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.

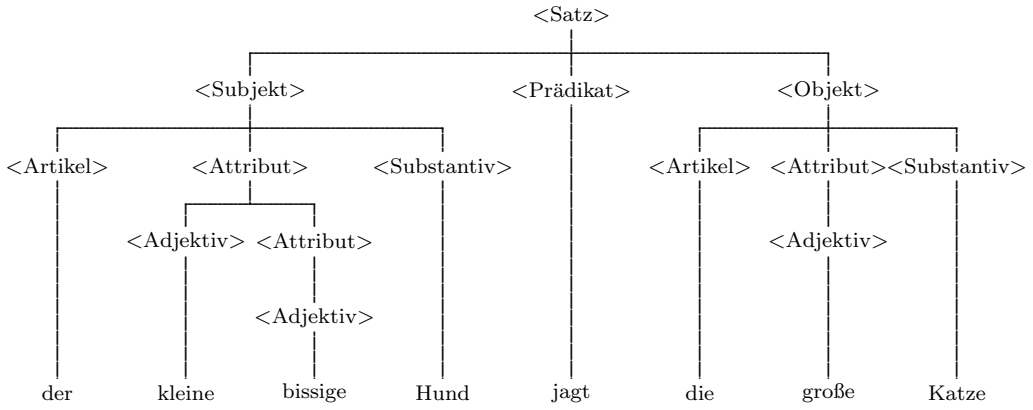

Abbildung 4.1: Syntaxbaum für den Satz „*der kleine bissige Hund jagt die große Katze*“

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).

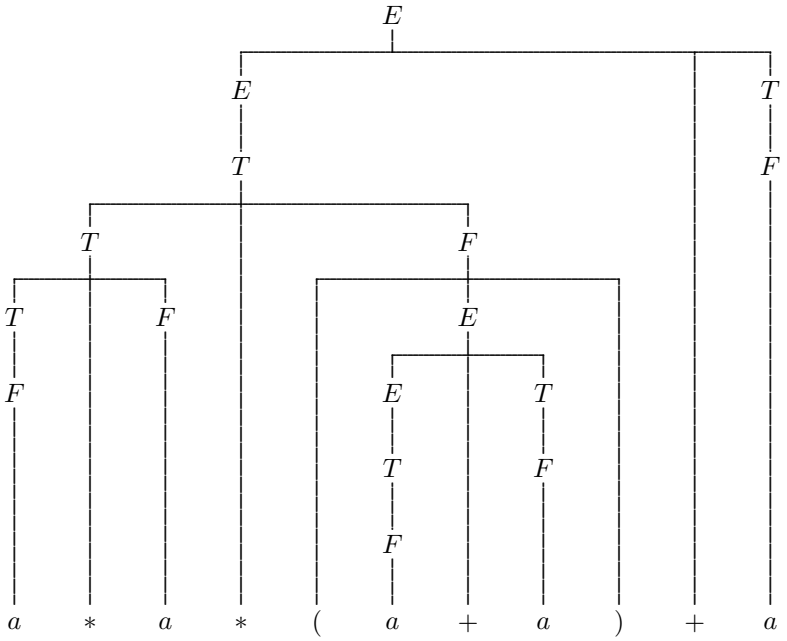

Abbildung 4.2: Ein Syntaxbaum für das Wort $a * a * (a + a) + a$

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* 2.67*).*

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 2.62 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

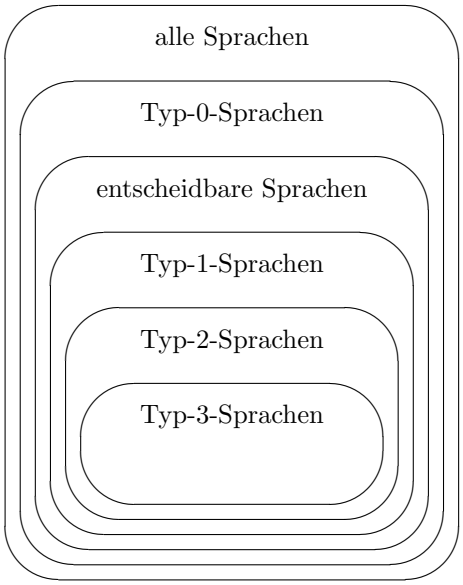

Abbildung 4.3: Die Chomsky-Hierarchie mit weiteren Sprachklassen

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:

$$\begin{aligned} T_0^n &= \{S\}, \\ T_{m+1}^n &= T_m^n \cup \{w \in (V \cup \Sigma)^* \mid |w| \leq n \text{ und } w' \Longrightarrow w \text{ für ein } w' \in T_m^n\}. \end{aligned}$$

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 die Wortprobleme von Teilklassen der Typ-1-Sprachen auch eine kleinere Komplexität haben. 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 dem Beweis zum Satz 4.21.

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

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

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.

4.3 Reguläre Sprachen

In diesem Kapitel beschäftigen wir uns etwas näher mit den regulären Sprachen, insbesondere mit der Möglichkeit verschiedener Charakterisierungen und den Eigenschaften.

4.3.1 Endliche Automaten

Zur Definition der regulären Sprachen benutzten wir reguläre Grammatiken. Grammatiken können Wörter über einem gewissen Alphabet *erzeugen* und so eine Menge von Wörtern, eine *Sprache*, beschreiben. Jetzt wollen wir einen anderen Mechanismus zur Beschreibung benutzen: die *endlichen Automaten*. Der Automat erhält ein Wort als Eingabe, „arbeitet“ über diesem Wort und *erkennt* (oder *akzeptiert*) es oder auch nicht. Alle Wörter, die ein Automat akzeptiert, bilden die von ihm beschriebene *Sprache*.

Definition 4.23 (Deterministischer endlicher Automat) *Ein deterministischer endlicher Automat (Wir wollen ihn kurz mit DEA bezeichnen) A ist ein 5-Tupel $A = (Z, \Sigma, \delta, z_0, E)$, wobei Z das Zustandsalphabet und Σ das Eingabealphabet mit $Z \cap \Sigma = \emptyset$ sind. $z_0 \in Z$ ist der Anfangszustand, $E \subseteq Z$ die Menge der Endzustände oder akzeptierenden Zustände. Mit δ bezeichnen wir die Zustandsüberführungsfunktion $\delta\colon Z \times \Sigma \to Z$.*

Wir interpretieren einen DEA als endliche Kontrolle, die sich in einem Zustand $z \in Z$ befindet und eine Folge von Symbolen aus Σ, die auf einem Band geschrieben stehen, liest (siehe Abbildung 4.4). In einem Schritt bewegt sich der Lesekopf des Automaten, der augenblicklich über der Zelle des

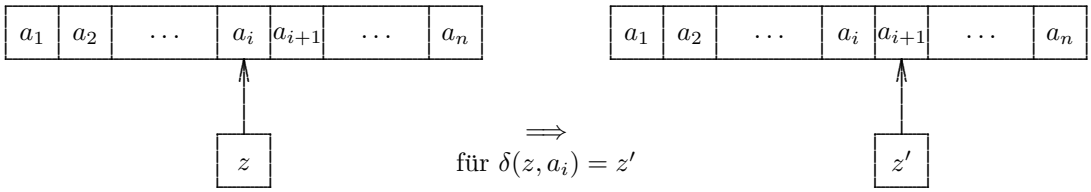

Abbildung 4.4: Interpretation der Arbeitsweise eines endlichen Automaten

Bandes mit dem Inhalt $a_i \in \Sigma$ steht und sich im Zustand $z \in Z$ befindet, einen Schritt nach rechts und geht in den Zustand $\delta(z, a_i) = z'$ über. Ist $\delta(z, a_i) = z' \in E$, d. h. ein akzeptierender Zustand, so hat der DEA das ganze Wort, das er, beginnend in Startzustand z_0, gelesen hat, akzeptiert.

Um die von einem DEA akzeptierte Sprache, d. h. die Menge aller von ihm akzeptierten Wörter, formal beschreiben zu können, erweitern wir die Zustandsfunktion δ zur Funktion $\hat{\delta}\colon Z \times \Sigma^* \to Z$ rekursiv durch folgende Definition.

Definition 4.24 (Erweiterte Zustandsüberführungsfunktion eines DEA) *Sei A ein deterministischer endlicher Automat $A = (Z, \Sigma, \delta, z_0, E)$, die erweiterte Zustandsüberführungsfunktion $\hat{\delta}\colon Z \times \Sigma^* \to Z$ wird definiert durch*

(i) $\hat{\delta}(z, \varepsilon) = z$ *für alle $z \in Z$,*

(ii) $\hat{\delta}(z, wa) = \delta(\hat{\delta}(z, w), a)$ *für alle $z \in Z$, $a \in \Sigma$, $w \in \Sigma^*$.*

Ich erwähne, dass in der Literatur $\hat{\delta}$ oft auch nur als δ bezeichnet wird. Das ist in der Tatsache begründet, dass für alle $z \in Z$ und für alle $a \in \Sigma$ die Gleichheit $\hat{\delta}(z, a) = \delta(z, a)$ gilt, d. h. δ ist eine Einschränkung der Funktion $\hat{\delta}$ auf $Z \times \Sigma$.

Jetzt können wir die von dem Automaten akzeptierte Sprache definieren:

Definition 4.25 (Akzeptierte Sprache eines DEA) *Für einen DEA $A = (Z, \Sigma, \delta, z_0, E)$ sei die von ihm akzeptierte Sprache $T(A)$ definiert durch $T(A) = \{w \in \Sigma^* \mid \hat{\delta}(z_0, w) \in E\}$.*

Nun ist es Zeit für ein Beispiel.

Beispiel 4.26 Wir betrachten den deterministischen endlichen Automaten

$$A = (\{z_0, z_1, z_2, z_3\}, \{a, b\}, \delta, z_0, \{z_3\}),$$

wobei δ durch die Tabelle

δ	z_0	z_1	z_2	z_3
a	z_1	z_1	z_3	z_3
b	z_0	z_2	z_0	z_3

gegeben ist. Um zu entscheiden, ob zum Beispiel die Wörter ab, $aaba$ oder $bababb$ von dem Automaten akzeptiert werden, müssen wir $\hat{\delta}(z_0, ab)$, $\hat{\delta}(z_0, aaba)$ und $\hat{\delta}(z_0, bababb)$ bestimmen. Dazu benutzen wir die Definition von $\hat{\delta}$ und natürlich die Definition des Automaten, insbesondere der Überführungsfunktion δ.

$$\begin{aligned}
\hat{\delta}(z_0, ab) &= \delta(\hat{\delta}(z_0, a), b) \\
&= \delta(\delta(\hat{\delta}(z_0, \varepsilon), a), b) \\
&= \delta(\delta(z_0, a), b) \\
&= \delta(z_1, b) \\
&= z_2
\end{aligned}$$

Also gilt $\hat{\delta}(z_0, ab) = z_2$ und somit $ab \notin T(A)$, da $z_2 \notin E$ gilt, z_2 also kein akzeptierender Zustand ist.

$$\begin{aligned}
\hat{\delta}(z_0, aaba) &= \delta(\hat{\delta}(z_0, aab), a) \\
&= \delta(\delta(\hat{\delta}(z_0, aa), b), a) \\
&= \delta(\delta(\delta(\hat{\delta}(z_0, a), a), b), a) \\
&= \delta(\delta(\delta(\delta(\hat{\delta}(z_0, \varepsilon), a), a), b), a) \\
&= \delta(\delta(\delta(\delta(z_0, a), a), b), a) \\
&= \delta(\delta(\delta(z_1, a), b), a) \\
&= \delta(\delta(z_1, b), a) \\
&= \delta(z_2, a) \\
&= z_3
\end{aligned}$$

Wegen $z_3 \in E$ gilt demnach: $aaba$ wird vom Automaten akzeptiert.

$$\begin{aligned}
\hat{\delta}(z_0, bababb) &= \delta(\hat{\delta}(z_0, aab), a) \\
&= \delta(\delta(\hat{\delta}(z_0, baba), b), b) \\
&= \delta(\delta(\delta(\hat{\delta}(z_0, bab), a), b), b) \\
&= \delta(\delta(\delta(\delta(\hat{\delta}(z_0, ba), b), a), b), b) \\
&= \delta(\delta(\delta(\delta(\delta(\hat{\delta}(z_0, b), a), b), a), b), b) \\
&= \delta(\delta(\delta(\delta(\delta(\delta(\hat{\delta}(z_0, \varepsilon), b), a), b), a), b), b) \\
&= \delta(\delta(\delta(\delta(\delta(\delta(z_0, b), a), b), a), b), b) \\
&= \delta(\delta(\delta(\delta(\delta(z_0, a), b), a), b), b) \\
&= \delta(\delta(\delta(\delta(z_1, b), a), b), b) \\
&= \delta(\delta(\delta(z_2, a), b), b) \\
&= \delta(\delta(z_3, b), b) \\
&= \delta(z_3, b) \\
&= z_3
\end{aligned}$$

Wegen $z_3 \in E$ gilt auch $bababb \in T(A)$.

Wir haben also $ab \notin T(A)$ und $aaba$, $bababb \in T(A)$. Nun gilt es zu bestimmen, welche Sprache $T(A)$ von diesem deterministischen endlichen Automaten A akzeptiert wird. Eine genaue Analyse der Überführungsfunktion δ würde ergeben (den exakten Beweis führen wir nicht, wird auch vom Hörer dieser Vorlesung nicht verlangt)

$$T(A) = \{w \in \{a,b\}^* \mid w = uabav \text{ für Wörter } u, v \in \{a,b\}^*\},$$

also akzeptiert der Automat alle Wörter über dem Alphabet $\{a,b\}$, die das Teilwort aba besitzen.

Im obigen Beispiel ist es nicht so leicht, die akzeptierte Sprache des Automaten zu bestimmen. Oft wird es etwas leichter, falls wir die Darstellung eines gerichteten Graphen, des sogenannten *Überführungsgraphen* oder auch *Transitionsgraphen* des deterministischen endlichen Automaten benutzen. Diese Darstellung sieht folgendermaßen aus.

Die Menge der Zustände Z wird die Knotenmenge des Graphen, $\delta(z_1, a) = z_2$ wird durch eine gerichtete Kante von z_1 zu z_2, die mit a markiert ist, dargestellt. Der Anfangszustand wird durch einen zum Knoten gehenden Pfeil dargestellt, Endzustände durch zwei konzentrische Kreise (siehe Abbildung 4.5). Wollen wir jetzt wissen, ob ein Wort vom Automaten akzeptiert wird, dann

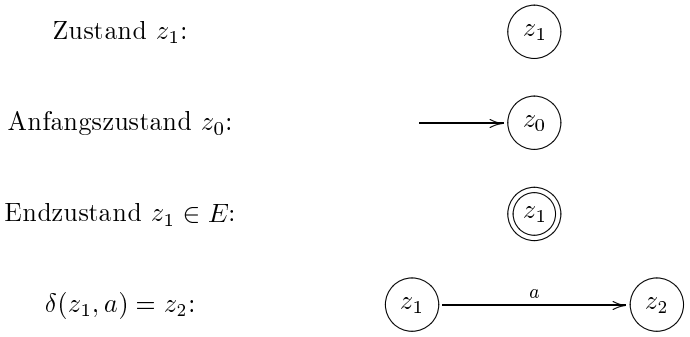

Abbildung 4.5: Konstruktion des Überführungsgraphen eines DEA

müssen wir, beginnend im Zustand z_0 die Kanten des Graphen entsprechend des Wortes entlang wandern.

In Abbildung 4.6 ist der Überführungsgraph zum DEA aus Beispiel 4.26 dargestellt. Wenn man

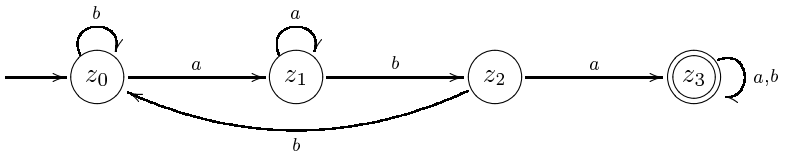

Abbildung 4.6: Überführungsgraph des DEA aus Beispiel 4.26

sich diesen Graphen genauer ansieht, erkennt man eher als aus der Tabelle, dass die akzeptierte Sprache genau die Menge aller Wörter mit dem Teilwort aba ist. Die Zustände sind assoziert mit den Teilen des Wortes aba, die bereits eingelesen wurden: z_0: noch nichts von aba, z_1: schon das a von aba, z_2: schon das ab von aba, z_3: das gesamte Teilwort aba. Wenn aba gelesen wurde (Zustand z_3), bleibt der Automat immer in diesem akzeptierenden Zustand.

