4 Formale Sprachen

4.1 Einführung

Ich erinnere an die Definitionen im Unterkapitel 1.6, die im weiteren Verlauf der Vorlesung von Bedeutung sein werden. Insbesondere werden wir uns mit *formalen Sprachen* über gegebenen *Alphabeten* Σ befassen.

Betrachten wir dazu ein Beispiel.

Beispiel 4.1 Sei $\Sigma = \{(,), +, -, *, /, a\}$, so betrachten wir die Menge *EXPR* der korrekt geklammerten arithmetischen Ausdrücke über diesem Alphabet, wobei a als Platzhalter für beliebige Konstanten und Variablen dienen soll. Zum Beispiel soll gelten:

$$\begin{aligned} (a-a)*a+a\,/\,(a+a)-a &\in \mathit{EXPR} \\ (((a))) &\in \mathit{EXPR} \\ ((a+)-a(&\notin \mathit{EXPR} \end{aligned}$$

Wie das Beispiel zeigt, sind formale Sprachen im Allgemeinen unendliche Mengen von Wörtern. In diesem Kapitel wird es darum gehen, diese unendlichen Mengen durch endliche Konstrukte zu charakterisieren, zu beschreiben. Dazu gehören zum Beispiel *Grammatiken* und *Automaten*.

Zunächst werden wir uns mit Grammatiken beschäftigen, die für die Theorie der Informatik eine ausgezeichnete Rolle spielen, insbesondere im Compilerbau. Allerdings sind die historischen Wurzeln in der Linguistik zu suchen, deshalb an dieser Stelle ein (stark vereinfachtes) Beispiel aus der Linguistik. (Dass die Linguistik nicht so einfach zu formalisieren ist, erkennt man an den Schwierigkeiten, die automatische Übersetzungssysteme (noch?) machen. Noch kann kein solches System einen Dolmetscher ersetzen.)

Beispiel 4.2 Wir betrachten eine Grammatik mit folgenden Regeln, wobei sogenannte *Variable* oder *Phrasen* durch spitze Klammern gekennzeichnet sind. Das heißt, sie gehören eigentlich nicht zum Alphabet, über dem die Sprache definiert werden soll.

$$\begin{aligned}
\text{<Satz>} &\to \text{<Subjekt><Prädikat><Objekt>} \\
\text{<Subjekt>} &\to \text{<Artikel><Attribut><Substantiv>} \\
\text{<Artikel>} &\to \varepsilon \\
\text{<Artikel>} &\to \text{<der>} \\
\text{<Artikel>} &\to \text{<die>} \\
\text{<Artikel>} &\to \text{<das>} \\
\text{<Attribut>} &\to \varepsilon \\
\text{<Attribut>} &\to \text{<Adjektiv>} \\
\text{<Attribut>} &\to \text{<Adjektiv><Attribut>} \\
\text{<Adjektiv>} &\to \text{<kleine>} \\
\text{<Adjektiv>} &\to \text{<bissige>} \\
\text{<Adjektiv>} &\to \text{<große>} \\
\text{<Substantiv>} &\to \text{<Hund>} \\
\text{<Substantiv>} &\to \text{<Katze>} \\
\text{<Prädikat>} &\to \text{<jagt>} \\
\text{<Objekt>} &\to \text{<Artikel><Attribut><Substantiv>}
\end{aligned}$$

Durch die obige Grammatik können wir zum Beispiel folgenden Satz bilden:

der kleine bissige Hund jagt die große Katze

Wie wir diesen Satz mittels der Regeln abgeleitet haben, kann man sehr gut an einem sogenannten *Syntaxbaum* veranschaulichen. Dieser ist in der Abbildung 4.1 dargestellt. Hierbei werden die linke und rechte Seite einer angewendeten Regel durch einen Vater- bzw. Sohnknoten dargestellt.

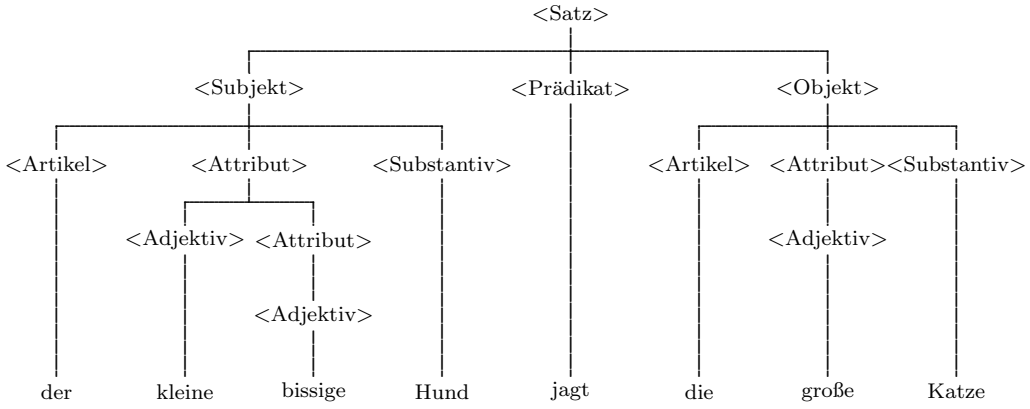

Abbildung 4.1: Syntaxbaum für den Satz „*der kleine bissige Hund jagt die große Katze*“

