_2 < \cdots < s_m$ eine Ordnung auf Σ. Wir definieren die **kanonische Ordnung** auf Σ^* für $u, v \in \Sigma^*$ wie folgt:*

$$\begin{aligned} u < v \iff & |u| < |v| \\ & \vee\, |u| = |v| \wedge u = x \cdot s_i \cdot u' \wedge v = x \cdot s_j \cdot v' \\ & \quad \text{für irgendwelche } x, u', v' \in \Sigma^* \text{ und } i < j. \end{aligned}$$

Unter dem Begriff Sprache verstehen wir jede Menge von Wörtern über einem festen Alphabet.

Definition 2.9. *Eine* ***Sprache*** L *über einem Alphabet* Σ *ist eine Teilmenge von* Σ^*. *Das Komplement* $\boldsymbol{L^{\complement}}$ *der Sprache* L *bezüglich* Σ *ist die Sprache* $\Sigma^* - L$.

$\boldsymbol{L_\emptyset} = \emptyset$ *ist die* ***leere Sprache***.

$\boldsymbol{L_\lambda} = \{\lambda\}$ *ist die einelementige Sprache, die nur aus dem leeren Wort besteht.*

Sind L_1 *und* L_2 *Sprachen über* Σ, *so ist*

$$\boldsymbol{L_1 \cdot L_2} = \boldsymbol{L_1 L_2} = \{vw \mid v \in L_1 \text{ und } w \in L_2\}$$

die ***Konkatenation*** *von* L_1 *und* L_2. *Ist* L *eine Sprache über* Σ, *so definieren wir*

$$\boldsymbol{L^0} := L_\lambda \text{ und } \boldsymbol{L^{i+1}} = L^i \cdot L \text{ für alle } i \in \mathbb{N},$$

$$\boldsymbol{L^*} = \bigcup_{i \in \mathbb{N}} L^i \text{ und } \boldsymbol{L^+} = \bigcup_{i \in \mathbb{N} - \{0\}} L^i = L \cdot L^*.$$

L^* *nennt man den* ***Kleene'schen Stern*** *von* L.

Die folgenden Mengen sind Sprachen über dem Alphabet $\Sigma = \{a, b\}$:

- $L_1 = \emptyset$,
- $L_2 = \{\lambda\}$,
- $L_3 = \{\lambda, ab, abab\}$,
- $L_4 = \Sigma^* = \{\lambda, a, b, aa, \ldots\}$,
- $L_5 = \Sigma^+ = \{a, b, aa, \ldots\}$,
- $L_6 = \{a\}^* = \{\lambda, a, aa, aaa, \ldots\} = \{a^i \mid i \in \mathbb{N}\}$,
- $L_7 = \{a^p \mid p \text{ ist eine Primzahl}\}$,
- $L_8 = \{a^i b^{2i} a^i \mid i \in \mathbb{N}\}$,
- $L_9 = \Sigma$,
- $L_{10} = \Sigma^3 = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$.

Die Menge aller grammatisch korrekten Texte im Deutschen ist eine Sprache über Σ_{Tastatur} und die Menge aller syntaktisch korrekten Programme in C++ ist eine Sprache über Σ_{Tastatur}.

Man bemerke, dass $\Sigma^i = \{x \in \Sigma^* \mid |x| = i\}$, und dass $L_\emptyset L = L_\emptyset = \emptyset$, $L_\lambda \cdot L = L$.

Aufgabe 2.9. Sei $L_1 = \{\lambda, ab, b^3a^4\}$ und $L_2 = \{ab, b, ab^2, b^4\}$. Welche Wörter liegen in der Sprache $L_1 L_2$?

