MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Chomsky-Hierarchie formaler und natürlicher Sprachen“

verfasst von / submitted by

Ramona Novinić, BEd

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Education (MEd)

Wien, 2020 / Vienna 2020

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 199 503 520 02
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Lehramt Sek (AB) UF Bosnisch/Kroatisch/Serbisch UF Mathematik
Betreut von / Supervisor:	ao. Univ.-Prof. Dr. Joachim Mahnkopf

Abstract

Diese Masterarbeit beschäftigt sich mit der Chomsky-Hierarchie formaler und natürlicher Sprachen. Der US-amerikanische Linguist Noam Chomsky hat diese Klassifizierung formaler Sprachen in den 1950er Jahren angegeben, die zu einem wichtigen Konzept der angewandten Algebra und theoretischen Informatik wurde. In dieser Arbeit soll eine Verbindung zu den natürlichen Sprachen dargestellt werden. Genauer stellt sich die Frage, ob auch Aussagen zur Klassifizierung *natürlicher* Sprachen gemäß Chomsky-Hierarchie getroffen werden können. In seinem Artikel *English is not a context-free language* (1984) stellt James Higginbotham die Behauptung auf, dass Englisch keine kontextfreie Sprache ist. Es werden zunächst die notwendigen Grundbegriffe über formale Sprachen, Grammatiken und Automaten eingeführt, das Ziel dieser Masterarbeit besteht dann in der Analyse der Argumentation von Higginbothams Beweis, der sich durch mathematische Stringenz auszeichnet.

This thesis deals with the Chomsky hierarchy of formal and natural languages. The american linguist Noam Chomsky introduced this classification of formal languages in the 1950s, which later became an important concept in applied algebra and theoretical computer science. This master thesis describes a connection to natural languages. More precisely the question arises whether or not *natural* languages can be classified in accordance with the Chomsky hierarchy. In his article *English is not a context-free language* (1984) James Higginbotham claims that English, a natural language, is not context-free. The aim of this thesis is the analysis of his proof and argumentation; to this end the fundamental concepts of formal languages, grammars and automatons are initially introduced.

Inhaltsverzeichnis

1 Einleitung

Wenn man eine Überschneidung zwischen Mathematik und Sprachwissenschaft finden möchte, entdeckt man schnell den Begriff der *mathematischen Linguistik*. Dahinter verbirgt sich wenig Sprachwissenschaft, aber viel angewandte Algebra und theoretische Informatik. Eine fundamentale Idee sind dabei die formalen Sprachen, die im Gegensatz zu den natürlichen Sprachen aber nicht der menschlichen Kommunikation dienen. Formale Sprachen eignen sich zur Kommunikation mit Maschinen, und nachdem auch Programmiersprachen als formale Sprachen aufgefasst werden können, gibt es vor allem in der modernen Informatik praktische Anwendungen.

Einige formale Sprachen können mithilfe einer Phrasenstrukturgrammatik erzeugt werden, und die sogenannte Chomsky-Hierarchie ist eine Klassifizierung bestimmter Grammatiktypen. Kann eine formale Sprache mittels Grammatik erzeugt werden, handelt es sich um eine *Typ-0-Sprache*, nach weiteren Einschränkungen unterscheidet man außerdem zwischen *kontextsensitiven*, *kontextfreien* und *regulären* Sprachen bzw. Grammatiken.

Im Mittelpunkt dieser Arbeit steht die Frage, ob Englisch eine kontextfreie Sprache ist. Diese Fragestellung ist spannend, weil Kontextfreiheit eigentlich eine Eigenschaft bestimmter formaler Sprachen darstellt. Daher ist es interessant, dieses Konzept auf die *natürlichen* Sprachen ausweiten zu wollen, und zu behaupten, dass Englisch, eine natürliche Sprache, nicht kontextfrei sein kann. Das Ziel dieser Masterarbeit ist, den diesbezüglichen Beweis von James Higginbotham [7] nachzuvollziehen und zu analysieren. Der Sprachwissenschaftler und Philosoph James Higginbotham gibt einen mathematisch stringenten Beweis, dessen Hilfsmittel wir in den folgenden Kapiteln erläutern werden, bevor im fünften Kapitel der eigentliche Beweis dargestellt wird.

Dazu steht am Beginn das zweite Kapitel über formale Sprachen. Hier wird zunächst die algebraische Struktur eines Monoids untersucht und der zentrale Begriff des freien Monoids definiert. Vom algebraischen Standpunkt aus gesehen, ist eine formale Sprache einfach eine Teilmenge des freien Monoids über einer bestimmen Menge an Symbolen, die auch Alphabet genannt wird. Bezeichnungen wie Alphabet, Buchstabe, Wort und Sprache sind aus dem alltäglichen Sprachgebrauch wohlbekannt, sollen hier aber formalisiert werden. Außerdem werden zahlreiche Beispiele formaler Sprachen gegeben, die einem besseren Verständnis dienen.

Das dritte Kapitel behandelt Grammatiken. Nachdem bestimmte formale Sprachen mithilfe von Grammatiken erzeugt werden können, besteht hier ein enger Zusammenhang zu natürlichen Sprachen. Neben der formalen Definition einer Phrasenstrukturgrammatik, ist der wichtigste Teil dieses Abschnitts die Definition der Chomsky-Hierarchie, und eine der Kategorien der Chomsky-Hierarchie ist die Kontextfreiheit. Um also entscheiden zu können, ob Englisch eine kontextfreie

Sprache ist, muss man sich zuerst mir der Chomsky-Hierarchie vertraut machen.

Abstrakte Automaten sind ein praktisches Werkzeug im Umgang mit formalen Sprachen und Grammatiken, von ihnen handelt das vierte Kapitel. Zunächst wird das mathematische Modell eines (Halb-)Automaten mit dazugehörigen Beispielen vorgestellt, dann werden zwei speziellere Automaten thematisiert: die endlichen Akzeptoren und die Kellerautomaten, diese korrespondieren genau mit den regulären bzw. kontextfreien Sprachen. Die wichtigsten Sätze dieses Kapitels sind die beiden Pumping-Lemmata und das Ogden-Lemma, das eine Verallgemeinerung des Pumping-Lemmas ist. Diese sind wichtige Hilfsmittel beim Nachweis, dass eine Sprache nicht regulär bzw. nicht kontextfrei ist. Dass Englisch keine kontextfreie Sprache ist, wird mithilfe des Ogden-Lemmas bewiesen.

Das fünfte Kapitel widmet sich schließlich dem Beweis von Higginbotham [7], dass Englisch keine kontextfreie Sprache sein kann, im sechsten Kapitel folgen noch Schlussbemerkungen.

Neben dem Artikel von Higginbotham, stützen sich die übrigen Kapitel auf zwei Werke der angewandten Algebra ([3] und [9]) und drei Bücher über theoretische Informatik bzw. Automatentheorie ([6], [8] und [13]). Die Quellen werden zu Beginn der einzelnen Kapitel oder Abschnitte angegeben, ansonsten stehen sie direkt bei der Definition, dem Satz oder dem Beispiel.

2 Formale Sprachen

Eine natürliche Sprache besteht aus Sätzen, die gewissen Ansprüchen genügen sollen. Meistens wird vorausgesetzt, dass ein Satz grammatikalisch korrekt ist und sinnvoll gebraucht wird. Aber natürliche Sprachen und ihre grammatikalischen Regeln sind außerordentlich komplex, das erkennt man spätestens beim Lernen einer Fremdsprache. Aus diesem Grund lassen sich natürliche Sprachen nur ansatzweise mit formalen Methoden erfassen. Ein großes Problem ist dabei die Mehrdeutigkeit, daher können natürliche Sprachen nur in sehr eingeschränkter Form für die Beschreibung von Aufgaben verwendet werden, die ein Computer lösen soll. Formale Sprachen wie Programmiersprachen eignen sich dafür besser. Die Theorie der formalen Sprachen bietet eine theoretische Grundlage für alle Sprachen, die mit automatischen Verfahren verarbeitet werden. Wichtige Begriffe wie Alphabet, Buchstabe oder Wort, die vom Umgang mit natürlichen Sprachen her geläufig sind, werden dabei verallgemeinert und formalisiert.
Abschnitt 2.1 beschäftigt sich mit Monoiden und folgt der Darstellung in [3, 395-399], besonders wichtig für den weiteren Verlauf ist das freie Monoid (siehe Definition 2.1.8). In Abschnitt 2.2, der sich an [6, 5-9] orientiert, soll die Verbindung zu den formalen Sprachen hergestellt werden. Abschließend werden noch zahlreiche Beispiele formaler Sprachen vorgestellt.

2.1 Monoide und Monoidoperationen

Die algebraische Struktur, die den formalen Sprachen zugrunde liegt, ist das sogenannte freie Monoid über einer bestimmen Menge. Daher folgen an dieser Stelle einleitende Definitionen und Eigenschaften.

Definition 2.1.1 Ein *Monoid* $(M, \circ, \varepsilon)$ ist eine nichtleere Menge M zusammen mit einer binären Verknüpfung $\circ$ auf M (also $\circ : M \times M \to M$, $(x, y) \mapsto x \circ y$) und einem *neutralen Element* ε (auch *Einselement* genannt), sodass:

(M1) $x \circ (y \circ z) = (x \circ y) \circ z \quad \forall x, y, z \in M$ (Assoziativgesetz),

(M2) $x \circ \varepsilon = \varepsilon \circ x = x \quad \forall x \in M$ (Existenz eines neutralen Elements).

Beispiel 2.1.2 Sei A eine Menge und die Funktion $f : A \to A$ eine darauf definierte Selbstabbildung. Dann ist die Menge $A^A := \{f : A \to A\}$ aller Selbstabbildungen gemeinsam mit der Verknüpfung von Abbildungen ein Monoid. Das neutrale Element ist dabei die identische Abbildung id_A.

Definition 2.1.3 Eine Teilmenge $U \subseteq M$ eines Monoids $(M, \circ, \varepsilon)$, die das neutrale Element ε enthält und bezüglich der Verknüpfung $\circ$ abgeschlossen ist, heißt *Untermonoid*.

Definition 2.1.4 Seien $(M, \circ_M, \varepsilon_M)$ und $(N, \circ_N, \varepsilon_N)$ zwei Monoide. Ein *Homomorphismus* ist eine Abbildung $f : M \to N$, sodass:

(H1) $f(x \circ_M y) = f(x) \circ_N f(y) \quad \forall x, y \in M$,

(H2) $f(\varepsilon_M) = \varepsilon_N$.

Definition 2.1.5 Eine *erzeugende Menge* eines Monoids $(M, \circ, \varepsilon)$ ist eine Teilmenge $X \subseteq M$, sodass jedes Element aus M als Produkt von Elementen aus X geschrieben werden kann.

Beispiel 2.1.6 Die Menge $\mathbb{N} = \{1, 2, 3, \dots\}$ aller natürlichen Zahlen ist gemeinsam mit der Multiplikation und dem neutralen Element 1 ein Monoid. Die Menge aller Primzahlen bildet demnach eine erzeugende Menge, da jede natürliche Zahl als Produkt von Primzahlen geschrieben werden kann.

Beispiel 2.1.7 Ebenso ist die Menge $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ gemeinsam mit der Addition und dem neutralen Element 0 ein Monoid. Die Menge $\{1\}$ ist eine erzeugende Menge.

Das nachfolgende Monoid spielt in der mathematischen Linguistik eine besonders wichtige Rolle. Im Abschnitt 2.2 wird darauf zurückgekommen werden.

Definition 2.1.8 Sei X eine beliebige Menge. Dann bezeichnet X^* die Menge aller endlichen Folgen mit Elementen aus X:

$$w = x_1 x_2 \dots x_r \quad (\text{wobei } x_i \in X,\ i \in \{1, \dots, r\},\ r \geq 0). \tag{2.1}$$

Die leere Folge wird geschrieben als ε. Die Multiplikation in X^* wird als Verkettung (bzw. Hintereinanderstellung) definiert:

$$(x_1 \dots x_r)(y_1 \dots y_s) = x_1 \dots x_r y_1 \dots y_s. \tag{2.2}$$

Das Assoziativgesetz ist leicht nachzuprüfen, außerdem sieht man, dass die leere Folge ε das neutrale Element ist. X^* wird *freies Monoid über X* genannt.

Beispiel 2.1.9 Wenn $X = \emptyset$ die leere Menge ist, dann wird X^* zum trivialen Monoid, das lediglich das neutrale Element ε enthält.

Beispiel 2.1.10 Abgesehen von diesem trivialen Fall, ist das einfachste freie Monoid jenes über einer einelementigen Menge $\{x\}$ gemeinsam mit der üblichen Multiplikation. Die Elemente sind dann $1, x, x^2, x^3, \dots$. Das neutrale Element ist 1. Man sieht, dass $\{x\}^*$ mit der Zuordnung $n \leftrightarrow x^n$ isomorph zu $\mathbb{N}_0$ aus Beispiel 2.1.7 ist. Interpretiert man $\{x\}$ als *Alphabet* und die darauf definierte Multiplikation als Hintereinanderstellung von *Buchstaben*, lassen sich alle *Wörter* des freien Monoids $\{x\}^*$ mithilfe der Kurzschreibweise $\varepsilon, x, x^2, x^3, \dots$ darstellen. Dabei ist $x^0 = \varepsilon$ und $x^n = \underbrace{xx \dots x}_{\text{n-mal}}$, wobei $n \in \mathbb{N}$ (siehe dafür auch Definition 2.2.1).

Definition 2.1.11 Die Darstellung (2.1) für ein Element w eines freien Monoids ist eindeutig, dies folgt direkt aus der Definition des freien Monoids. Die Anzahl r der Faktoren auf der rechten Seite ist invariant und wird *Länge* (oder *Grad*) genannt. Man schreibt dafür $|w|$.

Der Name *freies Monoid* ist aufgrund des folgenden Satzes gerechtfertigt.

Satz 2.1.12 *Jedes Monoid ist ein homomorphes Bild eines freien Monoids.*

Beweis. Sei $(M, \circ, \varepsilon)$ ein beliebiges Monoid und $A \subseteq M$ eine erzeugende Menge. Sei nun A' eine zu A gleichmächtige Menge und $\{A'\}^*$ das freie Monoid über A'. Es gibt dann eine Abbildung $f : \{A'\}^* \to M$, die definiert ist durch:

$$f(a'_1 \dots a'_r) = a_1 \dots a_r. \tag{2.3}$$

Dabei ist $a' \leftrightarrow a$ die Bijektion zwischen A' und A. Nachdem jedes Element aus $\{A'\}^*$ auf eindeutige Weise als Produkt $a'_1 \dots a'_r$ geschrieben werden kann, ist die Abbildung f durch den Ausdruck (2.3) wohldefiniert. Die Abbildung f ist surjektiv, weil A eine erzeugende Menge von M ist. Durch den Ausdruck (2.2) ist leicht zu sehen, dass f ein Homomorphismus ist. □

Definition 2.1.13 Sei $(M, \circ, \varepsilon)$ ein Monoid. Eine *M-Menge* (auch Menge mit *M-Operation*) ist eine Menge S zusammen mit einer Abbildung $S \times M \to S$, $(s, x) \mapsto sx$, sodass:

(S1) $s(xy) = (sx)y \quad \forall s \in S,\ \forall x, y \in M,$

(S2) $s\varepsilon = s.$

Bemerkung 2.1.14 Wenn R_x die Abbildung $s \mapsto sx$ auf S bezeichnet, dann kann man (S1) und (S2) schreiben als $R_{xy} = R_x R_y$ und $R_\varepsilon = \varepsilon$. Auf diese Weise kann man sagen, dass die Abbildung $R : x \to R_x$ ein Monoid-Homomorphismus von M nach S^S ist. Beispielsweise ist M selbst eine M-Menge, dabei ist die Verknüpfung auf M die M-Operation. Dies wird manchmal auch die *Standarddarstellung* von M genannt.

Satz 2.1.15 *Jedes Monoid kann genauso als Monoid aus Abbildungen dargestellt werden.*

Beweis. Sei M ein Monoid und $x \mapsto \rho_x$ die Standarddarstellung von M. Dann ist ρ ein Homomorphismus von M nach M^M. Wenn $\rho_x = \rho_y$ gilt, dann $x = \varepsilon\rho_x = \varepsilon\rho_y = y$, also ist der Homomorphismus injektiv. □

Bemerkung 2.1.16 Sei nun M wieder ein allgemeines Monoid und S eine M-Menge. Nach Satz 2.1.12 lässt sich M als homomorphes Bild eines freien Monoids X^* über einer Menge X schreiben. Demnach gibt es einen Homomorphismus $X^* \to M \to S^S$. Daher kann jede Menge S mit M-Operation auch als Menge mit X^*-Operation aufgefasst werden, wobei X einer erzeugenden Menge von M entspricht.

Definition 2.1.17 Ein freies Monoid hat verschiedene bemerkenswerte Eigenschaften, die auch dazu dienen können, das Monoid zu charakterisieren.

(i) Ein Monoid M heißt *konisch*, falls $xy = \varepsilon \Rightarrow x = y = \varepsilon$.

(ii) Sei M ein Monoid. Falls $\forall x, y \in M$, $xu = yu$ oder $ux = uy$ für ein $u \in M$ impliziert, dass $x = y$, dann heißt das Monoid M *kürzbar*.

(iii) Ein kürzbares Monoid M heißt *starr*, falls $ac = bd$, dann $\exists z \in M$, sodass entweder $a = bz$ oder $b = az$.

Proposition 2.1.18 *Jedes freie Monoid ist konisch und starr.*

Beweis. Dass jedes freie Monoid konisch ist, geht direkt aus der Darstellung (2.2) hervor.
Um die Kürzbarkeit nachzuweisen, sei zunächst festgehalten, dass in jedem Element $x_1 \dots x_r \neq \varepsilon$ der Faktor x_1 ganz links, ebenso wie der Faktor x_r ganz rechts, eindeutig sind. Demnach gilt $x_1 \dots x_r = y_1 \dots y_s$ nur, wenn $r = s$ und $x_i = y_i$ für $(i = 1, \dots, r)$. Es folgt dann aus $xu = yu$,

$$x_1 \dots x_r u_1 \dots u_t = y_1 \dots y_s u_1 \dots u_t,$$

dass beide Seiten die gleiche Länge haben und $x_i = y_i$ $(i = 1, \dots, r = s)$, also $x = x_1 \dots x_r = y_1 \dots y_r = y$. Beim Fall $ux = uy$ wird analog argumentiert.
Um die Starrheit nachzuweisen, sei $ac = bd$ mit $a = x_1 \dots x_r$, $b = y_1 \dots y_s$, $c = u_1 \dots u_h$ und $d = v_1 \dots v_k$. Dann gilt

$$x_1 \dots x_r u_1 \dots u_h = y_1 \dots y_s v_1 \dots v_k.$$

Aus Symmetriegründen kann man annehmen, dass $r \leq s$ ist. Dann gilt $x_1 = y_1, \dots, x_r = y_r$ und somit $b = x_1 \dots x_r y_{r+1} \dots y_s = az$ mit $z = y_{r+1} \dots y_s$. Also ist das freie Monoid starr. Es ist anzumerken, dass wenn $ac = bd$, dann $a = bz$ oder $b = az$, je nachdem ob $|a| \geq |b|$ oder $|a| \leq |b|$. □

Definition 2.1.19 Sei M ein Monoid mit neutralem Element ε. Ein Element u heißt *Einheit*, falls $\exists v \in M$ mit $uv = vu = \varepsilon$, das heißt also, falls u invertierbar ist.

Bemerkung 2.1.20 In einem konischen Monoid ist die einzige Einheit (also das einzige invertierbare Element) das neutrale Element ε. In einem kürzbaren Monoid reicht es, eine der Gleichungen anzunehmen, angenommen $uv = \varepsilon$. Dann $(vu)v = v(uv) = v\varepsilon = \varepsilon v$, und durch Kürzen somit $vu = \varepsilon$.

Definition 2.1.21 Ein Element heißt *irreduzibel*, falls es nicht invertierbar ist und nicht als Produkt zweier nicht invertierbarer Faktoren geschrieben werden kann.

Bemerkung 2.1.22 In einem freien Monoid sind die einzigen irreduziblen Elemente jene mit Länge 1. Dies zeigt nebenbei, dass in einem freien Monoid die freie erzeugende Menge eindeutig bestimmt ist als die Menge aller irreduziblen Elemente.

Satz 2.1.23 *Sei F ein Monoid und X die Menge aller irreduziblen Elemente. F ist frei über X als frei erzeugende Menge, genau dann wenn F konisch und starr ist und von X erzeugt wird.*

Beweis. In Proposition 2.1.18 wurde bereits gezeigt, dass ein freies Monoid konisch und starr ist. Gelten nun umgekehrt genau diese Bedingungen, dann muss gezeigt werden, dass jedes Element aus F auf eindeutige Weise als Produkt von Elementen aus X geschrieben werden kann. Ein beliebiges $a \in F$ kann auf mindestens eine Weise als so ein Produkt ausgedrückt werden, denn X erzeugt F. Wenn gilt

$$a = x_1 \dots x_r = y_1 \dots y_s \quad \text{(wobei } x_i, y_i \in X\text{)},$$

dann folgt aus der Starrheit $x_1 = y_1 b$ oder $y_1 = x_1 b$ für ein $b \in F$, angenommen ersteres gilt. Nachdem x_1 und y_1 irreduzibel sind, muss b invertierbar sein und somit $b = \varepsilon$, da F konisch ist. Somit folgt, dass $x_1 = y_1$, und nach Kürzung dieses Faktors bleibt $x_2 \dots x_r = y_2 \dots y_s$. Mittels vollständiger Induktion nach $\max\{r, s\}$ folgt $r - 1 = s - 1$, also $r = s$ und $x_2 = y_2, \dots, x_r = y_r$. Daher ist F tatsächlich frei über X. □

2.2 Alphabete, Buchstaben, Wörter, Sprachen

An dieser Stelle soll der Zusammenhang zwischen (freien) Monoiden und formalen Sprachen hergestellt werden. Der Begriff *freies Monoid* (siehe Definition 2.1.8) wurde bereits in Abschnitt 2.1 eingeführt.

Definition 2.2.1 Die nachfolgend definierten Begriffe kommen analog bei natürlichen Sprachen vor, diese parallele Verwendung kann einem besseren Verständnis dienen.

(i) Ein *Alphabet* V ist eine endliche, nichtleere Menge von *Zeichen* (manchmal auch *Symbole* oder *Buchstaben* genannt).

(ii) Sei V ein Alphabet und $k \in \mathbb{N}_0$. Eine endliche Folge

$$(x_1, \dots, x_k) \quad (\text{wobei } x_i \in V,\ i \in \{1, \dots, k\})$$

heißt *Wort über* V der *Länge* k. Es gibt genau ein Wort der Länge 0, das *Leerwort*, das mit ε bezeichnet wird. Die Länge eines Wortes x wird durch $|x|$ dargestellt.

(iii) Die *Menge aller Wörter* über V wird mit V^* bezeichnet.

(iv) Die Menge der *nichtleeren* Wörter über V ist $V^+ := V^* \setminus \{\varepsilon\}$.

(v) Jede (beliebige) Teilmenge von V^* wird als *(formale) Sprache* bezeichnet.

Bemerkung 2.2.2 Wenn es das Alphabet erlaubt, wird die Kurzschreibweise $x_1 x_2 \dots x_k$ für Wörter verwendet. Beim Alphabet $\{1, 11\}$ wäre dies nicht möglich, da man so beispielsweise beim Wort 111 nicht erkennen könnte, aus welchen Zeichen es besteht.

Bemerkung 2.2.3 In diesem allgemeinen Sinn wird jede Menge als Alphabet aufgefasst, die nichtleer und endlich ist, und jede Wortmenge wird als Sprache bezeichnet.