Wir können natürlich auch noch andere Sätze mittels dieser Grammatik bilden, zum Beispiel

der kleine bissige Hund jagt die große große große Katze

und

die kleine Katze jagt die große große Hund

Ersterer Satz macht deutlich, dass wir mit obiger (endlicher) Grammatik bereits unendlich viele Sätze bilden können. Der letze Satz zeigt die Schwächen und auch Grenzen von Grammatiken auf:

1. Die Grammatik lässt keine Fälle zu und ist somit unzureichend. Diese Schwäche allerdings könnten wir durch eine verbesserte Grammatik (dann aber wesentlich umfangreicher) beheben.
2. Es zeigt sich aber auch eine andere Schwäche, die nicht so einfach zu beheben ist: Ein *syntaktisch* einwandfreier Satz ist im Allgemeinen nicht automatisch *semantisch* korrekt.

4.2 Grammatiken

Wir bemerken, dass im obigen Beispiel zwei Arten von Symbolen auftauchen: Erstens sogenannte *Terminalsymbole*, das sind Symbole, aus denen die Wörter eigentlich bestehen, und sogenannte *Nichtterminalsymbole* oder *Variablen*. Das sind Symbole, die zwar während des Ableitungsprozesses benutzt werden (zum Beispiel <Attribut>), aber im eigentlichen Wort oder Satz nicht mehr auftauchen. In unserem Beispiel besteht jede Regel aus einer linken Seite und einer rechten Seite, wobei die linke Seite hier immer aus genau einer Variablen besteht. Im Allgemeinen kann eine linke Seite auch aus einem Wort mit mehreren Symbolen bestehen. Ein besonderes Symbol in unserer Beispielgrammatik war <Satz>, damit beginnt eine Ableitung. Eine Ableitung wiederum ist eine mehrfache Anwendung der Regeln, wobei eine Anwendung einer Regel bedeutet, dass in einem Wort ein Teilwort (die linke Seite einer Regel) durch die rechte Seite derselben Regel ersetzt wird.

Wir wollen jetzt den Begriff der Grammatik formalisieren.

Definition 4.3 *Eine Grammatik ist ein 4-Tupel $G = (V, \Sigma, P, S)$, wobei*

- *V ein Alphabet ist (Nichtterminalalphabet oder Alphabet der Variablen),*
- *Σ ein Alphabet ist (Terminalalphabet),*
- *$V \cap \Sigma = \emptyset$ gilt,*

- P *eine endliche Teilmenge von* $((V \cup \Sigma)^* \setminus \Sigma^*) \times (V \cup \Sigma)^*$ *ist (Menge der Regeln),*
- $S \in V$ *ist (die Startvariable oder Axiom).*

Die Elemente von P, also die *Regeln* oder auch *Produktionen* sind eigentlich geordnete Paare. Zur besseren Lesbarkeit werden wir aber $u \to v \in P$ für $(u, v) \in P$ schreiben.

Den Begriff der Ableitung haben wir schon informal eingeführt. Wir wollen jetzt definieren, was wir unter der *direkten Ableitung* exakt verstehen wollen.

Definition 4.4 *Sei* $G = (V, \Sigma, P, S)$ *eine Grammatik und* $u, v \in (V \cup \Sigma)^*$ *Wörter. Dann gilt* $u \Longrightarrow_G v$ *(in Worten:* u *erzeugt bezüglich* G *direkt* v*) genau dann, wenn*

(i) $u = \gamma_1 \alpha \gamma_2$ *mit* $\gamma_1, \gamma_2 \in (V \cup \Sigma)^*$,

(ii) $v = \gamma_1 \beta \gamma_2$ *und*

(iii) $\alpha \to \beta \in P$ *ist.*

Falls aus dem Kontext eindeutig hervorgeht, welche Grammatik G gemeint ist, schreiben wir $u \Longrightarrow v$ statt $u \Longrightarrow_G v$.

Wir können $\Longrightarrow$ als Relation in der Menge $(V \cup \Sigma)^*$ ansehen. Dann wollen wir mit $\overset{*}{\Longrightarrow}$ den reflexiven und transitiven Abschluss der Relation $\Longrightarrow$ bezeichnen. Wir können ihn auch elementweise definieren:

Definition 4.5 *Sei* $G = (V, \Sigma, P, S)$ *eine Grammatik und* $u, v \in (V \cup \Sigma)^*$ *Wörter. Dann gilt* $u \overset{*}{\Longrightarrow}_G v$ *genau dann, wenn* $u = v$ *gilt oder es ein* $n \in \mathbb{N}$ *und Wörter* $w_0, w_1, \ldots, w_n$ *gibt, so dass*

$$u = w_0 \Longrightarrow_G w_1 \Longrightarrow_G w_2 \Longrightarrow_G \cdots \Longrightarrow_G w_n = v$$

gilt.