Unser nächstes Ziel ist, den Umgang mit Sprachen ein wenig zu üben. Weil Sprachen Mengen sind, haben die üblichen Operationen Vereinigung ($\cup$) und Schnitt ($\cap$) eine klare Bedeutung. Zu diesen Operationen haben wir die Konkatenation und den Kleene'schen Stern hinzugefügt. Die erste Frage, die wir uns stellen, ist, ob Distributivgesetze bezüglich $\cup$ und Konkatenation bzw. bezüglich $\cap$ und Konkatenation gelten. Für die Konkatenation und $\cup$ gibt uns das nächste Lemma 2.1 eine positive Antwort. Um den Beweis der Gleichheit von zwei Mengen (Sprachen) A und B zu führen, benutzen wir die üblichen Methoden der Mengentheorie. Wir zeigen nacheinander $A \subseteq B$ und $B \subseteq A$, was $A = B$ impliziert. Um $A \subseteq B$ zu zeigen, reicht es zu beweisen, dass für jedes Element $x \in A$ gilt, dass $x \in B$.

Lemma 2.1. *Seien L_1, L_2 und L_3 Sprachen über einem Alphabet Σ. Dann gilt*

$$L_1L_2 \cup L_1L_3 = L_1(L_2 \cup L_3).$$

Beweis. Zuerst zeigen wir $L_1L_2 \cup L_1L_3 \subseteq L_1(L_2 \cup L_3)$. Die Bemerkungen in geschweiften Klammern sind die Begründung für den vorangegangenen Schritt.

Es gilt: $L_1L_2 \subseteq L_1(L_2 \cup L_3)$, weil

$$\begin{aligned} L_1L_2 &= \{xy \mid x \in L_1 \wedge y \in L_2\} && \{\text{Definition der Konkatenation}\} \\ &\subseteq \{xy \mid x \in L_1 \wedge y \in L_2 \cup L_3\} && \{\text{weil } L_2 \subseteq L_2 \cup L_3\} \\ &= L_1 \cdot (L_2 \cup L_3). && \{\text{Definition der Konkatenation}\} \end{aligned}$$

Analog zeigt man $L_1L_3 \subseteq L_1(L_2 \cup L_3)$. Daraus folgt $L_1L_2 \cup L_1L_3 \subseteq L_1(L_2 \cup L_3)$.

Jetzt zeigen wir die Inklusion $L_1(L_2 \cup L_3) \subseteq L_1L_2 \cup L_1L_3$. Sei im Folgenden $x \in L_1(L_2 \cup L_3)$. Dann gilt

$$\begin{aligned} & x \in \{yz \mid y \in L_1 \wedge z \in L_2 \cup L_3\} \\ & \{\text{Definition der Konkatenation}\} \\ \Longrightarrow\ & \exists y \in L_1 \wedge \exists z \in L_2 \cup L_3, \text{ so dass } x = yz \\ \Longrightarrow\ & \exists y \in L_1 \wedge (\exists z \in L_2 \vee \exists z \in L_3), \text{ so dass } x = yz \\ & \{\text{Definition von } \cup\} \\ \Longleftrightarrow\ & (\exists y \in L_1 \wedge \exists z \in L_2 \colon x = yz) \vee (\exists y \in L_1 \wedge \exists z \in L_3 \colon x = yz) \\ & \{\text{Distributivgesetz für } \wedge, \vee\} \\ \Longleftrightarrow\ & (x \in \underbrace{\{yz \mid y \in L_1 \wedge z \in L_2\}}_{L_1L_2}) \vee (x \in \underbrace{\{yz \mid y \in L_1 \wedge z \in L_3\}}_{L_1L_3}) \\ & \{\text{Definition der Konkatenation}\} \\ \Longleftrightarrow\ & x \in L_1L_2 \cup L_1L_3. \quad \{\text{Definition von } \cup\} \end{aligned}$$

□

Jetzt wollen wir uns mit der Frage nach der Gültigkeit des Distributivgesetzes für die Konkatenation und den Schnitt beschäftigen. Vielleicht ist es auf den ersten Blick überraschend, dass hier die Antwort im allgemeinen negativ ist. Es gilt nur eine Inklusion, die wir zuerst zeigen.