Betrachten wir ein weiteres Beispiel.

Beispiel 4.27 Es sei A der deterministische endliche Automat

$$A = (\{z_0, z_1, z_2, z_3\}, \{0,1\}, \delta, z_0, \{z_2, z_3\}),$$

mit der Überführungsfunktion δ, gegeben durch folgende Tabelle.

δ	z_0	z_1	z_2	z_3
0	z_0	z_3	z_3	z_0
1	z_1	z_2	z_2	z_1

Der dazugehörige Graph ist in Abbildung 4.7 dargestellt. Auch hier nutzen wir wiederum die

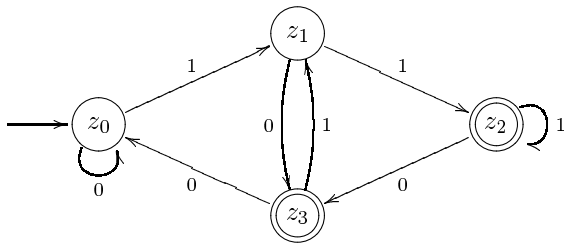

Abbildung 4.7: Überführungsgraph des DEA aus Beispiel 4.27

Zustände, um Informationen über den Teil des Wortes zu speichern, der bereits eingelesen wurde. Man erkennt, dass der Zustand die beiden zuletzt gelesenen Buchstaben repräsentiert. Der Zustand z_0 bedeutet, die letzten beiden Buchstaben waren 00, bei z_1: 01, bei z_2: 11 und bei z_3: 10. Akzeptiert wird im Zustand z_2 und z_3, also genau dann, wenn das vorletzte Zeichen eine 1 war. Also gilt

$$T(A) = \{w \in \{0,1\}^* \mid w = u1x \text{ mit } u \in \{0,1\}^*,\ x \in \{0,1\}\}$$

für die akzeptierte Sprache $T(A)$, also ist $T(A)$ die Menge aller Wörter über $\{0,1\}$, deren vorletztes Symbol eine 1 ist.

Es stellt sich natürlich die Frage, welche Sprachklasse durch deterministische endliche Automaten beschrieben wird. Die akzeptierten Sprachen in den Beispielen 4.26 und 4.27 sind regulär. Gilt das für alle von DEA akzeptierten Sprachen? Die Antwort gibt folgender Satz.

Satz 4.28 *Sei A ein deterministischer endlicher Automat. Dann ist die von A akzeptierte Sprache $T(A)$ regulär (vom Typ 3).*

Beweis. Sei $A = (Z, \Sigma, \delta, z_0, E)$ ein DEA. Wir konstruieren eine Grammatik $G = (V, \Sigma, P, S)$ folgendendermaßen: wir setzen $V = Z$ und $S = z_0$. Weiterhin sei

$$P = \{z_1 \to az_2 \mid \delta(z_1, a) = z_2\} \cup \{z_1 \to \varepsilon \mid z_1 \in E\}.$$

Offensichtlich ist die so konstruierte Grammatik G vom Typ 3, also regulär, und erzeugt die Sprache $T(A)$, was noch zu beweisen wäre. Wir verweisen aber an dieser Stelle für den interessierten Leser auf die Literatur. □

Wir wollen die Konstruktion aus obigem Beweis an einem Beispiel demonstrieren.

Beispiel 4.29 Sei $A = (\{z_0, z_1, z_2, z_3\}, \{a, b\}, \delta, z_0, \{z_3\})$ der DEA aus Beispiel 4.26 mit der Überführungsfunktion δ:

δ	z_0	z_1	z_2	z_3
a	z_1	z_1	z_3	z_3
b	z_0	z_2	z_0	z_3

Wir konstruieren jetzt die äquivalente Typ-3-Grammatik $G = (V, \Sigma, P, S)$:

$$
\begin{aligned}
V &= \{z_0, z_1, z_2, z_3\},\\
\Sigma &= \{a, b\},\\
S &= z_0,\\
P &= \{z_0 \to az_1,\ z_0 \to bz_0,\ z_1 \to az_1,\ z_1 \to bz_2,\ z_2 \to az_3,\ z_2 \to bz_0,\ z_3 \to az_3,\ z_3 \to bz_3\}\\
&\quad \cup \{z_3 \to \varepsilon\}.
\end{aligned}
$$

Bei der Konstruktion von P handelt sich in der ersten Zeile um die Regeln, die ausgehend von δ im ersten Schritt konstruiert werden, also zum Beispiel $z_0 \to az_1$ wegen $\delta(z_0, a) = z_1$ oder $z_0 \to bz_0$ wegen $\delta(z_0, b) = z_0$. In der zweiten Zeile kommt die terminierende Regel für den Endzustand $\{z_3\}$ hinzu.

4.3.2 Nichtdeterministische endliche Automaten

Nachdem wir wissen, dass jede von einem deterministischen endlichen Automaten akzeptierte Sprache vom Typ 3, also regulär ist, interessiert natürlich die Umkehrung, also ob jede Typ-3-Sprache auch von einem deterministischen endlichen Automaten akzeptiert werden kann. Eine naheliegende Idee wäre, die Konstruktion aus dem obigen Beweis einfach entsprechend zu invertieren.

Das führt allerdings zu zwei Schwierigkeiten.

1. Wenn wir zum Beispiel eine reguläre Grammatik mit den Regeln $A \to aB$ und $A \to aC$ hätten, dann müssten gleichzeitig $\delta(A, a) = B$ und $\delta(A, a) = C$ gelten, was nicht möglich ist.
2. Wenn wir zum Beispiel eine reguläre Grammatik mit einer Regel $A \to aabB$ hätten, müssten wir eine Überführung vom Zustand A zum Zustand B mit aab erzeugen, was nicht möglich ist.

Um die Idee der Invertierung des Beweises von Satz 4.28 jedoch nicht fallen zu lassen und zu zeigen, dass reguläre Sprachen von DEA akzeptiert werden können, führen wir einfach neue Modelle ein, um die Schwierigkeiten 1 und 2 zu überwinden.

Um die Schwierigkeit 2 zu beseitigen, betrachten wir eine Normalform von regulären Grammatiken:

Satz 4.30 *Zu jeder regulären (Typ-3) Grammatik $G = (V, \Sigma, P, S)$ gibt es eine äquivalente Grammatik $G' = (V', \Sigma, P', S')$, die nur Regeln der Form $A \to aB$ oder $A \to a$ mit $A, B \in V'$ und $a \in \Sigma$ hat, mit der Ausnahme $S' \to \varepsilon$, falls S' nicht auf der rechten Seite einer Regel vorkommt.*

Beweis. Der Beweis würde in zwei Schritten ablaufen: zuerst müssen wir die Regeln $A \to \varepsilon$ für $A \neq S$ der Grammatik G beseitigen. Das wollen wir hier nicht ausführen, sondern auf den entsprechenden Satz für kontextfreie Sprachen verweisen, dessen Beweis hier vollständig übernommen werden kann.

Zweitens müssen wir dann die Regeln $A \to wB$ oder $A \to w$ mit $|w| \geq 2$ ersetzen. Sei also $A \to a_1a_2 \dots a_nB$ eine solche Regel mit $A, B \in V$ und $n \geq 2$, dann nehmen wir neue Nichtterminale $A_1, A_2, \dots, A_{n-1}$ in die Menge V' auf, die noch nicht verwendet wurden und ersetzen die Regel $A \to a_1a_2 \dots a_nB$ durch die Regeln

$$A \to a_1A_1,\ A_1 \to a_2A_2\ , \dots,\ A_{n-1} \to a_nB$$

in P'. Entsprechend ersetzt man eine Regel $A \to a_1a_2 \dots a_n$ mit $A \in V$ und $n \geq 2$ durch die Regeln

$$A \to a_1B_1,\ B_1 \to a_2B_2\ , \dots,\ B_{n-1} \to a_n$$

mit den neuen Nichtterminalen $B_1, B_2, \dots, B_{n-1}$ in V'.

Man kann dann leicht zeigen, dass dann die Regeln $A \to a_1 a_2 \dots a_n B$ und $A \to a_1 a_2 \dots a_n$ durch die Ableitungen

$$A \Longrightarrow a_1 A_1 \Longrightarrow a_1 a_2 A_2 \Longrightarrow \cdots \Longrightarrow a_1 a_2 a_3 \dots a_{n-1} A_{n-1} \Longrightarrow a_1 a_2 a_3 \dots a_{n-1} a_n B$$

bzw.

$$A \Longrightarrow a_1 B_1 \Longrightarrow a_1 a_2 B_2 \Longrightarrow \cdots \Longrightarrow a_1 a_2 a_3 \dots a_{n-1} B_{n-1} \Longrightarrow a_1 a_2 a_3 \dots a_{n-1} a_n$$

simuliert werden und andererseits aber keine neuen Wörter durch die Grammatik G' erzeugt werden können, also $L(G) = L(G')$ gilt. □

Zur Umgehung der oben genannten Schwierigkeit 1 führen wir ein neues Automatenmodell ein, indem wie den Begriff des DEA erweitern und solche Überführungen $\delta(A, a) = B$ und $\delta(A, a) = C$ gleichzeitig zulassen, indem wir *Nichtdeterminismus* benutzen.

Definition 4.31 (Nichtdeterministischer endlicher Automat) *Ein nichtdeterministischer endlicher Automat (kurz mit NEA bezeichnet) A ist ein 5-Tupel $A = (Z, \Sigma, \delta, z_0, E)$, wobei Z das Zustandsalphabet und Σ das Eingabealphabet mit $Z \cap \Sigma = \emptyset$ sind. $z_0 \in Z$ ist der Anfangszustand, $E \subseteq Z$ die Menge der Endzustände. Mit δ bezeichnen wir die Zustandsüberführungsfunktion $\delta \colon Z \times \Sigma \to 2^Z$.*

Wie man an der Definition erkennt, ist der NEA gar nicht so weit vom DEA entfernt. Der einzige Unterschied liegt in der Definition der Überführungsfunktion δ. Funktionswerte von δ sind nicht einzelne Zustände (wie beim DEA) sondern *Mengen von Zuständen*, d. h. $\delta(z, a) = \{z_1, z_2, \dots, z_r\}$ mit $\{z_1, z_2, \dots, z_r\} \subseteq Z$. Wir bemerken, dass $\delta(z, a)$ auch die leere Menge sein kann.

Auch den NEA können wir wiederum in derselben Art und Weise wie beim DEA als Graph darstellen, wobei von einem Zustand für ein und dasselbe Symbol mehrere Pfeile ausgehen können (oder auch keiner), siehe Abbildung 4.8.

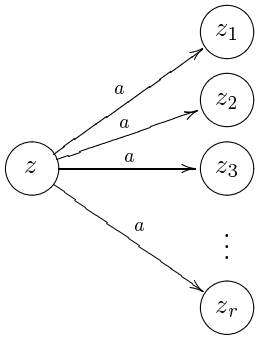

Abbildung 4.8: Nichtdeterminismus beim endlichen Automaten

Die Interpretation der Arbeitsweise des NEA ist analog der des DEA, wobei wir $\delta(z, a) = \{z_1, z_2, \dots, z_r\}$ so interpretieren, dass der Automat, wenn er sich im Zustand z befindet und ein a einliest, in einen der Zustände $z_1, z_2, \dots, z_r$ übergehen kann. Das heisst, der NEA kann ein und dasselbe Wort über *verschiedene* Wege einlesen, wobei alle Wege gleichwertig sein sollen. Das hat natürlich zur Folge, dass der NEA beim Einlesen ein und desselben Wortes einmal einen akzeptierenden Zustand erreichen kann und einmal nicht. Wir werden sagen, dass der NEA genau dann *ein Wort akzeptiert*, wenn er beim Einlesen des Wortes einen akzeptierenden Zustand erreichen *kann*, also *wenn es für das Wort im Graphen einen Pfad vom Anfangszustand zu einem Endzustand gibt.* Man kann diesen Nichtdeterminismus in gewisser Weise als Parallelverarbeitung auffassen.

Um die von einem NEA akzeptierte Sprache formal definieren zu können, benötigen wir wieder die erweiterte Zustandsfunktion $\hat{\delta} \colon Z \times \Sigma^* \to 2^Z$. Im Prinzip wird sie wieder wie beim NEA definiert, allerdings ist ja jetzt $\hat{\delta}(z, w)$ eine Menge von Zuständen und kann nicht direkt als Argument verwendet werden.

Definition 4.32 (Erweiterte Zustandsüberführungsfunktion eines NEA) *Sei A ein NEA mit $A = (Z, \Sigma, \delta, z_0, E)$, die erweiterte Zustandsfunktion $\hat{\delta}\colon Z \times \Sigma^* \to 2^Z$ wird definiert durch*

(i) $\hat{\delta}(z, \varepsilon) = \{z\}$ *für alle $z \in Z$,*

(ii) $\hat{\delta}(z, wa) = \bigcup_{z' \in \hat{\delta}(z,w)} \delta(z', a)$ *das heißt*
$= \{z'' \in Z \mid \exists z' \in Z$ *mit* $z' \in \hat{\delta}(z, w)$ *und* $z'' \in \delta(z', a)\}$ *für $z \in Z$, $a \in \Sigma$, $w \in \Sigma^*$.*

Definition 4.33 (Akzeptierte Sprache eines NEA) *Für einen NEA $A = (Z, \Sigma, \delta, z_0, E)$ sei die von ihm akzeptierte Sprache $T(A)$ definiert durch $T(A) = \{w \in \Sigma^* \mid \hat{\delta}(z_0, w) \cap E \neq \emptyset\}$.*

Bemerkung 4.34 Bitte beachten Sie, dass die Definitionen des NEA sowie der erweiterten Zustandsfunktion von den Definitionen bei Schöning in [13] etwas abweichen. Man kann aber sehr leicht zeigen, dass die Klasse der akzeptierbaren Mengen gleich sind.

Sehen wir uns ein Beispiel an, um den Mechanismus des Nichtdeterminismus besser zu verstehen.

Beispiel 4.35 Sei $A = (\{z_0, z_1, z_2\}, \{0, 1\}, \delta, z_0, \{z_2\})$, wobei δ durch die Tabelle

δ	z_0	z_1	z_2
0	$\{z_0\}$	$\{z_2\}$	$\emptyset$
1	$\{z_0, z_1\}$	$\{z_2\}$	$\emptyset$

gegeben ist, ein nichtdeterministischer endlicher Automat. Der dazugehörige Graph ist in der Abbildung 4.9 dargestellt. Betrachten wir nun die Eingabe 111 und fragen nach $\hat{\delta}(z_0, 111)$, also

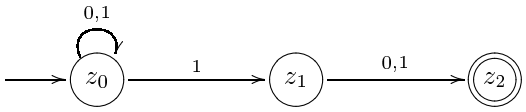

Abbildung 4.9: Überführungsgraph des NEA aus Beispiel 4.35

nach den Zuständen, die bei der Eingabe von 111 durch den NEA erreicht werden können. Beim Einlesen des ersten Symbols (eine 1) kann der Automat wählen zwischen dem Folgezustand z_0 oder z_1. Im letzteren Fall liest er dann die zweite 1 ein und landet (keine Wahlmöglichkeit) im Zustand z_2, von dem aus er die dritte 1 nicht mehr einlesen kann (es gibt keine Überführung mehr), also erreicht er über diesen Weg *keinen* Zustand. Im ersteren Fall jedoch kann er bei der Eingabe der zweiten 1 wiederum wählen zwischen den Folgezuständen z_0 und z_1: Wählt er z_1, landet er mit der letzten 1 in z_2, wählt er jedoch z_0, so kann er bei der letzten 1 wiederum zwischen den Folgezuständen z_0 und z_1 wählen. Summa summarum kann der Automat beim Einlesen von 111 die Zustände z_0, z_1, z_2 erreichen, also

$$\hat{\delta}(z_0, 111) = \{z_0, z_1, z_2\}.$$

Nun gilt

$$\hat{\delta}(z_0, 111) \cap E = \{z_0, z_1, z_2\} \cap \{z_2\} = \{z_2\} \neq \emptyset,$$

das heißt, für die Eingabe 111 gibt es einen Weg vom Anfangszustand in einen akzeptierenden Zustand, also gilt $111 \in T(A)$, d. h. die Eingabe 111 wird akzeptiert, gehört also zur akzeptierten Sprache.