Wiederum schreiben wir $u \overset{*}{\Longrightarrow} v$ statt $u \overset{*}{\Longrightarrow}_G v$, falls es kein Missverständnis geben kann.

Nun sind wir in der Lage, die *erzeugte Sprache* $L(G)$ einer Grammatik G als die Menge aller Wörter zu definieren, die von dieser Grammatik erzeugt wird.

Definition 4.6 *Sei* $G = (V, \Sigma, P, S)$ *eine Grammatik. Die von* G *erzeugte Sprache* $L(G)$ *wird definiert als*

$$L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Longrightarrow}_G w\}.$$

Betrachten wir ein Beispiel.

Beispiel 4.7 Es sei die Grammatik

$$G = (\{E, T, F\}, \{(,), a, +, *\}, P, E)$$

mit

$$P = \{E \to T,\ E \to E + T,\ T \to F,\ T \to T * F,\ F \to a,\ F \to (E)\}$$

gegeben. Diese Grammatik beschreibt eine Teilmenge der Menge *EXPR*, der Menge der exakt geklammerten arithmetischen Ausdrücke aus Beispiel 4.1, nämlich die Teilmenge ohne die Operationen *Division* / und *Subtraktion* $-$. Es gilt zum Beispiel

$$a * a * (a + a) + a \in L(G),$$

denn das Wort $a * a * (a + a) + a$ wird durch die Ableitung

$$\begin{aligned} E &\Longrightarrow E + T \Longrightarrow T + T \Longrightarrow T * F + T \Longrightarrow T * F * F + T \Longrightarrow F * F * F + T \\ &\Longrightarrow a * F * F + T \Longrightarrow a * a * F + T \Longrightarrow a * a * (E) + T \Longrightarrow a * a * (E + T) + T \\ &\Longrightarrow a * a * (T + T) + T \Longrightarrow a * a * (F + T) + T \Longrightarrow a * a * (a + T) + T \\ &\Longrightarrow a * a * (a + F) + T \Longrightarrow a * a * (a + a) + T \Longrightarrow a * a * (a + a) + F \\ &\Longrightarrow a * a * (a + a) + a \end{aligned}$$

aus dem Startwort E erzeugt. Hierbei wurde in jedem Ableitungsschritt immer die am weitesten links stehende Variable ersetzt. Wir können auch eine andere Ableitung für dieses Wort konstruieren, zum Beispiel:

$$\begin{aligned} E &\Longrightarrow E + T \Longrightarrow T + T \Longrightarrow T * F + T \Longrightarrow T * F * F + T \Longrightarrow F * F * F + T \\ &\Longrightarrow a * F * F + T \Longrightarrow a * a * F + T \Longrightarrow a * a * (E) + T \Longrightarrow a * a * (E + T) + T \\ &\Longrightarrow a * a * (T + T) + T \Longrightarrow a * a * (F + T) + T \Longrightarrow a * a * (F + F) + T \\ &\Longrightarrow a * a * (F + F) + F \Longrightarrow a * a * (a + F) + F \Longrightarrow a * a * (a + a) + F \\ &\Longrightarrow a * a * (a + a) + a. \end{aligned}$$

Beiden Ableitungen wird in diesem Beispiel ein und derselbe Syntaxbaum zugeordnet (siehe Abbildung 4.2).

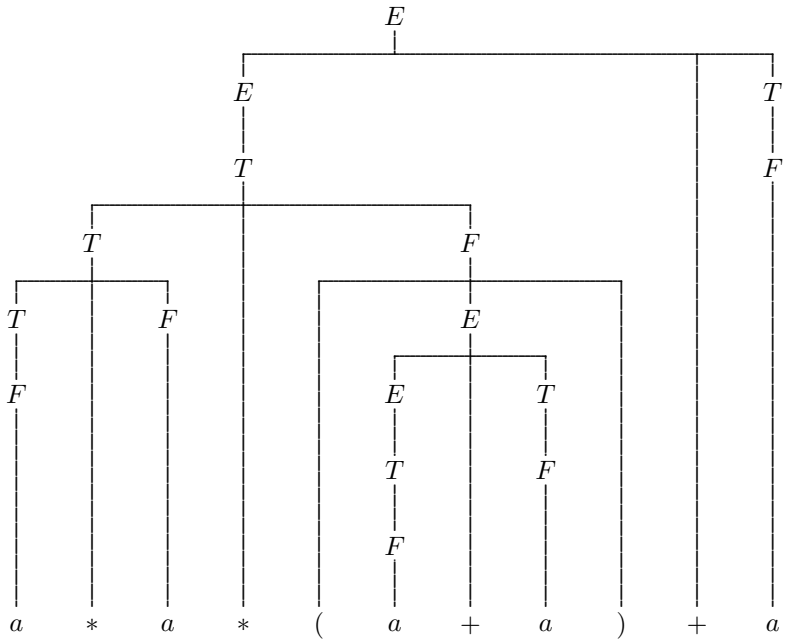

Abbildung 4.2: Ein Syntaxbaum für das Wort $a * a * (a + a) + a$

Betrachten wir ein weiteres, etwas komplexeres Beispiel.

