4 Kontextfreie Sprachen und Chomsky-Typ-2-Grammatiken

J.W. Backus führte 1959 eine Notation ein, die er zusammen mit P. Naur 1961 benutzte, um die Syntax der Programmiersprache ALGOL 60 zu formulieren. Schon 1955 hatte der Linguist Noam Chomsky die sogenannten semi-Thue-Systeme, die Axel Thue 1914 als allgemeine Ersetzungssysteme für Zeichenketten definiert hatte, abgewandelt, um damit Grammatiken für natürliche Sprachen zu definieren. Seine kontextfreie Grammatik ist die Grundlage der Backus/Naur-Notation, die wir hier als Beispiel bei der Definition eines Listentyps anwenden. Kontextfreie Grammatiken erlauben eine präzise Spezifikation der zulässigen syntaktischen Konstrukte und geben zugleich eine Möglichkeit für eine strukturelle Analyse der generierten Sätze oder Wörter. Damit sind natürliche Interpretationen oder sogar eingeschränkte Semantikzuordnungen möglich. Dass diese für die Analyse natürlicher Sprache nicht ausreichen, bedeutet keinen großen Nachteil, denn für regelbasierte Syntaxdefinitionen, wie die der meisten Programmiersprachen, sind diese Grammatiken ein nützliches Spezifikationsmittel.

4.1 Beispiel

⟨type list⟩ ::= ⟨simple variable⟩|⟨simple variable⟩, ⟨type list⟩

⟨simple variable⟩ ::= ⟨variable identifier⟩

⟨variable identifier⟩ ::= ⟨identifier⟩

⟨identifier⟩ ::= ⟨letter⟩|⟨identifier⟩⟨letter⟩|⟨identifier⟩⟨digit⟩

⟨digit⟩ ::= 0|1|2|3|4|5|6|7|8|9

⟨letter⟩ ::= $A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|\Sigma|\Gamma|Z$

Hierbei bedeutet die Schreibweise „⟨*Name*⟩“ eine metalinguistische Variable mit der Bezeichnung *Name*. Das Zeichen „::=“ wird gelesen als „soll sein“ oder „wird erklärt durch“. Der senkrechte Strich „|“ wird als Zeichen für die Alternative verwendet, während alles andere für sich selbst steht. Leichter zu lesen sind in der Regel die Syntaxdiagramme, die wir ohne Rekursion schon in Kapitel 3 gesehen haben. In Büchern zur Definition und Verwendung von Programmiersprachen, wie z.B. PASCAL oder MODULA, begegnen uns Syntaxdiagramme wie die aus Abbildung 4.1. Beim Lesen der Beschriftungen auf den Pfaden vom Anfang eines Syntaxdiagramms zum Ausgang muss der Inhalt eines jeden Kreises oder ovalen Feldes notiert werden, während die rechteckigen Kästchen bedeuten, dass an diesen Stellen an den Anfang des in dem Rechteck mit Namen bezeichneten Syntaxdiagramms gesprungen werden muss. Dieses wird dann solange interpretiert, bis der Ausgang erreicht wird, um dann an der Aussprungstelle des „aufrufenden“ Diagramms fortzufahren. Auf diese Weise können Mengen von Wörtern spezifiziert werden, die, wie wir sehen werden, nicht mehr regulär sind.

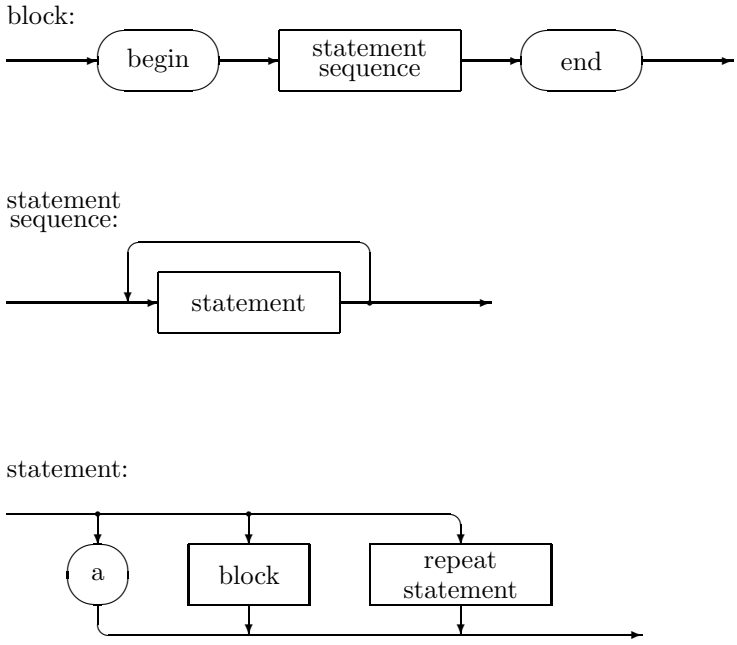


Abbildung 4.1: Drei Syntaxdiagramme

4.1 Grammatiken

In der theoretischen Informatik hat sich anstelle der Backus/Naur-Notation eine kürzere und vielleicht einfacher lesbare Notation eingebürgert. Bevor wir eine formale Definition für kontextfreie Grammatiken geben, wollen wir jedoch zunächst den allgemeineren Begriff einer Grammatik definieren, um dann die kontextfreien Grammatiken als einen Spezialfall zu betrachten.

4.2 Definition (Grammatik)
*Eine **Grammatik** wird beschrieben durch ein Quadrupel $G = (V_N, V_T, P, S)$ mit:*

- V_N *ist ein endliches Alphabet von **Nonterminalen** (auch: metalinguistischen Variablen, Hilfszeichen oder syntaktischen Kategorien).*
- V_T *ist ein mit V_N disjunktes endliches Alphabet von **Terminalen** (auch: Terminalsymbolen). Es muss also $V_T \cap V_N = \emptyset$ gelten. Das Gesamtalphabet wird mit $V := V_T \uplus V_N$ bezeichnet.*
- $P \subseteq V^* \setminus V_T \times V^*$ *ist eine endliche Menge von **Produktionen** (oder: Regeln).*
- $S \in V_N$ *ist das **Startsymbol**.*

Eine Regel $(u, v) \in P$ wird üblicherweise als $u \longrightarrow v$ notiert. Es ist zu beachten, dass nur eine einzige Bedingung an die Form einer Regel in einer allgemeinen Grammatik gestelt wird, nämlich die Bedingung, dass das zu ersetzende Teilwort mindestens ein Nonterminalsymbol enthält! Es können also mittels der Regeln einer allgemeinen Grammatik ganze Teilwörter durch andere Teilwörter ersetzt werden. Der Spezialfall der kontextfreien Grammatik, dem für die Spezifikation und Analyse der Syntax

von Programmiersprachen eine besondere Bedeutung zukommt, sieht hingegen nur vor, dass einzelne Nonterminale durch Wörter ersetzt werden können.

Im Unterschied zur kontextfreien Grammatik kann bei der allgemeinen Grammatik die Ersetzung eines Nonterminals also von den benachbarten Symbolen abhängen. Der Kontext, in dem die Ersetzung stattfindet ist also entscheidend! Aus diesem Grund erhielten die *kontextfreien Grammatiken* ihren Namen.

4.3 Definition (kontextfreie Grammatik)
Eine ***kontextfreie Grammatik*** *(CFG, context-free grammar) ist eine Grammatik* $G := (V_N, V_T, P, S)$*, für die einschränkend gilt:*

$P \subseteq V_N \times V^*$ *ist eine endliche Menge von* ***Produktionen*** *(oder: Regeln).*

Eine kontextfreie Regel $(A, w) \in P$ *wird üblicherweise als* $A \longrightarrow w$ *notiert. Wir bezeichnen* $A \longrightarrow \lambda$ *als* ***λ-Produktion****.*

Gilt $P \subseteq V_N \times V^+$*, so besitzt* G *keine λ-Produktionen und heißt* ***λ-frei****.*

Wir benötigen nun noch eine **Ableitungsrelation** zwischen Wörtern, um die Erzeugung von (abgeleiteten) Wörtern aus dem Startsymbol beschreiben zu können. Diese Ableitungsrelation wird – analog zu der erweiterten Zustandsüberführungsfunktion bei DFAs bzw. der Übergangsrelation bei NFAs – als eine Erweiterung der Produktionen definiert.

4.4 Definition (Ableitungsrelation)
Die ***einschrittige Ableitung*** *eines Wortes* v *aus einem Wort* u *mittels der Produktionen der Grammatik* G *wird notiert als* $u \underset{G}{\Rightarrow} v$*. Dabei ist die Relation* $\underset{G}{\Rightarrow} \subseteq V^* \times V^*$ *für alle* $u, v \in V^*$ *definiert durch:*

$$u \underset{G}{\Rightarrow} v$$

genau dann, wenn

$$\exists u_1, u_2 \in V^* : \exists w_l \in V^* V_N V^* : \exists w_l \longrightarrow w_r \in P : \; u = u_1 w_l u_2 \wedge v = u_1 w_r u_2$$

Als ***Ableitungsrelation*** *wird die reflexive, transitive Hülle der Relation* $\underset{G}{\Rightarrow}$ *bezeichnet, die als* $\overset{*}{\underset{G}{\Rightarrow}} \subseteq V^* \times V^*$ *geschrieben wird.*

Aus der vorangegangenen Definition ist leicht ersichtlich, dass ein Teilwort in einem Wort u, welches als linke Seite einer Produktion vorkommt, durch die rechte Seite dieser Produktion ersetzt werden kann.

Für kontextfreie Grammatiken vereinfacht sich die Definition der *einschrittigen Ableitungsrelation* aus Definition 4.4 folgendermaßen:

$$u \underset{G}{\Rightarrow} v$$

genau dann, wenn

$$\exists u_1, u_2 \in V^* : \exists A \in V_N : \exists A \longrightarrow w \in P : \; u = u_1 A u_2 \wedge v = u_1 w u_2$$

Es ist zu beachten, dass für jedes Wort $w \in V^*$, selbst wenn keine Produktion anwendbar ist, wegen der Reflexivität der Ableitungsrelation stets $w \overset{*}{\underset{G}{\Longrightarrow}} w$ gilt. Eine Zeichenkette $u \in V^*$, mit $S \overset{*}{\underset{G}{\Longrightarrow}} u$ nennt man **Satzform**, aber letztendlich interessant sind nur die Satzformen über dem Terminalalphabet. Wir definieren daher die von einer (kontextfreien) Grammatik erzeugte Sprache als die Menge aller aus dem Startsymbol erzeugbaren Terminalwörter:

4.5 Definition (erzeugte Sprache)
Sei $G = (V_N, V_T, P, S)$ eine Grammatik. $L(G) := \{w \in V_T^* \mid S \overset{*}{\underset{G}{\Longrightarrow}} w\}$ ist die von G generierte oder ***erzeugte Sprache****.*

Die Äquivalenz zweier endlicher Automaten hatten wir im vorangegangenen Kapitel über die Gleichheit der akzeptierten Sprachen definiert. Die folgende Definition führt einen analogen Äquivalenzbegriff auch für Grammatiken ein:

4.6 Definition (Äquivalenz von Grammatiken)
Zwei Grammatiken G_1 und G_2 heißen genau dann ***äquivalent*** *, wenn sie die gleiche Sprache erzeugen, also wenn $L(G_1) = L(G_2)$ gilt.*

Um im Folgenden wieder Aussagen über die Mächtigkeit der kontextfreien Sprachen machen zu können und diese mit anderen Sprachfamilien vergleichen zu können, führen wir die Familie der kontextfreien Sprachen auf naheliegende Weise ein:

4.7 Definition (Familie der kontextfreien Sprachen)
Die Familie aller kontextfreien Sprachen ist $\mathcal{CF} := \{M \mid \exists\, CFG\ G :\ M = L(G)\}$.

