Inhaltsverzeichnis

Kapitel 2

Formale Sprachen und Automaten

2.1 Die Sprachfamilien der Chomsky-Hierarchie

2.1.1 Definition der Sprachfamilien

Im Kapitel 1 haben wir mehrere gleichwertige Definitionen für Algorithmen behandelt. Als Grundlage dienten dabei einmal eine spezielle einfache Programmiersprache, die **LOOP**/ **WHILE**-Programme erzeugt, und ein anderes Mal ein spezieller Typ von Maschinen, die TURING-Maschinen. In diesem Kapitel werden wir uns direkt dem Studium von formalen Sprachen bzw. Automaten als Abstraktionen von Programmier- und natürlichen Sprachen bzw. von Computern und Rechenmaschinen zuwenden.
Wir beginnen dabei mit der Definition eines allgemeinen Typs von formalen Grammatiken und Sprachen und geben dann einige wichtige und interessante Spezialfälle an.

Jede natürliche Sprache basiert auf einer Grammatik, in der die Regeln zusammengestellt sind, nach denen sich syntaktisch richtige Sätze der Sprache bilden lassen. Eine ähnliche Rolle spielen die Handbücher für Programmiersprachen; auch sie enthalten verschiedene Anweisungen und Kommandos, durch deren Anwendung korrekte Programme erzeugt werden.

Die Syntax einer natürlichen Sprachen gibt an, wie ein Satz bzw. Teile eines Satzes aus grammatischen Einheiten aufgebaut werden kann. Wir erwähnen hier beispielhaft die folgenden Konstruktionen.

(Satz) → (Substantivphrase)(Verbphrase)
(Satz) → (Substantivphrase)(Verbphrase)(Objektphrase)
(Substantivphrase) → (Artikel)(Substantiv)
(Verbphrase) → (Verb)(Adverb)

Das erste Konstrukt besagt, dass ein Satz aus einem Substantiv und einem Verb bestehen kann, das zweite entspricht dem vom Englischunterricht her bekannten Aufbau eines Satzes aus Subjekt, Prädikat und Objekt (man sieht, dass für einen Satz verschiedene Zerlegungen in grammatikalische Teile möglich sind). Die beiden letzten Vorschriften sagen, wie eine Substantivphrase bzw. eine Verbphrase weiter zergliedert bzw. wie diese aufgebaut werden können. Weiterhin gibt es eine Zuordnung der Wörter der deutschen Sprache zu Wortarten. Dies kann durch die folgenden Konstruktionen beschrieben werden.

(Substantiv) → Hund
(Substantiv) → Banane
(Artikel) → der
(Artikel) → ein
(Verb) → geht
(Verb) → singt
(Adverb) → langsam

Durch Nacheinanderanwendung der obigen Vorschriften können u. a.

$$\begin{aligned}(\text{Satz}) &\Longrightarrow (\text{Substantivphrase})(\text{Verbphrase})\\ &\Longrightarrow (\text{Substantivphrase})(\text{Verb})(\text{Adverb})\\ &\Longrightarrow (\text{Substantivphrase})\ \text{geht}\ (\text{Adverb})\\ &\Longrightarrow (\text{Substantivphrase})\ \text{geht langsam}\\ &\Longrightarrow (\text{Artikel})(\text{Substantiv})\ \text{geht langsam}\\ &\Longrightarrow \text{der}\ (\text{Substantiv})\ \text{geht langsam}\\ &\Longrightarrow \text{der Hund geht langsam}\end{aligned}$$

und in analoger Weise kann auch

$$(\text{Satz}) \Longrightarrow \ldots \Longrightarrow \text{ein Banane singt langsam}$$

hergeleitet werden. Wir machen darauf aufmerksam, dass der letzte Satz zwar inhaltlich falsch, aber syntaktisch korrekt ist.
Kommen wir nun zu den Programmiersprachen. Hier legt das Programmierhandbuch fest, in welcher Weise das Programm selbst bzw. seine Teilstücke aufgebaut sein können. Als Beispiel geben wir nachfolgend einige Regeln, die sagen, wie Zahlen in einem PASCAL-Programm aussehen können.

(unsigned integer) → (digit) | (digit){digit}
(unsigned real) → (unsigned integer).(digit){digit} | (unsigned integer)E(scale factor)
(scale factor) → (unsigned integer) | (sign) (unsigned integer)
(digit) → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
(sign) → + | –

Hieraus erhalten wir die folgende Sequenz

$$\begin{aligned}(\text{unsigned real}) &\Longrightarrow (\text{unsigned integer})\text{E}(\text{scale factor})\\ &\Longrightarrow (\text{digit})\{\text{digit}\}\text{E}(\text{scale factor})\\ &\Longrightarrow 3\{\text{digit}\}\text{E}(\text{scale factor})\\ &\Longrightarrow 314\text{E}(\text{scale factor})\\ &\Longrightarrow 314\text{E}(\text{sign})(\text{unsigned integer})\\ &\Longrightarrow 314\text{E}–(\text{unsigned integer})\\ &\Longrightarrow 314\text{E}–(\text{digit})\\ &\Longrightarrow 314\text{E}–2\end{aligned}$$

aus der hervorgeht, dass (die Näherung) 3,14 (für π) eine reelle Zahl ist.
Wir stellen folgende Gemeinsamkeiten fest:

- Eigentlich handelt es sich bei den Vorschriften um Ersetzungsregeln. Gewisse Objekte werden durch andere ersetzt.
- Es gibt Objekte, die ersetzt werden (z. B. (Substantivphrase), (unsigned real)), und andere Objekte, die durch die Ersetzungen nicht verändert werden, sondern endgültigen Charakter haben (wie die Wörter der Sprache selbst oder die Ziffern 0,1,2,...,9 und die Zeichen + und –).
- Die Erzeugungen beginnen mit festgelegten Objekten (wie (Satz) oder (program)) und enden, wenn nur noch unveränderliche Objekte vorhanden sind.

Wir werden auf dieser Basis im Folgenden formale Grammatiken und Sprachen definieren. Dabei wollen wir Objekte als Buchstaben eines Alphabets auffassen, und die erzeugten Sätze bzw. Programme bzw. Programmstücke sind dann Wörter über dem Alphabet, das z. B. als Buchstaben alle deutschen Wörter bzw. die Elemente `if`, `while`, Ziffern usw. enthält.
Um die Möglichkeiten zur Wahl von Alphabeten nicht ausufern zu lassen, wollen wir im Folgenden immer annehmen, dass die betrachteten Alphabete (endliche) Teilmengen einer festen abzählbar-unendlichen Menge sind.
Unter einer Sprache über dem Alphabet V verstehen wir im Folgenden stets eine beliebige Teilmenge von V^*. In den folgenden Abschnitten werden verschiedene Möglichkeiten der Beschreibung von (unendlichen) Sprachen durch endliche Objekte untersucht.

Definition 2.1 *Eine Regelgrammatik (oder kurz Grammatik) ist ein Quadrupel*

$$G = (N, T, P, S),$$

wobei

- *N und T endliche, disjunkte Alphabete sind, deren Vereinigung wir mit V bezeichnen,*
- *P eine endliche Teilmenge von $(V^* \setminus T^*) \times V^*$ ist, und*
- *$S \in N$ gilt.*

Dabei ist N das Alphabet der Nichtterminale oder Hilfssymbole (wie (Substantivphrase) oder (unsigned real)) und T das Alphabet der Terminale. Im Folgenden werden wir meist große lateinische Buchstaben zur Bezeichnung der Nichtterminale und kleine lateinische Buchstaben für die Terminale verwenden. Die Elemente aus P heißen Regeln. Meistens werden wir das Paar (α, β) aus P in der Form $\alpha \longrightarrow \beta$ schreiben, da diese Notation der Anwendung von Regeln (in der nächsten Definition) als Ersetzung entspricht. S heißt Axiom oder Startwort (und entspricht (Satz) bzw. (program)).

Definition 2.2 *Es sei $G = (N, T, P, S)$ eine Regelgrammatik wie in Definition 2.1 beschrieben. Wir sagen, dass aus dem Wort $\gamma \in V^+$ das Wort $\gamma' \in V^*$ erzeugt wird, wenn*

$$\gamma = \gamma_1 \alpha \gamma_2,\ \gamma' = \gamma_1 \beta \gamma_2,\ \alpha \longrightarrow \beta \in P$$

für gewisse $\gamma_1, \gamma_2 \in V^*$ gelten. Wir schreiben dann

$$\gamma \Longrightarrow \gamma'.$$

Entsprechend Definition 2.2 entsteht γ' aus γ, indem ein Teilwort α in γ durch β ersetzt wird, wenn eine Regel $\alpha \longrightarrow \beta$ in P existiert. Die Regeln geben also an, welche lokalen Ersetzungen ausgeführt werden können, um aus einem Wort ein neues zu erzeugen.
Die Anwendung einer Regel nennen wir auch einen Ableitungsschritt oder sagen, dass γ' aus γ direkt abgeleitet oder generiert wird. Falls die bei der Erzeugung verwendete Regel $p = \alpha \longrightarrow \beta$ betont werden soll, so schreiben wir $\gamma \Longrightarrow_p \gamma'$. Durch $\Longrightarrow$ wird offenbar eine Relation, d.h. eine Teilmenge von $V^+ \times V^*$, definiert. Wie üblich kann hiervon der reflexive und transitive Abschluss $\Longrightarrow^*$ gebildet werden, d.h. es gilt

$$\gamma \Longrightarrow^* \gamma'$$

genau dann, wenn es eine natürliche Zahl $n \geq 0$ und Wörter $\delta_0, \delta_1, \delta_2, \ldots, \delta_{n-1}, \delta_n$ mit

$$\gamma = \delta_0 \Longrightarrow \delta_1 \Longrightarrow \delta_2 \Longrightarrow \ldots \Longrightarrow \delta_{n-1} \Longrightarrow \delta_n = \gamma'$$

gibt (im Fall $n = 0$ gilt $\gamma = \gamma'$, und im Fall $n = 1$ haben wir $\gamma \Longrightarrow \gamma'$). Somit gilt $\gamma \Longrightarrow^* \gamma'$ genau dann, wenn γ' durch iterierte Anwendung von (nicht notwendigerweise gleichen) Regeln aus γ entsteht. Gilt $\gamma \Longrightarrow^* \gamma'$, so sagen wir auch γ' ist aus γ (in mehreren Schritten) ableitbar oder erzeugbar.
Ein Wort $w \in V^*$ heißt *Satzform* von G, wenn $S \Longrightarrow^* w$ gilt, d.h. wenn w aus S erzeugt werden kann.

Definition 2.3 *Für eine Grammatik $G = (N, T, P, S)$ aus Definition 2.1 ist die von G erzeugte Sprache $L(G)$ durch*

$$L(G) = \{w : w \in T^* \text{ und } S \Longrightarrow^* w\}$$

definiert.

Entsprechend dieser Definition besteht die von G erzeugte Sprache also aus allen Satzformen von G, die nur Terminale enthalten. Ferner zeigt diese Definition die Notwendigkeit der Angabe von S in der Definition 2.1, da nur die aus S in mehreren Schritten ableitbaren Wörter über T die Sprache bilden.
Diese Definition macht auch deutlich, warum die Elemente aus N bzw. T Nichtterminale oder Hilfssymbole bzw. Terminale heißen. Die Elemente aus N werden für die Sprache selbst nicht benötigt, sie erscheinen nur in Zwischenschritten der Ableitung, haben daher Hilfscharakter. Die Terminale dagegen bilden das Alphabet, über dem die Endwörter definiert werden, wobei Endwort so zu verstehen ist, dass aus diesen Wörtern keine weiteren mehr abgeleitet werden können.
Wir betrachten nun einige Beispiele.

Beispiel 2.4 Wir betrachten die Regelgrammatik

$$G_1 = (\{S, A, B\}, \{a, b\}, \{p_1, p_2, p_3, p_4, p_5\}, S)$$

mit

$$p_1 = S \longrightarrow AB,\ p_2 = A \longrightarrow aA,\ p_3 = A \longrightarrow \lambda,\ p_4 = B \longrightarrow Bb,\ p_5 = B \longrightarrow \lambda.$$

Wir zeigen zuerst, dass jede Satzform von G_1 eine der folgenden Formen hat, wobei n und m beliebige natürliche Zahlen sind:

$$S,\ a^nABb^m,\ a^nAb^m,\ a^nBb^m,\ a^nb^m. \qquad (*)$$

Dies gilt offensichtlich für das Startwort S und das einzige daraus in einem Schritt ableitbare Wort AB ($n = m = 0$). Wir betrachten nun ein Wort der Form a^nABb^m. Hierfür ergeben sich nur die folgenden direkten Ableitungen

$$a^nABb^m \Longrightarrow_{p_2} a^naABb^m, \quad a^nABb^m \Longrightarrow_{p_3} a^n\lambda Bb^m,$$
$$a^nABb^m \Longrightarrow_{p_4} a^nABbb^m, \quad a^nABb^m \Longrightarrow_{p_5} a^nA\lambda b^m.$$

Folglich sind aus a^nABb^m nur die Wörter

$$a^{n+1}ABb^m,\ a^nBb^m,\ a^nABb^{m+1},\ a^nAb^m$$

in einem Schritt ableitbar, die alle von der gewünschten Form sind. Analog kann man leicht nachweisen, dass auch alle in einem Schritt aus a^nAb^m bzw. a^nBb^m ableitbaren Wörter von einer der Formen aus $(*)$ sind. Da aus a^nb^m keine Wörter ableitbar sind, ist damit die obige Aussage bewiesen.
Wir beweisen nun, dass sogar jedes Wort der in $(*)$ genannten Form eine Satzform von G_1 ist. Mit Ausnahme von a^nAb^m folgt dies aus der folgenden Ableitung:

$$\begin{aligned} S \ &\Longrightarrow_{p_1} \ \underbrace{AB \Longrightarrow aAB \Longrightarrow aaAB \Longrightarrow \ldots \Longrightarrow a^{n-1}AB}_{(n-1)-malige\ Anwendung\ von\ p_2} \\ &\Longrightarrow_{p_2} \ \underbrace{a^nAB \Longrightarrow a^nABb \Longrightarrow a^nABb^2 \Longrightarrow \ldots \Longrightarrow a^nAB^m}_{m-malige\ Anwendung\ von\ p_4} \\ &\Longrightarrow_{p_3} \ a^nBb^m \Longrightarrow_{p_5} a^nb^m. \end{aligned}$$

Da die von G_1 erzeugte Sprache nur Wörter über $\{a, b\}$ enthält, besteht $L(G_1)$ aus allen Wörtern der Form $(*)$ in $\{a, b\}^*$. Somit gilt

$$L(G_1) = \{a^nb^m : n \geq 0, m \geq 0\}.$$

Beispiel 2.5 Es sei

$$G_2 = (\{S\}, \{a, b\}, \{S \longrightarrow aSb, S \longrightarrow ab\}, S).$$

Mittels vollständiger Induktion zeigen wir nun, dass durch $n \geq 1$ Ableitungsschritten genau die Wörter a^nSb^n und a^nb^n aus S erzeugt werden können.
Dies gilt offenbar für $n = 1$, denn aus dem Axiom S werden durch Anwendung der beiden Regel $S \longrightarrow aSb$ bzw. $S \longrightarrow ab$ die Wörter aSb bzw. ab abgeleitet.
Sei nun w ein Wort, das durch n Ableitungsschritte aus S erzeugt wird. Nach Definition muss w dann durch Anwendung einer Regel auf ein Wort v entstehen, wobei sich v in $n - 1$ Schritten erzeugen lässt. Nach Induktionsannahme muss also $v = a^{n-1}Sb^{n-1}$ oder $v = a^{n-1}b^{n-1}$ gelten. Im ersten Fall sind durch Ersetzung von S entsprechend den beiden Regeln die Wörter a^nSb^n und a^nb^n ableitbar; im zweiten Fall enthält v nur Terminale,

womit aus v kein Wort mehr abgeleitet werden kann. Somit sind in n Schritten nur a^nSb^n und a^nb^n erzeugbar. Dies beweist aber gerade die Induktionsbehauptung.
Da die Wörter aus $L(G_2)$ in einer endlichen Anzahl von Schritten abgeleitet werden müssen und nur Terminale enthalten dürfen, folgt

$$L(G_2) = \{a^nb^n : n \geq 1\}.$$

Beispiel 2.6 Wir betrachten die Regelgrammatik

$$G_3 = (\{S, A\}, \{a, b\}, \{S \longrightarrow \lambda, S \longrightarrow aS, S \longrightarrow Sb\}, S).$$

Wie in Beispiel 2.4 können wir zeigen, dass die Menge der Satzformen aus allen Wörtern der Form a^nSb^m oder a^nb^m mit $n \geq 0$ und $m \geq 0$ besteht, oder wir beweisen in Analogie zu Beispiel 2.5, dass in $k \geq 1$ Schritten genau die Wörter $a^nSb^m, a^{n-1}b^m, a^nb^{m-1}$ mit $n + m = k$ erzeugt werden können. Daraus ergibt sich

$$L(G_3) = \{a^nb^m : n \geq 0, m \geq 0\}.$$

Beispiel 2.7 Es sei

$$G_4 = (\{S, A\}, \{a, b\}, \{S \longrightarrow \lambda, S \longrightarrow aS, S \longrightarrow a, S \longrightarrow A, A \longrightarrow bA, A \longrightarrow b\}, S).$$

In Abbildung 2.1 sind – bis auf $S \Longrightarrow \lambda$ – im Wesentlichen alle möglichen Ableitungen dargestellt, wobei die nach rechts gerichten Pfeile der Anwendung von $S \longrightarrow aS$ bzw. $A \longrightarrow bA$, die nach oben der von $S \longrightarrow a$ und die nach unten der von $A \longrightarrow b$ entsprechen; die durch die Regel $S \longrightarrow A$ hervorgebrachten Ableitungen sind noch zusätzlich einzutragen (jeweils senkrecht bis zum nächsten Wort). Daraus ist leicht zu ersehen, dass sich erneut

$$L(G_4) = \{a^nb^m : n \geq 0, m \geq 0\}$$

ergibt. Ein formaler Beweis wie in den vorangegangenen Beispielen bleibt dem Leser überlassen.

Beispiel 2.8 Es sei die Regelgrammatik

$$G_5 = (\{S, A, B, B', B''\}, \{a, b, c\}, \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}, S)$$

mit

$$p_1 = S \longrightarrow ABA,\ p_2 = AB \longrightarrow aAbB',\ p_3 = AB \longrightarrow abB'',\ p_4 = B'b \longrightarrow bB',$$
$$p_5 = B''b \longrightarrow bB'',\ p_6 = B'A \longrightarrow BAc,\ p_7 = B''A \longrightarrow c,\ p_8 = bB \longrightarrow Bb$$

gegeben. Durch eine Analyse aller möglichen Ableitungen wollen wir $L(G_5)$ bestimmen.
Für $n \geq 0$ sei $w_n = a^nABb^nAc^n$.
Wir betrachten zuerst den Fall $n \geq 2$. Die einzigen auf w_n anwendbaren Regeln sind p_2 und p_3.
Fall 1: Anwendung von p_2. Wir erhalten das Wort $a^{n+1}AbB'b^nAc^n$. Nun ist nur p_4 anwendbar, und die Anwendung dieser Regel liefert $a^{n+1}AbbB'b^{n-1}Ac^n$, d.h. wir haben B' um eine Position nach rechts verschoben. Erneut ist nur p_4 anwendbar, und wir können

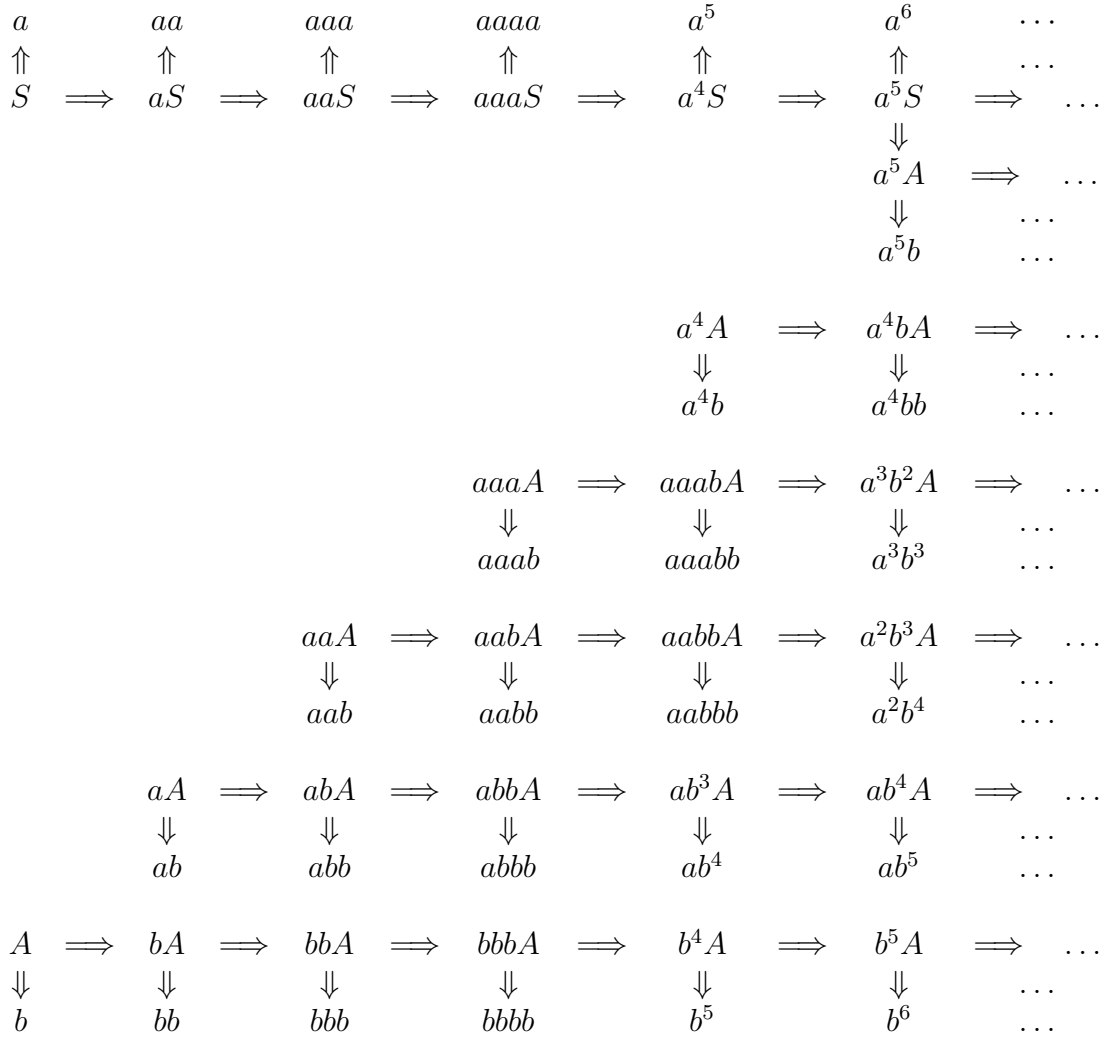


Abbildung 2.1: Ableitungen in Beispiel 2.7

B' um eine Position weiter nach rechts verschieben. Diese Situation hält an, bis wir das Wort $a^{n+1}Ab^{n+1}B'Ac^n$ erzeugt haben. Nun ist nur p_6 anwendbar, durch deren Anwendung $a^{n+1}Ab^{n+1}BAc^{n+1}$ entsteht. Jetzt kann nur p_8 angewendet werden, wodurch eine Verschiebung von B um eine Position nach links bewirkt wird. Erneut ist nur diese Verschiebung möglich, bis wir $w_{n+1} = a^{n+1}ABb^{n+1}Ac^{n+1}$ erhalten.