Beispiel 4.8 Es sei die Grammatik

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

mit

$$P = \{S \to aSBC,\ S \to aBC,\ CB \to BC,\ aB \to ab,\ bB \to bb,\ bC \to bc,\ cC \to cc\}$$

gegeben. Wir können zum Beispiel die Ableitung

$$\begin{aligned} S &\Longrightarrow aSBC \Longrightarrow aaSBCBC \Longrightarrow aaaBCBCBC \\ &\Longrightarrow aaaBBCCBC \Longrightarrow aaaBBCBCC \Longrightarrow aaaBBBCCC \\ &\Longrightarrow aaabBBCCC \Longrightarrow aaabbBCCC \Longrightarrow aaabbbCCC \\ &\Longrightarrow aaabbbcCC \Longrightarrow aaabbbccC \Longrightarrow aaabbbccc \end{aligned}$$

aufstellen, also gehört das Wort $aaabbbccc = a^3b^3c^3$ zur erzeugten Sprache $L(G)$, es gilt also $a^3b^3c^3 \in L(G)$.

Nun fragen wir uns natürlich, welche Menge $L(G)$ genau darstellt. Wenn man sich die Ableitung genauer anschaut, vermutet man leicht

$$L(G) = \{a^n b^n c^n \mid n \geq 1\}. \tag{4.1}$$

Der Nachweis dafür muss eigentlich exakt mathematisch geführt werden. Man macht es in zwei Schritten:

(i) Zunächst wird $L(G) \supseteq \{a^n b^n c^n \mid n \geq 1\}$ gezeigt,

(ii) dann $L(G) \subseteq \{a^n b^n c^n \mid n \geq 1\}$.

Uns genügt es allerdings, den Nachweis nur zu skizzieren:

(i) Um $L(G) \supseteq \{a^n b^n c^n \mid n \geq 1\}$ zu zeigen, müssen wir also nachweisen, dass jedes Wort $a^n b^n c^n$ für ein $n \geq 1$ vom Startwort S abgeleitet werden kann. Dazu schauen wir uns obige Ableitung für $a^3b^3c^3$ näher an und sehen, dass sie einfach verallgemeinert werden kann. Zunächst wird $n-1$ mal die Regel $S \to aSBC$ angewendet und einmal die Regel $S \to aBC$. Dann erhält man das Wort $a^n(BC)^n$ (siehe Zeile 1 in obiger Ableitung). Dann werden durch mehrmalige Anwendung der Regel $CB \to BC$ alle B's vor die C's getauscht (Zeile 2). Dann werden durch die Regeln $aB \to ab$ und $bB \to bb$ alle B's in b's umgewandelt (Zeile 3) und schließlich durch die Regeln $bC \to bc$ und $cC \to cc$ alle C's in c's (Zeile 4).

(ii) Schwieriger zu zeigen ist die Behauptung $L(G) \subseteq \{a^n b^n c^n \mid n \geq 1\}$. Das heißt, es ist zu zeigen, dass *nur* Wörter der Form $a^n b^n c^n$ für ein $n \geq 1$ abgeleitet werden können. Dies geschieht eigentlich streng mathematisch (wie eigentlich auch schon die Behauptung (i)) über das Beweisverfahren der vollständigen Induktion. Für uns reicht eine Diskussion in der Hinsicht, dass eine genaue Analyse der Regeln Folgendes zeigt:

Erstens werden nur Worte mit gleicher Anzahl von a's und b's und c's erzeugt. (Sieht man daran, dass durch die erste und zweite Regel jeweils für jedes a auch genau ein B und ein C erzeugt wird. Und die anderen Regeln wandeln höchstens Großbuchstaben in Kleinbuchstaben um, verändern aber nicht die Anzahl!)

Zweitens werden nur Wörter erzeugt, in denen alle a's vor allen b's stehen, und alle b's vor allen c's. (Zeigt eine genaue Diskussion aller Regeln.)

Beide Behauptungen (i) und (ii) zusammen bringen dann unsere gewünschte Aussage 4.1.

Wir werden jetzt noch zwei weitere Beispiele für Grammatiken bringen, wobei wir den Nachweis für die erzeugte Sprache nicht bringen werden. Wie im obigen Beispiel führt oft eine genaue Analyse der Regeln und deren Zusammenspiel zur gewünschten Aussage.