Eine kontextfreie Grammatik G wird üblicherweise einfach durch Angabe der einzelnen Produktionen spezifiziert. Dabei verabreden wir, dass die Elemente von V_N stets Großbuchstaben sind, während die Terminale durch Kleinbuchstaben bezeichnet werden. Die Mengen V_N und V_T ergeben sich eindeutig aus der vollständigen Spezifikation von P. Wird das Startsymbol ausnahmsweise nicht mit S bezeichnet, so muss dies vermerkt werden.

4.8 Beispiel
Wollen wir die Grammatik $G := (V_N, V_T, P, S)$ mit $P = \{S \longrightarrow aSb, S \longrightarrow \lambda\}$ notieren, so schreiben wir sie meist kurz als $S \longrightarrow aSb|\lambda$. Der senkrechte Strich kann dabei als „oder“ gelesen werden und stellt ein Trennsymbol dar, so dass wir es hier immernoch mit zwei Regeln zu tun haben.

G erzeugt die nicht reguläre Sprache $L(G) = DUP = \{a^n b^n \mid n \in \mathbb{N}\}$.

Beispiel 4.9 zeigt, dass die leere Menge auch als erzeugte Sprache von kontextfreien Grammatiken vorkommen kann.

4.9 Beispiel
Betrachten wir die Grammatik G mit folgenden Produktionen, so stellen wir fest, dass $L(G) = \emptyset$ ist:

$$S \longrightarrow AS \mid BA, \quad A \longrightarrow BAB \mid SB, \quad B \longrightarrow b$$

Offensichtlich sind alle Hilfszeichen hier überflüssig oder nutzlos.

Abgesehen von der Möglichkeit, die leere Menge durch eine kontextfreie Grammatik zu spezifizieren, hat Beispiel 4.9 gezeigt, dass möglicherweise nicht jedes Nonterminal und jede Regel einer Grammatik auch tatsächlich etwas zur erzeugten Sprache beiträgt. Solche überflüssigen Bestandteile sind natürlich unerwünscht, da sie das Verständnis wie auch den effizienten Umgang mit CFGs behindern. Selbst wenn man es selbst in der Hand hat, eine Grammatik anzugeben, die keine überflüssigen Bestandteile aufweist, so kann es durch die Anwendung von standardisierten Verfahren auf diese Grammatik passieren, dass überflüssige Symbole oder Regeln nachträglich (durch einen Algorithmus) eingefügt werden. Wir wollen deshalb zunächst formal definieren, wann eine kontextfreie Grammatik in diesem Sinne keine überflüssigen Bestandteile enthält und werden dnn erwartungsgemäß sehen, dass das Entfernen dieser Symbole, etc. nicht zu einer Veränderung der erzeugten Sprache führt.

4.10 Definition (reduzierte Grammatik)
*Eine kontextfreie Grammatik $G := (V_N, V_T, P, S)$ heißt genau dann **reduziert**, wenn gilt:*

1. *Jedes Nonterminal ist **produktiv**, d.h. $\forall A \in V_N$ gilt $\exists w \in V_T^* : \; A \overset{*}{\underset{G}{\Longrightarrow}} w$.*

2. *Jedes Nonterminal ist **erreichbar**, d.h. $\forall A \in V_N$ gilt $\exists u, v \in V^* : \; S \overset{*}{\underset{G}{\Longrightarrow}} uAv$.*

4.11 Theorem
Zu jeder CFG G kann effektiv eine äquivalente reduzierte CFG G' konstruiert werden.

Beweis: Die Konstruktion besteht aus zwei separaten Teilen:

Teil 1 entfernt Symbole (und Produktionen), so dass danach alle verwendeten Nonterminale stets auf Terminalworte abgeleitet werden können, d.h. G enthält danach nur noch produktive Nonterminalsymbole.

Teil 2 entfernt danach alle Symbole, die vom Startsymbol aus nicht erreichbar sind.

Teil 1 der Konstruktion: Zu gegebener CFG $G := (V_N, V_T, P, S)$ definieren wir Mengen $M_i \subseteq V$ durch: $M_0 := V_T$, $M_{i+1} := M_i \cup \{A \in V_N \mid \exists w \in M_i^* : \; A \longrightarrow w \in P\}$. Da für jedes $i \in \mathbb{N}$ $M_i \subseteq V$ endlich ist, und stets $M_i \subseteq M_{i+1}$ gilt, gibt es einen Index k, so dass $M_k = M_{k+1}$ ist. M_k enthält dann offensichtlich nur produktive Symbole. Nun bilden wir die neue CFG $G' = (V_N', V_T', P', S') := (M_k \cap V_N, V_T, P \cap (M_k \times M_k^*), S)$ mit $L(G') = L(G)$.

Teil 2 der Konstruktion: Nun werden alle erreichbaren Symbole in G' bestimmt: Man setze $M_0 := \{S'\}$ und $M_{i+1} := M_i \cup \{B \in V_N' \mid \exists A \in M_i \exists u, v \in V'^* : \; A \longrightarrow uBv \in P'\}$.

Wieder gilt $\exists k \in \mathbb{N} : \; M_k = M_{k+1}$ und M_k ist die Menge aller erreichbaren Nonterminale in G'. Man definiert nun $G'' := (V_N'', V_T'', P'', S'')$, indem von G' nur noch solche Produktionen beibehalten werden, deren rechte und linke Seiten nur mit erreichbaren Nonterminalen gebildet werden. Unter Umständen kann die Menge der dabei erreichbaren Terminalzeichen verkleinert werden.

□

Das Verfahren aus dem Beweis zu Theorem 4.11 wird falsch, wenn zuerst Schritt 2 und danach Schritt 1 angewendet wird, wie das folgende Beispiel zeigt: In der CFG mit den Produktionen $S \longrightarrow AB|\lambda$, $A \longrightarrow a$ sind alle Nonterminale erreichbar und keines würde in Schritt 2 gestrichen werden. Da das Symbol B jedoch unproduktiv ist, würden damit dann im nachfolgenden Schritt die Produktionen $S \longrightarrow \lambda$ und $A \longrightarrow a$ entstehen. Da beide Grammatiken aber nur λ als einziges Terminalwort erzeugen, ist die zweite davon nicht reduziert.

4.2 Chomsky-Normalform kontextfreier Grammatiken

Für die meisten Beweise und Konstruktionen mit kontextfreien Grammatiken sind Normalformen praktisch. Reduzierte Grammatiken können bereits als einer Normalform genügend angesehen werden, für die weitere Arbeit mit kontextfreien Grammatiken erweist sich aber die sogenannte Chomsky-Normalform als besonders wichtig.

4.12 Definition (Chomsky-Normalform)
*Eine kontextfreie Grammatik $G := (V_N, V_T, P, S)$ ist in **Chomsky-Normalform** (CNF) genau dann, wenn alle Produktionen in P von der Form $A \longrightarrow BC$ oder $A \longrightarrow a$ für $A, B, C \in V_N$ und $a \in V_T$ sind.*

Offensichtlich können kontextfreie Grammatiken in Chomsky-Normalform niemals das leere Wort λ erzeugen. Daher ist es praktisch, jede gewöhnliche CFG so umzuformen, dass $S \longrightarrow \lambda$ die einzige Produktion ist, die das leere Wort erzeugt, falls dieses in der erzeugten Sprache überhaupt vorkommt. Eine solche Grammatik wollen wir als *erweiterte Chomsky-Normalform* (eCNF) bezeichnen. Wir werden dies in den Beweis des nächsten Theorems integrieren, in dem gezeigt wird, dass tatsächlich zu jeder kontextfreien Grammatik eine äquivalente Grammatik in erweiterter Chomsky-Normalform existiert.

4.13 Theorem
Zu jeder CFG G kann effektiv eine äquivalente CFG $G' := (V'_N, V'_T, P', S')$ konstruiert werden, für die folgendes gilt:

Falls $\lambda \notin L(G)$, so ist G' in Chomsky-Normalform. Gilt $\lambda \in L(G)$, so ist $S' \longrightarrow \lambda$ die einzige λ-Produktion in P' und die Produktionen in $P' \setminus \{S' \longrightarrow \lambda\} \subseteq V'_N \times (V'_N \cdot V'_N \cup V'_T)$ sind in Chomsky-Normalform.

Beweis (Konstruktion einer Chomsky-Normalform-Grammatik): Sei $G := (V_N, V_T, P, S)$ eine beliebige CFG.