Fall 2: Anwendung von p_3. Wir erhalten das Wort $a^{n+1}bB''b^nAc^n$. Nun ist nur p_5 anwendbar, d.h. B'' wird um eine Position nach rechts verschoben. Diese Situation bleibt erhalten, bis wir das Wort $a^{n+1}b^{n+1}B''Ac^n$ erzeugt haben. Nun ist nur p_7 anwendbar, durch deren Anwendung $a^{n+1}b^{n+1}c^{n+1}$ entsteht.

Somit wird aus w_n entweder w_{n+1}, womit der eben beschriebene Prozess erneut gestartet werden kann, oder $a^{n+1}b^{n+1}c^{n+1}$ abgeleitet.

Analog kann man sich überlegen, dass w_0 und w_1 nur die Ableitungen

$$w_0 \Longrightarrow^* w_1, w_0 \Longrightarrow^* abc, w_1 \Longrightarrow^* w_2, w_1 \Longrightarrow^* a^2b^2c^2$$

gestatten. Wegen $S \Longrightarrow w_0$ gilt folglich

$$L(G_5) = \{a^n b^n c^n : n \geq 1\}.$$

Beispiel 2.9 Wir betrachten die Regelgrammatik

$$G_6 = (\{S, A, B, B', B''\}, \{a, b, c\}, \{p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}, S)$$

mit

$$\begin{aligned} &p_0 = S \to abc,\ p_1 = S \longrightarrow aABbA,\ p_2 = AB \longrightarrow aAbB',\\ &p_3 = AB \longrightarrow abB'',\ p_4 = B'b \longrightarrow bB',\ p_5 = B''b \longrightarrow bB'',\\ &p_6 = B'A \longrightarrow BAc,\ p_7 = B''A \longrightarrow cc,\ p_8 = bB \longrightarrow Bb. \end{aligned}$$

Wie im vorhergehenden Beispiel können wir

$$L(G_6) = \{a^n b^n c^n \mid n \geq 1\}$$

zeigen.

Beispiel 2.10 Wir betrachten die Regelgrammatik $G_7 = (N, T, P, S)$ mit

$$\begin{aligned} N &= \{S\},\\ T &= \{x, y, z, +, -, \cdot, :, (,)\},\\ P &= \{S \longrightarrow (S + S), S \longrightarrow (S - S), S \longrightarrow (S \cdot S), S \longrightarrow (S : S),\\ &\qquad S \longrightarrow x, S \longrightarrow y, S \longrightarrow z\}. \end{aligned}$$

Wir wollen beweisen, dass $L(G_7)$ aus allen exakt geklammerten arithmetischen Ausdrücken mit den Variablen x, y, z (wobei keine Vorrangregeln für die Operationen beachtet werden und auch äußere Klammern mitgeführt werden) besteht.
Hierfür zeigen wir erst, dass jede Satzform, die aus S erzeugt werden kann, ein exakt geklammerter Ausdruck in den Variablen S, x, y, z ist. Dies folgt aber sofort daraus, dass das Axiom ein solcher Ausdruck ist und aus exakt geklammerten Ausdrücken wieder nur solche entstehen, denn die Ersetzung von S durch x, y, z oder $(S \circ S)$ mit $\circ \in \{+, -, \cdot, :\}$ bewahrt exakte Klammerungen.
Wir zeigen nun mittels Induktion über die Anzahl der Schritte in der Konstruktion eines exakt geklammerten Ausdrucks, dass *alle* exakt geklammerten Ausdrücke in $L(G_7)$ sind. Für $n = 0$ erhalten wir nur Variable, und x, y, z sind aus S mittels der Anwendung der Regeln $S \longrightarrow x, S \longrightarrow y, S \longrightarrow z$ direkt erzeugbar. Seien nun $n \geq 1$ und w ein durch n Schritte erzeugter exakt geklammerter Ausdruck. Dann gilt $w = (w_1 \circ w_2)$ für eine Operation $\circ \in \{+, -, \cdot, :\}$ und exakt geklammerte Ausdrücke w_1 und w_2, von denen jeder durch höchstens $n - 1$ Konstruktionsschritte gewonnen wird. Nach Induktionsannahme gelten damit

$$S \Longrightarrow^* w_1 \quad \text{und} \quad S \Longrightarrow^* w_2.$$

Somit gibt es auch die Ableitung

$$S \Longrightarrow (S \circ S) \Longrightarrow^* (w_1 \circ S) \Longrightarrow^* (w_1 \circ w_2) = w.$$

Damit ist $w \in L(G_7)$ gezeigt.

Beispiel 2.11 In diesem Beispiel wollen eine Regelgrammatik angeben, die alle **LOOP**/-**WHILE**-Programme aus Abschnitt 1.1 erzeugt.
Entsprechend den Definitionen müssen sich alle **LOOP**/**WHILE**-Programme aus dem Startsymbol herleiten lassen. Die Regeln, mittels derer **LOOP**/**WHILE**-Programme erzeugt werden können, sind im Wesentlichen bei der Definition von **LOOP**/**WHILE**-Programmen angegeben worden; es handelt sich um die Grundanweisungen, das Hintereinanderausführen und den **LOOP**- bzw. **WHILE**-Befehl. Wir müssen diesen Prozess nur formal als Grammatik aufschreiben. Dafür verwenden wir das Nichtterminal A als Bezeichnung für ein beliebiges Programm und ersetzen es jeweils durch die zugelassen Befehle; wir haben also A für die Bezeichnungen Π, Π_1 und Π_2 von Programmen zu ersetzen. A ist dann natürlich auch das Axiom, da wir Programme erzeugen wollen. (Wir wählen die Bezeichnung A, da S bereits für die Nachfolgerfunktion vergeben ist.)
Ein Problem bereiten noch die Variablen, da wir davon unendlich viele benötigen, unsere Alphabete der Terminale und Nichtterminale aber endlich sein müssen. Deshalb gehen wir wie folgt vor. Anstelle von x_i verwenden wir die Notation $x[i]$ (wie in Programmiersprachen üblich). Nun muss i eine natürliche Zahl sein, und kann daher durch eine Folge von Ziffern repräsentiert werden. Wir gehen daher von $x[I]$ aus, wobei I ein zusätzliches Nichtterminal ist, aus dem wir alle Ziffernfolgen (ohne führende Nullen) ableiten.
Aus diesen Bemerkungen ergibt sich formal die Regelgrammatik

$$G_8 = (\{A, I, J\}, T, P, A)$$

mit dem Terminalalphabet

$$\begin{aligned} T \;=\; \{&S, P, \mathbf{LOOP}, \mathbf{WHILE}, \mathbf{BEGIN}, \mathbf{END}, :=, \neq, ;, , (,) \\ &0, 1, 2, 3, 4, 5, 6, 7, 8, 9, x, [,] \,\} \end{aligned}$$

(man beachte, dass das Semikolon ein Element von T ist, während die Kommata beim Aufschreiben von T als Trennzeichen zwischen den Elementen aus T fungieren) und der Regelmenge

$$\begin{aligned} P \;=\; \{&A \to x[I] := 0,\ A \to x[I] := x[I],\ A \to x[I] := S(x[I]),\ A \to x[I] := P(x[I]), \\ &A \to A; A,\ A \to \mathbf{LOOP}\ x[I]\ \mathbf{BEGIN}\ A\ \mathbf{END}, \\ &A \to \mathbf{WHILE}\ x[I] \neq 0\ \mathbf{BEGIN}\ A\ \mathbf{END}\} \\ &\cup \{I \to z,\ I \to Jz,\ J \to Jz \mid z \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}\} \\ &\cup \{J \to z \mid z \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}\} \end{aligned}$$

(zuerst erzeugen wir aus I die letzte Ziffer mittels $I \to z$ oder $I \to Jz$, wobei z eine beliebige Ziffer ist; nun werden aus J analog die davor stehenden Ziffern erzeugt; bei der abschließenden Terminierung durch $J \to z$ darf dann z nicht 0 sein, da sonst eine führende Null entstehen würde).

Wir führen nun einige spezielle Typen von Regelgrammatiken ein.

Definition 2.12 *Es sei $G = (N, T, P, S)$ eine Regelgrammatik wie in Definition 2.1. Wir sagen,*

- *G ist monoton, wenn für alle Regeln $\alpha \longrightarrow \beta \in P$ die Bedingung $|\alpha| \leq |\beta|$ erfüllt ist, wobei als Ausnahme $S \longrightarrow \lambda$ zugelassen ist, wenn $|\beta'|_S = 0$ für alle Regeln $\alpha' \longrightarrow \beta' \in P$ gilt,*

- *G ist kontextabhängig, wenn alle Regeln in P von der Form $uAv \longrightarrow uwv$ mit $u, v \in V^*$, $A \in N$ und $w \in V^+$ sind, wobei als Ausnahme $S \longrightarrow \lambda$ zugelassen ist, wenn $|\beta'|_S = 0$ für alle Regeln $\alpha' \longrightarrow \beta' \in P$ gilt,*

- *G ist kontextfrei, wenn alle Regeln in P von der Form $A \longrightarrow w$ mit $A \in N$ und $w \in V^*$ sind,*

- *G ist regulär, wenn alle Regeln in P von der Form $A \longrightarrow wB$ oder $A \longrightarrow w$ mit $A, B \in N$ und $w \in T^*$ sind.*

Die monotonen Grammatiken haben – abgesehen von der Ausnahmeregelung – die Eigenschaft, dass bei Anwendung einer Regel die Länge des abgeleiteten Wortes nicht kleiner ist als die des Ausgangswortes, d.h. $\longrightarrow$ ist bezüglich der Wortlänge eine monotone Relation. Bei kontextabhängigen Grammatiken wird bei Anwendung einer Regel $uAv \longrightarrow uwv$ eigentlich nur das Nichtterminal A durch das Wort w ersetzt; aber diese Ersetzung ist nur erlaubt, wenn links bzw. rechts von A das Wort u bzw. v stehen, d.h. es wird die Existenz eines lokalen Kontextes von A für die Ersetzung gefordert. Genau dieser Kontext wird bei kontextfreien Grammatiken nicht gefordert (daher wäre der Begriff „kontextunabhängig“ eigentlich besser, denn A steht in einem Kontext, der aber für die Ersetzung unerheblich ist; es hat sich aber „kontextfrei“ eingebürgert und durchgesetzt).
Reguläre Grammatiken sind entsprechend der Definition 2.12 ein Spezialfall kontextfreier Grammatiken, die durch zusätzliche strukturelle Forderungen an die rechten Seiten der Regeln gekennzeichnet sind.
Da das Leerwort als rechte Seite bei Regeln von Regelgrammatiken, kontextfreien und regulären Grammatiken zugelassen ist, ist klar, dass das Leerwort auch in der erzeugten Sprache liegen kann. Die Ausnahmeregelungen in der Definition monotoner und kontextabhängiger Grammatiken dienen dazu, diese Eigenschaft auch für diese Typen von Grammatiken abzusichern.
Außer den in Definition 2.12 eingeführten Bezeichnungen wird vielfach auch Typ 0 für beliebige Regelgrammatiken, Typ 1 für kontextabhängige, Typ 2 für kontextfreie und Typ 3 für reguläre Grammatiken benutzt.

Wir klassifizieren nun die Grammatiken aus den obigen Beispielen hinsichtlich der Eigenschaften von Definition 2.12.
G_1 ist wegen der Regel $p_3 = A \longrightarrow \lambda$ nicht monoton und nicht kontextabhängig. G_1 ist auch nicht regulär, da die Regel $p_4 = B \longrightarrow Bb$ in der Regelmenge von G_1 existiert. G_1 ist aber offensichtlich kontextfrei.
G_2 ist monoton, kontextabhängig (für alle Regeln gilt $u = v = \lambda$) und kontextfrei, aber nicht regulär.
G_3 ist nicht monoton und nicht kontextabhängig (wegen der gleichzeitigen Existenz der Regeln $S \longrightarrow \lambda$ und $S \longrightarrow aS$) und nicht regulär, aber kontextfrei.
G_4 ist regulär und damit auch kontextfrei, aber nicht monoton und nicht kontextabhängig.
G_5 hat keine der in Definition 2.12 gegebenen Eigenschaften. G_6 ist monoton, aber

weder kontextabhängig noch kontextfrei noch regulär. G_7 und G_8 sind monoton, kontextabhängig und kontextfrei, jedoch nicht regulär.

Definition 2.13 *Eine Sprache L heißt monoton (bzw. kontextabhängig, kontextfrei oder regulär), wenn es eine monotone (bzw. kontextabhängige, kontextfreie oder reguläre) Grammatik G mit $L = L(G)$ gibt.*

Nach dieser Definition ist $L = \{a^n b^m : n \geq 0, m \geq 0\}$ eine kontextfreie Sprache, denn es gilt $L = L(G_3)$, und G_3 ist eine kontextfreie Grammatik. Jedoch lässt sich aus der Tatsache, dass G_3 keine reguläre Grammatik ist, nicht schließen, dass L keine reguläre Sprache ist. Da nämlich G_4 ebenfalls die Sprache L erzeugt und G_4 eine reguläre Grammatik ist, ist L regulär.

Mit $\mathcal{L}(REG)$, $\mathcal{L}(CF)$, $\mathcal{L}(CS)$, $\mathcal{L}(MON)$ und $\mathcal{L}(RE)$ bezeichnen wir die Menge aller Sprachen, die von regulären, kontextfreien, kontextabhängigen, monotonen und beliebigen Regelgrammatiken erzeugt werden.[1]
Wir bemerken zuerst, dass für zwei Typen X und Y von Grammatiken aus dem Fakt, dass jede Grammatik vom Typ X auch eine vom Typ Y ist, sich die Aussage $\mathcal{L}(X) \subseteq \mathcal{L}(Y)$ ergibt. Hieraus folgt sofort das folgende Lemma.

Lemma 2.14 *$\mathcal{L}(CS) \subseteq \mathcal{L}(MON) \subseteq \mathcal{L}(RE)$ und $\mathcal{L}(REG) \subseteq \mathcal{L}(CF) \subseteq \mathcal{L}(RE)$.* □

Im nächsten Abschnitt werden weitere Beziehungen zwischen den eingeführten Mengen hergeleitet und festgestellt, ob die Inklusionen in Lemma 2.14 echt oder Gleichheiten sind.

2.1.2 Normalformen und Schleifensätze

Wir werden in diesem Abschnitt zuerst zeigen, dass für die im vorangegangenen Abschnitt eingeführten Typen von Grammatiken jeweils Normalformen existieren, d.h. Grammatiken dieses Typs mit weiteren Einschränkungen an die Regeln, die es aber trotzdem gestatten, jede Sprache dieses Typs von einer Grammatik in Normalform zu erzeugen. Wir benutzen diese Normalformen vor allem als beweistechnische Hilfsmittel und zur Herleitung von Eigenschaften, die uns den Nachweis gestatten, dass gewisse Sprachen nicht durch Grammatiken eines gegebenen Typs erzeugt werden können.
Wir beweisen jeweils nicht nur die Existenz der Normalform, sondern zeigen auch, dass eine Grammatik in Normalform konstruktiv gewonnen werden kann.

Wir beginnen mit Normalformen für monotone Grammatiken.

Lemma 2.15 *Zu jeder Regelgrammatik $G = (N, T, P, S)$ kann eine Regelgrammatik $G' = (N', T, P', S)$ so konstruiert werden, dass alle Regeln aus P' von der Form $\alpha \longrightarrow \beta$ mit $\alpha, \beta \in (N')^*$ oder $A \longrightarrow a$ mit $A \in N', a \in T$ sind und $L(G) = L(G')$ gilt. Ist außerdem G eine monotone, kontextabhängige bzw. kontextfreie Grammatik, so ist auch G' monoton, kontextabhängig bzw. kontextfrei.*

[1] Die hierbei verwendeten Bezeichnungen *REG, CF, CS, MON, RE* sind Abkürzungen der entsprechenden englischen Wörter regular, context-free, context-sensitive, monotone, recursively enumerable.

Beweis. Für jedes Terminal a sei a' ein neues Symbol (das also weder in N noch in T liegt). Ferner sei für $a \neq b, a, b \in T$ auch $a' \neq b'$. Wir setzen

$$N' = N \cup \{a' : a \in T\}.$$

Ist $w = x_1 x_2 \dots x_n$ ein Wort aus V^*, so sei $w' = y_1 y_2 \dots y_n$ das Wort aus $(N')^*$ mit

$$y_i = \begin{cases} x_i & \text{für } x_i \in N \\ x_i' & \text{für } x_i \in T \end{cases}$$

für $1 \leq i \leq n$. Wir definieren nun die Regelmenge von G' durch

$$P' = \{\alpha' \longrightarrow \beta' : \alpha \longrightarrow \beta \in P\} \cup \{a' \longrightarrow a : a \in T\}.$$

Wir beweisen nun $L(G') = L(G)$.
Sei dazu zuerst $w \in L(G)$. Dann gibt es in G eine Ableitung

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

Entsprechend der Konstruktion von P' gibt es dann in G' die Ableitung

$$S = w_0' \Longrightarrow w_1' \Longrightarrow w_2' \Longrightarrow \dots w_n' = w' = v_0 \Longrightarrow v_1 \Longrightarrow v_2 \Longrightarrow \dots \Longrightarrow v_m = w,$$

bei der wir für den Übergang von w_i' zu w_{i+1}' stets die Regel $\alpha' \longrightarrow \beta' \in P'$ anwenden, wenn w_{i+1} aus w_i durch Anwendung der Regel $\alpha \longrightarrow \beta \in P$ entstanden ist und die direkten Ableitungen $v_j \Longrightarrow v_{j+1}$ durch Anwendung einer Regel der Form $a' \longrightarrow a$ geschehen. Daher gilt auch $w \in L(G')$, womit $L(G) \subseteq L(G')$ gezeigt ist.
Sei nun $x \in L(G')$. Dann gibt es für x eine Ableitung der Form

$$S = x_0' \Longrightarrow x_1' \Longrightarrow x_2' \Longrightarrow \dots \Longrightarrow x_n' = x' = y_0 \Longrightarrow y_1 \Longrightarrow y_2 \Longrightarrow \dots \Longrightarrow y_m = x$$

(eine Ableitung dieser Form entsteht aus einer beliebigen Ableitung von w, indem man die Reihenfolge der angewendeten Regeln so vertauscht, dass im ersten Teil nur Regeln der Form $\alpha' \longrightarrow \beta'$ und im zweiten Teil nur Regeln der Form $a' \longrightarrow a$ angewendet werden, wodurch auch abgesichert ist, dass die im ersten Teil der Ableitung entstehenden Satzformen sämtlich nur Symbole aus N' enthalten). Wenn wir nun die Reihenfolge der Regelanwendung nicht ändern, aber stets statt $\alpha' \longrightarrow \beta' \in P'$ die Regel $\alpha \longrightarrow \beta \in P$ benutzen, so erhalten wir die Ableitung

$$S = x_0 \Longrightarrow x_1 \Longrightarrow x_2 \Longrightarrow \dots \Longrightarrow x_n = x$$

in G. Dies beweist $x \in L(G)$ und damit $L(G') \subseteq L(G)$.
Aus den beiden nachgewiesenen Inklusionen folgt $L(G) = L(G')$.
Bei der Konstruktion von P' wird eine Regel $\alpha \longrightarrow \beta$ mit $|\alpha| \leq |\beta|$ in eine Regel $\alpha' \longrightarrow \beta'$ mit $|\alpha'| \leq |\beta'|$ überführt, da $|\alpha| = |\alpha'|$ und $|\beta| = |\beta'|$ gelten. Damit ist G' monoton, wenn G monoton ist. Analog ist sofort zu sehen, dass Regeln der Form $uAv \longrightarrow uwv$ bzw. $A \longrightarrow w$ wieder in Regeln dieser Form übergehen. Hieraus folgt sofort die Aussage über die Kontextabhängigkeit und Kontextfreiheit. □

Satz 2.16 *Zu jeder monotonen Grammatik $G = (N, T, P, S)$ kann eine monotone Grammatik $G' = (N', T, P', S)$ so konstruiert werden, dass jede Regel aus P' von einer der Formen*

$$A \longrightarrow BC,\ A \longrightarrow B,\ AB \longrightarrow CB,\ AB \longrightarrow AC \text{ oder } A \longrightarrow a$$

mit $A, B, C \in N', a \in T$ oder $S \longrightarrow \lambda$ ist und $L(G) = L(G')$ gilt.