Bemerkung 2.2.4 Bei natürlichen Sprachen gibt es zwei Betrachtungsebenen. Auf der unteren Ebene lassen sich Wörter über einem Alphabet bilden, so können z. B. über dem deutschen Alphabet $\{\text{a}, \text{b}, \text{c}, \dots \text{z}\} \cup \{\text{ä}, \text{ö}, \text{ü}, \text{ß}\}$ alle Einträge des Dudens gebildet werden. Auf einer höheren Ebene besteht das Alphabet aus Wörtern, und über diesem Wortschatz lassen sich Sätze bilden. Nachfolgend werden wir deswegen nicht mehr zwischen Wörtern und Sätzen unterscheiden.

Definition 2.2.5 Sei V ein Alphabet. Die *Verkettung* von Wörtern $x, y \in V^*$ ergibt ein neues Wort xy, wobei gilt:

$$xy = \underbrace{x_1 \dots x_k}_{x} \underbrace{y_1 \dots y_l}_{y} \quad (\text{wobei } x_i, y_j \in V).$$

Bemerkung 2.2.6 Meistens wird die Abkürzung $x^n = \underbrace{x \dots x}_{\text{n-mal}}$ verwendet (wobei $x \in V^*$).

Dabei gilt $x^0 = \varepsilon$ und $x\varepsilon = \varepsilon x = x$ für alle $x \in V^*$, insbesondere $\varepsilon\varepsilon = \varepsilon$.

Bemerkung 2.2.7 An dieser Stelle sei der Zusammenhang nochmals festgehalten: Sei V eine endliche (nichtleere) Menge, die man Alphabet nennt. Dann handelt es sich bei der Menge aller Wörter über V aus der Definition 2.2.1 (iii) um das freie Monoid über V aus der Definition 2.1.8, das man mit V^* bezeichnet. Die Verkettung von Wörtern aus der Definition 2.2.5 ist dabei die Verknüpfung des freien Monoids V^* aus der Definition 2.1.8. Unter einer (formalen) Sprache über V versteht man somit eine beliebige Teilmenge des freien Monoids V^*. Die Definitionen sind also äquivalent.

Für ein besseres Verständnis werden nun verschiedene Beispiele für Alphabete und formale Sprachen vorgestellt.

Beispiel 2.2.8 $V = \{0, 1\}$. Die Wörter über V sind alle endlichen Folgen, die aus Nullern und Einsern bestehen, inklusive des Leerworts ε. Die Menge aller Wörter ist $V^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots\}$. Die Menge aller vorzeichenlosen Binärzahlen ohne führende Nullen $L = \{0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, \dots\}$ ist eine typische Sprache über V.

Beispiel 2.2.9 $V = \{0, 1, \dots, 9\}$. Dieses Alphabet ist geeignet zur Darstellung von natürlichen Zahlen wie 140690 in Dezimalschreibweise.

Beispiel 2.2.10 $V = \{1\}$. Dieses einelementige Alphabet kann zur *unären* Darstellung natürlicher Zahlen verwendet werden. Dabei gilt: $0 \mathrel{\widehat{=}} 1$, $1 \mathrel{\widehat{=}} 11$, $2 \mathrel{\widehat{=}} 111$, $3 \mathrel{\widehat{=}} 1111, \dots$.

Beispiel 2.2.11 $V = \{a, b, c, \dots, z\}$. Dies ist das lateinische Alphabet, das den Wörtern vieler natürlicher Sprachen zugrunde liegt. Eventuell muss es um Großbuchstaben, Umlaute oder Sonderzeichen erweitert werden. Wörter wie *Lasagne*, *Katze*, *hassen* oder *Montag* bilden den Wortschatz einer Sprache.

Beispiel 2.2.12 $V = \{\text{Worteinträge eines Lexikons}\}$. Beispielsweise bildet die Menge der Einträge im Duden ein Alphabet mit dem natürlichsprachliche Äußerungen formuliert werden können, z. B. *Katzen hassen Montage, aber lieben Lasagne*. Solche korrekt gebildeten Sätze stellen dann die deutsche Sprache dar. Auf dieser Sprachebene sind dies *Wörter* über dem Alphabet der Dudeneinträge, alternativ können Sätze der deutschen Sprache auch als Wörter über dem lateinischen Alphabet gesehen werden. Ob ein Satz korrekt ist, kann mit Hilfe von Grammatikregeln entschieden werden, diese werden im nächsten Kapitel thematisiert.

Beispiel 2.2.13 $V =$ ASCII (American Standard Code for Information Interchange) oder $V =$ EBCDIC (Extended Binary-Coded Decimal Interchange Code). Programme einer Programmiersprache werden üblicherweise mit einem dieser Alphabete formuliert. Man könnte Programme zwar als endliche Folgen solcher Zeichen auffassen, jedoch unterscheidet man zwei Betrachtungsebenen, ganz analog zu den natürlichen Sprachen. Auf der unteren Ebene werden mit einem der beiden Alphabete zunächst *Eingabesymbole* gebildet. Bei der Programmiersprache C unterscheidet man fünf Sorten von Eingabesymbolen: *Schlüsselwörter*, *Bezeichner*, *Literale*, *Operatoren* und *Begrenzer*. Der erste Schritt bei der Kompilierung eines Programms besteht darin, die

einzelnen Eingabesymbole herauszuarbeiten. Beispielsweise muss die Zeichenfolge `while` als Schlüsselwort identifiziert werden, dagegen wäre `whilex` ein Bezeichner.

Beispiel 2.2.14 $V = \{\text{Eingabesymbole einer Programmiersprache}\}$. Auf dieser Stufe bestehen Programme aus Wörtern von Eingabesymbolen. Als Programmiersprache kann man die Menge aller endlichen Folgen von Eingabesymbolen auffassen, die ein korrektes Programm darstellen. Dabei wird mithilfe von Grammatiken festgelegt, wie einzelne Eingabesymbole zu einem korrekten Programm zusammengefügt werden. Die folgende Anweisung der Programmiersprache C

```
while(x3 < 100) x3 += 1;
```

ist demnach ein Teilwort eines Programms bestehend aus den Eingabesymbolen `while` (Schlüsselwort), `(` (Begrenzer), `x3` (Bezeichner), `<` (Operator), `100` (Literal), `)` (Begrenzer), `x3` (Bezeichner), `+=` (Operator), `1` (Literal) und `;` (Begrenzer).

3 Grammatiken

Aus dem Umgang mit natürlichen Sprachen sind uns Grammatiken als Regelwerke bekannt, die die Struktur grammatikalisch korrekter Sätze definieren. Einige formale Sprachen können mithilfe von Grammatiken dargestellt werden. Das bedeutet insbesondere, dass nicht jede formale Sprache mithilfe einer Grammatik erzeugt werden kann.
Neben der formalen Definition einer Phrasenstrukturgrammatik (siehe Definition 3.1.5), ist besonders die Chomsky-Hierarchie (siehe Definition 3.2.1) für den weiteren Verlauf von Bedeutung. Falls nicht anders angegeben, basiert dieses Kapitel auf [3, 399-403] und [6, 23-37].

3.1 Grundlegende Definitionen

Die algebraische Sprachwissenschaft entstand durch den Versuch des US-amerikanischen Linguisten NOAM CHOMSKY, den Prozess der Satzbildung natürlicher Sprachen zu analysieren und zu präzisieren. Dabei hielt er fest, dass die Bedeutung eines Satzes irrelevant ist, wenn man den Satz auf grammatikalische Korrektheit prüfen will. Noam Chomsky illustriert dies durch einen sinnlosen Satz, der aber grammatikalisch korrekt ist: *Colorless green ideas sleep furiously.* Um seine Aussage zu unterstreichen, stellt er diesem Satz einen anderen gegenüber, der nicht korrekt ist: *Furiously sleep ideas green colorless*.
Man könnte grammatikalische Korrektheit auch so interpretieren, dass ein Satz eine Aussage darstellen muss, um grammatikalisch korrekt zu sein. Das bedeutet, man kann dem Satz eines der Attribute *wahr* oder *falsch* eindeutig zuordnen. Von einem ungrammatikalischen Satz kann man nicht sagen, ob dieser wahr oder falsch ist.

Das nächste Beispiel (siehe auch [4, 30-31]) soll den Satzbau einer natürlichen Sprache demonstrieren. Auf die Begriffe *Variablen*, *Axiom* und *Produktionen* wird anschließend eingegangen.

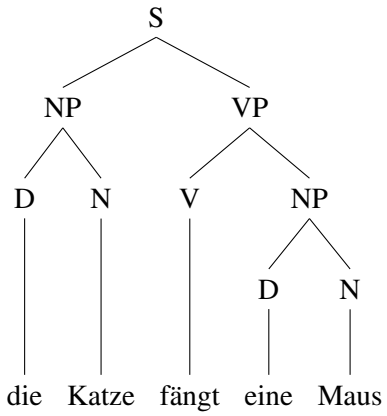
Beispiel 3.1.1 Das Vokabular sei $V = \{die, eine, Katze, Maus, fängt, frisst\}$. Die Variablen werden gebildet durch $\{S, NP, VP, D, N, V\}$. Dabei steht S für Satz, NP für Nominalphrase, VP für Verbalphrase, D für Artikel, N für Nomen und V für Verb. Das Axiom bzw. Startsymbol des Systems sei S. Die Produktionen sind:

$$\begin{aligned} S &\to NP\ VP && (3.1)\\ NP &\to D\ N && (3.2)\\ VP &\to V\ NP && (3.3)\\ D &\to die \mid eine && (3.4)\\ N &\to Katze \mid Maus && (3.5)\\ V &\to fängt \mid frisst && (3.6) \end{aligned}$$

Die ersten drei Regeln entsprechen ganz grob den Regeln der deutschen Syntax und können folgendermaßen ausformuliert werden:

1. *Für einen Satz schreibe man eine Nominalphrase, gefolgt von einer Verbalphrase.*
2. *Für eine Nominalphrase schreibe man einen Artikel, gefolgt von einem Nomen.*
3. *Für eine Verbalphrase schreibe man ein Verb, gefolgt von einer Nominalphrase.*

Die anderen Regeln gestatten, von einer grammatikalischen Kategorie zu einem bestimmten Wort überzugehen, welches unter die fragliche Kategorie fällt. Der nachfolgende Baum illustriert die Anwendung der Regeln.



In diesem Fall erhält man den Satz *die Katze fängt eine Maus*. Auf analoge Weise könnte man auch den Satz *die Katze frisst eine Maus* oder *die Maus fängt eine Katze* erhalten. All diese Sätze sind grammatikalisch korrekt.

Bemerkung 3.1.2 Im vorangegangenen Beispiel wurde die abkürzende Schreibweise für Regeln mit gleicher linker Seite verwendet, also

$$\mathrm{N} \to \text{Katze} \mid \text{Maus} \quad \text{statt} \quad \mathrm{N} \to \text{Katze}, \mathrm{N} \to \text{Maus}.$$

Bemerkung 3.1.3 Im Prinzip besteht eine Satzanalyse darin, die Bestandteile des Satzes zu ermitteln und zu überprüfen, ob die Komponenten anhand vorgeschriebener Regeln richtig zusammengesetzt wurden. Das mathematische Modell besteht aus einer Menge an Produktionsregeln, die unweigerlich zu allen Sätzen der Sprache (und zu keinen weiteren Sätzen) führen. Die wichtigen Begriffe *Alphabet*, *Buchstabe*, *Wort* und *Sprache* kann man in Definition 2.2.1 nachschlagen, den Zusammenhang zum *freien Monoid* findet man in Bemerkung 2.2.7.

Bemerkung 3.1.4 In Beispiel 3.1.1 sieht man eine Möglichkeit, wie eine Sprache beschrieben werden kann, nämlich indem es eine Menge an vorgegebenen Regeln gibt, die alle Wörter der Sprache produzieren. Auf diese Weise wird eine bestimmte formale Sprache durch Produktionsregeln herausgegriffen. Diese Regeln bilden die Grammatik einer Sprache.

Die folgende Definition verallgemeinert den Begriff der Grammatik auf beliebige Sprachen.

Definition 3.1.5 Ein Quadrupel $G = (V_N, V_T, P, S)$ heißt *Grammatik* (auch *Phrasenstrukturgrammatik*), falls gilt:

(i) V_N ist eine nichtleere, endliche Menge an *Variablen* (*nichtterminalen Symbolen*).

(ii) V_T ist eine nichtleere, endliche Menge an *terminalen Symbolen* (*Terminalen*).

(iii) P ist eine endliche Menge von Regeln der Form

$$\alpha \to \beta$$

mit $\alpha \in (V_N \cup V_T)^+$ und $\beta \in (V_N \cup V_T)^*$. Die Elemente von P werden *Produktionen*, *Produktionsregeln* oder *Grammatikregeln* genannt.

(iv) $S \in V_N$ ist ein *Startsymbol* (auch *Axiom* genannt).

Bemerkung 3.1.6 Die terminalen Symbole werden auch *lexikalische Einheiten* genannt, und die Variablen heißen manchmal *grammatische Kategorien*.

Bemerkung 3.1.7 Bei den bisher betrachteten Beispielen für Grammatikregeln hat die linke Seite immer aus einer einzelnen Variablen bestanden wie *Satz* oder *Anweisung*. Im Allgemeinen darf laut Definition bei einer Regel $\alpha \to \beta$ auch auf der linken Seite eine beliebige Symbolfolge bestehend aus Variablen und terminalen Symbolen stehen, nur $\alpha = \varepsilon$ ist ausgeschlossen.

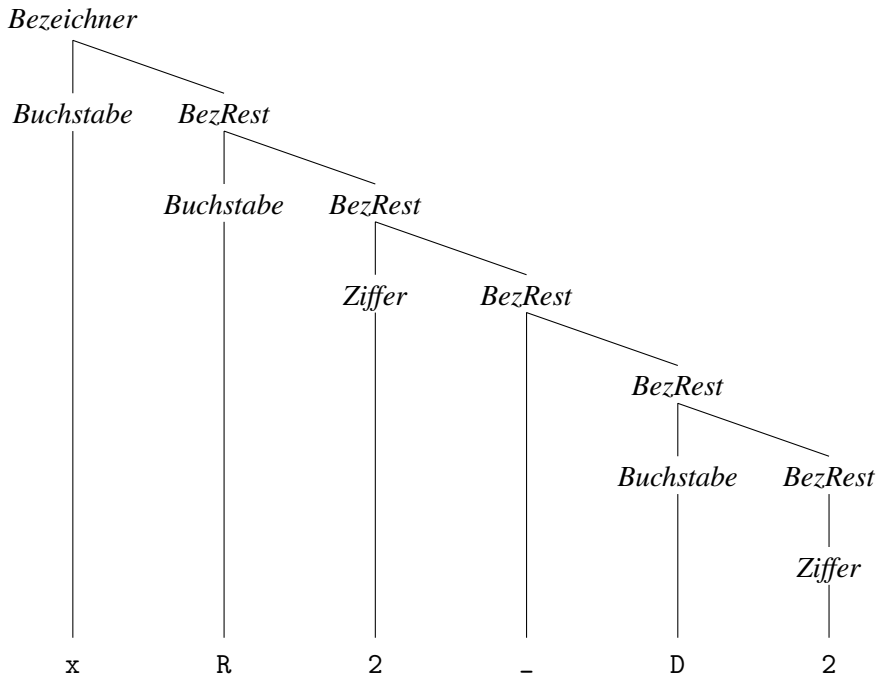
Bemerkung 3.1.8 Um ein Wort einer bestimmten Sprache zu erhalten, beginnt man also mit dem Startsymbol S und wendet so lange Ersetzungsregeln an, bis keine Variablen mehr übrig sind. Das entstandene Wort besteht dann nur noch aus terminalen Symbolen, man nennt es auch *Terminalwort*.

Beispiel 3.1.9 Dieses Beispiel soll eine Grammatik für Bezeichner der Programmiersprache C zeigen. Diese können mit Buchstaben und Ziffern sowie dem Unterstrich gebildet werden, wobei das erste Zeichen keine Ziffer sein darf. Beispielsweise sind `x2A_98` und `_3xy` zulässige Bezeichner, das Wort `3xyz` jedoch nicht. Sei nun die Grammatik $G = (V_N, V_T, P, S)$ folgendermaßen festgelegt:

$$\begin{aligned} V_N &= \{\textit{Bezeichner, BezRest, Buchstabe, Ziffer}\} \\ V_T &= \{\texttt{A},\texttt{B},\ldots,\texttt{Z},\texttt{a},\texttt{b},\ldots,\texttt{z},\texttt{\_},\texttt{1},\ldots,\texttt{9}\} \\ S &= \{\textit{Bezeichner}\} \end{aligned}$$

Die Menge P besteht aus folgenden Produktionsregeln:

$$\begin{aligned} \textit{Bezeichner} &\to \textit{Buchstabe} \mid \texttt{\_} \mid \textit{Buchstabe BezRest} \mid \texttt{\_}\ \textit{BezRest} \\ \textit{BezRest} &\to \textit{Buchstabe BezRest} \mid \texttt{\_}\ \textit{BezRest} \mid \textit{Ziffer BezRest} \mid \textit{Buchstabe} \mid \texttt{\_} \mid \textit{Ziffer} \\ \textit{Buchstabe} &\to \texttt{A} \mid \texttt{B} \mid \ldots \mid \texttt{Z} \mid \texttt{a} \mid \texttt{b} \mid \ldots \mid \texttt{z} \\ \textit{Ziffer} &\to \texttt{0} \mid \texttt{1} \mid \ldots \mid \texttt{9} \end{aligned}$$



Der obige Baum zeigt beispielsweise die Erzeugung des Bezeichners `xR2_D2`.

Bemerkung 3.1.10 Anhand des Beispiels 3.1.9 kann man sich nochmals den Unterschied zwischen terminalen und nichtterminalen Symbolen verdeutlichen. Die nichtterminalen Symbole sind $V_N = \{$*Bezeichner, BezRest, Buchstabe, Ziffer*$\}$, also eine Menge an Hilfsvariablen, die nicht Teil eines Terminalwortes sein können. $V_T = \{$`A,B,...,Z,a,b,...,z,_,1,...,9`$\}$ sind terminale Symbole, die Teil eines Terminalwortes, also eines Bezeichners der Programmiersprache C, sein können.

Das Zergliedern einer vorhandenen Struktur in immer feinere Teilstrukturen geschieht somit durch die Anwendung von Grammatikregeln. Das Ziel dabei ist, eine grobe Struktur solange zu verfeinern, bis eine weitere Zergliederung nicht mehr möglich ist. Der erste Teil der folgenden Definition beschreibt, wie ein solcher Verfeinerungsschritt erfolgen kann. Im zweiten Teil wird beschrieben, wie Wörter durch eine Folge solcher Einzelschritte bearbeitet werden.

Definition 3.1.11 Seien $G = (V_N, V_T, P, S)$, $V = V_N \cup V_T$ und $x, y \in V^*$.

(i) Anwenden einer Grammatikregel: Man sagt, *x wird unmittelbar übergeführt in y* und schreibt dafür

$$x \to y \quad \Leftrightarrow \quad (x = \gamma\alpha\delta),\ (y = \gamma\beta\delta) \ \wedge \ \alpha \to \beta \in P,$$

wobei $\alpha \in V^+$ und $\beta, \gamma, \delta \in V^*$. Andere Redewendungen sind *x führt unmittelbar zu y* oder *y ist direkt (unmittelbar) aus x ableitbar*.

(ii) Anwenden mehrerer Grammatikregeln nacheinander: Man sagt, x *wird übergeführt in* y und schreibt dafür

$$x \twoheadrightarrow y \quad \Leftrightarrow \quad \exists(w_0, \ldots, w_n) : w_0 = x, w_n = y \ \wedge \ w_{i-1} \to w_i$$

wobei $n \in \mathbb{N}_0$ und $i \in \{1, \ldots, n\}$. Diese Folge heißt *Ableitung* von y aus x mit der Bezeichnung $w_0 \to w_1 \to \cdots \to w_n$. Andere Redewendungen sind x *führt zu* y oder y *ist aus* x *ableitbar*.

Bemerkung 3.1.12 Ableitungen in 0 Schritten sind zugelassen, daher gilt generell $x \twoheadrightarrow x$.

Definition 3.1.13 Die von einer Grammatik $G = (V_N, V_T, P, S)$ *erzeugte* (auch *dargestellte*) Sprache ist

$$L(G) = \{x \in V_T^* | S \twoheadrightarrow x\}.$$

Anders ausgedrückt, die von einer Grammatik erzeugte Sprache ist die Menge aller Terminalwörter, die man aus dem Startsymbol ableiten kann.

Bemerkung 3.1.14 Die Wörter der von der Grammatik erzeugten Sprache werden auch als die *grammatikalisch korrekten* Wörter bezeichnet.

Bemerkung 3.1.15 Nachdem dies für das Verständnis von großer Bedeutung ist, sei nochmals betont, dass zwischen Wörtern einer formalen Sprache und Sätzen einer natürlichen Sprache nicht unterschieden wird. Sprachliche Sätze können im Rahmen formaler Sprachen einfach als sehr lange Wörter angesehen werden (siehe dazu auch Bemerkung 2.2.4).

Beispiel 3.1.16 Betrachtet man nun nochmals das Beispiel 3.1.9, dann ist $V = V_N \cup V_T = \{$*Bezeichner*, *BezRest*, *Buchstabe*, *Ziffer*, `A`, `B`, … , `Z`, `a`, `b`, … , `z`, `_`, `1`, … , `9`$\}$. V^* ist das freie Monoid über V, also die Menge aller endlichen Wörter mit Elementen aus V. Elemente aus V^* wären beispielsweise `c3`*Bezeichner*`po` oder `r2_d2`*Ziffer*. Wörter, die nur aus terminalen Symbolen bestehen, sind Elemente aus V_T^*, zum Beispiel `1A_c3po` oder `0815_r2_d2`. Die gegebene Grammatik erzeugt Bezeichner der Programmiersprache C, also ist die von dieser Grammatik erzeugte Sprache $L = \{x \in V_T^* | \textit{Bezeichner} \twoheadrightarrow x\}$. Hier kommen keine Terminalwörter vor, an deren Anfang eine Ziffer steht. Elemente der Sprache L wären beispielsweise `c3po` oder `r2_d2`. Offensichtlich gilt $L \subset V_T^* \subset V^*$, also ist die formale Sprache L tatsächliche eine Teilmenge des freien Monoids V^*.

Es folgen noch weitere Beispiele für formale Sprachen.

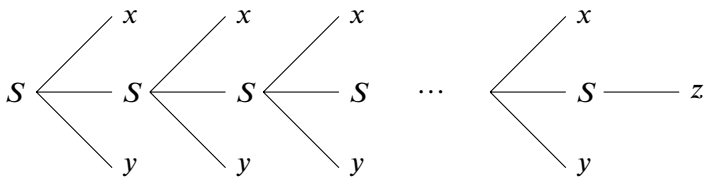
Beispiel 3.1.17 $L = \{x^n | n \geq 1\}$. Die Produktionsregeln sind $S \to x | Sx$. Eine typische Ableitung wäre $S \to Sx \to Sx^2 \to Sx^3 \to x^4$. Zum besseren Verständnis sei an dieser Stelle die erzeugende Grammatik G nochmals festgehalten. Die Grammatik $G = (V_N, V_T, P, S)$ besteht aus einer Menge an (nichtterminalen) Variablen $V_N = \{S\}$, einer Menge an terminalen Symbolen $V_T = \{x\}$, einer Menge an Grammatikregeln $P = \{S \to x, \ S \to Sx\}$ und einem ausgewiesenen Startsymbol $S \in V_N$.

Beispiel 3.1.18 $L = \{xy^n | n \geq 0\}$. Produktionsregeln sind $S \to x | Sy$.

Beispiel 3.1.19 $L = \{x^m y^n | m, n \geq 0\}$. Produktionsregeln sind $S \to xS|Sy|\varepsilon$.

Beispiel 3.1.20 $L = \{x^m y^n | 0 \leq m \leq n\}$. Produktionsregeln sind $S \to xSy|Sy|\varepsilon$.