Der Beweis besteht aus sechs Schritten, in denen die ursprüngliche Grammatik in jeweils äquivalente Grammatiken G_1 bis G_6 umgeformt wird, von denen die letzte die gewünschte Eigenschaft hat. Wir benennen zunächst die Schritte und beschreiben danach die jeweilige Konstruktion:

1. λ-frei-machen
2. reduzieren
3. Kettenregeln entfernen

4. Ersetzen langer Terminalregeln
5. Verkürzen zu langer Regeln
6. Wiederherstellen der ursprünglichen Sprache durch evtl. Hinzunahme einer λ-Regel

1. Schritt: Es werden die λ-Produktionen entfernt: Für jedes $A \in V_N$ entscheiden wir, ob $A \overset{*}{\underset{G}{\Longrightarrow}} \lambda$ gilt. Dies ist leicht mit bereits bekannten Methoden möglich. Wir bestimmen die Menge $V_\lambda \subseteq V_N$ von Nonterminalen, aus denen u.a. auch das leere Wort abgeleitet werden kann. Dazu setzen wir:

$$\begin{aligned} M_0 &:= \{A \in V_N \mid A \longrightarrow \lambda \in P\} \text{ und} \\ M_{i+1} &:= M_i \cup \{A \in V_N \mid \exists w \in M_i^* : \ A \longrightarrow w \in P\} \end{aligned}$$

Es existiert ein Index $k \in I\!N$ mit $M_k = M_{k+1}$ und es ist $V_\lambda := M_k$ die Menge die gesucht wurde. Wir definieren nun die endliche Substitution:

$$\begin{aligned} \sigma : V^* \longrightarrow 2^{V^*} \quad \text{durch} \quad & \sigma(A) := \{A\}, \text{ falls } A \in (V \setminus V_\lambda), \text{ und} \\ & \sigma(A) := \{\lambda, A\} \text{ falls } A \in V_\lambda \end{aligned}$$

Damit ist dann $\sigma(u) = \{v \mid v$ geht aus u dadurch hervor, dass einige (alle, keines) der Symbole der Menge V_λ in u gestrichen werden$\}$. Zum Beispiel gilt dann für $V_\lambda := \{A, B\}$: $\sigma(aABAbC) = \{aABAbC, aBAbC, aABbC, aAAbC, aAbC, aBbC, abC\}$. Als neue Grammatik erhalten wir nach dem ersten Schritt zunächst eine CFG G_1, welche die Sprache $L(G_1) = L(G) \setminus \{\lambda\}$ erzeugt:

Es sei $G_1 := (V_N, V_T, P_1, S)$ mit $P_1 := \{A \longrightarrow v \mid v \neq \lambda \wedge v \in \sigma(w)$ für $A \longrightarrow w \in P\}$. Damit sind in P_1 also keine der λ-Produktionen von G mehr enthalten, und $\lambda \notin L(G_1)$ ist offensichtlich. Den Beweis für die Behauptung $L(G_1) \supset L(G) \setminus \{\lambda\}$ liefert eine Induktion über die Länge der Ableitungen in G, wobei wir hier die folgende noch allgemeinere Aussage (4.1) beweisen, da sie leichter zu zeigen ist:

$$\forall A \in V_N : \ A \overset{*}{\underset{G}{\Longrightarrow}} w \ \wedge \ w \in V_T^+ \text{ impliziert } A \overset{*}{\underset{G_1}{\Longrightarrow}} w \tag{4.1}$$

Induktionsbasis: Für Ableitungen der Länge 1 in G gilt mit $A \underset{G}{\Longrightarrow} w \ \wedge \ w \in V_T^+$ stets $A \longrightarrow w \in P$ und damit auch $A \longrightarrow w \in P_1$, da $P \subseteq P_1$ nach Definition gilt.

Induktionsschritt: Jede Ableitung $A \overset{*}{\underset{G}{\Longrightarrow}} w$ der Länge $k+1$ für ein nicht leeres Wort $w \in V_T^+$ hat die Form $A \underset{G}{\Longrightarrow} \Sigma_1\Sigma_2 \ldots \Sigma_r \overset{*}{\underset{G}{\Longrightarrow}} w$, wobei für $1 \leq i \leq r$ Ableitungen der Form $\Sigma_i \overset{*}{\underset{G}{\Longrightarrow}} w_i$ mit $w = w_1 w_2 \ldots w_r$ existieren. Alle diese Ableitungen haben eine Länge kleiner als k, so dass wir für sie die Aussage (4.1) als bewiesen annehmen wollen. Falls nun $w_i \neq \lambda$ ist, so ist auch $\Sigma_i \overset{*}{\underset{G_1}{\Longrightarrow}} w_i$ erfüllt. Falls jedoch $w_i = \lambda$ gilt, so ist $\Sigma_i \in V_\lambda$. Durch Streichen dieser Σ_i in $\Sigma_1\Sigma_2 \ldots \Sigma_r$, erhält man ein Wort $v \neq \lambda$, für das $A \longrightarrow v \in P_1$ ist. Aus dieser ersten Produktion zusammen mit den Ableitungen $\Sigma_i \overset{*}{\underset{G_1}{\Longrightarrow}} w_i$ erhält man die gesuchte Ableitung für $A \overset{*}{\underset{G_1}{\Longrightarrow}} w$.

Die Umkehrung $L(G_1) \subseteq L(G) \setminus \{\lambda\}$ ist leichter einzusehen, denn alle Produktionen in G_1 entstammen gültigen Ableitungen in G.

2. Schritt: G_1 ist möglicherweise nicht reduziert, so dass wir in diesem Schritt eine reduzierte, äquivalente CFG $G_2 := (V_{N_2}, V_{T_2}, P_2, S)$ aus G_1 mit dem Verfahren aus dem Beweis zu Theorem 4.11 erhalten.

3. Schritt: In G_2 kann es nun noch sogenannte **Kettenregeln** geben, also Produktionen der Form $A \longrightarrow B \in V_N \times V_N$, die in der Chomsky-Normalform nicht vorkommen dürfen.

Wir definieren die Relation $\ll \subseteq V_N \times V_N$ folgendermaßen: $A \ll B$ gilt genau dann, wenn $A \longrightarrow B \in P_2$. Mit $\overset{*}{\ll}$ sei die reflexive transitive Hülle von $\ll$ bezeichnet, die nach den früheren Verfahren bestimmt werden kann. Nun wird im dritten Schritt die Grammatik $G_3 := (V_{N,3}, V_{T,3}, P_3, S)$ definiert, indem $V_{N,3} := V_{N_2}$, $V_{T,3} := V_{T_2}$ und die neue Produktionenmenge $P_3 := \{A \longrightarrow w \mid w \notin V_{N,3} \ \wedge \ \exists B \longrightarrow w \in P_2 \ \wedge \ A \overset{*}{\ll} B\}$ anstelle von P_2 gesetzt wird. Das Startsymbol darf das alte S bleiben, denn sollte die reduzierte Grammatik kein einziges Wort mehr erzeugen, dann wurde $L(G_2) = \emptyset$ schon durch $P_2 = \emptyset$ erreicht. Die Gleichheit von $L(G_3)$ und $L(G_2)$ zeigt man leicht auch formal.

4. Schritt Da Terminalsymbole bei Grammatiken in Chomsky-Normalform nur durch Produktionen der Form $A \longrightarrow a$ generiert werden dürfen, ersetzen wir in allen rechten Seiten, deren Länge größer 1 ist, von Produktionen in P_3 jedes Terminal a durch ein neues Nonterminal $\langle a \rangle$ und erweitern P_3 zu P_4 durch Hinzunahme der Produktionen $\langle a \rangle \longrightarrow a$. Man erhält so im vierten Schritt $G_4 := (V_{N,4}, V_{T,3}, P_4, S)$.

5. Schritt Da nun aber in P_4 noch Produktionen $A \longrightarrow w$ mit $|w| > 2$ vorhanden sein können, wird im fünften Schritt eine weitere Konstruktion nötig. Wir definieren neue Nonterminale $\langle v \rangle$ für jedes echte Präfix v mit $|v| \geq 2$ einer rechten Seite einer Produktion in P_4. Zusammen mit neuen Produktionen erhalten wir so:

$$G_5 := (V_{N,5}, V_{T,3}, P_5, S) \text{ mit}$$

$$V_{N,5} := \{\langle v \rangle \mid \exists u \neq \lambda : \exists A \longrightarrow w \in P_4 : \ w = vu \wedge |v| \geq 2\} \cup V_{N,4} \text{ und}$$

$$\begin{aligned} P_5 \ := \ & \{A \longrightarrow \langle v \rangle x \mid A \longrightarrow w \in P_4 : \ |w| \geq 3 \wedge w = vx \wedge x \in V_{N,4}\} \cup \\ & \{\langle v \rangle \longrightarrow \langle u \rangle y \mid \langle u \rangle, \langle v \rangle \in (V_{N,5} \setminus V_{N,4}) \wedge y \in V_{N,4} \wedge v = uy\} \cup \\ & \{\langle v \rangle \longrightarrow xy \mid \langle v \rangle \in (V_{N,5} \setminus V_{N,4}) \wedge x, y \in V_{N,4} \wedge v = xy\} \cup \\ & \{A \longrightarrow w \mid A \longrightarrow w \in P_4 \wedge |w| \leq 2\} \end{aligned}$$

G_5 ist nun eine Grammatik in Chomsky-Normalform und erzeugt dieselbe Sprache, wie die Ursprungsgrammatik G, falls $\lambda \notin L(G)$ gilt. Anderenfalls muss noch ein letzter Schritt hinzukommen, in dem aus der CNF-Grammatik G_5 eine eCNF-Grammatik konstruiert wird.

6. Schritt Im ersten Schritt wurde die Menge V_λ zur Ausgangsgrammatik G bestimmt. Dort konnte also die Frage „$\lambda \in L(G)$?“ auf die Frage „$S \in V_\lambda$?“ reduziert, und damit entschieden werden. Gilt also $\lambda \in L(G)$, so konstruieren wir G_6, indem wir zu den Nonterminalen von G_5 ein neues Startsymbol S_{neu}, sowie die neuen Produktionen $S_{neu} \longrightarrow \lambda$ und $\{S_{neu} \longrightarrow w \mid \exists S \longrightarrow w \in P_5\}$ zu P_5 hinzufügen.

□

Der Beweis des vorigen Theorems gestattet folgendes Korollar (Folgerung), welches hier noch einmal hervorgehoben werden soll:

4.14 Korollar

Es gibt einen Algorithmus, der zu jeder vorgelegten kontextfreien Grammatik G entscheidet, ob $\lambda \in L(G)$ ist.

Beweis: Man konstruiere die Menge V_λ wie im ersten Schritt des Beweises von Theorem 4.13. Dann ist $\lambda \in L(G)$ genau dann, wenn $S \in V_\lambda$ ist. □

4.3 Lineare Grammatiken und reguläre Mengen

Bevor wir an die Konstruktion weiterer kontextfreier Grammatiken gehen, wollen wir zeigen, dass jede reguläre Menge eine spezielle kontextfreie Sprache ist. Die Grammatiken, die solche regulären Mengen erzeugen, besitzen eine einfache Struktur: Alle Produktionen enthalten in dem Wort, das die rechte Seite bildet, stets nur ein Nonterminal und dieses immer nur als erstes oder stets nur als letztes Zeichen.

4.15 Definition (lineare Grammatik)

Eine kontextfreie Grammatik $G := (V_N, V_T, P, S)$ heißt

***linear**, falls $P \subseteq V_N \times (V_T^* \cdot V_N \cdot V_T^* \cup V_T^*)$,*

***linkslinear**, falls $P \subseteq V_N \times (V_N \cdot V_T^* \cup V_T^*)$, und*

***rechtslinear**, falls $P \subseteq V_N \times (V_T^* \cdot V_N \cup V_T^*)$ ist.*

*Eine CFG wird genau dann **einseitig linear** genannt, wenn sie entweder linkslinear oder rechtslinear ist.*

In einigen Büchern werden rechtslineare Grammatiken kurz als **reguläre Grammatiken** bezeichnet. Von Noam Chomsky wurden diese Grammatiken als **Typ-3-Grammatiken** bezeichnet. Wir behalten aber im Folgenden die definierte Notation bei.

4.16 Theorem

Eine Sprache $R \subseteq \Sigma^*$ ist regulär genau dann, wenn es eine einseitig lineare Grammatik G gibt, die sie erzeugt, d.h., wenn $L(G) = R$ gilt.