Beispiel 4.9 Es sei

$$G = (\{S\}, \{a, b\}, \{S \to aSb,\ S \to ab\}, S)$$

eine Grammatik, dann gilt

$$L(G) = \{a^n b^n \mid n \geq 1\}.$$

Eine Ableitung für das Wort a^4b^4 sieht dann so aus:

$$S \Longrightarrow aSb \Longrightarrow aaSbb \Longrightarrow aaaSbbb \Longrightarrow aaaabbbb.$$

Beispiel 4.10 Es sei

$$G = (\{S\}, \{a\}, \{S \to aS,\ S \to a\}, S)$$

eine Grammatik, dann gilt

$$L(G) = \{a^n \mid n \geq 1\}.$$

Eine Ableitung für das Wort a^5 sieht dann so aus:

$$S \Longrightarrow aS \Longrightarrow aaS \Longrightarrow aaaS \Longrightarrow aaaaS \Longrightarrow aaaaa.$$

Beispiel 4.11 Es sei

$$G = (\{S\}, \{a, b\}, \{S \to aS,\ S \to bS,\ S \to a,\ S \to b\}, S)$$

eine Grammatik, dann gilt

$$L(G) = \{a, b\}^+ = \{w \in \{a, b\}^* \mid w \neq \varepsilon\}.$$

Eine Ableitung für das Wort *aaba* sieht dann so aus:

$$S \Longrightarrow aS \Longrightarrow aaS \Longrightarrow aabS \Longrightarrow aaba.$$

4.2.1 Chomsky-Hierarchie

Wir wollen in diesem Kapitel eine Klassifikation der Grammatiken in sogenannte *Typ*-0- bis *Typ*-3-*Grammatiken* angeben. Sie stammt von Noam Chomsky aus dem Jahre 1958, einem Linguisten aus der Frühzeit der Theorie formaler Sprachen, trotzdem hat sie nichts an Aktualität verloren, im Gegenteil.

Definition 4.12 *Eine Grammatik $G = (V, \Sigma, P, S)$ heißt vom*

- *Typ* 0 *oder* Phrasenstrukturgrammatik*, wenn sie keinen Beschränkungen unterliegt,*
- *Typ* 1 *oder* kontextabhängig*, falls für alle Regeln $\alpha \to \beta$ in P gilt: $|\alpha| \leq |\beta|$, mit der Ausnahme $S \to \varepsilon$, falls S nicht auf der rechten Seite einer Regel vorkommt.*
- *Typ* 2 *oder* kontextfrei*, wenn jede Regel von der Form $A \to \beta$ mit $A \in V$ und $\beta \in (V \cup \Sigma)^*$ ist.*
- *Typ* 3 *oder* regulär*, wenn jede Regel von der Form $A \to wB$ oder $A \to w$ mit $A, B \in V$ und $w \in \Sigma^*$ ist.*

Bevor wir die Begriffe auf Sprachen erweitern, eine Bemerkung zu den Bezeichnungen *kontextfrei* und *kontextabhängig* (auch manchmal *kontextsensitiv* genannt):

Bei einer *kontextfreien* Regel $A \to \alpha$ kann in einem Wort der Buchstabe A unabhängig vom *Kontext* des Buchstaben A (d. h. des Textes links und rechts von A) durch α ersetzt werden.

Bei *kontextabhängigen* Grammatiken kann man zeigen, dass man sich auf Regeln der Form $\gamma_1 A \gamma_2 \to \gamma_1 \alpha \gamma_2$ mit $\gamma_1, \gamma_2 \in (V \cup \Sigma)^*$ und $\alpha \in (V \cup \Sigma)^+$beschränken kann (mit Ausnahme $S \to \varepsilon$), d. h. wiederum wird letztendlich die Variable A durch ein Wort α ersetzt, allerdings können wir diese Ersetzung nur dann vornehmen, wenn A in einem gewissen *Kontext* steht (hier γ_1 und γ_2), d. h. die Ersetzung ist vom *Kontext abhängig.*

Definition 4.13 *Eine Sprache $L \subseteq \Sigma^*$ heißt vom Typ* 0 *oder rekursiv aufzählbar (Typ* 1 *oder kontextabhängig, Typ* 2 *oder kontextfrei, Typ* 3 *oder regulär), falls es eine Grammatik $G = (V, \Sigma, P, S)$ vom Typ* 0 *(Typ* 1*, Typ* 2*, Typ* 3*) gibt, so dass $L = L(G)$ gilt.*

Betrachten wir unsere Beispielgrammatiken, so gilt:

- Die Grammatik und somit auch die erzeugte Sprache aus Beispiel 4.8 ist vom Typ 1.
- Die Grammatiken und somit auch die erzeugten Sprachen aus den Beispielen 4.7 sowie 4.9 sind vom Typ 2.
- Die Grammatiken und somit auch die erzeugten Sprachen aus den Beispielen 4.10 sowie 4.11 sind vom Typ 3.

