Inhaltsverzeichnis

Lemma 2.59 *Zu jedem Kellerautomaten $\mathcal{M}$ gibt es eine kontextfreie Grammatik G mit $L(G) = T(\mathcal{M})$.* □

Durch Kombination der beiden vorstehenden Lemmata erhalten wir das Hauptresultat dieses Abschnittes.

Satz 2.60 *Die beiden folgenden Aussagen sind für eine Sprache L äquivalent:*
i) L ist eine kontextfreie Sprache.
ii) $L = T(\mathcal{M})$ gilt für einen Kellerautomaten $\mathcal{M}$. □

Der Kellerautomat ist nach Definition nichtdeterministisch. Auch hier kann eine deterministische Variante eingeführt werden, bei der es zu jeder Konfiguration genau eine Folgekonfiguration gibt. Dafür reicht es zu fordern, dass alle Mengen $\delta(z, x, \gamma)$ einelementig sind. Der in Beispiel 2.57 angegebene Kellerautomat $\mathcal{M}$ ist deterministisch. Damit ist klar, dass deterministische Kellerautomaten nichtreguläre Sprachen akzeptieren können. Andererseits kann gezeigt werden, dass deterministische Kellerautomaten nicht in der Lage sind, die Sprache $\{ww^R : w \in \{a, b\}\}$ zu akzeptieren. Somit liegt die Menge der von deterministischen Kellerautomaten akzeptierten Sprachen echt zwischen der der regulären Sprachen und der der kontextfreien Sprachen.

2.3 Sprachen und algebraische Operationen

Nachdem wir im Abschnitt 2.1 verschiedene Typen formaler Sprachen mittels erzeugender Grammatiken definiert haben, gelang uns im Abschnitt 2.2 eine Charakterisierung der zugehörigen Sprachmengen mittels verschiedener Typen von Automaten. Ziel dieses Abschnittes ist es, eine weitere Charakterisierung der Menge der regulären Sprachen anzugeben, indem wir zeigen, dass sie sich als spezielle (universelle) Algebra beschreiben lassen.

Da Sprachen Mengen sind, können wir auf diese problemlos die mengentheoretischen Operationen Vereinigung und Durchschnitt anwenden. Eine weitere wichtige mengentheoretische Operation ist die Komplementbildung, bei der aber erst zu klären ist, bezüglich welcher Gesamtheit das Komplement zu bilden ist. Sei zum Beispiel $L \subseteq X^*$. Dann ist sicher $X^* \setminus L$ eine mögliche Definition des Komplements. Jedoch gilt natürlich für jedes Symbol $a \notin X$ auch $L \subseteq (X \cup \{a\})^*$, womit auch $(X \cup \{a\})^* \setminus L$ als Komplement möglich wäre. Wir wollen uns hier auf den Fall beschränken, dass das zugrunde liegende Alphabet minimal gewählt wird.
Für eine Sprache L definieren wir $alph(L)$ als die Menge aller Buchstaben, die in mindestens einem Wort von L vorkommen und das Komplement von L als

$$\overline{L} = (alph(L))^* \setminus L.$$

Wir definieren nun einige der Algebra entlehnten Operationen.

Definition 2.61 *Es seien L, L_1, L_2 Sprachen über einem Alphabet X. Wir definieren dann das Produkt von L_1 und L_2 durch*

$$L_1 \cdot L_2 = \{w_1 w_2 : w_1 \in L_1, w_2 \in L_2\}.$$

Weiterhin setzen wir

$$\begin{aligned} L^0 &= \{\lambda\}, \\ L^{n+1} &= L^n \cdot L \quad \textit{für} \quad n \geq 0 \end{aligned}$$

und definieren den KLEENE-*Abschluss (oder* KLEENE-$*$*) von* L *durch*

$$L^* = \bigcup_{n \geq 0} L^n$$

und den positiven KLEENE-*Abschluss (oder* KLEENE-$+$*) von* L *durch*

$$L^+ = \bigcup_{n \geq 1} L^n.$$

Falls keine Missdeutungen möglich sind, lassen wir wie üblich den Punkt als Operationszeichen beim Produkt fort.

Beispiel 2.62 Seien

$$L = \{ab, ac\} \quad \text{und} \quad L' = \{ab^n a : n \geq 1\}$$

gegeben. Dann ergeben sich:

$$\begin{aligned} L \cdot L = L^2 &= \{abab, abac, acab, acac\}, \\ L \cdot L' &= \{abab^n a : n \geq 1\} \cup \{acab^n a : n \geq 1\}, \\ (L')^3 &= \{ab^i aab^j aab^k a : i \geq 1, j \geq 1, k \geq 1\}, \\ L^* &= \{ax_1 ax_2 \ldots ax_r : r \geq 1, x_i \in \{b, c\}, 1 \leq i \leq r\} \cup \{\lambda\}, \\ (L')^+ &= \{ab^{s_1} aab^{s_2} a \ldots ab^{s_t} a : t \geq 1, s_j \geq 1, 1 \leq j \leq t\}. \end{aligned}$$

Vom algebraischen Standpunkt aus ist das Produkt das übliche Komplexprodukt in der (freien) Halbgruppe der Wörter über X. L^* ist dann die kleinste Halbgruppe mit neutralem Element, die L enthält, und L^+ ist entsprechend die kleinste Halbgruppe, die L enthält.
Wir bemerken, dass nach Definition stets

$$L^* = L^+ \cup L^0 = L^+ \cup \{\lambda\}$$

gilt, während $L^+ = L^* \setminus \{\lambda\}$ nur dann gilt, wenn $\lambda \notin L$ gilt.
Weiterhin merken wir an, dass im Spezialfall $L = X$ die Menge L^n aus genau allen Wörtern der Länge n über X besteht. Somit ist dann L^* die Menge aller Wörter über X, d.h. $L^* = X^*$, womit auch die Rechtfertigung für die Bezeichnung X^* in diesem Zusammenhang nachgewiesen ist.
Mit Hilfe der mengentheoretischen und den eben eingeführten Operationen lassen sich einige Sprachen sehr einfach beschreiben, für die wir bisher „relativ umständliche“ Definitionen gegeben haben. Wir wollen dies an einigen Beispielen demonstrieren.
Da offensichtlich nach Definition für jedes Symbol x

$$\{x\}^* = \{x^n : n \geq 0\} \quad \text{und} \quad \{x\}^+ = \{x^n : n \geq 1\} = \{x\}\{x\}^*$$

gelten, können wir die in den Beispielen 2.47 bzw. 2.48 akzeptierten (regulären) Sprachen wie folgt beschrieben:

$$\begin{aligned}\{c^{n_1}aac^{n_2}aa\ldots c^{n_k}aa : k \geq 1, n_1 \geq 0, n_i \geq 1, 2 \leq i \leq k\} &= \{c\}^*\{a\}\{a\}(\{c\}^+\{a\}\{a\})^* \\ &= \{c\}^*\{a\}\{a\}(\{c\}\{c\}^*\{a\}\{a\})^*\end{aligned}$$

und

$$\{a^n b^m : n \geq 1, m \geq 2\} = \{a\}^+\{b\}\{b\}^+.$$

Die Sprache R bestehe aus allen Wörtern über dem Alphabet X, die mindestens einen Buchstaben aus der Menge $Y \subseteq X$ enthalten. Hierfür ergibt sich

$$R = \bigcup_{x \in Y} X^*\{x\}X^*.$$

Satz 2.63 *Wenn L und L' reguläre Sprachen sind, so sind auch die Sprachen*
i) $L \cup L'$,
ii) $L \cap L'$,
iii) $V^* \setminus L$ (wobei $L \subseteq V^*$ gilt),
iv) $L \cdot L'$,
v) L^+ und L^*
regulär.

Beweis. i) Es seien L_1 und L_2 zwei reguläre Sprachen über dem Alphabet T. Wir haben zu zeigen, dass auch $L_1 \cup L_2$ eine reguläre Sprache (über T) ist. Dazu seien

$$G_1 = (N_1, T_1, P_1, S_1) \quad \text{und} \quad G_2 = (N_2, T_2, P_2, S_2)$$

zwei reguläre Grammatiken mit

$$L(G_1) = L_1 \quad \text{und} \quad L(G_2) = L_2.$$

Offenbar können wir ohne Beschränkung der Allgemeinheit annehmen, dass

$$T_1 = T_2 = T \quad \text{und} \quad N_1 \cap N_2 = \emptyset$$

gelten (notfalls sind die Nichtterminale umzubenennen). Ferner sei S ein Symbol, das nicht in $N_1 \cup N_2 \cup T$ liegt. Wir betrachten nun die reguläre Grammatik

$$G = (N_1 \cup N_2 \cup \{S\}, T, P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}, S).$$

Offenbar hat jede Ableitung in G die Form

$$S \Longrightarrow S_i \Longrightarrow^* w, \tag{2.1}$$

wobei $i \in \{1, 2\}$ gilt und $S_i \Longrightarrow^* w$ eine Ableitung in G_i ist (da wegen $N_1 \cap N_2 = \emptyset$ keine Symbole aus N_j, $j \neq i$ entstehen können und damit keine Regeln aus P_j anwendbar sind). Folglich gilt $w \in L(G_i)$. Hieraus folgt sofort

$$L(G) \subseteq L(G_1) \cup L(G_2) = L_1 \cup L_2.$$

Man sieht aber auch aus (2.1) sofort, dass jedes Element aus $L(G_i)$, $i \in \{1,2\}$, erzeugt werden kann, womit auch die umgekehrte Inklusion

$$L(G) \supseteq L(G_1) \cup L(G_2) = L_1 \cup L_2$$

gezeigt ist.

ii) Wir haben zu zeigen, dass für zwei reguläre Sprachen L_1 und L_2 auch ihr Durchschnitt $L_1 \cap L_2$ regulär ist. Wir führen den Beweis nur für den Fall dass $\lambda \notin L_1 \cap L_2$ liegt und überlassen dem Leser die Modifikationen für die allgemeine Situation.
Es seien dazu wieder

$$G_1 = (N_1, T_1, P_1, S_1) \quad \text{und} \quad G_2 = (N_2, T_2, P_2, S_2)$$

reguläre Grammatiken mit

$$L(G_1) = L_1 \quad \text{und} \quad L(G_2) = L_2.$$

Diesmal können wir ohne Beschränkung der Allgemeinheit neben $T = T_1 = T_2$ noch annehmen, dass G_1 und G_2 den in Satz 2.24 gegebenen Bedingungen genügen. Wir betrachten diesmal die reguläre Grammatik

$$G = (N_1 \times N_2, T, P, (S_1, S_2))$$

mit

$$\begin{aligned} P \;=\; & \{(A_1, A_2) \to a(B_1, B_2) : A_1 \to aB_1 \in P_1, A_2 \to aB_2 \in P_2\} \\ & \cup \{(A_1, A_2) \to a : A_1 \to a \in P_1, A_2 \to a \in P_2\}. \end{aligned}$$

Es ist leicht zu sehen, dass

$$(S_1, S_2) \Longrightarrow^* w'(A_1, A_2) \Longrightarrow^* w$$

genau dann gilt, wenn es in G_1 und G_2 Ableitungen

$$S_1 \Longrightarrow^* w'A_1 \Longrightarrow^* w \quad \text{und} \quad S_2 \Longrightarrow^* w'A_2 \Longrightarrow^* w$$

gibt. Folglich gilt $w \in L(G)$ genau dann, wenn auch $w \in L(G_1)$ und $w \in L(G_2)$ erfüllt sind. Somit ergibt sich

$$L(G) = L(G_1) \cap L(G_2) = L_1 \cap L_2.$$

Damit ist der Durchschnitt von L_1 und L_2 als regulär nachgewiesen.

iii) Es sei L eine reguläre Sprache. Dann gibt es einen endlichen Automaten

$$\mathcal{A} = (alph(L), Z, z_0, F, \delta)$$

mit $T(\mathcal{A}) = L$, der also L akzeptiert. Offenbar gilt daher genau dann $w \in \overline{L}$ oder gleichwertig $w \notin T(\mathcal{A})$, wenn $\delta(z_0, w) \notin F$, d.h. $\delta(z_0, w) \in Z \setminus F$ ist. Somit akzeptiert der endliche Automat

$$\mathcal{A}' = (alph(L), Z, z_0, Z \setminus F, \delta)$$

das Komplement von L, welches damit als regulär nachgewiesen ist.
iv) Es seien wieder

$$G_1 = (N_1, T_1, P_1, S_1) \quad \text{und} \quad G_2 = (N_2, T_2, P_2, S_2)$$

reguläre Grammatiken mit

$$L(G_1) = L_1 \quad \text{und} \quad L(G_2) = L_2$$

und $N_1 \cap N_2 = \emptyset$. Wir konstruieren aus G_1 und G_2 die reguläre Grammatik

$$G = (N_1 \cup N_2, T, P_1' \cup P_2, S_1)$$

mit

$$P_1' = \{A \to wB : A \to wB \in P_1, B \in N_1\} \cup \{A \to wS_2 : A \to w \in P_1, w \in T^*\}.$$

Entsprechend dieser Konstruktion sind die Ableitungen in G von der Form

$$S_1 \Longrightarrow^* w'A \Longrightarrow w'wS_2 \Longrightarrow^* w'ww_2,$$

wobei $S_1 \Longrightarrow^* w'A \Longrightarrow w'w = w_1$ eine Ableitung in G_1 und $S_2 \Longrightarrow^* w_2$ eine Ableitung in G_2 sind. Damit ergibt sich

$$L(G) = \{w_1w_2 : w_1 \in L(G_1), w_2 \in L(G_2)\} = L(G_1) \cdot L(G_2).$$

v) Wir beweisen die Aussage zuerst für L^+.
Es sei $G = (N, T, P, S)$ eine reguläre Grammatik mit $L(G) = L$. Wir konstruieren die reguläre GRammatik $G' = (N, T, P', S)$, wobei P' aus P entsteht, indem wir zu P die Regeln

$$A \to wS \text{ für } A \to w \in P, w \in T^*$$

hinzufügen. Die Ableitungen sind dann (bis auf die Reihenfolge der Anwendung der Regeln) von der Form

$$\begin{aligned} S \quad &\Longrightarrow \quad w_1'A_1 \Longrightarrow w_1'w_1''S \Longrightarrow^* w_1'w_2''w_2'A_2 \Longrightarrow w_1'w_1''w_2'w_2''S \\ &\Longrightarrow^* \quad w_1'w_1'' \dots w_{n-1}'w_{n-1}''S \Longrightarrow^* w_1'w_1'' \dots w_{n-1}'w_{n-1}''w_n, \end{aligned}$$

wobei $w_i'w_i'' \in L(G)$ for $1 \le i \le n-1$ und $w_n \in L(G)$ gelten. Hieraus folgt leicht die zu beweisende Aussage.
Wir geben nun die Modifikationen für den Kleene-$*$. Gilt $\lambda \in L$, so können wir wegen der dann gegebenen Gültigkeit von $L^* = L^+$ wie oben vorgehen. Ist $\lambda \notin L$, so haben wir $L^* = L^+ \cup \{\lambda\}$. Da $\{\lambda\}$ eine reguläre Sprache ist (erzeugt von der Grammatik mit der einzigen Regel $S \to \lambda$), folgt die Regularität von L^* aus Teil i) dieses Satzes. □

Wir haben oben Beispiele betrachtet, bei denen (reguläre) Sprachen erzeugt werden konnten, indem die Operationen Vereinigung, Produkt und (positiver) Kleene-Abschluss auf einelementige Mengen iteriert angewandt wurden. Wir wollen nun das auf S. C. Kleene zurückgehende Resultat zeigen, dass auf diese Weise genau die regulären Sprachen beschrieben werden können. Dafür verwenden wir reguläre Ausdrücke, die auch an anderer Stelle in der Informatik zur Beschreibung von Mengen eingesetzt werden.

Definition 2.64 *Reguläre Ausdrücke über einem Alphabet X sind induktiv wie folgt definiert:*

1. *$\emptyset$, λ und x mit $x \in X$ sind reguläre Ausdrücke.*
2. *Sind R_1, R_2 und R reguläre Ausdrücke, so sind auch $(R_1 + R_2)$, $(R_1 \cdot R_2)$ und R^* reguläre Ausdrücke.*
3. *Ein Ausdruck ist nur dann regulär, wenn dies aufgrund von 1. oder 2. der Fall ist.*

Wir ordnen nun jedem regulären Ausdruck über X eine Sprache über X zu.