Für weitere Eingaben gilt:

$$\begin{aligned}\hat{\delta}(z_0, 0) &= \{z_0\},\\ \hat{\delta}(z_0, 1) &= \{z_0, z_1\},\\ \hat{\delta}(z_0, 01) &= \{z_1\},\\ \hat{\delta}(z_0, 11) &= \{z_0, z_1, z_2\},\\ \hat{\delta}(z_0, 001) &= \{z_0, z_1\},\\ \hat{\delta}(z_0, 011) &= \{z_0, z_1, z_2\},\end{aligned}$$

also werden 11 sowie 011 akzeptiert und 0, 1, 01 sowie 001 nicht akzeptiert. Für die akzeptierte Sprache $T(A)$ gilt:

$$T(A) = \{w \in \{0,1\}^* \mid w = u1x \text{ mit } u \in \{0,1\}^*,\ x \in \{0,1\}\},$$

also ist $T(A)$ die Menge aller Wörter über $\{0,1\}$, deren vorletztes Symbol eine 1 ist.

Man kann jeden DEA natürlich als NEA auffassen, nämlich für den dann $\delta(z,a)$ immer eine Einermenge ist. Also:

Folgerung 4.36 *Jede von einem deterministischen endlichen Automaten akzeptierbare Sprache ist auch von einem nichtdeterministischen endlichen Automaten akzeptierbar.* □

Die Menge, die vom NEA im Beispiel 4.35 akzeptiert wurde, kann auch von einem DEA akzeptiert werden (siehe Beispiel 4.27). Natürlich ergibt sich sofort die Frage, ob jede von einem NEA akzeptierte Sprache auch von einem DEA akzeptiert werden kann. Die Antwort liefert der folgende Satz.

Satz 4.37 *Jede von einem nichtdeterministischen endlichen Automaten akzeptierbare Sprache ist auch von einem deterministischen endlichen Automaten akzeptierbar.*

Beweis. Sei $A = (Z, \Sigma, \delta, z_0, E)$ ein NEA. Wir konstruieren einen DEA $A' = (Z', \Sigma, \delta', z_0', E')$ durch:

$$\begin{aligned}Z' &= 2^Z,\\ z_0' &= \{z_0\},\\ E' &= \{z' \in Z' \mid z' \cap E \neq \emptyset\},\end{aligned}$$

sowie

$$\delta'(z', a) = \bigcup_{z \in z'} \delta(z, a)$$

für alle $z' \in Z'$ und $a \in \Sigma$. Dann gilt

$$T(A) = T(A'),$$

was wir an dieser Stelle nicht beweisen werden. □

Im obigen Beweis ist die Zustandsmenge des konstruierten DEA genau die Potenzmenge der Zustandsmenge des gegebenen NEA. Falls man zu einem konkret gegebenen DEA den äquivalenten NEA konstruiert, stellt man fest, dass oft nicht alle Teilmengen von Z auch wirklich erreicht werden können. Folglich reicht es aus, wenn wir, beginnend mit der Menge $\{z_0\}$ jeweils für alle $z' \in Z'$ die Teilmengen $\delta'(z', x)$ für alle $x \in \Sigma$ berechnen und die neu erzeugten Teilmengen in die Menge der Zustände Z' aufnehmen, falls sie noch nicht enthalten sind. Kommt kein neuer Zustand mehr hinzu, wären wir fertig mit der Konstruktion von δ'. Wir wollen dieses Vorgehen an einem Beispiel demonstrieren.

Beispiel 4.38 Wir nehmen den NEA

$$A = (\{z_0, z_1, z_2\}, \{0, 1\}, \delta, z_0, \{z_2\})$$

aus Beispiel 4.35 mit der in folgender Tabelle gegebenen Überführungsfunktion δ.

δ	z_0	z_1	z_2
0	$\{z_0\}$	$\{z_2\}$	$\emptyset$
1	$\{z_0, z_1\}$	$\{z_2\}$	$\emptyset$

Wir konstruieren jetzt den äquivalenten DEA

$$A' = (Z', \{0, 1\}, \delta', z', E')$$

gemäß Beweis des Satzes 4.37. Zuerst gilt $z'_0 = \{z_0\}$. Die weiteren Zustände sowie die Überführungsfunktion berechnen wir entsprechend der Definition in folgender Tabelle (spaltenweise).

δ'	$\{z_0\}$	$\{z_0, z_1\}$	$\{z_0, z_2\}$	$\{z_0, z_1, z_2\}$
0	$\{z_0\}$	$\{z_0, z_2\}$	$\{z_0\}$	$\{z_0, z_2\}$
1	$\{z_0, z_1\}$	$\{z_0, z_1, z_2\}$	$\{z_0, z_1\}$	$\{z_0, z_1, z_2\}$

Also gilt

$$Z' = \{\{z_0\}, \{z_0, z_1\}, \{z_0, z_2\}, \{z_0, z_1, z_2\}\}$$

und da nur die Zustände $\{z_0, z_2\}$ und $\{z_0, z_1, z_2\}$ einen Zustand aus E enthalten, sind sie die einzigen neuen Endzustände, also

$$E' = \{\{z_0, z_2\}, \{z_0, z_1, z_2\}\}.$$

Damit wäre der äquivalente DEA A' zum NEA A konstruiert. Zur besseren Lesbarkeit bezeichnen wir die Zustände um: $\{z_0\} =: q_0$, $\{z_0, z_1\} =: q_1$, $\{z_0, z_2\} =: q_2$ und $\{z_0, z_1, z_2\} =: q_3$. Dann gilt

$$A' = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta', q_0, \{q_2, q_3\})$$

mit

δ'	q_0	q_1	q_2	q_3
0	q_0	q_2	q_0	q_2
1	q_1	q_3	q_1	q_3

In Abbildung 4.10 finden Sie den Graphen zum Automaten A'. Man erkennt, dass die Automaten in

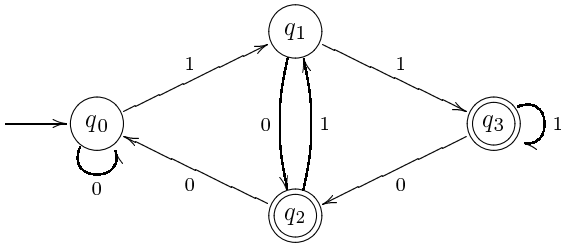

Abbildung 4.10: Überführungsgraph des DEA A' aus Beispiel 4.38

den Abbildungen 4.7 und 4.10 bis auf Bezeichnungen der Zustände identisch sind. Also akzeptieren sie auch die gleiche Sprache.

Bemerkung 4.39 1. Beim Konstruieren des äquivalenten DEA zum gegebenen NEA im obigen Beispiel haben wir den gleichen Automaten (bis auf Bezeichnungen) erhalten, den wir auch schon vorher betrachtet hatten. Das muss natürlich nicht immer sein. Insbesondere erhält man im Allgemeinen bei dieser Konstruktion Automaten, die nicht *minimal* in dem Sinne sind, dass man äquivalente DEA finden kann, die eventuell weniger Zustände haben.

2. Im obigen Beispiel hatte der NEA drei Zustände, der DEA vier. Führt man den gleichen Übergang vom NEA zum DEA für die (von der Struktur gleiche) Sprache

$$T(A) = \{w \in \{0,1\}^* \mid w = u1v \text{ mit } u \in \{0,1\}^*,\ v \in \{0,1\}^9\}$$

durch, also für die Menge aller Wörter über dem Alphabet $\{0,1\}$, deren zehntletztes Symbol eine 1 ist, so benötigt der NEA 11 Zustände, der DEA aber 2^{10} Zustände. Es lässt sich zeigen, dass es keinen DEA für diese Sprache mit weniger Zustände gibt. Dieses Resultat kann man für beliebiges n verallgemeinern. Die Anzahl der Zustände beim Übergang vom NEA zum DEA kann also *exponentiell wachsen*.

Wir haben das Modell des nichtdeterministischen endlichen Automaten eingeführt, weil wir letztendlich beweisen wollten, dass die Klasse der Typ-3-Sprachen und die Klasse der Sprachen, die von deterministischen endlichen Automaten akzeptiert werden, identisch sind. Mit dem folgenden Satz kommen wir diesem Beweis sehr nahe.

Satz 4.40 *Sei G eine reguläre Grammatik, also vom Typ 3, dann existiert ein nichtdeterministischer endlicher Automat A mit $T(A) = L(G)$.*

Beweis. Wie bereits angekündigt, benutzt der Beweis eigentlich die gleiche Idee wie beim Übergang vom DEA zur regulären Grammatik. Sei $G = (V, \Sigma, P, S)$ die reguläre Grammatik. Wir konstruieren einen NEA $A = (Z, \Sigma, \delta, z_0, E)$ wie folgt:

$$Z = V \cup \{X\},$$
$$z_0 = S,$$
$$E = \begin{cases} \{S, X\} & \text{für } S \to \varepsilon \in P, \\ \{X\} & \text{für } S \to \varepsilon \notin P, \end{cases}$$

Des weiteren definieren wir die Überführungsfunktion δ durch

$$\delta(A, a) = \begin{cases} \{B \mid A \to aB \in P\} \cup \{X\} & \text{für } A \to a \in P, \\ \{B \mid A \to aB \in P\} & \text{für } A \to a \notin P, \end{cases}$$

für $A \in V$ und $a \in \Sigma$.

Jetzt könnten und müssten wir beweisen, dass $T(A) = L(G)$ gilt, worauf wir aber an dieser Stelle wiederum verzichten wollen. □

Die Abbildung 4.11 fasst die Ergebnisse der Sätze 4.28, 4.40 sowie 4.37 zusammen. Damit gilt:

Folgerung 4.41 *Die Klasse der regulären Sprachen* (**Typ 3**) *ist gleich der Klasse der von nichtdeterministischen endlichen Automaten akzeptierten Sprachen* ($\mathcal{L}$(NEA)) *und der Klasse der von deterministischen endlichen Automaten akzeptierten Sprachen* ($\mathcal{L}$(DEA))*, also*

$$\textbf{Typ 3} = \mathcal{L}(\text{NEA}) = \mathcal{L}(\text{DEA}).$$

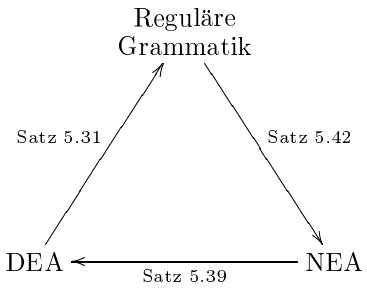

Abbildung 4.11: Beweisschema für Mechanismen zur Beschreibung regulärer Sprachen

4.3.3 Reguläre Ausdrücke

Nachdem wir in den vorhergehenden Kapiteln die Menge der regulären Sprachen (oder Typ-3-Sprachen) durch Grammatiken und Automaten beschrieben haben, wollen wir in diesem Kapitel eine Beschreibungsart betrachten, die algebraischer Natur ist, nämlich die regulären Ausdrücke. Obwohl die Beschreibung auf algebraische Operationen zurückgeht, also eigentlich recht mathematischer Natur ist, werden reguläre Ausdrücke in vielen Gebieten der Informatik angewendet, zum Beispiel bei der Suche in Editoren.

Die Menge der regulären Ausdrücke über einem Alphabet Σ werden wir induktiv definieren.

Definition 4.42 (Reguläre Ausdrücke) *Sei Σ ein Alphabet, dann gilt:*

(i) *$\emptyset$ ist ein regulärer Ausdruck über Σ.*

(ii) *ε ist ein regulärer Ausdruck über Σ.*

(iii) *Für jedes $a \in \Sigma$ ist a ein regulärer Ausdruck über Σ.*

(iv) *Wenn α und β reguläre Ausdrücke über Σ sind, so auch $\alpha\beta$, $(\alpha \mid \beta)$ und $(\alpha)^*$.*

Reguläre Ausdrücke über einem Alphabet Σ sind also erst einmal nur Wörter spezieller Art über diesem Alphabet. Nun ordnen wir solch einem Wort eine Sprache zu. Die Definition dieser Zuordnung erfolgt wiederum rekursiv.

Definition 4.43 (Sprache eines regulären Ausdrucks) *Sei Σ ein Alphabet und γ ein regulärer Ausdruck über Σ, dann wird die von γ beschriebene Sprache $L(\gamma) \subseteq \Sigma^*$ wie folgt definiert.*

(i) *Für $\gamma = \emptyset$ gilt $L(\gamma) = \emptyset$.*

(ii) *Für $\gamma = \varepsilon$ gilt $L(\gamma) = \{\varepsilon\}$.*

(iii) *Für $\gamma = a$ mit $a \in \Sigma$ gilt $L(\gamma) = \{a\}$.*

(iv) *Für $\gamma = \alpha\beta$ gilt $L(\gamma) = L(\alpha) \cdot L(\beta)$.*

(v) *Für $\gamma = (\alpha \mid \beta)$ gilt $L(\gamma) = L(\alpha) \cup L(\beta)$.*

(vi) *Für $\gamma = (\alpha)^*$ gilt $L(\gamma) = (L(\alpha))^*$.*

Beispiel 4.44 Wir betrachten den regulären Ausdruck

$$(0 \mid (0 \mid 1)^*00).$$

Dann können wir die zugeordnete Sprache wie folgt gemäß der Definition bilden (hier für ein erstes

Beispiel sehr ausführlich aufgeschrieben):

$$\begin{aligned}
L((0 \mid (0 \mid 1)^*00)) &= L(0) \cup L((0 \mid 1)^*00)) \\
&= L(0) \cup (L((0 \mid 1)^*0)) \cdot L(0)) \\
&= L(0) \cup ((L((0 \mid 1)^*)) \cdot L(0)) \cdot L(0)) \\
&= L(0) \cup (((L((0 \mid 1)))^* \cdot L(0)) \cdot L(0)) \\
&= L(0) \cup (((L(0) \cup L(1))^* \cdot L(0)) \cdot L(0)) \\
&= \{0\} \cup ((((\{0\} \cup \{1\})^* \cdot \{0\}) \cdot \{0\}) \\
&= \{0\} \cup ((\{0,1\}^* \cdot \{0\}) \cdot \{0\}) \\
&= \{0\} \cup (\{0,1\}^* \cdot \{00\}),
\end{aligned}$$

das heißt, die vom regulären Ausdruck $(0 \mid (0 \mid 1)^*00)$ beschriebene Sprache ist die Menge aller Wörter über dem Alphabet $\{0,1\}$, die gleich 0 sind oder auf 00 enden.

Bemerkung 4.45 1. Wir vereinbaren, dass wir Klammern, die nicht notwendigerweise gebraucht werden, weglassen können. Zum Beispiel können wir statt $(\alpha \mid (\beta \mid \gamma))$ auch $(\alpha \mid \beta \mid \gamma)$ schreiben. Wir schreiben auch $L(\alpha \mid \beta)$ statt $L((\alpha \mid \beta))$ sowie a^* statt $(a)^*$.

2. Wir benutzen die abkürzende Schreibweise α^n für $\underbrace{\alpha\alpha\ldots\alpha}_{n\text{-mal}}$.

3. Wir benutzen die abkürzende Schreibweise α^+ für $\alpha^*\alpha$.

4. In der Literatur findet man oft auch abweichende Definitionen der regulären Ausdrücke. Zum Beispiel findet man für $(\alpha \mid \beta)$ auch $(\alpha + \beta)$ oder auch $(\alpha \cup \beta)$. Auch wird natürlich oft $\alpha \cdot \beta$ für $\alpha\beta$ zugelassen.

5. Oft wird in der Literatur zwischen regulärem Ausdruck und beschriebener Sprache nicht unterschieden, das heißt, man identifiziert einen regulären Ausdruck mit der beschriebenen Sprache.