Beweis. Wegen Lemma 2.15 können wir annehmen, dass alle Regeln von P von der Form $\alpha \longrightarrow \beta$ oder $A \longrightarrow a$ mit $\alpha, \beta \in N^+, A \in N, a \in T$ (oder $S \longrightarrow \lambda$) sind.
Jeder Regel aus P werden wir nun eine Menge von Regeln und Nichtterminalen so zuordnen, dass die Mengen P' und N' mit den gewünschten Eigenschaften als Vereinigung aller dieser Mengen von Regeln bzw. aller dieser Mengen von Nichtterminalen und N entstehen. Die dabei neu eingeführten Symbole sollen stets paarweise verschieden sein und nicht in $N \cup T$ liegen.
Sei $p = X_1 X_2 \dots X_n \longrightarrow Y_1 Y_2 \dots Y_m$ eine Regel aus P.
Fall 1. $n = 1$ *und* $m \leq 2$. Dann setzen wir

$$P_p = \{p\} \quad \text{und} \quad N_p = \emptyset,$$

d.h. wir übernehmen die Regel p in P' und führen keine neue Hilfssymbole ein.
Fall 2. $n = 1$ *und* $m \geq 3$. Dann setzen wir

$$N_p = \{C_{p,1}, C_{p,2}, \dots, C_{p,m-2}\}$$

und

$$P_p = \{X_1 \longrightarrow Y_1 C_{p,1}, C_{p,1} \longrightarrow Y_2 C_{p,2}, \dots, C_{p,m-3} \longrightarrow Y_{m-2} C_{p,m-2}, C_{p,m-2} \longrightarrow Y_{m-1} Y_m\}.$$

Fall 3. $n \geq 2$. Dann gilt auch $m \geq 2$. Wir setzen nun

$$N'_p = \{C_{p,1}, C_{p,2}, \dots, C_{p,n}, D\}$$

und

$$\begin{aligned} P'_p \;=\; \{ & X_1 X_2 \longrightarrow C_{p,1} X_2, C_{p,1} X_2 \longrightarrow C_{p,1} C_{p,2}, C_{p,2} X_3 \longrightarrow C_{p,2} C_{p,3}, \\ & \dots, C_{p,n-2} X_{n-1} \longrightarrow C_{p,n-2} C_{p,n-1}, C_{p,n-1} X_n \longrightarrow C_{p,n-1} C_{p,n}, \\ & C_{p,1} C_{p,2} \longrightarrow Y_1 C_{p,2}, C_{p,2} C_{p,3} \longrightarrow Y_2 C_{p,3}, \\ & \dots, C_{p,n-2} C_{p,n-1} \longrightarrow Y_{n-2} C_{p,n-1}, C_{p,n-1} C_{p,n} \longrightarrow Y_{n-1} C_{p,n}, \\ & Y_{n-1} C_{p,n} \longrightarrow Y_{n-1} D, D \longrightarrow Y_n Y_{n+1} \dots Y_m \}. \end{aligned}$$

Die Mengen N_p und P_p entstehen nun aus N'_p und P'_p indem wir $D \in N'_p$ und $D \longrightarrow Y_n Y_{n+1} \dots Y_m \in P'_p$ entsprechend Fall 2 durch Nichtterminale und Regeln mit einer rechten Seite der Länge ≤ 2 ersetzen.
Wir konstruieren $G' = (N', T, P', S)$ durch

$$N' = N \cup \bigcup_{p \in P} N_p \qquad \text{und} \qquad P' = \bigcup_{p \in P} P_p.$$

Aus der Konstruktion ist sofort zu sehen, dass alle Regeln von P' von der geforderten Form sind.

Sei nun $v = w_1X_1X_2 \dots X_nw_2$ mit $w_1, w_2 \in V^*$ und $n \geq 2$ eine Satzform von G. Durch Anwendung von p entsteht $v' = w_1Y_1Y_2 \dots Y_mw_2$. In G' haben wir dann die folgende Ableitung

$$\begin{aligned}
v &\Longrightarrow w_1C_{p,1}X_2X_3 \dots X_nw_2 \Longrightarrow w_1C_{p,1}C_{p,2}X_3 \dots X_nw_2 \\
&\Longrightarrow \dots \Longrightarrow w_1C_{p,1}C_{p,2} \dots C_{p,n-1}X_nw_2 \Longrightarrow w_1C_{p,1}C_{p,2} \dots C_{p,n-1}C_{p,n}w_2 \\
&\Longrightarrow w_1Y_1C_{p,2} \dots C_{p,n-1}C_{p,n}w_2 \Longrightarrow w_1Y_1Y_2 \dots C_{p,n-1}C_{p,n}w_2 \\
&\Longrightarrow \dots \Longrightarrow w_1Y_1Y_2 \dots Y_{n-1}C_{p,n}w_2 \\
&\Longrightarrow w_1Y_1Y_2 \dots Y_{n-1}D_{p,n}w_2 \Longrightarrow w_1Y_1Y_2 \dots Y_{n-1}Y_nY_nD_{p,n+1}w_2 \\
&\Longrightarrow w_1Y_1Y_2 \dots Y_{n-1}Y_nY_{n+1}D_{p,n+2}w_2 \Longrightarrow \dots \\
&\Longrightarrow w_1Y_1Y_2 \dots Y_{n-1}Y_nY_{n+1} \dots Y_{m-1}D_{p,m}w_2 \\
&\Longrightarrow w_1Y_1Y_2 \dots Y_{n-1}Y_nY_{n+1} \dots Y_{m-1}Y_mw_2 = v',
\end{aligned}$$

wobei wir die Regeln aus P_p genau in der in Fall 3 angegebenen Reihenfolge anwenden. Damit ist gezeigt, dass wir die Anwendung von p in G durch Anwendung der Regeln aus P_p in G' simulieren können. Analoges gilt auch in den Fällen 1 und 2. Damit kann jede Ableitung in G in G' simuliert werden.
Wir zeigen nun, dass bis auf die Reihenfolge in der Anwendung von Regeln in G' nur derartige Simulationen möglich sind. Dies sieht man wie folgt ein: Wenden wir auf v die Regel $X_1X_2 \longrightarrow C_{p,1}X_2$ an, so können wir auf die entstehende Satzform $v_1 = w_1C_{p,1}X_2 \dots X_nw_2$ nur die Regel $C_{p,1}X_2 \longrightarrow C_{p,1}C_{p,2}$ aus P_p anwenden. Wir setzen dann die Ableitung mittels Regeln aus P_p wie oben fort oder durch Anwendung von $C_{p,1}C_{p,2} \longrightarrow Y_1C_{p,2}$ fort, wodurch $w_1Y_1C_{p,2}X_3 \dots X_nw_2$ entsteht. Auf letztere Satzform ist nur $C_{p,2}X_3 \longrightarrow C_{p,2}C_{p,3}$ anwendbar, wodurch $w_1Y_1C_{p,2}C_{p,3}X_4 \dots X_nw_2$ generiert wird. Auch nun gibt es die Möglichkeit durch Regeln aus P_p das Symbol $C_{p,2}$ durch Y_2 oder X_4 durch $C_{p,4}$ zu ersetzen. Man erkennt also, dass bis auf die Reihenfolge der Regeln schließlich $w_1Y_1 \dots Y_{n-1}D_{p,n}w_2$ erzeugt wird. Nun sind die folgenden anwendbaren Regeln stets eindeutig bestimmt, und wie oben wird v' abgeleitet.
Wir haben noch zu diskutieren, was passiert, wenn auf eine Satzform, die während dieser Simulation entsteht, eine Regel angewendet wird, die nicht zu P_p gehört und mindestens eines der Symbole $X_1, X_2, X_3, \dots, X_n$ verändert. Wir diskutieren dies nur für v_1; die Überlegungen bei den anderen Satzformen sind ähnlich. Es ist leicht zu sehen, dass die Regeln zur Änderung von Symbolen aus $N_p \setminus \{D_{p,m}\}$ (und mindestens das in v_1 vorkommende $C_{p,1} \in N_p$ ist zu ändern, damit die Ableitung auf ein Wort über T führt) ein weiteres Symbol aus N_p einführt. Damit kann v_1 nur dann in ein terminales Wort überführt werden, wenn nach einigen Schritten nur noch $D_{p,m}$ in der Satzform ist und $D_{p,m} \longrightarrow Y_m$ angewendet wird. Dies erfordert aber, dass alle Regeln aus P_p angewendet wurden und damit die Anwendung von p in G simuliert wurde.
Da somit in G' alle direkten Ableitungen in G simuliert werden können und nur Simulationen von Ableitungen in G möglich sind, gilt für Wörter w, w' über $N \cup T$, dass $w \Longrightarrow_G^* w'$ genau dann gilt, wenn auch $w \Longrightarrow_{G'}^* w'$ gültig ist. Hieraus folgt $S \Longrightarrow_G^* w$ mit $w \in T^*$ gilt genau dann, wenn $S \Longrightarrow_{G'}^* w$ gültig ist. Dies impliziert $L(G) = L(G')$. □

Folgerung 2.17 $\mathcal{L}(MON) = \mathcal{L}(CS)$.

Beweis. Am Ende von Abschnitt 2.1.1 wurde bereits bemerkt, dass $\mathcal{L}(CS) \subseteq \mathcal{L}(MON)$ gilt.
Wir haben also nur $\mathcal{L}(MON) \subseteq \mathcal{L}(CS)$ zu zeigen, d.h. wir müssen nachweisen, dass jede monotone Sprache auch kontextabhängig ist. Sei L eine monotone Sprache. Dann gibt es eine monotone Grammatik G mit $L = L(G)$. Nach Satz 2.16. gibt es dann eine monotone Grammatik G', deren Regeln alle von kontextabhängiger Form sind, d.h. G' ist kontextabhängig, und die $L = L(G) = L(G')$ erfüllt. Folglich ist L eine kontextabhängige Sprache. □

Entsprechend Satz 2.16 wird jede kontextfreie Sprache durch eine Grammatik erzeugt, die nur Regeln der Form

$$A \to BC, A \to B, A \to \lambda \text{ und } A \to a \text{ mit } A, B, C \in N, a \in T$$

hat. Wir zeigen nun, dass auch die Regeln der Form $A \to \lambda$ eliminiert werden können, wobei wir dann natürlich die gleiche Ausnahmeregelung zulassen müssen, die uns schon von den monotonen oder kontextabhängigen Grammatiken geläufig ist.

Lemma 2.18 *Zu jeder kontextfreien Grammatik $G = (N, T, P, S)$ existiert eine kontextfreie Grammatik $G' = (N', T, P', S)$ derart, dass*
i) P' keine Regel der Form $A \longrightarrow \lambda$ mit $A \neq S$ enthält,
ii) $|w|_S = 0$ für alle Regeln $A \longrightarrow w \in P'$ gilt, und
iii) $L(G) = L(G')$ ist.

Beweis. Wir konstruieren als erstes zu der Grammatik $G = (N, T, P, S)$ eine kontextfreie Grammatik $G'' = (N'', T, P'', S')$, die die Bedingung ii) und $L(G) = L(G'')$ erfüllt. Dazu fügen wir zu N ein neues Nichtterminal S' hinzu, d.h. wir setzen $N'' = N \cup \{S'\}$. Weiterhin erweitern wir die Regelmenge durch $P'' = P \cup \{S' \to S\}$. ii) gilt dann nach Definition. Da alle Ableitungen in G'' von der Form $S'' \Longrightarrow S \Longrightarrow^* w$ sind, haben wir auch $L(G'') = L(G)$.
Es sei

$$M = \{A : A \in N'', A \Longrightarrow^* \lambda\}.$$

Mit jeder Regel

$$q'' = A \longrightarrow v_1 A_1 v_2 A_2 \dots v_m A_m v_{m+1}$$

mit

$$m \geq 0, A_1, A_2, \dots, A_m \in N'', v_1, v_2, \dots, v_{m+1} \in T^*$$

assoziieren wir die Menge $P_{q''}$ aller Regeln der Form

$$A \longrightarrow v_1 X_1 v_2 X_2 \dots v_m X_m v_{m+1} \neq \lambda,$$

für die

$$X_i = A_i \text{ für } A_i \notin M \quad \text{und} \quad X_i \in \{A_i, \lambda\} \text{ für } A_i \in M$$

für $1 \leq i \leq m$ gilt. Aufgrund dieser Definition kann keine Menge $P_{q''}$ eine Regel der Form $Y \longrightarrow \lambda$ enthalten. Damit ist es nicht möglich das Leerwort unter Verwendung von Regeln aus $P_{q''}$ zu erzeugen. Deshalb setzen wir

$$\overline{P} = \begin{cases} \{S' \longrightarrow \lambda\} & \text{falls } S' \in M \\ \emptyset & \text{sonst} \end{cases}.$$

Weiterhin definieren wir $G' = (N', T, P', S')$ durch

$$N' = N'' \quad \text{und} \quad P' = \overline{P} \cup \bigcup_{q'' \in P''} P_{p''}.$$

Wir bemerken, dass bei der Konstruktion von P' aus P'' die Eigenschaft ii) erhalten geblieben ist, und dass P' nach Konstruktion die Eigenschaft i) hat.
Wir zeigen jetzt, dass auch die Bedingung iii) erfüllt ist. Dafür reicht es $L(G'') = L(G')$ zu zeigen.

Zuerst beweisen wir mittels vollständiger Induktion über die Anzahl der Ableitungsschritte, dass für jedes Nichtterminal A und jedes Wort $x \in T^+$ mit $A \Longrightarrow^*_{G''} x$ auch $A \Longrightarrow^*_{G'} x$ gilt.
Sei $n = 1$. Jede direkte Ableitung ist in beiden Grammatiken von der Form $A \Longrightarrow v$, bei der in beiden Fällen die Regel $A \longrightarrow v$ angewendet wird. Somit ist der Induktionsanfang gezeigt.
Sei nun x ein in $n \geq 2$ Schritten aus A ableitbares terminales Wort. Dann gilt

$$A \Longrightarrow_{G''} v_1 A_1 v_2 A_2 \dots v_m A_m v_{m+1} \Longrightarrow^*_{G''} v_1 x_1 v_2 x_2 \dots v_m x_m v_{m+1} = x,$$

wobei die Ableitungen $A_i \Longrightarrow^*_{G''} x_i$ für $1 \leq i \leq m$ sämtlich aus weniger als n Schritten bestehen. Wir unterscheiden nun zwei Fälle:
Fall 1. $x_i \neq \lambda$. Dann setzen wir $X_i = A_i$ und haben nach Induktionsannahme $X_i = A_i \Longrightarrow^*_{G'} x_i$.
Fall 2. $x_i = \lambda$. Dann gilt $A_i \in M$ und wir setzen $X_i = \lambda$.
Nach Konstruktion gibt es in P' die Regel $A \longrightarrow v_1 X_1 v_2 X_2 \dots v_m X_m v_{m+1}$ und wir erhalten in G' die Ableitung

$$A \Longrightarrow_{G'} v_1 X_1 v_2 X_2 \dots v_m X_m v_{m+1} \Longrightarrow^*_{G'} v_1 x_1 v_2 x_2 \dots v_m x_m v_{m+1},$$

wobei wir für $x_i = \lambda$ einfach $X_i = x_i = \lambda$ und für $x_i \neq \lambda$ die Ableitungen $X_i \Longrightarrow^*_{G'} x_i$ benutzen.
Betrachten wir die gerade bewiesene Aussage für $A = S$, so ist jedes vom Leerwort verschiedene Wort aus $L(G'')$ auch in G' ableitbar. Damit gilt $L(G'') \setminus \{\lambda\} \subseteq L(G') \setminus \{\lambda\}$. Da durch $\overline{P}$ gesichert ist, dass $\lambda \in L(G'')$ genau dann gilt, wenn $\lambda \in L(G')$ ist, ist sogar $L(G'') \subseteq L(G')$ gültig.

Wir zeigen nun wiederum mittels vollständiger Induktion die Umkehrung, d.h., dass jede Ableitung $A \Longrightarrow^*_{G'} y$ eines terminalen Wortes y auch eine Entsprechung $A \Longrightarrow^*_{G''} y$ findet.
Der Induktionsanfang ergibt sich wie oben.
Sei daher $A \Longrightarrow^*_{G'} y$ eine Ableitung aus $n \geq 2$ Schritten. Dann gilt

$$A \Longrightarrow v_1 X_1 v_2 X_2 \dots v_m X_m v_{m+1} \Longrightarrow^*_{G'} v_1 x_1 v_2 x_2 \dots v_m x_m v_{m+1},$$

wobei für $X_i = \lambda$ auch $x_i = \lambda$ ist, und für $X_i \neq \lambda$ ist $X_i \Longrightarrow^*_{G'} x_i$ eine Ableitung mit weniger als n Schritten. Nach Konstruktion der Regel $A \longrightarrow v_1X_1v_2X_2 \ldots v_mX_mv_{m+1}$ aus P' gibt es dann eine Ableitung $A_i \Longrightarrow^* \lambda = x_i$, falls $X_i = \lambda$ ist, und nach Induktionsvoraussetzung gilt auch $A_i \Longrightarrow^*_{G''} x_i$ für $X_i \neq \lambda$. Deshalb existiert in G'' die Ableitung

$$A \Longrightarrow_{G''} v_1A_1v_2A_2 \ldots v_mA_mv_{m+1} \Longrightarrow^*_{G'} v_1x_2v_2x_2 \ldots v_mx_mv_{m+1}.$$

Hiervon ausgehend zeigt man wie oben $L(G') \subseteq L(G'')$. □

Um die Grammatik G' aus dem vorstehenden Beweis wirklich konstruieren zu können, benötigen wir einen Algorithmus, der die Menge M bestimmt. Wir setzen

$$\begin{aligned}
M_0 &= \emptyset, \\
P_0 &= P, \\
M_i &= M_{i-1} \cup \{A : A \in N'', A \to \lambda \in P_{i-1}\}, \\
P_i &= \{A \to w_1w_2 \ldots w_{n+1} : A \to w_1A_1w_2A_2 \ldots w_nA_nw_{n+1} \in P_{i-1} \\
&\quad n \geq 0, w_j \in (N'' \setminus M_i)^* \text{ für } 1 \leq j \leq n+1, A_j \in M_i \text{ für } 1 \leq j \leq n\}
\end{aligned}$$

für $i \geq 1$. Für $i \geq 1$ erfordert die Konstruktion von M_i das Durchmustern aller Regeln von P_{i-1}, ob sie von der Form $A \to \lambda$ sind, und die Konstruktion von P_i das Ersetzen aller Symbole aus M_i durch das Leerwort in allen Regeln von P.
Wir zeigen zuerst mittels Induktion $M_i \subseteq M$ für $i \geq 0$. Für $i = 0$ und $i = 1$ ist dies nach Konstruktion klar. Für $A \in M_i$, $i \geq 2$, gibt es nach Definition von M_i eine Regel $A \to A_1A_2 \ldots A_n$ mit $A_j \in M_{i-1}$ für $1 \leq j \leq n$. Da nach Induktionsvoraussetzung $A_j \in M$ für $1 \leq j \leq n$ gilt, gibt es die Ableitung

$$A \Longrightarrow A_1A_2 \ldots A_n \Longrightarrow^* \lambda A_2A_3 \ldots A_n \Longrightarrow^* \lambda\lambda A_3 \ldots A_n \Longrightarrow^* \lambda^n = \lambda,$$

woraus $A \in M$ folgt.
Sei nun $A \in M$. Wir betrachten eine Ableitung $A \Longrightarrow^* \lambda$. In keiner Satzform dieser Ableitung kann ein Terminal vorkommen, die Satzformen sind also alle Wörter über N''. Durch Umordnen der Ableitungsschritte können wir eine Ableitung

$$A = w_0 \Longrightarrow^* w_1 \Longrightarrow^* w_2 \Longrightarrow^* \ldots \Longrightarrow^* w_m = \lambda$$

erreichen, bei der $w_{i-1} \Longrightarrow^* w_i$ dadurch entsteht, dass alle Nichtterminale aus w_{i-1} entsprechend einer Regel ersetzt werden. Offenbar gilt dann $w_{m-1} \in M_1^*$, da die darin enthaltenen Nichtterminale in einem Ableitungsschritt durch das Leerwort ersetzt werden. Für ein Nichtterminal B aus w_{m-2} gilt daher $B \to \lambda$ oder $B \to w \in M_1^+$, womit sich $B \in M_1$ oder $B \in M_2$ und damit sicher $B \in M_2$ ergibt. So fortfahrend erhalten wir $w_{m-3} \in M_3^*$, $w_{m-4} \in M_4^*$ und schließlich $A = w_0 = w_{m-m} \in M_m$.
Aus dem bisher Bewiesenem folgt

$$M = \bigcup_{i \geq 0} M_i.$$

Entsprechend den obigen Definitionen impliziert $M_i = M_{i+1}$ sofort $P_i = P_{i+1}$ und dann

$$M_i = M_{i+1} = M_{i+2} = \ldots \quad \text{und} \quad P_i = P_{i+1} = P_{i+2} = \ldots$$

Da außerdem stets $M_i \subseteq M_{i+1}$ gilt, tritt die Gleichheit spätestens bei M_t ein. Somit ergibt sich

$$M_t = \bigcup_{i \geq 0} M_i = M.$$

Beispiel 2.19 Wir illustrieren die eben beschriebene Konstruktion anhand der Grammatik

$$G = (\{S, A, B\}, \{a, b\}, \{S \to SA, S \to \lambda, A \to aAb, A \to B, B \to \lambda\}, S)\,.$$

Wir bemerken, dass

$$L(G) = \{a^{n_1} b^{n_1} a^{n_2} b^{n_2} \ldots a^{n_k} b^{n_k} : k \geq 0, n_i \geq 0, 1 \leq i \leq k\}$$

gilt, da durch die ersten beiden Regeln eine beliebige Anzahl von A's erzeugt wird, von denen jedes eine Sprache der Form $\{a^n b^n : n \geq 0\}$ erzeugt.
Es ergeben sich dann

$$\begin{aligned}
& N'' = N \cup \{S'\} = \{S, A, B, S'\}, \\
& P'' = \{S' \to S, S \to SA, S \to \lambda, A \to aAb, A \to B, B \to \lambda\} \\
& M_0 = \emptyset \text{ und } P_0 = P''\,, \\
& M_1 = \{S, B\} \text{ und } P_1 = \{S' \to \lambda, S \to A, S \to \lambda, A \to aAb, A \to \lambda, B \to \lambda\}\,, \\
& M_2 = \{S, B, S', A\} = N'' \\
& N' = N'' = \{S', S, A, B\}, \\
& \overline{P} = \{S \longrightarrow \lambda\}, \\
P' \;=\; & \overline{P} \cup \{S' \to S, S \to SA, S \to A, S \to S, A \to aAb, A \to ab\}, A \to B\}\,.
\end{aligned}$$

Man sieht sofort, dass P' offenbar überflüssige Regeln enthält. Dies trifft auf $S \to S$ zu, da diese Regel keine Änderung bei ihrer Anwendung bewirkt, und auf $A \to B$ zu, da P' keine Regeln enthält, die B auf der rechten Seite haben. Wir werden diese Regeln aber hier nicht streichen, da dies der Algorithmus im Beweis von Lemma 2.18 nicht vorsieht.