Definition 2.65 *Die einem regulären Ausdruck U über dem Alphabet X zugeordnete Menge $M(U)$ ist induktiv durch die folgenden Festlegungen definiert:*

- *$M(\emptyset) = \emptyset$, $M(\lambda) = \{\lambda\}$ und $M(x) = \{x\}$ für $x \in X$,*
- *Sind R_1, R_2 und R reguläre Ausdrücke, so gelten*

$$\begin{aligned} M((R_1 + R_2)) &= M(R_1) \cup M(R_2), \\ M((R_1 \cdot R_2)) &= M(R_1) \cdot M(R_2), \\ M(R^*) &= (M(R))^*. \end{aligned}$$

Beispiel 2.66 Sei $X = \{a, b, c\}$. Dann sind nach 1. aus Definition 2.64

$$R_0 = \lambda,\ R_1 = a,\ R_2 = b,\ R_3 = c$$

reguläre Ausdrücke über X. Nach 2. aus Definition 2.64 sind dann auch die folgenden Konstrukte reguläre Ausdrücke:

$$\begin{aligned} R_1' &= (R_1 \cdot R_1) = (a \cdot a), \\ R_1'' &= (R_1' \cdot R_1) = ((a \cdot a) \cdot a), \\ R_2' &= R_2^* = b^*, \\ R_2'' &= (R_2' + R_1'') = (b^* + ((a \cdot a) \cdot a))), \\ R_3' &= R_3^* = c^*, \\ R_3'' &= (R_3 \cdot R_3') = (c \cdot c^*), \\ R_4 &= (R_2'' \cdot R_3'') = ((b^* + ((a \cdot a) \cdot a))) \cdot (c \cdot c^*)), \\ R_5 &= (R_0 + R_4) = (\lambda + ((b^* + ((a \cdot a) \cdot a))) \cdot (c \cdot c^*))). \end{aligned}$$

Entsprechend Definition 2.65 erhalten wir die folgenden zugeordneten Mengen (wobei wir offensichtliche Vereinfachungen stets vornehmen):

$$\begin{aligned} M(R_0) &= \{\lambda\},\ M(R_1) = \{a\},\ M(R_2) = \{b\},\ M(R_3) = \{c\}, \\ M(R_1') &= = M((R_1 \cdot R_1)) = \{a\} \cdot \{a\} = \{a^2\}, \\ M(R_1'') &= M((R_1' \cdot R_1)) = \{a^2\} \cdot \{a\} = \{a^3\}, \\ M(R_2') &= M(R_2^*) = \{b\}^* = \{b^m : m \geq 0\}, \end{aligned}$$

$$
\begin{aligned}
M(R_2'') &= M((R_2' + R_1'')) = \{b^m : m \geq 0\} \cup \{a^3\}\,, \\
M(R_3') &= M(R_3^*) = \{c\}^* = \{c^n : n \geq 0\}\,, \\
M(R_3'') &= M((R_3 \cdot R_3')) = \{c\}\{c^n : n \geq 0\} = \{c^n : c \geq 1\}\,, \\
M(R_4) &= M((R_2'' \cdot R_3'')) = (\{b^m : m \geq 0\} \cup \{a^3\}) \cdot \{c^n : n \geq 1\} \\
&= \{b^m c^n : m \geq 0, n \geq 1\} \cup \{a^3 c^n : n \geq 3\}\,, \\
M(R_5) &= M((R_0 + R_4)) = \{\lambda\} \cup (\{b^m c^n : m \geq 0, n \geq 1\} \cup \{a^3 c^n : n \geq 3\}) \\
&= \{\lambda\} \cup \{b^m c^n : m \geq 0, n \geq 1\} \cup \{a^3 c^n : n \geq 3\}.
\end{aligned}
$$

Ist $U = ((\ldots((R_1 + R_2) + R_3) + \ldots) + R_n)$, so schreiben wir dafür kurz

$$U = \sum_{i=1}^{n} R_i\,.$$

Offenbar ist

$$M(U) = \bigcup_{i=1}^{n} M(R_i)\,.$$

In analoger Weise benutzen wir Summen bzw. Vereinigungen über gewisse Indexbereiche.

Satz 2.67 *Eine Sprache L ist genau dann regulär, wenn es einen regulären Ausdruck R mit $M(R) = L$ gibt.*

Beweis. $\Longrightarrow$) Sei L eine reguläre Sprache. Dann gibt es einen endlichen deterministischen Automaten

$$\mathcal{A} = (X, Z, z_0, F, \delta)$$

mit $T(\mathcal{A}) = L$. Ohne Beschränkung der Allgemeinheit können wir annehmen, dass

$$Z = \{0, 1, 2, \ldots r\} \quad \text{und} \quad z_0 = 0$$

für ein gewisses $k \geq 0$ gelten. Für $i, j, k \in Z$ bezeichnen wir mit $L_{i,j}^k$ die Menge aller Wörter w mit den beiden folgenden Eigenschaften:

- $\delta(i, w) = j$,
- für jedes $u \neq \lambda$ mit $w = uu'$ und $|u| < |w|$ gilt $\delta(i, u) < k$.

Offenbar gilt dann

$$L = T(\mathcal{A}) = \bigcup_{j \in F} L_{0,j}^{r+1}. \tag{2.2}$$

Wir beweisen nun, dass es für jede Menge $L_{i,j}^k$ einen regulären Ausdruck $R_{i,j}^k$ mit $M(R_{i,j}^k) = L_{i,j}^k$ gibt. Der Beweis hierfür wird nun mittels Induktion über k gezeigt.
Sei zuerst $k = 0$. Für $i \neq j$ besteht $L_{i,j}^0$ nach Definition aus allen Wörtern w, die den Zustand i direkt in den Zustand j überführen, da aufgrund der zweiten Bedingung keine Zwischenzustände auftreten können. Damit muss w ein Wort der Länge 1 sein, und es gilt

$$L_{i,j}^0 = \{x : x \in X, \delta(i, x) = j\}.$$

Wir schreiben dies als

$$L_{i,j}^0 = \bigcup_{\substack{x \in X \\ \delta(i,x)=j}} \{x\}.$$

Damit gilt auch

$$L_{i,j}^0 = M(\sum_{\substack{x \in X \\ \delta(i,x)=j}} x).$$

womit die Aussage bewiesen ist. Gilt $i = j$, so kommt zu den Wörtern der Länge 1, die i in i transformieren, noch das leere Wort hinzu. Daher ist auch in diesem Fall

$$L_{i,j}^0 = M(\lambda + \sum_{\substack{x \in X \\ \delta(i,x)=i}} x).$$

Sei nun $k \geq 1$ und für alle Mengen der Form $L_{i,j}^s$ mit $s < k$ existiere ein regulärer Ausdruck $R_{i,j}^s$ mit $L_{i,j}^s = M(R_{i,j}^s)$. Wir zeigen zuerst

$$L_{i,j}^k = L_{i,k-1}^{k-1}(L_{k-1,k-1}^{k-1})^* L_{k-1,j}^{k-1} \cup L_{i,j}^{k-1}\,. \tag{2.3}$$

Sei $w = x_1 x_2 \dots x_n$ ein Wort aus $L_{i,j}^k$. Für $1 \leq p \leq n-1$ setzen wir

$$z_p = \delta(i, x_1 x_2 \dots x_p).$$

Gilt $z_p < k-1$ für $1 \leq p \leq n-1$, so ist w auch in $L_{i,j}^{k-1}$. Folglich erhalten wir $w \in R$. Deshalb sei nun für gewisse $t \geq 1$ und $1 \leq p_1 \leq p_2 \leq \ldots \leq p_t \leq n-1$

$$z_{p_1} = z_{p_2} = \ldots = z_{p_t} = k-1 \quad \text{und} \quad z_p < k-1 \text{ für } p \notin \{p_1, p_2, \dots, p_t\}.$$

Dann gelten

$$\begin{aligned}
&\delta(i, x_1 x_2 \dots x_{p_1}) = k-1,\\
&\delta(k-1, x_{p_q+1} x_{p_q+2} \dots x_{p_{q+1}}) = k-1 \quad \text{für} \quad 1 \leq q \leq t-1,\\
&\delta(k-1, x_{p_t} x_{p_t+1} \dots x_n) = j.
\end{aligned}$$

Weiterhin wird bei keiner dieser Überführungen als Zwischenschritt der Zustand $k-1$ erreicht. Daher erhalten wir

$$\begin{aligned}
&x_1 x_2 \dots x_{p_1} \in L_{i,k-1}^{k-1},\\
&x_{p_q} x_{p_q+1} x_{p_q+2} \dots x_{p_{q+1}} \in L_{k-1,k-1}^{k-1} \quad \text{für} \quad 1 \leq q \leq t-1,\\
&x_{p_t} x_{p_t+1} x_{p_t+2} \dots x_n \in R_{k-1,j}^{k-1}.
\end{aligned}$$

und

$$w = x_1 \dots x_{p_1} \dots x_{p_2} \dots x_{p_t} \dots x_n \in L_{i,k-1}^{k-1}(L_{k-1,k-1}^{k-1})^* L_{k-1,j}^{k-1}.$$

Folglich ist

$$L_{i,j}^k \subseteq L_{i,k-1}^{k-1}(L_{k-1,k-1}^{k-1})^* L_{k-1,j}^{k-1} \cup L_{i,j}^{k-1}\,.$$

Die umgekehrte Inklusion und damit die Gleichheit aus (2.3) folgt durch analoge Schlüsse.

(2.3) liefert nun sofort

$$\begin{aligned} L_{i,j}^k &= M(R_{i,k-1}^{k-1})M(R_{k-1,k-1}^{k-1})^* M(R_{k-1,j}^{k-1}) \cup M(L_{i,j}^{k-1}) \\ &= M((((R_{i,k-1}^{k-1} \cdot [R_{k-1,k-1}^{k-1}]^*) \cdot R_{k-1,j}^{k-1}) + R_{i,j}^{k-1}))\,, \end{aligned}$$

womit gezeigt ist, dass jede Menge $L_{i,j}^k$ durch einen regulären Ausdruck $R_{i,j}^k$ beschrieben werden kann.
Beachten wir nun noch die aus (2.2) herrührende Relation

$$L = \bigcup_{j \in F} L_{0,j}^{r+1} = M(\sum_{j \in F} R_{0,j}^{r+1})$$

so ist diese Richtung des Satzes von KLEENE gezeigt.

$\Longleftarrow$) Wir zeigen induktiv, dass für jeden regulären Ausdruck U die zugehörige Menge $M(U)$ regulär ist.
Ist U ein regulärer nach Ausdruck nach 1. aus Definition 2.64, so sind die zugehörigen Mengen $M(\emptyset) = \emptyset$, $M(\lambda) = \{\lambda\}$ und $M(x) = \{x\}$ mit $x \in X$ alle endlich und folglich auch regulär (siehe auch Übungsaufgabe 5).
Sei nun U ein regulärer Ausdruck, der aus den regulären Ausdrücken R_1, R_2 und R entsprechend 2. aus Definition 2.64 gebildet wurde, wobei die Mengen $M(R_1)$, $M(R_2)$ und $M(R)$ nach Induktionsvoraussetzung regulär sind. Falls $U = (R_1 + R_2)$ gilt, so erhalten wir $M(U) = M(R_1) \cup M(R_2)$. Nach Satz 2.63 i) ist $M(U)$ regulär. Gelten $U = (R_1 \cdot R_2)$ bzw. $U = R^*$, so sind nach den Saztz 2.63 die zugehörigen Mengen $M(U) = M(R_1) \cdot M(R_2)$ bzw. $M(U) = (M(R))^*$ ebenfalls regulär. □

Wir geben noch eine andere Formulierung des Satzes von KLEENE an, bei der wir statt der regulären Ausdrücke eine direkte Beschreibung durch die Mengenoperationen angeben, die bei der Interpretation der Ausdrücke durch Mengen auftreten.

Satz 2.67' *Eine Sprache $L \subseteq X$ ist genau dann regulär, wenn sie in endlich vielen Schritten mittels der Operationen Vereinigung, Produkt und* KLEENE*-Abschluss aus den Mengen $\emptyset$, $\{\lambda\}$ und $\{x\}$ für $x \in X$ erzeugt werden kann.* □

Das folgende Beispiel verdeutlicht die in den Beweisen der vorstehenden Lemmata angegebenen Konstruktionen.

Beispiel 2.68 Wir betrachten den endlichen Automaten $\mathcal{A}$ aus Beispiel 2.47 und konstruieren zu der durch ihn akzeptierten Sprache die Darstellung durch Vereinigung, Produkt und KLEENE-Abschluss. Zur Vereinfachung der Schreibweisen werden wir dabei statt z_i die Bezeichnung i verwenden. Es ergibt sich

$$\begin{aligned} T(\mathcal{A}) &= L_{0,2}^4 \\ &= L_{0,3}^3 (L_{3,3}^3)^* L_{3,2}^3 \cup L_{0,2}^3 \\ &= L_{0,2}^3 (\text{wegen } L_{3,2}^3 = \emptyset) \\ &= L_{0,2}^2 (L_{2,2}^2)^* L_{2,2}^2 \cup L_{0,2}^2 \\ &= L_{0,2}^2 (L_{2,2}^2)^* (\text{wegen } \lambda \in L_{0,2}^2) \\ &= (L_{0,1}^1 (L_{1,1}^1)^* L_{1,2}^1 \cup L_{0,2}^1)(L_{2,1}^1 (L_{1,1}^1)^* L_{1,2}^1 \cup L_{2,2}^1)^* \end{aligned}$$

$$
\begin{aligned}
&= L^1_{0,1}\{a\} \cdot (L^1_{2,1}\{a\})^* \text{wegen } L^1_{1,2} = \{a\}, L^1_{1,1} = L^1_{0,2} = L^1_{2,2} = \emptyset) \\
&= (L^0_{0,0}(L^0_{0,0})^* L^0_{0,1} \cup L^0_{0,1})\{a\} \cdot ((L^0_{2,0}(L^0_{0,0})^* L^0_{0,1} \cup L^0_{2,1})\{a\})^* \\
&= (\{\lambda, c\}\{\lambda, c\}^*\{a\} \cup \{a\})\{a\} \cdot (((\{c\}\{\lambda, c\}^*\{a\})\{a\})^*,
\end{aligned}
$$

woraus die abschließende Darstellung

$$T(\mathcal{A}) = ((((((\lambda + c) \cdot (\lambda + c)^*) \cdot a) + a) \cdot a) \cdot (((c \cdot (\lambda + c)^*) \cdot a) \cdot a^*))$$

gewonnen wird.
Wir bemerken, dass diese Darstellung nicht mit der auf auf Seite 98 gegebenen Darstellung

$$T(\mathcal{A}) = \{c\}^*\{a\}\{a\}(\{c\}\{c\}^*\{a\}\{a\})^*$$

identisch ist. Daher zeigt dieses Beispiel auch noch, dass es mehrere Beschreibungen durch Operationen für eine reguläre Menge geben kann.

Wir setzen das Beispiel jetzt fort, indem wir ausgehend von der Beschreibung von $T(\mathcal{A})$ eine Grammatik konstruieren, die $T(\mathcal{A})$ erzeugt. Zur Abkürzung des Prozesses starten wir mit der letzten oben gegebenen Darstellung für $T(\mathcal{A})$.
Offenbar ist für alle nachfolgenden Grammatiken die Menge T der Terminale durch die Eingabemenge $\{a, b, c\}$ von $\mathcal{A}$ gegeben.
Wir konstruieren nun zuerst Grammatiken, die die notwendigen (einelementigen) Mengen erzeugen. Ferner sichern wir dabei die Disjunktheit aller Mengen von Nichtterminalen, da diese in den Beweisen der Abgeschlossenheit unter Vereinigung, Produkt und Kleene-Abschluss teilweise vorausgesetzt wurde. Wir gehen daher von

$$
\begin{aligned}
G_i &= (\{S_i\}, T, \{S_i \to c\}, S_i) \quad \text{für } i \in \{1, 4, 5\} \\
G_j &= (\{S_j\}, T, \{S_j \to a\}, S_j) \quad \text{für } i \in \{2, 3, 6, 7\}
\end{aligned}
$$

aus, für die

$$L(G_i) = \{c\} \quad \text{und} \quad L(G_j) = \{a\}$$

und damit auch

$$T(\mathcal{A}) = L(G_1)^* L(G_2) L(G_3)(L(G_4) L(G_5)^* L(G_6) L(G_7))^*$$

gelten. Wir gehen nun entsprechend den Konstruktionen des Satzes 2.63 vor. In der folgenden Tabelle geben wir stets die erzeugte Sprache, die Regeln und das Axiom an (die Nichtterminale können aus den Regeln abgelesen werden).