Ich gebe noch ein paar weitere Beispiele an:

Beispiel 4.46 Weitere Beispiele für reguläre Ausdrücke über $\Sigma = \{a, b\}$ und deren zugeordneten Sprachen sind:

$(a \mid b)^*$ beschreibt die Menge aller Wörter über dem Alphabet $\{a, b\}$.

$(a \mid b)^+$ beschreibt die Menge aller Wörter über dem Alphabet $\{a, b\}$, die nicht dem leeren Wort entsprechen.

$(a \mid b)^*aba(a \mid b)^*$ beschreibt die Menge aller Wörter über dem Alphabet $\{a, b\}$, die das Teilwort aba haben.

$(a \mid b)^*a(a \mid b)^2$ beschreibt die Menge aller Wörter über dem Alphabet $\{a, b\}$, deren drittletztes Symbol ein a ist.

$((a \mid b)(a \mid b))^*$ beschreibt die Menge aller Wörter über dem Alphabet $\{a, b\}$, deren Länge gerade ist.

$(b \mid \varepsilon)(ab)^*(a \mid \varepsilon)$ beschreibt die Menge aller Wörter über dem Alphabet $\{a, b\}$, die nicht das Teilwort aa und nicht das Teilwort bb enthalten.

Wenn man sich die regulären Ausdrücke und die beschriebenen Sprachen im obigen Beispiel anschaut, erkennt man, dass alle Sprachen vom Typ 3 sind, also regulär. Man stellt sich natürlich die Frage, ob alle durch reguläre Ausdrücke beschreibbaren Sprachen regulär sind und ob umgekehrt für jede reguläre Sprache ein regulärer Ausdruck existiert, der sie beschreibt. Die Bezeichnung *reguläre* Ausdrücke suggeriert natürlich die Antwort.

Satz 4.47 (Kleene) *Die Menge der durch reguläre Ausdrücke beschreibbaren Sprachen ist genau die Menge der regulären Sprachen.*

Beweis. Der Beweis muss in zwei Richtungen geführt werden.

Teil 1: Einerseits muss gezeigt werden, dass jeder reguläre Ausdruck eine reguläre Sprache beschreibt. Diesen Teil wollen wir an dieser Stelle skizzieren. Wir werden zeigen, dass zu jedem regulären Ausdruck ein NEA existiert, der genau die vom regulären Ausdruck beschriebene Sprache akzeptiert, womit wegen Folgerung 4.41 diese Sprache regulär oder vom Typ 3 ist.

Sei Σ ein Alphabet und γ ein regulärer Ausdruck über Σ. Wir betrachten zuerst die Fälle, in denen γ die Form $\gamma = \emptyset$, $\gamma = \varepsilon$ oder $\gamma = a$ mit $a \in \Sigma$ hat. In Abbildung 4.12 sind NEA's

Abbildung 4.12: DEA's für $L(\emptyset)$, $L(\varepsilon)$ sowie $L(a)$ (von links nach rechts)

angegeben, die jeweils die Mengen $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$ sowie $L(a) = \{a\}$ akzeptieren.

Wir nehmen jetzt an, dass γ ein regulärer Ausdruck ist, der aus regulären Ausdrücken zusammengesetzt ist. Dann gibt es für γ die drei Fälle $\gamma = \alpha\beta$, $\gamma = (\alpha \mid \beta)$ und $\gamma = (\alpha)^*$.

Sei zunächst γ ein regulärer Ausdruck der Form $\gamma = \alpha\beta$. Dabei können wir annehmen, dass es bereits NEA's für $L(\alpha)$ und $L(\beta)$ gibt, also dass NEA's A_α und A_β existieren mit $T(A_\alpha) = L(\alpha)$ und $T(A_\beta) = L(\beta)$. Nun konstruieren wir den Automaten A, indem wir die Automaten A_α und

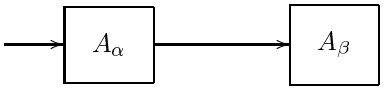

Abbildung 4.13: Schema des NEA's für $L(\alpha\beta)$

A_β im Prinzip hintereinander schalten („in Reihe") (siehe Schema in Abbildung 4.13).

Die genaue Konstruktion für den NEA A ist folgende: Die Automaten $A_\alpha = (Z_\alpha, \Sigma, \delta_\alpha, z_{\alpha 0}, E_\alpha)$ und $A_\beta = (Z_\beta, \Sigma, \delta_\beta, z_{\beta 0}, E_\beta)$ seien die Automaten mit $T(A_\alpha) = L(\alpha)$ und $T(A_\beta) = L(\beta)$. Wir konstruieren $A = (Z, \Sigma, \delta, z_0, E)$ wie folgt.

- Jeder Zustand von A_α und A_β ist auch Zustand von A, also $Z = Z_\alpha \cup Z_\beta$.
- Der Anfangszustand von A_α ist auch Anfangszustand für A, also $z_0 = z_{\alpha 0}$.
- Die Menge der Endzustände von A_β wird die Menge der Endzustände von A, also $E = E_\beta$.
- Alle Überführungen von A_α und A_β gelten auch für A. Von allen Zuständen von A_α, für die es Überführungen zu Endzuständen von A_α gibt, gibt es zusätzlich Überführungen für das gleiche Symbol zum Anfangszustand von A_β. Formal gilt für alle $z \in Z$ und $a \in \Sigma$:

$$\delta(z,a) = \begin{cases} \delta_\beta(z,a) & \text{für } z \in Z_\beta, \\ \delta_\alpha(z,a) & \text{für } z \in Z_\alpha \text{ und } \delta_\alpha(z,a) \cap E_\alpha = \emptyset, \\ \delta_\alpha(z,a) \cup \{z_{\beta 0}\} & \text{für } z \in Z_\alpha \text{ und } \delta_\alpha(z,a) \cap E_\alpha \neq \emptyset. \end{cases}$$

Für den Fall $\varepsilon \in L(\alpha)$ muss man noch Sonderregelungen treffen, darauf verzichten wir hier.

Wie man nachweisen kann, gilt dann $T(A) = L(\alpha\beta)$, das heißt A akzeptiert ein Wort genau dann, wenn ein erster Teil des Wortes von A_α und der Rest des Wortes von A_β akzeptiert wird.

Habe γ nun die Form $\gamma = (\alpha \mid \beta)$. Wir setzen wieder voraus, dass die Automaten $A_\alpha = (Z_\alpha, \Sigma, \delta_\alpha, z_{\alpha 0}, E_\alpha)$ und $A_\beta = (Z_\beta, \Sigma, \delta_\beta, z_{\beta 0}, E_\beta)$ die Automaten mit $T(A_\alpha) = L(\alpha)$ und $T(A_\beta) = L(\beta)$ seien. Wir konstruieren dann den Automaten A in folgender Art und Weise (wir schalten die Automaten A_α und A_β im Prinzip „parallel", siehe Abbildung 4.14):

$$A = (Z_\alpha \cup Z_\beta \cup \{z_0\}, \Sigma, \delta, z_0, E_\alpha \cup E_\beta),$$

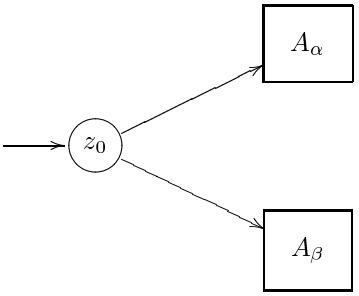

Abbildung 4.14: Schema des NEA's für $L((\alpha \mid \beta))$

die Funktion δ ist dabei wie folgt definiert.

$$\delta(z,a) = \begin{cases} \delta_\alpha(z,a) & \text{für } z \in Z_\alpha, \\ \delta_\beta(z,a) & \text{für } z \in Z_\beta, \\ \delta_\alpha(z_{\alpha 0},a) \cup \delta_\beta(z_{\beta 0},a) & \text{für } z = z_0. \end{cases}$$

Der Automat akzeptiert dann ein Wort genau dann, wenn es von A_α *oder* A_β akzeptiert wird. Also gilt $T(A) = A_\alpha \cup A_\beta$, das heißt $T(A) = L((\alpha \mid \beta))$.

Habe γ nun die Form $\gamma = (\alpha)^*$. Wir setzen wiederum voraus, dass der Automat $A_\alpha = (Z_\alpha, \Sigma, \delta_\alpha, z_{\alpha 0}, E_\alpha)$ der Automat mit $T(A_\alpha) = L(\alpha)$ ist. Wir konstruieren A, indem wir im Prinzip

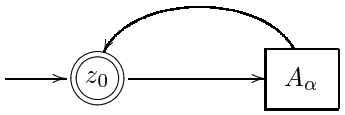

Abbildung 4.15: Schema des NEA's für $L((\alpha)^*)$

eine Schleife erzeugen (siehe Abbildung 4.15). Exakt wird dann $A = (Z, \Sigma, \delta, z_0, E)$ folgendermaßen konstruiert.

$$Z = Z_\alpha \cup \{z_0\}, \qquad E = \{z_0\}$$

und für die Überführungsfunktion δ gilt für alle $z \in Z$ und $a \in \Sigma$:

$$\delta(z,a) = \begin{cases} \delta_\alpha(z_{\alpha 0},a) & \text{für } z = z_0, \\ \delta_\alpha(z,a) & \text{für } z \in Z_\alpha \text{ und } \delta_\alpha(z,a) \cap E_\alpha = \emptyset, \\ \delta_\alpha(z,a) \cup \{z_0\} & \text{für } z \in Z_\alpha \text{ und } \delta_\alpha(z,a) \cap E_\alpha \neq \emptyset. \end{cases}$$

Zur Interpretation: Anfangszustand ist also ein neuer Zustand, der auch gleichzeitig der einzige Endzustand ist. Vom Anfangszustand gibt es dann die gleichen Überführungen wie vom Anfangszustand des Automaten A_α. Von allen Zuständen von A_α, für die es Überführungen zu Endzuständen von A_α gibt, gibt es zusätzlich Überführungen für das gleiche Symbol zum Anfangszustand z_0.

Damit wird das leere Wort akzeptiert (wegen $z_0 \in E$) und auch alle Wörter, die auch A_α akzeptiert. Weiter können wir natürlich die Schleife mehrmals durchlaufen, das heißt für die akzeptierte Sprache von A gilt

$$\begin{aligned} T(A) &= \{\varepsilon\} \cup T(A_\alpha) \cup (T(A_\alpha))^2 \cup (T(A_\alpha))^3 \cup \cdots \\ &= (T(A_\alpha))^0 \cup T(A_\alpha)^1 \cup (T(A_\alpha))^2 \cup (T(A_\alpha))^3 \cup \cdots \\ &= (T(A_\alpha))^*, \end{aligned}$$

also $T(A) = (L(\alpha))^*$.

Damit hätten wir alle Fälle behandelt und haben den ersten Teil des Beweises vollendet (abgesehen von den Lücken, die wir hier nicht vollständig bewiesen haben). Das heißt, wir haben gezeigt, zu jedem regulären Ausdruck gibt es einen äquivalenten nichtdeterministischen endlichen Automaten, also ist jede von einem regulären Ausdruck beschriebene Sprache regulär.

Noch eine allgemeine Bemerkung zu den obigen Konstruktionen, natürlich müssen wir $Z_\alpha \cap Z_\beta = \emptyset$ fordern, und falls wir einen Anfangszustand z_0 neu hinzunehmen, darf er natürlich in den gegebenen Automaten nicht vorkommen. Das ist aber keine Einschränkung, da wir durch einfache Umbenennungen der Zustände diese Bedingungen immer absichern können.

Teil 2: Im zweiten Teil des Beweises zeigen wir, dass jede reguläre Sprache von einem regulären Ausdruck beschrieben wird.

Sei L eine reguläre Sprache. Dann wird L von einem deterministischen endlichen Automaten $A = (Z, \Sigma, \delta, z_0, E)$ akzeptiert, wobei $\Sigma = \{a_1, a_2, \ldots, a_k\}$ gelte. Für jeden Zustand $z \in Z$ definieren wir die Wortmenge $L_z = \{w \in \Sigma^* \mid \hat{\delta}(z, w) \in E\}$. Für alle diese L_z werden wir zeigen, dass sie von regulären Ausdrücken beschrieben werden können. Wegen $L = T(A) = L_{z_0}$ würde dann die Behauptung folgen.

Die Mengen L_z stehen in folgenden Beziehungen zueinander.

$$\begin{aligned} L_z &= \bigcup_{i=1}^{k} \{a_i\} \cdot L_{\delta(z,a_i)} && \text{für } z \notin E \text{ und} \\ L_z &= \bigcup_{i=1}^{k} \{a_i\} \cdot L_{\delta(z,a_i)} \cup \{\varepsilon\} && \text{für } z \in E. \end{aligned} \tag{4.2}$$

Wir werden jetzt dieses System von Gleichungen, in denen die L_z als Unbekannte auftreten, schrittweise auflösen, wobei wir nur die Operationen Vereinigung, Konkatenation und Iteration verwenden, so dass letztendlich nur L_{z_0} stehen bleibt und somit von einem regulären Ausdruck beschrieben werden kann.

Die Tatsache, dass Unbekannte auf beiden Seiten ein und derselben Gleichung auftreten können, führt dazu, dass hier ein „normales“ Eliminieren nicht möglich ist.

Hier hilft folgendes Lemma,

Lemma 4.48 *Für $B, C \in \Sigma^*$ gilt: Ist $\varepsilon \notin B$, so ist $L = B^* \cdot C$ die einzige Lösung der Gleichung $L = B \cdot L \cup C$.*

Beweis. Der Beweis erfolgt durch das Zeigen der Inklusionen $B^* \cdot C \subseteq L$ sowie $L \subseteq B^* \cdot C$.

Teil 1: $B^* \cdot C \subseteq L$. Wir zeigen durch Induktion über k, dass jedes $B^k \cdot C$ in jeder Lösung L enthalten ist.

Induktionsanfang. Für $k = 0$ gilt:

$$B^0 \cdot C = \{\varepsilon\} \cdot C = C \subseteq B \cdot L \cup C = L.$$

Induktionsschritt. Mit der Induktionsvoraussetzung $B^k \cdot C \subseteq L$ schließt man

$$B^{k+1} \cdot C = B \cdot (B^k \cdot C) \subseteq B \cdot L \subseteq B \cdot L \cup C = L.$$

Teil 2: $L \subseteq B^* \cdot C$. Wir zeigen durch Induktion über die Länge des Wortes w, dass aus $w \in L$ stets $w \in B^* \cdot C$ folgt.

Induktionsanfang. Ist $|w| = 0$, so ist $w = \varepsilon$. Gilt $\varepsilon \in L$, so folgt aus $L = B \cdot L \cup C$ und $\varepsilon \notin B$ sofort $\varepsilon \in C$. Dann ist aber auch $\varepsilon \in B^* \cdot C$.

Induktionsschritt. Es sei $|w| > 0$, und nach Induktionsvoraussetzung folge für alle v mit $|v| < |w|$ aus $v \in L$ stets $v \in B^* \cdot C$.

Es sei nun $w \in L = B \cdot L \cup C$. Im Falle $w \in C$ folgt sofort $w \in B^* \cdot C$. Im Falle $w \in B \cdot L$ gibt es ein $u \in B$ und ein $v \in L$ mit $w = u \cdot v$. Wegen $\varepsilon \notin B$ gilt $|u| > 0$ und folglich $|v| < |w|$. Nach Induktionsvoraussetzung haben wir damit $v \in B^* \cdot C$, und wir schließen $w = u \cdot v \in B \cdot (B^* \cdot C) \subseteq B^* \cdot C$. □

Mit diesem Lemma können wir jetzt L_{z_0} bestimmen, indem wir schrittweise alle anderen Mengenvariablen L_z wie folgt eliminieren. Kommt in der Gleichung, bei der links L_z steht, auch rechts L_z vor, so wird das Lemma angewendet. In jedem Fall hat man dann eine Gleichung $L_z = R$, wobei in R die Variable L_z nicht mehr vorkommt. Ersetzt man jetzt in allen anderen Gleichungen L_z durch R, so ist L_z eliminiert.

Wenn wir alle Variablen bis auf L_{z_0} eliminiert haben, ist L_{z_0} dargestellt als endlich oftmalige Anwendung der Operationen Konkatenation, Vereinigung und Iteration über Symbolen aus Σ. Somit kann $L_{z_0} = T(A)$ durch einen regulären Ausdruck beschrieben werden.