Beispiel 3.1.21 $L = \{x^n z y^n | n \geq 0\}$. Produktionsregeln sind $S \to xSy|z$. Das Erzeugen dieser Sprache soll im nachfolgenden Baumdiagramm dargestellt werden.



Beispiel 3.1.22 Die *leere* Sprache $L = \emptyset$ hat die Produktionsregel $S \to S$.

Beispiel 3.1.23 Die *universelle* Sprache V_T^* hat die Produktionsregeln $S \to Sx|\varepsilon$ (mit $x \in V_T$).

3.2 Chomsky-Hierarchie

Im letzten Abschnitt hat man gesehen, dass bestimmte formale Sprachen von einer Grammatik erzeugt werden können. Das Konzept einer formalen Sprache ist offensichtlich zu weit gefasst, um von praktischem Nutzen zu sein. Daher gibt es bestimmte Klassen von Sprachen, indem von der erzeugenden Grammatik gewisse Bedingungen gefordert werden. Die folgende Klassifikation formaler Sprachen ist bekannt als *Chomsky-Hierarchie*. Noam Chomsky, einer der herausragendsten Linguisten, hat diese hierarchische Beziehung verschiedener Grammatiktypen bzw. der zugehörigen Sprachklassen um das Jahr 1956 angegeben (siehe hierfür [6, 32]).

Die nachfolgende Definition ist [3, 400-401] und [6, 32] nachempfunden.

Definition 3.2.1 (Chomsky-Hierarchie) Sei $G = (V_N, V_T, P, S)$ eine Grammatik und $V = V_N \cup V_T$. Unter einer *Phrasenstruktursprache* (auch *Typ-0-Sprache*) versteht man jede Sprache, die von einer Phrasenstrukturgrammatik erzeugt wird.

(i) Eine Sprache heißt *kontextsensitiv* (auch *Typ-1-Sprache* oder *CS-Sprache*), wenn sie von einer Grammatik erzeugt werden kann, deren Produktionsregeln folgende Form haben:

$$f\alpha g \to fug \quad (\text{wobei } \alpha \in V_N,\ u \in V^+ \text{ und } f, g \in V^*).$$

(ii) Eine Sprache heißt *kontextfrei* (auch *Typ-2-Sprache* oder *CF-Sprache*), wenn sie von einer Grammatik erzeugt werden kann, deren Produktionsregeln folgende Form haben:

$$\alpha \to u \quad (\text{wobei } \alpha \in V_N \text{ und } u \in V^+).$$

(iii) Eine Sprache heißt *regulär* (auch *Typ-3-Sprache*), wenn sie von einer Grammatik erzeugt werden kann, deren Produktionsregeln folgende Form haben:

$$\alpha \to x\beta,\ \alpha \to x \quad (\text{wobei } x \in V_T \text{ und } \alpha, \beta \in V_N).$$

Bemerkung 3.2.2 Bei Grammatiken vom Typ 2 und Typ 3 könnten auch Produktionen der Form $\alpha \to \varepsilon$ zugelassen werden, wobei α eine vom Startsymbol verschiedene Variable ist. Dadurch erhöht sich die Mächtigkeit des entsprechenden Grammatiktyps jedoch nicht, und da die vorgestellte Form die hierarchische Struktur besser deutlich macht, beschränken wir uns der Einfachheit halber auf diese Variante.

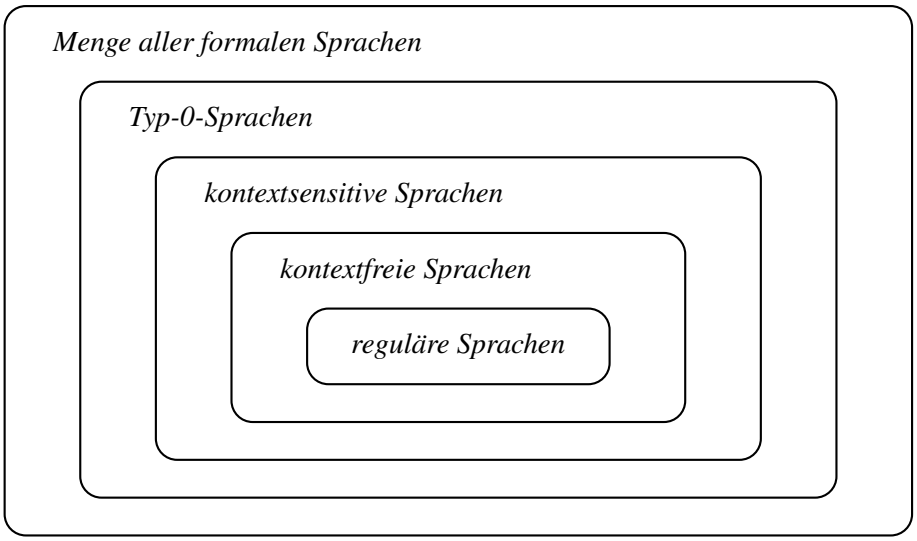
Bemerkung 3.2.3 Die Begriffe der Chomsky-Hierarchie werden analog für die entsprechenden Grammatiktypen verwendet, also beispielsweise Typ-3-Grammatik, CF-Grammatik oder kontextsensitive Grammatik.

Bemerkung 3.2.4 Es sind keinesfalls alle Sprachen vom Typ 0, das heißt, nicht alle formalen Sprachen werden von einer Grammatik erzeugt. Nachdem das Alphabet und die Menge an Produktionsregeln endlich sind, ist die Menge aller Typ-0-Sprachen abzählbar, wohingegen es überabzählbar viele Sprachen gibt, da eine unendliche Menge überabzählbar viele Teilmengen hat.

Wenn $\mathscr{L}_i$ $(i = 0, 1, 2, 3)$ die Klasse aller zugehörigen Sprachen vom Typ i bezeichnet, dann gilt offensichtlich:

$$\mathscr{L}_0 \supseteq \mathscr{L}_1 \supseteq \mathscr{L}_2 \supseteq \mathscr{L}_3.$$

Beispielsweise ist jede reguläre Grammatik auch kontextfrei, aber auch kontextsensitiv und vom Typ 0, oder jede kontextfreie Sprache ist auch eine kontextsensitive Sprache. Das Ziel bei der Einordnung einer Grammatik oder Sprache ist immer, den speziellsten Typ anzugeben. Das nachfolgende Mengendiagramm illustriert die Beziehung dieser Sprachklassen.



Bemerkung 3.2.5 Tatsächlich handelt es sich in dieser Mengenhierarchie um echte Teilmengenbeziehungen. Das heißt, wenn $\mathscr{L}_i$ $(i = 0, 1, 2, 3)$ die Klasse aller zugehörigen Sprachen vom Typ i bezeichnet, gilt sogar $\mathscr{L}_0 \supsetneq \mathscr{L}_1 \supsetneq \mathscr{L}_2 \supsetneq \mathscr{L}_3$. Dies lässt sich durch entsprechende trennende Beispiele nachweisen. Durch eine kontextfreie Grammatik mit den Produktionsregeln $S \to 0S1|01$ findet man beispielsweise die kontextfreie Sprache $\{0^n1^n|n \geq 0\}$. Dass diese Sprache nicht regulär ist, zeigt man mithilfe des Pumping-Lemmas für reguläre Sprachen (siehe Satz 4.2.12).

Also werden die kontextfreien und regulären Sprachen durch $\{0^n1^n | n \geq 0\}$ getrennt und es gilt $\mathscr{L}_2 \supsetneq \mathscr{L}_3$. Für eine genaue Argumentation dieses Beispiels siehe [6, 69-72]. Die Sprache $\{0^n1^n2^n | n \geq 1\}$ trennt wiederum die kontextsensitiven und kontextfreien Sprachen. Dass diese Sprache nicht kontextfrei ist, werden wir in Propositon 4.3.17 mithilfe des Pumping-Lemmas für kontextfreie Sprachen (siehe Satz 4.3.14) zeigen. Somit gilt auch $\mathscr{L}_1 \supsetneq \mathscr{L}_2$.

Bemerkung 3.2.6 In der Informatik gibt es neben der Unterteilung gemäß Chomsky-Hierarchie noch andere Sprachklassen. Man kann demnach auch zwischen *abzählbaren*, *aufzählbaren* und *entscheidbaren* Sprachen unterscheiden. Bei den *(rekursiv) aufzählbaren* (manchmal auch *semientscheidbaren* oder *erkennbaren*) Sprachen handelt es sich genau um die Typ-0-Sprachen. Ohne hier näher auf diese Typen eingehen zu wollen, bleibt zu erwähnen, dass es eines deutlich komplizierteren Beispiels bedarf, um zu verdeutlichen, dass $\mathscr{L}_0 \supsetneq \mathscr{L}_1$ gilt (siehe [6, 36-37]).

Da eine Sprache normalerweise von mehreren Grammatiken erzeugt werden kann, ist es im Allgemeinen nicht leicht zu entscheiden, zu welcher Klasse eine Sprache gehört. Demnach muss man nur eine erzeugende CF-Grammatik finden, um zu zeigen, dass eine gegebene Sprache kontextfrei ist. Aber um zu beweisen, dass eine Sprache *nicht* kontextfrei ist, muss man zeigen, dass keine ihrer erzeugenden Grammatiken eine CF-Grammatik ist.

Der nachfolgende Satz ist noch eine alternative Definition für Typ-1- und Typ-2-Grammatiken:

Satz 3.2.7 *Sei $G = (V_N, V_T, P, S)$ eine Grammatik. Dann gilt:*

(i) Ist G eine CS-Grammatik, dann gilt für jede Produktionsregel $u \to v$ in G, dass

$$|u| \leq |v|. \tag{3.7}$$

(ii) Ist G eine CF-Grammatik, dann gilt für jede Produktionsregel $u \to v$ in G, dass

$$|u| = 1. \tag{3.8}$$

Beweis. (i) Sei G eine CS-Grammatik. Dann hat jede Produktion in G die Form $f\alpha g \to fug$, wobei $u \neq \varepsilon$ und somit $|u| \geq 1$. Es folgt $|fug| \geq |f| + 1 + |g| = |f\alpha g|$ und somit auch (3.7).
Umgekehrt, wenn (3.7) für jede Produktionsregel gilt, kann man den Effekt der Regel $u \to v$ mit schrittweisem Ersetzen der Buchstaben in u durch Buchstaben in v erreichen. Man muss darauf achten, dass zuletzt eine (neue) Variable bleibt. Ein typisches Beispiel wäre, wenn $u = u_1\alpha u_2 u_3$ und $v = v_1 \dots v_5$, dann wird $u \to v$ durch die Regeln $u_1\alpha u_2 u_3 \to v_1\beta u_2 u_3 \to v_1\beta u_2 v_5 \to v_1\beta v_4 v_5 \to v$, wobei β nicht woanders vorkommt.
(ii) Aus der Definition einer CF-Sprache ist klar, dass ihre Produktionsregeln (3.8) erfüllen. □

Bemerkung 3.2.8 Es gilt umgekehrt, falls (3.7) (bzw. (3.8)) von G erfüllt werden, dann existiert eine erzeugende CS-Grammtik (bzw. erzeugende CF-Grammatik) für $L(G)$, siehe [3, 401-402].

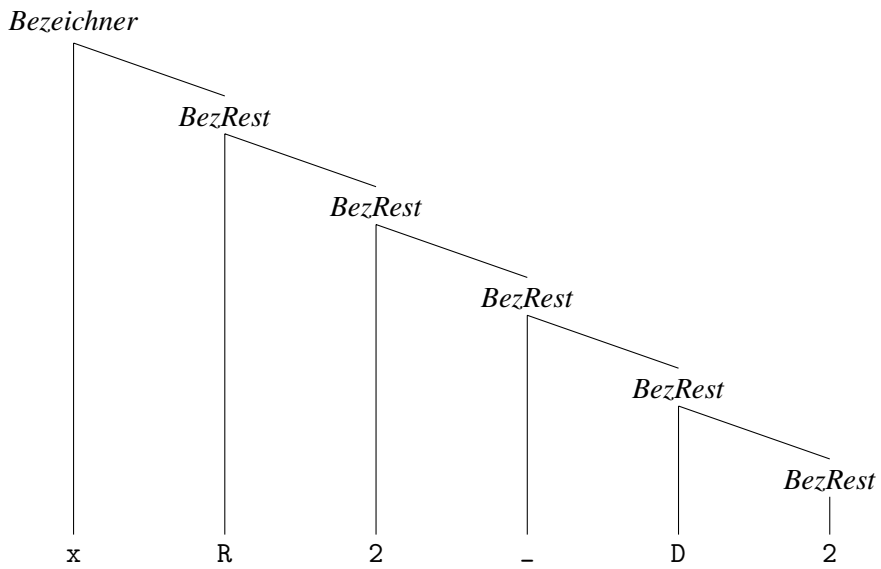
Beispiel 3.2.9 Wir kehren zurück zu Beispiel 3.1.9, eine Grammatik für Bezeichner der Programmiersprache C. Die Produktionsregeln sind:

$$
\begin{aligned}
\textit{Bezeichner} &\to \textit{Buchstabe} \mid \texttt{\_} \mid \textit{Buchstabe BezRest} \mid \texttt{\_}\ \textit{BezRest} \\
\textit{BezRest} &\to \textit{Buchstabe BezRest} \mid \texttt{\_}\ \textit{BezRest} \mid \textit{Ziffer BezRest} \mid \textit{Buchstabe} \mid \texttt{\_} \mid \textit{Ziffer} \\
\textit{Buchstabe} &\to \texttt{A} \mid \texttt{B} \mid \ldots \mid \texttt{Z} \mid \texttt{a} \mid \texttt{b} \mid \ldots \mid \texttt{z} \\
\textit{Ziffer} &\to \texttt{0} \mid \texttt{1} \mid \ldots \mid \texttt{9}
\end{aligned}
$$

Anhand der Produktionen sieht man, dass diese Grammatik kontextfrei, aber nicht regulär ist. Es stellt sich die Frage, ob es auch eine reguläre Grammatik für diese Sprache gibt. Wir geben eine Grammatik mit folgenden Regeln an:

$$
\begin{aligned}
\textit{Bezeichner} \to\ & \texttt{A} \mid \texttt{B} \mid \ldots \mid \texttt{Z} \mid \texttt{a} \mid \texttt{b} \mid \ldots \mid \texttt{z} \mid \texttt{\_} \mid \\
& \texttt{A}\ \textit{BezRest} \mid \texttt{B}\ \textit{BezRest} \mid \ldots \mid \texttt{Z}\ \textit{BezRest} \mid \\
& \texttt{a}\ \textit{BezRest} \mid \texttt{b}\ \textit{BezRest} \mid \ldots \mid \texttt{z}\ \textit{BezRest} \mid \\
& \texttt{\_}\ \textit{BezRest} \\
\textit{BezRest} \to\ & \texttt{A} \mid \texttt{B} \mid \ldots \mid \texttt{Z} \mid \texttt{a} \mid \texttt{b} \mid \ldots \mid \texttt{z} \mid \texttt{\_} \mid \texttt{0} \mid \texttt{1} \mid \ldots \mid \texttt{9} \mid \\
& \texttt{A}\ \textit{BezRest} \mid \texttt{B}\ \textit{BezRest} \mid \ldots \mid \texttt{Z}\ \textit{BezRest} \mid \\
& \texttt{a}\ \textit{BezRest} \mid \texttt{b}\ \textit{BezRest} \mid \ldots \mid \texttt{z}\ \textit{BezRest} \mid \\
& \texttt{\_}\ \textit{BezRest} \mid \\
& \texttt{0}\ \textit{BezRest} \mid \texttt{1}\ \textit{BezRest} \mid \ldots \mid \texttt{9}\ \textit{BezRest} \mid
\end{aligned}
$$

Man sieht, dass diese Grammatik regulär ist, und somit ist die Sprache für Bezeichner der Programmiersprache C sogar regulär.



Bemerkung 3.2.10 Die Sprachen in den Beispielen 3.1.17, 3.1.18, 3.1.19, 3.1.22 und 3.1.23 sind regulär. Bei den Sprachen der Beispiele 3.1.20 und 3.1.21 handelt es sich um kontextfreie Sprachen, die nicht regulär sind.

Beispiel 3.2.11 Die Sprache $L = \{0^n 1^n 2^n | n \geq 0\}$ wird von der Grammatik $G = (V_N, V_T, P, S)$ erzeugt, mit $V_N = \{S, \lambda, \mu\}$, $V_T = \{0, 1, 2\}$ und folgenden Produktionsregeln:

$$\begin{aligned} S &\to 0S\lambda\mu \\ S &\to 01\mu \\ S &\to \varepsilon \\ \mu\lambda &\to \lambda\mu \\ 1\lambda &\to 1^2 \\ \mu &\to 2 \end{aligned}$$

Die ersten beiden Regeln erzeugen alle Wörter der Form $0^n 1\mu(\lambda\mu)^{n-1}$. Die vierte Regel bewegt die λ vorbei an den μ neben die Einser. Die fünfte Regel ersetzt jedes λ mit einem Einser. Die letzte Regel ersetzt jedes μ mit einem Zweier. Um die gleiche Sprache ohne ε zu erhalten, lässt man einfach die Produktion $S \to \varepsilon$ weg. Diese Sprache ist kontextsensitiv, aber nicht kontextfrei, das werden wir später in Proposition 4.3.17 beweisen.

4 Automaten

Logische Maschinen sind ein nützliches Werkzeug, wenn man sich mit rekursiven Funktionen beschäftigt. Tatsächlich ist es so, dass bestimmte Maschinen und bestimmte Sprachen bzw. Grammatiken miteinander korrespondieren. Der britische Mathematiker ALAN TURING entwarf 1936 die Turingmaschine, ein wichtiges Modell der theoretischen Informatik. Turingmaschinen korrespondieren mit Typ-0-Grammatiken, daher erwartet man, dass die eingeschränkteren Grammatiken der Typen 1-3 gleichermaßen mit spezielleren Maschinen korrespondieren. Dies ist tatsächlich der Fall, nachfolgend werden einige dieser spezielleren Maschinen vorgestellt. Turingmaschinen sind nicht Teil dieser Arbeit.
Grundlage des gesamten Kapitels ist [3, 403-411], sonstige Quellen werden am Anfang der einzelnen Abschnitte oder unmittelbar bei den betreffenden Aussagen gegeben.

4.1 Abstrakte Automaten und Automatenarten

In unserem Alltag begegnen uns Automaten und Maschinen in den verschiedensten Formen. Einen *Automaten* kann man sich einfach als einen Kasten vorstellen, der verschiedene *Zustände* annehmen kann, die durch eine *Eingabe* (bzw. *Input*) in andere Zustände übergeführt werden können. Zusätzlich kann der Automat dann noch eine *Ausgabe* (bzw. *Output*) erzeugen. Neben [3, 403-411] halten sich die Definitionen und Beispiele dieses Abschnitts an [9, 35-41].

Die nachfolgenden beiden Definitionen beschreiben das mathematische Modell eines Automaten.

Definition 4.1.1 Ein *Halbautomat* ist ein Tripel $H = (Z, V, \delta)$. Dabei gilt:

(i) Z ist eine nichtleere Menge, die *Zustandsmenge* heißt.

(ii) V ist eine nichtleere Menge, die *Eingabealphabet* (auch *Inputalphabet*) heißt.

(iii) $\delta : Z \times V \to Z$ ist eine Funktion, die *Zustandsüberführungsfunktion* heißt.

Definition 4.1.2 Ein *Automat* ist ein Quintupel $M = (Z, V, W, \delta, \lambda)$. Dabei gilt:

(i) (Z, V, δ) ist ein Halbautomat.

(ii) W ist eine nichtleere Menge, die *Ausgabealphabet* (auch *Outputalphabet*) heißt.

(iii) $\lambda : Z \times V \to W$ ist eine Funktion, die *Ausgabefunktion* (auch *Outputfunktion*) heißt.

Bemerkung 4.1.3 Sind $z \in Z$ und $v \in V$, dann interpretiert man $\delta(z, v) \in Z$ als den Folgezustand, in den z durch Einwirkung von v übergeführt wird. $\lambda(z, v) \in W$ ist dann der Output von z bei Einwirkung von v. Das heißt, befindet sich der Automat im Zustand $z \in Z$ und bekommt den Input $v \in V$, so ändert er sich in den Zustand $\delta(z, v) \in Z$ und spuckt dabei $\lambda(z, v) \in W$ aus.

Definition 4.1.4 Ein (Halb-)Automat heißt *endlich*, wenn alle auftretenden Mengen Z, V, W endlich sind. Endliche Automaten heißen auch *Mealy-Automaten* (nach GEORGE H. MEALY).

Definition 4.1.5 Ist ein spezieller Zustand $z_0 \in Z$ vorgegeben, so nennt man den (Halb-)Automaten *initial* und z_0 den *Anfangszustand*. Man schreibt dafür (Z, V, δ, z_0). Automaten, bei denen λ nur von z abhängt, heißen *Moore-Automaten* (nach EDWARD F. MOORE).

Definition 4.1.6 Einen (Halb-)Automaten nennt man *nichtdeterministisch*, falls es sich bei δ und λ um Relationen handelt. Sind δ und λ Funktionen, wie in den Definitionen 4.1.1 und 4.1.2, dann heißt der (Halb-)Automat *deterministisch*.

Beispiel 4.1.7 In diesem Beispiel betrachtet man eine (zugegebenermaßen vereinfacht dargestellte) Situation. Man geht davon aus, dass eine Frau stets entweder verärgert, gleichgültig oder freudig gestimmt ist. Die Regierung macht entweder nichts, kürzt dem Frauenministerium das Budget oder behandelt die Themen des Frauenvolksbegehrens. Macht die Regierung nichts, ändert sich die Stimmung der Frau nicht. Wird das Budget des Frauenministeriums gekürzt, senkt das die Stimmung der Frau um eine Stufe. Werden die Themen des Frauenvolksbegehrens behandelt, stimmt das die Frau freudig.
Diese Situation soll nun als ein Halbautomat (Z, V, δ) mit $Z = \{z_1, z_2, z_3\}$ und $V = \{v_1, v_2, v_3\}$ beschrieben werden. Dabei gelten folgende Bedeutungen:

z_1 … Die Frau ist verärgert.
z_2 … Die Frau ist gleichgültig.
z_3 … Die Frau ist freudig gestimmt.
v_1 … Die Regierung macht nichts.
v_2 … Die Regierung kürzt dem Frauenministerium das Budget.
v_3 … Die Regierung behandelt die Themen des Frauenvolksbegehrens.

Die Funktion δ wird durch eine Wertetabelle angegeben, die man *Überführungstabelle* nennt.

δ	v_1	v_2	v_3
z_1	z_1	z_1	z_3
z_2	z_2	z_1	z_3
z_3	z_3	z_2	z_3

Beispiel 4.1.8 Nun soll der Halbautomat aus Beispiel 4.1.7 zu einem Automaten $(Z, V, W, \delta, \lambda)$ erweitert werden. Die Lage der Dinge sei wie vorher, jedoch kommen Outputs $W = \{w_1, w_2\}$ hinzu. Dabei gelten folgende Bedeutungen:

w_1 … Die Frau schreit wütend auf.
w_2 … Die Frau sagt nichts.

Die Frau schreit nur dann wütend auf, wenn sie verärgert ist und die Regierung dem Frauenministerium das Budget kürzt. In allen anderen Fällen sagt die Frau nichts.
Die Funktion λ wird durch eine Wertetabelle angegeben, die man *Ergebnistabelle* nennt.