Es ist offenbar, dass – mit Ausnahme der eventuell existierenden Regel $S \longrightarrow \lambda$ – für alle anderen Regel $A \longrightarrow w \in P'$ bei der in Lemma 2.18 konstruierten Grammatik G' die Beziehung $w \in (N' \cup T)^+$ und damit $|w| \geq 1 = |A|$ gilt. Dies bedeutet, dass G' eine monotone Grammatik ist. Somit erhalten wir das folgende Resultat.

Folgerung 2.20 $\mathcal{L}(CF) \subseteq \mathcal{L}(MON)$. □

Wir zeigen nun, dass die in Satz 2.16 zugelassenen Regeln der Form $A \longrightarrow B$ mit $A, B \in N$ ebenfalls eliminiert werden können.

Lemma 2.21 *Zu jeder kontextfreien Grammatik $G = (N, T, P, S)$ kann eine kontextfreie Grammatik $G' = (N, T, P', S)$ so konstruiert werden, dass P' keine Regel der Form $A \longrightarrow B$ mit $A, B \in N$ enthält und $L(G) = L(G')$ gilt.*

Beweis. Nach Lemma 2.18 können wir ohne Beschränkung der Allgemeinheit annehmen, dass G – mit Ausnahme des möglichen Sonderfalles $S \to \lambda$ keine Regeln der Form $A \to \lambda$ enthält.
Für ein Nichtterminal A definieren wir

$$M_A = \{B : B \Longrightarrow_G^* A, B \in N\}$$

(man beachte, dass nach Definition stets $A \in M_A$ gilt). Für eine Regel $p = A \to w$ mit $w \notin N$ setzen wir

$$P_p = \{B \to w : B \in M_A\}$$

(d.h. wir ersetzen eine Ableitung

$$B \Longrightarrow B_1 \Longrightarrow B_2 \Longrightarrow \ldots \Longrightarrow B_k = A \Longrightarrow w$$

durch eine Regel $B \to w$). Wir setzen nun

$$P' = \bigcup_{p \in P} P_p.$$

Offensichtlich erfüllt P' nach Konstruktion die geforderte Bedingung. Die Gültigkeit von $L(G) = L(G')$ lässt sich nun in Analogie zum Beweis von Lemma 2.18. zeigen. □

Beispiel 2.22 Wenden wir die im Beweis von Lemma 2.21 gegebene Konstruktion auf Beispiel 2.19 an, so erhalten wir

$$M_B = \{B, A, S, S'\},\ M_A = \{A, S, S'\},\ M_S = \{S, S'\} \text{ und } M_{S'} = \{S'\}$$

und daher

$$\begin{aligned} P_{S' \to \lambda} &= \{S' \to \lambda\}, \\ P_{S \to SA} &= \{S \to SA, S' \to SA\}, \\ P_{A \to aAb} &= \{A \to aAb, S \to aAb, S' \to aAb\}, \\ P_{A \to ab} &= \{A \to ab, S \to ab, S' \to ab\} \end{aligned}$$

und die gesamte Regelmenge ergibt sich als Vereinigung der vier vorstehenden Mengen.

Wir geben nun die Normalform an, die auf N. Chomsky zurückgeht und durch Kombination der vorstehenden Normalform gewonnen werden kann.

Satz 2.23 *Zu jeder kontextfreien Grammatik $G = (N, T, P, S)$ kann eine kontextfreie Grammatik $G' = (N', T, P', S)$ so konstruiert werden, dass P' nur Regeln der Form*

$$A \longrightarrow BC \quad \textit{und} \quad A \longrightarrow a \quad \textit{mit} \quad A, B, C \in N',\ a \in T$$

enthält, wobei $S \longrightarrow \lambda$ als Ausnahme zugelassen ist, falls S in keiner rechten Seite einer Regel aus P' vorkommt, und $L(G) = L(G')$ gilt.

Beweis. Durch Nacheinanderausführung der Konstruktionen in den Beweisen von Lemma 2.15, Satz 2.16, Lemma 2.18 und Lemma 2.21 erreichen wir eine Grammatik, die keine Regeln der Form $A \longrightarrow w$ mit $|w| > 2$ oder $w = \lambda$ bei $A \neq S$ und keine der Form $A \longrightarrow B$ mit Nichtterminalen A und B enthält. □

Wir geben nun noch eine Normalform für reguläre Grammatiken.

Satz 2.24 *Zu jeder regulären Grammatik $G = (N, T, P, S)$ kann eine reguläre Grammatik $G' = (N', T, P', S)$ der Größe $O(\#(N) \cdot k(G))$ in der Zeit $O(\#(N) \cdot k(G))$ so konstruiert werden, dass P' nur Regeln der Form*

$$A \longrightarrow aB \quad \textit{und} \quad A \longrightarrow a \quad \textit{mit} \quad A, B \in N',\ a \in T$$

enthält, wobei $S \to \lambda$ als Ausnahme zugelassen ist, falls P' keine Regel der Form $A \to aS$ enthält, und $L(G) = L(G')$ gilt.

Beweis. Entsprechend Lemma 2.18 und 2.21 können wir ohne Beschränkung der Allgemeinheit annehmen, dass die Regelmenge P der gegebenen Grammatik $G = (N, T, P, S)$ unter Beachtung der Ausnahmeregel $S \longrightarrow \lambda$ und den damit verbundenen Bedingungen nur Regeln der Form $A \longrightarrow wB$ und $A \longrightarrow w$ mit $A, B \in N$, $w \in T^+$ enthält.
Mit der Regel

$$p = A \longrightarrow a_1 a_2 \dots a_n B \text{ mit } a_1, a_2, \dots, a_n \in T$$

assozieren wir nun die Menge

$$N_p = \{B_{p,1}, B_{p,2}, \dots, B_{p,n-1}\}$$

zusätzlicher Nichtterminale und die Menge

$$\begin{aligned} P_p \;=\; \{&A \longrightarrow a_1 B_{p,1}, B_{p,1} \longrightarrow a_2 B_{p,2}, B_{p,2} \longrightarrow a_3 B_{p,3}, \dots \\ &\dots, B_{p,n-2} \longrightarrow a_{n-1} B_{p,n-1}, B_{p,n-1} \longrightarrow a_n B\} \end{aligned}$$

von Regeln. Für die Regel

$$q = A \longrightarrow a_1 a_2 \dots a_n \text{ mit } a_1, a_2, \dots, a_n \in T$$

setzen wir ebenfalls

$$N_q = \{B_{q,1}, B_{q,2}, \dots, B_{q,n-1}\}$$

und

$$\begin{aligned} P_q \;=\; \{&A \longrightarrow a_1 B_{q,1}, B_{q,1} \longrightarrow a_2 B_{q,2}, B_{q,2} \longrightarrow a_3 B_{q,3}, \dots \\ &\dots, B_{q,n-2} \longrightarrow a_{n-1} B_{q,n-1}, B_{q,n-1} \longrightarrow a_n\}. \end{aligned}$$

Hierbei seien alle neu eingeführten Symbole wieder paarweise voneinander verschieden. Wir definieren dann $G' = (N', T, P', S)$ durch

$$N' = N \cup \bigcup_{r \in P} N_r \quad \text{und} \quad P' = \bigcup_{r \in P} P_r \cup \overline{P},$$

wobei $\overline{P}$ erneut genau dann die leere Menge ist, wenn $S \longrightarrow \lambda$ nicht in P liegt und sonst nur aus dieser Regel besteht. Es ist leicht zu sehen, dass durch die Anwendung der Regeln aus P_r in der in der Definition angegebenen Reihenfolge zu einer Simulation der Anwendung von r führt, und umgekehrt jede Anwendung einer Regel $A \longrightarrow a_1 B_{r,1}$ die Simulation von r zur Folge hat. Daher gilt $L(G) = L(G')$.
Die Aussagen zur Komplexität können in Analogie zu den entsprechenden Aussagen über kontextfreie Grammatiken bewiesen werden. Wir überlassen die Details dem Leser. □

Wir geben nun zwei Folgerungen aus den in Satz 2.23 und Satz 2.24 gegebenen Normalformen an, die es uns dann gestatten, zu beweisen, dass gewisse Sprachen nicht kontextfrei bzw. nicht regulär sind.
Für reguläre Sprachen leistet der folgende Satz das Gewünschte.

Satz 2.25 *Sei L eine reguläre Sprache. Dann gibt es eine (von L abhängige) Konstante k derart, dass es zu jedem Wort $z \in L$ mit $|z| \geq k$ Wörter u, v, w gibt, die den folgenden Eigenschaften genügen:*
i) $z = uvw$,
ii) $|uv| \leq k$, $|v| > 0$, und
iii) $uv^i w \in L$ für alle $i \geq 0$.

Beweis. Wegen Satz 2.24 können wir annehmen, dass $L = L(G)$ für eine reguläre Grammatik $G = (N, T, P, S)$ gibt, deren Regelmenge P (mit Ausnahme von vielleicht $S \longrightarrow \lambda$) nur Regeln der Form $A \longrightarrow aB$ und $A \longrightarrow a$ mit $A, B \in N$ und $a \in T$ enthält. Wir setzen dann $k = |N| + 1$.
Aufgrund der Form der Regeln aus P gibt es für ein Wort

$$z = a_1 a_2 \dots a_n \text{ mit } a_i \in T \text{ für } 1 \leq i \leq n \text{ und } n \geq k$$

eine Ableitung

$$\begin{aligned} S = A_0 \implies & a_1 A_1 \Longrightarrow a_1 a_2 A_2 \Longrightarrow a_1 a_2 a_3 A_3 \Longrightarrow \dots \\ & \Longrightarrow a_1 a_2 \dots a_{n-1} A_{n-1} \Longrightarrow a_1 a_2 \dots a_{n-1} a_n = z. \end{aligned}$$

Dann muss die Menge $\{A_0, A_1, A_2, \dots, A_{n-1}\}$ wegen der Wahl von k ein Nichtterminal doppelt enthalten. Es sei $A = A_i = A_j$ mit $0 \leq i < j \leq n-1$ und für A_t mit $t \leq i$ gelte $A_t \neq A_s$ für $t \neq s$. Wir setzen

$$u = a_1 a_2 \dots a_i,\ v = a_{i+1} a_{i+2} \dots a_j \text{ und } w = a_{j+1} a_{j+2} \dots a_n.$$

Man sieht sofort, dass die Bedingungen i) und ii) erfüllt sind.
Mit den eingeführten Bezeichnungen erhält die obige Ableitung von z die folgende Form

$$S = A_0 \Longrightarrow^* uA \Longrightarrow^* uvA \Longrightarrow^* uvw = z,$$

und wir haben überdies für $i \geq 2$ die Ableitungen

$$S = A_0 \Longrightarrow^* uA \Longrightarrow^* uvA \Longrightarrow^* uvvA \Longrightarrow^* uvvvA \Longrightarrow^* \dots \Longrightarrow^* uv^i A \Longrightarrow^* uv^i w \in T^*$$

und für $i = 0$ die Ableitung $S \Longrightarrow^* uA \Longrightarrow^* uw \in T^*$. Hieraus folgt $uv^i w \in L(G) = L$ für $i \geq 0$, womit auch Bedingung iii) nachgewiesen ist. □

Wir benutzen die Aussage von Satz 2.25, um zu zeigen, dass die kontextfreie Sprache

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

aus Beispiel 2.5 nicht regulär ist.
Wir zeigen dies indirekt. Sei also angenommen, dass L regulär ist. Ferner sei k die Konstante aus Satz 2.25 und $z = a^k b^k$. Dann gibt es eine Zerlegung $z = uvw$ von z mit

$$|uv| \leq k,\ |v| > 0, \text{ und } uv^i w \in L \text{ für alle } i \geq 1. \qquad (*)$$

Aus den beiden erstgenannten Bedingungen und $z = uvw$ folgen

$$u = a^r,\ v = a^s \text{ und } w = a^{k-r-s}b^k \text{ mit } r \geq 0 \text{ und } s \geq 1.$$

Damit folgt

$$uv^i w = a^r a^{is} a^{k-r-s} b^k = a^{k+(i-1)s} b^k.$$

Wegen der Form der Wörter in L ist daher $uv^i w \notin L$ für $i \geq 2$. Dies widerspricht aber der oben abgeleiteten Aussage in $(*)$.
Somit haben wir das folgende Lemma bewiesen.

Lemma 2.26 $L = \{a^n b^n : n \geq 1\} \in \mathcal{L}(CF) \setminus \mathcal{L}(REG)$. □

Wir wollen nun den Begriff eines *Ableitungsbaumes* t für eine Satzform $w \neq \lambda$ einer kontextfreien Grammatik $G = (N, T, P, S)$ einführen, den wir im Beweis des folgenden Resultats benötigen, der aber auch sonst zur Veranschaulichung von Ableitungen geeignet ist. Wir benutzen dafür vollständige Induktion über die Anzahl n der Schritte zur Ableitung von w und wir setzen voraus, dass G in der Normalform aus Lemma 2.18 ist, also keine Regeln $A \longrightarrow \lambda$ enthält.
Wir werden t als Paar (K, E) beschreiben, wobei K die Menge der Knoten und E die der Kanten bezeichnet. Wir werden die Konstruktion so gestalten, dass S die Wurzel des Baumes sein wird und die Blätter beim Lesen von links nach rechts die Satzform w ergeben.
Sei $n = 0$. Dann handelt es sich um die „Ableitung“ $S \Longrightarrow^* S$ von $w = S$. Der Ableitungsbaum ist für $n = 0$ der Baum, der keine Kanten enthält und dessen einziger Knoten S ist. S ist dann sowohl Wurzel als auch Blatt.
Sei $n = 1$. Dann hat die Ableitung die Form $S \Longrightarrow w$, wobei die Regel $S \longrightarrow w$ angewendet wird. Sei $w = x_1 x_2 \ldots x_m$ mit $x_i \in N \cup T$. Dann wird die Menge K der Knoten von t durch die Symbole $S, x_1, x_2, \ldots, x_m$ gebildet, und die Menge E besteht aus allen Kanten (S, x_i), $1 \leq i \leq m$. S ist dabei die Wurzel des Baumes, und $x_1, x_2, \ldots, x_m$ sind die Blätter von t. Dabei ordnen wir die Kanten so an, dass wir $w = x_1 x_2 \ldots x_m$ erhalten, wenn wir die Blätter von links nach rechts lesen.
Sei $n \geq 2$. Dann gibt es eine Ableitung

$$S \Longrightarrow^* u = y_1 y_2 \ldots y_s A z_1 z_2 \ldots z_r \Longrightarrow y_1 y_2 \ldots y_s x_1 x_2 \ldots x_m z_1 z_2 \ldots z_r = w,$$

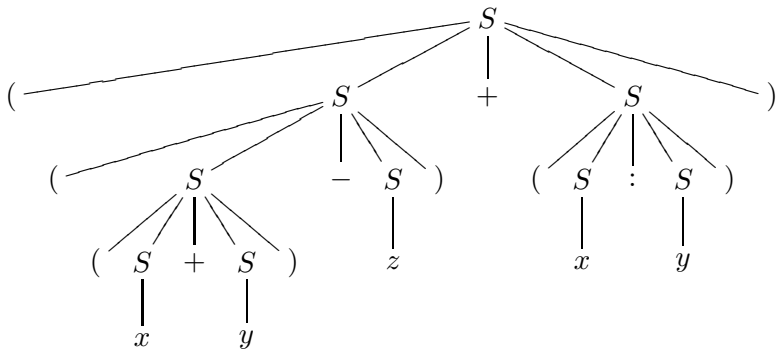
wobei $x_i, y_j, z_k \in N \cup T$ für $1 \leq i \leq m, 0 \leq j \leq s, 0 \leq k \leq r$ gilt. Die Ableitung $S \Longrightarrow^* u$ besteht dabei aus $n-1$ Schritten, und der zu ihr gehörende Ableitungsbaum $t' = (K', E')$

hat daher die Wurzel S und die Blätter ergeben von links nach rechts gelesen u. Wir konstruieren nun $t = (K, E)$ durch die Setzungen

$$K = K' \cup \{x_1, x_2, \ldots, x_m\} \quad \text{und} \quad E = E' \cup \{(A, x_i) : 1 \leq i \leq m\},$$

wobei wir die neuen Kanten wieder so anordnen, dass die Blätter von links nach rechts gelesen gerade w ergeben.

Zur Illustration geben wir den Ableitungsbaum für das Wort $(((x+y)-z)+(x:y))$, das von der in Beispiel 2.9 gegebenen Grammatik G_6 erzeugt wird. Dabei schreiben wir unter den Baum noch einmal die Blätter, um zu dokumentieren, dass sie von links nach rechts gelesen die zur Diskussion stehende Satzform ergeben.

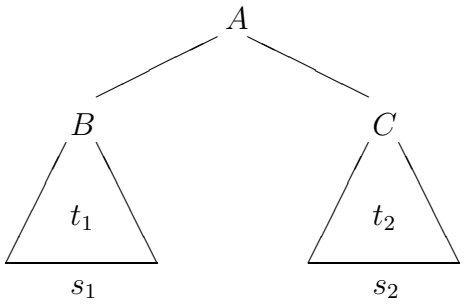


(((x + y) − z) + (x : y))

Wir geben jetzt ein Analogon zu Satz 2.25 für kontextfreie Sprachen.

Satz 2.27 *Sei L eine kontextfreie Sprache. Dann gibt es eine (von L abhängige) Konstante k derart, dass es zu jedem Wort $z \in L$ mit $|z| \geq k$ Wörter u, v, w, x, y gibt, die folgenden Eigenschaften genügen:*
i) $z = uvwxy$,
ii) $|vwx| \leq k$, $|v| + |x| > 0$, und
iii) $uv^iwx^iy \in L$ für alle $i \geq 0$.

Beweis. Wegen Satz 2.23 können wir annehmen, dass $L = L(G)$ für eine kontextfreie Grammatik $G = (N, T, P, S)$ in CHOMSKY-Normalform gilt. Es sei $n = |N|$. Dann setzen wir $k = 2^n$.
Sei $A \Longrightarrow^* s \in T^*$ eine Ableitung, deren zugehöriger Ableitungsbaum die Tiefe m hat. Wir zeigen zuerst mittels vollständiger Induktion über die Tiefe m des Ableitungsbaumes, dass dann $|s| < 2^m$ gilt.
$m = 1$. Die Ableitung kann nur aus einem Schritt bestehen und ist folglich aufgrund der CHOMSKY-Normalform $A \Longrightarrow \lambda$ oder $A \Longrightarrow a$ für ein $a \in T$. Dies bedeutet aber $s = \lambda$ oder $s = a$, woraus sofort $|s| \leq 1 < 2 = 2^1$ folgt. Damit ist der Induktionsanfang gezeigt.
Sei nun $A \Longrightarrow w$ eine Ableitung mit einem Ableitungsbaum t der Tiefe $m \geq 2$. Dann hat t wegen der CHOMSKY-Normalform die Form



wobei t_1 und t_2 Ableitungsbäume mit einer maximalen Tiefe $m-1$ sind und $s_1 s_2 = s$ gilt. Nach Induktionsannahme gilt dann

$$|s| = |s_1| + |s_2| < 2^{m-1} + 2^{m-1} = 2^m,$$

womit auch die Induktionsbehauptung gezeigt ist.
Wir benutzen die gerade bewiesene Aussage für Wörter $z \in L$ mit $|z| \geq k$. Sie liefert, dass der zu z gehörige Ableitungsbaum t' entsprechend der obigen Wahl von k eine Tiefe $m \geq n+1$ hat. Damit hat t' die Form gemäß Abbildung 2.2.