Somit ist der zweite Teil des Satzes von Kleene und damit auch der Satz bewiesen. □

Wir betrachten ein Beispiel für die Konstruktion eines äquivalenten regulären Ausdrucks für einen gegebenen deterministischen endlichen Automaten.

Beispiel 4.49 In der Abbildung 4.16 ist ein DEA über dem Alphabet $\Sigma = \{a, b\}$ durch einen

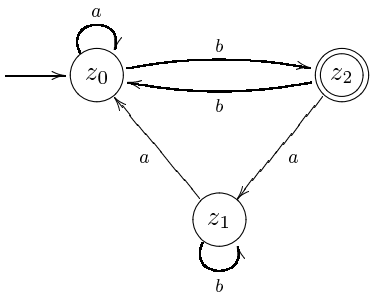

Abbildung 4.16: Ein deterministischer endlicher Automat

Überführungsgraphen gegeben. Das dazugehörige Gleichungssystem für die Wortmengen L_z besteht somit aus den drei Gleichungen

$$\left.\begin{aligned} L_{z_0} &= \{a\} \cdot L_{z_0} \cup \{b\} \cdot L_{z_2}, \\ L_{z_1} &= \{a\} \cdot L_{z_0} \cup \{b\} \cdot L_{z_1}, \\ L_{z_2} &= \{a\} \cdot L_{z_1} \cup \{b\} \cdot L_{z_0} \cup \{\varepsilon\}. \end{aligned}\right\} \tag{4.3}$$

Die Variable (Wortmenge) L_{z_2} in der ersten Gleichung können wir ohne Probleme eliminieren, indem wir sie durch die rechte Seite der dritten Gleichung ersetzen. Somit erhalten wir das zu (4.3) äquivalente Gleichungssystem

$$\begin{aligned} L_{z_0} &= \{a\} \cdot L_{z_0} \cup \{b\} \cdot (\{a\} \cdot L_{z_1} \cup \{b\} \cdot L_{z_0} \cup \{\varepsilon\}), \\ L_{z_1} &= \{a\} \cdot L_{z_0} \cup \{b\} \cdot L_{z_1} \end{aligned}$$

oder, indem wir die rechte Seite der ersten Gleichung zusammenfassen,

$$\begin{aligned} L_{z_0} &= (\{a\} \cup \{bb\}) \cdot L_{z_0} \cup \{ba\} \cdot L_{z_1} \cup \{b\}, \\ L_{z_1} &= \{a\} \cdot L_{z_0} \cup \{b\} \cdot L_{z_1}. \end{aligned}$$

Nun wenden wir auf die zweite Gleichung das Lemma 4.48 an und erhalten $L_{z_1} = \{b\}^*\{a\} \cdot L_{z_0}$. Damit können wir jetzt L_{z_1} in der ersten Gleichung des Gleichungssystems ersetzen.

$$\begin{aligned} L_{z_0} &= (\{a\} \cup \{bb\}) \cdot L_{z_0} \cup \{ba\} \cdot L_{z_1} \cup \{b\} \\ &= (\{a\} \cup \{bb\}) \cdot L_{z_0} \cup \{ba\} \cdot (\{b\}^*\{a\} \cdot L_{z_0}) \cup \{b\} \\ &= (\{a\} \cup \{bb\} \cup \{ba\} \cdot \{b\}^*\{a\}) \cdot L_{z_0} \cup \{b\}. \end{aligned}$$

Nun wenden wir abermals unser Lemma an und erhalten

$$L_{z_0} = (\{a\} \cup \{bb\} \cup \{ba\} \cdot \{b\}^*\{a\})^*\{b\}.$$

Somit können wir L_{z_0}, also die vom gegebenen Automaten akzeptierte Sprache, durch den regulären Ausdruck

$$(a \mid bb \mid bab^*a)^*b$$

beschreiben.

Betrachten wir ein weiteres Beispiel.

Beispiel 4.50 Es ist ein regulärer Ausdruck zu bestimmen, der die Menge aller Wörter über dem Alphabet $\{a, b\}$, die nicht das Teilwort *bab* enthalten, beschreibt.

Wir konstruieren zuerst den DEA, der die Menge aller Wörter über dem Alphabet $\{a, b\}$, die das Teilwort *bab* enthalten, beschreibt (siehe Abbildung 4.17). Jetzt können wir daraus sehr einfach

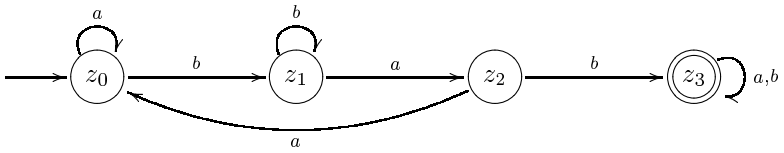

Abbildung 4.17: Ein DEA für die Menge aller Wörter über $\{a, b\}$, die das Teilwort *bab* enthalten

einen DEA konstruieren, der die Komplementärmenge akzeptiert, nämlich durch Vertauschen der akzeptierenden und nichtakzeptierenden Zustände (siehe Abbildung 4.18).

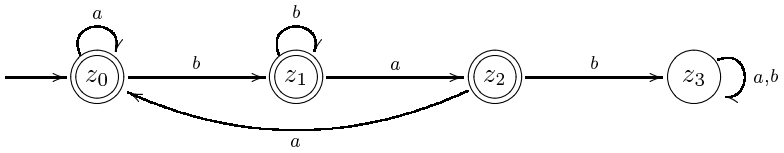

Abbildung 4.18: Ein DEA für die Menge aller Wörter über $\{a, b\}$, die nicht das Teilwort *bab* enthalten

Nun können wir für den DEA aus Abbildung 4.18 das Gleichungssystem der Wortmengen für die einzelnen Zustände aufstellen.

$$\begin{aligned}
L_{z_0} &= \{a\} \cdot L_{z_0} \cup \{b\} \cdot L_{z_1} \cup \{\varepsilon\},\\
L_{z_1} &= \{a\} \cdot L_{z_2} \cup \{b\} \cdot L_{z_1} \cup \{\varepsilon\},\\
L_{z_2} &= \{a\} \cdot L_{z_0} \cup \{b\} \cdot L_{z_3} \cup \{\varepsilon\},\\
L_{z_3} &= \{a\} \cdot L_{z_3} \cup \{b\} \cdot L_{z_3}.
\end{aligned}$$

Aus der vierten Gleichung erhalten wir $L_{z_3} = \{a, b\} \cdot L_{z_3}$ und durch Anwendung des Lemmas 4.48 schließlich $L_{z_3} = \{a, b\}^* \cdot \emptyset = \emptyset$. Setzen wir dies in die dritte Gleichung ein, erhalten wir folgendes neue Gleichungssystem.

$$\begin{aligned}
L_{z_0} &= \{a\} \cdot L_{z_0} \cup \{b\} \cdot L_{z_1} \cup \{\varepsilon\},\\
L_{z_1} &= \{a\} \cdot L_{z_2} \cup \{b\} \cdot L_{z_1} \cup \{\varepsilon\},\\
L_{z_2} &= \{a\} \cdot L_{z_0} \cup \{\varepsilon\}.
\end{aligned}$$

Mittels der dritten Gleichung können wir L_{z_2} in der zweiten Gleichung ersetzen und erhalten

$$\begin{aligned}
L_{z_1} &= \{a\} \cdot L_{z_2} \cup \{b\} \cdot L_{z_1} \cup \{\varepsilon\}\\
&= \{a\} \cdot (\{a\} \cdot L_{z_0} \cup \{\varepsilon\}) \cup \{b\} \cdot L_{z_1} \cup \{\varepsilon\}\\
&= \{aa\} \cdot L_{z_0} \cup \{b\} \cdot L_{z_1} \cup \{\varepsilon, a\}\\
&= \{b\} \cdot L_{z_1} \cup \{aa\} \cdot L_{z_0} \cup \{\varepsilon, a\}.
\end{aligned}$$

Wenden wir nun auf diese Gleichung unser Lemma an, erhalten wir

$$\begin{aligned} L_{z_1} &= \{b\}^* \cdot (\{aa\} \cdot L_{z_0} \cup \{\varepsilon, a\}) \\ &= \{b\}^* \cdot \{aa\} \cdot L_{z_0} \cup \{b\}^* \cdot \{\varepsilon, a\}. \end{aligned}$$

Setzen wir nun dieses Ergebnis in die Gleichung für L_{z_0} ein, so haben wir

$$\begin{aligned} L_{z_0} &= \{a\} \cdot L_{z_0} \cup \{b\} \cdot L_{z_1} \cup \{\varepsilon\} \\ &= \{a\} \cdot L_{z_0} \cup \{b\} \cdot (\{b\}^* \cdot \{aa\} \cdot L_{z_0} \cup \{b\}^* \cdot \{\varepsilon, a\}) \cup \{\varepsilon\} \\ &= (\{a\} \cup \{b\} \cdot \{b\}^* \cdot \{aa\}) \cdot L_{z_0} \cup \{b\} \cdot \{b\}^* \cdot \{\varepsilon, a\} \cup \{\varepsilon\}. \end{aligned}$$

Daraus erhalten wir nach Anwendung unseres Lemmas

$$L_{z_0} = (\{a\} \cup \{b\} \cdot \{b\}^* \cdot \{aa\})^* \cdot (\{b\} \cdot \{b\}^* \cdot \{\varepsilon, a\} \cup \{\varepsilon\}),$$

und somit beschreibt der reguläre Ausdruck

$$(a \mid bb^*aa)^*(bb^*(\varepsilon \mid a) \mid \varepsilon)$$

unsere gegebene Menge der Wörter über dem Alphabet $\{a, b\}$, die nicht das Teilwort *bab* enthalten.

4.3.4 Das Pumping Lemma

In diesem Kapitel behandeln wir einen Satz, mit dem man zeigen kann, dass eine Sprache *nicht* regulär ist.

Satz 4.51 (Pumping Lemma) *Sei L eine reguläre Sprache. Dann gibt es eine Konstante $k \in \mathbb{N}$, so dass für alle Wörter $z \in L$ mit $|z| \geq k$ Wörter u, v und w existieren, so dass gilt:*

(i) $z = uvw$,

(ii) $|uv| \leq k$ *and* $|v| \geq 1$,

(iii) *für alle* $i \in \mathbb{N}$ *gilt* $uv^iw \in L$.

Beweis. Sei L eine reguläre Sprache, dann gibt es einen DEA A, der L akzeptiert. Sei nun k die Anzahl der Zustände von A.

Wir betrachten ein Wort z, welches von A akzeptiert wird und für das $|z| \geq k$ gilt. Beim Einlesen von z durchläuft der DEA offensichtlich $|z| + 1 \geq k + 1$ Zustände (einschließlich des Startzustands). Da der Automat aber nur k Zustände hat, muß der DEA A beim Einlesen von z zweimal denselben Zustand durchlaufen, also durchläuft der Automat eine Schleife. Wir bezeichnen mit u den Teil des Wortes, den A vor Erreichen der Schleife eingelesen hat, mit v den Teil, der während der Schleife eingelesen wurde, mit w den Rest des Wortes. Offensichtlich gilt $|uv| \leq k$ und $v \geq 1$.

Da wir eine Schleife durchlaufen haben, kann der Automat natürlich auch die Schleife meiden ($i = 0$) oder zweimal durchlaufen ($i = 2$) oder beliebig oft durchlaufen. Mit anderen Worten, für alle $i \geq 0$ gilt $uv^iw \in L$, was zu beweisen war. □

Bemerkung 4.52 Schaut man sich die logische Struktur des Pumping Lemmas an, erkennt man, dass es sich um eine Implikation handelt, dass heißt, die Umkehrung *muss nicht* wahr sein. Hier ist es tatsächlich so, es gibt Sprachen, die nicht regulär sind, für die aber die Behauptungen des Pumping Lemmas erfüllt sind (siehe Abbildung 4.19). Damit eignet sich das Pumping Lemma gut, um zu zeigen, dass gewisse Sprachen *nicht regulär* sind, man kann aber auf *keinen Fall* zeigen, dass gewisse Sprachen *regulär* sind!!!.

Beispiel 4.53 Wir betrachten die Sprache

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

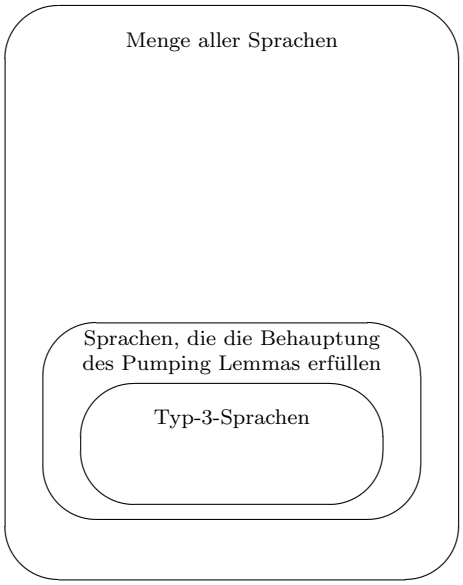

Abbildung 4.19: Hierarchie mit Menge der Sprachen, die die Behauptung des Pumping Lemmas erfüllen

von der wir schon länger vermuten, dass sie nicht regulär ist. Das wollen wir jetzt mit dem Pumping Lemma durch die indirekte Beweismethode beweisen.

Dazu nehmen wir an, L sei regulär. Dann gibt es laut Pumping Lemma eine Konstante k, so dass für alle Wörter $z \in L$ mit $|z| \geq k$ Wörter u, v und w existieren, so dass gilt:

(i) $z = uvw$,

(ii) $|uv| \leq k$ and $|v| \geq 1$,

(iii) für alle $i \in \mathbb{N}$ gilt $uv^i w \in L$.

Betrachten wir speziell das Wort $z = a^k b^k$, dann gilt offensichtlich $|z| = 2\,k \geq k$, also gibt es eine Zerlegung $z = a^k b^k = uvw$ mit $|uv| \leq k$ and $|v| \geq 1$. Daraus ergibt sich:

$$\begin{aligned} uv &= a^r \quad \text{für } r \leq k, \\ w &= a^{k-r} b^k, \\ v &= a^s \quad \text{für } 1 \leq s \leq r \leq k. \end{aligned}$$

Nach (iii) ist jedes Wort $uv^i w \in L$, also auch

$$uv^2 w \in L. \tag{4.4}$$

Andererseits gilt

$$uv^2 w = uvvw = a^r a^s a^{k-r} b^k = a^{k+s} b^k$$

mit $s \geq 1$, also $k + s \neq k$, das heißt

$$uv^2 w \notin L, \tag{4.5}$$

was einen Widerspruch zu 4.4 darstellt. Folglich war unsere Annahme (L regulär) falsch, also kann $L = \{a^n b^n \mid n \geq 1\}$ *nicht regulär* sein.

Beispiel 4.54 Mit Hilfe des Pumping Lemmas kann man auch die Nichtregularität der Sprachen

$L = \{a^{n^2} \mid n \geq 1\}$,
$L = \{a^{2^n} \mid n \geq 1\}$,
$L = \{a^p \mid p \text{ Primzahl}\}$,
$L = \{a^n b^n c^n \mid n \geq 1\}$,
$L = \{ww^R \mid w \in \{a, b\}^*\}$,
$L = \{ww \mid w \in \{a, b\}^*\}$,
$L = \text{EXPR}$

und vielen anderen zeigen.

4.3.5 Abschlusseigenschaften

Wir wollen in diesem Kapitel untersuchen, unter welchen Operationen die Menge der regulären Sprachen abgeschlossen ist. Eine Menge M ist *unter der k-stelligen Operation* op *abgeschlossen*, falls für alle $a_1, a_2, \ldots, a_k \in M$ auch $\text{op}(a_1, a_2, \ldots, a_k) \in M$ gilt.

Satz 4.55 *Die Menge der regulären Sprachen ist unter den Operationen*

(i) *Vereinigung,*
(ii) *Durchschnitt,*
(iii) *Komplement,*
(iv) *Produkt (Konkatenation) und*
(v) *Kleene-Stern*

abgeschlossen.