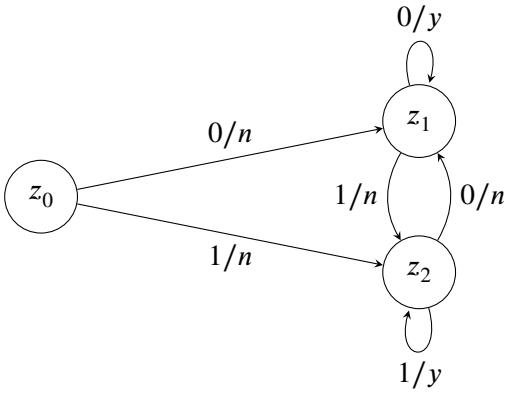
λ	v_1	v_2	v_3
z_1	w_2	w_1	w_2
z_2	w_2	w_2	w_2
z_3	w_2	w_2	w_2

Beispiel 4.1.9 Gegeben sei ein initialer endlicher Automat $M = (Z, V, W, \delta, \lambda, z_0)$, also ein initialer Mealy-Automat. Dabei gilt $Z = \{z_0, z_1, z_2\}$, $V = \{0, 1\}$ und $W = \{y, n\}$. Die Funktionen δ, λ kann man folgenden Wertetabellen entnehmen:

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

λ	0	1
z_0	n	n
z_1	y	n
z_2	n	y

Sind die beiden letzten Eingabesymbole gleich, dann ist die Ausgabe des Automaten y. Die nachfolgende Abbildung zeigt den sogenannten *Zustandsgraphen* des Automaten.



Bemerkung 4.1.10 In einem Zustandsgraphen wie in Beispiel 4.1.9 stellt man die Zustände $z_1, \ldots, z_k$ zunächst als Kreise dar. Dann zieht man einen mit v_i versehenen Pfeil von z_r nach z_s, falls $\delta(z_r, v_i) = z_s$ gilt. Bei einem Automaten markiert man den Pfeil zusätzlich mit dem Ergebnis $\lambda(z_r, v_i) = w_j$. In unserem konkreten Fall gilt z. B. $\delta(z_0, 0) = z_1$ und $\lambda(z_0, 0) = n$, daher wurde der Pfeil von z_0 nach z_1 mit $0/n$ markiert.

Bemerkung 4.1.11 Im Allgemeinen wird der Input nicht nur aus einem Symbol bestehen, sondern aus einem Wort über dem Eingabealphabet V, also einem $x \in V^*$. Dann liest der Automat das Wort x Buchstabe für Buchstabe und liefert in Abhängigkeit von λ jeweils einen Output aus dem Ausgabealphabet W. Währenddessen werden in Abhängigkeit von δ verschiedene Zustände durchlaufen. Der Output ist daher ein Wort über dem Ausgabealphabet W, also ein $y \in W^*$, das die gleiche Länge wie x hat. Seien nun die Abbildungen $\delta' : Z \times V^* \to Z$ und $\lambda' : Z \times V^* \to W^*$ so definiert, dass

$$\delta'(z, \varepsilon) = z, \quad \delta'(z, ux) = \delta(\delta'(z, u), x), \tag{4.1}$$

$$\lambda'(z, \varepsilon) = \varepsilon, \quad \lambda'(z, ux) = \lambda'(z, u)\lambda(\delta'(z, u), x), \tag{4.2}$$

wobei $z \in Z, x \in V, u \in V^*$. Es ist klar, dass die rekursiv definierten Funktionen δ' und λ' die ursprünglichen Funktionen δ und λ entsprechend für Wörter erweitern. Daher kann man unmissverständlich die Striche weglassen und es ist klar, dass

$$\delta(z, \varepsilon) = z, \quad \delta(z, uv) = \delta(\delta(z, u), v) \quad (\text{wobei } z \in Z,\ u, v \in V^*) \tag{4.3}$$

dann eine Operation des freien Monoids V^* auf Z ist. Das funktioniert auch ohne Bedingungen an die Funktion δ.

Bemerkung 4.1.12 Das heißt, durch die Menge aller Quadrupel der Form $(z, x, \lambda(z, x), \delta(z, x))$ ist ein Automat vollständig bestimmt. Einen Automaten M kann man dann auch als Teilmenge $P = P(M) \subseteq Z \times V \times W \times Z$ auffassen. Elemente aus P nennt man *Kanten*, und jede Kante (z, x, y, z') hat dann einen Anfangszustand z, einen Input x, einen Output y und einen Endzustand z'. Zwei Kanten heißen *aufeinanderfolgend* (auch *konsekutiv*), falls der Endzustand der ersten Kante der Anfangszustand der zweiten Kante ist. Unter einem *Pfad* versteht man eine Folge $u = (u_1, \dots, u_n)$ von aufeinanderfolgenden Kanten $u_i = (z_{i-1}, x_i, y_i, z_i)$. Man nennt n seine *Länge*, z_0 den Anfangszustand, z_n den Endzustand, $x_1, \dots, x_n$ das *Eingabe-Label* und $y_1, \dots, y_n$ das *Ausgabe-Label*.

Bemerkung 4.1.13 Turingmaschinen sind ebenfalls abstrakte Automaten und am Anfang wurde erwähnt, dass Turingmaschinen mit Typ-0-Grammatiken korrespondieren. Verschiedene Automatentypen korrespondieren mit bestimmten Sprachklassen. Turingmaschinen erkennen rekursiv-aufzählbare Sprachen. Linear beschränkte Automaten sind eine spezielle Art von Turingmaschinen, die kontextsensitive Sprachen erkennen. Kellerautomaten erkennen kontextfreie und Akzeptoren erkennen reguläre Sprachen. Diese Beziehungen werden hier nochmals dargestellt:

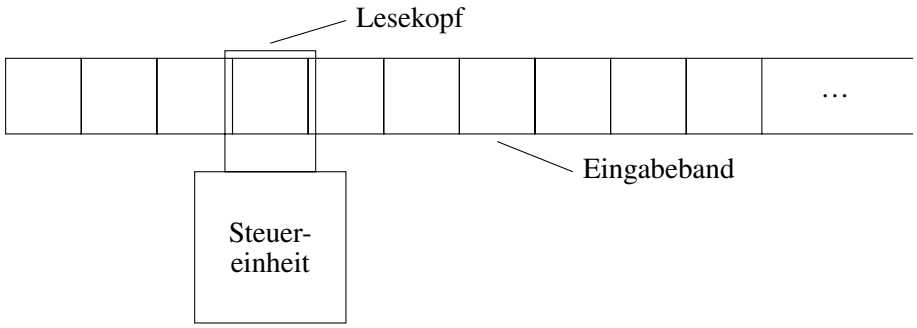
$$\begin{aligned}
\text{Turingmaschinen} &\leftrightarrow \text{rekursiv-aufzählbare Sprachen (Typ 0)} \\
\text{linear beschränkte Automaten} &\leftrightarrow \text{kontextsensitive Sprachen (Typ 1)} \\
\text{Kellerautomaten} &\leftrightarrow \text{kontextfreie Sprachen (Typ 2)} \\
\text{endliche Akzeptoren} &\leftrightarrow \text{reguläre Sprachen (Typ 3)}
\end{aligned}$$

Die nächsten beiden Abschnitte widmen sich den endlichen Akzeptoren und den Kellerautomaten. Die Äquivalenz endlicher Akzeptoren und regulärer Sprachen wird im Satz 4.2.7 bewiesen. Satz 4.3.5 behandelt die äquivalente Beziehung zwischen Kellerautomaten und kontextfreien Sprachen.

4.2 Akzeptoren und reguläre Sprachen

Nachdem die Begriffe des Halbautomaten bzw. Automaten (Definitionen 4.1.1 und 4.1.2) am Anfang des vorherigen Abschnitts gleich mithilfe des mathematischen Modells eingeführt wurden, soll an dieser Stelle ein kurzer Schritt zurück gemacht werden. Die Arbeitsweise des endlichen Akzeptors, eines speziellen Automaten, wird mit einem genau ausformulierten Ablauf und einer Skizze erklärt. Zusätzlich zu [3, 403-411] bezieht sich dieser Abschnitt auf [6, 49-80].

Endliche Akzeptoren kann man sich wie Scanner vorstellen, da ihre Arbeitsweisen sich ähneln. Der Akzeptor bekommt ein Eingabewort, das auf ein einseitig unendliches Eingabeband geschrieben ist. Das Wort wird zeichenweise mithilfe eines Lesekopfes bis zum Wortende gelesen. Nach dem Lesen jedes Zeichens kann der endliche Akzeptor den Zustand der Steuereinheit ändern. Somit besteht die Möglichkeit, gewisse Informationen zu speichern.



Die Abarbeitung eines Eingabewortes wird auf folgende Art und Weise durchgeführt:

- Zu Beginn befindet sich die Steuereinheit im Anfangszustand. Der Lesevorgang beginnt mit dem von links gesehen ersten Buchstaben des Wortes.
- Der Lesekopf geht auf dem Eingabeband schrittweise nach rechts und liest die Buchstaben, die auf den einzelnen Feldern stehen.
- In jedem Schritt kann die Steuereinheit, abhängig vom augenblicklichen Zustand und dem gelesenen Buchstaben, den Zustand ändern.
- Das Wort wird von dem endlichen Akzeptor erkannt, wenn sich die Steuereinheit nach dem vollständigen Lesen des Wortes in einem gewissen Endzustand befindet.

Aufbauend auf die bereits eingeführten Definitionen soll nun das mathematische Modell definiert werden. Ein endlicher Akzeptor ist ein spezieller initialer Mealy-Automat, der keine Ausgabe liefert. Stattdessen gibt es eine ausgewiesene Menge $F \subseteq Z$ an Endzuständen. Das Lesen eines Eingabewortes führt dann entweder zu einem Zustand $z \in F$ (Akzeptanz) oder zu einem Zustand $z \notin F$ (Ablehnung).

Definition 4.2.1 Ein *endlicher Akzeptor* ist ein Quintupel $A = (Z, V, \delta, z_0, F)$. Dabei gilt:

(i) (Z, V, δ) ist ein endlicher Halbautomat.

(ii) $z_0 \in Z$ ist ein *Anfangszustand* (auch *Initialzustand* genannt).

(iii) $F \subseteq Z$ ist eine *Menge an Endzuständen* (auch *Finalzustandsmenge* genannt). Ein Endzustand wird auch als *akzeptierender Zustand* bezeichnet.

Definition 4.2.2 Sei $A = (Z, V, \delta, z_0, F)$ ein endlicher Akzeptor. Das Wort $x \in V^*$ sei auf das Eingabeband geschrieben, beginnend auf dem ersten Feld von links. A starte im Anfangszustand z_0 über dem ersten Feld des Eingabebands.

(i) Das Wort x wird von A *erkannt (akzeptiert)*, falls A nach dem vollständigen Lesen von x einen Endzustand $z \in F$ angenommen hat.

(ii) Die von A *erkannte (akzeptierte, dargestellte) Sprache* ist die Menge

$$L(A) = \{x \in V^* | \ x \text{ wird von } A \text{ erkannt}\}.$$

Bemerkung 4.2.3 Stoppt A frühzeitig, was unter gewissen Umständen vorkommen kann, so wird das Wort x nicht erkannt. Das Leerwort ϵ wird genau dann erkannt, wenn $z_0 \in F$.

Beispiel 4.2.4 Sei $A = (Z, V, \delta, z_0, F)$ ein Akzeptor mit $Z = \{z_0, z_1, z_2\}$ und $V = \{0, 1\}$. Dabei ist z_0 der Anfangszustand und z_2 der einzige Endzustand, also $F = \{z_2\}$. Die Überführungsfunktion δ wird in der Tabelle dargestellt. Danach sollen zwei Wörter bearbeitet werden.

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

Die Abarbeitung des Wortes $x_1 = 010110010$ sieht folgendermaßen aus:

0	1	0	1	1	0	0	1	0	ϵ	$\cdots$
z_0	z_0	z_1	z_1	z_2	z_1	z_1	z_1	z_2	z_2	

Die Abarbeitung des Wortes $x_2 = 010110011$ sieht folgendermaßen aus:

0	1	0	1	1	0	0	1	1	ϵ	$\cdots$
z_0	z_0	z_1	z_1	z_2	z_1	z_1	z_1	z_2	z_1	

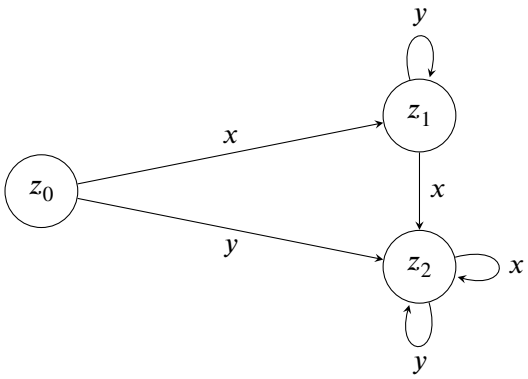
Das Wort x_1 wird erkannt, da A in einem Endzustand (nämlich $z_2 \in F$) stoppt. Das Wort x_2 wird nicht erkannt, da A in diesem Fall in keinem Endzustand stoppt (nämlich $z_1 \notin F$).
Nach der Anfangsphase wechselt der Automat nur noch zwischen den Zuständen z_1 und z_2. Ein Zustandswechsel findet statt, wenn 1 gelesen wird. Wenn 0 gelesen wird, hat dies keine Auswirkung auf den Zustand. Der Automat befindet sich immer im Zustand z_1, wenn eine ungerade Anzahl Einser gelesen wurde und im Zustand z_2 bei einer geraden Anzahl Einser. Somit gilt:

$$L(A) = \{x \in V^* | \text{ Anzahl der Einser in } x \text{ ist gerade und} \geq 2\}.$$

Beispiel 4.2.5 Gegeben sei ein Akzeptor $A = (Z, V, \delta, z_0, F)$ mit $Z = \{z_0, z_1, z_2\}$ und $V = \{x, y\}$. Die Überführungsfunktion δ wird in der Tabelle dargestellt.

δ	x	y
z_0	z_1	z_2
z_1	z_2	z_1
z_2	z_2	z_2

Der Zustandsgraph sieht folgendermaßen aus:



Falls $F = \{z_1\}$, dann ist die akzeptierte Sprache $L(A) = \{xy^n | n \geq 0\}$ (siehe Beispiel 3.1.18). Falls $F = \{z_2\}$, dann ist $L(A) = \{xy^n xu | (n \geq 0) \wedge (u \in V^*)\} \cup \{yu | u \in V^*\}$.

Bemerkung 4.2.6 In Beispiel 4.2.5 wurde ein Akzeptor vorgestellt, der in Abhängigkeit von seinen Endzuständen zwei verschiedene Sprachen akzeptiert. Nun sollen diese beiden Sprachen mithilfe einer anderen Notation, nämlich der regulären Ausdrücke, dargestellt werden. Dieser Formalismus hat sich als äußerst nützlich erwiesen.

$$\{xy^n | n \geq 0\} = xy^* \tag{4.4}$$

$$\{xy^n xu | (n \geq 0) \wedge (u \in V^*)\} \cup \{yu | u \in V^*\} = xy^* xV^* \cup yV^* \tag{4.5}$$

Bei dem Operator $\cup$ handelt es sich um die Vereinigung von Sprachen, die aus der Mengenlehre wohlbekannt ist. Der Operator * heißt *Kleene-Stern* (nach STEPHEN C. KLEENE), der bei der Bildung von regulären Ausdrücken verwendet wird. Er beschreibt die Menge aller Zeichenreihen, die durch die Verkettung einer beliebigen Anzahl von Zeichenreihen der Menge gebildet werden können, wobei Wiederholungen zulässig sind. Wenn $L = \{0, 1\}$, dann enthält L^* alle Zeichenreihen aus Nullern und Einsern. Wenn $M = \{0, 11\}$, dann enthält M^* alle Zeichenreihen aus Nullern und Einsern, in denen die Einser paarweise auftreten, z. B. sind 011, 11110 oder ε Elemente von M^*. Keine Elemente von M^* sind z. B. 01011 oder 101. Bereits bei der Definition des freien Monoids (siehe Definition 2.1.8) wurde der Sternoperator eingeführt. Der Kleene-Stern ist im Grunde genommen dasselbe, bis auf eine subtile Typenunterscheidung. An dieser Stelle sollen reguläre Ausdrücke lediglich mithilfe der beiden Beispiele (4.4) und (4.5) veranschaulicht werden. Im Laufe der Arbeit sollte es eigentlich zu keinen Missverständnissen kommen. Für genauere Erläuterungen zu regulären Ausdrücken und weiteren Beispielen siehe [6, 72-80] oder [8, 93-134].

Sowohl Grammatiken als auch Automaten, wie zum Beispiel endliche Akzeptoren, beschreiben Sprachen, allerdings unter einem jeweils anderen Gesichtspunkt. Wir werden nun zeigen, dass die endlichen Akzeptoren genau die Sprachen akzeptieren, die von den regulären Grammatiken erzeugt werden, also die regulären Sprachen. Dabei ist unerheblich, ob man deterministische oder nichtdeterministische endliche Akzeptoren verwendet, weil beide Konzepte gleichmächtig sind.

Der nachfolgende Satz ist von zentraler Bedeutung für den Zusammenhang zwischen regulären Sprachen und endlichen Akzeptoren.

Satz 4.2.7 *Sei V ein Alphabet und $L \subseteq V^*$. Dann sind die folgenden Aussagen äquivalent:*

(i) *L ist regulär.*

(ii) *Es gibt einen nichtdeterministischen endlichen Akzeptor, der L erkennt.*

(iii) *Es gibt einen deterministischen endlichen Akzeptor, der L erkennt.*

Beweis. Die Äquivalenz der Aussagen wird durch einen Ringschluss gezeigt. Es handelt sich jeweils um konstruktive Beweise, das heißt, es werden nur die erforderlichen Konstruktionsalgorithmen angegeben. Dass die Ergebnisse das Gewünschte auch tatsächlich leisten, ist aus der Konstruktion ersichtlich, für genauere Begründungen siehe [13] oder [8].

Für (i) ⇒ (ii) wird aus einer regulären Grammatik ein nichtdeterministischer endlicher Akzeptor konstruiert. Sei $G = (V_N, V_T, P, S)$ eine reguläre Grammatik mit $L := L(G)$. Man definiert den nichtdeterministischen endlichen Akzeptor $N = (Z, V, \delta, z_0, F)$.

$$Z := V_N \cup \{E\},\ E \notin V_N$$

$$V := V_T$$

$$z_0 := S$$

$$F := \begin{cases} \{S, E\} \text{ falls } (S \to \varepsilon) \in P \\ \{E\} \text{ sonst} \end{cases}$$

δ sei gegeben durch: $(A, a, B) \in \delta \Leftrightarrow A \to aB \in P$ und $(A, a, E) \in \delta \Leftrightarrow A \to a \in P$ mit $A, B \in Z, a \in V$, dabei ist $(E, a, z) \notin \delta$ für alle $a \in V, z \in Z$

Für (ii) ⇒ (iii) wird aus einem nichtdeterministischen endlichen Akzeptor ein deterministischer endlicher Akzeptor konstruiert. Sei $N = (Z, V, \delta, z_0, F)$ ein nichtdeterministischer endlicher Akzeptor. Man definiert den deterministischen endlichen Akzeptor $D = (Z', V, \delta', z_0', F')$.

$$Z' := \mathcal{P}(Z) \text{ (Potenzmenge von } Z, \text{ d. h. } Q \in Z' \Leftrightarrow Q \in Z)$$

$$z_0' := \{z_0\}$$

$$F' := \{Q \in Z | \exists q \in Q \text{ mit } q \in F\}$$

$$\delta'(Q, a) := \{p \in Z | \exists q \in Q \text{ mit } (q, a, p) \in \delta\}$$

Für (iii) ⇒ (i) wird aus einem deterministischen endlichen Akzeptor eine reguläre Grammatik konstruiert. Sei $D = (Z, V, \delta, z_0, F)$ ein deterministischer endlicher Akzeptor. Man definiert $G = (V_N, V_T, P, S)$.

$$V_N := Z$$

$$V_T := V$$

$$S := z_0$$

P sei gegeben durch: $A \to aB \in P \Leftrightarrow (A, a) = B$ und $A \to a \in P \Leftrightarrow (A, a) \in F$

Spezialfall: Falls $z_0 \in F$, muss G so abgeändert werden, dass ε auch erzeugt werden kann. Man definiert dafür ein neues Startsymbol S', fügt die Regel $S' \to \varepsilon$ sowie für alle Regeln $S \to \alpha \in P$ die Regel $S' \to \alpha$ hinzu.
Somit sind die drei Aussagen (i), (ii) und (iii) äquivalent. □

Nachdem die Potenzmengenkonstruktion in (ii) ⇒ (iii) schnell zu komplizierten deterministischen Akzeptoren führt, sollen in den nächsten beiden Beispielen nur (i) ⇒ (ii) und (iii) ⇒ (i) veranschaulicht werden.

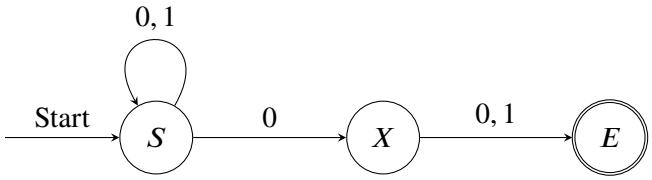
Beispiel 4.2.8 Sei $G = (V_N, V_T, P, S)$ eine reguläre Grammatik mit $V_N = \{S, X\}$, $V_T = \{0, 1\}$ und Produktionsregeln:

$$S \to 0S \mid 1S \mid 0X$$
$$X \to 0 \mid 1$$

Dann ist die erzeugte Sprache $L = \{x \in \{0,1\}^* | x \text{ hat an vorletzter Stelle einen Nuller}\}$. Nun soll ein nichtdeterministischer endlicher Akzeptor $N = (Z, V, \delta, z_0, F)$ aus dieser regulären Grammatik konstruiert werden. Dabei gilt $Z = V_N \cup \{E\} = \{S, X, E\}$, $V = V_T = \{0, 1\}$ und $z_0 = S$. Da $S \to \varepsilon \notin P$, setzt man $F = \{E\}$. Die Relation δ sei gegeben durch:

$$(S, 0, S), (S, 1, S), (S, 0, X), (X, 0, E), (X, 1, E).$$

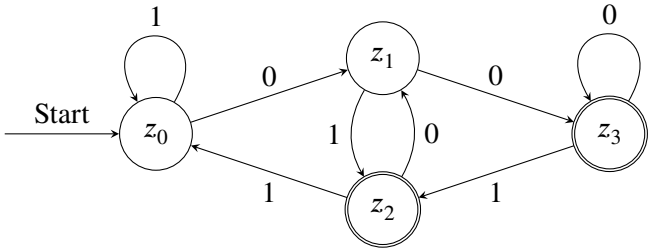
δ ergibt sich aus den Produktionsregeln P, wie im Beweisteil (i) ⇒ (ii) des Satzes 4.2.7 beschrieben wurde. Aus der Regel $S \to 0S$ ergibt sich beispielsweise $(S, 0, S) \in \delta$. Die Regel $X \to 1$ führt zu $(X, 1, E) \in \delta$. Der Zustandsgraph für $N = (Z, V, \delta, z_0, F)$ sieht dann so aus:



Somit wurde anhand gewisser Konstruktionsregeln aus der regulären Grammatik G ein nichtdeterministischer endlicher Akzeptor N kreiert.

Bemerkung 4.2.9 Aufbauend auf die Erklärung der Zustandsgraphen aus 4.1.10 soll hier noch erwähnt werden, dass Endzustände mit zwei konzentrischen Kreisen dargestellt werden.

Beispiel 4.2.10 Sei ein deterministischer endlicher Akzeptor D gegeben, der die gleiche Sprache wie $N = (Z, V, \delta, z_0, F)$ aus Beispiel 4.2.8 akzeptiert. Dass die gleiche Sprache erkannt wird, kann mithilfe des folgenden Zustandsgraphen nachvollzogen werden.