Beweis: Sei $R \in \mathcal{R}eg$ definiert durch einen DFA $A_1 = (Z, \Sigma, K, z_0, Z_{\text{end}})$, dann werde eine rechtslineare CFG $G_R := (V_N, V_T, P, S)$ erklärt durch:

$$\begin{aligned} V_N &:= \{[z] \mid z \in Z\} \\ V_T &:= \Sigma \\ S &:= [z_0] \\ P &:= \{[z] \longrightarrow a[z'] \mid (z, a, z') \in K\} \cup \\ &\quad \{[z] \longrightarrow a \mid \exists z' \in Z_{\text{end}} : (z, a, z') \in K\} \cup \\ &\quad \{[z_0] \longrightarrow \lambda \mid z_0 \in Z_{\text{end}}\} \end{aligned}$$

Es ist leicht zu sehen, dass zu jedem Erfolgspfad in A die entsprechende Ableitung in G_R gefunden werden kann, und umgekehrt. Wenn wir $P' := \{[z] \longrightarrow [z']a \mid \exists(z, a, z') \in K\} \cup \{[z] \longrightarrow a \mid \exists z' \in Z_{\text{end}} : (z, a, z') \in K\}$ als linkslineare Regelmenge verwenden, dann generiert G'_R gerade $L(G'_R) = R^{rev}$. Da $L^{rev} \in \mathcal{Reg}$ nach Theorem 3.3.67 für jede Sprache $L \in \mathcal{Reg}$, kann $R = (R^{rev})^{rev}$ auch von einer linkslinearen CFG generiert werden.

Zu zeigen ist jetzt nur noch die Umkehrung, dass nämlich zu jeder rechtslinearen, (bzw. linkslinearen) CFG $G := (V_N, V_T, P, S)$ ein endlicher Automat konstruiert werden kann, der $L(G)$ akzeptiert. Wir zeigen dies hier nur für die rechtslinearen Grammatiken, denn ist eine CFG G linkslinear, so ersetzen wir vorab jede Produktion $A \longrightarrow Ba$ ($A, B \in V_N$, $a \in V_T$) durch die rechtslineare Produktion $A \longrightarrow aB$ und erhalten die rechtslinearen CFG G' mit $L(G') = L(G)^{rev}$. Wenn dann zu $L(G')$ ein endlicher Automat A' konstruiert wurde, kann dieser nach Theorem 3.3.67 in denjenigen umgeformt werden, der $L(A')^{rev} = L(G)$ akzeptiert. Dazu definieren wir einen NFA $A_2 := (Z, \Sigma, K, Z_{\text{start}}, Z_{\text{end}})$ durch

$$\begin{aligned}
Z &:= \{z_A \mid A \in V_N\} \cup \{[\lambda]\} \\
\Sigma &:= V_T \\
Z_{\text{start}} &:= \{z_S\} \\
Z_{\text{end}} &:= \{z_\lambda\} \\
K &:= \{(z_Q, u, z_R) \mid \exists Q, R \in V_N : \exists u \in V_T^* : \ Q \longrightarrow uR \in P\} \cup \\
&\quad\ \{(z_Q, u, z_\lambda) \mid \exists Q \in V_N : \exists u \in V_T^* : \ Q \longrightarrow u \in P\}
\end{aligned}$$

Auch hier ist es offensichtlich, dass zu jeder Ableitung eines Wortes $w \in V_T^*$ ein entsprechender Erfolgspfad im NFA gehört, und nur solche Wörter akzeptiert werden, die auch von der Grammatik generiert werden können. □

4.4 Ableitungen und Ableitungsbäume

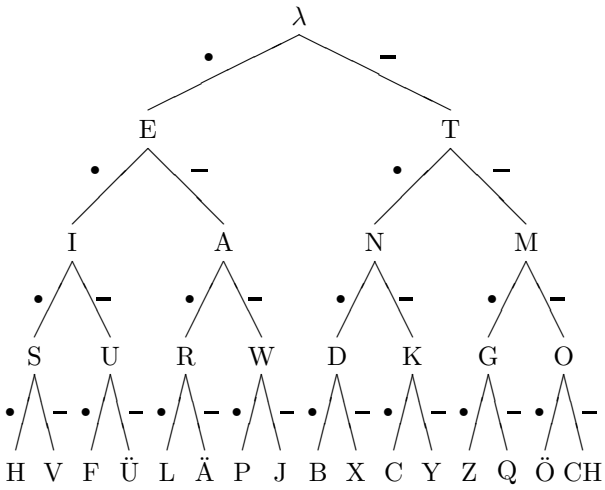
Zur Analyse von Wörtern einer kontextfreien Sprache ist es nötig, deren syntaktische Struktur zu bestimmen, denn in dieser sind die syntaktischen Kategorien durch die Nonterminale beschrieben. Diese Strukturen sind in dem sogenannten Ableitungsbaum des untersuchten Wortes sehr einprägsam dargestellt. Wir geben daher hier zunächst die nötigen Definitionen, bevor wir uns in Kapitel 6 den Analyseverfahren widmen.

In der Graphentheorie ist ein Baum jeder ungerichtete Graph ohne Kreise. Ein Baum heißt **lokal endlich**, wenn alle seine Knoten stets nur endlich viele direkte Nachbarn besitzen. In der Informatik verwenden wir häufig Bäume mit speziellen zusätzlichen Eigenschaften.

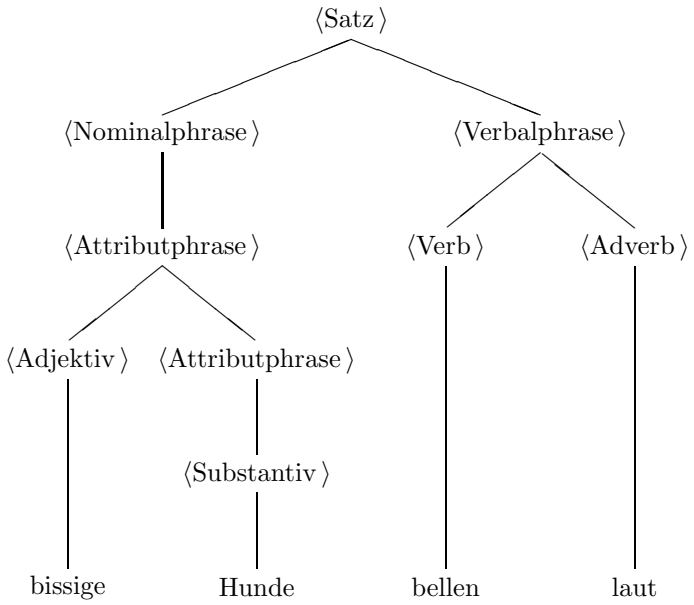
4.17 Definition (Baum)

*Unter einem **Baum** verstehen wir im Folgenden einen gerichteten, geordneten, zyklenfreien und knotenmarkierten Graphen mit einem ausgezeichneten Knoten, der keinen Vorgängerknoten besitzt, der sogenannten **Wurzel**. Die Wurzel wird gewöhnlich oben gezeichnet und die Richtung der Kanten wird entweder durch Pfeile oder durch die Anordnung gegeben. **Vorgänger** stehen über ihren **Nachfolgern**. Knoten ohne Nachfolger heißen **Blätter** und die Nachfolgerknoten sind* von links nach rechts angeordnet. *Die einzelnen Knoten werden mit Anschriften oder **Bezeichnungen** versehen, die aus einer beliebigen Menge gewählt werden können. **Kantenmarkierungen** können auf die gleiche Weise erlaubt sein.*

Als Beispiel für einen Baum betrachten wir den Kodierungsbaum für den Morse-Code. Dies ist ein binärer Baum, bei dem die Kantenbezeichnungen auf dem Pfad von der Wurzel λ zu einem Knoten den Morsecode seiner (Knoten-)Bezeichnung angeben:



Die Zerlegung eines Satzes einer natürlichen Sprache gemäß der syntaktischen Kategorien zeigt folgendes Beispiel eines Ableitungsbaumes. Hier sind wie in der Backus/Naur-Notation die syntaktischen Kategorien in spitzen Klammern eingerahmt. Die Terminalsymbole sind hier Wörter der „natürlichen“ Sprache:



Ableitungsbäume zu kontextfreien Grammatiken, manchmal auch Syntaxbäume genannt, sind spezielle Bäume, bei denen die inneren Knoten auf spezielle Weise mit den Nonterminalen beschriftet werden und die Blätter mit den Terminalsymbolen der generierten Zeichenkette. Als Wurzel eines Ableitungsbaumes fungiert das Startsymbol S der kontextfreien Grammatik.

4.18 Definition (Ableitungsbaum)
*Sei $G := (V_N, V_T, P, S)$ eine beliebige CFG, dann wird für jedes Symbol $A \in V$ ein A-**Ableitungsbaum** $B(A)$ wie folgt erklärt:*

1. *Jedes Blatt von $B(A)$ ist mit einem Symbol $a \in V_T \cup \{\lambda\}$ beschriftet.*
2. *Jeder Knoten von $B(A)$, der kein Blatt ist, ist mit einem Symbol $\Sigma \in V_N$, die Wurzel mit dem Symbol A beschriftet.*
3. *Ein (innerer) Knoten ist mit dem Symbol $\Sigma \in V_N$ genau dann beschriftet, wenn seine Nachfolgerknoten von links nach rechts mit $\Sigma_1, \Sigma_2, \ldots, \Sigma_k$ beschriftet sind und $\Sigma \longrightarrow \Sigma_1\Sigma_2 \ldots \Sigma_k$ eine Produktion von G ist.*

Notiert werden knotenbeschriftete Wurzelbäume oft auch als Terme, wobei die Terme zu den obigen Ableitungsbäumen einer gegebenen CFG G dann wiederum durch eine kontextfreie Grammatik G_T folgendermaßen definiert werden können:

4.19 Definition (Termform für Ableitungsbäume)
Zu einer beliebigen CFG $G := (V_N, V_T, P, S)$ definieren wir die entsprechende kontextfreie Grammatik für die Ableitungsbäume von G in Termform $G_T := (V'_N, V'_T, P', S')$ durch:

$$\begin{aligned} V'_N &:= \{X' \mid X \in V_N\} \\ V'_T &:= V_N \cup V_T \cup \{\Lambda, (,), ;\} \\ P' &:= \{X' \longrightarrow X(X''_1; X''_2; \ldots; X''_k) \mid X \longrightarrow X_1X_2 \ldots X_k \in P \text{ wobei} \\ &\quad X''_i = X'_i, \text{ falls } X_i \in V_N;\ X''_i = X_i, \text{ falls } X_i \in V_T, \text{ und} \\ &\quad X''_1 = \Lambda, \text{ falls } k = 1 \text{ und } X_1 = \lambda\} \end{aligned}$$

$L(G_T)$ ist nun genau die Menge aller derjenigen Terme, die Ableitungsbäume von G beschreiben.