$L(G_1)^* = \{a\}^*$	$S_1' \to \lambda, S_1' \to S_1, S_1 \to cS_1, S_1 \to c$	S_1'
$L(G_1)^*L(G_2)$	$S_1' \to S_2, S_1' \to S_1, S_1 \to cS_1, S_1 \to cS_2,$ $S_2 \to a$	S_1'
$L(G_1)^*L(G_2)L(G_3)$	$S_1' \to S_2, S_1' \to S_1, S_1 \to cS_1, S_1 \to cS_2,$ $S_2 \to cS_3, S_3 \to c$	S_1'
$L(G_5)^*$	$S_5' \to \lambda, S_5' \to S_5, S_5 \to cS_5, S_5 \to c$	S_5'
$L(G_4)L(G_5)^*$	$S_4 \to cS_5', S_5' \to \lambda, S_5' \to S_5, S_5 \to cS_5,$ $S_5 \to c$	S_4
$L(G_4)L(G_5)^*L(G_6)L(G_7)$	$S_4 \to cS_5', S_5' \to S_6, S_5' \to S_5, S_5 \to cS_5,$ $S_5 \to cS_6, S_6 \to aS_7, S_7 \to a$	S_4
$(L(G_4)L(G_5)^*L(G_6)L(G_7))^*$	$S_4' \to \lambda, S_4' \to S_4, S_4 \to cS_5', S_5' \to S_6,$ $S_5' \to S_5, S_5 \to cS_5, S_5 \to cS_6, S_6 \to aS_7,$ $S_7 \to a$	S_4'
$T(\mathcal{A})$	$S_1' \to S_2, S_1' \to S_1, S_1 \to cS_1, S_1 \to cS_2,$ $S_2 \to cS_3, S_3 \to cS_4', S_4' \to \lambda, S_4' \to S_4,$ $S_4 \to cS_5', S_5' \to S_6, S_5' \to S_5, S_5 \to cS_5,$ $S_5 \to cS_6, S_6 \to aS_7, S_7 \to a$	S_1'

2.4 Entscheidbarkeitsprobleme bei formalen Sprachen

Formale Sprachen sind für uns ein Modell, das als theoretische Grundlage der Untersuchung von Programmiersprachen, der Syntaxanalyse und der Compilerkonstruktion benutzt werden kann. In diesem Zusammenhang ist das folgende natürliche Entscheidungsprobleme von besonderem Interesse.

Das *Mitgliedsproblem* ist die Frage, ob eine gegebene Grammatik ein gegebenes Wort erzeugt. Hierbei ist aber wichtig, wie die Sprache gegeben ist. Entsprechend den vorhergehenden Abschnitten kann dies sowohl durch eine Grammatik als auch durch einen akzeptierenden Automaten (und im Fall einer regulären Sprache auch durch einen regulären Ausdruck) geschehen. Daraus resultieren mindestens die zwei folgenden Varianten des Mitgliedsproblems für kontextfreie Sprachen:

Gegeben: Grammatik $G = (N, T, P, S)$ und Wort $w \in T^*$
Frage : Ist w in $L(G)$ enthalten ?

oder

Gegeben: Kellerautomat $\mathcal{M} = (X, Z, \Gamma, z_0, F, \delta)$ und Wort $w \in X^*$
Frage: Ist W in $T(\mathcal{M})$ enthalten ?

Wir haben das Problem nur für kontextfreie Grammatiken bzw. Kellerautomaten angeben. Natürlich kann die gleiche Frage auch für andere Typen von Grammatiken gestellt werden, für beliebige Regelgrammatiken (bzw. Turing-Maschinen) oder kontextsensitive Grammatiken oder monotone Grammatiken (bzw. linear beschränkte Automaten) oder reguläre Grammatiken.
Im Folgenden interessieren wir uns zuerst dafür, ob das Problem entscheidbar ist oder nicht, d.h. wir untersuchen, ob es einen Algorithmus gibt, der die Frage beantwortet. Die

Antwort ist dann unabhängig von der Formulierung des Problems, da sowohl der Übergang von einer kontextfreien Grammatik G zu einem Kellerautomaten $\mathcal{M}$ mit $L(G) = T(\mathcal{M})$ als auch der umgekehrte Übergang von einem Kellerautomaten zu einer kontextfreien Grammatik konstruktiv - also mittels eines Algorithmus - erfolgen. Folglich haben beide Formulierungen stets die gleiche Antwort.
Eine analoge Situation ist auch hinsichtlich der anderen Typen von Grammatiken und zugehörigen Automaten gegeben.
Im Fall der Existenz eines Algorithmus zur Beantwortung des Problems ist natürlich auch die Komplexität des Algorithmus von großem Interesse. Hier ist eine Abhängigkeit vom Problem gegeben, da schon die Größe der Eingabe Grammatik bzw. Automat (Maschine) unterschiedlich sind. Wir geben hier stets nur die Komplexität des Algorithmus bezogen auf die Größe der Grammatik an. Ist man an der Komplexität bezogen auf die (hier noch nicht definierte) Größe des Automaten interessiert, so lässt sich diese meist leicht dadurch ermitteln, dass man den Aufwand für den Übergang vom Automaten zur Grammatik noch hinzufügt. Letzterer Aufwand kann aus den Konstruktionen in Abschnitt 2.2 relativ einfach ermittelt werden.
Wir bestimmen nun den Entscheidbarkeitsstatus und die Komplexität des Mitgliedsproblems für die Grammatiken der CHOMSKY-Hierarchie.

Satz 2.69 *Das Mitgliedsproblem ist für (beliebige) Regelgrammatiken unentscheidbar.*

Beweis. Aus den Sätzen 2.33 und 2.43 ergibt sich sofort, dass $w \in L(G)$ genau dann gilt, wenn die zugehörige TURING-Maschine auf w stoppt. Die Entscheidbarkeit des Mitgliedsproblems würde daher die Entscheidbarkeit der Frage, ob eine TURING-Maschine auf einem Wort stoppt, zur Folge haben. Das widerspricht aber Satz 1.28. □

Satz 2.70 *Das Mitgliedsproblem ist für monotone (oder kontextsensitive) Grammatiken entscheidbar.*

Beweis. Es seien die monotone Grammatik $G = (N, T, P, S)$ und das Wort $w \in T^*$ gegeben.
Entsprechend der Definition von monotonen Grammatiken kann $\lambda \in L(G)$ nur gelten, wenn P die Regel $S \to \lambda$ enthält. Daher ist das Mitgliedsproblem für $w = \lambda$ entscheidbar, und wir können von nun ab voraussetzen, dass $w \in T^+$ gilt.
Es sei

$$S = w_0 \Longrightarrow w_1 \Longrightarrow w_2 \Longrightarrow ... \Longrightarrow w_n = w$$

eine Ableitung von w in G. Falls $w_i = w_j$ für $i < j$ gilt, so ist auch

$$S = w_0 \Longrightarrow w_1 \Longrightarrow w_2 \Longrightarrow ... \Longrightarrow w_i \Longrightarrow w_{j+1} \Longrightarrow w_{j+2} \Longrightarrow ... \Longrightarrow w_n = w$$

eine Ableitung von w in G. Daher können wir ohne Beschränkung der Allgemeinheit annehmen, dass bei $w \in L(G)$ eine Ableitung von w in G existiert, in der keine Satzform mehrfach auftritt. Da bei monotonen Grammatiken $|w_{i-1}| > |w_i|$ ausgeschlossen ist und nur $\#(V)^k$ Wörter der Länge k über $V = N \cup T$ existieren, tritt innerhalb einer Ableitung von w stets nach höchstens $\#(V)^{|w|}$ Schritten eine Verlängerung der Satzform ein. Daher muss es, falls $w \in L(G)$ gilt, eine Ableitung von w in G geben, die höchstens $|w|\#(V)^{|w|+1}$

Schritte hat. Da es höchstens $\#(P)^{|w|\#(V)^{|w|+1}}$ Ableitungen dieser Länge gibt, besteht die Möglichkeit diese durchzutesten und damit festzustellen, ob $w \in L(G)$ gilt. □

Der eben beschriebene Algorithmus zur Lösung des Mitgliedsproblems für monotone (kontextsensitive) Grammatiken hat exponentielle Komplexität bez. der Länge von w, da $\#(P)^{|w|\#(V)^{|w|+1}}$ mögliche Ableitungen zu testen sind.
Aus Satz 2.70 folgt sofort, dass die monotonen Sprachen rekursiv sind. Damit ergibt sich unter Beachtung von Satz 2.37 die folgende Aussage, die dann die verbliebene Lücke bei der Behandlung der CHOMSKY-Hierarchie in Abschnitt 2.1 schließt.

Satz 2.71 $\mathcal{L}(MON) \subset \mathcal{L}(RE)$ □

Aus Satz 2.70 folgt natürlich sofort, dass das Mitgliedsproblem für kontextfreie und reguläre Grammatiken ebenfalls entscheidbar ist. Wir sind aber in der Lage für diese Grammatiktypen die Komplexität näher zu bestimmen. Zur Formulierung der Aussage benötigen wir den Begriff der Größe $k(G)$ einer Grammatik $G = (N, T, P, S)$, der durch

$$k(G) = \sum_{\alpha \to \beta \in P} |\alpha| + |\beta| + 1$$

definiert ist (wir fassen Eine Regel als Wort auf und addieren die Längen aller Regeln).

Satz 2.72 *i) Das Mitgliedsproblem ist für kontextfreie Grammatiken $G = (N, T, P, S)$ in* CHOMSKY*-Normalform in der Zeit $O(\#(P) \cdot |w|^3\})$ entscheidbar.*
ii) Das Mitgliedsproblem ist für kontextfreie Grammatiken $G = (N, T, P, S)$ in der Zeit $O(k(G) \cdot \#(N) \cdot \#(P) \cdot |w|^3)$ entscheidbar.

Beweis. i) Es seien die kontextfreie Grammatik $G = (N, T, P, S)$ in CHOMSKY-Normalform und ein Wort $w = a_1 a_2 \dots a_n$ der Länge n gegeben. Wir konstruieren schrittweise die Mengen $V_{i,j}$ mit $0 \leq i < j \leq n$ wie folgt: Zuerst setzen wir

$$V_{i-1,i} = \{A \mid A \in N, A \to a_i \in P\}.$$

Sind dann für $i < k < j$ die Mengen $V_{i,k}$ und $V_{k,j}$ bereits definiert, so setzen wir

$$V_{i,j} = \{A \mid A \in N, A \to BC \in P, B \in V_{i,k}, C \in V_{k,j} i < k < j\}.$$

Da es höchstens n Möglichkeiten für k gibt und für jedes k alle Regeln von P durchzumustern sind, kann jede Menge $V_{i,j}$ in $\#(P) \cdot n$ Schritten konstruiert werden. Da insgesamt $\frac{n(n+1)}{2}$ Mengen zu konstruieren sind, ergibt sich damit ein durch $\frac{\#(P)n^2(n+1)}{2}$ nach oben beschränkter Gesamtaufwand für die Konstruktion der Mengen.
Wir beweisen nun mittels Induktion über die Differenz $j - i$, dass

$$V_{i,j} = \{A \mid A \in N, A \Longrightarrow^* a_{i+1} a_{i+2} \dots a_j\} \tag{2.4}$$

ist.
Für $j - i = 1$ gilt dies nach Konstruktion.

Es sei nun $A \in V_{i,j}$. Dann gibt es nach Konstruktion Nichtterminale $B \in V_{i,k}$ und $C \in V_{k,j}$ mit $A \to BC \in P$. Nach Induktionsvoraussetzung gelten dann

$$B \Longrightarrow^* a_{i+1}a_{i+2}\dots a_k \quad \text{und} \quad C \Longrightarrow^* a_{k+1}a_{k+2}\dots a_j.$$

Folglich ergibt sich

$$A \Longrightarrow BC \Longrightarrow^* a_{i+1}a_{i+2}\dots a_kC \Longrightarrow^* a_{i+1}a_{i+2}\dots a_ka_{k+1}a_{k+2}\dots a_j.$$

Gilt umgekehrt $A \Longrightarrow^* a_{i+1}a_{i+2}\dots a_j$, so muss es wegen der CHOMSKY-Normalform Nichtterminale B und C und ein k mit $i < k < j$ und

$$A \to BC \in P, \quad B \Longrightarrow^* a_{i+1}a_{i+2}\dots a_k, \quad C \Longrightarrow^* a_{k+1}a_{k+2}\dots a_j$$

geben. Nach Induktionsvoraussetzung haben wir $B \in V_{i,k}$ und $C \in V_{k,j}$, woraus wir nach Konstruktion von $V_{i,j}$ dann $A \in V_{i,j}$ erhalten.
Somit ist (2.4) bewiesen.
Aus (2.4) ergibt sich aber genau dann $S \Longrightarrow^* a_1a_2\dots a_n = w$, wenn $S \in V_{0,n}$ gilt. Damit sind $w \in L(G)$ und $S \in V_{0,n}$ gleichwertig. Um $w \in L(G)$ zu entscheiden, reicht es also die Mengen $V_{i,j}$ mit $0 \le i < j \le n$ zu konstruieren und $S \in V_{0,n}$ zu überprüfen. Nach obigem ist daher die Entscheidung des Mitgliedproblems für G und w in $\theta(\#(P) \cdot |w|^3)$ Schritten möglich.

ii) folgt aus i) sofort, wenn wir beachten dass bei der Umwandlung einer beliebigen kontextfreien Grammatik $G = (N, T, P, S)$ in eine Grammatik $G' = (N', T, P', S')$ in CHOMSKY-Normalform entsprechend den Konstruktionen aus Abschnitt 2.1.2 die Beziehung $\#(P') = O(k(G) \cdot \#(N) \cdot \#(P))$ gilt. □

Beispiel 2.73 Wir illustrieren den eben beschriebenen Algorithmus, den sogenannten Cocke-Younger-Kasami-Algorithmus, anhand der Grammatik

$$G = (\{S, T, U\}, \{a, b\}, P, S)$$

mit den Regeln

$$S \to ST, T \to TU, T \to TT, U \to TS, S \to a, T \to a, U \to b$$

in P. Wir wollen zuerst untersuchen, ob das Wort $w = aabaa$ in $L(G)$ liegt. Wir müssen also die zugehörigen Mengen $V_{i,j}$ mit $0 \le i < j \le 5$ konstruieren. Es ergeben sich

$$\begin{aligned}
V_{0,1} &= \{A \mid A \to a \in P\} = \{S, T\},\\
V_{1,2} &= \{A \mid A \to a \in P\} = \{S, T\},\\
V_{2,3} &= \{A \mid A \to b \in P\} = \{U\},\\
V_{0,2} &= \{A \mid A \to BC \in P, B \in V_{0,1}, C \in V_{1,2}\} = \{S, T, U\},\\
V_{1,3} &= \{A \mid A \to BC \in P, B \in V_{1,2}, C \in V_{2,3}\} = \{T\},\\
V_{0,3} &= \{A \mid A \to BC \in P, B \in V_{0,1}, C \in V_{1,3}\}\\
&\quad \cup \{A' \mid A' \to B'C' \in P, B' \in V_{0,2}, C' \in V_{2,3}\}\\
&= \{S, T\} \cup \{T\} = \{S, T\}.
\end{aligned}$$

Die weiteren Mengen können der nachfolgenden Tabelle entnommen werden, wobei das i-te Symbol des Wortes w im Schnittpunkt der Zeile i und Spalte i und die Menge $V_{i,j}$ im Schnittpunkt der Zeile i und Spalte j eingetragen und die Mengenklammern fortgelassen wurden.