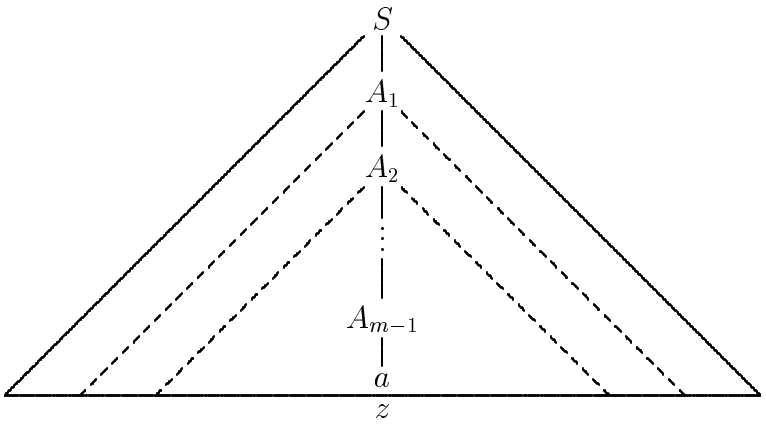


Abbildung 2.2:

Nun müssen wegen $m-1 \geq n$ mindestens zwei Elemente aus $\{S, A_1, A_2, \ldots, A_{m-1}\}$ identisch sein. Sei A dieses Nichtterminal. Damit ergibt sich für t' dann die Form gemäß Abbildung 2.3.
Dabei gilt $vx \neq \lambda$, da G eine Grammatik in CHOMSKY-Normalform ist. Weiterhin können wir ohne Beschränkung der Allgemeinheit annehmen, dass $|vwx| \leq k$ ist, da sonst im Ableitungsbaum zu $A \Longrightarrow^* vwx$ ein Weg existiert, auf dem ein Nichtterminal A' doppelt auftritt, und wir könnten dann mit A' anstelle von A argumentieren. Damit sind die Bedingungen i) und ii) nachgewiesen.
Ferner entnehmen wir dem Ableitungsbaum t' auch die Existenz der folgenden Ableitungen:

$$S \Longrightarrow^* uAy, \qquad A \Longrightarrow^* vAx, \qquad A \Longrightarrow^* w.$$

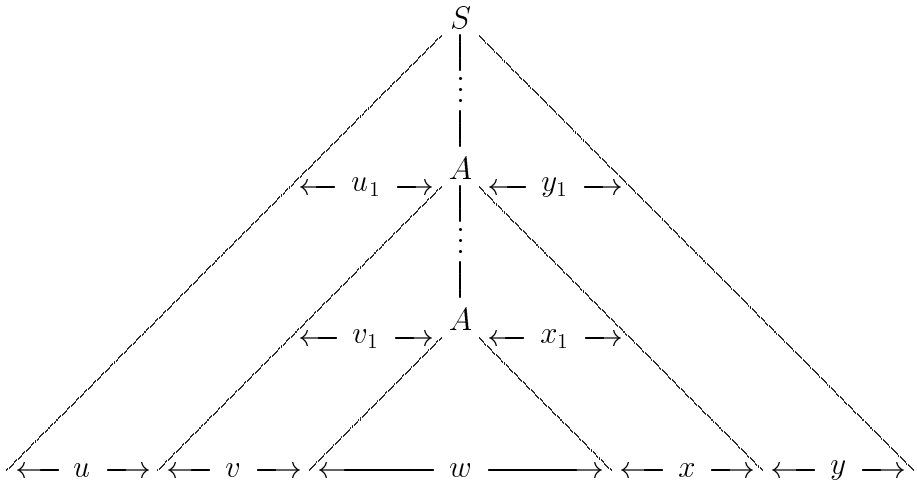


Abbildung 2.3:

Damit gibt es auch die Ableitung

$$S \Longrightarrow^* uAy \Longrightarrow^* uvAxy \Longrightarrow^* uvvAxxy \Longrightarrow^* uvvvAxxxy \Longrightarrow^* \ldots$$
$$\ldots \Longrightarrow^* uv^iAx^iy \Longrightarrow^* uv^iwx^iy$$

für $i \geq 0$ (für $i = 1$ entsteht gerade z). Da diese Ableitung zu einem Wort aus T^* führt, gilt $uv^iwx^iy \in L(G) = L$ für $i \geq 0$, womit auch Bedingung iii) nachgewiesen ist. □

Die in Satz 2.25 und 2.27 gegebenen Aussagen heißen *Schleifensätze* (oder auch *Pumping-Sätze*), da sie im Wesentlichen besagen, dass gewisse Ableitungen wie eine Schleife beliebig oft hintereinander ausgeführt werden können (oder einige Teilwörter „aufgepumpt“ werden können, z.B. v zu v^i).

Wir benutzen nun Satz 2.27, um ein zu Lemma 2.26 analoges Resultat herzuleiten.

Lemma 2.28 $L = \{a^nb^nc^n : n \geq 1\} \in \mathcal{L}(MON) \setminus \mathcal{L}(CF)$.

Beweis. Die Aussage $L \in \mathcal{L}(MON)$ folgt aus Beispiel 2.9.
Wir nehmen nun an, dass L kontextfrei ist. Nach Satz 2.27 gibt es dann eine Konstante k und für $z = a^kb^kc^k$ eine Zerlegung $z = uvwxy$ mit den in Satz 2.27 genannten Eigenschaften. Wir betrachten im Folgenden nur den Fall $v \neq \lambda$; die Überlegungen für $v = \lambda, x \neq \lambda$ verlaufen analog.
Wir unterscheiden die folgenden Fälle (wegen $|vwx| \leq k$ ist diese Fallunterscheidung vollständig):
Fall 1. $v = a^rb^s$ mit $r \geq 1, s \geq 0$. Wegen $|vwx| \leq k$ enthält vwx kein Vorkommen von c. Damit enthält uv^2wx^2y mindestens $k + r > k$ Vorkommen des Buchstaben a, aber nur k Vorkommen von c. Aufgrund der Form der Wörter in L, ergibt sich daraus $uv^2wx^2y \notin L$ im Widerspruch zur Eigenschaft iii) aus Satz 2.27.
Fall 2. $v = b^sc^t$ mit $s \geq 1, t \geq 0$. Dann enthält vwx kein Vorkommen von a, und daher gelten $|uv^2wx^2y|_a = k$ und $|uv^2wx^2y|_b \geq k + s > k$, womit sich in Analogie zum Fall 1 ein Widerspruch ergibt.
Fall 3. $v = c^t$ mit $t \geq 1$. Erneut enthält vwx kein Vorkommen von a, und daher gelten $|uv^2wx^2y|_a = k$ und $|uv^2wx^2y|_c \geq k + t > k$, womit sich in Analogie zum Fall 1 ein Widerspruch ergibt. □

Wir kombinieren nun die Aussagen der Lemmata 2.14, 2.26 und 2.28 und der Folgerungen 2.17 und 2.20 und erhalten den folgenden Satz. Man entnimmt ihm, dass die in Definition 2.12 eingeführten Sprachmengen eine Hierarchie bilden, die nach N. CHOMSKY benannt wird.

Satz 2.29 $\mathcal{L}(REG) \subset \mathcal{L}(CF) \subset \mathcal{L}(CS) = \mathcal{L}(MON) \subseteq \mathcal{L}(RE)$. □

Entsprechend Satz 2.29 ist also nur noch die Bestimmung der genauen Relation zwischen $\mathcal{L}(RE)$ und $\mathcal{L}(MON)$ offen. Wir werden die Klärung dieses Problems erst im Kapitel 2.4 herbeiführen.

2.2 Sprachen als akzeptierte Wortmengen

Zur Beschreibung von Sprachen haben wir im vorangehenden Abschnitt Grammatiken benutzt; bei diesen werden die Worte der Sprache mittels eines Ableitungsprozesses aus einem Startwort generiert. Ein grundsätzlich anderes Vorgehen liegt der Beschreibung von Sprachen durch Automaten zugrunde. Hier wird ein Wort als Eingabe verwendet und der Automat sagt „ja“, falls das Wort zu der Sprache gehört, und „nein“, falls das Wort nicht zu der Sprache gehört. Dies erinnert an die Funktionen, die mit Entscheidungsproblemen verbunden sind und im ersten Kapitel (insbesondere in Abschnitt 1.2) untersucht wurden. Wir werden hier eine Modifikation des dortigen Vorgehens betrachten, die sich von der in Kapitel 1 dadurch unterscheidet, dass die Antwort „ja“ oder „nein“ nicht der Ausgabe des Automaten entnommen wird, sondern mittels der Zustände gegeben wird, da die Ausgabe in diesem Zusammenhang nicht von Bedeutung ist.

2.2.1 TURING-Maschinen als Akzeptoren

Wir formalisieren den oben beschriebenen Ansatz.

Definition 2.30 *Eine akzeptierende* TURING-*Maschine M ist ein Sechstupel*

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

wobei X, Z, z_0, Q und δ wie in Definition 1.15 gegeben sind und $F \subseteq Q$ gilt. Die von M akzeptierte Sprache $T(M)$ wird durch

$$T(M) = \{w : w \in X^*,\ (\lambda, z_0, w) \models^* (v_1, q, v_2) \text{ für ein } q \in F\}$$

definiert.

Liegt ein Wort w in $T(M)$ für eine TURING-Maschine M, so sagen wir, dass w von M akzeptiert wird. F heißt die Menge der akzeptierenden Zustände.
Wir bemerken, dass es zwei Möglichkeiten gibt, die dazu führen, dass ein Wort w nicht akzeptiert wird: entweder die TURING-Maschine stoppt bei Eingabe von w nicht, oder sie stoppt in einem Zustand $q \in Q \setminus F$.

Beispiel 2.31 Wir betrachten Modifikationen M_1' und M_2' der TURING-Maschinen M_1 und M_2 aus Beispiel 1.19.
M_1 merkt sich den ersten Buchstaben, löscht diesen und fügt ihn ans Ende des Wortes an. Bei M_1' betrachten wir statt eines Stopzustandes q in M_1 zwei Stopzustände q_a und q_b in Abhängigkeit davon, ob sich a oder b gemerkt und an das Wort angefügt wurde. Formal ergibt dies die TURING-Maschine

$$M_1' = (\{a,b\}, \{z_0, z_a, z_b, q_a, q_b\}, z_0, \{q_a, q_b\}, \delta', \{q_a\})$$

mit δ aus Abb. 2.4. M_1' erreicht wie M_1 aus Beispiel 1.19 stets einen Stopzustand, akzep-

δ	z_0	z_a	z_b
$*$	$(q, *, N)$	(q_a, a, N)	(q_b, b, N)
a	$(z_a, *, R)$	(z_a, a, R)	(z_b, a, R)
b	$(z_b, *, R)$	(z_a, b, R)	(z_b, b, R)

Abbildung 2.4: Überführungsfunktion von M_1'

tiert aber nur die Wörter, bei denen sich die Maschine a gemerkt (und schließlich auf das Band geschrieben) hat. Folglich erhalten wir

$$T(M_1') = \{aw \mid w \in \{a,b\}^*\}\,.$$

Um M_2' aus M_2 zu erhalten legen wir nur die Menge der akzeptierenden Zustände fest. Wir wollen in jedem Stoppzustand akzeptieren, d.h. wir setzen

$$M_2' = (\{a,b\}, \{z_0, z_1, q\}, z_0, \{q\}, \delta, \{q\}),$$

wobei δ wie bei M_2 aus Beispiel 1.19 gegeben sei. Entsprechend den Betrachtungen für M_2 in Beispiel 1.19 erhalten wir

$$T(M_2') = \{w : w \in \{a,b\}^*, |w| \; \textit{ungerade}\}.$$

Bei der Behandlung von TURING-Machinen im Abschnitt 1.1.2 haben wir eine Normalform hergeleitet, bei der nur ein Stoppzustand benutzt wurde. Wir wollen nun ein analoges Resultat für akzeptierende TURING-Maschinen angeben.

Lemma 2.32 *Zu jeder akzeptierenden* TURING*-Maschine M gibt es eine akzeptierende* TURING*-Maschine M', deren Menge der Stopzustände mit der Menge der akzeptierenden Zustände übereinstimmt und für die $T(M) = T(M')$ gilt. Dabei kann die Menge der Stoppzustände von M' einelementig gewählt werden.*

Beweis. Sei $M = (X, Z, z_0, Q, \delta, F)$ eine TURING-Maschine. Wir konstruieren aus M die TURING-Maschine $M' = (X, Z', z_0, \{q\}, \delta', \{q\})$ mit

$$\begin{aligned}
Z' &= Z \cup \{q\} \text{ wobei } q \notin Z,\\
\delta'(z,x) &= \delta(z,x) \text{ für } z \in Z \setminus Q,\\
\delta'(z,x) &= (z,x,N) \text{ für } z \in Q \setminus F,\ x \in X \cup \{*\},\\
\delta'(z,x) &= (q,x,N) \text{ für } z \in F.
\end{aligned}$$

Entsprechend diesen Setzungen
- verhält sich M' wie M solange M keinen seiner Stopzustände erreicht hat,
- geht M' in eine Schleife, wenn M einen nichtakzeptierenden Stopzustand erreicht hat,
- stoppt M' nach einem weiteren Schritt, wenn M einen akzeptierenden Stopzustand erreicht hat.
Hieraus folgt $T(M') = T(M)$ sofort. □

Aus Lemma 2.32 folgt sofort der folgende Satz, der die Verbindung zu den Betrachtungen aus Abschnitt 1.1.2 herstellt.

Satz 2.33 *Eine Sprache wird genau dann von einer* TURING*-Maschine akzeptiert, wenn sie Definitionsbereich einer* TURING*-berechenbaren Funktion ist.* □

Lemma 2.32 legt die Frage nahe, warum in der Definition 2.30 der akzeptierenden TURING-Maschine die Menge der akzeptieenden Zustände eingeführt wurde. Beim Beweis der Normalform wurden die nichtakzeptierenden Stoppzustände einfach in Zustände überführt, in denen die Maschine nicht stoppt. Dies verbietet sich aber dann, wenn – wie bei anderen Typen von Automaten – stets eine Stoppsituation eintritt oder man für jede Eingabe eine Antwort braucht. In diesen Fällen muss dann die Akzeptanz bzw. Nichtakzeptanz allein mittels der Zustände geschehen können. Die akzeptierenden Zustände entsprechen dem „ja“ und die nichtakzeptierenden dem „nein“.
Fordert man stets ein Erreichen eines Stoppzustandes bei TURING-Maschinen kommt man zum Begriff der rekursiven Sprache.

Definition 2.34 *Eine Sprache $L \subseteq X^*$ heißt rekursiv, falls es eine akzeptierende* TURING*-Maschine $M = (X, Z, z_0, Q, \delta, F)$ gibt, die auf jeder Eingabe stoppt und L akzeptiert.*

Satz 2.35 *Eine Sprache $L \subseteq X^*$ ist genau dann rekursiv, wenn sowohl L als auch $X^* \setminus L$ von* TURING*-Maschinen akzeptiert werden.*

Beweis. Es sei zuerst L eine rekursive Sprache. Dann gibt es eine akzeptierende TURING-Maschine $M = (X, Z, z_0, Q, \delta, F)$, die L akzeptiert und auf jeder Eingabe stoppt. Die akzeptierende TURING-Maschine $M' = (X, Z, z_0, Q, \delta, Q \setminus F)$ akzeptiert dann offenbar genau die Eingaben, die von M verworfen werden. Damit gilt $T(M') = X^* \setminus L$.
Es seien nun L und $X^* \setminus L$ von den akzeptierenden TURING-Maschinen N und N' akzeptiert. Wir nehmen ohne Beschränkung der Allgemeinheit an, dass N und N' in der Normalform aus Lemma 2.32 gegeben sind. Wir betrachten dann die akzeptierende TURING-Maschine N'', die wie folgt arbeitet: Zuerst schreibt N'' eine Kopie des Eingabewortes hinter die Eingabe auf das Band. Im Folgenden wird auf dem ersten Wort N und auf dem zweiten Wort N' simuliert. N'' führt diese Simulationsschritte abwechselnd aus und stoppt, falls ein Stoppzustand von N bzw. N' erreicht wird. Dabei fungieren die Stoppzustände von N als akzeptierende Stoppzustände und die von N' als ablehnende Stoppzustände. Da entweder $w \in X$ oder $w \in X^* \setminus L$ gilt, erreicht N'' bei Eingabe von w im ersten Fall einen Stoppzustand von N und im zweiten Fall einen Stoppzustand aus N'. Damit wird in jedem Fall ein Stoppzustand erreicht und L akzeptiert. Dies bedeutet, dass L rekursiv ist. □

Satz 2.36 *Für eine rekursive Sprache L ist die charakteristische Funktion*

$$\varphi_L(x) = \begin{cases} 0 & x \notin L \\ 1 & x \in L \end{cases}$$

von L algorithmisch berechenbar.

Beweis. Es seien L eine rekursive Menge und M die akzeptierende TURING-Maschine, die auf jeder Eingabe stoppt und L akzeptiert. Wir betrachten, die TURING-Maschine M', die zuerst M simuliert und bei Erreichen eines akzeptierenden Stoppzustandes den gesamten Bandinhalt durch eine 1 ersetzt bzw. bei Erreichen eines ablehnenden Stoppzustandes den gesamten Bandinhalt durch eine 0 ersetzt. Offenbar berechnet M' die charakteristische Funktion von L. □

Satz 2.37 *Die Menge der rekursiven Sprachen ist echt in der Menge der von* TURING-*Maschinen akzeptierbaren Sprachen enthalten.*

Beweis. Mit den Bezeichnungen aus dem Beweis von Satz 1.28 definieren wir die Menge

$$L_{halt} = \{w : w \in \{0,1\}^*, w = w_M \text{ für eine TURING-Maschine } M, f_M(w_M) \text{ ist definiert}\},$$

die im Wesentlichen dem Halteproblem von TURING-Maschinen entspricht, denn sie besteht aus allen Beschreibungen von TURING-Maschinen, die auf ihrer Beschreibung stoppen. Da wegen der Unentscheidbarkeit des Halteproblems die charakteristische Funktion von L_{halt} nicht berechenbar ist, ist L_{halt} nach Satz 2.36 nicht rekursiv.
L_{halt} wird aber von der folgenden TURING-Maschine N akzeptiert, deren Arbeitsweise wir nur informell beschreiben (die exakte Beschreibung von N bleibt dem Leser überlassen). N stellt zuerst fest, ob das Wort w auf dem Band die Kodierung w_M einer TURING-Maschine M ist. Bei negativer Antwort geht N in eine Schleife und stoppt nicht. Bei positiver Antwort kopiert N das Eingabewort $w = w_M$ ein zweites Mal auf das Band. Nun simuliert N auf dem ersten Wort $w = w_M$ die Arbeit von M, wobei N die erforderlichen Informationen über M aus der zweiten Kopie von $w = w_M$ auf dem Band bezieht. N stoppt, falls M stoppt. Daher stoppt N genau dann, wenn die Eingabe w Kodierung w_M einer TURING-Maschine M ist und $f_M(w_M)$ definiert ist. Somit gilt $T(N) = L_{halt}$. □

Die nächsten Aussagen geben das Verhältnis der von TURING-Maschinen akzeptierten Sprachen zu den im vorhergehenden Abschnitt untersuchten Sprachen, die von Grammatiken erzeugt werden, an.

Lemma 2.38 *Zu jeder akzeptierenden* TURING*-Maschine M gibt es eine Regelgrammatik G mit $L(G) = T(M)$.*

Beweis. Es sei die TURING-Maschine $M = (X, Z, z_0, Q, \delta, F)$ gegeben. Wir konstruieren zuerst die TURING-Maschine $M'' = (X, Z'', z_0, \{q''\}, \delta'', \{q''\})$ entsprechend dem Beweis von Lemma 2.32 und aus M'' die TURING-Maschine $M' = (X \cup \{\S, \#\}, Z', z_0', \{q'\}, \delta', \{q'\})$ entsprechend dem Beweis von Lemma 1.21. Jede Folge von Konfigurationen von M' hat die folgende Form:

$$(*) \quad K_0 = (\lambda, z_0', w) \models^* K_1 = (\S, z_0, w\#) \models^* K_2 = (\S v_1, q, v_2\#) \models^* K_3 = (\lambda, q', v),$$

wobei $v_1v_2 = *^r v *^s$ gilt. Außerdem ist sofort zu sehen, dass $T(M) = T(M')$ gilt. Ferner nehmen wir ohne Beschränkung der Allgemeinheit an, dass $X \cap Z' = \emptyset$ und $\S, \#, * \notin Z'$ gelten. Daher ist es möglich, eine Konfiguration (w_1, z, w_2) auch als w_1zw_2 zu beschreiben. Wir werden jetzt eine Regelgrammatik $G = (N, T, P, S)$ so konstruieren, dass im Wesentlichen $w_1zw_2 \models w_1'z'w_2'$ genau dann gilt, wenn $w_1'z'w_2' \Longrightarrow w_1zw_2$ gilt, d.h. wir werden die Überführungen in M' schrittweise in umgekehrter Reihenfolge simulieren. Dadurch wird erreicht, dass die Grammatik in einer Ableitung eine „Endkonfiguration“ in eine „Anfangskonfiguration“ überführt, aus der durch „Streichen“ des Zustands das akzeptierte Wort entsteht.
Wir geben nun die formale Definition von G. Dazu setzen wir

$$\begin{aligned} N &= Z' \cup \{\S, \#, *, S, S', A\}, \\ T &= X \end{aligned}$$

und definieren P als die Menge aller Regeln der folgenden Formen:

$$\begin{aligned} (i) \quad & S \longrightarrow \S S'\#, \\ & S' \longrightarrow xS', S' \longrightarrow S'x \quad \text{für } x \in X \cup \{*\}, \\ & S' \longrightarrow q \quad \text{für } q \in Q \end{aligned}$$

(mittels dieser Regeln wird eine Ableitung $S \Longrightarrow^* \S v_1 q v_2 \#$ realisiert und damit die Beschreibung der Konfiguration K_2 aus $(*)$ erreicht),

$$\begin{aligned} (ii) \quad & z'ab' \longrightarrow azb \quad \text{für } z, z' \in Z', a, b, b' \in X \cup \{*\}, \delta'(z, b) = (z', b', L), \\ & z'b' \longrightarrow zb \quad \text{für } z, z' \in Z', b, b' \in X \cup \{*\}, \delta'(z, b) = (z', b', N), \\ & b'z' \longrightarrow zb \quad \text{für } z, z' \in Z, b, b' \in X \cup \{*\}, \delta'(z, b) = (z', b', R) \end{aligned}$$

(dies ist eine direkte Simulation einer inversen Überführung, bei der die TURING-Maschine über einem Element aus $X \cup \{*\}$ stand),

$$\begin{aligned} (iv) \quad & \S z_0 \longrightarrow A, \\ & Aa \longrightarrow aA \quad \text{für } a \in X, \\ & A\# \longrightarrow \lambda \end{aligned}$$

(durch diese Regeln wird aus einem Wort der Form $\S z_0 w\#$ mit $w \in X^*$ das Wort w abgeleitet).
Aufgrund der im Anschluss an die Regeln gegebenen Ausführungen ist klar, dass jede Ableitung in G die Form

$$(**) \quad S \Longrightarrow^* u_1 = \S v_1 q v_2 \# \Longrightarrow^* u_2 = \S z_0 w \# \Longrightarrow^* w$$

hat, wobei die Ableitung $u_1 \Longrightarrow^* u_2$ durch schrittweise Simulation der Überführungsschritte von $K_1 \models^* K_2$ in umgekehrter Reihenfolge erhalten wird.
Damit ist gezeigt, dass es eine Ableitung $(**)$ in G genau dann gibt, wenn es auch eine Überführung $(*)$ gibt. Hieraus folgt sofort, dass $w \in L(G)$ genau dann gilt, wenn auch $w \in T(M)$ gilt. Somit erhalten wir $L(G) = T(M)$. □

Die Idee des Beweises von Satz 2.38 besteht im Wesentlichen darin, dass die Überführungen $w_1zw_2 \models w_1'z'w_2'$ der TURING-Maschine in umgekehrter Reihenfolge durch einen direkten Ableitungsschritt $w_1'z'w_2' \Longrightarrow w_1zw_2$ simuliert werden. Es ist naheliegend, diesen Gedanken auch dafür zu verwenden, um zu zeigen, dass jede von einer Regelgrammatik erzeugte Sprache von einer TURING-Maschine akzeptiert wird. Dabei tritt aber die Schwierigkeit, dass eine Satzform einer Grammatik durch Anwendung verschiedener Regeln auf verschiedene Satzformen entstanden sein kann. Bei der Umkehrung erfordert dies, dass aus einer Konfiguration mehrere verschiedene Konfigurationen entstehen können müssen. Um diese Schwierigkeit zu überwinden, definieren wir daher eine nichtdeterministische Variante der TURING-Maschine, bei der auf eine Konfiguration dann mehrere Konfigurationen folgen können.