Nun soll aus diesem deterministischen Akzeptor eine reguläre Grammatik $G = (V_N, V_T, P, S)$ konstruiert werden. Dabei gilt $V_N = Z = \{z_0, z_1, z_2, z_3\}$, $V_T = V = \{0, 1\}$ und $S = z_0$. Die Produktionsregeln sind:

$$\begin{aligned} z_0 &\to 1z_0 \mid 0z_1, \\ z_1 &\to 1z_2 \mid 1 \mid 0z_3 \mid 0, \\ z_2 &\to 1z_0 \mid 0z_1, \\ z_3 &\to 1z_2 \mid 1 \mid 0z_3 \mid 0. \end{aligned}$$

Die Grammatikregeln P ergeben sich aus δ, wie im Beweisteil (iii) $\Rightarrow$ (i) des Satzes 4.2.7 beschrieben wurde. Aus $\delta(z_0, 1) = z_0$ ergibt sich beispielsweise die Regel $z_0 \to 1z_0$. Die Regel $z_1 \to 1$ entsteht aus $\delta(z_1, 1) = z_2$. Somit wurde aus dem deterministischen Akzeptor D die reguläre Grammatik G konstruiert.

Bemerkung 4.2.11 Allein durch die Betrachtung der beiden Zustandsgraphen in den Beispielen 4.2.8 und 4.2.10 sieht man, dass es oft einfacher ist, einen nichtdeterministischen endlichen Akzeptor zu konstruieren. In manchen Fällen ist jedoch die Handhabung deterministischer endlicher Akzeptoren praktischer.

Zum Abschluss des Abschnitts wird ein Werkzeug zum Nachweis der Nichtregularität von Sprachen vorgestellt. Dieser Satz wird *Pumping-Lemma für reguläre Sprachen* genannt. Die Darstellung, der Beweis von 4.2.12 und Beispiel 4.2.14 sind zurückzuführen auf [13, 91-93].

Satz 4.2.12 (Pumping-Lemma für reguläre Sprachen) *Es sei L eine reguläre Sprache. Dann gibt es eine Konstante $n \in \mathbb{N}_0$, die nur von der Sprache L abhängt, sodass sich jedes Wort $p \in L$ mit $|p| \geq n$ in $p = uvw$ zerlegen lässt, sodass gilt:*

(i) $|v| \geq 1$,

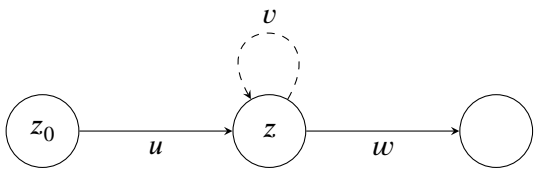
(ii) $|uv| \leq n$,

(iii) $uv^i w \in L$ *für alle* $i \geq 0$.

Beweis. Angenommen L ist regulär. Dann gibt es einen deterministischen endlichen Akzeptor $A = (Z, V, \delta, q_0, F)$, der L erkennt. Wir setzen die Konstante $n = |Z|$, wobei $|Z|$ die Anzahl der Zustände des Akzeptors A ist. Der deterministische endliche Akzeptor A durchläuft bei Abarbeitung von $p \in L$ genau $|p| + 1$ Zustände. Wenn nun p genügend lang ist, also $|p| \geq n$, so durchläuft der deterministische endliche Automat mehr als n Zustände.

Da es aber nur n Zustände gibt, wiederholt sich mindestens ein Zustand z bereits beim Lesen der ersten n Buchstaben von p, es wird also ein Kreis durchlaufen.

Es sei nun u Präfix von p, nach dessen Lesen A das erste Mal diesen Zustand z erreicht, und uv das Präfix von p, nach dessen Lesen A zum zweiten Mal z erreicht. Dann gilt: $|uv| \leq n$ und $|v| \geq 1$. Das Suffix w wird so gewählt, dass $p = uvw$ ist.



Da $p \in L$ ist, folgt $\delta(z_0, p) \in F$. Für das Eingabewort $uv^i w$ gilt dann $\delta(z_0, uv^i w) = \delta(z, v^i w) = \delta(z, vw) = \delta(z, w) \in F$. Also ist $uv^i w \in L$. Wegen $\delta(z, v) = z$ hat das Einfügen oder Löschen von v im Zustand z auf den erreichten Endzustand keinen Einfluss. □

Bemerkung 4.2.13 Das Pumping-Lemma hat seinen Namen daher, dass unter gewissen Umständen Teile von Wörtern $p = uvw$ einer regulären Sprache vervielfacht (also *aufgepumpt*) werden können, und die so entstandenen Wörter $uv^i w$ ebenfalls in der regulären Sprache enthalten sind.

Das nachfolgende Beispiel zeigt eine Anwendung des Pumping-Lemmas für reguläre Sprachen.

Beispiel 4.2.14 Gegeben sei $L = \{x^n y^n | n \geq 1\}$. Angenommen, L sei regulär und n sei die Konstante aus dem Pumping-Lemma. Dann lässt sich $x^n y^n$ in uvw mit $|uv| \leq n$ und $|v| \geq 1$ zerlegen. Es gilt $v = x^j$, $1 \leq j \leq n$. Dann folgt aber, dass $x^{n-j}(x^j)^i y^n \notin L$. Das bedeutet also, dass $L = \{x^n y^n | n \geq 1\}$ keine reguläre Sprache ist.

In Beispiel 4.2.14 sieht man, dass $L = \{x^n y^n | n \geq 1\}$ keine reguläre Sprache ist, das heißt L wird von keinem endlichen Akzeptor erkannt. Intuitiv merkt man, dass ein endlicher Akzeptor keine Möglichkeit hat, die Exponenten von x und y zu vergleichen. Um so einen Vergleich zu schaffen, braucht es eine gewisse Möglichkeit zur Speicherung von Informationen. Im nachfolgenden Abschnitt wird ein Automat vorgestellt, der eine Speichermöglichkeit hat, der sogenannte Kellerautomat. Es wird wird sich herausstellen, dass dieser genau mit den kontextfreien Sprachen korrespondiert.

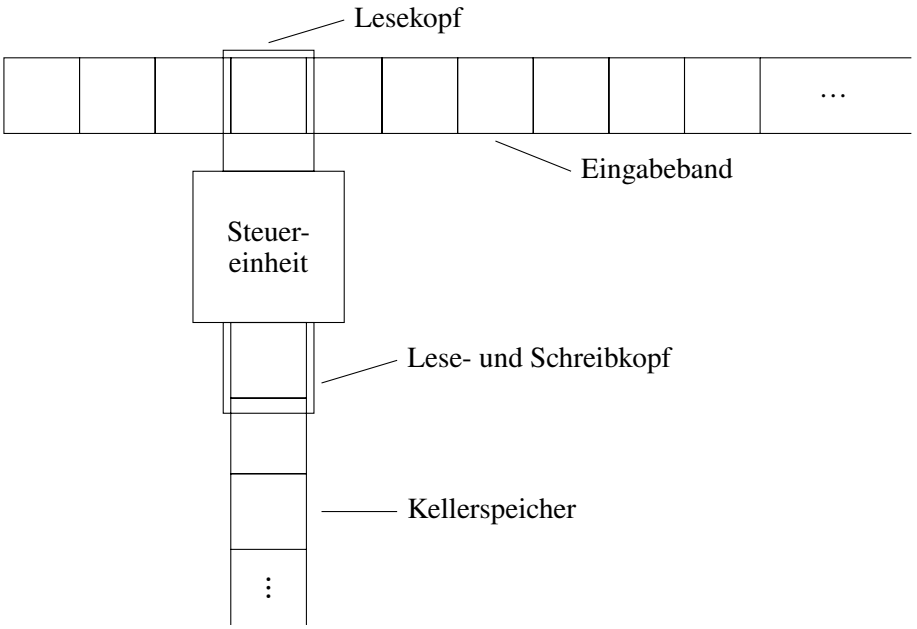
4.3 Kellerautomaten und kontextfreie Sprachen

Wie bereits erwähnt, können endliche Akzeptoren nur den Zustand in der Steuereinheit speichern. Um allerdings beliebig viele Informationen zu speichern, kann man einen zusätzlichen Speicher hinzufügen, der nicht in der Größe begrenzt ist. Es genügt sogar, den Speicher in Form eines Kellers zu verwalten, bei dem immer nur von einem Ende Information in den Speicher eingefügt bzw. aus dem Speicher entfernt werden können. Dieses erweiterte Automatenmodell ergibt das Konzept der Kellerautomaten, das nachfolgend vorgestellt wird. Zusätzlich zu [3, 403-411] stützt sich dieser Abschnitt auf [6, 112-122].

Die Abarbeitung eines Eingabewortes wird auf folgende Art und Weise durchgeführt:

- Zu Beginn befindet sich die Steuereinheit im Anfangszustand. Der Kellerspeicher ist leer, bis auf ein Anfangskellerzeichen. Das zu bearbeitende Wort steht auf dem Eingabeband, und der Lesevorgang beginnt mit dem von links gesehen ersten Buchstaben des Wortes.
- Der Lesekopf geht auf dem Eingabeband schrittweise nach rechts und liest die Buchstaben, die auf den einzelnen Feldern stehen.

- Die einzelnen Arbeitsschritte können auf zwei verschiedene Weisen geschehen:
  - Abhängig vom augenblicklichen Zustand, dem gelesenen Buchstaben auf dem Eingabeband sowie dem obersten Kellerzeichen geht die Steuereinheit in den nächsten Zustand über, schreibt ein Wort in den Keller und rückt den Lesekopf auf dem Eingabeband ein Feld weiter.
  - Abhängig vom augenblicklichen Zustand und dem obersten Kellerzeichen geht die Steuereinheit in den nächsten Zustand über, schreibt ein Wort in den Keller, rückt aber den Lesekopf auf dem Eingabeband nicht weiter. Dabei spielt der Buchstabe, der über dem der Lesekopf auf dem Eingabeband steht, keine Rolle.
- Das Eingabewort wird von dem Kellerautomaten erkannt, wenn er sich nach dem vollständigen Lesen des Wortes entweder in einem Endzustand befindet (bei Kellerautomaten mit Endzuständen) oder wenn der Keller vollständig leer ist (bei Kellerautomaten ohne Endzuständen).



Hier werden nur Kellerautomaten mit Endzuständen betrachtet, da sie eine direkte Verallgemeinerung endlicher Akzeptoren sind, lediglich ein Kellerspeicher wird hinzugefügt. Das mathematische Modell eines Kellerautomaten wird in der nächsten Definition festgehalten.

Definition 4.3.1 Ein Septupel $P = (Z, V, U, \delta, z_0, k_0, F)$ heißt *Kellerautomat* (oder *Push-Down-Automat*), wenn gilt:

(i) Z ist eine nichtleere, endliche Menge, die *Zustandsmenge* heißt.

(ii) V ist eine nichtleere, endliche Menge, die *Eingabealphabet* heißt.

(iii) U ist eine nichtleere, endliche Menge, die *Kelleralphabet* heißt.

(iv) $\delta : Z \times (V \cup \{\varepsilon\}) \times U \to Z \times U^*$ ist eine *Überführungsfunktion*.

(v) $z_0 \in Z$ ist der *Anfangszustand*.

(vi) $k_0 \in U$ ist das *Anfangskellerzeichen*.

(vii) $F \subseteq Z$ ist die *Menge der Endzustände* oder *Finalzustandsmenge*.

Definition 4.3.2 Sei $P = (Z, V, U, \delta, z_0, k_0, F)$ ein Kellerautomat. Das Wort $x \in V^*$ sei auf das Eingabeband geschrieben, beginnend auf dem ersten Feld von links. P starte im Anfangszustand z_0 über dem ersten Feld des Eingabebands. Der Keller sei leer, bis auf das Anfangskellerzeichen.

(i) Das Wort x wird von P *erkannt (akzeptiert)*, falls es in P eine Möglichkeit gibt, x vollständig zu lesen, und zwar so, dass P anschließend in einem Endzustand stoppt.

(ii) Die von P *erkannte (akzeptierte, dargestellte) Sprache* ist die Menge

$$L(P) = \{x \in V^* | \ x \text{ wird von } P \text{ erkannt}\}.$$

Bemerkung 4.3.3 Ganz analog zu (Halb-)Automaten und endlichen Akzeptoren unterscheidet man auch bei Kellerautomaten zwischen deterministischen und nichtdeterministischen Kellerautomaten, je nachdem ob es sich bei δ um eine Relation oder eine Funktion handelt (siehe auch Definition 4.1.6). In der Definition 4.3.1 wird ein deterministischer Kellerautomat angegeben. Anders als bei endlichen Akzeptoren ist bei Kellerautomaten die nichtdeterministische Variante mächtiger als die deterministische. Anhand eines Beispiels kann man begründen, dass nur mit Hilfe des Nichtdeterminismus alle kontextfreien Sprachen von einem Kellerautomaten erkannt werden können (siehe dazu auch [6, 114-119]).

Das nachfolgende Beispiel soll die Arbeitsweise eines nichtdeterministischen Kellerautomaten nochmals genauer erklären, daher wird jeder Schritt ausführlich beschrieben.

Beispiel 4.3.4 Es soll ein nichtdeterministischer Kellerautomat P angeben werden, der die Sprache $L = \{x^n y^n | n \geq 1\}$ akzeptiert (siehe Beispiel 4.2.14). Durch die Unbeschränktheit des Kellerspeichers können im Gegensatz zu endlichen Akzeptoren nun beliebig große n verglichen werden.

Sei $P = (Z, V, U, \delta, z_1, K, F)$ mit $Z = \{z_1, z_2, z_3\}$, $V = \{x, y\}$, $U = \{A, K\}$ und $F = \{z_3\}$. Die Überführungsrelation δ ist gegeben durch:

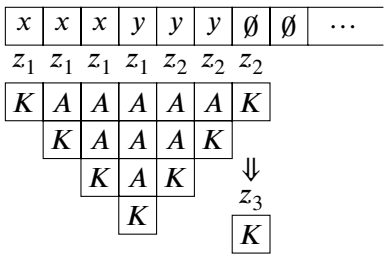
$$\begin{aligned} z_1 x K &\mapsto z_1 A K \\ z_1 x A &\mapsto z_1 A A \\ z_1 y A &\mapsto z_2 \varepsilon \\ z_2 y A &\mapsto z_2 \varepsilon \\ z_2 \varepsilon K &\mapsto z_3 K \end{aligned}$$

Anhand zweier Übergänge soll die Arbeitsweise von P genauer erklärt werden.

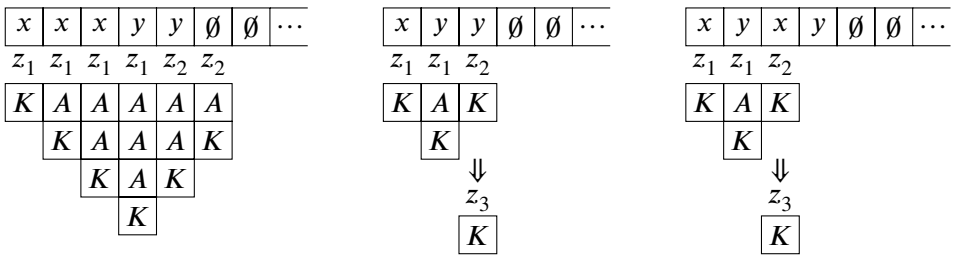
- Die Anweisung $z_1 x K \mapsto z_1 A K$ bedeutet: liest P im Zustand z_1 das Eingabezeichen x, und ist K das oberste Kellerzeichen, so geht P in den Zustand z_1 über und ersetzt K im Keller durch das Wort AK. Der Lesekopf auf dem Eingabeband wird dann ein Feld weitergerückt.
- Die Anweisung $z_2 \varepsilon K \mapsto z_3 K$ bedeutet: ist P im Zustand z_2, und ist K das oberste Kellerzeichen, so geht P ohne das Eingabezeichen zu lesen in den Zustand z_3 über und ersetzt K im Keller durch K. Hierbei wird der Lesekopf auf dem Eingabeband nicht weitergerückt. Dieser Übergang heißt auch *spontaner Übergang* oder *ε-Übergang*.

In beiden Fällen wird immer das oberste Kellerzeichen gelöscht. Allgemein gilt für ein Wort α, das neu in den Keller geschrieben wird, folgende Reihenfolge: Ist $\alpha = B_1, \ldots, B_n$, so kommt B_n zuerst in den Keller und B_1 zuletzt, sodass B_1 anschließend das oberste Kellerzeichen ist.

Ein nichtdeterministischer Kellerautomat stoppt, falls er für eine Situation, die durch den Zustand, das Zeichen unter dem Lesekopf auf dem Eingabeband sowie das oberste Kellerzeichen gegeben ist, keine Anweisung enthält. Insbesondere, wenn der Keller leer ist, dann gibt es definitionsgemäß keine Anweisung, und der Kellerautomat stoppt ebenfalls. Mithilfe spontaner Übergänge könnten Kellerautomaten (im Gegensatz zu endlichen Akzeptoren) auch unendlich weiterlaufen ohne zu stoppen.
Ein Eingabewort muss vollständig gelesen werden, damit es erkannt wird, und der Kellerautomat muss anschließend in einem Endzustand stoppen. Der einzige akzeptierte Endzustand des nichtdeterministischen Kellerautomaten P ist $F = \{z_3\}$.
Beispielsweise erkennt P dann das Wort $x^3y^3 = xxxyyy$. Nachfolgende Abbildung soll die Arbeitsweise von P darstellen, das Symbol $\emptyset$ steht in diesem Beispiel für ein Leerzeichen.



Die Wörter $x^3y^2 = xxxyy$, $xy^2 = xyy$ und $xyxy$ werden nicht erkannt, wie in den nachfolgenden Abbildungen gezeigt wird.



Das Wort $x^3y^2 = xxxyy$ wird nicht erkannt, weil P nicht im Endzustand z_3 stoppt. Die Wörter $xy^2 = xyy$ und $xyxy$ werden nicht erkannt, weil P nicht hinter dem Eingabewort stoppt.

Der Zusammenhang zwischen speziellen Automatentypen und gewissen Sprachklassen wurde bereits bei den endlichen Akzeptoren und regulären Sprachen genau argumentiert. Daher wird der folgende Satz ohne Beweis gegeben. Für eine genauere Argumentation siehe [6, 120-122].

Satz 4.3.5 *Sei V ein Alphabet und $L \subseteq V^*$. Dann sind die folgenden Aussagen äquivalent:*

(i) L ist kontextfrei.

(ii) Es gibt einen nichtdeterministischen Kellerautomaten, der L erkennt.

Bemerkung 4.3.6 Das heißt, die nichtdeterministischen Kellerautomaten erkennen genau die kontextfreien Sprachen.

Wenn man kontextfreie Grammatiken von überflüssigen Regeln befreien und die verbleibenden Regeln auf eine gewisse Standardform bringen möchte, bietet sich die Chomsky-Normalform an. Die nachfolgenden Definitionen, Sätze und deren Beweisführung finden sich in [13, 122-125].

Definition 4.3.7 Sei $G = (V_N, V_T, P, S)$ eine kontextfreie Grammatik und $x \in V_N \cup V_T$.

(i) x heißt *terminierend*, falls gilt $x \to\to w$ mit $w \in {V_T}^*$.

(ii) x heißt *erreichbar*, falls gilt $S \to\to uxv$ mit $u, v \in (V_N \cup V_T)^*$.

Eine Grammatik G heißt *reduziert*, falls in $V_N \cup V_T$ nur terminierende und erreichbare Symbole vorkommen.

Den Beweis des nächsten Satzes lässt sich in [13, 122-123] nachschlagen.

Satz 4.3.8 *Zu jeder kontextfreien Grammatik gibt es eine äquivalente reduzierte Grammatik.*

Definition 4.3.9 Eine kontextfreie Grammatik $G = (V_N, V_T, P, S)$ ist in *Chomsky-Normalform*, wenn alle Produktionen die folgenden Formen haben:

$$A \to BC \quad (\text{wobei } A, B, C \in V_N)$$
$$A \to a \quad (\text{wobei } A \in V_N, a \in V_T)$$

Satz 4.3.10 *Zu jeder kontextfreien Grammatik G mit erzeugter Sprache $L(G) \neq \emptyset$ gibt es eine äquivalente Grammatik G' in Chomsky-Normalform.*

Beweis. Aus der Grammatik $G = (V_N, V_T, P, S)$ wird die Grammatik $G' = (V'_N, V_T, P', S)$ in Chomsky-Normalform schrittweise konstruiert.

1. Wegen Satz 4.3.8, können wir von einer reduzierten Grammatik G ausgehen.

2. Regeln, die bereits der Chomsky-Normalform entsprechen, bleiben bestehen.
   In allen anderen Regeln werden sämtliche Terminalsymbole x durch neue Nichtterminale $x' \in (V'_N \backslash V_N)$ ersetzt und die Regeln $x' \to x$ werden ergänzt.

3. Regeln der Form $A_i \to A_{i+1}$ mit $A_i, A_{i+1} \in V'_N$ werden wie folgt eliminiert: Ist eine Folge $f := A_1 \to A_2 \to A_3 \to \cdots \to A_{k-1} \to A_k$ von Regeln mit $A_1, \ldots, A_k \in N'$ anwendbar, so muss schließlich eine Regel $A_k \to B_1 B_2 \ldots B_m$ mit $m \geq 2$, $B_1, B_2, \ldots, B_m \in N'$ oder eine Regel $A_k \to a$ mit $a \in V_T$ folgen. Das gleiche Ergebnis wird erzielt, wenn statt der Folge f nur die Regel $A_1 \to B_1 B_2 \ldots B_m$ oder $A_1 \to a$ verwendet wird. Also wird jede Folge $f \to B_1 B_2 \ldots B_m$ durch die Regel $A_1 \to B_1 B_2 \ldots B_m$ und jede Folge $f \to a$ durch die Regel $A_1 \to a$ ersetzt.

4. Jede Regel der Form $A \to B_1 B_2 \ldots B_m$ mit $A, B_1, B_2, \ldots, B_m \in N'$, $m \geq 2$ wird durch folgende Regeln ersetzt:

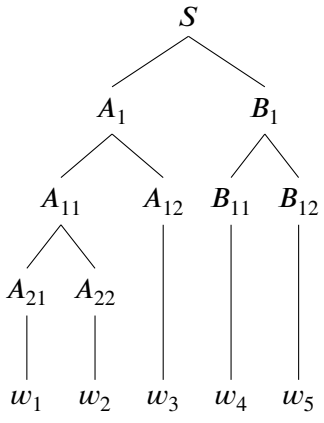
$$\begin{aligned} A &\to B_1 C_1 \\ C_i &\to B_{i+1} C_{i+1} \quad \text{(wobei } i \in \{1, \ldots, m-3\}\text{)} \\ C_{m-2} &\to B_{m-1} B_m \end{aligned}$$

   Dabei sind $C_1, \ldots, C_{m-2} \in N'$ neue bisher nicht vorhandene Nichtterminale.

Somit entsprechen alle Produktionen der konstruierten Grammatik $G' = (V'_N, V_T, P', S)$ den Vorgaben der Chomsky-Normalform. □

Beispiel 4.3.11 Sei $G = (V_N, V_T, P, S)$ eine kontextfreie Grammatik in Chomsky-Normalform mit $V_N = \{S, A_1, A_{11}, A_{12}, A_{21}, A_{22}, B_1, B_{11}, B_{12}\}$ und $V_T = \{w_1, w_2, w_3, w_4, w_5\}$. Die Produktionsregeln P und ein möglicher Syntaxbaum sind nachfolgend angegeben.

$$\begin{aligned} S &\to A_1\ B_1 \\ A_1 &\to A_{11}\ A_{12} \\ B_1 &\to B_{11}\ B_{12} \\ A_{11} &\to A_{21}\ A_{22} \\ A_{21} &\to w_1 \\ A_{22} &\to w_2 \\ A_{12} &\to w_3 \\ B_{11} &\to w_4 \\ B_{12} &\to w_5 \end{aligned}$$



Bemerkung 4.3.12 Syntaxbäume werden manchmal auch *Parsebäume* genannt, wir werden hier nicht näher darauf eingehen. Im Beweis des nachfolgenden Satzes werden wir aber auf dieses Beispiel zurückkommen, um gewisse Eigenschaften nachvollziehen zu können. Für weitere Erläuterungen zu Syntaxbäumen siehe [8, 193-200].