	0	1	2	3	4	5
0		S,T	S,T,U	S,T	S,T,U	S,T,U
1		a	S,T	T	T,U	T,U
2			a	U	$\emptyset$	$\emptyset$
3				b	S,T	S,T,U
4					a	S,T
5						a

Wegen $S \in V_{0,5}$ folgt $w = aabaa \in L(G)$.
Für $v = abaaa$ ergibt sich die Tabelle

	0	1	2	3	4	5
0		S,T	T	T,U	T,U	T,U
1		a	U	$\emptyset$	$\emptyset$	$\emptyset$
2			b	S,T	S,T,U	S,T,U
3				a	S,T	S,T,U
4					a	S,T
5						a

und damit $v \notin L(G)$ wegen $S \notin V_{0,5}$.

Eine genaue Analyse des Cocke-Younger-Kasami-Algorithmus ergibt, dass die Bestimmung der Mengen $V_{i,j}$ eine Analogie zur Matrizenmultiplikation aufweist. Hierdurch ist bei fester Grammatik G (und damit festem P) eine Verbesserung möglich, da Algorithmen für die Matrizenmultiplikation bekannt sind, die $O(n^\alpha)$ mit $\alpha < 3$ erfordern. So erfordert z.B. die Multiplikation von Matrizen nach STRASSEN nur $O(n^{\log_2(7)})$.

Für reguläre Sprachen läßt sich die folgende Verschärfung von Satz 2.72 angeben.

Satz 2.74 *Für eine reguläre Grammatik $G = (N, T, P, S)$ und ein Wort w ist in der Zeit $O(k(G) \cdot \#(N) \cdot |w|)$ entscheidbar, ob $w \in L(G)$ gilt.*

Beweis. Zuerst konstruieren wir entsprechend Satz 2.24 in der Zeit $O(\#(N)k(G))$ die reguläre Grammatik $G' = (N', T, P', S')$ zu G, die nur Regeln der Form $A \to aB$ oder $A \to a$ mit $A, B \in N', a \in T$ besitzt (vielleicht mit Ausnahme der Regel $S' \to \lambda$) und $L(G') = L(G)$ erfüllt. Für G' gelten außerdem $\#(N') = \theta(k(G))$ und $\#(P') \leq 4 \cdot k(G') = O(\#(N)k(G))$ nach dem Beweis von Satz 2.24.
Es sei $w = a_1 a_2 \dots a_n$. Dann setzen wir $M_0 = \{S\}$ und

$$M_i = \{A \mid B \to a_i A \text{ für ein } B \in M_{i-1}\}$$

für $1 \leq i \leq n-1$. Die Bestimmung von M_i, $1 \leq i \leq n$, aus M_{i-1} kann in der Zeit $O(\#(P'))$ erfolgen, da einmal die Regeln aus P' durchzumustern sind. Aus der Konstruktion der Mengen folgt sofort, dass $A \in M_i$ genau dann gilt, wenn es die Ableitung

$S \Longrightarrow^* a_1 a_2 \dots a_i A$ gibt. Nun überprüfen wir, ob es ein Nichtterminal A in M_{n-1} gibt, für das eine Regel $A \to a_n$ in P vorhanden ist. Gibt es ein solches Nichtterminal, so existiert die Ableitung

$$S \Longrightarrow^* a_1 a_2 \dots a_{n-1} A \Longrightarrow a_1 a_2 \dots a_{n-1} a_n = w,$$

womit $w \in L(G') = L(G)$ gilt. Ist dagegen kein solches Nichtterminal vorhanden, so kann es keine nach Erzeugung von a_n terminierende Ableitung geben, woraus $w \notin L(G') = L(G)$ folgt. Da die Existenz eines solchen Nichtterminals erneut in der Zeit $O(\#(P'))$ getestet werden kann, erhalten wir als gesamten Zeitbedarf

$$O(\#(P')|w|) = O(k(G) \cdot \#(N) \cdot |w|).$$

□

Übungsaufgaben

1. Bestimmen Sie die von der Grammatik

   $$G = (\{S, X_1, X_2, X_3\}, \{a, b, c\}, P, S)$$

   mit

   a) $P = \{S \to X_1 S X_2, S \to X_3, X_1 \to a X_1 b, X_1 \to \lambda, X_2 \to b X_2 a, X_2 \to \lambda, X_3 \to c\}$

   b) $P = \{S \to a X_1 X_2, a X_1 \to a a X_1 b, X_1 b \to b X_1 X_3, X_3 b \to b X_3, X_3 X_2 \to X_2 c,$
   $X_1 X_2 \to bc\}$

   c) $P = \{S \to a S X_1, S \to a X_2, X_2 X_1 \to b X_2 c, c X_1 \to X_1 c, X_2 \to bc\}$

   erzeugte Sprache.

2. Geben sei die Grammatik

   $$G = (\{S\}, \{a, b\}, \{S \to SS, S \to aaSb, S \to bSaa, S \to \lambda\}, S).$$

   Gilt

   $$L(G) = \{w : w \in T^*, |w|_a = 2 \cdot |w|_b\} \ ?$$

3. Geben Sie für die folgenden Sprachen kontextfreie Grammatiken an:

   a) $\{a^n b^n c^m : n \geq 1, m \geq 3\}$,

   b) $\{a_1^{n_1} a_2^{n_2} \dots a_k^{n_k} b_k^{n_k} b_{k-1}^{n_{k-1}} \dots b_2^{n_2} b_1^{n_1} : n_i \geq 1 \text{ für } 1 \leq i \leq k\}$.

4. Geben Sie eine reguläre Grammatik an, die die Menge aller Wörter $w \in \{a, b, c\}^*$, die genau drei Vorkommen von a und höchstens zwei Vorkommen von c haben, erzeugt.

5. Beweisen Sie, dass jede endliche Sprache regulär ist.

6. Es sei
$$L = \{x_1 x_2 \dots x_n x_n x_{n-1} \dots x_1 : n \geq 1, x_i \in T, 1 \leq i \leq n\}.$$
Beweisen Sie, dass
i) L eine kontextfreie Sprache ist,
ii) L keine reguläre Sprache ist.

7. Es sei
$$L = \{a^{2^n} : n \geq 1\}.$$
Beweisen Sie, dass
i) L eine monotone Sprache ist,
ii) L keine kontextfreie Sprache ist.

8. Beweisen Sie, dass
$$\{ww : w \in T^+\} \in \mathcal{L}(CS)$$
und
$$\{ww : w \in T^+\} \notin \mathcal{L}(CF)$$
gelten.

9. Geben Sie für die Grammatik $G = (N, T, P, S)$ mit
$$\begin{aligned} N &= \{S, A, B\}, \\ T &= \{a, b, c\}, \\ P &= \{S \to cSc, S \to AB, A \to aAb, B \to cBb, A \to ab, B \to \lambda\} \end{aligned}$$
eine Grammatik G' in Chomsky-Normalform mit $L(G') = L(G)$.

10. Eine Grammatik $G = (N, T, P, S)$ heißt *linear*, falls P nur Regeln der Form
$$A \longrightarrow w_1 B w_2 \text{ und } A \longrightarrow w \quad \text{mit} \quad A, B \in N \text{ und } w_1, w_2, w \in T^*$$
enthält.
a) Beweisen Sie, daß es eine lineare Sprache gibt, die nicht regulär ist.
b) Beweisen Sie, daß es eine kontextfreie Sprache gibt, die nicht linear ist.

11. Für eine Sprache L über dem Alphabet V mit $a \in V$ sei
$$L_a = \{w : aw \in L\} \quad \text{und} \quad L^a = \{v : va \in L\}.$$
Zeigen Sie, dass für reguläres L auch L_a und L^a regulär sind.

12. Für eine Sprache L sei L_{ger} die Menge der in L enthaltenen Wörter gerader Länge. Beweisen Sie, dass für reguläres L auch L_{ger} regulär ist.

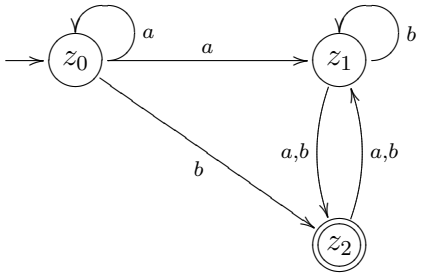
13. Gegeben sei der endliche Automat
$$\mathcal{A} = (\{a, b\}, \{z_0, z_1, z_2\}, z_0, \{z_2\}, \delta)$$

mit

$$\delta(z_0, a) = \delta(z_2, b) = z_0,$$
$$\delta(z_0, b) = \delta(z_1, b) = z_1,$$
$$\delta(z_1, a) = \delta(z_2, a) = z_2.$$

a) Beschreiben Sie $\mathcal{A}$ durch einen Graphen.
b) Welche der Wörter *abaa*, *bbbabb*, *bababa* und *bbbaa* werden von $\mathcal{A}$ akzeptiert?
c) Bestimmen Sie die von $\mathcal{A}$ akzeptierte Sprache.

14. Gegeben sei der Graph



der den Automaten $\mathcal{A}$ beschreibt.
a) Geben Sie alle möglichen Überführungen bei Eingabe von *aaabb* an.
b) Wird *aaabb* von $\mathcal{A}$ akzeptiert.
c) Bestimmen Sie die von $\mathcal{A}$ akzeptierte Sprache.
d) Geben Sie einen deterministischen endlichen Automaten $\mathcal{B}$ mit $T(\mathcal{A}) = T(\mathcal{B})$ an.

15. Konstruieren Sie einen (nichtdeterministischen) endlichen Automaten, der die Sprache aller Wörter über $\{1, 2, 3\}$ akzeptiert, bei denen die Quersumme durch 6 teilbar ist.

16. Konstruieren Sie einen (nichtdeterministischen) endlichen Automaten, der die Sprache aller Wörter über $\{a, b, c\}$ akzeptiert, bei denen jedes Teilwort der Länge 3 mindestens ein a enthält.

17. Gegeben sei ein endlicher Automat $\mathcal{A}$ mit n Zuständen. Beweisen Sie, dass
a) $T(\mathcal{A})$ genau dann nicht leer ist, wenn $T(\mathcal{A})$ ein Wort der Länge m mit $m < n$ enthält,
b) $T(\mathcal{A})$ genau dann unendlich ist, wenn $T(\mathcal{A})$ ein Wort der Länge m mit $n \leq m < 2n$ enthält.

18. Geben Sie für die regulären Sprachen aus den Aufgaben 4, 13, 14 und 16 eine Darstellung mittels Vereinigung, Produkt und KLEENE-Abschluss.

19. Gegeben sei der Kellerautomat

$$\mathcal{M} = (\{z_0, z_1, z_2\}, \{a, b\}, \{X\}, z_0, \{z_0\}, \delta)$$

mit

$$\delta(z_0, b, \#) = \{(z_0, X)\}, \quad \delta(z_0, b, X) = \{(z_0, XX)\},$$
$$\delta(z_0, a, X) = \{(z_1, \lambda)\}, \quad \delta(z_1, a, X) = \{(z_1, \lambda)\},$$
$$\delta(z_1, a, \#) = \{(z_0, \lambda)\}$$

und $\delta(z, x, \gamma) = \{(z_2, \lambda)\}$ in allen anderen Fällen. a) Untersuchen Sie, ob *baabaa*, *babaaaa* und *baaabaa* von M akzeptiert werden.
b) Bestimmen Sie die von $\mathcal{M}$ akzeptierte Wortmenge.

20. Bestimmen Sie für die nachfolgend genannten Sprachen jeweils einen Kellerautomaten, der sie akzeptiert.
a) $\{wdw^R : w \in \{a, b\}^*\}$,
b) die Menge aller Palindrome über $\{a, b\}$,
c) die Menge aller Wörter über $\{a, b\}$, bei denen die Anzahl der Vorkommen von a und b gleich sind,
d) die Menge aller Wörter über $\{(,)\}$, die einer korrekten Klammerung entsprechen.

21. Konstruieren Sie zu der Grammatik

$$G = (\{S, A, B\}, \{a, b, c\}, P, S)$$

mit

$$P = \{S \to aABA, S \to aBB, A \to bA, A \to b, B \to cB, B \to c\}$$

einen Kellerautomaten $\mathcal{M}$ mit $T(\mathcal{M}) = L(G)$.

22. Bestimmen Sie zu regulären Grammatiken G, G_1 und G_2 reguläre Grammatiken H und H' mit $L(H) = \overline{L(G)}$ und $L(H') = L(G_1) \cap L(G_2)$.

23. Beweisen Sie, dass zu kontextfreien Grammatiken G_1 und G_2 kontextfreie Grammatiken G und G' mit $L(G) = L(G_1) \cup L(G_2)$, $L(G') = L(G_1) \cdot L(G_2)$ und $k(G) = \theta(max\{k(G_1), k(G_2)\})$ und $k(G') = \theta(max\{k(G_1), k(G_2)\})$ gibt.

24. Gegeben seien die Grammatik $G = (\{S, A, B, C\}, \{a, b\}, P, S)$ mit

$$P = \{S \to AB, S \to BC, A \to BA, A \to a, B \to CC, B \to b, C \to AB, C \to a\}$$

und die Wörter $w_1 = abbba$, $w_2 = baaba$ und $w_3 = bbbaaa$. Stellen Sie mittels des Cocke-Younger-Kasami-Algorithmus fest, welche der Wörter w_1, w_2, w_3 in $L(G)$ liegen.

Kapitel 3

Elemente der Komplexitätstheorie

3.1 Definitionen und ein Beispiel

Im ersten Abschnitt haben wir gesehen, dass es Probleme gibt, die durch keinen Algorithmus gelöst werden können, zu deren Lösung es also kein für alle Eingaben korrekt arbeitendes Programm gibt. Neben dieser generellen Schranke für Algorithmen gibt es aber auch der Praxis entstammende Grenzen. Stellen wir uns vor, dass durch ein Programm entschieden wird, ob ein Element zu einer Menge gehört[1] und dass bei einem Element der Größe[2] n für die Entscheidung vom Computer $f(n)$ Operationen ausgeführt werden müssen. In der folgenden Tabelle sind einige Zeiten zusammengestellt, die sich bei verschiedenen Funktionen f ergeben, wobei wir annehmen, dass der Computer eine Million Operationen in der Sekunde ausführt. (Wenn wir eine andere Geschwindigkeit des Computers annehmen, z.B. 10^9 Operationen je Sekunde, so ändern sich die Tabellenwerte nur um einen konstanten Faktor. Wir merken aber an, dass aus physikalischen Gründen eine Schranke für die Geschwindigkeit existiert.)

n / f	5	10	50	100
n^2	0,000025 s	0,0001 s	0,0025 s	0,01 s
n^5	0,003125 s	0,1 s	312,5 s	ca. 3 Std.
2^n	0,000032 s	0,001024 s	ca. 36 Jahre	ca. 10^{17} Jahre
n^n	0,003125 s	ca. 3Std.	$> 10^{71}$ Jahre	

Abb. 2.1.

Man sieht an den Werten deutlich, dass bei den Funktionen $f(n) = 2^n$ und $f(n) = n^n$ der Zeitaufwand bei praktisch relevanten Fällen mit Eingaben einer Größe ≥ 50 so groß ist, dass sie nicht in erträglicher Zeit zu einem Ergebnis führen. Hieraus resultiert, dass nur solche Algorithmen von Interesse sind, die nicht zu zeitaufwendig sind. Gleiches gilt für die zur Lösung notwendige Speicherkapazität. Wir formalisieren nun diesen Ansatz zur Messung der Effizienz eines Algorithmus.

[1] Wir erinnern daran, dass jedes entscheidbare Problem auf eine solche Frage zurückgeführt werden kann.

[2] Wir gehen hier nicht näher auf den Begriff der Größe ein. Dies kann z. B. bei Wörtern die Länge, bei Matrizen die Anzahl der Zeilen, bei Polynomen der Grad sein.

Dabei verwenden wir TURING-Maschinen als Beschreibung der Algorithmen. Um eine einheitliche Definition zu erreichen, werden wir TURING-Maschinen mit k-Arbeitsbändern verwenden. Wir beginnen daher mit der Definition solcher Maschinen.

Definition 3.1 *Eine akzeptierende k-Band-*TURING*-Maschine ist ein 6-Tupel*

$$M = (k, X, Z, z_0, Q, F, \delta),$$

wobei $k \geq 1$ eine natürliche Zahl ist, X, Z, z_0, Q und F wie bei einer TURING*-Maschine definiert sind, δ eine Funktion*

$$(Z \setminus Q) \times (X \cup \{*\})^{k+1} \longrightarrow Z \times (X \cup \{*\})^k \times \{R, L, N\}^{k+1}$$

ist und $* \notin X$ gilt.