Aus den Definitionen der Chomsky-Grammatiken folgt sofort:

Folgerung 4.14 (i) *Jede Typ-*1*-Grammatik ist vom Typ* 0*.*

(ii) *Jede Typ-*2*-Grammatik ist vom Typ* 0*.*

(iii) *Jede Typ-*3*-Grammatik ist vom Typ* 2*.* □

Wegen der kanonischen Definition der Sprachen folgt aus der Folgerung 4.14 sofort:

Lemma 4.15 (i) *Jede Typ-*1*-Sprache ist vom Typ* 0*.*

(ii) *Jede Typ-*2*-Sprache ist vom Typ* 0*.*

(iii) *Jede Typ-*3*-Sprache ist vom Typ* 2*.* □

Ich bemerke, dass nicht jede Typ-2-Grammatik vom Typ 1 ist, da Typ-2-Grammatiken sogenannte ε-Regeln enthalten dürfen (auch für Variablen, die nicht das Startwort sind). Wir werden aber später zeigen, dass man jede kontextfreie Grammatik „ε-Regel-frei“ machen kann, so dass man auch zeigen kann, dass jede Typ-2-sprache auch vom Typ 1 ist.

Führen wir die Bezeichnungen **Typ 0**, **Typ 1**, **Typ 2** und **Typ 3** für die Menge aller Typ-0-Sprachen, Typ-1-Sprachen, Typ-2-Sprachen bzw. Typ-3-Sprachen ein, so werden wir letztendlich folgenden Satz beweisen, der hier an dieser Stelle schon mal wegen der Vollständigkeit genannt wird.

Satz 4.16 *Es gilt:*

$$\textbf{Typ 3} \subsetneq \textbf{Typ 2} \subsetneq \textbf{Typ 1} \subsetneq \textbf{Typ 0}.$$

Das heißt, alle Inklusionen in Lemma 4.15 sind echt. Satz 4.16 stellt eine der wichtigsten Aussagen nicht nur in dieser Vorlesung, sondern in der gesamten Theorie der Informatik dar und wird als sogenannte *Chomsky-Hierarchie* bezeichnet.

Beispiele für Sprachen, die die *Echtheit der Inklusionen* in der Chomsky-Hierarchie zeigen, sind folgende (hier ohne Beweis):

Satz 4.17 (i) *Die Sprache* $L = \{a^n b^n \mid n \geq 1\}$ *ist vom Typ* 2*, aber nicht vom Typ* 3*.*

(ii) *Die Sprache* $L' = \{a^n b^n c^n \mid n \geq 1\}$ *ist vom Typ* 1*, aber nicht vom Typ* 2*.*

(iii) *Die Sprache* $L'' = L_H$ *ist vom Typ* 0*, aber nicht vom Typ* 1*, dabei ist* L_H *das „Halteproblem“ aus dem ersten Teil der Vorlesung (siehe Definition* 2.65*).*

Folgender Satz gibt die Beziehung der Chomsky-Hierarchie zu weiteren Sprachklassen.

Satz 4.18 (i) *Die Menge der Typ-*0*-Sprachen und die Menge der semi-entscheidbaren Sprachen (siehe Definition* 2.56*) sind identisch.*

(ii) *Es gibt Sprachen, die sind nicht vom Typ* 0*.* □

Zusammengefasst stellen wir die Aussagen aus den Sätzen 4.16, 4.18 und 2.60 in der Abbildung 4.3 dar.

Von Interesse für die Informatik sind insbesondere die kontextfreien und die regulären Sprachen. Deshalb werden sie auch in unseren weiteren Überlegungen die Hauptrolle spielen. Regulären