Der Induktionsbeweis zum nächsten Satz findet sich in [8, 284-285].

Satz 4.3.13 *Angenommen, es liegt ein Syntaxbaum entsprechend einer Grammatik in Chomsky-Normalform $G = (V_N, V_T, P, S)$ vor, und angenommen dieser Syntaxbaum ergibt die terminale Zeichenreihe p. Wenn der längste Pfad die Länge n hat, dann gilt $|p| \leq 2^{n-1}$.*

Wie für reguläre Sprachen gibt es auch für kontextfreie Sprachen ein Pumping-Lemma. Der nachfolgende Beweis stammt aus [8, 285-288] bzw. [13, 125-126].

Satz 4.3.14 (Pumping-Lemma für kontextfreie Sprachen) *Es sei L eine kontextfreie Sprache. Dann gibt es eine Konstante $n \in \mathbb{N}_0$, die nur von der Sprache L abhängt, sodass sich jedes Wort $p \in L$ mit $|p| \geq n$ in $p = uvwxy$ zerlegen lässt, sodass gilt:*

(i) $|vx| \geq 1$,

(ii) $|vwx| \leq n$,

(iii) $uv^i wx^i y \in L$ *für alle* $i \geq 0$.

Beweis. Angenommen L ist eine kontextfreie Sprache. Wegen Satz 4.3.10 gibt es dann eine kontextfreie Grammatik $G = (V_N, V_T, P, S)$ in Chomsky-Normalform, die L erzeugt.

Die Erzeugung jedes Wortes $p \in L$ kann mithilfe eines Syntaxbaumes dargestellt werden. Bei einer Grammatik in Chomsky-Normalform gibt es nur Regeln der Form $A \to BC$ oder $A \to a$, wobei $A, B, C \in V_N$ und $a \in V_T$.

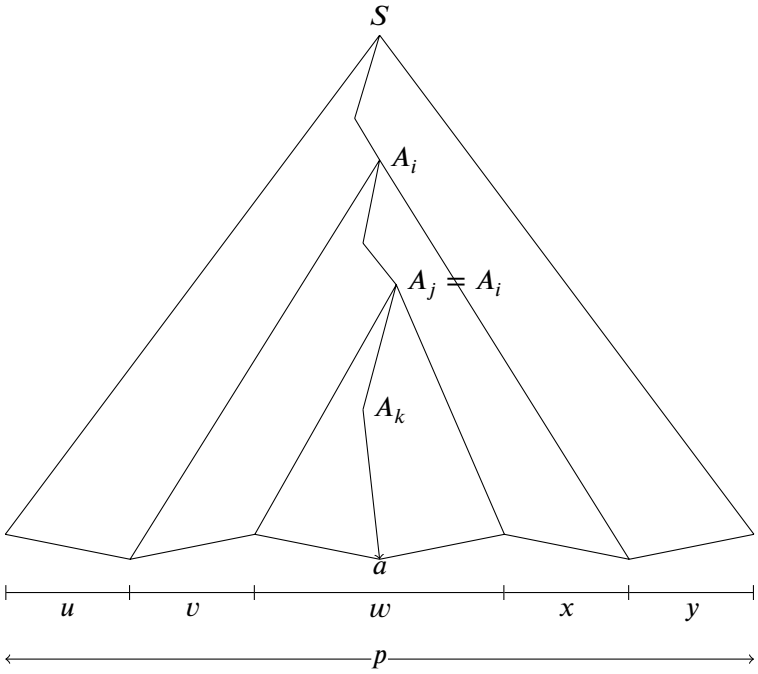
Man beginnt oben mit dem Startsymbol S. Bei jeder Anwendung einer Regel der From $A \to BC$ verzweigt sich der Syntaxbaum in zwei Kanten. Im Beispiel 4.3.11 wird demnach die Produktion $S \to A_1\ B_1$ so dargestellt, dass vom Startsymbol S zwei Wege wegführen, einer zu A_1 und einer zu B_1. Mit einer Regel der Form $A \to a$ endet ein Pfad. Gleichermaßen wird im Syntaxbaum des Beispiels 4.3.11 die Regel $A_{21} \to w_1$ veranschaulicht, sodass nämlich genau ein Pfad von A_{21} wegführt, nämlich zum Terminal w_1.

Wir setzen die Konstante $n = 2^{|V_N|}$, wobei $|V_N|$ die Anzahl der Variablen ist. Sei nun $p \in L$ mit $|p| \geq n$. Nach Satz 4.3.13 muss jeder Syntaxbaum, dessen längster Pfad höchstens die Länge $|V_N|$ hat, ein „Ergebnis“ der Länge $2^{|V_N|-1} = n/2$ oder eine geringere Länge haben. Ein solcher Syntaxbaum kann also nicht p erzeugen, weil $|p| = n$ zu lang ist. Also muss jeder Syntaxbaum, der p erzeugt, mindestens die Länge $|V_N| + 1$ haben.

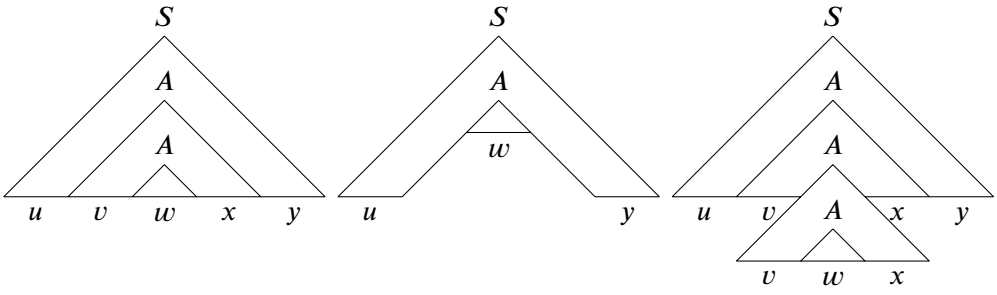
Der längste Pfad im Syntaxbaum für p sei $S = A_0 \to A_1 \to A_2 \to \cdots \to A_k \to a$ für gewisse $A_0, A_1 \ldots, A_k \in V_N$, $a \in V_T$. Nachdem der längste Pfad mindestens die Länge $|V_N| + 1$ hat und es nur $|V_N|$ viele nichtterminale Symbole gibt, existieren i, j mit $0 \leq i < j \leq k$ und $A_i = A_j$. Es ergibt sich der nachfolgende schematische Syntaxbaum.

Es ist möglich, den Syntaxbaum wie unten dargestellt aufzuteilen. Die Zeichenreihe w ist das Ergebnis des Teilbaums, der von A_j als Wurzel ausgeht. Bei v und x handelt es sich um die Zeichenreihen, die sich im Ergebnis des Baumes mit der Wurzel A_i links bzw. rechts von w befinden. Schließlich sind u und y diejenigen Teile von p, die sich im Ergebnis des Baumes mit der Wurzel S links von v bzw. rechts von x befinden.

Wegen $i \neq j$ folgt $|vx| \geq 1$, außerdem sieht man, dass $|vwx| \leq n$ gilt, da $S \neq A_i$.



Nun können aus diesem ursprünglichen Syntaxbaum mit $A_i = A_j = A$ neue Bäume konstruiert werden, das funktioniert aufgrund des Doppelvorkommens von $A_i = A_j$. Fügen wir unmittelbar an A_i den Ableitungsbaum von A_j an, so wird $uwy = uv^0wx^0y$ abgeleitet. Fügen wir an A_j den Teilbaum von A_i an, so wird uv^2wx^2y abgeleitet. Die nächste Abbildung zeigt diese Art des Konstruierens.



Genauso könnte man den Baum von uv^3wx^3y oder allgemein uv^iwx^iy konstruieren. Folglich sind alle $uv^iwx^iy \in L$ und das Pumping-Lemma wurde bewiesen. □

Eine Verallgemeinerung zum Pumping-Lemma, das Ogden-Lemma, wird ohne Beweis angegeben, siehe dafür auch [13, 126].

Satz 4.3.15 (Ogdens-Lemma) *Es sei L eine kontextfreie Sprache. Dann gibt es eine Konstante $n \in \mathbb{N}_0$, sodass für jedes Wort $p \in L$ mit $|p| \geq n$ gilt: Werden in p mindestens n Positionen markiert, dann lässt es sich in $p = uvwxy$ zerlegen, sodass gilt:*

(i) in vx ist mindestens eine Position markiert,

(ii) in vwx sind höchstens n Positionen markiert,

(iii) $uv^iwx^iy \in L$ für alle $i \geq 0$.

Beispiel 4.3.16 Die Sprache $L = \{a^ib^jc^kd^l | i \neq 0 \Rightarrow j = k = l\}$ ist nicht kontextfrei. Der Nachweis, dass diese Sprache nicht kontextfrei ist, kann nicht mit dem Pumping-Lemma für kontextfreie Sprachen geführt werden, aber mit dem Ogden-Lemma.
Wir nehmen indirekt an, L sei kontextfrei. Dann existiert eine Konstante n, sodass es für jedes Wort $|p| \geq n$ mit mindestens n markierten Positionen eine Zerlegung gibt, die alle Eigenschaften des Ogden-Lemmas erfüllt.
Insbesondere für das Wort $z = ab^nc^nd^n \in L$ mit Markierung auf dem Teil b^n muss es eine Zerlegung $z = uvwxy$ geben, die (i), (ii) und (iii) des Ogden-Lemmas erfüllt. Eigenschaft (ii) ist stets erfüllt, da es nur n markierte Positionen in z gibt. Aus (i) folgt, dass vx mindestens ein b enthält. Wir betrachten alle möglichen Fälle der Zerlegung mit mindestens einem b in vx:

1. Aus $vx \in \{a, b, c\}^*$ folgt, dass uv^2wx^2y mehr bs als ds hat. Denn in v^2 und x^2 kommen keine ds vor, aber v oder x enthalten mindestens ein b und diese Teile werden aufgepumpt.

2. Aus $vx \in \{a, b, d\}^*$ folgt, dass uv^2wx^2y mehr bs als cs enthält. Denn in v^2 und x^2 kommen keine cs vor, aber v oder x enthalten mindestens ein b und diese Teile werden aufgepumpt.

3. Aus $vx \in \{a, b, c, d\}^*$ und $vx \notin \{a, b, c\}^*$ bzw. $vx \notin \{a, b, d\}^*$ folgt, dass in v oder in x zwei verschiedene Buchstaben vorkommen müssen. Beim Aufpumpen entspricht dann uv^2wx^2y nicht mehr der Form $a^*b^*c^*d^*$.

Nachdem das Wort uv^2wx^2y in keinem der Fälle in L liegt, folgt ein Widerspruch zur Eigenschaft (iii). Daher ist L nicht kontextfrei.

Abschließend sollen noch Eigenschaften kontextfreier Sprachen behandelt werden, die für das nächste Kapitel gebraucht werden. Proposition 4.3.17 und Satz 4.3.18 gehen auf [8, 296-297] zurück.

Proposition 4.3.17 *Seien L_1 und L_2 kontextfreie Sprachen. Der Durchschnitt $L_1 \cap L_2$ ist im Allgemeinen nicht kontextfrei. Das heißt, kontextfreie Sprachen sind bezüglich der Durchschnittsbildung nicht abgeschlossen.*

Beweis. Wir geben ein Gegenbeispiel. Sei $L_1 = \{0^n1^n2^i | n \geq 1, i \geq 1\}$ und G_1 eine erzeugende kontextfreie Grammatik, $G_1 = (\{S, A, B\}, \{0, 1, 2\}, P, S)$. Die Produktionsregeln sind:

$$S \to AB$$
$$A \to 2A1 \mid 01$$
$$B \to 2B \mid 2$$

Sei $L_2 = \{0^i 1^n 2^n | n \geq 1, i \geq 1\}$ und $G_2 = (\{S, A, B\}, \{0, 1, 2\}, P, S)$ eine kontextfreie Grammatik, die L_2 erzeugt. Die Produktionsregeln sind:

$$\begin{aligned} S &\to AB \\ A &\to 0A \mid \\ B &\to 1B2 \mid 12 \end{aligned}$$

Der Durchschnitt der beiden Sprachen ist $L = L_1 \cap L_2 = \{0^n 1^n 2^n | n \geq 1\}$. Mithilfe des Pumping-Lemmas für kontextfreie Sprachen lässt sich zeigen, dass L nicht kontextfrei ist.

Indirekt angenommen, L sei kontextfrei. Dann gibt es nach dem Pumping-Lemma für kontextfreie Sprachen eine Konstante n, sodass die Eigenschaften des Pumping-Lemmas gelten. Sei nun $p = 0^n 1^n 2^n \in L$. Nachdem $|p| \geq n$, lässt sich p in $uvwxy$ zerlegen, wobei $|vwx| \leq n$ und $|vx| \geq 1$. Der Teil vwx kann nicht sowohl Nuller als auch Zweier enthalten, da der letzte Nuller und der erste Zweier $n + 1$ Positionen von einander entfernt sind. Es werden nun zwei Fälle unterschieden:

1. vwx enthält keine Zweier, also sind alle Zweier in y enthalten. Dann besteht vx nur aus Nullern und Einsern und verfügt über mindestens eines dieser Symbole, denn $|vx| \geq 1$. Somit enthält $uwy = uv^0wx^0y$ genau n Zweier und weniger als n Nuller oder weniger als n Einser. Daher ist uv^0wx^0y nicht in L enthalten.

2. vwx enthält keine Nuller, also sind alle Nuller in u enthalten. Dann besteht vx nur aus Einsern und Zweiern und verfügt über mindestens eines dieser Symbole, denn $|vx| \geq 1$. Somit enthält $uwy = uv^0wx^0y$ genau n Nuller und weniger als n Einser oder weniger als n Zweier. Daher ist uv^0wx^0y nicht in L enthalten.

Da in beiden Fällen $uv^0wx^0y \notin L$, kommt man zu einem Widerspruch zur Eigenschaft (iii) des Pumping-Lemmas für reguläre Sprachen. Das heißt, die ursprüngliche Annahme war falsch. Es gilt also, dass $L = L_1 \cap L_2 = \{0^n 1^n 2^n | n \geq 1\}$ nicht kontextfrei ist. □

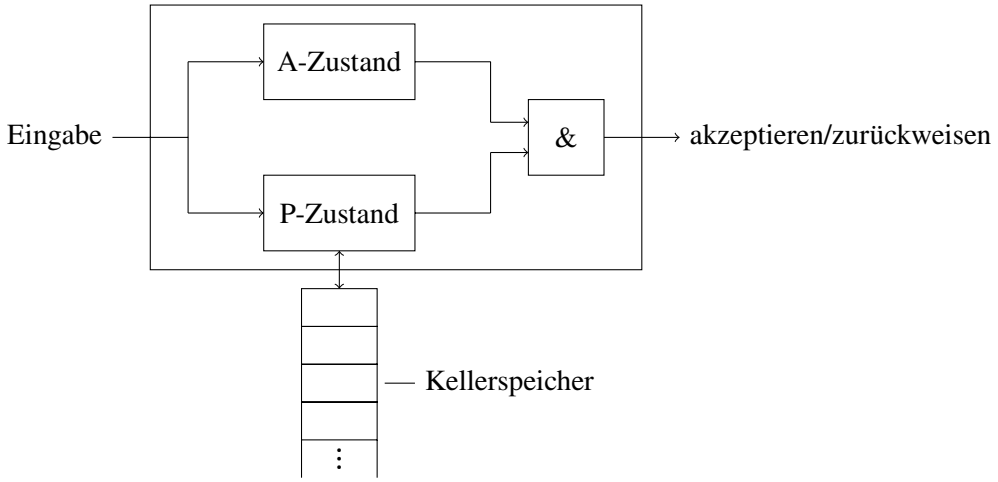
Satz 4.3.18 *Wenn L eine kontextfreie Sprache und R eine reguläre Sprache ist, dann ist $L \cap R$ eine kontextfreie Sprache.*

Beweis. Zunächst wird die kontextfreie Sprache L als Kellerautomat und die reguläre Sprache R als endlicher Akzeptor dargestellt. Um den Durchschnitt dieser beiden Sprachen zu erhalten, werden die beiden Automaten *parallel* ausgeführt, dann erhält man einen Kellerautomaten.

Sei $P = (Z_P, V, U, \delta_P, z_{0_P}, k_0, F_P)$ ein Kellerautomat, der L akzeptiert, und der endliche Akzeptor $A = (Z_A, V, \delta_A, z_{0_A}, F_A)$ akzeptiere die Sprache R. Nun wird folgender Kellerautomat konstruiert:

$$P' = (Z_P \times Z_A, V, U, \delta, (z_{0_P}, z_{0_A}), k_0, F_P \times F_A).$$

Dabei ist $\delta : (Z_P \times Z_A) \times (V \cup \{\varepsilon\}) \times U \to (Z_P \times Z_A) \times U^*$ mit $\delta((z_P, z_A), x, k) \mapsto ((q_P, q_A), \gamma)$, sodass $q_A = \delta_A(z_A, x)$ und (z_P, γ) in $\delta_P(z_P, x, k)$ enthalten ist. Nachdem das Konzept deterministischer und nichtdeterministischer Akzeptoren gleichwertig ist (siehe Satz 4.2.7), kann A als deterministisch betrachtet werden.



Das heißt, für jeden Schritt des Kellerautomaten P wurde auch ein Schritt des Kellerautomaten P' festgelegt, zudem wird in einer zweiten Zustandskomponente von P' der Zustand des endlichen Akzeptors A mitgeführt.

Da (z_P, z_A) genau dann ein akzeptierender Zustand von P' ist, wenn z_P ein akzeptierender Zustand von P und z_A ein akzeptierender Zustand von A ist, kann man daraus schließen, dass P' ein Wort w genau dann akzeptiert, wenn sowohl P als auch A das Wort w akzeptieren. Das bedeutet, der Kellerautomat P' akzeptiert genau die Sprache $L \cap R$ und somit ist $L \cap R$ kontextfrei. □

5 Englisch ist keine kontextfreie Sprache

In seinem Artikel *English Is Not a Context-Free Language* [7] stellt James Higginbotham die Behauptung auf, dass Englisch keine kontextfreie Sprache ist und beweist dies mithilfe des bereits erwähnten Ogden-Lemmas (siehe Satz 4.3.15).

5.1 Beweisidee

Die Beweisidee ist, dass eine formale reguläre Sprache L konstruiert wird, deren Schnitt mit der englischen Sprache nicht kontextfrei ist. Aus Satz 4.3.18 wissen wir: wenn L eine kontextfreie und R eine reguläre Sprache ist, dann ist $L \cap R$ eine kontextfreie Sprache, also

$$\left.\begin{array}{r} L \text{ ist regulär} \\ (L \cap \text{Englisch}) \text{ ist nicht kontextfrei} \end{array}\right\} \Rightarrow \text{ Englisch ist nicht kontextfrei.}$$

Zunächst wird die Menge L folgendermaßen definiert:

$$\begin{aligned} L = \quad & \textit{the woman such that } (\textit{the man such that})^* \textit{ she } (\textit{gave } (\textit{this} \cup \textit{him}) \textit{ to} \\ & (\textit{this} \cup \textit{him}))^* \textit{ left is here} \end{aligned}$$

Bei den Operatoren * und $\cup$ handelt es sich um den Kleene-Star bzw. die Vereinigung von Sprachen (siehe Bemerkung 4.2.6). Der Einfachheit halber werden Abkürzungen eingeführt:

$$\begin{aligned} X &= \textit{the woman such that}, \\ Y &= \textit{the man such that}, \\ Z_1 &= \textit{gave him to him}, \\ Z_2 &= \textit{gave him to this}, \\ Z_3 &= \textit{gave this to him}, \\ Z_4 &= \textit{gave this to this}, \\ W &= \textit{left is here}, \\ Z &= \{Z_1\} \cup \{Z_2\} \cup \{Z_3\} \cup \{Z_4\}. \end{aligned}$$

Dabei steht Z für irgendein Z_i mit $i \in \{1, 2, 3, 4\}$. Wörter der Sprache L haben also die Form $X\, Y^i$ she $Z^j\, W$, wobei $i, j \in \mathbb{N}_0$.

Um sicherzugehen, dass L eine reguläre Sprache ist, wird eine reguläre Grammatik angegeben, die L erzeugt. Sei $G = (V_N, V_T, P, S)$ mit $V_N = \{S, \alpha, \beta, \gamma, \delta\}$, $V_T = \{X, Y, Z, W\}$ und fol-

genden Produktionsregeln:

$$
\begin{aligned}
S &\to X\alpha \mid X\beta \\
\alpha &\to Y\ \alpha \mid Y\ \beta \\
\beta &\to \textit{she}\ \gamma \mid \textit{she}\ \delta \\
\gamma &\to Z\ \gamma \mid Z\ \delta \\
\delta &\to W
\end{aligned}
$$

Man sieht, dass die von der regulären Grammtik G erzeugte Sprache L ist. Daher folgt, dass L eine reguläre Sprache ist.

Nun werden zwei Behauptungen aufgestellt:

$$
\begin{aligned}
L \cap \text{Englisch} = \{ X\ Y^n\ \textit{she}\ Z^n\ W \mid (n \geq 0) \ \wedge\ &(\text{von links nach rechts gelesen über-} \\
&\text{steigt die Anzahl der } \textit{this} \text{ die Anzahl der } \textit{him} \text{ nie um mehr als 1})\}. \qquad (5.1)
\end{aligned}
$$

Sei A nun die in Behauptung (5.1) beschriebene Menge, die sich auf der rechten Seite des Gleichheitszeichens befindet. Dann gilt:

$$A \text{ ist keine kontextfreie Sprache.} \qquad (5.2)$$

Aus den beiden Behauptungen (5.1) und (5.2) zusammen folgt schließlich, dass Englisch keine kontextfreie Sprache ist.

5.2 $L \cap$ Englisch $= A$

Dieser Abschnitt beschäftigt sich damit, wie der Durchschnitt der vorhin konstruierten Menge L mit der englischen Sprache aussieht. Wir werden versuchen, die Behauptung (5.1) zu beweisen.

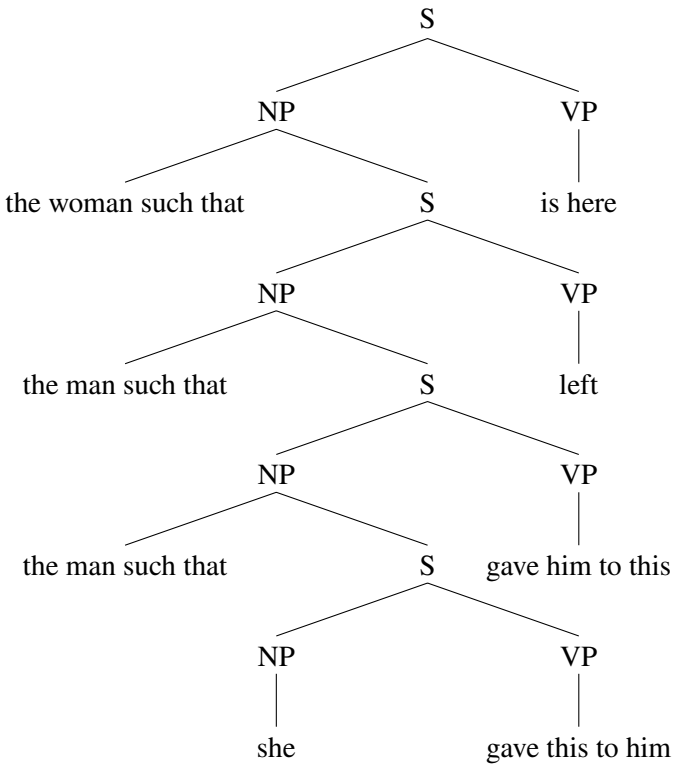
Zum besseren Verständnis betrachtet man zunächst zwei Beispiele.

$$\textit{the woman such that she left is here} = X\ Y^0\ \textit{she}\ Z^0\ W \qquad (5.3)$$

Beispiel (5.3) ist das kürzeste Wort der regulären Sprache L. Man sieht, dass es sich dabei um einen englischen Satz handelt, der aus dem Subjekt (bzw. Nominalphrase) *the woman such that she left* und dem Prädikat (bzw. Verbalphrase) *is here* besteht. Außerdem gehört (5.3) zu A, da man es als $X\ Y^0\ \textit{she}\ Z^0\ W$ ausdrücken kann, und nachdem weder *this* noch *him* vorkommen, wird die Bedingung, dass die Anzahl der *this* die Anzahl der *him*, von links nach rechts gelesen, nie um mehr als 1 übersteigt, trivialerweise erfüllt.

$$
\begin{aligned}
&\textit{the woman such that the man such that the man such that she gave this to him} \\
&\textit{gave him to this left is here} = X\ Y^2\ \textit{she}\ Z_3\ Z_2\ W \qquad (5.4)
\end{aligned}
$$

Auch Beispiel (5.4) gehört sowohl zu A als auch zu L. Dabei handelt es sich ebenfalls um einen englischen Satz, wie der nachfolgenden Abbildung entnommen werden kann. Dieser Baum zeigt die Phrasenstruktur des Beispiels (5.4).