Die k-Band-TURING-Maschine verfügt über ein Eingabeband, auf dem nur gelesen werden darf, und k Arbeitsbänder mit jeweils einem Lese-Schreibkopf. Wir interpretieren X, Z, z_0, Q und die Elemente aus $\{R, L, N\}$ wie bei einer TURING-Maschine. Falls

$$\delta(z, x_e, x_1, x_2, \ldots, x_k) = (z', y_1, y_2, \ldots, y_k, r_e, r_1, r_2, \ldots, r_k)$$

gilt, so interpretieren wir dies wie folgt: Die Maschine liest im Zustand z auf dem Eingabeband den Buchstaben x_e, auf dem i-ten Arbeitsband den Buchstaben x_i, $1 \leq i \leq k$, geht in den Zustand z' über, schreibt den Buchstaben y_i auf das i-te Arbeitsband, $1 \leq i \leq k$, der Lesekopf des Eingabebandes bewegt sich nach $r_e \in \{R, N, L\}$ und der Kopf des i-ten Arbeitsbandes nach $r_i \in \{R, L, N\}$, $1 \leq i \leq k$.

Definition 3.2 *Sei M eine k-Band-*TURING*-Maschine wie in Definition 3.1.*
Eine Konfiguration von M ist ein $2k + 5$-Tupel

$$(z, w_e, w'_e, w_1, w'_1, w_2, w'_2, \ldots, w_k, w'_k, w_a, w'_a), \tag{3.1}$$

wobei $z \in Z$, $w_e, w'_e \in (X \cup \{*\})^*$ und $w_i, w'_i \in (X \cup \{*\})^*$ für $1 \leq i \leq k$ gelten.
Eine Konfiguration heißt Anfangskonfiguration, falls $z = z_0$, $w_e = w_1 = w_2 = \ldots = w_k = \lambda$ und $w'_a = w'_1 = w'_2 = \ldots = w'_k = *$ gelten.
Eine Konfiguration heißt Endkonfiguration, falls z in Q liegt.

Wir interpretieren eine Konfiguration (3.1) wie folgt: Die Maschine befindet sich im Zustand z, auf dem Eingabeband steht $w_e w'_e$ und der Lesekopf steht über dem ersten Buchstaben von w'_e, für $1 \leq i \leq k$ steht auf dem i-ten Arbeitsband $w_i w'_i$ und steht der Kopf über dem ersten Buchstaben von w'_i.
Bei einer Anfangskonfiguration ist die Maschine im Zustand z_0, auf dem Eingabeband steht w'_e, der Lesekopf befindet sich über dem ersten Buchstaben von w'_e, alle Arbeitsbänder sind leer, aber durch ein angegebenes $*$ wird die Position des Kopfes des Bandes angegeben.

Wir überlassen dem Leser eine formale Definition der Änderung der Konfiguration K_1 in die Konfiguration K_2, die sich aus dem bisher gesagtem in Analogie zu Definition 1.17 ergibt und die wir wieder mit $K_1 \vdash K_2$ bezeichnen.

Definition 3.3 *Sei M eine k-Band-*TURING*-Maschine wie in Definition 3.1. Die von M akzeptierte Sprache besteht aus allen Wörtern $w \in X^*$, die die Anfangskonfiguration*

$$K = (z_0, \lambda, w, \lambda, *, \lambda, *, \ldots, \lambda, *)$$

eine Endkonfiguration

$$K' = (q, w_e, w'_e, w_1, w'_1 w_2, w'_2, \ldots, w_k, w'_k) \text{ mit } q \in F$$

überführen.

Wir definieren nun die Grundbegriffe der Komplexitätstheorie.

Definition 3.4 *Sei $M = (k, X, Z, z_0, Q, \delta, F)$ eine deterministische akzeptierende k-Band-*TURING*-Maschine, die bei jeder Eingabe einen Stoppzustand erreicht. Ferner sei $r = \#(X)$.*
i) Mit $t_M(w)$, $w \in X^*$, bezeichnen wir die Anzahl der (direkten) Überführungsschritte, die M ausführt, um die Anfangskonfiguration $(z_0, \lambda, w, \lambda, *, \lambda, *, \ldots, \lambda, *)$ in die zugehörige Endkonfiguration zu transformieren, und nennen $t_M(w)$ die Zeitkomplexität von w bezüglich M.
ii) Für eine natürliche Zahl n setzen wir

$$t_M(n) = max\{t_M(w) : |w| = n\}$$

und

$$\overline{t_M}(n) = \frac{\sum_{|w|=n} t_M(w)}{r^n}.$$

Die Funktionen t_M und $\overline{t_M}$ von $\mathbf{N}$ in $\mathbf{N}$ heißen Zeitkomplexität des ungünstigsten Falles (worst-case time complexity) und durchschnittliche Zeitkomplexität (average time complexity) von M.

Bei $t_M(n)$ wird die Zeitkomplexität $t_M(w)$ des Wortes w der Länge n genommen, für das M am meisten Schritte benötigt, d.h. die Komplexität des ungünstigsten Wortes wird benutzt. Bei der durchschnittlichen Zeitkomplexität wird zuerst die Summe der Zeitkomplexitäten aller Wörter der Länge n gebildet und dann - wie bei Durchschnittsbildungen üblich - durch die Anzahl r^n aller Wörter der Länge n dividiert.
Wir betrachten die Funktionen t_M und $\overline{t_M}$ als Maße für die Effizienz des durch M realisierten Algorithmus.

Neben dem Zeitaufwand zur Lösung eines Problems ist auch noch der Speicherbedarf eine wesentliche Kenngröße.

Definition 3.5 *Seien M und r wie in Definition 3.4 gegeben.*
i) Mit $s_M(w)$, $w \in X^*$, bezeichnen wir die Anzahl der Zellen auf den Arbeitsbändern, über denen während der Überführung der Anfangskonfiguration $(z_0, \lambda, w, \lambda, *, \lambda, *, \ldots, \lambda, *)$ in die zugehörige Endkonfiguration mindestens einmal der Lese-/Schreibkopf stand. $s_M(w)$ heißt die Raumkomplexität von w auf M.

ii) Für $n \in \mathbf{N}$ setzen wir

$$s_M(n) = max\{s_M(w) : |w| = n\}$$

und

$$\overline{s_M}(n) = \frac{\sum_{|w|=n} s_M(w)}{r^n}.$$

s_M und $\overline{s_M}$ heißen Raumkomplexität des ungünstigsten Falles bzw. durchschnittliche Raumkomplexität von M.

Wir illustrieren die Begriffe nun an einem Beispiel.

Beispiel 3.6 Wir betrachten die Sprache

$$L = \{a^n b^n : n \geq 1\}$$

und die 1-Band-TURING-Maschine M, die wie folgt arbeitet:

- Ist M im Anfangszustand z_0 und liest ein a, so geht sie in den Zustand z_a und schreibt ein a auf das Arbeitsband.
- Ist M im Zustand z_a, so bleibt sie in z_a, solange sie ein a liest und schreibt jedes Mal beim Lesen eines a auch ein a zusätzlich auf das Arbeitsband. Beim Lesen des ersten b in z_a geht M in z_b und löscht ein a auf dem Arbeitsband.
- Den Zustand z_b verändert M nicht, solange ein b gelesen wird, und bei jedem Lesen eines b wird ein a gelöscht. Wird dann ein $*$ gelesen und ist das Arbeitsband leer, so geht M in den akzeptierenden Stopzustand z_{akz}.
- In allen Fällen wechselt M in den ablehnenden Stopzustand z_{abl}.

Hieraus ergeben sich folgende Aussagen zur Komplexität von $w \in \{a, b\}^*$:

w	$t_M(w)$	$s_M(w)$
$a^r b^s$, $r \geq s \geq 1$	$r + s + 1$	r
$a^r b^s$, $s \geq r \geq 1$	$2r + 1$	r
bw', $w' \in \{a, b\}^*$	1	0
$a^r b^s a w''$, $r \geq 1, s \geq 1, w'' \in \{a, b\}^*$	$\min\{r + s + 1, 2r + 1\}$	r

Daher gelten stets $|w|+1 \geq \min\{r+s+1, 2r+1\}$ und $|w| \geq r$. Ferner gelten $t_M(a^n) = n+1$ und $s_M(a^n) = n$. Somit erhalten wir

$$t_M(n) = n + 1 \qquad \text{und} \qquad s_M(n) = n$$

als Zeit- bzw. Raumkomplexität des schlechtesten Falles. In beiden Komplexitätsmaßen erhalten wir lineare Funktionen als Komplexitäten des schlechtesten Falles.
Wir betrachten nun die durchschnittliche Raumkomplexität. Dazu bemerken wir zunächst, dass jedes Wort der Länge n entweder a^n oder von der Form $a^r b w''$ mit $r \geq 0$ und $w'' \in \{a, b\}^*$ ist. Dann gelten

$$s_M(a^n) = n \quad \text{und} \qquad s_M(a^r b w'') = \begin{cases} r & r \geq 1 \\ 0 & r = 0 \end{cases}.$$

Ferner gibt es genau 2^{n-r-1} verschiedene Wörter w'' der Länge $n-r-1$, womit sich

$$\overline{s_M}(n) = \frac{n + \sum_{r=1}^{n-1} r 2^{n-r-1}}{2^n} = \frac{n + (2^n - n - 1)}{2^n} = 1 - \frac{1}{2^n}$$

ergibt. Die durchschnittliche Raumkomplexität von M ist also durch eine Konstante beschränkt.
Ohne Beweis merken wir an, dass dies auch für die durchschnittliche Zeitkomplexität von M gilt.

Wir können zur Entscheidung von L aber auch die 1-Band-TURING-Maschine M' benutzen, die sich von M nur dadurch unterscheidet, dass sie zuerst 0 auf das Arbeitsband schreibt und dann bei Lesen von a in z_a bzw. z_0 die Zahl auf dem Arbeitsband um Eins erhöht und bei Lesen von b in z_b um Eins erniedrigt wird.
Die Komplexitäten ändern sich dann wie folgt: Die Länge des Wortes auf dem Eingabeband ist bei binärer Zahlendarstellung dann durch $\log_2(n)$ beschränkt (und bei Verwendung einer anderen Basis zur Darstellung wird dieser Wert nur um einen konstanten Faktor verkleinert). Somit gilt unter Verwendung der Landau-Symbole

$$s_{M'}(n) = O(\log_2(n)).$$

Da die Addition von 1 unter Umständen mehrere Schritte erfordert, ist die Betrachtung der Zeit im schlechtesten Fall etwas komplizierter. Wenn wir bei der Addition wie in Beispiel 1.19 vorgehen, so ergibt sich für die Anzahl der Schritte bei der Addition von 2^k Einsen zu 0 die Rekursion

$$t_{M'}(2^k) = 2 \cdot t_{M'}(2^{k-1}) + 2\log_2(n),$$

woraus letztlich

$$t_{M'}(n) = O(n)$$

resultiert.
Während wir hinsichtlich der Raumkomplexität im schlechtesten Fall also bei M' gegenüber M eine deutliche größenordnungsmäßige Verbesserung konstatieren können, ist für die Zeitkomplexität im schlechtesten Fall in beiden Fällen Linearität vorhanden (jedoch ist der konstante Koeffizient bei n bei M kleiner).
Es erhebt sich nun die Frage, ob es eine noch bessere k-Band-TURING-Maschine zur Berechnung von M gibt. Wir wollen nun zeigen, dass dies hinsichtlich der Raumkomplexität im schlechtesten Fall nicht der Fall ist, genauer gesagt wir beweisen die folgende Aussage:
*Für jede k-Band-*TURING-*Maschine M'', die L entscheidet, gilt $s_{M''}(n) = O(\log_2(n))$.*
Wir bemerken zuerst, dass wir uns auf 1-Band-TURING-Maschinen beschränken können. Dies folgt daraus, dass wir statt k Bändern ein Band mit k Spuren betrachten können und jeweils der Reihe nach die Spuren in Analogie zu den Bändern ändern. Dies erfordert jeweils ein Suchen des (markierten) Symbols der Spur über dem der Kopf gerade steht und damit einen zusätzlichen Zeitaufwand, aber der Raumbedarf wird dadurch nicht größer. Vielmehr ist nun der Raumbedarf durch den maximalen Raum auf einem der Bänder gegeben, der aber (da die Zahl der Bänder für eine Maschine fest ist) nur um einen konstanten Faktor kleiner ist, als der Platzbedarf auf allen Bändern.

Wir nehmen erst einmal an, dass sich der Lesekopf des Eingabebandes stets über einer Zelle steht, in der sich ein Buchstabe des Eingabeworts befindet.
Wir bezeichnen mit $U(n)$ die Menge der möglichen Teilkonfigurationen, die aus dem Tripel (z, k, w) bestehen, wobei z den Zustand, w das Wort auf dem Arbeitsband und k die Position des Kopfes auf dem Arbeitsband (d.h. der Kopf steht über dem k-ten Buchstaben von w) angeben, und die bei Eingabe eines Wortes der Länge n erreicht werden können. Dann gibt es höchstens $s_{M''}(n)$ Positionen für den Kopf und höchstens $2^{s_{M''}(n)}$ verschiedene Wörter auf dem Band bei einer Eingabe der Länge n. Damit gilt

$$\#(U(n)) \leq \#(Z) \cdot s_{M''}(n) \cdot 2^{s_{M''}(n)}.$$

Durch Logarithmieren gewinnen wir

$$\log_2(\#(U(n))) \leq \log_2(\#(Z)) + \log_2(s_{M''}(n)) + s_{M''}(n).$$

Wir nehmen nun an, dass

$$s_{M''}(n) = o(\log_2(n))$$

gilt, womit aus der vorstehenden Ungleichung

$$\log_2(\#(U(n))) \leq s_{M''}(n) = o(\log_2(n)) < \log_2\left(\frac{n}{2}\right)$$

für hinreichend großes n folgt. Dies impliziert, dass $U(n)$ für hinreichend großes n weniger als $n/2$ Elemente enthält.
Wir betrachten die Arbeit von M'' auf $a^n b^n$ mit hinreichend großem n.
Falls M'' das Wort $a^n b^n$ bereits akzeptiert, ohne ein b zu lesen, so wird auch $a^n b^{n+1}$ akzeptiert. Dies ist aber ein Widerspruch zur Definition von L.
Daher muss M'' also mindestens ein b von der Eingabe $a^n b^n$ lesen und somit mindestens jedes a. Mit u_i, $1 \leq i \leq 2n$, bezeichnen wir das Element von $U(2n)$, das vorliegt, wenn das erste Mal der i-te Buchstabe von $a^n b^n$ gelesen wird. Da $U(2n)$ weniger als n Elemente enthält, muss es Zahlen i und j mit $1 \leq i < j \leq n$ derart geben, dass $u_i = u_j$ gilt.
Wir betrachten nun die Eingabe $a^{n+n!}b^n$. Sei v_s, $1 \leq s \leq n + n!$, das Element von $U(2n + n!)$, das beim erstmaligen Lesen des s-ten Buchstaben von $a^{n+n!}b^n$ vorliegt. Dann gilt $u_i = v_i$ und $u_j = v_j$, da in beiden Fällen ausgehend von der gleichen Ausgangssituation die gleichen Elemente auf dem Eingabeband gelesen werden. Ferner folgt aus $v_k = v_t$ auch $v_{k+1} = v_{t+1}$ für $k, t \leq n + n!$, da ausgehend von gleichen Konfiguration auf dem Arbeitsband nur a's gelesen werden. Damit gilt

$$u_i = v_i = u_j = v_j = v_{i+(j-i)} = v_{i+2(j-i)} = \ldots = v_{i+r(j-i)}$$

mit $r = \frac{n!}{j-i}$. Wegen $i + r(j - i) = i + n!$ erhalten wir $u_i = v_{i+n!}$. Daraus ergibt sich

$$u_i = v_{i+n!}, u_{i+1} = v_{i+1+n!}, \ldots, u_n = v_{n+n!}, \ldots u_{2n} = v_{2n+n!}\,.$$

Dies impliziert, dass M'' auch die Eingabe $a^{n+n!}b^n$ akzeptiert, womit erneut ein Widerspruch zur Definition von L gegeben ist.
Daher muss unsere (einzige) Annahme, nämlich dass die Raumkomplexität von M'' größenordnungsmäßig kleiner als $\log_2(n)$ ist, falsch sein.