Beweis. (i), (iv) und (v): Der Abschluss unter den Operationen Vereinigung, Produkt und Kleene-Stern ist eine direkte Folgerung aus der Definition der regulären Ausdrücke, da mit zwei gegebenen regulären Ausdrücken α und β auch $\alpha\beta$, $(\alpha|\beta)$ sowie $(\alpha)^*$ reguläre Ausdrücke sind und somit die beschriebenen Sprachen regulär sind.

(iii): Der Abschluss unter der Operation Komplementbildung wird folgendermaßen nachgewiesen. Sei L eine reguläre Sprache. Dann existiert ein DEA $A = (Z, \Sigma, \delta, z_0, E)$ mit $T(A) = L$. Wir konstruieren den DEA $A' = (Z, \Sigma, \delta, z_0, \Sigma \setminus E)$. Offensichtlich gilt dann $T(A') = \overline{L}$, d. h., durch das Vertauschen von akzeptierenden und nichtakzeptierenden Zuständen akzeptiert A' genau die Wörter, die A nicht akzeptiert hat, also das Komplemt von L.

(ii): Offensichtlich gilt

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

(folgt direkt aus Gleichung 1.42 auf Seite 11), damit ist durch den Abschluss unter Komplement und Vereinigung auch der Abschluss unter Durchschnitt gegeben.

Der Abschluss unter Durchschnittsbildung kann allerdings auch direkt durch Konstruktion des sogenannten *Produktautomaten* nachgewiesen werden (siehe z. B. [13]). □

4.3.6 Wortproblem und andere Entscheidbarkeitsprobleme

Nach Satz 4.21 ist das Wortproblem für Typ-3-Grammatiken *entscheidbar*, dann natürlich *auch für deterministische endliche Automaten*, denn wir können zu einem deterministischen endlichen Automaten eine äquivalente Typ-3-Grammatik konstruieren. Allerdings weist der Algorithmus, der im Beweis des Satzes 4.21 benutzt wurde, eine exponentielle Laufzeit aus, für praktische Anwendungen nicht brauchbar.

Wir betrachten jetzt das Wortproblem für DEA.

Gegeben: DEA $A = (Z, \Sigma, \delta, z_0, E)$, und Wort $w \in \Sigma^*$,

Frage: Gilt $w \in T(A)$?

Dann gilt.

Satz 4.56 *Das Wortproblem für deterministische endliche Automaten ist in linearer Zeit entscheidbar.*

Beweis. Der Beweis ist trivial: Nach dem „Einlesen“ des Wortes w wissen wir, ob $w \in T(A)$ gilt oder nicht (je nachdem ob ein akzeptierender Zustand erreicht wurde oder nicht). Und das Einlesen benötigt genau n Schritte, falls das Wort w die Länge n hat. □

Man betrachtet noch andere Entscheidbarkeitsprobleme für deterministische endliche Automaten.

Leerheitsproblem:

Gegeben: DEA A.

Frage: Gilt $T(A) = \emptyset$?

Endlichkeitsproblem:

Gegeben: DEA A.

Frage: Ist $T(A)$ endlich?

Schnittproblem:

Gegeben: Zwei DEA's A_1 und A_2.

Frage: Gilt $T(A_1) \cap T(A_2) = \emptyset$?

Äquivalenzproblem:

Gegeben: Zwei DEA's A_1 und A_2.

Frage: Gilt $T(A_1) = (A_2)$?

Zusammenfassend gilt folgender Satz, den wir aber hier nicht beweisen wollen.

Satz 4.57 *Das Leerheitsproblem, das Endlichkeitsproblem, das Schnittproblem sowie das Äquivalenzproblem für DEA sind entscheidbar.* □

4.4 Kontextfreie Sprachen

In diesem Kapitel wollen wir uns näher mit den kontextfreien Sprachen beschäftigen. Wir wissen aus dem vorhergehenden Kapitel, dass die Menge der regulären Sprachen zwar einfach handhabbar ist, aber leider gehören schon etwas komplexere Sprachen nicht zu den regulären Sprachen, wie folgende Beispiele zeigen.

Beispiel 4.58 Die nichtreguläre Sprache $L = \{a^n b^n \mid n \geq 1\}$ wird offensichtlich durch die kontextfreie Grammatik $G = (\{S\}, \{a, b\}, \{S \to aSb,\ S \to ab\}, S)$ erzeugt.

Beispiel 4.59 Wir betrachten die kontextfreie Grammatik

$$G = (\{S, X, C\}, \{x, c, 0, +, -, ;, :=, \neq, \text{LOOP}, \text{WHILE}, \text{DO}, \text{END}\}, P, S)$$

mit folgenden Regeln in der Menge P.

$$\begin{aligned}
&S \to S; S\\
&S \to \text{LOOP } X \text{ DO } S \text{ END}\\
&S \to \text{WHILE } X \neq 0 \text{ DO } S \text{ END}\\
&S \to X := X + C\\
&S \to X := X - C\\
&X \to x\\
&C \to c
\end{aligned}$$

Die erzeugte Sprache ist die Menge aller korrekten LOOP/WHILE-Programme. Ein korrektes Programm muss auch korrekt *geklammert* sein, nämlich die Klammerung durch die Schlüsselworte DO und END. Man kann zeigen, dass solche Sprachen nicht regulär sind.

Ein weiteres Beispiel für eine kontextfreie Sprache, die nicht regulär ist, ist die Menge der korrekt geklammerten arithmetischen Ausdrücke EXPR (siehe Beispiele 4.7 und 4.1).

4.4.1 Normalformen

Wir wollen jetzt Normalformen kontextfreier Grammatiken betrachten, das heißt Grammatiken, in denen die Regeln von sehr einfacher Natur sind.

Definition 4.60 (Chomsky Normalform) *Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ heißt in Chomsky Normalform, falls jede Regel in P von der Form*

(i) *$A \to BC$ mit $A, B, C \in V$ oder*

(ii) *$A \to a$ mit $A \in V$ und $a \in \Sigma$ oder*

(iii) *$S \to \varepsilon$, falls S auf keiner rechten Seite einer Regel vorkommt,*

ist.

Wir sprechen von einer *Normalform*, da folgender Satz gilt.

Satz 4.61 *Zu jeder kontextfreien Grammatik G kann man eine äquivalente kontextfreie Grammatik G' in Chomsky Normalform konstruieren, das heißt, es gilt dann $L(G) = L(G')$.* □

Wir wollen den Beweis hier nicht erbringen, sondern auf die Literatur verweisen. Man findet die Konstruktionen in fast jedem Lehrbuch. Ich möchte an dieser Stelle nur kurz die einzelnen Schritte skizzieren.

Wenn man eine Grammatik G hat, so macht man sie zunächst „ε-frei“, wir haben schon darauf hingewiesen. Dann beseitigt man alle „Kettenregeln“, das heißt Regeln vom Typ $A \to B$. Schließlich führt man für jedes Terminalzeichen a eine neue Regel $X_a \to a$ ein und ersetzt in allen rechten

Seiten, die länger als eins sind, alle Terminalzeichen a durch das jeweilige Nichtterminalzeichen X_a und letztlich „verkürzt“ man noch alle rechten Seiten, die mehr als zwei Nichtterminale enthalten. Ich möchte an dieser Stelle für den interessierten Leser ein Beispiel für solch eine Konstruktion angeben.

Beispiel 4.62 Wir betrachten die Grammatik $G = (\{S, A, B\}, \{a, b, c\}, P, S)$ mit folgenden Regeln in P.

$$S \to cSc \mid AB,$$
$$A \to aAb \mid ab,$$
$$B \to cBb \mid \varepsilon.$$

Das Konstruieren einer äquivalenten kontextfreien Grammatik G' in Chomsky-Normalform zu G geschieht folgendermaßen.

Das Beseitigen der ε-Regeln (also von Regeln $A \to \varepsilon$ für $A \neq S$) geschieht in zwei Teilschritten, zuerst wird eine Grammatik G_1 konstruiert, in der in keiner rechten Seite einer Regel das Startsymbol S auftaucht: $G_1 = (\{S, S', A, B\}, \{a, b, c\}, P_1, S)$ mit folgenden Regeln in P_1:

$$S \to cS'c \mid AB, \qquad A \to aAb \mid ab,$$
$$S' \to cS'c \mid AB, \qquad B \to cBb \mid \varepsilon.$$

Dann beseitigen wir die ε-Regeln, d. h. wir konstruieren $G_2 = (\{S, S', A, B\}, \{a, b, c\}, P_2, S)$ „ohne“ ε-Regeln. Falls $Y \overset{*}{\Longrightarrow} \varepsilon$ gilt, nehmen wir für jede Regel $X \to \alpha Y \beta$ mit $\alpha\beta \neq \varepsilon$ auch die Regel $X \to \alpha\beta$ hinzu. Konkret erhalten wir:

$$S \to cS'c \mid AB \mid A, \qquad A \to aAb \mid ab,$$
$$S' \to cS'c \mid AB \mid A, \qquad B \to cBb \mid cb.$$

Jetzt beseitigt man die Kettenregeln. Falls eine Regel $X \to Y$ wegfällt, und $Y \to \alpha$ existiert, gibt es Ableitungen $X \Longrightarrow Y \Longrightarrow \alpha$, die natürlich nicht ersatzlos wegfallen dürfen. Deshalb führen wir die direkte Regel $X \to \alpha$ ein. Konkret für unser Beispiel erhalten wir $G_3 = (\{S, S', A, B\}, \{a, b, c\}, P_3, S)$, wobei P_3 aus den Regeln

$$S \to cS'c \mid AB \mid aAb \mid ab, \qquad A \to aAb \mid ab,$$
$$S' \to cS'c \mid AB \mid aAb \mid ab, \qquad B \to cBb \mid cb$$

besteht.

Dann wird sichergestellt, dass die rechten Seiten der Regeln entweder genau aus einem Terminal oder aber aus einem nonterminalen Wort bestehen. Dabei ergibt sich die Grammatik $G_4 = (\{S, S', A, B, X_a, X_b, X_c\}, \{a, b, c\}, P_4, S)$, wobei in P_4 genau die Regeln

$$S \to X_cS'X_c \mid AB \mid X_aAX_b \mid X_aX_b, \qquad X_a \to a,$$
$$S' \to X_cS'X_c \mid AB \mid X_aAX_b \mid X_aX_b, \qquad X_b \to b,$$
$$A \to X_aAX_b \mid X_aX_b, \qquad X_c \to c,$$
$$B \to X_cBX_b \mid X_cX_b$$

enthalten sind.

Im letzten Schritt werden jetzt die nonterminalen Wörter auf den rechten Seiten der Regeln „verkürzt“. Damit ist $G' = (\{S, S', A, B, X_a, X_b, X_c, D_1, D_2, D_3, D_4, D_5, D_6\}, \{a, b, c\}, P', S)$, wo-

bei in P' genau die Regeln

$$\begin{aligned}
S &\to X_cD_1 \mid AB \mid X_aD_2 \mid X_aX_b, & D_1 &\to S'X_c,\\
S' &\to X_cD_3 \mid AB \mid X_aD_4 \mid X_aX_b, & D_2 &\to AX_b,\\
A &\to X_aD_5 \mid X_aX_b, & D_3 &\to S'X_c,\\
B &\to X_cD_6 \mid X_cX_b, & D_4 &\to AX_b,\\
X_a &\to a, & D_5 &\to AX_b,\\
X_b &\to b, & D_6 &\to BX_b,\\
X_c &\to c
\end{aligned}$$

enthalten sind.

Nachdem wir alle Schritte durchgeführt haben, ist obige Grammatik G' eine zu G äquivalente kontextfreie Grammatik in Chomsky-Normalform.

Für eine kontextfreie Grammatik in Chomsky Normalform kann man folgende Beobachtungen machen:

1. Der Syntaxbaum einer Ableitung ist ein binärer Baum (nur die Blätter hängen immer einzeln am darüberliegenden Knoten).
2. Damit kann man für die Tiefe t eines Syntaxbaumes für ein Wort w abschätzen: $\log_2 |w| < t$.
3. Jede Ableitung für ein Wort w der Sprache besteht aus genau $2 \cdot |w| - 1$ Schritten.

Wir möchten noch eine weitere Normalform für kontextfreie Sprachen, nämlich die GREIBACH[10] *Normalform*, nennen.

Definition 4.63 (Greibach Normalform) *Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ heißt in Greibach Normalform, falls jede Regel in P von der Form*

(i) *$A \to aB_1B_2 \dots B_n$ mit $n \geq 0$, $A, B_1, \dots, B_n \in V$ und $a \in \Sigma$ oder*

(ii) *$S \to \varepsilon$, falls S auf keiner rechten Seite einer Regel vorkommt,*

ist.

Auch hier gilt folgender Satz, den wir ohne Beweis angeben wollen.

Satz 4.64 *Zu jeder kontextfreien Grammatik G kann man eine äquivalente kontextfreie Grammatik G' in Greibach Normalform konstruieren, das heißt, es gilt dann $L(G) = L(G')$.* □

Wir bemerken, dass die Greibach Normalform in gewisser Weise eine Verallgemeinerung der regulären Grammatik darstellt und dass man noch weiter $0 \leq n \leq 2$ fordern kann.

4.4.2 Das Pumping Lemma

In Analogie zum Pumping Lemma für reguläre Sprachen kann man ein Pumping Lemma für kontextfreie Sprachen aufstellen.

Satz 4.65 (Pumping Lemma) *Sei L eine kontextfreie Sprache. Dann gibt es eine Konstante $k \in \mathbb{N}$, so dass für alle Wörter $z \in L$ mit $|z| \geq k$ Wörter u, v, w, x und y existieren, so dass gilt:*

(i) *$z = uvwxy$,*

(ii) *$|vwx| \leq k$ und $|vx| \geq 1$,*

(iii) *für alle $i \in \mathbb{N}$ gilt $uv^iwx^iy \in L$.* □

[10] S. A. GREIBACH, amerikanische Mathematikerin

Wir wollen keinen Beweis angeben. Mit dem Pumping Lemma für kontextfreie Sprachen kann man beweisen, dass Sprachen *nicht* kontextfrei sind, zum Beispiel die Sprachen

$L = \{a^{n^2} \mid n \geq 1\}$,
$L = \{a^{2^n} \mid n \geq 1\}$,
$L = \{a^p \mid p \text{ Primzahl}\}$,
$L = \{a^n b^n c^n \mid n \geq 1\}$,
$L = \{ww \mid w \in \{a, b\}^*\}$.

Ich möchte auch hier darauf hinweisen, dass das Pumping Lemma eine notwendige Eigenschaft für kontextfreie Sprachen formuliert. Also auch hier gilt in Analogie zum regulären Fall:

> **Mit dem Pumping Lemma für kontextfreie Sprachen kann man nicht beweisen, dass eine Sprache kontextfrei ist; sondern nur, dass eine Sprache nicht kontextfrei ist!**

4.4.3 Abschlusseigenschaften

Auch die Menge der kontextfreien Sprachen wollen wir wieder auf Abschlusseigenschaften hin untersuchen.

Satz 4.66 *Die Menge der kontextfreien Sprachen ist unter den Operationen*

(i) *Vereinigung,*

(ii) *Produkt (Konkatenation) und*

(iii) *Kleene-Stern*

abgeschlossen. Sie ist unter den Operationen

(iv) *Durchschnitt und*

(v) *Komplement*

nicht abgeschlossen.

Beweis. Wir wollen für die positiven Aussagen ohne weitere Bemerkungen nur die Konstruktionen angeben. Es seien zwei kontextfreie Grammatiken $G_1 = (V_1, \Sigma, P_1, S_1)$ und $G_2 = (V_2, \Sigma, P_2, S_2)$ für zwei kontextfreie Sprachen L_1 und L_2 gegeben.

(i) Wir konstruieren $G = (V, \Sigma, P, S)$ durch

$V = V_1 \cup V_2 \cup \{S\}$,
$P = P_1 \cup P_2 \cup \{S \to S_1,\, S \to S_2\}$.

Offensichtlich ist dann auch G kontextfrei und es gilt $L(G) = L_1 \cup L_2$.

(ii) Wir konstruieren $G' = (V', \Sigma, P', S')$ durch

$V' = V_1 \cup V_2 \cup \{S'\}$,
$P' = P_1 \cup P_2 \cup \{S' \to S_1 S_2\}$.