Die Gleichheit ($L \cap$ Englisch) $= A$ zeigt man in zwei Schritten.

1. Alle Wörter aus ($L \cap$ Englisch) müssen in A enthalten sein, also ($L \cap$ Englisch) $\subset A$.
2. Alle Wörter aus A müssen in ($L \cap$ Englisch) enthalten sein, also $A \subset (L \cap$ Englisch).

Proposition 5.2.1 ($L \cap$ *Englisch*) $\subset A$.

Beweis. Zuerst überlegt man sich, was beim Weglassen oder Hinzufügen einzelner Y oder Z im Beispiel (5.4) passiert. Würden folgende Wörter auch noch zum Englischen gehören?

$$X\ Y\ \textit{she}\ Z_3\ Z_2\ W$$
$$X\ Y^3\ \textit{she}\ Z_3\ Z_2\ W$$
$$X\ Y^2\ \textit{she}\ Z_3\ W$$
$$X\ Y^2\ \textit{she}\ Z_3^2\ Z_2\ W$$

Nein, das Resultat wäre grammatikalisch falsch. Wörter aus L, die der englischen Sprache angehören, haben null oder mehr eingebettete Nebensätze, die mit *such that* eingeleitet werden. Das Weglassen oder Hinzufügen einzelner Y oder Z würde die Anzahl der Nominalphrasen und Verbalphrasen ins Ungleichgewicht bringen. Formaler ausgedrückt: $X =$ *the woman such that* leitet eine Nominalphrase ein, *she* ist eine Nominalphrase. $W =$ *left is here* beinhaltet zwei Verbalphrasen, nämlich *left* und *is here*. Jedes $Y =$ *the man such that* ist Einleitung einer neuen Nominalphrase. Jedes Z ist eine Verbalphrase, z. B. $Z_2 =$ *gave him to this*. Das heißt, das kürzeste

Wort (X *she* W) (siehe Beispiel 5.3) beinhaltet zwei Nominalphrasen und zwei Verbalphrasen. Allgemein gilt dann, dass im Wort ($X\ Y^i$ *she* $Z^j\ W$) die Anzahl der Nominalphrasen $i + 2$ und die Anzahl der Verbalphrasen $j + 2$ ist. Wenn dieses Wort zur englischen Sprache gehören soll, wenn also die Anzahl der Nominal- und Verbalphrasen gleich sein soll, muss gelten:

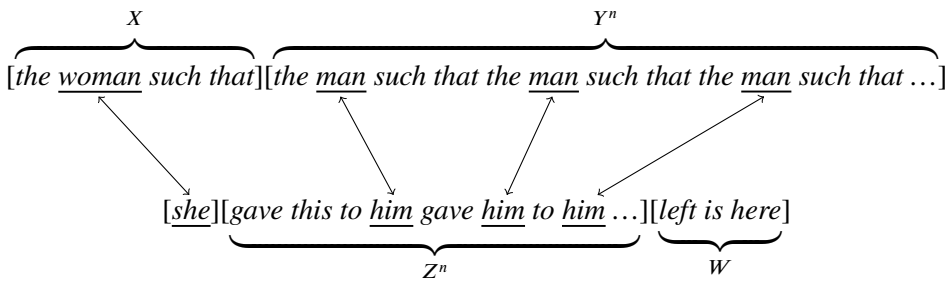
$$z = (X\ Y^i\ \textit{she}\ Z^j\ W) \in (L \cap \text{Englisch}) \quad \Rightarrow \quad i = j \tag{5.5}$$

Somit erfüllen alle $z \in (L \cap \text{Englisch})$ die erste Voraussetzung für eine Zugehörigkeit zur Sprache A, nämlich die gleiche Anzahl von Y und Z.

Daneben gibt es aber eine weitere Bedingung, die Wörter der Sprache A erfüllen müssen, dass, von links nach rechts gelesen, die Anzahl der *this* die Anzahl der *him* nie um mehr als 1 übersteigt. Dazu benutzt man die Tatsache, dass die englische Sprache keine sogenannte *leere Quantifizierung* (Quantoren, die keine Variablen binden) erlaubt. Auf die englische Sprache umgemünzt bedeutet das:

> Relativkonstruktionen, die mit *such that* eingeleitet werden,
> müssen ein passendes Bezugswort haben. (5.6)

In einem Wort ($X\ Y^n$ *she* $Z^n\ W$) aus ($L \cap$ Englisch) ist die Anzahl der Y genau n. Also muss sich aufgrund der Bedingung (5.6) jedes *the man* aus einer Nominalphrase Y = *the man such that* auf genau ein *him* aus den Verbalphrasen Z^n beziehen. Außerdem kann kein *him* zweimal zu diesem Zweck benutzt werden, das heißt, ein *him* darf nicht mit zwei verschiedenen Y verknüpft werden. Daher folgt, dass jedes $z \in (L \cap \text{Englisch})$ nicht nur gleich viele Z und Y hat, sondern auch mindestens so viele *him* wie Y.



Es gilt noch mehr: Die Abbildung der Phrasenstruktur des Beispiels (5.4) zeigt die einzige Möglichkeit, wie Wörter aus ($L \cap$ Englisch) konstruiert werden können. Wegen der Bedingung (5.6) ist es notwendig, dass in jedem Anfangssegment Z^k eines Abschnitts Z^n, die Anzahl der *him* mindestens k ist. Anders ausgedrückt bedeutet das, von links nach rechts gelesen, darf die Anzahl der *this* die Anzahl der *him* nie um mehr als 1 übersteigen. Daher gilt:

$$z \in (L \cap \text{Englisch}) \quad \Rightarrow \quad \begin{array}{l}\text{in } z \text{ übersteigt die Anzahl der } \textit{this} \text{ die Anzahl der } \textit{him},\\ \text{von links nach rechts gelesen, nie um mehr als 1}\end{array} \tag{5.7}$$

Aus (5.5) und (5.7) folgt schließlich, dass $\forall z \in (L \cap \text{Englisch})$ gilt, dass $z \in A$. Somit wurde die Behauptung ($L \cap$ Englisch) $\subset A$ bewiesen. □

Proposition 5.2.2 $A \subset (L \cap$ *Englisch*$)$.

Beweis. Es gilt, dass

$$A \subset (L \cap \text{Englisch}) \Leftrightarrow (A \subset L) \wedge (A \subset \text{Englisch}).$$

Zunächst kann man festhalten, dass Wörter aus A die Form $(X\ Y^n$ *she* $Z^n\ W)$ haben und somit jedenfalls auch in L enthalten sind, also $\forall z \in A$ gilt, dass $z \in L$.

$$z = (X\ Y^n \textit{ she } Z^n\ W) \in A \quad \Rightarrow \quad z \in L \tag{5.8}$$

Nun muss man sich noch überlegen, ob alle $z \in A$ auch Teil der englischen Sprache sind.

Im Englischen kann man folgende Regel beobachten: Es sei D ein gewöhnlicher englischer Deklarativsatz (z. B. *she gave this to him*), der ein Pronomen der 3. Person (z. B. *him*) enthält, dessen Bezugswort aber nicht unbedingt in D liegen muss. N sei irgendein Nomen, das in Zahl und Geschlecht mit dem Pronomen aus D übereinstimmt (z. B. *man*).

Dann ist (*the* N *such that* D) eine wohlgeformte englische Nominalphrase. (5.9)

Dabei wird aufgrund der Bedingung (5.6) implizit angenommen, dass das Bezugswort dieses Pronomens der 3. Person nicht im Deklarativsatz D liegt, also das Pronomen nicht in D gebunden ist. Nur dann kann sich das Pronomen auf N beziehen und (*the* N *such that* D) ist eine wohlgeformte englische Nominalphrase.

Man betrachtet nun Wörter der Sprache A. Diese haben laut Voraussetzung die Eigenschaft, dass, von links nach rechts gelesen, die Anzahl der *this* die Anzahl der *him* nie um mehr als 1 übersteigt. Also hat in einem Wort $z = (X\ Y^n$ *she* $Z^n\ W) \in A$ auch kein Anfangssegment Z^k von Z^n mehr *this* als *him*. Das Anfangssegment Z^k kann beispielsweise nicht mit $Z_4 =$ *gave this to this* beginnen (denn dann wäre die Bedingung sofort verletzt), das erste Z_4 kann erst dann auftreten, wenn davor mindestens ein $Z_1 =$ *gave him to him* vorgekommen ist.

Man kann nun jedem $Y =$ *the man such that* genau ein *him* aus den Z^n wie folgt zuordnen: das innerste Y (d. h. das am nächsten zu *she* gelegene Y) korrespondiert zum innersten *him* (d. h. das am nächsten zu *she* gelegene *him*), das zweitinnerste Y korrespondiert zum zweitinnersten *him*, usw. Beispiel (5.4) erfüllt diese Eigenschaften. Allgemein können diese Zuordnungen aber auf verschiedene Arten erfüllt sein, es gibt eine Vielfachheit an möglichen Zuordnungen, und zwar genau dann, wenn ein Anfangssegment Z^k von Z^n mehr *him* als *this* enthält. Ein Beispiel dafür ist

the woman such that the man such that the man such that she gave him to him
gave this to this left is here $= X\ Y^2$ *she* $Z_1\ Z_4\ W$. (5.10)

Dabei handelt es sich um ein gültiges Wort der Sprache A, das auch Teil des Englischen ist. Aber in Beispiel (5.10) lassen sich die $Y =$ *the man such that* nicht eindeutig zu einem *him* aus den Z zuordnen.

the woman such that the man such that the man such that she gave him to him gave this to this left is here = $X\ Y^2\ she\ Z_1\ Z_4\ W$. (5.11)

the woman such that the man such that the man such that she gave him to him gave this to this left is here = $X\ Y^2\ she\ Z_1\ Z_4\ W$. (5.12)

Sowohl Zuordnung (5.11) als auch Zuordnung (5.12) sind im Englischen gültig. Es herrscht also eine Vielfachheit an möglichen Zuordnungen.

Wir wollen nun beweisen, dass alle Wörter $z = (X\ Y^n\ she\ Z^n\ W) \in A$ auch Teil des Englischen sind. Es soll bewiesen werden, dass

$$\forall n \in \mathbb{N}_0 : \quad (X\ Y^n\ she\ Z^n\ W) \in \mathrm{A} \quad \Rightarrow \quad (X\ Y^n\ she\ Z^n\ W) \in (\text{Englisch}).$$

Das beweist man mittels vollständiger Induktion.

Sei $n = 0$, dann ist $(X\ Y^0\ she\ Z^0\ W) =$ (*the woman such that she left is here*) ein englischer Satz und es gilt $(X\ Y^0\ she\ Z^0\ W) \in$ (Englisch), also gilt der Induktionsanfang für $n = 0$.

Im Induktionsschritt wollen wir nun zeigen, dass aus der Induktionsvoraussetzung

$$(X\ Y^{n-1}\ she\ Z^{n-1}\ W) \in \mathrm{A} \quad \Rightarrow \quad (X\ Y^{n-1}\ she\ Z^{n-1}\ W) \in (\text{Englisch})$$

die Induktionsbehauptung

$$(X\ Y^n\ she\ Z^n\ W) \in \mathrm{A} \quad \Rightarrow \quad (X\ Y^n\ she\ Z^n\ W) \in (\text{Englisch})$$

folgt.

Sei $(X\ Y^n\ she\ Z^n\ W) = (X\ Y^n\ she\ Z^{n-1}\ Z_q\ W) \in A$, wobei $n \neq 0$, $q \in \{1, 2, 3, 4\}$. Dann folgt daraus, dass $(X\ Y^{n-1}\ she\ Z^{n-1}\ W) \in A$. Das funktioniert, weil lediglich ein Y und das letzte Z_q gestrichen wurden. Wenn $(X\ Y^{n-1}\ she\ Z^{n-1}\ W)$ die Bedingung an die Anzahl der *this* und *him* verletzen würde, dann müsste auch $(X\ Y^n\ she\ Z^{n-1}\ Z_q\ W)$ diese verletzen.

Nach Induktionsvoraussetzung folgt $(X\ Y^{n-1}\ she\ Z^{n-1}\ W) \in$ (Englisch). Das heißt, $(X\ Y^{n-1}$ *she* $Z^{n-1}\ W)$ ist ein englischer Satz und $(X\ Y^{n-1}\ she\ Z^{n-1}$ *left*) ist eine englische Nominalphrase, indem man einfach die Verbalphrase *is here* weglässt.

Somit ist $(Y^{n-1}$ *she* Z^{n-1} *left*) wiederum ein englischer Satz (durch Entfernen des $X =$ *the woman such that*) und $(Y^{n-1}\ she\ Z^{n-1})$ eine englische Nominalphrase (durch Entfernen des *left*).

Bei jedem Z_q handelt es sich um eine Verbalphrase. Also muss es sich bei der Zusammensetzung (Y^{n-1} *she* Z^{n-1} Z_q) um einen englischen Satz handeln. In diesem Ausdruck (Y^{n-1} *she* Z^{n-1} Z_q) gibt es mindestens ein *him* mehr als es Y gibt, das gilt aufgrund der Eigenschaft der Wörter aus A. Nachdem (Y^{n-1} *she* Z^{n-1} Z_q) ein englischer Satz ist, gibt es darin mindestens ein *him*, das keinem Y zugeordnet ist.

Nun folgt aus den Bedingungen (5.6) und (5.9), dass

$$\textit{the} \underbrace{\textit{man}}_{N} \textit{ such that } \underbrace{(Y^{n-1} \textit{ she } Z^{n-1}\ Z_q)}_{D} \;=\; (Y^n \textit{ she } Z^{n-1}\ Z_q)$$

eine wohlgeformte englische Nominalphrase ist. Somit ist (Y^n *she* Z^{n-1} Z_q *left*) ein englischer Satz (durch Hinzufügen der Verbalphrase *left*). Aus (5.6) und (5.9) folgt wiederum, dass

$$\textit{the} \underbrace{\textit{woman}}_{N} \textit{ such that } \underbrace{(Y^n \textit{ she } Z^{n-1}\ Z_q \textit{ left})}_{D} \;=\; (X\ Y^n \textit{ she } Z^{n-1}\ Z_q \textit{ left})$$

eine wohlgeformte englische Nominalphrase ist. Somit ist (X Y^n *she* Z^{n-1} Z_q *left is here*) ein englischer Satz. Dabei handelt es sich genau um den Ausdruck (X Y^n *she* Z^{n-1} Z_q W) = (X Y^n *she* Z^n W). Also wurde der Induktionsschritt $(n-1) \to n$ gezeigt und mittels vollständiger Induktion bewiesen, dass (X Y^n *she* Z^n W) $\in$ (Englisch).

Jedes $z \in A$ hat für ein $n \in \mathbb{N}_0$ die Form (X Y^n *she* Z^n W). Also gilt, dass

$$z = (X\ Y^n \textit{ she } Z^n\ W) \in A \quad \Rightarrow \quad z \in (\text{Englisch}) \tag{5.13}$$

Aus (5.8) folgt, dass $A \subset L$ und aus (5.13) folgt, dass $A \subset$ (Englisch). Insgesamt folgt schließlich, dass $A \subset (L \cap \text{Englisch})$ und die Proposition wurde bewiesen. □

Proposition 5.2.3 ($L \cap$ *Englisch*) = A.

Beweis. Die Behauptungen (5.2.1) und (5.2.2) wurden beide bewiesen, somit

$$\left.\begin{array}{l}(L \cap \text{Englisch}) \subset A \\ A \subset (L \cap \text{Englisch})\end{array}\right\} \Rightarrow (L \cap \text{Englisch}) = A.$$

Es gilt also tatsächlich die Gleichheit ($L \cap$ Englisch) = A. □

5.3 A ist keine kontextfreie Sprache

In diesem Abschnitt wird mithilfe des Ogden-Lemmas (siehe Satz 4.3.15) indirekt bewiesen, dass A keine kontextfreie Sprache ist. Wir werden versuchen, die Behauptung (5.2) zu beweisen.

Nochmals zur Erinnerung:

$$A := \{X\ Y^n\ \textit{she}\ Z^n\ W \mid (n \geq 0)\ \wedge\ (\text{von links nach rechts gelesen übersteigt die Anzahl der } \textit{this} \text{ die Anzahl der } \textit{him} \text{ nie um mehr als 1})\}.$$

Die Leerzeichen werden nunmehr mit dem terminalen Symbol # dargestellt, indem man # rechts neben jedes Wort setzt. Demnach hat man beispielsweise für $Z_4 = \textit{gave\#this\#to\#this\#}$.

Ogdens-Lemma spricht von sogenannten *markierten Buchstaben* oder *markierten Positionen*. Damit es nicht zu Verwirrungen kommt, wird nachfolgend immer von Positionen die Rede sein. Als Beispiel dient dieses Wort der Sprache A:

$$\underline{\textit{th}}\textit{e\#}\underline{\textit{wo}}\textit{man\#}\underline{\textit{s}}\textit{such\#that\#she\#}\underline{\textit{left}}\textit{\#is\#here\#}.$$

Hier wurden die 2., 5., 6. und 10. Position markiert, auch alle Positionen in W wurden markiert.

Higginbotham verwendet in seinem Beweis eine leicht abgewandelte Variante des Ogden-Lemmas mit vier charakteristischen Eigenschaften. Diesen Satz geben wir ohne Beweis an, siehe dafür auch [5, 186-187].

Satz 5.3.1 (Ogdens-Lemma) *Sei L eine kontextfreie Sprache. Dann gibt es eine Zahl $n \in \mathbb{N}$, sodass für alle Wörter $z \in L$ gilt: wenn in z mindestens n Positionen markiert werden, dann gibt es eine Zerlegung $z = uvwxy$, sodass gilt:*

(i) *w enthält mindestens eine markierte Position,*

(ii) *entweder u und v enthalten beide markierte Positionen oder x und y enthalten beide markierte Positionen,*

(iii) *vwx enthält höchstens n markierte Positionen,*

(iv) *$uv^iwx^iy \in L$ für alle $i \geq 0$.*

Proposition 5.3.2 *A ist keine kontextfreie Sprache.*

Beweis. Angenommen, A ist eine kontextfreie Sprache. Dann gibt es nach Ogdens-Lemma ein $n \in \mathbb{N}$, sodass für jedes $z \in A$ und jede Markierung von mindestens n Positionen eine Zerlegung $z = uvwxy$ existiert, die alle Eigenschaften des Ogden-Lemmas erfüllt. Sei nun n diese Konstante und sei z das folgende Wort:

$$z = X\ Y^{2k}\ \textit{she\#}\ Z_1^k\ Z_4^k\ W \quad (\text{wobei } k > n).$$

Nachdem z gleich viele Y und Z hat, und da die Anzahl der *this*#, von links nach rechts gelesen, die Anzahl der *him*# niemals übersteigt, gehört z zur Sprache A (also $\Rightarrow z \in A$).

Sei nun die geforderte Markierung auf dem Teil Z_4^k. Es wurden somit mindestens n Positionen markiert, da nach Voraussetzung $k > n$ gilt. Daher kann man $z = uvwxy$ so faktorisieren, dass

alle vier Eigenschaften des Ogden-Lemmas erfüllt sind.

In weiterer Folge wird man beim Aufpumpen des Ausdrucks z aber sehen, dass $uv^i wx^i y$ für ein $i \geq 0$ nicht mehr zu A gehört, also die Eigenschaft (iv) des Ogden-Lemmas verletzt wird.

Sei nun die Zerlegung $z = uvwxy$, die alle Eigenschaften des Ogden-Lemmas erfüllt, nochmals in einer anderen Notation festgehalten, nämlich $\varphi = (u, v, w, x, y)$. Eine Zerlegung besteht dann aus sogenannten *Faktoren*, beispielsweise ist $\varphi_4 = x$ der vierte Faktor der Zerlegung φ.

Man bestimmt nun eine weitere Zerlegung $\psi = (X, Y^{2k}, she\#, Z_1^k, Z_4^k, W)$ des Wortes z. Im Gegensatz zur unbestimmten Zerlegung φ ist die Zerlegung ψ explizit gegeben und man sieht auf Anhieb, ψ ist keine Zerlegung wie in Ogdens-Lemma.

Nachfolgend werden zwei Beziehungen dieser Zerlegungen definiert.

1. Man sagt, ein Faktor von ψ überlappt einen Faktor von φ, wenn sie mindestens eine Position gemeinsam haben.
2. Man sagt ein Faktor von ψ ist in einem Faktor von φ *enthalten*, wenn alle Positionen des Faktors von ψ auch Positionen des Faktors von φ sind.

Zum besseren Verständnis wird hier ein Beispiel dargestellt.

$$\varphi = (\underbrace{u}_{\varphi_1}, \underbrace{v}_{\varphi_2}, \underbrace{w}_{\varphi_3}, \underbrace{x}_{\varphi_4}, \underbrace{y}_{\varphi_5})$$
$$\psi = (\underbrace{X}_{\psi_1}, \underbrace{Y^{2k}}_{\psi_2}, \underbrace{she\#}_{\psi_3}, \underbrace{Z_1^k}_{\psi_4}, \underbrace{Z_4^k}_{\psi_5}, \underbrace{W}_{\psi_6})$$

In diesem Fall gilt: ψ_2 überlappt φ_1, φ_4 überlappt ψ_6, ψ_4 ist enthalten in φ_3, φ_5 ist enthalten in ψ_6.

Im nächsten Teil geht es darum, aus den Eigenschaften des Ogden-Lemmas verschiedene Schlüsse für die Zerlegung von $z = uvwxy$ zu ziehen.

Aus der Eigenschaft (i) des Ogden-Lemmas folgt, dass $\varphi_3 = w$ mindestens eine markierte Position enthalten muss. Nachdem nur in $\psi_5 = Z_4^k$ Positionen markiert wurden, folgt

$$\Rightarrow \quad \psi_5 = Z_4^k \quad \text{überlappt} \quad \varphi_3 = w. \tag{5.14}$$

Aus der Eigenschaft (ii) des Ogden-Lemmas folgt, dass entweder $\varphi_1 = u$ und $\varphi_2 = v$ beide markierte Positionen enthalten oder $\varphi_4 = x$ und $\varphi_5 = y$ beide markierte Positionen enthalten.
Angenommen ersteres gilt, also $\varphi_1 = u$ und $\varphi_2 = v$ enthalten beide markierte Positionen. Dann muss $\psi_5 = Z_4^k$ sowohl $\varphi_1 = u$ als auch $\varphi_2 = v$ überlappen und die ersten vier Faktoren von ψ (also $\psi_1, \psi_2, \psi_3, \psi_4$) sind in $\varphi_1 = u$ enthalten.

$$\varphi = (\underbrace{u}_{\varphi_1}, \underbrace{v}_{\varphi_2}, \underbrace{w}_{\varphi_3}, \underbrace{x}_{\varphi_4}, \underbrace{y}_{\varphi_5})$$
$$\psi = (\underbrace{X}_{\psi_1}, \underbrace{Y^{2k}}_{\psi_2}, \underbrace{she\#}_{\psi_3}, \underbrace{Z_1^k}_{\psi_4}, \underbrace{Z_4^k}_{\psi_5}, \underbrace{W}_{\psi_6})$$

$\varphi_2 = v$ hat dann die Form

$$\varphi_2 = v = \tilde{x}\ Z_4^j\ \tilde{y} \quad \text{(für ein } j \geq 0\text{)}.$$

Nachdem $\psi_5 = Z_4^k$ sowohl $\varphi_1 = u$, $\varphi_2 = v$ und $\varphi_3 = w$ überlappt, sind $\tilde{x}$ und $\tilde{y}$ beide in $\psi_5 = Z_4^k$ enthalten, sie sind also Teilzeichenfolgen von Z_4 (z. B. *gave#thi* oder *is#to#this#*).