Sollte sich der Eingabekopf nicht immer über einer Zelle befinden, in der ein Buchstabe des Eingabeworts steht, so gibt es eine natürliche Zahl h so, dass sich M'' für jedes $i \geq 1$ nach dem Lesen des i-ten Buchstaben von $a^n b^n$ nur noch maximal h Zellen nach links bewegt. Wäre dies nämlich nicht der Fall, so würde es öfter als $\#(U(2n))$ mal das erste a lesen. Dies würde implizieren, dass zweimal das gleiche Element von $U(2n)$ beim Lesen des ersten Buchstaben vorliegt. Damit würde sich eine Schleife ergeben, die dazu führt, dass $a^n b^n$ nicht akzeptiert wird.
Nun können wir obigen Beweis dahingehend modifizieren, dass wir jeweils die Anzahl der Buchstaben um h erhöhen, da nach dem Lesen des $h + i$-ten Buchstaben nur Zellen betreten werden, in denen Buchstaben des Eingabewortes stehen.

Wir haben die Komplexitäten bisher anhand der k-Band-TURING-Maschine eingeführt. Für die TURING-Maschine (ohne Arbeitsbänder und ohne separatem Ausgabeband) lässt sich die Zeitkomplexität in völliger Analogie definieren. Dagegen ist die Definition der Raumkomplexität $s_M(w)$ für eine TURING-Maschine M und ein Eingabewort w etwas problematisch, da zum einen auf dem Band stets schon $|w|$ Zellen beschriftet sind und zum anderen in der Regel die Eingabe vollständig gelesen werden muss, wodurch $s_M(w) \geq |w|$ als notwendig erscheint. Dies würde logarithmische Komplexität wie im obigen Beispiel unmöglich machen. Wir diskutieren daher für TURING-Maschinen nur die Zeitkomplexität.
Der folgende Satz gibt einen Zusammenhang zwischen den Komplexitäten der verschiedenen Varianten von Maschinen.

Satz 3.7 *Zu jeder deterministischen akzeptierenden k-Band-*TURING*-Maschine M, die auf jeder Eingabe stoppt, gibt es eine deterministische akzeptierende* TURING*-Maschine M' derart, dass*

$$T(M') = T(M) \quad \textit{und} \quad t_{M'}(n) = O((t_M(n))^2)$$

gelten und M' auf jeder Eingabe stoppt.

Beweis. Wir verwenden die Simulation von M durch M' in der Weise, dass wir alle Bänder zu einem Band zusammenfassen, in dessen Zellen dann $(k+1)$-Tupel von Bandsymbolen stehen. Die Simulation wird wie folgt vorgenommen. Durch das Koppeln von Lesesymbol und Zustand auf einem Band von M, d.h. statt x steht (x, z) in der Komponente des entsprechendes Bandes, wird die Stelle, wo sich der Kopf des Bandes befindet markiert. Die Simulation eines Schrittes von M besteht nun darin, dass der Kopf von M' von Beginn des beschriebenen Teils mehrfach über das Eingabeband von M' läuft und dabei der Reihe die Position des Lesekopfes über dem Eingabeband, die die Inhalte der Arbeitsbänder entsprechend der Arbeitsweise von M ändert. Jede Änderung bei M', die einer Änderung eines Bandinhaltes entspricht erfordert nur eine endliche Anzahl von Schritten. Ferner kann bei jedem Schritt von M der Inhalt eines Arbeitsbandes höchstens um 1 vergrößert werden, so dass jedes Band von M höchstens ein Wort der Länge $t_M(n)$ enthält. Damit sind von M' in jedem Simulationsschritt höchstens $2(k+1)t_M(n)+ck$ Schritte erforderlich, wobei c eine Konstante ist. Hieraus folgt die Behauptung, da $t_M(n)$ Schritte zu simulieren sind. □

Ohne Beweis geben wir das folgende Resultat, das zeigt, dass es Funktionen gibt, für deren Berechnung ein beliebig großer vorgegebener Zeitaufwand erforderlich ist.

Satz 3.8 *Zu jeder Funktion g von $\mathbf{N}$ in $\mathbf{N}$ gibt es eine Sprache L derart, dass für jede* TURING*-Maschine M, die L entscheidet,*

$$t_M(n) \geq g(n)$$

gilt. □

Um zu verdeutlichen, wie katastrophal die Aussage des Satzes 3.8 ist, betrachten wir die durch

$$g(0) = 2 \quad \text{und} \quad g(n+1) = g(n)^{g(n)}$$

gegebene Funktion g. Wir erhalten

$$f(1) = 4, \quad f(2) = 256, \quad f(3) = 256^{256} \approx 3 \cdot 10^{616},$$

d.h. es gibt eine Funktion, deren Berechnung auf einer beliebigen Maschine bereits auf Eingaben der Länge 3 mindestens $3 \cdot 10^{616}$ Schritte erfordert und damit praktisch unlösbar ist.

Da in den meisten Fällen von praktischer Bedeutung die Funktion $t_M(n)$ nicht genau bestimmt werden kann und man sich daher mit Abschätzungen zufrieden geben muss, führen wir folgende Sprechweisen ein.

Definition 3.9 *Es seien $t : \mathbf{N} \to \mathbf{N}$ eine Funktion, $f : X^* \to X^*$ eine* TURING*-berechenbare Funktion und $M = (X', Z, z_0, Q, \delta)$ eine deterministische* TURING*-Maschine mit $X \subseteq X'$ und $f_M = f$. Wir sagen, dass M die Funktion f in der Zeit t berechnet, wenn M für jedes Wort w aus dem Definitionsbereich von f nach höchstens $t(|w|)$ Überführungsschritten einen Stopzustand erreicht.*

Definition 3.10 *Es seien $t : \mathbf{N} \to \mathbf{N}$ eine Funktion und $L \subset X^*$ eine rekursive Sprache und $M = (X', Z, z_0, Q, \delta, F)$ eine akzeptierende deterministische* TURING*-Maschine mit $X \subset X'$ und $L = T(M)$. Wir sagen, dass M die Sprache L in der Zeit t entscheidet, wenn M für jedes Wort $w \in X^*$ nach höchstens $t(|w|)$ Überführungsschritten einen Stopzustand erreicht.*

Bisher haben wir nur deterministische TURING-Maschinen betrachtet. Auf nichtdeterministische TURING-Maschinen lassen sich die Begriffe nicht so einfach übertragen. Zuerst erinnern wir daran, dass bei Akzeptanz von w durch die nichtdeterministische TURING-Maschine M mindestens einmal bei Abarbeitung von w auf M ein akzeptierenden Zustand erreicht wird, bei anderen Abarbeitungen aber sowohl ablehnende als auch akzeptierende Zustände erreicht werden können. Daher ist $t_M(w)$ nicht eindeutig definierbar. Dies legt es nahe, nur eine Übertragung von Definition 3.10 vorzunehmen. Da auch bei Erreichen eines Stopzustandes bei einer Abarbeitung, bei einer nichtdeterministischen TURING-Maschine die Möglichkeit besteht, dass bei einer anderen Abarbeitung kein Stopzustand erreicht wird, ist es naheliegend, statt der Entscheidbarkeit einer Menge nur die Akzeptanz der Menge zu verlangen. Dies führt zu folgender Definition.

Definition 3.11 *Es seien $t : \mathbf{N} \to \mathbf{N}$ eine Funktion und $L \subset X^*$ eine rekursiv-aufzählbare Sprache und $M = (X', Z, z_0, Q, \delta, F)$ eine akzeptierende (deterministische oder nichtdeterministische)* TURING*-Maschine mit $X \subset X'$ und $L = T(M)$. Wir sagen, dass M die Sprache L in der Zeit t akzeptiert, wenn M für jedes Wort $w \in L$ nach höchstens $t(|w|)$ Überführungsschritten einen akzeptierenden Stopzustand erreicht.*

3.2 Nichtdeterminismus und das P-NP-Problem

Wir betrachten einführend das Erfüllungsproblem SAT, das der Illustration der Problematik dieses Abschnitts dienen soll, aber auch von großer theoretischer Bedeutung dafür ist.
Unter einer Disjunktion oder Alternative in n Booleschen Variablen (die nur mit den Wahrheitswerten 1 für *wahr* und 0 für *falsch* belegt werden können) verstehen wir einen logischen Ausdruck $E(x_1, x_2, ..., x_n)$ der Form

$$E(x_1, x_2, ..., x_n) = x_{i_1}^{\sigma_{i_1}} \vee x_{i_2}^{\sigma_{i_2}} \vee ... \vee x_{i_r}^{\sigma_{i_r}},$$

wobei $r \geq 1$, $i_j \in \{1, 2, \ldots, n\}$ und $\sigma_{i_j} \in \{0, 1\}$ für $1 \leq j \leq r$ gelten, x^1 die Identität und x^0 die Negation sind.

Problem:	SAT
Gegeben:	n Boolesche Variable $x_1, x_2, ..., x_n$ und m Alternativen $E_i(x_1, x_2, ..., x_n)$, $1 \leq i \leq m$.
Frage:	Gibt es eine Belegung $b : x_i \to a_i \in \{0, 1\}$ der Variablen derart, dass $E_j(a_1, a_2, ..., a_n)$ den Wert 1 für $1 \leq j \leq m$ annimmt, d.h. dass alle Alternativen bei b wahr werden.

Zur Lösung von SAT gibt es offenbar folgenden naheliegenden Algorithmus. Wir erzeugen alle 2^n möglichen Belegungen der logischen Variablen $x_1, x_2, ..., x_n$ und testen für jede Belegung, ob alle Alternativen auf dieser Belegung den Wert 1 annehmen. Da das Testen einer Belegung auf einer Disjunktion höchstens die Berechnung von n Negationen und $n - 1$ zweistelligen Alternativen erfordert, ergibt sich für den Gesamtaufwand die obere Schranke $(2n - 1) \cdot m \cdot 2^n$. Andererseits ist für diesen Algorithmus eine untere Schranke durch die Zahl 2^n der möglichen Belegungen gegeben. Damit gilt für diesen naheliegenden Algorithmus A

$$2^n \leq t_A(n) \leq (2n - 1) \cdot m \cdot 2^n.$$

Das exponentielle Wachstum der Zeitkomplexität von A zeigt, dass dieser Algorithmus praktisch für große Werte von n nicht verwendbar ist. (Wir werden im Folgenden zeigen, dass alle bisher bekannten Algorithmen zur Lösung von SAT ebenfalls exponentielles Verhalten der Zeitkomplexität aufweisen.) Offenbar ergibt sich der exponentielle Charakter von t_A aus der Tatsache, dass wir der Reihe nach - also sequentiell - die möglichen Belegungen durchtesten. Eine Verbesserung ist daher zu erwarten, wenn wir das Überprüfen der Werte der Belegungen auf den Disjunktionen „gleichzeitig“ („parallel“) durchführen könnten. Beim Algorithmenbegriff auf der Basis von Turing-Maschinen ist dies nicht möglich, weil die Überführungsfunktion δ eine Funktion auf der Menge der Konfigurationen erzeugt.
Daher ist es naheliegend, auch in diesem Zusammenhang nichtdeterministische Maschinen zu betrachten. Diese könnten nichtdeterministisch in n Schritten alle mögliche Belegungen erstellen (wir haben nur nichtdeterministisch für jede Variable die Belegung 1 oder 0 zu wählen) und können dann ebenfalls in $(2n - 1)m$ Schritten die Belegung testen. Damit ergibt sich höchstens die Komplexität $n + (2n - 1)m$.

Aus Satz 2.43 wissen wir, dass nichtdeterministische und deterministische Turing-Maschinen die gleiche Mengen von Sprachen akzeptieren. Aufgrund unserer Betrachtungen

zum Erfüllungsproblem SAT ist aber zu vermuten, dass der Aufwand zur Lösung eines Problems beim Übergang zu nichtdeterministischen Algorithmen sinken kann. Wir wollen dies nun für den polynomialen Fall etwas näher untersuchen. Dazu führen wir die folgenden Mengen von Sprachen ein.

Definition 3.12 **P** *sei die Menge aller Sprachen, die von einer deterministischen akzeptierenden* TURING*-Maschinen in polynomialer Zeit entschieden werden können.*
NP *sei die Menge aller Sprachen, die von einer nichtdeterministischen akzeptierenden* TURING*-Maschine in polynomialer Zeit akzeptiert werden können.*

Eine Sprache L liegt also genau dann in **P**, wenn es eine deterministische akzeptierenden TURING-Maschine M und ein Polynom p derart gibt, dass $T(M) = L$ gilt und M die Sprache L in der Zeit p entscheidet. Analog liegt L genau dann in **NP**, wenn es eine nichtdeterministische akzeptierenden TURING-Maschine M und ein Polynom p derart gibt, dass $T(M) = L$ gilt und M die Sprache L in der Zeit p akzeptiert.
Da deterministische TURING-Maschinen als ein Spezialfall der nichtdeterministischen TURING-Maschinen angesehen werden können, erhalten wir

$$\mathbf{P} \subseteq \mathbf{NP}\,.$$

Um zu zeigen, dass **P** echt in **NP** enthalten ist, reicht es ein Beispiel anzugeben, dass in **NP** aber nicht in **P** enthalten ist. Für den Nachweis der Gleichheit der beiden Mengen ist dagegen zu beweisen, dass jede Sprache aus **P** auch in **NP** liegt. Ziel dieses Abschnittes ist es, zu zeigen, dass auch für den Beweis der Gleichheit ein Beispiel ausreicht, da es Sprachen in **NP** mit folgender Eigenschaft gibt: falls diese Sprache in **P** liegt, so gilt **P**=**NP**.

Definition 3.13 *Seien $L_1 \subseteq X_1^*$ und $L_2 \subseteq X_2^*$ zwei Sprachen. Wir sagen, dass L_1 auf L_2 transformierbar ist, falls es eine Funktion τ gibt, die X_1^* auf X_2^* so abbildet, dass $a \in L_1$ genau dann gilt, wenn $\tau(a) \in L_2$ ist.*

Wir wollen diese Definition auch für Probleme angeben. Dazu erinnern wir zuerst daran, dass jedes Problem P durch eine Funktion $f_P : X_1 \times X_2 \times \ldots \times X_n \to \{0,1\}$ repräsentiert werden kann, wobei $f_P(a_1, a_2, \ldots, a_n) = 1$ genau dann gilt, wenn die Antwort auf die hinter dem Problem stehende Frage bei der Belegung der Variablen mit $a_1, a_2, \ldots, a_n$ „wahr“ ist. Im folgenden schreiben wir immer kurz X_P für das Produkt $X_1 \times X_2 \times \ldots \times X_n$ und $\underline{a}$ für $(a_1, \ldots, a_n)$.
Seien P_1 und P_2 zwei Probleme. Wir sagen, dass P_1 auf P_2 transformierbar ist, falls es eine Funktion τ gibt, die X_{P_1} auf X_{P_2} so abbildet, dass $f_{P_1}(\underline{a}) = 1$ genau dann gilt, wenn $f_{P_2}(\tau(\underline{a})) = 1$ ist.

Beispiel 3.14 Es sei $G = (V, E)$ ein Graph. Eine Teilmenge $V' \subseteq V$ heißt *Clique* in G, falls $(v, v') \in E$ für alle paarweise verschiedenen $v, v' \in V'$ gilt, d.h. die Knoten aus V' sind paarweise durch Kanten verbunden. Wir betrachten das *Cliquenproblem*

Gegeben:	Graph $G = (V, E)$, natürliche Zahl $k \geq 1$,
Frage:	Gibt es eine k-elementige Clique in G ?

und zeigen dass SAT auf das Cliquenproblem transformiert werden kann.
Seien die Alternativen

$$A_i(x_1, x_2, \ldots, x_n) = x_{i,1}^{\sigma_{i,1}} \vee x_{i,2}^{\sigma_{i,2}} \vee \ldots x_{i,r_i}^{\sigma_{i,r_i}}, \quad 1 \leq i \leq m,$$

gegegen. Wir konstruieren nun wie folgt den Graphen $G = (V, E)$. Zuerst setzen wir

$$V = \{(A_i, x_{i,j}^{\sigma_{i,j}}) : 1 \leq i \leq m, 1 \leq j \leq r_i\}.$$

Die Knoten (A, x^{σ}) und $(A', x'^{\sigma'})$ werden genau dann durch eine Kante verbunden, wenn $A \neq A', x \neq x'$ oder $A \neq A', x = x', \sigma = \sigma'$ gelten. E sei die Menge aller so konstruierten Kanten. Ferner setzen wir $k = m$.
Wir illustrieren die eben beschriebene Konstruktion durch ein Beispiel. Wir betrachten die Menge der Alternativen

$$A_1 = x \vee y, \quad A_2 = \overline{x} \vee \overline{y} \vee \overline{z}, \quad A_3 = y \vee z. \tag{3.2}$$

Der zugehörige Graph ist in Abbildung 3.1 dargestellt.