4.20 Beispiel
Betrachte die CFG G_1 mit den folgenden Produktionen:

$$S \longrightarrow SS \mid aSb \mid \lambda$$

Die Terme t_1 und t_2 sind zwei unterschiedliche Beschreibungen von Ableitungsbäumen für das gleiche Wort $aabb$:

$$\begin{aligned} t_1 &:= S(S(a, S(a, S(\Lambda), b), b), S(\Lambda)) \text{ und} \\ t_2 &:= S(S(\Lambda), S(a, S(a, S(\Lambda), b), b)) \end{aligned}$$

Die von dieser Grammatik generierte Sprache $L(G_1)$ ist die **Dyck-Sprache** D_1 aller korrekt geklammerten Ausdrücke, bei denen a die öffnende und b die schließende Klammer bedeutet. Die gleiche Sprache kann auch durch die Grammatik G_2 mit den folgenden Produktionen erzeugt werden:

$$S \longrightarrow aSbS \mid \lambda$$

Bei G_2 ist es nie möglich, dass ein Wort der Sprache zwei unterschiedliche Ableitungsbäume besitzt.

Die in Beispiel 4.20 für die Grammatik G_2 festgestellte Eigenschaft, für jedes Wort der erzeugten Sprache genau einen Ableitungsbaum zu haben, ist gerade für solche kontextfreie Grammatiken von Interesse, deren Semantik durch die Struktur des Ableitungsbaums bestimmt wird. Dies sind insbesondere auch jene Grammatiken, die die Grundlage für die Definition der Syntax von Programmiersprachen bilden. Daher ist der folgende Begriff von großer praktischer Bedeutung:

4.21 Definition (Mehrdeutigkeit)

*Eine CFG G heißt **mehrdeutig** (ambiguous) genau dann, wenn es für mindestens ein Wort $w \in L(G)$ zwei verschiedene Ableitungsbäume gibt. Andernfalls heißt G **eindeutig**.*

*Eine kontextfreie Sprache $L \in \mathcal{C}f$ heißt **eindeutig** (unambiguous) genau dann, wenn es eine eindeutige CFG G mit $L = L(G)$ gibt. Andernfalls heißt L **mehrdeutig**.*

An einer konkreten Grammatik lässt sich nicht ohne weiteres feststellen, ob die erzeugte Sprache eindeutig ist, ob also eine äquivalente eindeutige Grammatik existiert.

Nicht nur, dass ein einzelnes Wort einer Grammatik eventuell mehr als einen Ableitungsbaum besitzt, es gibt in der Regel zu einem bestimmten Ableitungsbaum auch mehrere verschiedene Ableitungen. Diese unterscheiden sich alle nur in der Reihenfolge der jeweils ersetzten Nonterminale. Sicher ist jedoch, dass immer dann, wenn in der Ableitung das jeweils am weitesten links stehend Nonterminal als nächstes ersetzt wird, diese sogenannte Linksableitung für das Wort, zu dem der Ableitungsbaum gehört, durch diesen eindeutig bestimmt ist.

4.22 Definition (Linksableitung)

*Eine Ableitung in einer CFG heißt **Linksableitung** (bzw. **Rechtsableitung**) genau dann, wenn in jedem Schritt der Ableitung das ersetzte Zeichen das am weitesten links (bzw. rechts) stehende Nonterminal ist. Die zugehörigen Ableitungsrelationen werden mit $\underset{\text{li}}{\Longrightarrow}$ und $\underset{\text{re}}{\Longrightarrow}$ bezeichnet.*

Die Eindeutigkeit der Ableitungsbäume ist gleichbedeutend mit der Eindeutigkeit der Linksableitung jeder einzelnen Ableitung. Die Dyck-Sprache D_1 ist wegen der Existenz der eindeutigen CFG G_2 aus Beispiel 4.20 eine eindeutige Sprache. Wir werden später sehen, dass alle deterministischen kontextfreien Sprachen eindeutige Sprachen sind. Die Definition der Familie der deterministischen kontextfreien Sprachen werden wir mit Hilfe eines geeigneten Automatenmodells in Kapitel 5 geben. Es ist allerdings festzustellen, dass nicht alle kontextfreien Sprachen eindeutig sind. Wir formulieren dieses Ergebnis ohne Beweis:

4.23 Theorem

Die Sprache $L_M := \{a^r b^s c^t \mid \exists r, s, t \in \mathbb{N} : \ r = s \vee s = t\}$ ist kontextfrei aber nicht eindeutig.

Für die eindeutige Syntaxanalyse bei kontextfreien Sprachen ist es insbesondere bei der sogenannten „top-down“-Analyse wichtig, dass in der verwendeten Grammatik keine linksrekursive Produktion der Art $A \longrightarrow Aw$ vorkommt.

4.24 Definition (Linksrekursion)

*Jede Produktion $A \longrightarrow w$, bei der das Nonterminal A auf der linken Seite vorkommt, nennen wir eine **A-Produktion**. Diese A-Produktion heißt **linksrekursiv**, wenn für die rechte Seite $w = Av$ gilt.*

Wir werden im Folgenden zeigen, dass jede kontextfreie Sprache von einer Grammatik generiert werden kann, die keine linksrekursive Produktion enthält.

4.25 Theorem
Zu jeder CFG G gibt es eine äquivalente CFG G', deren Produktionen nicht linksrekursiv sind.

Beweis: Sei $G := (V_N, V_T, P, S)$ eine CFG und $A \longrightarrow Au_1 \mid Au_2 \mid \ldots \mid Au_r$ alle linksrekursiven A-Produktionen und $A \longrightarrow v_1 \mid v_2 \mid \ldots \mid v_s$ die verbleibenden A-Produktionen. Eine neue CFG $G' := (V'_N, V'_T, P', S')$ wird aus G gebildet durch Hinzufügen des neuen Nonterminals $\overline{A}$ und Ersetzen aller linksrekursiven A-Produktionen von G durch die Produktionen: $A \longrightarrow v_1\overline{A} \mid v_2\overline{A} \mid \ldots \mid v_s\overline{A}$ und $\overline{A} \longrightarrow u_1\overline{A} \mid u_2\overline{A} \mid \ldots \mid u_r\overline{A} \mid u_1 \mid u_2 \mid \ldots \mid u_r$. Dass eine beliebige Linksableitung $A \Longrightarrow Au_{i_1} \Longrightarrow Au_{i_2}u_{i_1} \Longrightarrow \cdots \Longrightarrow Au_{i_p} \ldots u_{i_2}u_{i_1} \Longrightarrow v_ju_{i_p} \ldots u_{i_2}u_{i_1}$ mit A-Produktionen von G, ersetzt werden kann durch $A \Longrightarrow v_j\overline{A} \Longrightarrow v_ju_{i_p}\overline{A} \Longrightarrow \cdots \Longrightarrow v_ju_{i_p} \ldots u_{i_2}\overline{A} \Longrightarrow v_ju_{i_p} \ldots u_{i_2}u_{i_1}$, ist offensichtlich. Die umgekehrte Transformation ergibt sich ebenso leicht. Wenn diese Konstruktion nun für alle Nonterminale A von G nacheinander in einer beliebig festgelegten Reihenfolge durchgeführt wird, erhalten wir die gewünschte äquivalente CFG ohne linksrekursive Produktionen. □

Ausgehend von dieser Konstruktion definieren wir eine weitere Normalform für kontextfreie Grammatiken:

4.26 Definition (Greibach-Normalform)
*Eine CFG $G := (V_N, V_T, P, S)$ ist in **Greibach-Normalform** genau dann, wenn $P \subseteq V_N \times V_T \cdot V_N^*$ ist.*

Von Normalformen wird in der Regel verlangt, dass jedes Objekt aus einer hinreichend großen durch eine Eigenschaft charakterisierten Menge von Objekten in diese Form transformierbar ist. Wir zeigen, dass jede CFG effektiv in eine Greibach-Normalform transformiert werden kann, welche (mit Ausnahme des leeren Wortes) dieselbe Sprache erzeugt.

4.27 Theorem
Zu jeder CFG G kann effektiv eine CFG G' in Greibach-Normalform konstruiert werden, für die $L(G') = L(G) \setminus \{\lambda\}$ gilt.

Beweis(nach Sheila Greibach): Wir konstruieren G' in mehreren Schritten: Zunächst wird G in eine CFG G_1 umgeformt, die keine Kettenregeln besitzt, und bei der $S \longrightarrow \lambda$ die einzige λ-Produktion ist, welches dann auch in keiner rechten Seite einer Produktion mehr vorkommt. Alle anderen Regeln sind in Chomsky-Normalform. D.h. G_1 ist in erweiterter Chomsky-Normalform.

Die nötigen Verfahren waren Bestandteil des Beweises von Theorem 4.13. Wir erhalten so die CFG $G_1 := (V_{N,1}, V_T, P_1, S_1)$. Nehmen wir nun an, dass o.B.d.A. $V_{N,1} := \{A_1, A_2, \ldots, A_n\}$ und $S_1 := A_1$ gesetzt wird.

Wichtig für den Beweis ist die angenommene Ordnung auf dem Nonterminalalphabet.

Mit Algorithmus 4.28 werden wir durch sukkzessive Veränderung der Regelmenge P_1 aus G_1 eine CFG $G_2 := (V_{N,2}, V_T, P_2, S_2)$ konstruieren, für die aus $A_i \longrightarrow A_jw \in P_2$ stets $j > i$ folgt. Ein zweites Verfahren (Algorithmus 4.29) erzeugt dann aus G_2 die gesuchte CFG in Greibach-Normalform.

4.28 Algorithmus

```
begin
  for k := 1 to n do
    for j := 1 to k − 1
      for jede Produktion der Form A_k ⟶ A_j u do
        begin
          for alle Produktionen A_j ⟶ v do
            füge neue Produktion A_k ⟶ vu hinzu
            und entferne die Produktion A_k ⟶ A_j u;
        end
      end (* for Produktionen A_k ⟶ A_j u *)
    end (* for j *)
    for jede Produktion der Form A_k ⟶ A_k u do
      begin
        definiere neues Nonterminal B_k und ergänze neue Produktionen
        B_k ⟶ u und B_k ⟶ uB_k und
        entferne die Produktion A_k ⟶ A_k u
      end;
      for jede Produktion der Form A_k ⟶ u,
        bei der u nicht mit dem Nonterminal A_k beginnt do
        füge die Produktion A_k ⟶ uB_k hinzu
      end (* for Produktionen A_k ⟶ u *)
    end (* for Produktionen A_k ⟶ A_k u *)
  end (* for k *)
end (* Algorithmus *)
```

Die neu entstandene Regelmenge wird jetzt mit P_2 bezeichnet.

Der Aufruf der Schleife „**for** jede Produktion der Form $A_k \longrightarrow u$“ innerhalb der Schleife „**for** jede Produktion der Form $A_k \longrightarrow A_k u$ **do**“ in Algorithmus 4.28 ist hier richtig aber in [Hopcroft&Ullman], auch in der deutschen Ausgabe, falsch!)