Definition 2.39 *Eine nichtdeterministische* TURING-*Maschine M ist ein Sechstupel*

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

wobei X, Z, z_0, Q und F wie bei einer (akzeptierenden deterministischen) TURING-*Maschine definiert sind und τ eine Funktion*

$$\tau : (Z \setminus Q) \times (X \cup \{*\}) \to 2^{Z \times (X \cup \{*\}) \times \{R,N,L\}}$$

ist.

Entsprechend dieser Definition besteht $\tau(z, x)$ aus einer Menge von Elementen der Form (z', x', r) mit $z' \in Z, x' \in (X \cup \{*\}), r \in \{R, L, N\}$.
Die in Definition 1.15 angegebenene TURING-Maschine ist der Spezialfall, dass die Menge $\tau(z, x)$ nur aus dem Element $\delta(z, x)$ besteht.
Wir definieren nun die Konfiguration einer nichtdeterministischen TURING-Maschine wie bei einer (deterministischen) TURING-Maschine (Definition 1.16) und die Relation $K_1 \models K_2$ wie in Definition 1.17, wobei wir nur $\delta(z, x) = (z', x', r)$ durch die Forderung $(z', x', r) \in \tau(z, x)$ ersetzen.
Hieraus folgt offensichtlich, dass aus einer Konfiguration K_1 mehrere Konfigurationen K_2 erzeugt werden können, wenn $\tau(z, x)$ mehrere Elemente enthält. Wir definieren die von einer nichtdeterministischen TURING-Maschine akzeptierte Wortmenge in Analogie zu Definition 2.30.

Definition 2.40 *Es sei $M = (X, Z, z_0, Q, \tau, F)$ eine nichtdeterministische* TURING-*Maschine wie in Definition 2.39. Die von M akzeptierte Sprache $T(M)$ wird durch*

$$T(M) = \{w : w \in X^*,\ (\lambda, z_0, w) \models^* (v_1, q, v_2) \text{ für ein } q \in F\}$$

definiert.

Wir geben nun ein Beispiel.

Beispiel 2.41 Wir betrachten die nichtdeterministische TURING-Maschine

$$M = (\{a, b\}, \{z_0, z_0', z_0'', z_{0,2}z_{1,2}, z_2, z_2', z_2'', z_{0,3}z_{1,3}, z_{2,3}, z_3, z_3', z_3'', q\}, z_0, \{q\}, \tau, \{q\})$$

mit

$$\begin{aligned}
\tau(z_0, a) &= \{(z_0, a, R)\},\\
\tau(z_0, b) &= \{(z_0, b, N)\},\\
\tau(z_0, *) &= \{(z_0', *, L)\},\\
\tau(z_0', a) &= \{(z_0', a, L)\},\\
\tau(z_0', *) &= \{(z_0'', *, R)\}
\end{aligned}$$

(die Maschine entscheidet, ob auf dem Band nur as stehen; ist dies nicht der Fall, so geht sie in eine Schleife),

$$\begin{aligned}
\tau(z_0'', *) &= \{(z_o'', *, N)\},\\
\tau(z_0'', x) &= \{(z_2, x, N), (z_3, x, N)\} \quad \text{für } x \in \{a, b\}
\end{aligned}$$

(steht kein Buchstabe auf dem Band, so geht die Maschine in eine Schleife; ansonsten entscheidet sich die Maschine nichtdeterministisch zwischen zwei Varianten, die im Index 2 bzw. 3 festgelegt sind),

$$\begin{aligned}
\tau(z_i, a) &= \{(z_i', a, R)\} \quad \text{für } i \in \{2, 3\},\\
\tau(z_i, b) &= \{(z_i, b, R)\} \quad \text{für } i \in \{2, 3\},\\
\tau(z_i', a) &= \{(z_i'', a, R)\} \quad \text{für } i \in \{2, 3\},\\
\tau(z_i', b) &= \{(z_i', b, R)\} \quad \text{für } i \in \{2, 3\},\\
\tau(z_i'', x) &= \{(z_i'', x, R)\} \quad \text{für } x \in \{a, b\},\\
\tau(z_i, *) &= \{(z_i, *, N)\} \quad \text{für } i \in \{2, 3\},\\
\tau(z_i', *) &= \{(q, *, N)\} \quad \text{für } i \in \{2, 3\},\\
\tau(z_i'', *) &= \{(z_{0,i}, *, L)\} \quad \text{für } i \in \{2, 3\}
\end{aligned}$$

(die Maschine liest von links nach rechts das Wort auf dem Band und prüft, ob es kein a, genau ein a oder mindestens zwei a enthält, wozu die Zustände z_i, z_i' und z_i'' dienen; ist kein a vorhanden, so geht die Maschine in eine Schleife und stoppt nicht; ist genau ein a vorhanden, so akzeptiert die Maschine die Eingabe; sind mindestens zwei a vorhanden, so wird die nächste Phase eingeleitet),

$$\begin{aligned}
\tau(z_{0,2}, a) &= \{(z_{1,2}, b, L)\},\\
\tau(z_{1,2}, a) &= \{(z_{0,2}, a, L)\},\\
\tau(z_{j,2}, b) &= \{(z_{i,2}, b, L)\} \quad \text{für } j \in \{0, 1\}
\end{aligned}$$

(die Maschine liest das Wort auf dem Band von rechts nach links; abwechselnd wird dabei ein a durch ein b ersetzt bzw. stehengelassen; somit erfolgt eine Halbierung der Anzahl der Vorkommen von a; im Zustand $z_{j,2}$ gibt j die Anzahl der gelesenen a modulo 2 an),

$$\begin{aligned}
\tau(z_{0,3}, a) &= \{(z_{1,2}, b, L)\},\\
\tau(z_{1,3}, a) &= \{(z_{2,3}, b, L)\},\\
\tau(z_{2,3}, a) &= \{(z_{0,3}, a, L)\},\\
\tau(z_{j,3}, b) &= \{(z_{j,3}, b, L)\} \quad \text{für } j \in \{0, 1, 2\}
\end{aligned}$$

(analog erfolgt eine Drittelung der Anzahl der Vorkommen von a; in $z_{j,3}$ gibt j die Anzahl der gelesenen a modulo 3 an),

$$\begin{aligned}\tau(z_{0,i},*) &= \{(z_i,*,R)\} \quad \text{für } i \in \{2,3\},\\ \tau(z_{j,i},*) &= \{(z_{j,i},*,N)\} \quad \text{für } j \in \{1,2\}\ , i \in \{2,3\}\end{aligned}$$

(ist die Halbierung bzw. Drittelung ganzzahlig möglich, d.h. $z_{0,2}$ bzw. $z_{0,3}$ liegt vor, so wird der Gesamtprozess iteriert, anderenfalls geht die Maschine in eine Schleife und akzeptiert daher nicht).
Nach diesen Erklärungen ist klar, dass die TURING-Maschine nur solche Wörter akzeptiert bei denen iterierte Halbierung bzw. Drittelung der Anzahl der Vorkommen von a zu einem Wort auf dem Band führt, dass genau ein a enthält und akzeptiert dann. Somit ergibt sich als akzeptierte Sprache

$$T(M) = \{w : \#_a(w) = 2^n \text{ oder } \#_a(w) = 3^n \text{ für ein } n \geq 0\}.$$

Mit diesem Typ von TURING-Maschinen sind wir nun in der Lage, die Umkehrung von Lemma 2.38 zu beweisen.

Lemma 2.42 *Zu jeder Regelgrammatik G gibt es eine nichtdeterministische* TURING-*Maschine M mit $T(M) = L(G)$.*

Beweis. Wir geben hier keinen detaillierten vollständigen Beweis, sondern erläutern nur die wesentliche Idee der Konstruktion.
Es sei die Grammatik $G = (N, T, P, S)$ gegeben. Wir konstruieren nun eine nichtdeterministische TURING-Maschine M mit dem Eingabealphabet $N \cup T \cup \{\S\}$ und folgender Arbeitsweise auf einer Eingabe w.

1. Da nur Wörter über T akzeptiert werden sollen, testet M als erstes, ob w in T^* liegt. Ist dies nicht der Fall, so geht M in eine Schleife (und akzeptiert daher w nicht); gilt dagegen $w \in T^*$, so erreicht M die Konfiguration (λ, z_1, w), bei der der Zustand z_1 den Beginn der zweiten Phase andeutet.

2. In dieser Phase testet M, ob auf dem Band nur S steht. Ist dies der Fall, so stoppt M; steht nicht nur S auf dem Band, erreicht die Maschine die Konfiguration (λ, z_2, w), bei der z_2 den Beginn der Phase 3 markiert.

3. Diese Phase dient der Simulation eines Ableitungsschrittes, wobei wir wie bereits im Beweis von Lemma 2.38 die Richtung umkehren, d.h. wir simulieren die Anwendung einer Regel $u \longrightarrow u'$ und damit die Ableitung

   $$xuy \Longrightarrow xu'y = w$$

   durch den Übergang

   $$(\lambda, z_2, w) = (\lambda, z_2, xu'y) \models^* (\lambda, z_1, xuy).$$

   Hierzu bestimmt M zuerst nichtdeterministisch eine Stelle, an der die Anwendung der Regel $p = u \longrightarrow u'$ simuliert werden soll, d.h. M erreicht die Konfiguration

(x, z_p, x'), bei der z_p den Beginn der Simulation von p markiert. M testet nun, ob das Wort u' hinter x auf dem Band steht. Ist dies nicht der Fall, so geht M in eine Schleife. Ist dies aber der Fall arbeitet M wie folgt. Falls $|u'| - |u| = m \geq 0$ ist, ersetzt M das Wort u' durch $u\S^m$, wodurch $(xu\S^m, z'_p, y)$ entsteht, verschiebt y um m Zellen nach links und kehrt an den Wortanfang und in den Zustand z_1 zurück. Falls $|u| - |u'| = m' > 0$ ist, verschiebt M zuerst das hinter u' stehende Wort y um m' Zellen nach rechts, schreibt in die entstehende Lücke $\S^{m'}$, ersetzt dann $u'\S^{m'}$ durch u und kehrt dann an den Wortanfang und in den Zustand z_1 zurück. Damit entsteht jeweils die Konfiguration (λ, z_1, xuy) aus $(\lambda, z_2, xu'y)$, womit die Simulation abgeschlossen ist.

Danach wird erneut Phase 2 gestartet.

Entsprechend dieser Arbeitsweise wird jede Ableitung

$$S \Longrightarrow w_1 \Longrightarrow w_2 \Longrightarrow \ldots \Longrightarrow w_{n-1} \Longrightarrow w_n = w$$

durch

$$\begin{aligned}(\lambda, z_0, w_n) \quad &\models^* \quad (\lambda, z_1, w_n) \models^* (\lambda, z_2, w_n)\\ &\models^* \quad (\lambda, z_1, w_{n-1}) \models^* (\lambda, z_2, w_{n-1})\\ &\models^* \quad \ldots \models^* (\lambda, z_1, w_2) \models^* (\lambda, z_2, w_2)\\ &\models^* \quad (\lambda, z_1, w_1) \models^* (\lambda, z_2, w_1)\\ &\models^* \quad (\lambda, z_1, S) \models^* (\lambda, q, S)\end{aligned}$$

simuliert. Weiterhin erreicht M nur einen Endzustand, wenn M eine Ableitung simuliert, da M sonst in eine Schleife geht. Setzen wir nun noch die Menge der akzeptierenden Zustände als die Menge aller Stoppzustände, so gilt $T(M) = L(G)$. □

Satz 2.43 *Die folgenden Aussagen sind äquivalent:*
i) L wird von einer Regelgrammatik erzeugt.
ii) L wird von einer deterministischen TURING*-Maschine akzeptiert.*
iii) L wird von einer nichtdeterministischen TURING*-Maschine akzeptiert.*

Beweis. Wegen Lemma 2.38 und 2.42 reicht es zu zeigen, dass jede Sprache, die von einer nichtdeterministischen TURING-Sprache akzeptiert wird auch von einer (deterministischen TURING-Maschine akzeptiert wird.
Wir geben hier erneut keinen vollständigen formalen Beweis sondern nur die Beweisidee. Es seien $M = (X, Z, z_0, Q, \tau, F)$ eine nichtdeterministische TURING-Maschine und

$$n = max\{\#(\tau(z, x)) : z \in Z, x \in X \cup \{*\}\} \quad \text{und} \quad N = \{1, 2, ..., n\}.$$

In der Menge der Folgen über N führen wir eine Ordnung ein, bei der zuerst nach der Länge und bei gleicher Länge lexikographisch sortiert wird. $NFOLGE$ sei die Funktion, bei der der Folge x die auf x entsprechend der Ordnung folgende Folge $NFOLGE(x)$ zugeordnet wird. Wir sagen, dass die Folge $d_1 d_2 ... d_r$ durch M abgearbeitet wird, wenn bei Vorliegen des Zustandes z und Lesen von x nach $i - 1$ Schritten im i-ten Schritt das d_i-te Element aus $\tau(z, x)$ benutzt wird, soweit es vorhanden ist.

Wir betrachten nun die TURING-Maschine M', die wie folgt auf der Eingabe w arbeitet (dabei ist \$ ein gesondertes Trennzeichen):

	Programm	Bandinhalt
	BEGIN	w
	$f := \lambda$	
	Schreibe f hinter das Wort auf dem Band	$w\$f$
A:	Schreibe hinter das Wort auf dem Band eine Kopie von w und zwei Kopien von $f' = NFOLGE(f)$	$w\$f\$w\$f'\$f'$
	Lösche $f\$$	$w\$w\$f'\$f'$
	Arbeite f' auf erstem w ab (dabei wird $f'\$$ gelöscht)	$w'\$w\$f'$
	(falls das d_i-te Element nicht vorhanden ist, lösche $w\$$ und $f'\$$ und **GOTO** A)	$(w\$f')$
	IF erreichter Zustand $z \notin Q$ **THEN** lösche $w'\$$ und **GOTO** A	$w\$f'$
	END	

Jeder einzelne dieser Schritte ist deterministisch realisierbar, und durch spezielle Komponenten in den Zuständen kann deterministisch der Schritt, in dem sich M' befindet, gespeichert werden.
Verwenden wir F auch als Menge der akzeptierenden Zustände von M', so akzeptiert M' entsprechend ihrer Arbeitsweise genau dann ein Wort w, wenn es eine Folge $d_1 d_2 \dots d_r$ gibt, bei deren Abarbeitung M das Wort w akzeptiert. Daher gilt $T(M') = T(M)$. □

Satz 2.43 kann auch wie folgt formuliert werden: *Eine Sprache L wird genau dann von einer (nichtdeterministischen)* TURING*-Maschine akzeptiert, wenn $L \in \mathcal{L}(RE)$ gilt.*

Durch Kombination dieser Formulierung mit Satz 2.33 und den Übungsaufgaben 12und 13 zu Abschnitt 1 wird die Bezeichnung RE als Abkürzung von rekursiv-aufzählbar (engl. recursively enumerable) als sinnvoll nachgewiesen.

Wir wollen nun ein Analogon zu Satz 2.43 für kontextabhängige Sprachen geben. Wegen Folgerung 2.17 können wir eine monotone Grammatik zur Erzeugung der Sprache verwenden. Wir werfen nun einen Blick auf den Beweis von Lemma 2.42. Da bei monotonen Grammatiken für alle Regeln $u \to u'$ die Beziehung $|u| \leq |u'|$ gilt, wird bei der von der TURING-Maschine in umgekehrter Richtung durchgeführten Simulation der Übergang $w_1 u' w_2 \models w_1 u w_2$ zu einer Verkürzung des Bandinhaltes führen. Folglich stehen auf dem Band nur Wörter, deren Länge höchstens die Länge des Wortes ist, das zu Beginn auf dem Band steht. Außerdem bewegt sich der Kopf der Maschine immer nur auf Zellen, in denen ein Buchstabe des Eingabealphabetes steht, oder auf den mit $*$ gefüllten Zellen direkt vor oder direkt hinter dem Wort (dies ist nötig, um den Wortanfang oder das Wortende zu finden). Damit ist durch die um 2 erhöhte Länge des Wortes, das zu Beginn auf dem Band steht, eine obere Schranke für die Anzahl der Zellen der TURING-Maschine, über denen sich während der Arbeit der Kopf befinden kann. Dies führt zur folgenden Definition.

Definition 2.44 *Ein linear beschränkter Automat ist eine nichtdeterministische* TURING*-Maschine $M = (X, Z, z_0, Q, \delta, F)$, deren Kopf sich während der Abarbeitung der Eingabe $w \in X^*$ höchstens über $|w| + 2$ verschiedenen Zellen befindet.*

Aus den vorstehend gemachten Ausführungen folgt sofort, dass jede von einer monotonen oder kontextabhängigen Grammatik erzeugte Sprache von einem linear beschränkten Automaten akzeptiert wird. Ohne Beweis geben wir an, dass auch die Umkehrung gilt. Damit erhalten wir den folgenden Satz.

Satz 2.45 *Eine Sprache ist genau dann kontextabhängig, wenn sie von einem linear beschränkten Automaten akzeptiert werden kann.* □

Für TURING-Maschinen haben wir gezeigt, dass deterministische und nichtdeterministische Varianten die gleiche Menge von Sprachen akzeptieren. Die analoge Frage ist für deterministische linear beschränkte Automaten noch offen, d.h. es ist weder ein Beweis gegeben worden, dass jede kontextabhängige Sprache von einem deterministischen linear beschränkten Automaten akzeptiert werden kann, noch ein Beispiel einer kontextabhängigen Sprache bekannt, die nicht von einem deterministischen linear beschränkten Automaten akzeptiert werden kann.

2.2.2 Endliche Automaten

Im vorangehenden Abschnitt haben wir Charakterisierungen der Sprachfamilien $\mathcal{L}(RE)$ und $\mathcal{L}(CS)$ mittels TURING-Maschinen bzw. linear beschränkten Automaten angegeben. Wir wollen nun eine analoge Charakterisierung für die Familie der regulären Sprachen herleiten. Zuerst definieren dazu den hierfür geeigneten Automatentyp.

Definition 2.46 *i) Ein endlicher Automat ist ein Quintupel*

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

wobei
- X und Z Alphabete sind,
- $z_0 \in Z$ und $F \subseteq Z$ gelten,
- δ eine Funktion von $Z \times X$ in Z ist.
ii) Die Erweiterung δ^* von δ auf $Z \times X^*$ ist durch

$$\delta^*(z, \lambda) = z,$$
$$\delta^*(z, wx) = \delta(\delta^*(z, w), x) \text{ für } w \in X^*, x \in X$$

definiert.
iii) Die durch $\mathcal{A}$ akzeptierte Wortmenge ist durch

$$T(\mathcal{A}) = \{w : w \in X^*, \delta^*(z_0, w) \in F\}$$

definiert.

Wie bei TURING-Maschinen nennen wir die Elemente von X erneut Eingabesymbole und die von Z Zustände; z_0 ist der Anfangszustand, und F ist die Menge der akzeptierenden Zustände; δ heißt erneut Überführungsfunktion. Im Folgenden werden wir meistens zwischen der Funktion δ und ihrer Erweiterung δ^* nicht unterscheiden und beide mit δ

bezeichnen, zumal aus der Definition sofort $\delta^*(z,x) = \delta(\delta^*(z,\lambda),x) = \delta(z,x)$ für $x \in X$ und $z \in Z$ folgt.
Die Arbeitsweise eines endlichen Automaten können wir uns wie folgt vorstellen: Der Automat liest von links nach rechts die Buchstaben des Eingabewortes und ändert bei jedem Lesevorgang seinen Zustand entsprechend δ, wobei er im Zustand z_0 beginnt. Ein Wort wird genau dann akzeptiert, wenn er nach Lesen des gesamten Wortes in einen akzeptierenden Zustand gelangt ist.
Entsprechend dieser Interpretation kann ein endlicher Automat als TURING-Maschine aufgefasst werden, bei der sich der Kopf nur nach rechts bewegt und beim Lesen des $*$ hinter dem Wort ein Stoppzustand erreicht wird. Das Schreiben auf das Band ist bei endlichen Automaten – wie wir sie definiert haben – nicht von Interesse, da die Zellen, in die geschrieben werden kann, wegen der ständigen Rechtsbewegung nicht mehr gelesen werden können, so dass das Schreiben keinen Einfluss auf die Akzeptanz hat. Wir merken aber an, dass dann, wenn man sich nicht nur für das Akzeptanzverhalten von endlichen Automaten interessiert, auch eine entsprechende Modifikation des Begriffs zum endlichen Automaten mit Ausgabe möglich ist.

Um einen endlichen Automaten zu beschreiben, ist es nach Definition notwendig, die einzelnen Komponenten X, Z, z_0, F, δ von $\mathcal{A}$ anzugeben. Vielfach wird aber eine Beschreibung von $\mathcal{A}$ durch einen gerichteten Graphen $G = (V, E)$, dessen Kanten bewertet (oder markiert) sind, bevorzugt. Als Knotenmenge V verwenden wir die Zustandsmenge Z, und es gibt genau dann eine Kante von z nach z', die mit x bewertet ist, falls $\delta(z,x) = z'$ gilt. Zur Auszeichnung des Anfangszustandes bzw. der akzeptierenden Zustände benutzen wir einen auf den Knoten gerichteten Pfeil bzw. einen doppelten Kreis. In dieser Beschreibung wird $\delta^*(z, x_1x_2\ldots x_n) = z'$ durch die Existenz eines Weges von z nach z' widergespiegelt, bei dem die Folge der Bewertungen durch $x_1x_2\ldots x_n$ gegeben ist.