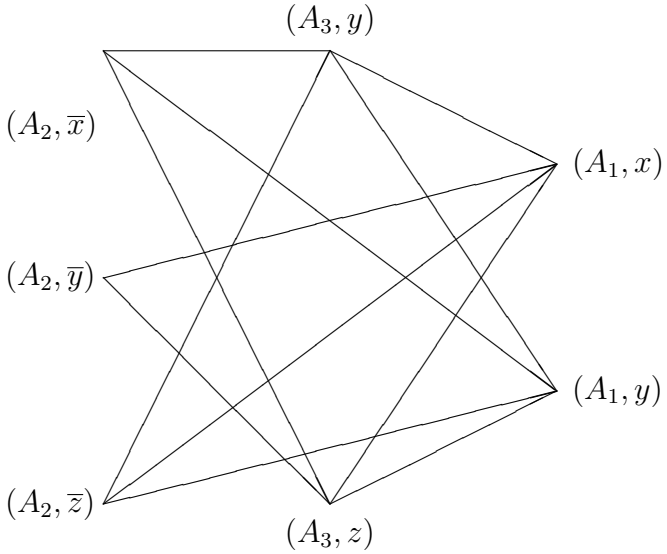


Abbildung 3.1: Graph zu den Alternativen aus (3.1)

Damit haben wir die in Definition 3.13 geforderte Funktion konstruiert, und es bleibt zu zeigen, dass genau dann eine Belegung existiert, für die alle m Alternativen *wahr* werden, wenn es in G eine m-elementige Clique gibt.
Sei zuerst $V' \subseteq V$ eine m-elementige Clique in G. Da nach Konstruktion zwei Knoten, die zur gleichen Alternative A gehören, durch keine Kante verbunden sind, muss V' zu jeder Alternative genau einen Knoten enthalten, d.h.

$$V' = \{(A_1, x_{1,j_1}^{\sigma_{1,j_1}}), (A_2, x_{2,j_2}^{\sigma_{2,j_2}}), \ldots, (A_m, x_{m,j_m}^{\sigma_{m,j_m}}).$$

Gilt für zwei Knoten aus V' die Beziehung $x_{s,j_s} = x_{t,j_t}$, so ist nach Konstruktion von G auch $\sigma_{s,j_s} = \sigma_{t,j_t}$, d.h. jede Variable taucht nur negiert oder nur unnegiert auf. Daher

können wir eine Belegung a_r, $1 \leq r \leq n$, so wählen, dass $a_{i,j_i}^{\sigma_{i,j_i}} = 1$ für $1 \leq i \leq m$ gilt. Damit gilt auch $A_i(a_1, a_2, \ldots, a_n) = 1$ für $1 \leq i \leq m$.
Gilt umgekehrt $A_i(a_1, a_2, \ldots, a_n) = 1$, so gibt es ein j_i, $1 \leq j_i \leq r_i$ mit $x_{i,j_i}^{\sigma_{i,j_i}} = 1$. Es ist nun leicht zu sehen, dass

$$V' = \{(A_1, x_{1,j_1}^{\sigma_{1,j_1}}), (A_2, x_{2,j_2}^{\sigma_{2,j_2}}), \ldots, (A_m, x_{m,j_m}^{\sigma_{m,j_m}})$$

eine m-elementige Clique ist.
In unserem Beispiel entsprechen die Belegungen $(0,1,1)$ bzw. $(1,0,1)$ den Cliquen $\{(A_1, y), (A_2, \overline{x}), (A_3, z)\}$ bzw. $\{(A_1, x), (A_2, \overline{y}), (A_3, z)\}$.

Beispiel 3.15 Wir betrachten das *Problem des Geschäftsreisenden*

Gegeben:	$n \geq 1$, n Städte $C_1, C_2, ..., C_n$, die Entfernungen $d(C_i, C_j)$ zwischen den Städten C_i und C_j für $1 \leq i, j \leq n$, $B \geq 0$
Frage:	Gibt es eine Rundreise $C_{i_1}, C_{i_2}, ..., C_{i_n}$ durch alle Städte, für die $(\sum_{j=1}^{n-1} d(C_{i_j}, C_{i_{j+1}}) + d(C_{i_n}, C_{i_1}) \leq B$ gilt?

und das *Problem der Existenz von* HAMILTON*-Kreisen*

Gegeben:	Graph $G = (V, E)$ mit $\#(V) = n$
Frage:	Enthält G einen HAMILTON-Kreis, d.h. gibt es eine Folge $v_1, v_2, ..., v_n$ von paarweise verschiedenen Knoten des Graphen G so, dass $(v_i, v_{i+1}) \in E$ für $1 \leq i \leq n$ und $(v_n, v_1) \in E$ gelten?

Wir geben nun eine Transformation des Problems der Existenz eines HAMILTON-Kreises auf das Problem des Geschäftsreisenden.
Sei $G = (V, E)$ ein gegebener Graph mit der Knotenmenge

$$V = \{a_1, a_2, ..., a_n\}.$$

Dann setzen wir

$$\begin{aligned} \tau(a_i) &= C_i \quad \text{für} \quad 1 \leq i \leq n, \\ d(C_i, C_j) &= \begin{cases} 1 & (a_i, a_j) \in E \\ 2 & (a_i, a_j) \notin E \end{cases} \end{aligned}$$

und

$$B = n.$$

Ist nun durch die Folge der Knoten $v_1 = a_{i_1}, v_2 = a_{i_2}, ..., v_n = a_{i_n}$
ein HAMILTON-Kreis gegeben, so definiert die Folge $C_{i_1} = \tau(a_{i_1}), C_{i_2} = \tau(a_{i_2}), ..., C_{i_n} = \tau(a_{i_n})$ eine Rundreise durch alle Städte, bei der

$$(\sum_{j=1}^{n-1} d(C_{i_j}, C_{i_{j+1}})) + d(C_{i_n}, C_{i_1}) = (n-1) + 1 = n = B$$

gilt, womit gezeigt ist, dass das durch $n, C_1, ..., C_n$, die Abstandsfunktion d und B gegebene Problem des Geschäftsreisenden eine Lösung besitzt.
Sei umgekehrt für das durch $n, C_1, C_2, ..., C_n$, die obige Abstandsfunktion d und $B = n$ beschriebene Problem des Geschäftsreisenden die Lösung $C_{i_1}, C_{i_2}, ..., C_{i_n}$ gegeben. Wegen $B = n$ müssen $d(C_{i_j}, C_{i_{j+1}}) = 1$ für $1 \leq j \leq n-1$ und $d(C_{i_n}, C_{i_1}) = 1$ gelten. Das besagt aber gerade, dass $a_{i_1}, a_{i_2}, ..., a_{i_n}$ ein HAMILTON-Kreis in G ist.

Definition 3.16 *Wir sagen, dass die Sprache L_1 polynomial auf die Sprache L_2 transformierbar ist, wenn L_1 durch eine Funktion τ auf L_2 transformiert wird, die mit polynomialer Zeitkomplexität berechnet werden kann, d.h. τ wird von einer deterministischen* TURING-MASCHINE *M mit $t_M(n) = \theta(n^r)$ für ein gewisses $r \in \mathbf{N}$ induziert. Wir bezeichnen dies durch $L_1 \alpha L_2$.*

Die Transformation in Beispiel 3.14 ist offenbar polynomial, denn wenn SAT durch n Variablen und m Alternativen gegeben ist, hat der zugehörige Graph höchstens $n \cdot m$ Knoten und höchstens $n \cdot n(m-1)$ Kanten, die alle mittels $nm + n^2(m-1)$-maligen Durchmustern aller Alternativen bestimmt werden können.
Auch die Transformation in Beispiel 3.15 ist polynomial, wie aus der Definition von τ sofort zu sehen ist.

Lemma 3.17 *i) α ist eine transitive Relation auf der Menge der Sprachen.*
ii) Aus $L_2 \in \mathbf{P}$ und $L_1 \alpha L_2$ folgt $L_1 \in \mathbf{P}$.
iii) Aus $L_2 \in \mathbf{NP}$ und $L_1 \alpha L_2$ folgt $L_1 \in \mathbf{NP}$.

Beweis. i) folgt aus der leicht zu verifizierenden Tatsache, dass aus der Berechenbarkeit von f_1 und f_2 in polynomialer Zeit die Berechenbarkeit von $f_1 \circ f_2$ in polynomialer Zeit folgt.
ii) Wir haben zu zeigen, dass $w \in L_1$ in polynomialer Zeit durch eine deterministische TURING-Maschine entscheiden werden kann. Nach Voraussetzung können wir in polynomialer Zeit $\tau(w)$ mittels einer deterministischen TURING-Maschine M_1 bestimmen. Wegen $L_2 \in \mathbf{P}$ kann $\tau(w) \in L_2$ nun in polynomialer Zeit von einer deterministischen Turing-Maschine M_2 entschieden werden. Nach der Definition der Transformierbarkeit gilt $w \in L_1$ genau dann, wenn $\tau(w) \in L_2$ gültig ist. Somit kann $w \in L_1$ durch die deterministische TURING-Maschine, die zuerst wie M_1 und dann wie M_2 arbeitet, in polynomialer Zeit entschieden werden.
iii) wird analog zu ii) bewiesen. □

Definition 3.18 *Eine Sprache L heißt* **NP***-vollständig, wenn folgende Bedingungen erfüllt sind:*
i) $L \in \mathbf{NP}$,
ii) $L' \alpha L$ gilt für jede Sprache $L' \in \mathbf{NP}$.

Satz 3.19 *Die folgenden Aussagen sind gleichwertig:*
i) $\mathbf{P} = \mathbf{NP}$.
ii) $L \in \mathbf{P}$ gilt für jede **NP***-vollständige Sprache L.*
iii) $L \in \mathbf{P}$ gilt für eine **NP***-vollständige Sprache L.*

Beweis. i) $\Rightarrow$ ii). Sei L eine **NP**-vollständige Sprache. Da nach Definition $L \in \mathbf{NP}$ gilt, folgt aus $\mathbf{P} = \mathbf{NP}$ sofort $L \in \mathbf{P}$.
ii) $\Rightarrow$ iii). Diese Implikation ist trivial.
iii) $\Rightarrow$ i). Seien L eine **NP**-vollständige Sprache und L' eine Sprache aus **NP**. Aus der Definition der **NP**-Vollständigkeit folgt $L' \alpha L$. Wegen Lemma 3.17, ii) gilt nun $L' \in \mathbf{P}$ wegen der Voraussetzung $L \in \mathbf{P}$.
Damit ist die Inklusion $\mathbf{NP} \subseteq \mathbf{P}$ bewiesen. Wegen der Gültigkeit der umgekehrten Inklusion folgt die Behauptung. □

Die Bedeutung der **NP**-vollständigen Sprachen besteht nach Satz 3.19 in Folgendem: Können wir für eine **NP**-vollständige Sprache zeigen, dass sie in **P** liegt, so gilt $\mathbf{P} = \mathbf{NP}$; beweisen wir dagegen für eine **NP**-vollständige Sprache, dass sie nicht in **P** ist, so gilt $\mathbf{P} \neq \mathbf{NP}$. **NP**-vollständige Sprachen sind also Scharfrichter für die Frage „**P**=**NP** ?“.

Wir beweisen nun erst einmal die Existenz **NP**-vollständiger Probleme.

Satz 3.20 *SAT ist* **NP***-vollständig.*

Beweis. Entsprechend der Definition **NP**-vollständiger Probleme, müssen wir zum einen zeigen, dass das Erfüllbarkeitsproblem für aussagenlogische Ausdrücke in konjunktiver Normalform in **NP** liegt, und zum anderen haben wir zu zeigen, dass jede Sprache aus **NP** polynomial auf dieses Erfüllbarkeitsproblem transformierbar ist.

$SAT \in \mathbf{NP}$ haben wir bereits informell bewiesen. Ein formaler Beweis bleibt dem Leser überlassen.

Es sei L eine beliebige Sprache aus **NP**. Dann gibt es eine nichtdeterministische Turing-Maschine $M = (X, Z, z_0, Q, \tau)$, die L in polynomialer Zeit akzeptiert, die also für eine Eingabe $w \in L$ höchstens $p(|w|)$ Schritte benötigt, wobei p ein Polynom ist. Es seien $X = \{a_1, a_2, \ldots, a_r\}$, $w = a_{i_1} a_{i_2} \ldots a_{i_n}$, $* = a_0$, $Z = \{z_0, z_1, \ldots, z_m\}$ und ohne Beschränkung der Allgemeinheit $Q = \{z_1\}$. Ferner sei

$$q = \max\{\#\tau(z, a) \mid z \in Z,\ a \in X\}\,.$$

Wir nummerieren die Zellen des Bandes mit ganzen Zahlen in der Weise, dass die Zelle mit der Nummer 1 zu Beginn der Arbeit den ersten Buchstaben von w enthält und setzen nach rechts (bzw. links) durch Addition (bzw. Subtraktion) von 1 die Nummerierung fort. Setzen wir noch $t = p(|w|) + 1$, so kann der Kopf während der Arbeit von M nur über den Zellen stehen, die mit einer Zahl k, $-t \leq k \leq t$, nummeriert sind.
Wir definieren nun einen aussagenlogischen Ausdruck, der die Arbeit von M auf der Eingabe w beschreibt. Als Variablen benutzen wir

$$Z_{ij}, 1 \leq i \leq t, 0 \leq j \leq m,$$
$$H_{ik}, 1 \leq i \leq t, -t \leq k \leq t,$$
$$S_{ikl}, 1 \leq i \leq t, -t \leq k \leq t, 0 \leq l \leq r,$$

die folgende Bedeutung haben:

- Z_{ij} nimmt genau dann den Wert 1 an, wenn M zur Zeit i im Zustand z_j ist,

- H_{ik} nimmt genau dann den Wert 1 an, wenn der Kopf von M zur Zeit i über der Zelle k steht, und

- S_{ikl} nimmt genau dann den Wert 1 an, wenn zur Zeit i in der Zelle k auf dem Band von M der Buchstabe a_l steht.

Wir betrachten die folgenden Ausdrücke:

(1) $(Z_{i0} \vee Z_{i1} \vee \cdots \vee Z_{im})$ für $1 \leq i \leq t$,
(2) $(\neg Z_{ij} \vee \neg Z_{ij'})$ für $1 \leq i \leq t$, $0 \leq j < j' \leq m$,
(3) $(H_{i,-t} \vee H_{i,-t+1} \vee \cdots \vee H_{it})$ für $1 \leq i \leq t$,
(4) $(\neg H_{ik} \vee \neg H_{ik'})$ für $1 \leq i \leq t$, $-t \leq k < k' \leq t$,
(5) $(S_{ik0} \vee S_{ik1} \vee \cdots \vee S_{ikr})$ für $1 \leq i \leq t$, $-t \leq k \leq t$,
(6) $(\neg S_{ikl} \vee \neg S_{ikl'})$ für $1 \leq i \leq t$, $-t \leq k \leq t$, $0 \leq l < l' \leq r$,
(7) Z_{10},
(8) H_{11},
(9) $S_{11i_1}, S_{12i_2}, \ldots, S_{1ni_n}$ und S_{1k0} für $-t \leq k \leq t$, $k \notin \{1, 2, \ldots, n\}$,
(10) Z_{t1},
(11) $(\neg Z_{ij} \vee \neg H_{ik} \vee \neg S_{ikl} \vee (Z_{i+1,j_1} \wedge H_{i+1,k_1} \wedge S_{i+1,k,l_1}) \vee \ldots \vee (Z_{i+1,j_u} \wedge H_{i+1,k_u} \wedge S_{i+1,k,l_u}))$
für $1 \leq i \leq t-1$, $0 \leq j \neq 1 \leq m$, $-t \leq k \leq t$, $0 \leq l \leq r$,
$\delta(z_j, a_l) = \{(z_{j_1}, a_{l_1}, d_1), (z_{j_2}, a_{l_2}, d_2), \ldots, (z_{j_u}, a_{l_u}, d_u)$,
$k_p = k-1$ für $d_p = L$, $k_p = k$ für $d_p = N$, $k_p = k+1$ für $d_p = R$, $1 \leq p \leq u$,
(12) $(\neg Z_{i1} \vee \neg H_{ik} \vee \neg S_{ikl} \vee (Z_{i+1,1} \wedge H_{i+1,k} \wedge S_{i+1,k,l}))$
für $1 \leq i \leq t-1$, $-t \leq k \leq t$, $0 \leq l \leq r$,
(13) $(\neg S_{ikl} \vee \neg H_{ik'} \vee S_{i+1,k,l})$ für $1 \leq i \leq t-1$, $-t \leq k \neq k' \leq t$, $0 \leq l \leq r$.