Offensichtlich ist dann auch G kontextfrei und es gilt $L(G) = L_1 \cdot L_2$.

(iii) Wir konstruieren $G'' = (V'', \Sigma, P'', S'')$ durch

$V'' = V_1 \cup \{S''\}$,
$P'' = P_1 \cup \{S'' \to \varepsilon,\, S'' \to S_1,\, S_1 \to S_1 S_1\}$.

Offensichtlich ist dann auch G kontextfrei und es gilt $L(G) = L_1^*$.

(iv) Kommen wir jetzt zum Beweis, dass die Menge der kontextfreien Sprachen nicht unter Durchschnittbildung abgeschlossen ist. Dazu reicht die Angabe eines Gegenbeispiels. Sei

$L_1 = \{a^n b^n c^m \mid m, n \geq 1\}$ und $L_2 = \{a^n b^m c^m \mid m, n \geq 1\}$. Offensichtlich sind L_1 und L_2 kontextfreie Sprachen (siehe Übungsaufgaben oder Beispiel 4.68). Der Durchschnitt beider Mengen $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 1\}$ ist aber nicht kontextfrei.

(v) Zum Schluss der Nachweis der Nichtabgeschlossenheit der Menge der kontextfreien Sprachen unter Komplementbildung. Wir werden den Nachweis indirekt führen. Also Annahme: die Menge der kontextfreien Sprachen ist unter Komplement abgeschlossen. Dann gilt gemäß der Gleichung (1.42) für den Durchschnitt zweier Mengen

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}},$$

also wäre mit den kontextfreien Sprachen L_1 und L_2 wegen (i) und unserer Annahme auch $L_1 \cap L_2$ kontextfrei, was einen Widerspruch zur obigen Aussage (iv) darstellt. Damit ist unsere Annahme falsch, die Aussage (v) bewiesen und der Beweis des Satzes somit komplett. □

4.4.4 Entscheidbarkeit und der Algorithmus von Cocke, Younger und Kasami

Das Wortproblem für Typ-1-Sprachen ist entscheidbar, ein entsprechender Algorithmus benötigt aber exponentiellen Zeitaufwand. Für kontextfreie Sprachen können wir einen Algorithmus angeben, der das Wortproblem effizienter löst. Allerdings muss dazu die gegebene Grammatik in Chomsky Normalform vorliegen.

Eingabe: Grammatik $G = (V, \Sigma, P, S)$ in Chomsky Normalform und Wort $x = a_1 a_2 \dots a_n$
Ausgabe: **JA**, falls $x \in L(G)$; **Nein**, sonst
`FOR` $i := 1$ `TO` n `DO`
  $T[i,1] := \{A \mid A \to a_i \in P\}$
`END;`
`FOR` $j := 2$ `TO` n `DO`
  `FOR` $i := 1$ `TO` $n + 1 - j$ `DO`
    $T[i,j] := \emptyset$;
    `FOR` $k := 1$ `TO` $j - 1$ `DO`
      $T[i,j] := T[i,j] \cup \{A \mid A \to BC \in P,\ B \in T[i,k],\ C \in T[i+k, j-k]\}$
    `END`
  `END`
`END;`
`IF` $S \in T[1,n]$ `THEN`
  `return JA`
`ELSE`
  `return NEIN`
`END`

Abbildung 4.20: Der Algorithmus von Cocke, Younger und Kasami

Satz 4.67 *Der Algorithmus von Cocke, Younger und Kasami löst das Wortproblem für kontextfreie Grammatiken in Chomsky Normalform mit einem Zeitaufwand von $O(n^3)$.* □

Beweisen wollen wir den Satz nicht, eine genaue Analyse des Algorithmus liefert aber relativ schnell das gewünschte Ergebnis. Die wesentlich Beobachtung ist, dass $T[i,j]$ die Menge aller Nichtterminale ist, die das an der Stelle i beginnende Teilwort der Länge j, also $a_i a_{i+1} \cdots a_{i+j-1}$, erzeugen. Zum Zeitverhalten muss man sagen, dass natürlich auch das Herstellen der Chomsky Normalform Zeit kostet und somit beachtet werden muss, allerdings fällt der Aufwand zum Herstellen der Chomsky Normalform ja nur einmal an und ist nur abhängig von der Größe der Grammatik.

Wir wollen den Algorithmus von Cocke, Younger und Kasami an einem Beispiel anwenden.

Beispiel 4.68 Die Sprache $L = \{a^n b^n c^m \mid m, n \geq 1\}$ ist kontextfrei und wird von der Grammatik $G = (V, \Sigma, P, S)$ mit den Regeln

$$S \to AB,$$
$$A \to aAb \mid ab,$$
$$B \to cB \mid c$$

erzeugt. Eine kontextfreie Grammatik in Chomsky Normalform für L hätte folgende Menge von Regeln:

$$S \to AB, \qquad C \to a,$$
$$A \to CD \mid CF, \qquad D \to b,$$
$$B \to EB \mid c, \qquad E \to c,$$
$$F \to AD.$$

Sei $x = aaabbbcc$. Dann erzeugt der Algorithmus die Tabelle in Abbildung 4.21.

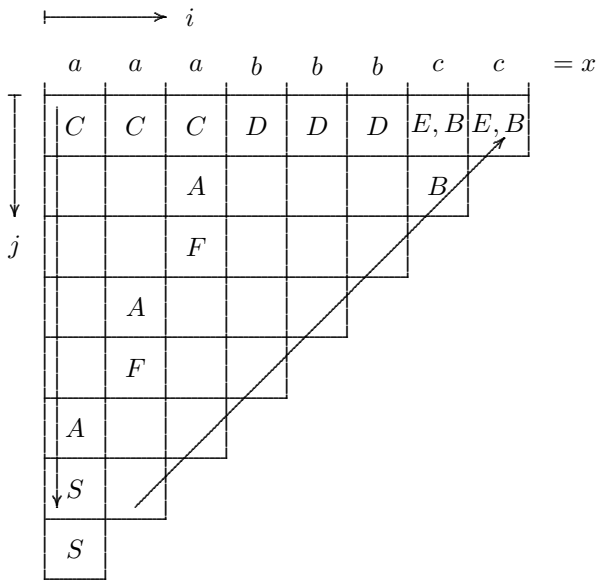

Abbildung 4.21: Tabelle zum Algorithmus von CYK

In der Tabelle stellt jede Zelle eine Menge $T[i, j]$ von Nichtterminalen dar. Die Koordinatenachsen für i und j sind eingezeichnet.

Der Algorithmus besteht aus drei Teilen. Im ersten Teil wird die oberste Zeile ($T[i, 1]$) berechnet.

Im zweiten Teil werden zeilenweise die weiteren Mengen $T[i, j]$ berechnet, indem immer die darüberliegende Spalte von oben und die nach rechts oben führende Diagonale betrachtet werden. Die Menge $T[1, 8]$ würde dann zum Beispiel folgendermaßen ermittelt werden (zur besseren Lesbarkeit wurden die Variablen A, B, C hier mit X, Y, Z bezeichnet, um sie von den Nichterminalzeichen

abzugrenzen). Leere Zellen stehen dabei für die leere Menge.

$$\begin{aligned}
T[1,8] &= \{X \mid X \to YZ \in P,\ Y \in T[1,1],\ Z \in T[2,7]\} && \text{(für } k=1\text{)}\\
&\quad \cup \{X \mid X \to YZ \in P,\ Y \in T[1,2],\ Z \in T[3,6]\} && \text{(für } k=2\text{)}\\
&\quad \cup \{X \mid X \to YZ \in P,\ Y \in T[1,3],\ Z \in T[4,5]\} && \text{(für } k=3\text{)}\\
&\quad \cup \{X \mid X \to YZ \in P,\ Y \in T[1,4],\ Z \in T[5,4]\} && \text{(für } k=4\text{)}\\
&\quad \cup \{X \mid X \to YZ \in P,\ Y \in T[1,5],\ Z \in T[6,3]\} && \text{(für } k=5\text{)}\\
&\quad \cup \{X \mid X \to YZ \in P,\ Y \in T[1,6],\ Z \in T[7,2]\} && \text{(für } k=6\text{)}\\
&\quad \cup \{X \mid X \to YZ \in P,\ Y \in T[1,7],\ Z \in T[8,1]\} && \text{(für } k=7\text{)}\\
&= \{X \mid X \to YZ \in P,\ Y \in \{C\},\ Z \in \emptyset\}\\
&\quad \cup \{X \mid X \to YZ \in P,\ Y \in \emptyset,\ Z \in \emptyset\}\\
&\quad \cup \{X \mid X \to YZ \in P,\ Y \in \emptyset,\ Z \in \emptyset\}\\
&\quad \cup \{X \mid X \to YZ \in P,\ Y \in \emptyset,\ Z \in \emptyset\}\\
&\quad \cup \{X \mid X \to YZ \in P,\ Y \in \emptyset,\ Z \in \emptyset\}\\
&\quad \cup \{X \mid X \to YZ \in P,\ Y \in \{A\},\ Z \in \{B\}\}\\
&\quad \cup \{X \mid X \to YZ \in P,\ Y \in \{S\},\ Z \in \{E,B\}\}\\
&= \emptyset \cup \emptyset \cup \emptyset \cup \emptyset \cup \emptyset \cup \{S\} \cup \emptyset\\
&= \{S\}.
\end{aligned}$$

Im dritten Teil des Algorithmus schließlich wird sich die Menge $T[1,n]$ angeschaut, hier $T[1,8]$. Gilt $S \in T[1,n]$, dann und nur dann gibt der Algorithmus „ja“ aus, das heißt, es gibt eine Ableitung $S \overset{*}{\Longrightarrow} x$, hier $S \overset{*}{\Longrightarrow} aaabbbcc$, somit erzeugt die gegebene Grammatik $x = aaabbbcc$.

Noch eine Bemerkung: aus der aufgestellten Tabelle kann man auch die Ableitung für das Wort x ablesen, indem wir rückwärts die Regeln anwenden, die auf S in $T[1,n]$ geführt haben. Hier sieht die Ableitung für *aaabbbc* dann folgendermaßen aus (die Nonterminale, die ersetzt werden, sind jeweils unterstrichen).

$$\begin{aligned}
&\underline{S} \Longrightarrow \underline{A}B \Longrightarrow C\underline{F}B \Longrightarrow C\underline{A}DB \Longrightarrow CC\underline{F}DB \Longrightarrow CC\underline{A}DDB \Longrightarrow \underline{C}CCDDDB\\
&\quad \Longrightarrow a\underline{C}CDDDB \Longrightarrow aa\underline{C}DDDB \Longrightarrow aaa\underline{D}DDB \Longrightarrow aaab\underline{D}DB \Longrightarrow aaabb\underline{D}B\\
&\quad \Longrightarrow aaabbb\underline{B} \Longrightarrow aaabbbc
\end{aligned}$$

Für die anderen Entscheidbarkeitsprobleme, die wir noch im Kapitel 4.3.6 für die Menge der regulären Sprachen formuliert haben kann man folgenden Satz angeben.

Satz 4.69 *Das Leerheitsproblem und das Endlichkeitsproblem für kontextfreie Grammatiken sind entscheidbar. Das Schnittproblem und das Äquivalenzproblem für kontextfreie Grammatiken sind unentscheidbar.* □

Wir wollen den Satz nicht beweisen, möchten nur bemerken, dass die Unentscheidbarkeit des Äquivalenzproblems für die hohe Komplexität schon der kontextfreien Grammatiken spricht.

4.4.5 Kellerautomaten

Wir wollen jetzt versuchen, ein Automatenmodell für die kontextfreien Grammatiken zu finden. Der Mangel der endlichen Automaten ist, dass sie im Prinzip nicht „zählen“ können, sie können sich nur etwas im Zustand „merken“, aber es gibt nur endlich viele Zustände. Deshalb können sie zum Beispiel die Sprache

$$\begin{aligned}
L &= \{wcw^R \mid w \in \{a,b\}^*\}\\
&= \{a_1a_2\ldots a_nca_n\ldots a_2a_1 \mid n \geq 0,\ a_i \in \{a,b\},\ 1 \leq i \leq n\}
\end{aligned}$$

nicht akzeptieren.

Wir führen jetzt das Modell des *Kellerautomaten* ein, indem wir den endlichen Automaten mit einem zusätzlichen Speicher ausstatten, dem sogenannten *Keller*. Der *Kellerautomat* (englisch *pushdown automaton*, deshalb mit PDA abgekürzt) kann im Speicher schreiben und natürlich lesen, wobei die Überführungen nun auch vom gelesenen Kellersymbol abhängen. Der Keller funktioniert nach dem Prinzip „Last In First Out“ (LIFO), das heißt, er kann nur immer das oberste Symbol lesen. Wir statten den Keller noch mit einem Anfangssymbol aus (#). Zusätzlich erlauben wir auch noch sogenannte „ε-Übergänge“, das heißt der Automat „liest“ auf dem Eingabeband ein ε, also nichts und kann so im Keller „rechnen“, ohne die Eingabe zu verarbeiten. Der Kellerautomat wird als Standardversion immer als *nichtdeterministischer Automat* definiert. Formal sieht die Definition wie folgt aus.

Definition 4.70 (Kellerautomat) *Ein (nichtdeterministischer) Kellerautomat (mit PDA abgekürzt von pushdown automaton) M ist ein 6-Tupel $M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$. Dabei ist*

- *Z das Zustandsalphabet,*
- *Σ das Eingabealphabet,*
- *Γ das Kelleralphabet,*
- *$z_0 \in Z$ der Anfangszustand,*
- *$\delta\colon Z \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2_E^{Z \times (\Gamma \setminus \{\#\})^*}$ die Zustandsüberführungsfunktion (2_E^X bedeutet dabei die Menge der endlichen Teilmengen von X),*
- *$\# \in \Gamma$ das Kelleranfangssymbol.*

Die Arbeitsweise wird folgendermaßen interpretiert:

- Bei $(z', B_1 B_2 \dots B_k) \in \delta(z, a, A)$ mit $a \in \Sigma$ befindet sich der Automat im Zustand z, liest auf dem Eingabeband ein a, liest im Keller ein A (das oberste Symbol) und entfernt dieses A aus dem Keller, geht in den Zustand z' über, bewegt den Lesekopf auf dem Eingabeband einen Schritt nach rechts und schreibt auf den Keller das Wort $B_1 B_2 \dots B_k$ derart, dass B_1 das neue oberste Symbol des Kellers ist.
- Bei $(z', B_1 B_2 \dots B_k) \in \delta(z, \varepsilon, A)$ liest er im Gegensatz zur obigen Aktion auf dem Eingabeband nichts ein und bewegt also seinen Lesekopf auf dem Eingabeband nicht.
- Der Kellerautomat beginnt seine Arbeit immer im Zustand z_0 und mit dem Kellerinhalt #.
- Er akzeptiert die zu Beginn der Arbeit auf dem Eingabeband stehende Zeichenkette, wenn er diese Eingabe vollständig eingelesen hat und der Keller leer ist (auch das Kelleranfangssymbol # steht nicht mehr im Keller).

Formal führen wir wie bei den Turingmaschinen *Konfigurationen* zur augenblicklichen Situationsbeschreibung ein.

Definition 4.71 (Konfiguration eines PDA) *Sei $M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$ ein Kellerautomat, unter einer Konfiguration k verstehen wir ein Tripel $k \in Z \times \Sigma^* \times \Gamma^*$.*

Anschaulich beschreibt eine Konfiguration $k = (z, v, \gamma)$ folgende augenblickliche Situation des PDA: er befindet sich im Zustand z, v ist die noch nicht verarbeitete Eingabe auf dem Eingabeband und im Keller steht das Wort γ, wobei das erste Symbol von γ das oberste Kellersymbol sein soll.

In der Menge der Konfigurationen eines Kellerautomaten können wir jetzt die Überführungsrelation $\vdash$ einführen, wobei

$$k_1 \vdash k_2$$

bedeutet, der Kellerautomat überführt die Konfiguration k_1 in genau einem Schritt in die Konfiguration k_2; auf die formale Definition verzichten wir hier, zur Interpretation siehe oben und siehe Abbildung 4.22.