Beispiel 2.47 Der endliche Automat $\mathcal{A} = (X, Z, z_0, F, \delta)$ sei durch

$$\begin{aligned} X &= \{a,b,c\} \\ Z &= \{z_0,z_1,z_2,z_3\}, \\ F &= \{z_2\}, \\ \delta(z,x) &= \begin{cases} z_1 & \text{für } z = z_0, x = a \\ z_2 & \text{für } z = z_1, x = a \\ z_0 & \text{für } z \in \{z_0,z_2\}, x = c \\ z_3 & \text{sonst} \end{cases} \end{aligned}$$

gegeben. Die Darstellung von $\mathcal{A}$ durch einen Graphen wird in Abb. 2.5 gezeigt.
Wir bestimmen nun die von $\mathcal{A}$ akzeptierte Wortmenge. Wir geben die Erläuterungen dabei immer durch Bezug auf die Überführungsfunktion; der Leser möge sie jedoch auch anhand des Graphen zu verfolgen.
Wir stellen dazu erst einmal fest, dass wegen $\delta(z_3,x) = z_3$ für alle $x \in X$ der Zustand z_3 nicht mehr verlassen werden kann, womit eine Akzeptanz ausgeschlossen ist. Da außerdem $\delta(z,b) = z_3$ für alle $z \in Z$ gilt, wird z_3 erreicht, wenn im Wort ein b vorkommt. Hieraus folgt, dass ein Wort nur dann akzeptiert werden kann, wenn es kein b enthält. Wir bemerken noch, dass die einzige Möglichkeit des Übergangs von z_0 zu z_2 durch $\delta(z_0,aa) = z_2$ gegeben ist. Da $\delta(z_2,c^n) = \delta(z_0,c^{n-1}) = z_0$ für beliebige natürliche Zahlen $n \geq 1$ gelten,

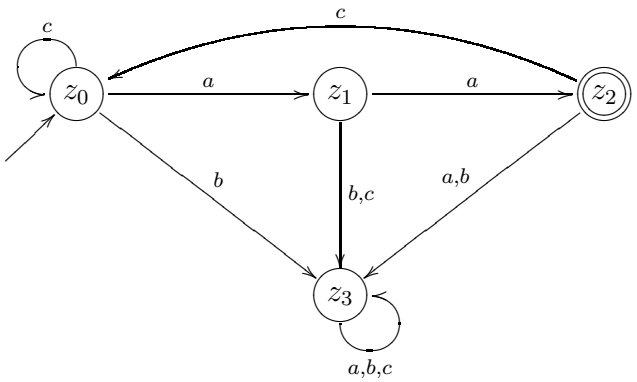


Abbildung 2.5:

erhalten wir, dass $T(\mathcal{A})$ aus allen Wörtern besteht, bei denen einer beliebigen Anzahl von Vorkommen von c stets aa folgt und die kein b enthalten, d.h.

$$T(\mathcal{A}) = \{c^{n_1}aac^{n_2}aa \ldots c^{n_k}aa : \ k \geq 1, \ n_1 \geq 0, \ n_i \geq 1 \text{ für } 1 \leq i \leq k\}.$$

Beispiel 2.48 Wir wollen einen endlichen Automaten $\mathcal{A}$ so bestimmen, dass

$$T(\mathcal{A}) = \{a^n b^m : n \geq 1, m \geq 2\}$$

gilt.[2] Offensichtlich können wir $X = \{a, b\}$ annehmen. Ferner benutzen wir Zustände, um zu zählen, wieviele Buchstaben a bzw. b bereits im gelesenen Teil des Wortes enthalten sind. Folgende Zustände entsprechen folgenden Situationen:
z_1 – es ist mindestens ein a und kein b gelesen worden,
z_2 – es sind mindestens ein a und genau ein b gelesen worden,
z_3 – es sind mindestens ein a und mindestens zwei b gelesen worden.
Ferner haben wir zu beachten, dass bei zu akzeptierenden Wörtern kein a auf ein b folgen darf. Ein endlicher Automat mit dieser Eigenschaft ist offenbar durch Abb. 2.6 gegeben.

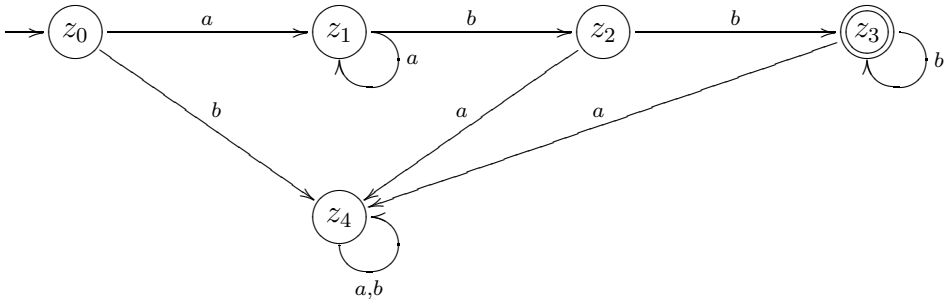


Abbildung 2.6:

Wir definieren nun eine nichtdeterministische Variante des endlichen Automaten. Dabei gehen wir analog zu TURING-Maschinen vor, d.h. die Bilder bei δ werden Mengen von Zuständen anstelle von einzelnen Zuständen sein.

[2]Wir diskutieren hier nicht die Frage, ob diese Sprache überhaupt von einem endlichen Automaten akzeptiert werden kann, da wir in diesem Abschnitt diese Frage generell klären werden.

Definition 2.49 *i) Ein nichtdeterministischer endlicher Automat ist ein Quintupel* $\mathcal{A} = (X, Z, z_0, F, \delta)$*, wobei für* X, Z, z_0, F *die gleichen Bedingungen wie in Definition 2.46 gelten und* δ *eine Funktion von* $Z \times X$ *in die Menge der Teilmengen von* Z *ist.*
ii) Wir definieren $\delta^*(z, \lambda) = \{z\}$ *für* $z \in Z$*, und für* $w \in X^*$*,* $x \in X$ *und* $z \in Z$ *gelte* $z' \in \delta^*(z, wx)$ *genau dann, wenn es einen Zustand* $z'' \in \delta^*(z, w)$ *mit* $z' \in \delta(z'', x)$ *gibt.*
iii) Die von $\mathcal{A}$ *akzeptierte Wortmenge ist durch*

$$T(\mathcal{A}) = \{w : \delta^*(z_0, w) \cap F \neq \emptyset\}$$

definiert.

Erneut ist damit der (deterministische) endliche Automat der Spezialfall des nichtdeterministischen Automaten, bei dem jede Menge $\delta(z, x)$ und damit dann auch jede Menge $\delta^*(z, w)$ einelementig ist. Daher stimmt dann auch die vom so interpretierten nichtdeterministischen Automaten akzeptierte Sprache mit der des deterministischen überein.

Wir beweisen nun die Äquivalenz von deterministischen und nichtdeterministischen endlichen Automaten bezüglich ihrer Akzeptierfähigkeit, die wir für TURING-Maschinen schon in Satz 2.43 gezeigt haben.

Satz 2.50 *Die beiden folgenden Aussagen sind für eine Sprache* L *äquivalent:*
i) L *wird von einem (deterministischen) endlichen Automaten akzeptiert.*
ii) L *wird von einem nichtdeterministischen endlichen Automaten akzeptiert.*

Beweis. i) $\Rightarrow$ ii) folgt sofort aus den obigen Bemerkungen, dass (deterministische) endliche Automaten als spezielle nichtdeterministische aufgefasst werden können.
ii) $\Rightarrow$ i). Es sei $\mathcal{A} = (X, Z, z_0, F, \delta)$ ein nichtdeterministischer Automat. Wir konstruieren den (deterministischen) endlichen Automaten $\mathcal{A}' = (X, Z', z_0', F', \delta')$ mit

$$\begin{aligned} Z' &= \{U : U \subseteq Z\}, \\ z_0' &= \{z_0\}, \\ F' &= \{U : U \in Z',\ U \cap F \neq \emptyset\}, \\ \delta'(U, x) &= \cup_{z \in U} \delta(z, x). \end{aligned}$$

Mittels vollständiger Induktion über die Wortlänge zeigen wir nun

$$(*) \quad (\delta')^*(\{z_0\}, w) = \delta^*(z_0, w)$$

für alle Wörter $w \in X^*$.
Für $w = \lambda$ folgt dies direkt aus den Definitionen der Erweiterungen. Damit ist der Induktionsanfang bewiesen.
Sei nun $w = w'x$ und die Aussage bereits für w' gültig. Dann gilt

$$(\delta')^*(\{z_0\}, w'x) = \delta'((\delta')^*(\{z_0\}, w'), x) = \delta'(\delta^*(z_0, w'), x) = \cup_{z \in \delta^*(z_0, w')} \delta(z, x) = \delta^*(z_0, w'x).$$

Damit gilt $(\delta')^*(\{z_0\}, w) = \delta^*(z_0, w)$, womit auch der Induktionschritt nachgewiesen ist.

Sei nun $w \in T(\mathcal{A})$. Dann gilt $\delta^*(z_0, w) \cap F \neq \emptyset$. Nach Definition heißt dies $\delta^*(z_0, w) \in F'$. Wegen $(*)$ gilt auch $(\delta')^*(\{z_0\}, w) \in F'$, womit $w \in T(\mathcal{A}')$ gezeigt ist.
Durch Umkehrung der eben durchgeführten Schlüsse können wir zeigen, dass aus $w \in T(\mathcal{A}')$ auch $w \in T(\mathcal{A})$ folgt. Damit ist dann $T(\mathcal{A}) = T(\mathcal{A}')$ gezeigt. □

Wir kommen nun zum Hauptresultat dieses Abschnittes, in dem wir zeigen, dass die von (nichtdeterministischen) endlichen Automaten akzeptierten Sprachen mit den regulären Sprachen übereinstimmen.

Satz 2.51 *Für eine Sprache L sind die folgenden Aussagen äquivalent.*

i) L ist regulär.

ii) L wird von einem nichtdeterministischen endlichen Automaten akzeptiert.
iii) L wird von einem (deterministischen) endlichen Automaten akzeptiert.

Beweis. i) ⇒ ii). Wir geben den Beweis zuerst für den Fall, dass L das Leerwort nicht enthält.
Es sei $G = (N, T, P, S)$ eine reguläre Grammatik mit $L(G) = L$. Entsprechend Satz 2.24 können wir ohne Beschränkung der Allgemeinheit annehmen, dass alle Regeln in P von der Form $A \to xB,\ A \to x$ mit $A, B \in N,\ x \in T$ sind. Wir konstruieren nun zuerst die reguläre Grammatik $G' = (N', T, P', S)$ mit

$$\begin{aligned} N' &= N \cup \{\$\}, \\ P' &= \{A \to xB : A \to xB \in P\} \cup \{A \to x\$: A \to x \in P\} \cup \{\$ \to \lambda\}, \end{aligned}$$

wobei \$ ein zusätzliches Symbol ist ($\$ \notin N \cup T$). Da die terminierenden Ableitungen in G bzw. G' die Form

$$S \Longrightarrow^* wA \Longrightarrow wa$$

bzw.

$$S \Longrightarrow^* wA \Longrightarrow wa\$ \Longrightarrow wa$$

haben, ist leicht zu sehen, dass $L(G) = L(G') = L$ gilt.
Wir konstruieren nun einen nichtdeterministischen endlichen Automaten $\mathcal{A}$, für den $T(\mathcal{A}) = L$ gilt. Damit ist dann die Behauptung gezeigt.
Wir setzen dazu $\mathcal{A} = (T, N', S, \{\$\}, \delta)$, wobei die Überführungsfunktion durch

$$\delta(A, x) = \{B : A \to xB \in P\}$$

gegeben ist.
Wir zeigen zuerst mittels vollständiger Induktion über die Wortlänge, dass eine Ableitung $A \Longrightarrow^* x_1 x_2 \dots x_n B$ genau dann in G' existiert, wenn $B \in \delta(A, x_1 x_2 \dots x_n)$ gilt.
Der Induktionsanfang, d.h. diese Aussage für $n = 1$, gilt nach der Definition von δ.
Es sei nun eine Ableitung $A \Longrightarrow x_1 x_2 \dots x_{n-1} B' \Longrightarrow x_1 x_2 \dots x_{n-1} x_n B$ in G' gegeben. Nach Induktionsvoraussetzung und Definition von δ gelten dann $B' \in \delta(A, x_1 x_2 \dots x_{n-1})$ und $B \in \delta(B', x_n)$. Folglich ist $B \in \delta(A, x_1 x_2 \dots x_{n-1} x_n)$.

Gilt umgekehrt $B \in \delta(A, x_1x_2 \dots x_n)$. Dann gibt es einen Zustand B' (d.h. ein Nichtterminal B') derart, dass $B \in \delta(B', x_n)$ und $B' \in \delta(A, x_1x_2 \dots x_{n-1})$ gelten. Nach Induktionsvoraussetzung gibt es damit eine Ableitung $A \Longrightarrow^* x_1x_2 \dots x_{n-1}B'$ in G', und aus der Definition von δ folgt $B' \Longrightarrow x_nB$. Somit existiert eine Ableitung $A \Longrightarrow^* x_1x_2 \dots x_{n-1}B' \Longrightarrow x_1x_2 \dots x_{n-1}x_nB$.
Damit ist auch der Induktionsschritt vollzogen.
Wir betrachten jetzt ein Wort $w \in L(G')$. Dann gibt es eine Ableitung $S \Longrightarrow^* w\$ \Longrightarrow w$ in G'. Entsprechend der oben bewiesenen Aussage gilt dann $\$ \in \delta(S, w)$ und damit $w \in T(\mathcal{A})$.
Umgekehrt folgt aus $w \in T(\mathcal{A})$, also $\$ \in \delta(S, w)$, mittels der obigen Aussage die Existenz einer Ableitung $S \Longrightarrow^* w\$$ in G' und damit wegen $\$ \to \lambda \in P'$ auch $S \Longrightarrow^* w$.
Aus den beiden letzten Bemerkungen folgt $T(\mathcal{A}) = L(G')$, womit wegen $L = L(G) = L(G')$ auch $T(\mathcal{A}) = L$ bewiesen ist.

Gilt $\lambda \in L$, so modifizieren wir die Konstruktion wie folgt. Die Grammatik in der Normalform aus Satz 2.24 enthält dann zusätzlich die Regel $S \to \lambda$ und S kommt in keiner rechten Seite von Regeln aus P vor. Wir nehmen diese zusätzliche Regel auch in P' auf, und da diese nur die direkte Ableitung des Leerwortes bewirkt, muss auch S in die Menge der akzeptierenden Zustände von $\mathcal{A}$ aufgenommen werden. Nun laufen die Argumentationen für $L(G) = T(\mathcal{A})$ wie oben ab.

ii) $\Rightarrow$ iii) ist durch Satz 2.50 gegeben.

iii) $\Rightarrow$ i). Sei ein deterministischer endlicher Automat $\mathcal{A} = (X, Z, z_0, F, \delta)$ gegeben. Wir konstruieren dazu die reguläre Grammatik $G = (Z, X, P, z_0)$ mit

$$P = \{z \to az' : z' \in \delta(z, a)\} \cup \{z \to \lambda : z \in F\}.$$

Wie im ersten Teil dieses Beweises können wir zeigen, dass $z \in \delta(z_0, w)$ für ein $z \in F$ genau dann gilt, wenn es eine Ableitung $z_0 \Longrightarrow^* wz \Longrightarrow w$ gibt, woraus $T(\mathcal{A}) = L(G)$ folgt. □

Wir illustrieren die beiden Konstruktionen im Beweis von Satz 2.51 durch jeweils ein Beispiel.

Beispiel 2.52 Wir betrachten die Grammatik

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

bei der P aus den Regeln

$$S \to \lambda, S \to aA, S \to a, S \to b, S \to bB, A \to a,$$
$$A \to b, A \to aA, A \to bB, B \to bB, B \to bB, B \to b$$

besteht. Die im Beweis zuerst vorgenommene Umformung liefert dann

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

mit

$$\begin{aligned} P' \;=\; & \{S \to \lambda, S \to aA, S \to a\$, S \to b\$, S \to bB, A \to a\$, \\ & A \to b\$, A \to aA, A \to bB, B \to bB, B \to b\$, \$ \to \lambda\}. \end{aligned}$$

Der gesuchte Automat $\mathcal{B}$ ergibt sich dann durch

$$\mathcal{B} = (\{a, b\}, \{S, A, B, \$\}, S, \{S, \$\}, \delta)$$

mit

$$\begin{aligned}
&\delta(S, a) = \delta(A, a) = \{A, \$\},\\
&\delta(S, b) = \delta(A, b) = \delta(B, b) = \{B, \$\},\\
&\delta(B, a) = \delta(\$, a) = \delta(\$, b) = \emptyset.
\end{aligned}$$

Weiterhin konstruieren wir noch einen (deterministischen) endlichen Automaten $\mathcal{B}'$ an, der die gleiche Menge wie $\mathcal{B}$ akzeptiert. Dazu gehen wir wie im Beweis von Satz 2.50 vor. Die Menge Z der Zustände von $\mathcal{B}'$ wird dann von allen Teilmengen von $\{S, A, B, \$\}$ und die Menge F der akzeptierenden Zustände von allen den Teilmengen, die S oder $\$$ enthalten, gebildet. Wir erhalten dann

$$\mathcal{B}' = (\{a, b\}, Z, \{S\}, F, \delta),$$

wobei

$$\begin{aligned}
&\delta'(\{S\}, a) = \delta'(\{A\}, a) = \delta'(\{S, A\}, a) = \delta'(\{S, B\}, a) = \delta'(\{A, B\}, a)\\
&\qquad = \delta'(\{S, A, B\}, a) = \{A, \$\},\\
&\delta'(\{B\}, a) = \delta'(\emptyset, a) = \delta'(\emptyset, b) = \emptyset,\\
&\delta'(\{S\}, b) = \delta'(\{A\}, b) = \delta'(\{B\}, b) = \delta'(\{S, A\}, b) = \delta'(\{S, B\}, b)\\
&\qquad = \delta'(\{A, B\}, b) = \delta'(\{S, A, B\}, b) = \{B, \$\},\\
&\delta'(U \cup \{\$\}, x) = \delta'(U, x) \cup \{\S\} \quad \text{für } U \subseteq \{S, A, B\}, x \in \{a, b\}
\end{aligned}$$

gesetzt wird.

Beispiel 2.53 Haben wir eben zu einer Grammatik G einen (nichtdeterministischen) endlichen Automaten angegeben, der $L(G)$ akzeptiert, so konstruieren wir nun umgekehrt zu dem Automaten $\mathcal{A}$ aus Beispiel 2.47 eine reguläre Grammatik G mit $L(G) = T(\mathcal{A})$. Entsprechend dem Beweis von Satz 2.51 ergibt sich

$$G = (\{z_0, z_1, z_2, z_3\}, \{a, b, c\}, P, z_0)$$

mit

$$\begin{aligned}
P \;=\; &\{z_0 \to az_1, z_0 \to bz_3, z_0 \to cz_0, z_1 \to az_2, z_1 \to bz_3, z_1 \to cz_3,\\
&\quad z_2 \to az_3, z_2 \to bz_3, z_2 \to cz_0, z_3 \to az_3, z_3 \to bz_3, z_3 \to cz_3\}.
\end{aligned}$$

2.2.3 Kellerautomaten

Die im vorhergehenden Abschnitt angegebenen Charakterisierungen von Sprachen mittels Maschinen oder Automaten ergänzen wir in diesem Abschnitt durch eine solche Charakterisierung der kontextfreien Sprachen. Endliche Automaten können kontextfreie, aber nicht reguläre Sprachen wie $\{a^n b^n : n \geq 1\}$ oder $\{wcw^R : w \in \{a, b\}^*\}$ im Wesentlichen deshalb

nicht akzeptieren, weil eine endliche Menge von Zuständen nicht ausreicht, um sich die Länge bzw. die Struktur des schon gelesenen Wortanfangs zu merken. Um kontextfreie Sprachen zu akzeptieren, müssen wir den Automaten daher mit einer Möglichkeit zum Speichern dieser Information versehen. Hierfür werden wir ein zusätzliches Arbeitsband benutzen.
Wenn wir keine Restriktionen an das Arbeiten auf dem Arbeitsband stellen, so könnten wir die Eingabe von links nach rechts lesen und dabei auf das Arbeitsband übertragen und anschließend das Arbeitsband wie bei einer TURING-Maschine nutzen. Dann könnten offenbar wie bei TURING-Maschinen alle rekursiv-aufzählbaren Sprachen akzeptiert werden. Da wir einen Automatentyp suchen, der nur die kontextfreien Sprachen akzeptiert, müssen wir eine Einschränkung der Arbeit auf dem Arbeitsband vornehmen.
Wir nehmen folgende Einschränkungen vor:

- Die Symbole des Eingabebandes können nur von links nach rechts gelesen werden, d.h. Kopfbewegungen nach links sind verboten (im Gegensatz zum endlichen Automaten gestatten wir aber das Verharren des Lesekopfes an einer Stelle, damit Veränderungen des Arbeitsbandes ohne gleichzeitiges Lesen vorgenommen werden können).

- Eine Zelle des Arbeitsbandes ist mit einem speziellen Symbol # markiert, das nicht gelöscht und überschrieben werden kann. Der Lese-/Schreibkopf des Arbeitsbandes bewegt sich nicht auf die Zellen rechts von der mit # markierten Zelle. Dadurch entsteht im Prinzip ein einseitig begrenztes Arbeitsband.

- Die Arbeit auf dem Arbeitsband erfolgt wie bei der Datenstruktur *Keller*. Dies bedeutet, dass jeweils nur das am weitesten links stehende Symbol verändert werden kann und nur Anfügungen nach links vorgenommen werden können. Damit werden nach einem Symbol γ rechts erzeugte Symbole nicht bearbeitet, bevor γ bearbeitet wird. Daher bezeichnet man diese Arbeitsweise auch als *zuletzt hinein - zuerst hinaus* (engl. *last in - first out* oder abgekürzt LIFO). [3]

  Wir werden das Arbeitsband auch als Keller bezeichnen.