Es ist nicht schwer nachzuprüfen, dass folgendes gilt: Wenn $A_i \longrightarrow w$ für $1 \leq i \leq n$ eine Produktion in P_2 ist, so beginnt w entweder mit einem Terminal oder einem Nonterminal A_j mit $j > i$, oder es handelt sich um die λ-Produnktion $A_1 \longrightarrow \lambda$. Insbesondere beginnen die rechten Seiten w der Produktionen $A_n \longrightarrow w \in P_2$ stets mit einem Terminalsymbol! Weiter gilt für jede Produktion $B_i \longrightarrow w' \in P_2$, dass w' entweder mit einem Terminal oder dem Nonterminal A_i beginnt. Fortgesetzte Substitution der Variablen A_k mit Index $k > j$, an erster Stelle einer rechten Seite von A_j-Produktionen durch rechte Seiten von A_k-Produktionen ergibt eine Situation, nach der jede rechte Seite einer A_k-Produktion mit einem Terminalsymbol beginnt. Dies wird durch Algorithmus 4.29 erreicht.

4.29 Algorithmus

Als Eingabe wird die mit Algorithmus 4.28 erzeugte CFG G_2 erwartet.

begin
  for $k := n$ **downto** 2 **do** (* *k wird schrittweise um 1 verringert* *)
    for $j := k - 1$ **downto** 1 **do** (* *j wird schrittweise um 1 verringert* *)
      Für jede Produktion $A_k \longrightarrow w \in P_2$ und jede Produktion
      $A_j \longrightarrow A_k v \in P_2$ füge die Produktion $A_j \longrightarrow wv$ zu P_2 hinzu und
      streiche danach $A_j \longrightarrow A_k v$ aus P_2 heraus.
    end ($*$ *for j* $*$)
  end ($*$ *for k* $*$)
  Nenne die so erhaltene Regelmenge P_3
end ($*$ *Algorithmus* $*$)

Nach Anwendung von Algorithmus 4.29 beginnt also jede rechte Seite einer Produktion ungleich $A_1 \longrightarrow \lambda$ in G' mit einem Terminalsymbol und $P_3 \setminus \{A_1 \longrightarrow \lambda\}$ ist fast schon die gewünschte Produktionenmenge von G'. Da die rechte Seite w' einer B_i-Produktion $B_i \longrightarrow w' \in P_2$ noch mit dem Nonterminal A_i beginnen kann, müssen diese für jedes $1 \leq i \leq n$ durch die entsprechenden Seiten der A_i-Produktionen ersetzt werden. So erhält man die Produktionenmenge P' und die gewünschte Grammatik ist konstruiert. □

4.5 Das Pumping-Lemma der kontextfreien Sprachen

Bei den regulären Mengen hatten wir uns die Frage gestellt, ob es wohl auch formale Sprachen gibt, die nicht regulär sind. Diese Frage können wir nun natürlich auch für die kontextfreien Sprachen stellen und erhalten eine ähnliche Charakterisierung, wie bei den regulären Mengen. Wir beweisen auch für die kontextfreien Sprachen ein *Pumping-Lemma*, das sogenannte „*uvwxy*-Theorem".

4.30 Theorem (Bar-Hillel, Perles, Shamir 1961)
Für jede kontextfreie Sprache $L \in \mathcal{C}f$ gibt es eine Zahl $n \in \mathbb{N}$, so dass jedes Wort $z \in L$ mit $|z| \geq n$ eine Zerlegung $z = uvwxy$ besitzt, für die folgendes gilt:

$$\textit{(i)}: |vx| \geq 1 \qquad \textit{(ii)}: |vwx| \leq n \qquad \textit{(iii)}: \forall i \geq 0 : \ uv^i wx^i y \in L$$

Beweis: Sei $L \in \mathcal{C}f$ beliebig und $G := (V_N, V_T, P, S)$ eine CFG in Chomsky-Normalform für $L \setminus \{\lambda\}$. Setze $n := 2^k$, wobei $k = |V_N|$ ist. Betrachte den Ableitungsbaum für ein Wort $z \in L$ mit $|z| \geq n$. Bis auf die letzten Schritte, mit denen die Terminale erzeugt werden, ist dieser ein Binärbaum, d.h. jeder Knoten hat entweder zwei oder keinen Nachfolger. Die Zahl der Knoten im Ableitungsbaum mit genau einem Nachfolger ist genau $|z|$. Mit Induktion beweist man leicht, dass für jeden Binärbaum (ohne Knoten mit nur einem Nachfolger) folgende Beziehung gilt:

$$p \leq 2^q \tag{4.2}$$

wobei p die Zahl der Blätter bezeichnet und q die Anzahl der Kanten des längsten Pfades von der Wurzel zu einem Blatt. Also folgt, mit $|z| \geq 2^k$ sofort $2^q \geq |z| \geq 2^k$ oder $q \geq k$.

Da auf dem längsten Pfad mit q Kanten in dem Ableitungsbaum zu z genau $q + 1$ Knoten liegen, die mit Nonterminalen beschriftet sind, die Anzahl aller zur Verfügung stehenden Nonterminale k aber

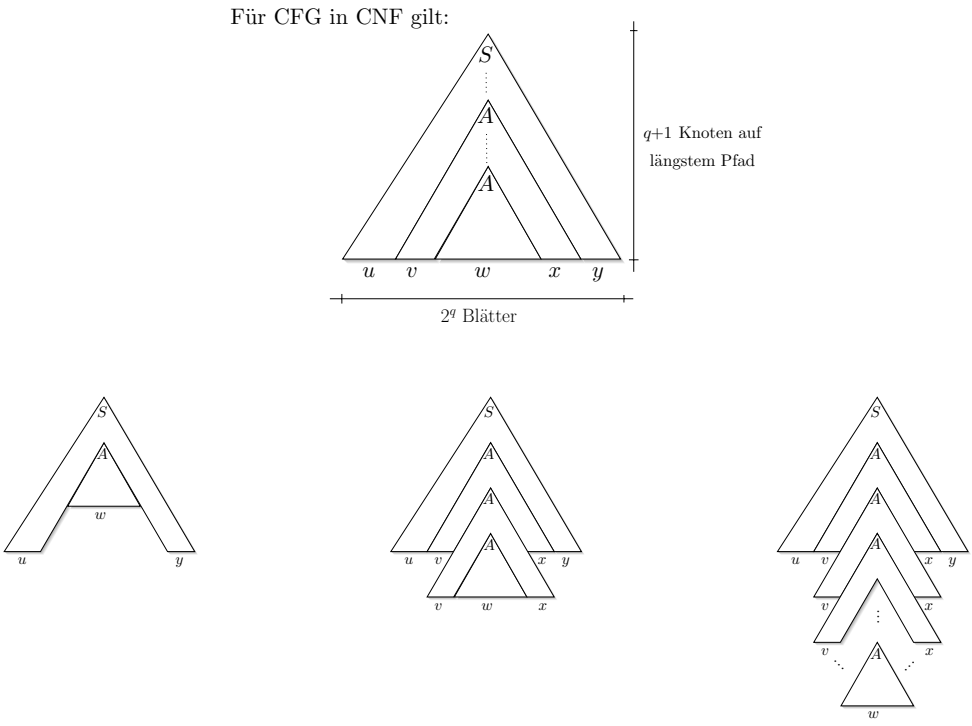


Abbildung 4.2: Zur Aussage des Pumping-Lemma

kleiner als $q + 1$ ist, kommt also auf mindestens einem Pfad in dem Ableitungsbaum zu z mindestens ein Nonterminal doppelt vor. Wir wählen dasjenige Symbol, welches von den Blättern her gesehen sich zum ersten Mal wiederholt. Dieses sei hier mit A bezeichnet, und ist bei jedem Auftreten auf dem Pfad die Wurzel eines A-Ableitungsbaumes. Wegen der Wahl des zuletzt wiederholten Nonterminals A, ist sichergestellt, dass der Pfad bis zum vorletzten A höchsten k Kanten besitzt.

Der A-Ableitungsbaum vom vorletzten A erzeugt das Teilwort vwx von z und der letzte A-Ableitungsbaum generiert das w. Wegen der Beziehung (4.2) folgt $|vwx| \leq n = 2^k$ und $|vx| \geq 1$ folgt daraus, dass der vorletzte A-Knoten zwei Nachfolger besitzt, von denen nur einer für den Pfad zum letzten A-Knoten benötigt wird. Es ist lediglich noch Eigenschaft (iii): $\forall i \geq 0 : \ uv^iwx^iy \in L$ zu zeigen. Der A-Ableitungsbaum am vorletzten A-Knoten kann wiederholt an der Position des letzten A-Knotens eingesetzt werden. Dadurch ergeben sich Ableitungen der Form:

$$S \overset{*}{\underset{G'}{\Longrightarrow}} uAy \overset{*}{\underset{G}{\Longrightarrow}} uvAxy \overset{*}{\underset{G'}{\Longrightarrow}} uvvAxxy \overset{*}{\underset{G}{\Longrightarrow}} uv^iwx^iy$$

Aber auch $i = 0$ ist möglich, indem der zweite A-Ableitungsbaum beim vorletzten A-Knoten eingehängt wird: $S \overset{*}{\underset{G}{\Longrightarrow}} uAy \overset{*}{\underset{G'}{\Longrightarrow}} uwy$.

Abbildung 4.2 veranschaulicht die Argumentation.

□

Da jede kontextfreie Sprache von einer kontextfreien Grammatik in Chomsky-Normalform erzeugt wird, kann für die Zahl n aus dem Pumping-Lemma die Anzahl $|V_N|$ der Nonterminale einer solchen Gram-

matik zugrundegelegt werden, so dass sich n als $2^{|V_N|}$ ergibt.

> Obwohl die Folgerung des Pumping-Lemmas (d.h. die Aufteilbarkeit der „langen“ Wörter mit den angegebenen Eigenschaften) für alle kontextfreien Sprachen gilt, gibt es auch nicht-kontextfreie Mengen, für die diese Aussagen gelten. Daher kann mit Hilfe des Pumping-Lemmas nicht bewiesen werden, dass eine Menge tatsächlich kontextfrei ist, sondern nur umgekehrt, dass sie *nicht* kontextfrei sein kann!

4.31 Beispiel

Wir wenden das $uvwxy$-Theorem an, um zu zeigen, dass folgende Sprachen nicht kontextfrei sein können: $L_1 := \{a^n b^n c^n \mid n \in I\!N\}$ und $L_2 := \{a^k b^{k^2} \mid k \in I\!N\}$. Der Beweis ist indirekt, d.h., wir nehmen an, L_1 (bzw. L_2) sei kontextfrei und werden dann einen Widerspruch herleiten, so dass wir diese Annahme widerrufen müssen.