Lemma 2.2. *Seien L_1, L_2, L_3 Sprachen über einem Alphabet Σ. Dann gilt*

$$L_1(L_2 \cap L_3) \subseteq L_1L_2 \cap L_1L_3.$$

Beweis. Sei $x \in L_1(L_2 \cap L_3)$. Dies ist äquivalent zu

$$\begin{aligned}
& x \in \{yz \mid y \in L_1 \wedge z \in L_2 \cap L_3\} \\
& \quad \{\text{Definition der Konkatenation}\} \\
\iff & \exists y, z \in \Sigma^*, y \in L_1 \wedge (z \in L_2 \wedge z \in L_3), \text{ so dass } x = yz \\
& \quad \{\text{Definition von } \cap\} \\
\iff & \exists y, z \in \Sigma^*, (y \in L_1 \wedge z \in L_2) \wedge (y \in L_1 \wedge z \in L_3) \colon x = yz \\
\implies & \exists y, z \in \Sigma^*, (yz \in L_1L_2) \wedge (yz \in L_1L_3) \colon x = yz \\
& \quad \{\text{Definition der Konkatenation}\} \\
\iff & x \in L_1L_2 \cap L_1L_3. \quad \{\text{Definition von } \cap\}
\end{aligned}$$

□

Um zu zeigen, dass $L_1(L_2 \cap L_3) \supseteq L_1L_2 \cap L_1L_3$ nicht im allgemeinen gilt, reicht es, drei konkrete Sprachen U_1, U_2 und U_3 zu finden, so dass $U_1(U_2 \cap U_3) \subsetneq U_1U_2 \cap U_1U_3$. Die Idee für die Suche nach solchen U_1, U_2, U_3 beruht auf der Tatsache, dass die einzige Implikation in dem Beweis von Lemma 2.2 nicht umgekehrt werden kann. Wenn ein Wort x in L_1L_2 und auch in L_1L_3 liegt, bedeutet das noch nicht $x = yz$, wobei $y \in L_1$ und $z \in L_2 \cap L_3$. Es kann passieren, dass es zwei verschiedene Zerlegungen von x gibt, so dass gilt $x = y_1z_1 = y_2z_2$ mit $y_1 \neq y_2$ und $y_1 \in L_1$, $z_1 \in L_2$ und $y_2 \in L_1$, $z_2 \in L_3$.

Lemma 2.3. *Es existieren U_1, U_2, $U_3 \in (\Sigma_{\text{bool}})^*$, so dass*

$$U_1(U_2 \cap U_3) \subsetneq U_1U_2 \cap U_1U_3.$$

Beweis. Zuerst wählen wir $U_2 = \{0\}$ und $U_3 = \{10\}$. Damit ist $U_2 \cap U_3 = \emptyset$ und damit auch $U_1(U_2 \cap U_3) = \emptyset$ für jede Sprache U_1. Jetzt reicht es aus, U_1 so zu wählen, dass $U_1U_2 \cap U_1U_3$ nicht leer wird. Wir setzen $U_1 = \{\lambda, 1\}$. Dann ist $U_1U_2 = \{0, 10\}$, $U_1U_3 = \{10, 110\}$ und damit $U_1U_2 \cap U_1U_3 = \{10\}$. □

Aufgabe 2.10. Seien L_1, L_2 und L_3 Sprachen über dem Alphabet $\{0\}$. Gilt

$$L_1(L_2 \cap L_3) = L_1L_2 \cap L_1L_3?$$

Aufgabe 2.11. Seien $L_1 \subseteq \Sigma_1^*$ und $L_2, L_3 \subsetneq \Sigma_2^*$ für zwei Alphabete Σ_1 und Σ_2 mit $\Sigma_1 \cap \Sigma_2 = \emptyset$. Gilt

$$L_1(L_2 \cap L_3) = L_1L_2 \cap L_1L_3?$$

Aufgabe 2.12. Existieren Sprachen L_1, L_2 und L_3, so dass $L_1(L_2 \cap L_3)$ endlich und $L_1L_2 \cap L_1L_3$ unendlich ist?

Im Folgenden üben wir noch den Umgang mit dem Kleene'schen Stern.

http://www.springer.com/978-3-658-06432-7

Theoretische Informatik
Formale Sprachen, Berechenbarkeit, Komplexitätstheorie, Algorithmik, Kommunikation und Kryptographie
Hromkovič, J.
2014, XVIII, 349 S. 87 Abb., Softcover
ISBN: 978-3-658-06432-7