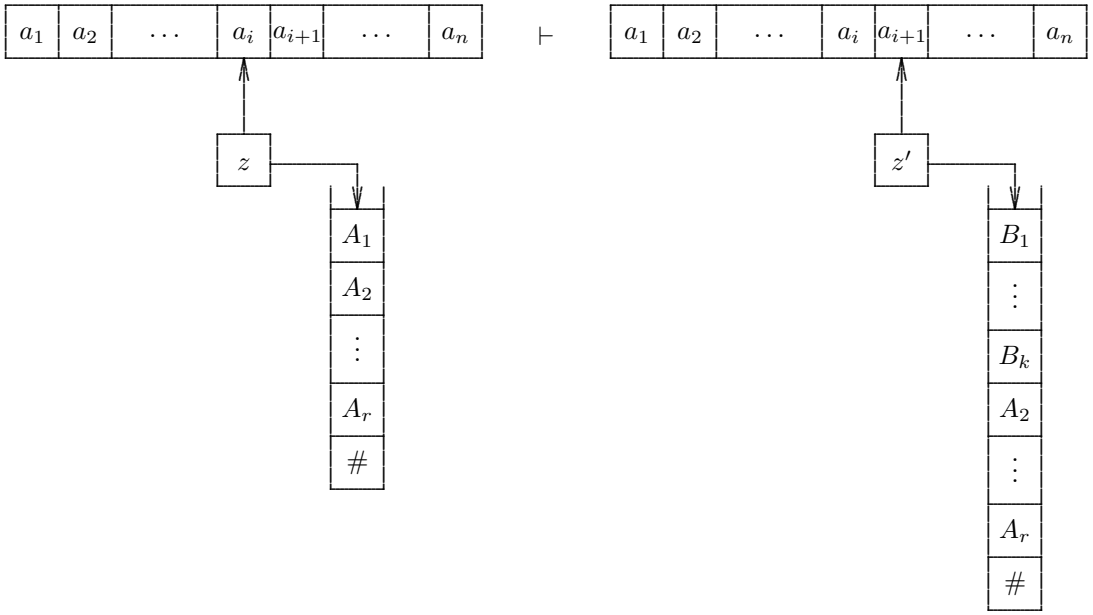

Abbildung 4.22: Interpretation der Überführung $(z', B_1 \dots B_k) \in \delta(z, a_i, A_1)$ eines Kellerautomaten

Unter $k_1 \vdash^* k_2$ wollen wir wieder verstehen, dass der PDA k_1 in endlich vielen Schritten (auch null) in k_2 überführt.

Dann können wir die von einem Kellerautomaten akzeptierte Sprache definieren.

Definition 4.72 (Akzeptierte Sprache eines PDA) *Für einen PDA* $M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$ *sei die von ihm akzeptierte Sprache* $T(M)$ *durch*

$$T(M) = \{w \in \Sigma^* \mid (z_0, w, \#) \vdash^* (z, \varepsilon, \varepsilon) \text{ für ein } z \in Z\}$$

definiert

Es ist Zeit für ein Beispiel.

Beispiel 4.73 Wir geben einen Kellerautomaten für die Sprache

$$L = \{wcw^R \mid w \in \{a, b\}^*\} = \{a_1 a_2 \dots a_n c a_n \dots a_2 a_1 \mid n \geq 0,\ a_i \in \{a, b\},\ 1 \leq i \leq n\}$$

an. Sei

$$M = (\{z_0, z_1\}, \{a, b, c\}, \{A, B, \#\}, \delta, z_0, \#)$$

ein Kellerautomat, wobei δ durch

$$\begin{aligned}
&\delta(z_0, a, X) = \{(z_0, AX)\} \qquad \text{für } X \in \{A, B, \#\},\\
&\delta(z_0, b, X) = \{(z_0, BX)\} \qquad \text{für } X \in \{A, B, \#\},\\
&\delta(z_0, c, X) = \{(z_1, X)\} \qquad \text{für } X \in \{A, B, \#\},\\
&\delta(z_1, a, A) = \{(z_1, \varepsilon)\},\\
&\delta(z_1, b, B) = \{(z_1, \varepsilon)\},\\
&\delta(z_1, \varepsilon, \#) = \{(z_1, \varepsilon)\}
\end{aligned}$$

definiert ist. Dann gilt bei der Eingabe *bacab*

$$\begin{aligned}
(z_0, bacab, \#) &\vdash (z_0, acab, B\#) \vdash (z_0, cab, AB\#) \vdash (z_1, ab, AB\#) \vdash (z_1, b, B\#) \vdash (z_1, \varepsilon, \#)\\
&\vdash (z_1, \varepsilon, \varepsilon),
\end{aligned}$$

also $bacab \in T(M)$.

Folgenden Satz wollen wir ohne Beweis angeben.

Satz 4.74 *Eine Sprache L ist kontextfrei genau dann, wenn ein (nichtdeterministischer) Kellerautomat M mit $T(M) = L$ existiert.* □

Wenn man sich den Kellerautomat im Beispiel 4.73 genauer anschaut, hat jede Konfiguration dieses Automaten höchstens eine mögliche Folgekonfiguration, also ist dieser PDA sogar ein deterministischer. Betrachten wir ein Beispiel für einen echt nichtdeterministischen Kellerautomaten.

Beispiel 4.75 Sei

$$M = (\{z_0, z_1\}, \{a, b\}, \{A, B, \#\}, \delta, z_0, \#)$$

ein Kellerautomat, mit

$$\begin{aligned}
&\delta(z_0, a, X) = \{(z_0, AX)\} \qquad \text{für } X \in \{B, \#\},\\
&\delta(z_0, a, A) = \{(z_1, AA), (z_1, \varepsilon)\},\\
&\delta(z_0, b, X) = \{(z_0, BX)\} \qquad \text{für } X \in \{A, \#\},\\
&\delta(z_0, b, B) = \{(z_0, BB), (z_1, \varepsilon)\},\\
&\delta(z_0, \varepsilon, \#) = \{(z_1, \varepsilon)\},\\
&\delta(z_1, a, A) = \{(z_1, \varepsilon)\},\\
&\delta(z_1, b, B) = \{(z_1, \varepsilon)\},\\
&\delta(z_1, \varepsilon, \#) = \{(z_1, \varepsilon)\},
\end{aligned}$$

dann gilt offensichtlich

$$T(M) = \{ww^R \mid w \in \{a, b\}^*\} = \{a_1 a_2 \dots a_n a_n \dots a_2 a_1 \mid n \geq 0,\ a_i \in \{a, b\},\ 1 \leq i \leq n\}.$$

Man kann zeigen, dass für die Sprache $\{ww^R \mid w \in \{a, b\}^*\}$ *kein* deterministischer Kellerautomat existiert. Die von deterministischen Kellerautomaten akzeptierten Sprachen heißen deterministisch kontextfrei. Es gilt dann folgender Satz.

Satz 4.76 *Die Menge der deterministisch kontextfreien Sprachen ist eine echte Teilmenge der Menge der kontextfreien Sprachen.*

Die deterministisch kontextfreien Sprachen spielen eine wichtige Rolle im Compilerbau. Sie werden durch sogenannte $LR(k)$-Grammatiken beschrieben, die ausreichen, um die Syntax von Programmiersprachen zu beschreiben. Da das Wortproblem für deterministisch kontextfreie Sprachen in linearer Zeit entscheidbar ist, ist somit die Syntaxüberprüfung von Programmen auch in linearer Zeit möglich, ein wesentlicher Vorteil gegenüber der $O(n^3)$-Laufzeit des CYK-Algorithmus, der das Wortproblem für kontextfreie Sprachen entscheidet.

4.5 Rekursiv aufzählbare und kontextabhängige Sprachen

Schließlich sollen noch Automatencharakterisierungen für die beiden umfangreichsten Familien von Sprachen in der Chomsky-Hierarchie angegeben werden. Dabei kommen wir auf ein bekanntes und bereits ausführlich diskutiertes Modell zurück: die Turingmaschine.

Satz 4.77 *Eine Sprache L ist genau dann rekursiv aufzählbar (d. h. vom Typ 0), wenn es eine (nichtdeterministische) Turingmaschine M gibt, die sie akzeptiert, also für die $T(M) = L$ gilt.*

Beweis. Die Beweisidee soll hier nur kurz angedeutet werden. Zu einer Grammatik $G = (V, \Sigma, P, S)$ konstruiert man eine nichtdeterministische Turingmaschine M mit folgender Arbeitsweise. Als Eingabe erhält M ein Wort $w \in \Sigma^*$. Solange auf dem Band von M nicht das Wort S steht, werden nichtdeterministisch eine Regel $\alpha to \beta$ aus P sowie ein Vorkommen von β im aktuellen Bandwort (sofern ein solches existiert) gewählt, das dann durch α ersetzt wird. Es ist offensichtlich, dass ein akzeptierender Lauf von M einer Ableitung bezüglich G in umgekehrter Reihenfolge entspricht.

Umgekehrt sei $M = (Z, \Sigma, \Gamma, \delta, z_0, \Box, E)$ eine NTM. Wir konstruieren die Grammatik $G = (V, \Sigma \cup \{\$, \&\}, P, S)$ mit dem Nichtterminal-Aphabet $V = Z \cup (\Gamma \setminus \Sigma) \cup \{S, A, B\}$. Die Regelmenge P besteht aus folgenden Regeln:

- $S \to \$A, A \to xA, A \to qB, B \to xB, B \to \$$ für alle $x \in \Gamma, q \in E$.
- $bz' \to za$, falls $(z', b, R) \in \delta(z, a)$,
  $z'b \to za$, falls $(z', b, N) \in \delta(z, a)$,
  $z'xb \to xza$, falls $(z', b, R) \in \delta(z, a), x \in \Gamma$.
- $\$\Box \to \$, \Box\$ \to \$$.
- $\$z_0 \to \$\&$.

Mit Hilfe der Regeln der ersten Art erzeugt man ein Wort der Form $\$k\$$, wobei k eine beliebige Endkonfiguration von M ist. Die Regeln der zweiten Art sind so konstruiert, dass $\$k'\$ \Leftarrow_G \$k\$$ genau dann gilt, wenn k' bezüglich M eine Nachfolgekonfiguration von k ist. Durch die Regeln der dritten Art lassen sich führende und abschließende Blanksymbole der Konfiguration entfernen. Schließlich kann man durch Anwendung der letzten Regel ein Wort $\$z_0w\$$ mit $w \in \Sigma^*$ in das Terminalwort $\$\&w\$$ überführen. Nach unserer Konstruktion kann man das Wort $\$\&w\$$ genau dann in G erzeugen, wenn w von M akzeptiert wird. Mit einigen weiteren technischen Änderungen kann man aus G eine Grammatik G' erhalten, so dass $L(G') = T(M)$ gilt. □

Der Unterschied zwischen kontextabhängigen und allgemeinen Grammatiken besteht nur darin, dass bei kontextabhängigen Grammatiken keine verkürzenden Regeln zugelassen sind. Damit sind die in einer Ableitung eines Terminalwortes w auftretenden Satzformen in ihrer Länge durch $|w|$ beschränkt.

Vollzieht man den ersten Teil des Beweises von Satz 4.77 mit einer kontextabhängigen Grammatik nach, so entsteht eine nichtdeterministische Turingmaschine, die nur auf den Zellen der Eingabe sowie zwei zusätzlichen Randzellen links und rechts der Eingabe arbeitet. Eine derart beschränkte NTM heißt *linear beschränkter Automat* (kurz LBA). Auf eine formale Definition des linear beschränkten Automaten wollen hier verzichten.

Führt man umgekehrt die Konstruktion des zweiten Teils des Beweises mit einem LBA durch, so kann man auf die verkürzenden Regeln $\$\Box \to \$$ und $\Box\$ \to \$$ verzichten. Es entsteht also eine Grammatik vom Typ 1 (wobei einige technische Details wiederum noch zu klären wären). Wir können deshalb formulieren:

Satz 4.78 *Eine Sprache L ist genau dann kontextabhängig (d. h. vom Typ 1), wenn es einen (nichtdeterministischen) linear beschränkten Automaten M gibt, die sie akzeptiert, also für den $T(M) = L$ gilt.*

Die Frage nach der Äquivalenz von deterministischen und nichtdeterministischen LBA muss leider offen bleiben; sie ist bis heute nicht beantwortet und ist in der Literatur unter dem Namen *LBA-Problem* bekannt.

4.6 Tabellarischer Überblick

Hier wollen wir die wichtigsten Ergebnisse des Kapitels 4 in tabellarischer Form zusammenfassen. Dabei tauchen in den Tabellen auch Ergebnisse auf, die wir vorher noch nicht explizit genannt, geschweige denn bewiesen haben.

Zunächst betrachten wir die Sprachfamilien der Chomsky Hierarchie (zusätzlich wurden die deterministisch kontextfreien Sprachen aufgenommen, die nicht zur eigentlichen Chomsky Hierarchie zählen) und deren *Charakterisierungsmöglichkeiten*.

Sprache	Grammatik	Automat	Andere
Regulär	Typ 3	Endlicher Automat (EA)	Regulärer Ausdruck
Deterministisch kontextfrei	$LR(k)$	Deterministischer Kellerautomat (DPDA)	
Kontextfrei	Typ 2	(Nichtdeterministischer) Kellerautomat (PDA)	
Kontextabhängig	Typ 1	(Nichtdeterministischer) Linear beschränkter Automat (LBA)	
Rekursiv aufzählbar	Typ 0	Turingmaschine (TM)	

Tabelle 4.2: Charakterisierungen der Sprachfamilien der Chomsky Hierarchie

In der Tabelle 4.2 werden teilweise die Automatentypen hinsichtlich Determinismus spezifiziert, teilweise nicht. Einen Überblick über die Problematik des Verhältnisses von *deterministischen und nichtdeterministischen Varianten* von Automaten gibt die Tabelle 4.3. Die offene Frage der Äquivalenz von deterministischen und nichtdeterministischen linear beschränkten Automaten wird als sogenanntes LBA-Problem bezeichnet.

Nichtdeterminismus	Determinismus	Äquivalenz
NEA	DEA	ja
PDA	DPDA	nein
LBA	DLBA	?
NTM	DTM	ja

Tabelle 4.3: Beziehungen zwischen deterministischen und nichtdeterministischen Automaten

Einen Überblick über die *Abschlusseigenschaften* der Sprachen der Chomsky Hierarchie (erweitert um die deterministisch kontextfreien Sprachen) bezüglich ausgewählter Operationen liefert Tabelle 4.4.

	Typ 3	Det. kf.	Typ 2	Typ 1	Typ 0
Vereinigung	ja	nein	ja	ja	ja
Durchschnitt	ja	nein	nein	ja	ja
Komplement	ja	ja	nein	ja	nein
Konkatenation	ja	nein	ja	ja	ja
Kleene-Stern	ja	nein	ja	ja	ja

Tabelle 4.4: Abschlusseigenschaften

Ausgewählte *Entscheidbarkeitsprobleme* der Sprachen der Chomsky Hierarchie werden in der Tabelle 4.5 betrachtet, wobei „ja“ für entscheidbar und „nein“ für unentscheidbar steht.

	Typ 3	Det. kf.	Typ 2	Typ 1	Typ 0
Wortproblem	ja	ja	ja	ja	nein
Leerheitsproblem	ja	ja	ja	nein	nein
Schnittproblem	ja	nein	nein	nein	nein
Äquivalenzproblem	ja	ja	nein	nein	nein

Tabelle 4.5: Entscheidbarkeitsprobleme

In der Tabelle 4.6 sind die Ergebnisse über die *Zeitkomplexität des Wortproblems* für die Sprachfamilien der Chomsky Hierarchie zusammengefasst, wobei Erwähnung finden sollte, dass man das Wortproblem für kontextfreie Sprachen, gegeben durch kontextfreie Grammatiken in Chomsky Normalform (CNF), durch verbesserte Algorithmen heute bereits in einer Zeitkomplexität von $O(n^{2,38})$ lösen kann. Für das Wortproblem der Typ-1-Sprachen ist bis heute kein Algorithmus mit polynomialem Zeitverhalten bekannt.

Sprache	Komplexität
Typ 3 (DEA gegeben)	linear
Deterministisch kontextfrei	linear
Typ 2 (CNF gegeben)	$O(n^3)$
Typ 1	exponentiell
Typ 0	unentscheidbar

Tabelle 4.6: Komplexität des Wortproblems