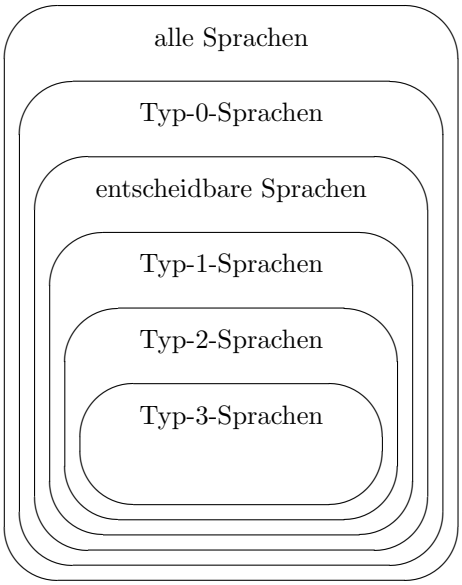

Abbildung 4.3: Die Chomsky-Hierarchie mit weiteren Sprachklassen

Sprachen spielen unter Anderem eine große Rolle bei der lexikalischen Analyse im Compilerbau, beim Suchen und Ersetzen in Editoren, bei Netzwerkprotokollen etc. Die Theorie kontextfreier Sprachen ist eng mit dem Compilerbau, insbesondere mit der Syntaxanalyse verbunden. Eine weitere Sprachklasse, die hier von besonderem Interesse ist, ist die Klasse der *deterministisch kontextfreien* Sprachen, die in der Hierarchie unterhalb der kontextfreien aber oberhalb der regulären Sprachen liegen. In diesem Zusammenhang wurden auch die $LL(k)$- und $LR(k)$-Sprachen untersucht. Obige Gründe führten zu einer weitgehenden Theorie der regulären und kontextfreien Sprachen, insbesondere auch deshalb, da diese Sprachklassen sich theoretisch „leicht“ erschließen ließen. Allerdings gilt: „Die Welt ist nicht kontextfrei“. Schon die Menge aller korrekten Programme in einer gängigen Programmiersprache (PASCAL, C++, PROLOG, EIFFEL, JAVA, etc.) ist leider nicht kontextfrei. Allerdings wurde für die Beschreibung dieser Sprachen trotzdem eine kontextfreie Syntax benutzt (zu den Gründen später). Das hat zur Folge, dass ein syntaktisch korrektes Programm noch lange nicht korrekt sein muss, sondern dass noch weitere Überprüfungen notwendig sind.

4.2.2 Wortproblem

Ein gegebenes Programm ist syntaktisch korrekt, falls es der Syntax *entspricht*, d. h. falls es aus den syntaktischen Regeln abgeleitet werden kann. Für die Syntaktische Überprüfung eines Prgrammes muss man also untersuchen, ob es aus den syntaktische Regeln aufgebaut werden kann. Wenn man bedenkt, dass die Syntax nichts Anderes ist als eine Menge von Regeln, stellt also ein Programm nichts Anderes dar als ein Wort. Syntaktisch korrekt ist das Programm (Wort), wenn es der Syntax (Regeln) entspricht. Mit anderen Worten, interessiert die Frage, ob ein gegebenes Wort von einer gegebenen Grammatik erzeugt werden kann, das aber ist genau das *Wortproblem* für Grammatiken. Genauer formuliert:

Definition 4.19 (Wortproblem) *Sei $i \in \{0,1,2,3\}$. Unter dem Wortproblem für Typ-i-Grammatiken versteht man folgendes Problem:*

Gegeben: *Grammatik $G = (V, \Sigma, P, S)$ vom Typ i, $i \in \{0,1,2,3\}$, und Wort $w \in \Sigma^*$,*

Frage: *Gilt $w \in L(G)$?*

Da das Halteproblem für Turingmaschinen unentscheidbar ist und es nach Satz 4.17 in der Menge der Typ-0-Sprachen liegt, gilt:

Folgerung 4.20 *Das Wortproblem für Typ-0-Sprachen ist unentscheidbar.* □

Das ist natürlich hinsichtlich der eingangs gemachten Bemerkungen zur Syntaxüberprüfung eine katastrophale Aussage. Glücklicherweise kann man schon für Typ-1-Grammatiken die Entscheidbarkeit retten. Es gibt also einen Algorithmus, der bei Eingabe einer Typ-1-Grammatik $G = (V, \Sigma, P, S)$ und einem Wort $w \in \Sigma^*$ in endlicher Zeit entscheidet, ob $w \in L(G)$ gilt oder nicht. Der folgende Satz hält die Aussage exakt fest. Ursache ist die Monotonie der Ableitungen, d. h. die Bedingung $|\alpha| \leq |\beta|$ für alle Regeln $\alpha \to \beta$ in P. Deshalb brauchen nämlich nur endlich viele Ableitungen untersucht werden, dem geneigten Leser ist der korrekte Beweis angefügt.

Satz 4.21 *Das Wortproblem für Typ-1-Grammatiken ist entscheidbar.*

Beweis. Sei $G = (V, \Sigma, P, S)$ die gegebene Grammatik vom Typ 1 und $w \in \Sigma^*$ das gegebene Wort. Wir definieren Mengen T_m^n für alle $m, n \in \mathbb{N}$ wie folgt.

$$T_m^n = \{w \in (V \cup \Sigma)^* \mid |w| \leq n \text{ und } w \text{ lässt sich aus } S \text{ in höchstens } m \text{ Schritten ableiten}\}.$$

Diese Mengen T_m^n, $n \geq 1$ lassen sich induktiv über m wie folgt definieren:

$$\begin{aligned} T_0^n &= \{S\}, \\ T_{m+1}^n &= \mathrm{Abl}_n(T_m^n), \end{aligned}$$

wobei

$$\mathrm{Abl}_n(X) = X \cup \{w \in (V \cup \Sigma)^* \mid |w| \leq n \text{ und } w' \Longrightarrow w \text{ für ein } w' \in X\}.$$

Diese Darstellung ist natürlich nur für Typ-1-Grammatiken anwendbar.