Durch diese Wahl der Ausdrücke wird folgendes erreicht: (1) nimmt genau dann den Wert 1 an, wenn mindestens eine der Variablen Z_{ij}, $0 \leq j \leq m$, den Wert 1 annimmt, d.h. wenn sich die Maschine M zur Zeit i in mindestens einem Zustand z_j befindet. Die Alternative (2) nimmt genau dann den Wert 0 an, wenn Z_{ij} und $Z_{ij'}$ den Wert 1 annehmen, d.h. wenn sich M zur Zeit i sowohl im Zustand z_j als auch im Zustand $z_{j'}$ befindet. Die Alternativen (1) und (2) sind also genau dann beide wahr, wenn sich M zur Zeit i in genau einem Zustand befindet.
Analog sichern (3) und (4), dass sich der Kopf von M zur Zeit i über genau einer Zelle befindet, und (5) und (6) bedeuten, dass in der Zelle k zur Zeit i genau ein Buchstabe steht.
Die Alternativen (7), (8) und (9) beschreiben die Anfangskonfiguration; (10) sichert das Erreichen einer Endkonfiguration.
Der Ausdruck (11) beschreibt das Verhalten von M, wenn noch kein Endzustand erreicht ist. Bei Wahrheit von Z_{ij}, H_{ik} und S_{ikl} muss eine der Konjunktionen $(Z_{i+1,j_p} \wedge H_{i+1,k_p} \wedge S_{i+1,k,l_p})$, $1 \leq p \leq u$, wahr werden. Wenn M zur Zeit i im Zustand z_j ist und das Symbol a_l in Zelle k liest, dann schreibt M das Symbol a_{l_p} in die Zelle k, geht in den Zustand z_{j_p} und bewegt den Kopf zur Zelle k_p. Folglich wird eine der möglichen Aktionen von M ausgeführt.
Analog sichert (12), dass bei Erreichen eines Endzustandes keine Änderung mehr vorgenommen wird, d.h. wir setzen die Arbeit von M im Unterschied zur formalen Definition auch bei Erreichen des Endzustandes fort, um den Zeitpunkt t zu erreichen. Die Alterna-

tive (13) besagt, dass der Inhalt der Zelle nicht verändert wird, wenn sich der Kopf nicht über der Zelle befindet.
Es sei B die Konjunktion aller Ausdrücke aus (1) – (15). Aus obigen Bemerkungen folgt sofort, dass es genau dann eine Belegung der Variablen gibt, bei der alle Ausdrücke (1) – (15) den Wert 1 annehmen, wenn die Akzeptanz der Eingabe w höchstens $p(|w|)$ Schritte erfordert. Somit liegt eine Transformation von L auf das Erfüllbarkeitsproblem für aussagenlogische Ausdrücke vor.
Wir haben noch zu zeigen, dass diese Transformation polynomial ist. Dazu reicht es aus, festzustellen, dass der aus M und w konstruierte Ausdruck B höchstens die Länge

$$\begin{aligned}(2m+4)t + 8 \cdot \frac{1}{2}m(m+1)t + (4t+4)t + 8 \cdot \frac{1}{2}(2t+1)2t^2 + (2r+4)(2t+1)t \\ +8 \cdot \frac{1}{2}r(r+1)(2t+1)t + 2 \cdot 1 + 2 \cdot 1 + 2 \cdot (2t+1) + 2 \cdot 1 \\ +(8q+11)(m+1)(2t+1)(t-1)(r+1) + 10(r+1)(2t+1)2t^2 \\ \leq (2r+8q+40)(m^2+1)(r^2+1)2t^2(2t+1)\end{aligned}$$

hat, wobei sich die ersten 10 Summanden aus den Längen der Alternativen der Typen (1) – (10) ergeben, der elfte Summand eine obere Abschätzung der Länge der Ausdrücke aus (11) und (12) ist und der letzte Summand die Länge der Alternativen vom Typ (13) ist (dabei gibt bei jedem Summanden der erste Faktor jeweils die um Eins vergrößerte Länge eines Ausdrucks der Form an, wobei die hinzugefügte Eins das in B dem Ausdruck folgende $\wedge$ erfasst; das Produkt der anderen Faktoren gibt die Anzahl der entsprechende Ausdrücke an). □

Wir haben bereits oben auf die Bedeutung der **NP**-vollständigen Sprachen für die Lösung des Problems „**P**=**NP** ?“ hingewiesen. Daher wollen wir nun eine Reihe von **NP**-vollständigen Sprachen aus verschiedenen Bereichen der Mathematik und Informatik angeben. Auf Beweise werden wir dabei weitgehend verzichten. In den Fällen, wo wir einen Beweis geben werden, wird der folgende Satz angewandt.

Satz 3.21 *Ist die* **NP***-vollständige Sprache L polynomial auf die Sprache L' aus* **NP** *transformierbar, so ist L' auch* **NP***-vollständig.*

Beweis. Für jede Sprache Q aus **NP** gilt $Q \alpha L$. Weiterhin haben wir nach Voraussetzung $L \alpha L'$. Damit folgt $Q \alpha L'$ für alle $Q \in$ **NP**. □

Diese Methode ist also erneut die Reduktion eines Problems auf ein anderes, wobei sich die **NP**-Vollständigkeit überträgt.
Bei den folgenden Beispielen werden wir – der Anschaulichkeit halber – statt Sprachen die zugehörigen Probleme verwenden.

Satz 3.22 *Das Cliquenproblem ist* **NP***-vollständig.*

Beweis. Nach Beispiel 3.14 und der Bemerkung nach Definition 3.16 ist SAT polynomial auf das Cliquenproblem transformierbar. Außerdem ist das Cliquenproblem sicher in **NP**, da wir nichtdeterministisch in polynomialer Zeit eine k-elementige Menge V' von Knoten auswählen und dann in polynomialer Zeit testen können, ob V' eine Clique ist. Nach Satz 3.21 ist das Cliquenproblem damit als **NP**-vollständig nachgewiesen. □

Ohne Beweis geben wir nun die folgende Aussage.

Satz 3.23 *Das Problem der Existenz von* HAMILTON*-Kreisen ist* **NP***-vollständig.* □

Satz 3.24 *Das Problem des Geschäftsreisenden ist* **NP***-vollständig.*

Beweis. Nach dem Beispiel 3.15 ist das Problem der Existenz von HAMILTON-Kreisen auf das Problem des Geschäftsreisenden polynomial transformierbar. Satz 3.24 ist daher nach Satz 3.21 bewiesen, wenn wir gezeigt haben, dass das Problem des Geschäftsreisenden in **NP** liegt. Dies folgt aber leicht, wenn wir nichtdeterministisch alle möglichen Rundreisen erzeugen und dann testen, ob sich für eine Rundreise ein Wert $\leq B$ ergibt, da beide Teilschritte mit polynomialen Aufwand erledigt werden können. □

Wir betrachten noch eine Variante des Problems des Geschäftsreisenden, die ein spezielles diskretes Optimierungsproblem darstellt.

Problem:	Minimale Rundreise
Gegeben:	natürliche Zahl $n \geq 1$,
	Städte $C_1, C_2, \ldots, C_n$ mit den Abständen $d(C_i, C_j)$, $1 \leq i, j \leq n$,
Frage:	Wie groß ist der minimale Wert von $d(C_{i_n}, C_{i_1}) + \Sigma_{j=1}^{n-1} d(C_{i_j}, C_{i_{j+1}})$, wobei das Minimum über alle Permutation von $\{1, 2, \ldots n\}$ zu nehmen ist?

Satz 3.25 *Das Problem der minimalen Rundreise ist* **NP***-vollständig.*

Beweis. Sei

$$m = max\{d(C_i, C_j) : 1 \leq i, j \leq n\}.$$

Dann ist das gesuchte Minimum beim Problem der minimalen Rundreise sicher höchstens $m \cdot (n+1)$. Somit kann das Problem der minimalen Rundreise durch sequentielles Abarbeiten des Problems des Geschäftsreisenden mit den Werten $B_i = i$, $1 \leq i \leq m(n+1)$, gelöst werden.
Umgekehrt liefert die Bestimmung des Minimums auch die Antwort auf die Frage nach einer Rundreise mit einer Länge $\leq B$. □

Satz 3.26 *Das Problem der (Knoten-)Färbbarkeit von Graphen*

Gegeben:	*Graph $G = (V, E)$ und natürliche Zahl $k \geq 3$*
Frage:	*Gibt es eine Färbung der Knoten von G mit k Farben, so dass durch eine Kante verbundene Knoten jeweils verschieden gefärbt sind?*

ist **NP***-vollständig.* □

Für $k = 2$ gibt es eine Lösung des Färbbarkeitproblems mit polynomialem Aufwand.

Satz 3.27 *Das Problem der Teilmengensumme*

Gegeben:	*endliche Menge $A \subseteq \mathbf{N}$ und natürliche Zahl $b \in \mathbf{N}$*
Frage:	*Gibt es eine Teilmenge $A' \subseteq A$ derart, dass $\Sigma_{a \in A'} a = b$ gilt?*

ist **NP***-vollständig.* □

Satz 3.28 *Das Problem der Lösbarkeit diophantischer quadratischer Gleichungen*

Gegeben: *natürliche Zahlen* a, b, c
Frage: *Gibt es eine Lösung von* $ax^2 + by = c$ *in natürlichen Zahlen?*

ist **NP***-vollständig.* □

Wir wollen nun ein Problem aus der Theorie der Datenbanken betrachten, für das wir das CODDsche relationale Datenbankmodell zugrundelegen. Es besteht aus Objekten und zugeordneten Attributwerten. Die Notation erfolgt meist in Form einer Tabelle, in deren erster Spalte die Objekte stehen und in den weiteren Spalten, die den Attributen entsprechen, stehen in der Zeile von einem Objekt die ihm zugeordneten Attributwerte. Die folgende Tabelle gibt ein Beispiel.

Objekt	Name	Vorname	Immatrikulations-nummer	Universität	Fakultät/ Fachbereich
1	Meyer	Heike	12345678	RWTH Aachen	Informatik
2	Schulz	Ulrike	21436587	TU München	Elektrotechn.
3	Müller	Heike	12348765	TU Dresden	Elektrotechn.
4	Muster	Fritz	56781234	TH Darmstadt	Mathematik.
5	Meyer	Ulrich	65874321	TU Berlin	Mathematik
6	Müller	Fritz	87654321	RWTH Aaachen	Informatik

Für das Objekt i und das Attribut A sei der i zugeordnete Attributwert mit $A(i)$ bezeichnet. Wir sagen, dass das Attribut A von den Attributen $B_1, B_2, \ldots, B_k$ abhängig ist, wenn die durch $f(B_1(i), B_2(i), \ldots, B_k(i)) = A(i)$ gegebene Abbildung eine Funktion ist, d.h. wenn der Wert $A(i)$ für jedes i bereits durch die Werte $B_1(i), B_2(i), \ldots, B_k(i)$ eindeutig festgelegt ist. Wir schreiben hierfür $\{B_1, B_2, \ldots, B_k\} \succ A$.

Im obigen Beispiel gelten z.B. {Immatrikulationsnummer} $\succ$ Name und {Name, Vorname} $\succ$ Immatrikulationsnummer, aber nicht {Name} $\succ$ Vorname und nicht {Vorname} $\succ$ Name.

Es sei eine Datenbank mit der Menge H von Attributen gegeben. Eine Teilmenge K von H heißt Schlüssel, falls $K \succ B$ für jedes $B \in H$ gilt.

Satz 3.29 *Das Problem der Existenz von Schlüsseln in einer Datenbank*

Gegeben: *Datenbank mit Menge H von Attributen, natürliche Zahl k*
Frage: *Gibt es einen Schlüssel K für F mit* $\#(K) \leq k$ *?*

ist **NP***-vollständig.* □

Wie in Satz 3.24 kann ausgehend von Satz 3.29 anstelle von Satz 3.25 bewiesen werden, dass auch das Problem der Bestimmung eines minimalen Schlüssels (hinsichtlich der Mächtigkeit) **NP**-vollständig ist.

Das Problem, ob **P**=**NP** gilt, ist bis heute noch ungelöst. Insbesondere gibt es also für alle bekannten **NP**-vollständigen Probleme bis heute keinen deterministischen Algorithmus, der sie in polynomialer Zeit löst, aber es gibt auch kein solches Problem, für das die Nichtexistenz eines polynomialen Algorithmus gezeigt werden konnte. Hat man ein

NP-vollständiges Problem gegeben, ist daher nicht zu erwarten, dass man dafür einen polynomialen Algorithmus findet, und sollte sich mit einem exponentiellen Algorithmus zufriedengeben. Dies wird noch dadurch unterstützt, dass allgemein die Relation **P**≠**NP** vermutet wird.

Die Überlegungen, die wir in diesem Kapitel bezüglich der Zeitkomplexität durchgeführt haben, lassen sich im wesentlichen auch für die Raumkomplexität anstellen.

Übungsaufgaben

1. Gegeben sei der Graph $G = (V, E)$ mit

$$\begin{aligned} V &= \{1,2,3,4,5,6,7,8,9,10,11\}, \\ E &= \{(1,2),(1,3),(1,4),(1,5),(1,6),(1,7),(1,8),(1,9),(1,10),(1,11), \\ &\quad (2,4),(2,10),(3,5),(3,7),(3,9),(3,11),(4,6),(5,7),(5,9),(5,11), \\ &\quad (6,8),(7,9),(7,11),(8,10),(9,11)\}. \end{aligned}$$

   Eine Überdeckung von G ist eine Menge $V' \subseteq V$ derart, dass $\{v, v'\} \cap V' \neq \emptyset$ für alle Kanten $(v, v') \in E$ gilt.
   Bestimmen Sie

   (a) die maximale Zahl k, für die es eine Clique aus k Elementen gibt,

   (b) die minimale Zahl k, für die G k-knotenfärbbar ist,

   (c) die minimale Zahl k, für die eine Überdeckung aus k Elementen existiert.

2. Geben Sie eine Transformation des Cliquenproblems auf das *Überdeckungsproblem*:

   Gegeben: Graph $G = (V, E)$, $k \in \mathbf{N}$,
   Frage : Gibt es eine k-elementige Überdeckung von G ?

   (Die Definition der Überdeckung ist in Übungsaufgabe 3. gegeben.
   Hinweis: Man verwende den Komplementärgraph $G' = (V, E')$ mit $E' = \{(v, v') : (v, v') \notin E\}$.)

3. Beweisen Sie die **NP**-Vollständigkeit von $3-SAT$, das sich von SAT dadurch unterscheidet, dass alle Alternativen die Form $x_i^{\sigma_i} \vee x_j^{\sigma_j} \vee x_k^{\sigma_k}$ für gewisse $1 \leq i < j < k \leq n$ haben.
   (Hinweis: Man ersetze eine beliebige Alternative A unter Einbeziehung von zusätzlichen Variablen durch eine Menge von Alternativen mit jeweils genau drei Variablen, so dass A genau dann *wahr* wird, wenn alle Alternativen aus M *wahr* werden.)

4. Konstruieren Sie entsprechend Beispiel 3.14 den Graphen für die Alternativen

$$x \vee y \vee \overline{z}, \quad \overline{x} \vee \overline{y} \vee z, \quad y \vee \overline{z}.$$

5. Beweisen Sie, dass das Cliquenproblem für festes k in **P** liegt.

6. Beweisen Sie, dass das Problem der Knotenfärbung für $k = 2$ in **P** liegt.