Angenommen, dass in $\varphi_2 = v$ mindestens ein *this#* vorkommt. Dann betrachtet man den aufgepumpten Ausdruck uv^iwx^iy für ein $i \geq 2$. Nachdem $\psi_4 = Z_1^k$ ganz in $\varphi_1 = u$ enthalten ist, und dieser Teil nicht aufgepumpt wird, ist die Anzahl der *him#* in uv^iwx^iy genau $2k$. Aber die Anzahl der *this#* ist dann mindestens $2k+1$, weil angenommen wurde, dass $\varphi_2 = v$ mindestens ein *this#* enthält. Somit gehören die aufgepumpten Ausdrücke uv^iwx^iy nicht mehr zu A und man kommt zu einem Widerspruch. Daher muss gelten, dass $\varphi_2 = v$ kein *this#* enthält.

Als nächstes wird argumentiert, dass $j = 0$ sein muss, und dass der Buchstabe g und der Buchstabe l in $\varphi_2 = v$ nicht vorkommen können. Nach Voraussetzung gilt, dass die Anzahl der Y und die Anzahl der Z in Wörtern der Sprache A gleich sein müssen. Diese ist auch gleich der Anzahl der g, da der Buchstabe g einmalig in Z vorkommt. Die Anzahl der Y im aufgepumpten Wort uv^iwx^iy ist $2k$, da $\psi_2 = Y^{2k}$ in $\varphi_1 = u$ enthalten ist, und dieser Teil nicht aufgepumpt wird. Somit darf beim Aufpumpen in $\varphi_2 = v = \tilde{x}\ Z_4^j\ \tilde{y}$ kein g aus einem Z_4 vorkommen, und es gilt $j = 0$. Außerdem gilt, dass Wörter der Sprache A genau ein l enthalten, daher kann der Buchstabe l nicht im aufgepumpten Teil $\varphi_2 = v$ auftreten.

Insgesamt folgt also, dass $\varphi_2 = v = \tilde{x}\tilde{y}$ eine nichtleere Teilzeichenfolge von

$$ave\#this\#to\#this\#$$

ist, die nicht so weit reicht, dass sie ein ganzes *this#* enthält. Aber egal, wie man diese Teilzeichenfolge nun wählt, ist sofort ersichtlich, dass beim Aufpumpen von v^i der Ausdruck uv^iwx^iy nicht mehr zur Sprache A gehören kann. Wählt man beispielsweise für $v = ave\#th$, dann wäre $v^3 = ave\#thave\#thave\#th$ und das kann kein Teil eines Wortes aus A sein.

Daher folgt ein Widerspruch und unsere Annahme ist falsch. Es gilt daher, dass $\varphi_1 = u$ und $\varphi_2 = v$ keine markierten Positionen enthalten, also $\psi_5 = Z_4^k$ überlappt nicht $\varphi_1 = u$ und $\varphi_2 = v$. Aufgrund der Eigenschaft (ii) des Ogden-Lemmas muss also folgen, dass

$$\Rightarrow \quad \psi_5 = Z_4^k \quad \text{überlappt} \quad \varphi_4 = x \ \text{ und } \ \varphi_5 = y. \tag{5.15}$$

$$\varphi = (\underbrace{u}_{\varphi_1}, \underbrace{v}_{\varphi_2}, \underbrace{w}_{\varphi_3}, \underbrace{x}_{\varphi_4}, \underbrace{y}_{\varphi_5})$$

$$\psi = (\underbrace{X}_{\psi_1}, \underbrace{Y^{2k}}_{\psi_2}, \underbrace{she\#}_{\psi_3}, \underbrace{Z_1^k}_{\psi_4}, \underbrace{Z_4^k}_{\psi_5}, \underbrace{W}_{\psi_6})$$

Als nächstes kann man festhalten, dass Wörter der Sprache A genau ein *she#* enthalten. Aus (5.14) und (5.15) folgt, dass $\psi_3 = she\#$ überlappt $\varphi_1 = u$ oder $\varphi_2 = v$ oder $\varphi_3 = w$.

Angenommen, $\psi_3 = she\#$ überlappt $\varphi_1 = u$. Dann ist $\psi_2 = Y^{2k}$ enthalten in $\varphi_1 = u$ und die Anzahl der Y in jedem aufgepumpten Ausdruck uv^iwx^iy ist genau $2k$. Somit kann $\varphi_4 = x$ weder den Buchstaben g noch den Buchstaben l enthalten, da ansonsten die Anzahl der g beim Aufpumpen des Ausdruck uv^iwx^iy mehr als $2k$ wird, und mehr als ein l vorkommen würde. Also muss $\varphi_4 = x$ eine nichtleere Teilzeichenfolge von ave#this#to#this# sein. Aber der aufgepumpte Ausdruck uv^iwx^iy liegt nicht mehr in der Sprache A. Daher folgt ein Widerspruch und es gilt, $\psi_3 = she\#$ überlappt nicht $\varphi_1 = u$.

Weiters kann $\psi_3 = she\#$ nicht in $\varphi_2 = v$ enthalten sein, da in diesem Fall beim Aufpumpen mehr als ein *she#* in uv^iwx^iy auftreten würde. Daher folgt, dass

$$\Rightarrow \quad \psi_3 = she\# \quad \text{überlappt} \quad \varphi_3 = w. \tag{5.16}$$

Nun folgt aus (5.14) und (5.16), dass

$$\Rightarrow \quad \psi_4 = Z_1^k \quad \text{ist enthalten in} \quad \varphi_3 = w. \tag{5.17}$$

$$\varphi = (\underbrace{u}_{\varphi_1}, \underbrace{v}_{\varphi_2}, \underbrace{w}_{\varphi_3}, \underbrace{x}_{\varphi_4}, \underbrace{y}_{\varphi_5})$$

$$\psi = (\underbrace{X}_{\psi_1}, \underbrace{Y^{2k}}_{\psi_2}, \underbrace{she\#}_{\psi_3}, \underbrace{Z_1^k}_{\psi_4}, \underbrace{Z_4^k}_{\psi_5}, \underbrace{W}_{\psi_6})$$

Aus (5.14) und (5.15) folgt, dass $\varphi_4 = x$ enthalten ist in $\psi_5 = Z_4^k$. Aber aus (5.17) folgt, dass alle *him#* im mittleren Faktor $\varphi_3 = w$ enthalten sind, und somit die Anzahl der *him#* in jedem aufgepumpten Ausdruck uv^iwx^iy genau $2k$ ist.

Wenn also die Ausdrücke uv^iwx^iy zur Sprache A gehören sollen, darf der aufgepumpte Faktor $\varphi_4 = x$ kein *this#* enthalten. Auch der Buchstabe l darf in $\varphi_4 = x$ nicht vorkommen. Somit muss $\varphi_4 = x$ eine nichtleere Teilzeichenfolge von

$$gave\#this\#to\#this\#gave\#this\#to\#this\#$$

sein, die kein ganzes *this#* enthält. Aber egal, wie man diese Teilzeichenfolge nun wählt, wird das aufgepumpte x^i zu einem Ausdruck uv^iwx^iy führen, der nicht mehr zur Sprache A gehört.

Somit führen alle Fälle zu einem Widerspruch und es gilt, dass A nicht kontextfrei ist. Das heißt die Behauptung (5.2) wurde bewiesen. □

5.4 Ergebnis

Satz 5.4.1 *Englisch ist keine kontextfreie Sprache.*

Beweis. Insgesamt wurden beide Behauptungen (5.1) und (5.2) bewiesen, also

$$\left.\begin{array}{r} L \cap \text{Englisch} = A \\ A \text{ ist nicht kontextfrei} \end{array}\right\} \Rightarrow \text{Englisch ist nicht kontextfrei.}$$

In diesem Kapitel wurde die natürliche Sprache Englisch untersucht, und man kommt zu dem Schluss, dass Englisch keine kontextfreie Sprache ist. □

5.5 Kommentare und Anmerkungen

In seinem Artikel [7] betrachtet Higginbotham die Behauptung, dass A nicht kontextfrei ist, zunächst von einem empirischen Standpunkt aus. Er argumentiert nicht formal mathematisch, sondern behauptet, dass in ($L \cap$ Englisch) mehrfache Abhängigkeiten auftreten, und dass kontextfreie Grammatiken mit solchen mehrfachen Abhängigkeiten nicht „mithalten“ können. Um die Behauptung (5.2) in dieser Arbeit zu zeigen, haben wir uns aber auf die mathematische Beweisführung mittels Ogdens-Lemma beschränkt, die im Appendix von [7] zu finden ist.

Higginbotham merkt in der Conclusio noch zwei technische Punkte an. Erstens scheint es nicht möglich zu sein, eine Sprache L zu entwerfen, die statt der Relativkonstruktion mit *such that*, echte Relativpronomen (z. B. *who*) verwendet. Der Grund dafür ist, dass die Bezugswörter der Relativpronomen nicht zu tief eingebettet sein dürfen. Bereits nachfolgendes Beispiel wäre grammatikalisch falsch, die möglichen Bezugswörter werden hier mit Lücken angegeben: *the woman who the man whom ___ gave this to ___ left is here*. Zweitens werden bei der entworfenen Sprache in der Satzmitte *such that* Konstruktionen eingeschoben. Solche Satzeinschübe können bekanntermaßen zu Schwierigkeit im Verständnis führen, doch eine Lösung mit einer einfachgerichteten Konstruktion scheint nicht offensichtlich zu sein, obwohl es seiner Meinung nach durchaus möglich sein könnte (siehe [7, 231]).

Abschließend merkt Higginbotham noch an, dass im Englischen nicht alle Sprecher*innen die Relativkonstruktion mit *such that* verwenden, obwohl diese doch weitverbreitet zu sein scheint. Streng genommen wurde demnach nur bewiesen, dass die Sprache des Autors (und wahrscheinlich vieler anderer englischsprachiger Personen) nicht kontextfrei ist. In einer derartigen Untersuchung seien solche Einschränkungen aber notwendig.

Die zentralen Annahmen über die englische Sprache in Higginbothams Beweis sind die Behauptungen (5.6) und (5.9). Dies sind tatsächlich auch die einzigen Annahmen über die englische Grammatik, die Higginbotham braucht, um die Gleichheit ($L \cap$ Englisch) $= A$ zu zeigen. Würde beispielsweise die Bedingung (5.6), also das Verbot der sogenannten *leeren Quantifizierung*, nicht gelten, dann wäre Higginbothams Beweis ungültig, aber die Gleichheit ($L \cap$ Englisch) $= A$ könnte trotzdem noch richtig sein. Um den Beweis dahingehend zu widerlegen, braucht man ein Gegenbeispiel zu (5.6), das in der Sprache L liegt.

Neben James Higginbotham beschäftigte sich in den 1980er Jahren auch Geoffrey K. Pullum, ein britisch-amerikanischer Linguist, mit dem Zusammenhang zwischen natürlichen und kontextfreien Sprachen. In *On Two Recent Attempts to Show that English Is Not a CFL* (1984) [10] widerspricht Pullum der Aussage von Higginbotham in [7] und behauptet, Higginbothams Beweis sei falsch. Seine Kritik basiert im Grunde auf dem oben dargebrachten Ansatz, Pullum weist also die Behauptung (5.6) über die englische Sprache zurück. An dieser Stelle soll nun Pullums Kritik kurz dargestellt und diskutiert werden.

Obwohl die mathematische Seite des Beweises fehlerfrei zu sein scheint und alle Schritte mit

großer Sorgfalt dargebracht wurden, behauptet Pullum, dass Higginbothams These bei genauerer Betrachtung nicht standhält. Seine empirischen Behauptungen über die englische Sprache, vor allem aber das Verbot der leeren Quantifizierung, also Behauptung (5.6), seien schlicht und ergreifend falsch. Higginbotham selbst gibt zunächst folgende Beispiele:

every book such that it rains (5.18)

the man such that I saw Mary (5.19)

Beispiele (5.18) und (5.19) seien nicht nur grammatikalisch falsch gebildete Nominalphrasen, sondern lassen auch keine mögliche Interpretation zu. Demgegenüber stellt er noch zwei weitere Nominalphrasen:

every triangle such that two sides are equal (5.20)

the number system such that 2 and 3 make 5 (5.21)

Für Higginbotham sind die Beispiele (5.20) und (5.21) ebenfalls keine grammatikalisch korrekten englischen Nominalphrasen, obwohl diese Ellipsen (also als Satzteile mit Auslassungen) darstellen und daher interpretiert werden können. Diese beiden Beispiele ändert Higginbotham dann nochmals ab:

every triangle such that two sides ***of it*** *are equal* (5.22)

the number system such that 2 and 3 make 5 ***therein*** (5.23)

Erst bei den Beispielen (5.22) und (5.23) spricht Higginbotham von grammatikalisch korrekten Nominalphrasen im Sinne seiner Prämissen der englischen Sprache. Higginbotham nennt schließlich auch Elemente der regulären Sprache L, die aber kein Teil des Englischen sind.

the woman such that the man such that the man such that she gave this to him gave this to this left is here $= X\ Y^2$ *she* $Z_3\ Z_4\ W$ (5.24)

Beispiel (5.24) sei kein Teil der englischen Sprache, weil wegen der Bedingung (5.6) jedes *man* ein passendes Bezugswort haben muss (also keine leere Quantifizierung erlaubt ist) und das *him* sich nicht auf zwei verschiedene *man* beziehen darf.

the woman such that the man such that the man such that she gave this to this gave him to him left is here $= X\ Y^2$ *she* $Z_4\ Z_1\ W$ (5.25)

Auch Beispiel (5.25) gehöre nicht zur englischen Sprache, weil im innersten Teil *the man such that she gave this to this* die Bedingung (5.6) verletzt wird, es sich also ebenfalls um eine leere Quantifizierung handelt.

Genau diese Behauptung (5.6) kritisiert Pullum, das Verbot einer leeren Quantifizierung in der englischen Sprache sei demnach falsch. Als Gegenbeispiel gibt Pullum zunächst folgenden Satz:

Over many years, it has become clear that Lee and Sandy were just one of those couples such that people always reported loving her but hating him. (5.26)

Dieser Satz enthält eine Nominalphrase der Form (*N such that D*), wobei *D* kein passendes Pronomen enthält, das sich auf $N = couples$ bezieht. Dennoch ist das Beispiel (5.26) für Pullum ein grammatikalisch vollkommen korrekter Satz. Später gibt Pullum noch ein weiteres Gegenbeispiel, das in einem Prosawerk des Historikers G. O. Trevelyan aus dem Jahre 1876 zu finden ist. Außerdem merkt Pullum an, dass sich Higginbotham offensichtlich der Existenz solcher Gegenbeispiele bewusst ist, da er (5.18) – (5.21) bzw. (5.24) und (5.25) auflistet, es aber nicht in Betracht zieht, dass alle diese Beispiele grammatikalisch korrekt sind.

Genau genommen widerlegen Gegenbeispiele wie (5.26) die Gleichheit $(L \cap \text{Englisch}) = A$ aber nicht, da es eines Gegenbeispiels zu Behauptung (5.6) aus der Sprache *L* bedarf. Eventuell könnte man Higginbothams Bedingung (5.6) sogar abschwächen, damit Sätze wie (5.26) integriert werden. Es gibt in diesem Fall zwar kein passendes Pronomen, dass sich auf *couples* bezieht, aber aus naheliegenden Gründen ließe sich das Vorkommen der beiden Pronomen *her* und *him* so interpretieren, dass sie sich zusammen auf das Nomen *couples* beziehen.

Entscheind ist, dass Pullum dann noch ein Gegenbeispiel anführt, dass in der Sprache *L* liegt. Er bedient sich dafür ebenfalls des Satzes (5.24) und meint, dass solche Sätze bzw. Nominalphrasen Teil der englischen Sprache sind (siehe [10, 185]).

Laut Pullum scheitert Higginbothams Beweis also an ebendieser Tatsache, dass Nebensätze, die mit *such that* eingeleitet werden, kein passendes Bezugswort zum Hauptsatz haben müssen, eine leere Quantifizierung in der englischen Sprache also sehr wohl erlaubt ist. Somit muss die Bedingung an Wörter der Menge *A*, dass die Anzahl der *this* die Anzahl der *him*, von links nach rechts gelesen, nie um mehr als 1 übersteigt, nicht zwangsweise auch auf Wörter der Menge *L* zutreffen, damit sie Teil des Englischen sind. Pullum behauptet, es gäbe zum damaligen Zeitpunkt jedenfalls keinen Grund annzunehmen, dass die englische Sprache nicht kontextfrei ist.

Andererseits erscheinen Higginbothams Annahmen über die englische Sprache nicht absurd. Aber nachdem natürliche Sprachen immer im Wandel sind, muss man sich dessen bewusst sein, dass dieser Beweis mit den Regeln der englischen Grammatik steht und fällt. Doch wenn man davon ausgeht, dass Higginbothams Annahmen (5.6) und (5.9) über die englische Sprache richtig sind, kann man den Beweis jedenfalls so führen.

6 Schlussbemerkungen

Das Ziel dieser Masterarbeit war es, den Beweis von James Higginbotham [7], dass Englisch keine kontextfreie Sprache ist, nachvollziehen zu können und zu analysieren. Dafür musste Vorarbeit geleistet werden, indem man sich mit freien Monoiden, formalen Sprachen, Phrasenstrukturgrammatiken, der Chomsky-Hierarche und Automaten beschäftigte. Im Hauptteil der Arbeit wurde Higginbothams Beweis dann in mehrere Behauptungen unterteilt, die schrittweise gezeigt wurden, und es wurde versucht, seiner Argumentation genau zu folgen.

Natürlich wirft eine solche Untersuchung auch neue Fragen auf. Wenn Englisch keine kontextfreie Sprache ist, ist Englisch dann eine kontextsensitive Sprache? Kann man überhaupt eine (generative) Grammatik angeben, die das Englische erzeugt? Sind natürliche Sprachen zu komplex, um eine *formale* erzeugende Grammatik zu finden? Nachdem man im Deutschen ähnliche Relativkonstruktionen erfinden kann, gibt es dann eine Sprache, deren Schnitt mit der deutschen Sprache ebenfalls nicht kontextfrei ist? Wie könnte so ein Verfahren bei Sprachen aussehen, die anders aufgebaut sind (z. B. den agglutinierenden Sprachen wie Finnisch, Ungarisch oder Türkisch)? Lässt sich diese Aussage verallgemeinern und könnte es sein, dass alle natürlichen Sprachen kontextfrei sind?

Keine dieser Fragen scheint abschließend geklärt zu sein, und Higginbotham war nicht der einzige, der sich mit solch einer Fragestellung beschäftigte. In seiner Publikation *Three Models for the Description of Language* (1956) [2] stellte Noam Chomsky selbst schon die Frage: Wenn menschliche Sprachen lediglich als Mengen von Wörtern betrachtet werden (sogenannte *string-sets*), fallen sie dann immer unter die Klasse der kontextfreien Sprachen? Chomsky hat zugegeben, die Antwort auf diese Frage nicht zu kennen.

Auch Geoffrey K. Pullum beschäftigte sich in *Natural Languages and Context-Free Languages* (1982) [11] mit Chomskys ursprünglicher Fragestellung und dem Zusammenhang natürlicher und kontextfreier Sprachen. In diesem Artikel behauptet er, dass alle bisher veröffentlichten Gegenargumente entweder empirisch oder formal falsch seien. Demnach handelt es sich bei der Problemstellung, ob eine natürliche Sprache (z. B. das Englische) kontextfrei ist, immer noch um eine offene und unbeantwortete Fragestellung. In demselben Jahr wie Higginbotham veröffentlichte Pullum dann den Artikel *On Two Recent Attempts to Show that English Is Not a CFL* (1984) [10], in dem er unter anderem die Argumentation von Higginbothams Beweis scharf kritisiert (siehe Kapitel 5.5).

In *Evidence Against the Context-Freeness of Natural Language* (1988) [12] findet Stuart M. Shieber, ein Informatiker und Computerlinguist, mehrere Argumente gegen die Kontextfreiheit natürlicher Sprachen, die Beispiele in seinem Artikel stammen dabei aus dem Schweizerdeutschen.

Manchmal wird im Bezug auf die Kontextfreiheit natürlicher Sprachen auch zwischen *schwachen* und *starken* kontextfreien Sprachen unterschieden. So behaupten Bresnan et. al. in *Cross-Serial Dependencies in Dutch* (1982) [1], dass die holländische Sprache nicht streng kontextfrei sei. Die Frage nach der schwachen Kontextfreiheit des Holländischen wird jedoch offen gelassen.

Diese Masterarbeit stellt schließlich keinesfalls den Anspruch zu behaupten, dass Englisch eine kontextfrei Sprache ist oder nicht. Der entscheidende Punkt des dargestellten Beweises liegt in Higginbothams Annahmen über die englische Sprache, nämlich Behauptungen (5.6) und (5.9), denen Pullum beispielsweise nicht zustimmt. Aber unter diesen Annahmen hält Higginbothams Beweis einer mathematischen Analyse stand und kann somit als richtig erachtet werden. In Hinblick auf moderne Anwendungsmöglichkeiten, wie der linguistischen Datenverarbeitung (auch *Natural language processing (NLP)* genannt), erscheint es grundsätzlich lohnenswert, weiter in diese Richtung zu arbeiten.

Literaturverzeichnis

[1] J. Bresnan, R. M. Kaplan, S. Peters und A. Zaenen: *Cross-Serial Dependencies in Dutch*. Linguistic Inquiry, 13(4):613–635, 1982.

[2] N. Chomsky: *Three Models for the Description of Language*. I. R. E. Transactions on Information Theory, IT-2:113–124, 1956.

[3] P. M. Cohn: *Further Algebra and Applications*. Springer, 2003.

[4] M. Gross und A. Lentin: *Mathematische Linguistik*. Springer, 1971.

[5] M. A. Harrison: *Introduction to Formal Language Theory*. Addison-Wesley, 1978.

[6] U. Hedtstück: *Einführung in die Theoretische Informatik*. Oldenbourg, 2012.

[7] J. Higginbotham: *English Is Not a Context-Free Language*. Linguistic Inquiry, 15(2):225–234, 1984.

[8] J. Hopcroft, R. Motwani und J. D. Ullman: *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie*. Pearson Studium, 2002.

[9] R. Lidl und G. Pilz: *Angewandte abstrakte Algebra I*. B.I.-Wissenschaftsverlag, 1982.

[10] G. K. Pullum: *On Two Recent Attempts to Show that English Is Not a CFL*. Computational Linguistics, 10(3-4):182–186, 1984.

[11] G. K. Pullum und G. Gazdar: *Natural Languages and Context-Free Languages*. Linguistics and Philosophy, 4(4):471–504, 1982.

[12] S. M. Shieber: *Evidence Against the Context-Freeness of Natural Language*. Kluwer Academic Publishers, 1988.

[13] R. Winter: *Theoretische Informatik*. Oldenbourg, 2002.