Da es nur endlich viele Wörter in $(V \cup \Sigma)^*$ gibt, die höchstens die Länge n haben, ist $\bigcup_{m \geq 0} T_m^n$ für jedes $n \in \mathbb{N}$ eine endliche Menge. Folglich gibt es ein m mit

$$T_m^n = T_{m+1}^n = T_{m+2}^n = \cdots$$

Falls nun w, mit $|w| = n$, in $L(G)$ liegt, so muss w in $\bigcup_{m \geq 0} T_m^n$ und damit in T_m^n für ein m liegen. Das ist aber in endlicher Zeit überprüfbar. □

Der aus dem Beweis des Satzes 4.21 resultierende Algorithmus zur Entscheidung des Wortproblems ist leider exponentiell. Bis heute ist auch kein „besserer“ Algorithmus bekannt. Es ist auch nicht zu vermuten, dass es bald einen besseren Algorithmus gibt, da das Wortproblem für Typ-1-Grammatiken $\mathbb{NP}$-*hart* (siehe Kapitel 3) ist.

Für eine praktikable Syntaxüberprüfung ist also eine Syntax, die als allgemeine Typ-1-Grammatik konstruiert wurde, nicht zu gebrauchen, denn über die katastrophalen Auswirkungen von Algorithmen mit exponentiellem Laufzeitverhalten wurde bereits auch in Kapitel 3 informiert.

Glücklicherweise kann man zeigen, dass das Wortproblem von Teilklassen der Typ-1-Sprachen auch eine kleinere Komplexität haben. Ich hoffe, in der Vorlesung noch darauf eingehen zu können. Ansonsten merken Sie sich bitte: Das Wortproblem für Typ-2-Grammatiken ist von kubischer Zeitkomplexität, das Wortproblem für $LL(k)$- und $LR(k)$-Grammatiken ist von linearer Zeitkomplexität. Glücklicherweise kann man die Syntax von gängigen Programmiersprachen schon durch $LL(k)$- und $LR(k)$-Grammatiken realisieren. Das wird bei modernen Programmiersprachen verwendet, so dass also ein *heutiger* Compiler die Syntaxüberprüfung in Linearzeit erledigt.

Betrachten wir zum besseren Verständnis ein Beispiel zur Anwendung des Algorithmus, basierend auf den Beweis zum Satz 4.21.

Beispiel 4.22 Gegeben sei die Grammatik aus Beispiel 4.8. Sei $n = 4$. Dann erhalten wir:

$T_0^4 = \{S\}$,
$T_1^4 = \{S, aSBC, aBC\}$,
$T_2^4 = \{S, aSBC, aBC, abC\}$,
$T_3^4 = \{S, aSBC, aBC, abC, abc\}$,
$T_4^4 = \{S, aSBC, aBC, abC, abc\} = T_3^4$.

Das heißt, das einzige terminale Wort der Sprache $L(G)$ der Länge ≤ 4 ist abc.

4.2.3 Syntaxbäume

In der Einleitung zu diesem Kapitel haben wir Syntaxbäume schon informell kennengelernt. Wir wollen auch hier nicht allzuviel hinzufügen, sondern nur einige wichtige Eigenschaften zusammenfassend ohne nähere Betrachtung nennen. Den interessierten Leser verweisen wir auf die reichhaltige Literatur.

1. Jeder Ableitung eines Wortes w in einer Typ-2- oder Typ-3-Grammatik kann ein Syntaxbaum zugeordnet werden.
2. Sei $w \in L(G)$ und $S = w_1 \Longrightarrow w_2 \Longrightarrow \cdots \Longrightarrow w_n = w$ eine Ableitung für w. Dann wird der Syntaxbaum folgendermaßen definiert: Die Wurzel wird S; falls bei der Ableitung eine Regel $A \to \alpha$ angewendet wird, heißt das, dem Vaterknoten A werden $|\alpha|$ viele Söhne zugeordnet, nämlich alle Symbole von α. Die Blätter des Syntaxbaumes sind dann genau (von links nach rechts gelesen) die Buchstaben von w.
3. Falls die Grammatik regulär ist, ist jeder Syntaxbaum „entartet“ (Kette).
4. Verschiedene Ableitungen für ein und dasselbe Wort können den gleichen Syntaxbaum haben, oder auch nicht.
5. Mehrdeutig heißt eine Grammatik, wenn es ein Wort gibt, für das verschiedene Syntaxbäume existieren.
6. Inhärent mehrdeutig heißt eine Sprache, wenn jede Grammatik G mit $L(G) = L$ mehrdeutig ist.
7. Die Sprache $L = \{a^i b^j c^k \mid i = j \text{ oder } j = k\}$ ist ein Beispiel für eine inhärent mehrdeutige, kontextfreie Sprache.
8. Es ist nicht entscheidbar, ob zu einer gegebenen kontextfreien Grammatik eine äquivalente kontextfreie Grammatik existiert, die nicht mehrdeutig ist.