Wenn $L_1 \in \mathcal{C}f$, dann gilt die Folgerung aus dem $uvwxy$-Theorem und es existiert eine Konstante $n \in I\!N$, so dass jedes $z \in L_1$ mit $|z| \geq n$ eine Zerlegung $z = uvwxy$ besitzt mit den drei Eigenschaften des $uvwxy$-Theorems. Wir wählen das Wort $z := a^n b^n c^n$. Es ist offensichtlich, dass bei keiner Aufteilung dieses Wortes in die Faktoren $uvwxy$ das Teilwort v die Form $a^i b^j$ oder $b^i c^j$ haben kann, weil dann sofort $uvvwxxy \notin a^* b^* c^*$ wäre. Gleiches gilt für das Teilwort x und daher sind als mögliche Aufteilungen nur noch die Fälle denkbar, bei denen $v \in a^* + b^* + c^*$ wie auch $x \in a^* + b^* + c^*$ jeweils nur aus einem einzelnen Symbol gebildet werden. Da $|vx| \neq 0$ folgt daraus, dass beim Pumpen der Teilwörter v und x höchstens zwei, mindestens jedoch eines der zwei Teilwörter a^n oder c^n **nicht** verändert werden kann. (Wegen $|vwx| \leq n$ kann es nicht sein, dass a^n und c^n verändert werden, nicht jedoch b^n!) Also gibt es im Widerspruch zur Annahme keine passende Zerlegung von $z := a^n b^n c^n$ und $L_1 := \{a^n b^n c^n \mid n \in I\!N\} \notin \mathcal{C}f$ folgt schlüssig.

Wenn $L_2 \in \mathcal{C}f$, dann existiert $n \in I\!N$, so dass jedes $z \in L_2$ mit $|z| \geq n$ eine Zerlegung $z = uvwxy$ besitzt mit den Eigenschaften des $uvwxy$-Theorems. Wir probieren es mit $z = a^n b^{n^2}$. Offensichtlich kann weder v noch x von der Form $a^p b^p$ mit $p \geq 1$ sein, denn dann wäre das Wort $uvvwxxy$ kein Teilwort mehr von $a^* b^*$, also auch nicht mehr in L. Ebenfalls kann nicht $v \in b^*$ gelten, denn dann wäre auch $x \in b^*$ und somit würde schon das Wort $uwy = a^n b^{n^2 - r}$ für ein $r \geq 1$ nicht mehr in L liegen. Wäre nun $v = a^p$ für $p \geq 1$ so könnte für x entweder (i) $x = a^j$ oder (ii) $x = b^j$ gelten. Im Fall (i) bedeutet dies $uwy = a^{n-p-j} b^{n^2} \notin L$. Also kann höchstens (ii) gelten. Dann jedoch ergibt sich $\forall i \in I\!N : \; uv^i wx^i y = a^{n+i \cdot p} b^{n^2 + i \cdot j}$. Für $i = n$ und $j \leq n$ ergibt sich auch hier ein Widerspruch und L_2 ist nicht kontextfrei.

4.6 Ein Entscheidbarkeitsresultat

So wie bei den regulären Mengen, interessiert auch für kontextfreie Grammatiken bzw. kontextfreie Sprachen das Wortproblem. Wir definieren es zunächst und diskutieren dann die Entscheidbarkeit dieses Problems.

4.32 Definition (Wortproblem für eine kontextfreie Sprache L)

Eingabe: *Ein Wort* $w \in \Sigma^*$

Frage: *Gilt* $w \in L$*?*

4.33 Theorem

Das spezielle Wortproblem für kontextfreie Sprachen ist entscheidbar, d.h. es gibt ein Verfahren, das zu einer durch eine CFG G spezifizierten Sprache $L = L(G)$ für jedes w feststellt, ob $w \in L$ gilt.

Beweis: O.B.d.A. sei $L = L(G)$ durch eine CFG $G = (V_N, V_T, P, S)$ in Greibach-Normalform gegeben, und $w \in V_T^*$. Da in einer Ableitung $S \overset{*}{\underset{G}{\Longrightarrow}} v$ mit jedem Schritt ein weiteres Nonterminal erzeugt wird, brauchen nur die endlich vielen Ableitungen der Länge $|w|$ daraufhin überprüft zu werden, ob eine darunter ist, die das Wort w generiert. □

Das eben verwendete Verfahren erfordert, bezogen auf die Länge des Eingabewortes w und die Größe der Grammatik G, im allgemeinen exponentiellen Aufwand. Man kann aber auf diese Weise auch das allgemeine Wortproblem lösen, in dem man zu einer beliebigen vorgegebenen CFG zunächst deren Greibach-Normalform konstruiert und dann dieses Verfahren anwendet. Dieses ist aber allemal exponentiell im Aufwand, bezogen auf die Größe der gegebenen Eingabedaten. Besser, aber auch noch nicht optimal für das spezielle Wortproblem, ist das Verfahren von Cocke, Younger und Kasami. Dieses basiert auf der Idee, dass in einer Grammatik in Chomsky-Normalform ein Wort $w = x_1 x_2 \dots x_n$ immer dann aus dem Nonterminal A abgeleitet werden kann, wenn Wörter $x_1 x_2 \dots x_j$ und $x_{j+1} x_{j+2} \dots x_n$ aus den Nonterminalen B bzw C abgeleitet werden können, und die Regel $A \longrightarrow BC$ in der Grammatik existiert. Mit Methoden der dynamischen Programmierung gewinnt man so einen Algorithmus, der in Polynomzeit arbeitet.

4.34 Theorem

Das spezielle Wortproblem für kontextfreie Sprachen in Chomsky-Normalform ist in Polynomzeit lösbar.

Beweis: Es gibt einen Algorithmus, der für eine kontextfreie Grammatik $G = (V_N, V_T, P, S)$ in Chomsky-Normalform und ein Wort $w \in V_T^*$ in $c_1 \cdot |P| \cdot |w|^3 + c_2$ Schritten entscheidet, ob $w \in L(G)$ ist. Hierbei sind c_1 und c_2 Konstanten, die nicht von G abhängen.

4.35 Algorithmus (Cocke-Younger-Kasami)

Eingaben: Eine kontextfreie Grammatik $G = (V_N, V_T, P, S)$ in Chomsky-Normalform und ein Wort $w \in V_T^*$ mit $w = x_1 x_2 \dots x_n$, für $x_i \in V_T$.

Für alle $i, j \in \{0, 1, 2, \dots n\}$ mit $0 \leq i \leq j$ definieren wir Mengen $V_{i,j} \subseteq V_N$ durch $V_{i-1,j} := \{A \in V_N \mid A \overset{*}{\Longrightarrow} x_i x_{i+1} \dots x_j\}$, die sukzessive als die Elemente der Felder $V_{i,j}$ einer $(n+1) \times (n+1)$-Matrix mit Elementen aus V bestimmt werden.

for $i := 1$ **to** n **do**
  $V_{i-1,i} := \{A \in V_N \mid A \longrightarrow x_i \in P\}$
end (∗ *for i* ∗)
Setze $V_{0,0} := \emptyset$ und $\forall 1 \leq i \leq n : \ V_{i,i} := \{x_i\}$
for $m := 2$ **to** n **do**
  for $i := 0$ **to** $n - m$ **do**
    $j := i + m$
    $V_{i,j} := \{A \in V_N \mid A \longrightarrow BC \in P : \ \exists k : i < k < j : B \in V_{i,k} \ \wedge \ C \in V_{k,j}\}$
  end (∗ *for* ∗)
end (∗ *for* ∗)

Am Ende ist $V_{0,n}$ bestimmt, und es gilt $S \in V_{0,n}$ genau dann, wenn $w \in L(G)$ ist. Das ist auch ohne formal strengen Beweis einzusehen, denn es ist ja für $w = x_1 x_2 \dots x_n$ gerade $S \Longrightarrow w$ für $n = 1$, wenn $S \longrightarrow x_1 \in P$ und bei $n \geq 2$, falls eine Produktion $S \longrightarrow BC \in P$ existiert, so dass $B \in V_{0,k}$ und $C \in V_{k,n}$ für ein k gilt, d.h. Ableitungen $B \stackrel{*}{\Longrightarrow} x_1 x_2 \dots x_k$ und $C \stackrel{*}{\Longrightarrow} x_{k+1} x_{k+2} \dots x_n$ existieren. □

Für eine feste Grammatik G arbeitet das CYK-Verfahren in $c_1 \cdot |w|^3 + c_2$ Schritten, wobei c_1 und c_2 Konstante sind, die nicht von G, sondern nur von der Implementation der einzelnen Schritte abhängen. Der CYK-Algorithmus läßt sich auch wie folgt in eine Matrixmultiplikationsaufgabe umdeuten.

Zu gegebener CFG $G = (V_N, V_T, P, S)$ und für Mengen $U, W \subseteq V_N$ werden die Operationen $*$ und $+$ definiert:

$$\begin{aligned} U * W &:= \{A \in V_N \mid \exists B \in U : \exists C \in W : \ A \longrightarrow BC \in P\} \\ U + W &:= U \cup W \end{aligned}$$

Es gilt nun $(U_1 + U_2) * W = U_1 * W + U_2 * W$ und die Berechnung der Mengen $V_{i,j}$ geschieht wie folgt:

$$\begin{aligned} V_{i,j} &:= \sum_{k=1}^{n} V_{i,k} * V_{k,j} \text{ mit} \\ V_{i-1,i} &:= \{A \in V_N \mid A \longrightarrow x_i \in P\} \text{ und} \\ V_{k,j} &:= \emptyset \text{ falls } k > j. \end{aligned}$$

Es gilt dann für $j - i \geq 2$ gerade:

$$V_{i,j} := \sum_{k=i+1}^{j-1} V_{i,k} * V_{k,j}$$

Das spezielle Wortproblem für allgemeine kontextfreie Sprachen wird häufig mit dem hier nicht vorgestellten Verfahren nach Earley (1970) entschieden. Dieses benötigt im schlechtesten Fall ebenfalls $c_1 \cdot |w|^3 + c_2$ Schritte, hat aber den Vorteil, dass dieses Verfahren für eindeutige kontextfreie Sprachen nur $c_1 \cdot |w|^2 + c_2$ Schritte benötigt und sogar häufig in Linearzeit arbeitet.

Unter Verwendung der schnellen Matrixmultiplikation von Valiant und Strassen ist dieses Problem sogar in $|w|^{2,7}$ zu lösen. Spätere Verbesserungen ergaben Laufzeiten proportional zu $|w|^{2,4\dots}$ und sogar zu $|w|^{2,36\dots}$. Da dieses Problem in den meisten praktischen Fällen gar nicht für beliebige kontextfreie Sprachen gelöst werden muss, ist die Frage nach dem schnellsten Algorithmus für dieses Problem aber ohnehin eher von theoretischem Interesse.