Anschaulich ist diese Interpretation in der Abbildung 2.7 dargestellt.
Wir geben nun die formale Definition des entsprechenden Automatentyps.

Definition 2.54 *Ein Kellerautomat ist ein Sechstupel*

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

wobei
- X das Eingabealphabet ist,
- Z die endliche Menge von Zuständen ist,
- Γ das Bandalphabet ist,
- $z_0 \in Z$ und $F \subseteq Z$ gelten,
- δ eine Funktion von $Z \times X \times (\Gamma \cup \{\#\})$ in die Menge der endlichen Teilmengen von $Z \times \{R, N\} \times \Gamma^*$ ist, wobei $\# \notin \Gamma$, R und N zusätzliche Symbole sind.

[3] Wir bemerken, dass dieses Prinzip auch bei einem üblichen Keller gilt; was als letztes in den Keller getragen wird, ist auch als erstes herauszuholen, sonst kann auf vorher Eingelagertes nicht zugegriffen werden. Hierin liegt der Grund für die Bezeichnung des Automatentyps.

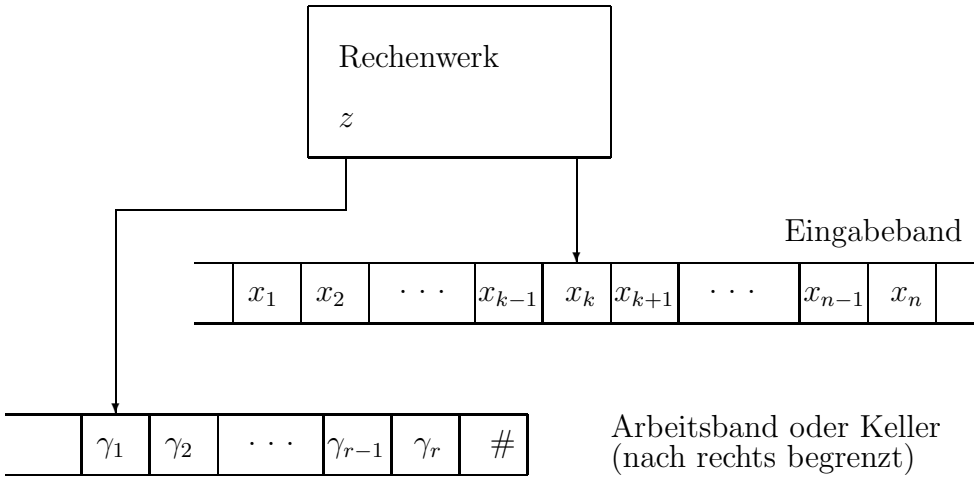


Abbildung 2.7: Schematische Darstellung eines Kellerautomaten

Die Arbeitsweise des Kellerautomaten wird wie folgt festgelegt.

Definition 2.55 *Es sei $\mathcal{M} = (X, Z, \Gamma, z_0, F, \delta)$ ein Kellerautomat wie in Definition 2.54. Eine Konfiguration K des Kellerautomaten $\mathcal{M}$ ist ein Tripel $(w, z, \alpha\#)$ mit $w \in X^*$, $z \in Z$ und $\alpha \in \Gamma^*$.*
Der Übergang von einer Konfiguration K_1 in die nachfolgende Konfiguration K_2 (den wir erneut mit $K_1 \models K_2$ bezeichnen) wird wie folgt beschreiben: Für $x \in X, v \in X^*, z \in Z, z' \in Z, \gamma \in \Gamma, \beta \in \Gamma^*, \alpha \in \Gamma^*$ gilt

$$\begin{array}{lll}
(xv, z, \gamma\alpha\#) \models (v, z', \beta\alpha\#), & \textit{falls} & (z', R, \beta) \in \delta(z, x, \gamma), \\
(xv, z, \gamma\alpha\#) \models (xv, z', \beta\alpha\#), & \textit{falls} & (z', N, \beta) \in \delta(z, x, \gamma), \\
(xv, z, \#) \models (v, z', \beta\#), & \textit{falls} & (z', R, \beta) \in \delta(z, x, \#), \\
(xv, z, \#) \models (xv, z', \beta\#), & \textit{falls} & (z', N, \beta) \in \delta(z, x, \#).
\end{array}$$

Eine Konfiguration $K = (w, z, \alpha\#)$ gibt den noch nicht gelesenen Teil w des Eingabewortes, den Zustand z des Automaten und das Wort auf dem Arbeitsband (im Keller) an.
Anschaulich wird bei einer Konfigurationsüberführung ausgehend vom Zustand, in dem sich das Rechenwerk befindet, dem gelesenen Symbol und dem ersten Kellersymbol ein neuer Zustand ermittelt und der Inhalt des Kellers verändert, indem das erste Symbol durch ein Wort ersetzt wird oder ein Wort vorangestellt wird. Genauer heißt dies:

- $(z', R, \beta) \in \delta(z, x, \gamma)$ bedeutet, dass der Kellerautomat, der sich im Zustand z befindet, das Symbol x liest und γ als erstes Symbol im Keller hat, in den Zustand z' geht, den Lesekopf des Eingabebandes nach rechts bewegt und das Symbol γ durch das Wort β ersetzt,

- $(z', N, \beta) \in \delta(z, x, \gamma)$ bedeutet, dass der Kellerautomat, der sich im Zustand z befindet, das Symbol x liest und γ als erstes Symbol im Keller hat, in den Zustand z'

geht, den Lesekopf nicht bewegt[4] und das Symbol γ durch das Wort β ersetzt,

- $(z', R, \beta) \in \delta(z, x, \#)$ bedeutet, dass der Kellerautomat, der sich im Zustand z befindet, das Symbol x liest und nur # im Keller hat, in den Zustand z' geht, den Lesekopf des Eingabebandes nach rechts bewegt und das Wort β vor # in den Keller schreibt,

- $(z', N, \beta) \in \delta(z, x, \#)$ bedeutet, dass der Kellerautomat, der sich im Zustand z befindet, das Symbol x x liest und nur # im Keller hat, in den Zustand z' geht, keine Kopfbewegung ausführt und β vor # in den Keller schreibt.

Der Lese-/Schreibkopf des Kellers (Arbeitsbandes) wird stets auf das erste Symbol im Keller gesetzt.

Mit $\models^*$ bezeichnen wir erneut den reflexiven und transitiven Abschluss der Relation $\models$.

Definition 2.56 *Es sei $\mathcal{M}$ ein Kellerautomat wie in Definition 2.54. Die von $\mathcal{M}$ akzeptierte Sprache ist durch*

$$\mathcal{M} = \{w : (w, z_0, \#) \models^* (\lambda, q, \#) \textit{ für ein } q \in F\}$$

definiert.

Ein Wort wird entsprechend dieser Definition akzeptiert, wenn ausgehend vom Anfangszustand und einem leeren Keller (d.h. der Keller enthält nur das Markierungssymbol #) nach dem vollständigen Lesen des Eingabewortes der Keller wieder leer ist und sich der Automat in einem akzeptierenden Zustand befindet. (Es lassen sich noch andere Varianten des Akzeptierens denken, so z. B. durch die Forderung, dass unabhängig vom Kellerinhalt nur ein akzeptierender Zustand erreicht wird oder unabhängig vom Zustand der Keller leer ist. Man kann beweisen, dass diese Varianten die Menge der akzeptierbaren Sprachen jedoch nicht verändern.)

Beispiel 2.57 Wir betrachten den Kellerautomaten

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

mit

$$\begin{aligned} &X = \{a, b\}, \quad \Gamma = \{a\}, \quad Z = \{z_0, z_1, z_2\}, \quad F = \{z_1\}, \\ &\delta(z_0, a, \#) = \{(z_0, R, aa)\}, \quad \delta(z_0, a, a) = \{(z_0, R, aaa)\}, \\ &\delta(z_0, b, a) = \{(z_1, R, \lambda)\}, \qquad \delta(q, b, a) = \{(z_1, R, \lambda)\} \end{aligned}$$

und

$$\delta(z, x, \gamma) = \{(z_2, R, \gamma)\}$$

[4]In manchen Lehrbüchern wird das Bewegen des Kopfes anders interpretiert. Eine Bewegung nach rechts entspricht dem Lesen des Symbols, während die Nichtbewegung als Nichtlesen gedeutet wird.

in allen sonstigen Fällen. Dann ergeben sich für die Eingaben $aabbbb$ und aba die folgenden Folgen von Konfigurationen:

$$\begin{aligned}(aabbbb, z_0, \#) &\models (abbbb, z_0, aa\#) \models (bbbb, z_0, aaaa\#) \models (bbb, z_1, aaa\#) \\ &\models (bb, z_1, aa\#) \models (b, z_1, a\#) \models (\lambda, z_1, \#)\end{aligned}$$

und

$$(aba, z_0, \#) \models (ba, z_0, aa\#) \models (a, z_1, a\#) \models (\lambda, z_2, \#).$$

Damit gelten $aabbbb \in T(\mathcal{M})$ und $aba \notin T(\mathcal{M})$.
Es ist leicht zu sehen, dass im Zustand z_0 beim Lesen eines a auf dem Band und einem a oder # an der Spitze des Kellers jeweils zwei a zusätzlich in den Keller geschrieben werden. Beim Lesen des ersten b wird in den Zustand z_1 gewechselt, und es beginnt der Prozeß des Kürzens des Kellers um ein a. Dies wird solange fortgesetzt, wie b gelesen werden und a im Keller sind. In allen anderen Situationen wird der Zustand z_2 erreicht, den der Kellerautomat nicht mehr verlassen kann, womit eine Akzeptanz des Wortes verhindert ist.
Da doppelt soviele a in den Keller geschrieben werden wie gelesen werden, müssen doppelt soviele b wie a gelesen werden, um den Keller wieder zu leeren. Folglich gilt

$$T(\mathcal{M}) = \{a^n b^{2n} : n \geq 1\}.$$

Die Idee hinter $\mathcal{M}$ ist im Wesentlichen folgende: Im Keller wird die Struktur des gelesenen Teilwortes gespeichert und dann mit dem noch nicht gelesenen Teilwort verglichen, wobei nur bei positiven Ausgang des Vergleichs das Eingabewort akzeptiert wird.
Eine grundsätzlich andere Idee für eine Konstruktion eines Kellerautomaten, der $L = \{a^n b^{2n} : n \geq 1\}$ akzeptiert, besteht darin, im Keller durch Speicherung der Satzformen im Wesentlichen die Ableitung der Wörter aus L zu simulieren, und dann das terminale Wort mit dem Eingabewort zu vergleichen. So einfach ist diese Idee aber nicht zu realisieren, da bei der Ableitung auch Nichtterminale zu ersetzen sind, die nicht am Wortanfang stehen, was nach der Arbeitsweise des Kellers nicht möglich ist. Gelöst wird dies Problem dadurch, dass immer bereits die Satzform und das Eingabewort soweit verglichen werden, wie dies zu dem Zeitpunkt möglich ist.
Wir formalisieren nun die eben beschriebenen Vorgehensweise, in dem wir den zugehörigen Kellerautomaten angeben. Dafür benötigen wir eine L erzeugende Grammatik. Dies ist

$$G = (\{S\}, \{a, b\}, \{S \to aSbb, S \to abb\}, S).$$

Wir definieren nun

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

mit

$$\delta'(z_0', x, \#) = \{(z_1', N, S)\} \quad \text{für } x \in \{a, b\}$$

(wir initialisieren den Keller mit dem Startsymbol S, das die zu Beginn vorliegende Satzform ist),

$$\delta'(z_1', x, S) = \{(z_1', N, aSbb), (z_1', N, abb)\} \quad \text{für } x \in \{a, b\}$$

(wir simulieren die Anwendung einer Regel für S im Keller, d.h. wir ersetzen S im Keller durch die rechte Seite einer Regel),

$$\delta'(z_1', x, x) = \{(z_1', R, \lambda)\} \quad \text{für } x \in \{a, b\}$$

(wir vergleichen das erste Symbol des Kellers mit dem gerade gelesenen Buchstaben auf dem Band) und

$$\delta'(z, x, \gamma) = \{(z_2', R, \lambda)\}$$

in allen weiteren Fällen. Für das obige Eingabewort $aabbbb$ erhalten wir

$$\begin{aligned}(aabbbb, z_0', \#) &\models (aabbbb, z_1', S\#) \models (aabbbb, z_1', aSbb\#) \models (abbbb, z_1', Sbb\#) \\ &\models (abbbb, z_1', abbbb\#) \models (bbbb, z_1', bbbb\#) \models (bbb, z_1', bbb\#) \\ &\models (bb, z_1', bb\#) \models (b, z_1', b\#) \models (\lambda, z_1', \#)\end{aligned}$$

als Konfigurationsfolge. Dagegen ist

$$(aba, z_0', \#) \models (aba, z_1', S\#) \models (aba, z_1', abb\#) \models (ba, z_1', bb\#) \models (a, z_1', b\#) \models (\lambda, z_2', \#)$$

nur eine mögliche Folge von Konfigurationen für die Eingabe aba, bei der eine Konfiguration vorliegt, die nicht mehr verändert werden kann, jedoch kann man sich leicht überlegen, dass wir bei jeder anderen Konfigurationsfolge auch $(\lambda, z_2', \#)$ erreichen.
Im Keller sei $Sb^{2n}\#$ enthalten (aus der Anfangskonfiguration $(w, z_0, \#)$ erhalten wir diese Situation mit $n = 0$ im ersten Schritt der Arbeit von $\mathcal{M}$'). Nun wird eine der Regeln $S \to aSbb$ oder $S \to abb$ simuliert, wodurch im Keller $aSb^{2(n+1)}\#$ oder $ab^{2(n+1)}\#$ entsteht. Im ersten Fall lesen wir einen Buchstaben, vergleichen diesen mit dem Spitzensymbol a des Kellers und erhalten $Sb^{2(n+1)}\#$, d.h. ein Wort der Form wie zu Beginn der Betrachtungen, oder erreichen den Zustand z_2', wodurch Akzeptanz ausgeschlossen wird. Im zweiten Fall vergleichen wir das noch nicht gelesene Wort mit dem Kellerinhalt und kommen bei Übereinstimmung zur Akzeptanz oder bei Nichtübereinstimmung in den Zustand z_2'.
Hieraus folgt, dass wir das Eingabewort dann akzeptieren, wenn es mit einem bei der Simulation erzeugten terminalen Satzform übereinstimmt. Somit gilt

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

Wir verallgemeinern nun die Idee der zweiten Konstruktion, um zu zeigen, dass durch Simulation von Ableitungen in kontextfreien Grammatiken die Akzeptierbarkeit der zugehörigen Sprache bewiesen werden kann.
Im Beispiel haben wir nach partiellem Vergleich immer das erste Nichtterminal der Satzform erreicht und dann nachfolgend keine Schwierigkeiten bekommen, da dies auch das einzige Nichtterminal der Satzform ist. Für die Verallgemeinerung ist es daher notwendig, zu zeigen, dass wir die erzeugte Sprache nicht verändern, wenn wir stets das am weitesten links stehende Nichtterminal ersetzen. Derartige Ableitungen nennen wir *Linksableitungen.*
Seien nun eine Satzform $w_1Aw_2Bw_3$ und zwei Regeln $A \to v_1$ und $B \to v_2$ gegeben. Dann bestehen die Ableitungen

$$w_1Aw_2Bw_3 \Longrightarrow w_1v_1w_2Bw_3 \Longrightarrow w_1v_1w_2v_2w_3$$

und

$$w_1Aw_2Bw_3 \Longrightarrow w_1Aw_2v_2w_3 \Longrightarrow w_1v_1w_2v_2w_3,$$

die beide zum gleichen Ergebnis führen. Somit ist eine derartige Vertauschung der Reihenfolge der Regelanwendungen möglich, ohne das erzeugte Wort zu verändern. Fortgesetzte derartige Änderung der Reihenfolge führt dazu, dass wir eine Linksableitung erhalten und das gleiche Wort erzeugen. Daraus folgt, dass wir bei Beschränkung auf Linksableitungen die gleiche Sprache erzeugen wie mittels beliebiger Ableitungen.

Lemma 2.58 *Für jede kontextfreie Sprache L gibt es einen Kellerautomaten $\mathcal{M}$ mit $T(\mathcal{M}) = L$.*

Beweis. Es sei $G = (N', T, P, S)$ eine kontextfreie Grammatik mit $L(G) = L$. Wir konstruieren nun zu G den Kellerautomaten

$$\mathcal{M} = (T, \{z_0, z_1, z_2\}, N' \cup T, z_0, \{z_1\}, \delta)$$

mit

$$\begin{aligned}
&\delta(z_0, x, \#) = \{(z_1, N, S)\} \quad \text{für } x \in T,\\
&\delta(z_1, x, A) = \{(z_1, N, v) : A \to v \in P\} \quad \text{für } x \in T,\\
&\delta(z_1, x, x) = \{(z_1, R, \lambda)\} \quad \text{für } x \in T
\end{aligned}$$

und

$$\delta(z, x, \gamma) = \{(z_2, R, \lambda)\}$$

in allen weiteren Fällen.

Zuerst bemerken wir, dass - abgesehen von der Anfangskonfiguration - nur Konfigurationen entstehen, die den Zustand z_1 oder z_2 enthalten. Ferner wird bei Erreichen des Zustands z_2 dieser nicht mehr verändert, womit eine Akzeptanz von w nicht mehr möglich ist. Wir untersuchen daher jetzt, welche Konfigurationen mit dem Zustand z_1 erreicht werden können. Wir zeigen, dass

$$(*) \qquad (w_1w_2, z_0, \#) \models^* (w_2, z_1, v\#)$$

genau dann gilt, wenn es eine Linksableitung

$$(**) \qquad S \Longrightarrow^* w_1v$$

in G gibt. Hieraus folgt dann mit $w = w_1, \lambda = w_2, v = \lambda$, dass $(\lambda, z_1, \#)$ genau dann erreicht wird, wenn es eine Linksableitung $S \Longrightarrow^* w$ gibt. Somit wird ein Wort genau dann akzeptiert, wenn es durch eine Linksableitung erzeugt werden kann. Nach den Bemerkungen vor diesem Lemma gilt folglich $T(\mathcal{M}) = L(G) = L$, womit das Lemma bewiesen ist.

$(*) \to (**)$. Wir benutzen absteigende Induktion über die Länge des noch nicht gelesenen Wortes. Für $w_1 = \lambda, w_2 = w, v = S$ gilt die Behauptung, da ausgehend von $(w, z_0, \#)$ in einem Schritt nur $(w, z_1, S\#)$ erreicht werden kann und $S \Longrightarrow^* S$ eine Linksableitung (mit null Ableitungsschritten) ist.

Sei nun $(w_1w_2, z_0, \#) \models^* (w_2, z_1, v\#)$ eine Überführung, bei der $w_2 \neq \lambda$ ist und für die eine Linksableitung $S \Longrightarrow^* w_1v$ existiert. Wir unterscheiden drei Fälle:
a) $v = av'$ für ein $a \in T$. Gilt auch $w_2 = aw_2'$, so gelten

$$(w_1aw_2', z_0, \#) \models^* (aw_2', z_1, av'\#) \models (w_2', z_1, v'\#)$$

und

$$S \Longrightarrow^* w_1v = w_1av',$$

womit die Aussage für das kürzere Wort w_2' gilt. Gilt dagegen $w_2 = bw_2'$ mit $a \neq b$, so kann nur in den Zustand z_2 übergegangen werden.
b) $v = Av'$ für ein $A \in N$. Ferner sei $A \to Xx$ eine Regel aus P. Dann erhalten wir durch Simulation dieser Regel

$$(w_2, z_1, Av'\#) \models (w_2, z_1, Xxv'\#).$$

Ist $X \in T$, so erreichen wir damit die unter a) diskutierte Situation, ist $X \in N$ fahren wir wie beschrieben fort, bis wir zu einer Anwendung einer Regel kommen, deren rechte Seite mit einem Terminal beginnt.
c) $v = \lambda$. Wegen $w_2 \neq \lambda$ gehen wir in den Zustand z_2.
$(**) \to (*)$. Wir führen den Beweis mittels vollständiger Induktion über die Anzahl der Ableitungsschritte in $(**)$ und zeigen sogar die schärfere Aussage: Ist nach n Ableitungsschritten in einer Linksableitung das Wort w_1Au mit $w_1 \in T^*$ erzeugt worden, so gibt es die Überführung $(w_1w_2, z_0, \#) \models^* (w_2, z_1, Au)$.
Für $n = 0$ folgt die Aussage mit $w_1 = \lambda, w_2 = w$ direkt aus der nach Definition existierenden Überführung $(w_2, z_0, \#) \models (w_2, z_1, S\#)$ und dem Fakt, dass in null Schritten nur S erzeugbar ist.
Sei die Aussage nun schon für n bewiesen. Ferner sei

$$S \Longrightarrow^* w_1Au \Longrightarrow w_1v_1Bv_2u$$

eine Linksableitung aus $n+1$ Ableitungsschritten, wobei der letzte Schritt in der Anwendung der $A \to v_1Bv_2$ mit $v_1 \in T^*$ besteht. Da es eine Linksableitung ist, gilt $w_1 \in T^*$. Nach Induktionsvoraussetzung gilt daher

$$(w_1w_2, z_0, \#) \models^* (w_2, z_1, Au\#).$$

Aufgrund der Definition von $\mathcal{M}$ gilt dann weiterhin

$$(w_2, z_1, Au\#) \models (w_2, z_1, v_1Bv_2u\#).$$

Falls $w_2 = v_1w_3$ erhalten wir außerdem

$$(v_1w_3, z_1, v_1Bv_2u\#) \models^* (w_3, z_1, Bv_2u\#)$$

und durch Kombination dieser Relation

$$(w_1v_1w_3, z_0, \#) \models^* (w_3, z_1, Bv_2u\#),$$

womit die Aussage auch für die Ableitung aus $n+1$ Schritten gilt. Ist aber $w_2 \neq v_1w_3$ für alle $w_3 \in T^*$, so erreichen wir ausgehend von $(w_2, z_1, v_1Bv_2u\#)$ den Zustand z_2. □

Ohne Beweis geben wir das folgende Lemma.