Die Syntax von Programmiersprachen basiert in der Regel auf eindeutigen oder gar deterministischen Sprachen, die durch weitere Einschränkungen letztlich sogar nicht mehr kontextfrei bleiben. Trotzdem ist das Herz jedes Compilers ein Analyseverfahren für spezielle, meist deterministische, kontextfreie Grammatiken. Für die Definition von deterministischen kontextfreien Sprachen sei wiederum auf Kapitel 5 verwiesen.

4.7 Wichtige Abschlusseigenschaften

Im Folgenden werden wir untersuchen, ob die Familie der kontextfreien Sprachen ähnlich gute Eigenschaften besitzt, wie die Familie der regulären Mengen.

4.36 Theorem

Die Familie $\mathcal{C}f$ ist abgeschlossen gegenüber den drei regulären Operatoren $\vee$, $\cdot$ und *:

1. *Vereinigung, d.h. $\mathcal{C}f \vee \mathcal{C}f \subseteq \mathcal{C}f$*
2. *Komplexprodukt, d.h. $\mathcal{C}f \cdot \mathcal{C}f \subseteq \mathcal{C}f$*
3. *Kleene'sche Hülle (Sternbildung), d.h. $\mathcal{C}f^* \subseteq \mathcal{C}f$*

Beweis: Für $i \in \{1, 2\}$ seien kontextfreie Sprachen L_i gegeben durch $L_i := L(G_i)$ für $G_i := (V_{i,N}, V_{i,T}, P_i, S_i)$.

1. $L_1 \cup L_2 = L(G_3)$ für $G_3 := (V_{1,N} \uplus V_{2,N} \uplus \{S_3\}, V_{1,T} \cup V_{2,T}, P_3, S_3)$ mit $P_3 := P_1 \cup P_2 \cup \{S_3 \longrightarrow S_1,\ S_3 \longrightarrow S_2\}$
2. $L_1 \cdot L_2 = L(G_4)$ für $G_4 := (V_{1,N} \uplus V_{2,N} \uplus \{S_4\}, V_{1,T} \cup V_{2,T}, P_4, S_4)$ mit $P_4 := P_1 \cup P_2 \cup \{S_4 \longrightarrow S_1 S_2\}$
3. $L_1^* = L(G_5)$ für $G_5 := (V_{1,N} \uplus \{S_5\}, V_{1,T}, P_5, S_5)$ mit $P_5 := P_1 \cup \{S_5 \longrightarrow S_1 S_5,\ S_5 \longrightarrow \lambda\}$

□

4.37 Theorem

*Die Familie der kontextfreien Sprachen ist **nicht** abgeschlossen gegenüber folgenden Operatoren:*

1. *Durchschnittsbildung*
2. *Komplementbildung*
3. *Bildung einer Mengendifferenz*

Beweis: Wir zeigen zunächst 1. durch Angabe eines Gegenbeispiels:

$L_1 := \{a^n b^n \mid n \in I\!N\}$ wie auch $L_2 := \{b^m c^m \mid m \in I\!N\}$ sind kontextfreie Sprachen, deren Grammatiken leicht konstruiert werden können. Wegen $\mathcal{R}eg \subseteq \mathcal{C}f$ und Theorem 4.36 sind dann auch die Sprachen $L_3 := L_1 \cdot \{c\}^*$ und $L_4 := \{a\}^* \cdot L_2$ kontextfrei. Wäre die Familie $\mathcal{C}f$ gegenüber Durchschnittsbildung abgeschlossen, so wäre nun auch $L_5 := L_3 \cap L_4 = \{a^n b^n c^n \mid n \in I\!N\} \in \mathcal{C}f$. L_5 ist nun aber identisch zu der Sprache L_1 aus Beispiel 4.31 auf Seite 110, für die wir bereits mit Hilfe des Pumping-Lemmas gezeigt haben, dass sie nicht kontextfrei ist. Somit ist die Menge L_5 also nicht kontextfrei und die Durchschnittsbildung kann keine Abschlusseigenschaft für $\mathcal{C}f$ sein.

2. folgt aus 1.:

$$\begin{aligned}
\overline{L_5} &= \{a,b,c\}^* \setminus L_5 = L_6 \cup L_7 \cup L_8 \text{ mit} \\
L_6 &:= \{a^r b^s c^t \mid r \neq s,\ r,s,t \in I\!N\}, \\
L_7 &:= \{a^r b^s c^t \mid s \neq t,\ r,s,t \in I\!N\} \text{ und} \\
L_8 &:= \overline{a^* b^* c^*}
\end{aligned}$$

Letztere Menge L_8 ist regulär und damit kontextfrei. L_6 wird mit nachstehenden Regeln erzeugt, bei denen alle Großbuchstaben Nonterminale sind:

$$\begin{array}{lcllcl} S_6 & \longrightarrow & ABC, & S_6 & \longrightarrow & A'B'C, \\ A & \longrightarrow & aA, & A & \longrightarrow & a, \\ B & \longrightarrow & aBb, & B & \longrightarrow & \lambda, \\ A' & \longrightarrow & aA'b, & A' & \longrightarrow & \lambda, \\ B' & \longrightarrow & bB', & B' & \longrightarrow & b, \\ C & \longrightarrow & cC, & C & \longrightarrow & \lambda \end{array}$$

Es ist leicht nachzuprüfen, dass gilt:

$$AB \stackrel{*}{\Longrightarrow} a^r b^s c^t, \forall r,s,t \in I\!N \text{ mit } r > s \ \wedge \ A'B' \stackrel{*}{\Longrightarrow} a^r b^s c^t, \forall r,s,t \in I\!N \text{ mit } r < s$$

Eine Grammatik für L_7 ist ganz entsprechend zu bilden. Damit ist $\overline{L_5}$ als Vereinigung kontextfreier Sprachen nach Theorem 4.36 selbst kontextfrei, ihr Komplement jedoch nicht, womit 2. vollständig gezeigt ist.

Da die Komplementbildung eine spezielle Mengendifferenz ist, folgt 3. sofort aus 2. □

Obwohl der Durchschnitt zweier kontextfreier Sprachen in der Regel nicht kontextfrei ist, kann eine etwas schwächere, aber dennoch sehr bedeutende, Abschlusseigenschaft bezüglich der Durchschnittsbildung gezeigt werden. Dieses häufig verwendete Resultat besagt, dass die Familie der kontextfreien Sprachen gegenüber Durchschnittsbildung mit regulären Mengen abgeschlossen ist.

4.38 Theorem
Für $L \in \mathcal{C}f$ und $R \in \mathcal{R}eg$ gilt $L \cap R \in \mathcal{C}f$, kurz $\mathcal{C}f \wedge \mathcal{R}eg \subseteq \mathcal{C}f$.

Beweis: Sei $L = L(G)$ für eine CFG $G = (V_N, V_T, P, S)$ und $R = L(A)$ für einen NFA $A = (Z, \Sigma, K, Z_{\text{start}}, Z_{\text{end}})$. Wir konstruieren die neue CFG $G' := (V'_N, V'_T, P', S')$ mit den neuen Hilfszeichen $V'_N := \{[z, X, z'] \mid z, z' \in Z \text{ und } X \in V_N\} \cup \{S'\}$ und den R und L gemeinsamen Terminalzeichen $V'_T := \Sigma \cap V_T$.

O.B.d.A. nehmen wir an, dass G in Chomsky-Normalform vorliegt und definieren daraus die Regelmenge P'. Für jede Produktion $A \longrightarrow BC \in P$ seien folgende Produktionen in P' enthalten:

$$[z, A, z''] \longrightarrow [z, B, z'][z', C, z''] \ \in P' \ \text{ für alle } z, z', z'' \in Z$$

Weiterhin seien für jede Produktion $A \longrightarrow a$ in P folgende Produktionen in P':

$$[z, A, z'] \longrightarrow a \ \in P' \ \text{ für alle } (z, a, z') \in K$$

Ausserdem werden als Startproduktionen zu P' folgende Produktionen hinzugenommen:

$$S' \longrightarrow [z, S, z'] \ \in P' \ \text{ für alle } z \in Z_{\text{start}} \text{ und } z' \in Z_{\text{end}}$$

Es gilt $L(G') = L \cap R$, denn in jeder abgeleiteten Satzform in G' bildet die erste Komponente des Nonterminals eine Folge von Zuständen, die mit dem Startzustand von A beginnt und in der zweiten Komponente des letzten Zeichens mit einem Endzustand endet. Genau dann, wenn die Kante (z, a, z')

in K vorkommt, kann das Terminalsymbol a aus $[z, A, z']$ erzeugt werden, so dass zu jedem generierten Wort $w \in L$ immer auch ein Erfolgspfad in A gehört. Dank der vielen Produktionen in G' wird auch jeder mögliche Erfolgspfad von A in den Satzformen erzeugt, und es folgt $L(G') = L \cap R$. □

Wir betrachten nun als Beispiel für eine indirekte Anwendung des *uvwxy*-Theorems die Sprache $L := \{a^k b^l c^m d^n \mid k, l, m, n \in I\!N : k = 0 \text{ oder } l = m = n\}$. Der Versuch, mit dem Pumping-Lemma zu beweisen, dass L nicht kontextfrei ist, muss fehlschlagen, denn wenn $z \in L$ den Buchstaben a nicht enthält, so gilt dies auch für alle Wörter $uv^i wx^i y$, ganz gleich welche Zerlegung gewählt wurde.

Enthält das Wort z das Symbol a, so wählen wir eine Zerlegung mit $u = v = w = \lambda$, $x = a$ und y als geeignetem Suffix. Alle Wörter der Form $uv^i wx^i y$ sind dann wieder Elemente der Menge L. Also kommt kein Widerspruch zu der Annahme L sei kontextfrei zustande. Trotzdem ist L nicht kontextfrei, und das läßt sich so beweisen:

Wäre L kontextfrei, so wäre auch $L \cap \{a\}\{b\}^*\{c\}^*\{d\}^* = \{ab^n c^n d^n \mid n \in I\!N\}$ eine kontextfreie Sprache (nach Theorem 4.38). Von letzterer Menge kann unter Anwendung des *uvwxy*-Theorems leicht gezeigt werden (siehe Beweis zu 1. von Theorem 4.37), dass sie nicht kontextfrei ist. Also konnte auch L dies nicht sein, obwohl das Pumping-Lemma nicht direkt anwendbar war.

Es sind schärfere Ausformulierungen des Pumping-Lemmas bekannt, mit denen es möglich ist, direkt zu zeigen, dass L nicht kontextfrei ist. Aber auch diese Varianten stellen keine *notwendige Bedingung* für die Eigenschaft einer Menge kontextfrei zu sein dar, d.h. die Folgerung des Pumping-Lemmas kann auch für Sprachen zutreffen, die nicht kontextfrei sind. In eigentlich allen Fällen kommt man mit dem hier verwendeten *uvwxy*-Theorems aus, wenn dazu noch einfache Abschlußeigenschaften der Familie $\mathcal{C}f$, wie oben exemplarisch gezeigt, hinzugenommen werden.