a^ib^i und eine beliebige Anzahl von a's zum Schluß, und L_2 erzeugt b^ia^i und eine beliebige Anzahl von a's vorneweg. Also ist $a^ib^ia^i = L$ die Schnittmenge von L_1 und L_2. L ist aber nicht kontextfrei.

Anzumerken ist noch, daß der Schnitt einer kontextfreien Sprache mit einer regulären Sprache immer eine kontextfreie Sprache ist. Der Beweis findet sich in Hopcroft & Ullman (1994). Wir können also z.B., wenn wir für ein Sprachfragment eine kontextfreie Grammatik formuliert haben und für ein weiteres Sprachfragment eine reguläre Grammatik, für den Schnitt beider Fragmente nur eine kontextfreie Grammatik formulieren. Hätten wir zwei kontextfreie Sprachen, könnten wir uns bei dem Schnitt beider Sprachen nicht sicher sein, daß wir für diesen automatisch eine kontextfreie Grammatik formulieren können.

Komplement: Kontextfreie Sprachen sind nicht unter Komplementbildung abgeschlossen. Wären sie unter Komplementbildung abgeschlossen, müßten sie nach dem DeMorganschen Gesetz auch unter Schnittbildung abgeschlossen sein, was unserem Beweis widerspricht.

Differenz: Ähnlich verhält es sich mit der Differenz. Auf Grund von

$$L_1 \cap L_2 = (L_1 \cup L_2) \setminus ((L_1 \setminus L_2) \cup (L_2 \setminus L_1))$$

wären die beiden Sprachen wiederum unter Schnittbildung abgeschlossen, wenn sie unter Differenz abgeschlossen wären.

4.4 Das *Pumping Lemma* für kontextfreie Sprachen

Wir werden jetzt das Pumping Lemma für kontextfreie Sprachen kennenlernen, das dem Pumping Lemma für reguläre Sprachen ähnelt. Während bei den regulären Sprachen die Idee des Pumping Lemmas darin bestand, einen Teil einer Zeichenkette „aufzupumpen" – i.e. beliebig oft zu kopieren – und das Ergebnis wieder eine reguläre Menge ist, besteht die Idee bei kontextfreien Sprachen darin, *zwei* Teile einer Zeichenkette beliebig zu wiederholen, und das Ergebnis ist immer noch ein Wort derselben kontextfreien Sprache. Warum ist das möglich?

Wenn wir einen genügend großen Syntaxbaum einer kontextfreien Grammatik in Chomsky–Normalform nehmen, taucht mindestens ein nichtterminales Symbol mehrmals auf: Für kontextfreie Grammatiken in CNF mit k nichtterminalen Symbolen enthält jeder Ableitungsbaum für Wörter der Mindestlänge 2^{k+1} einen Weg mit $k+1$ nichtterminalen Symbolen, d.h. mindestens eine Variable A_i taucht mehrmals auf. Ein Wort z läßt sich dann in Teilwörter $uvwxy$ aufteilen mit den Ableitungen: $S \xrightarrow{*} uA_iy, A_i \xrightarrow{*} vA_ix$ und $A_i \xrightarrow{*} w$ mit $u, v, w, x, y \in \Sigma^*$. Die Ableitung $A_i \xrightarrow{*} vA_ix$ kann dann entweder weggelassen oder beliebig oft wiederholt werden. Mit anderen Worten: Wörter der Form uv^iwx^iy mit $i \geq 0$ sind ebenfalls Wörter der von der Grammatik erzeugten Sprache.

Lemma 4.1
Sei L eine kontextfreie Sprache. Dann gibt es eine Konstante n, so daß für jedes Wort $z = uvwxy \in L$ mit $|z| \geq n$ gilt:

1. $|vx| \geq 1$
   (Die uns interessierende Teil–Zeichenkette enthält mindestens ein Zeichen.)

2. $|vwx| \leq n$
   (Die uns interessierende Teil–Zeichenkette ist höchstens n Zeichen lang.)

3. für alle $i \geq 0$ liegt uv^iwx^iy in L
   (Die i–mal kopierte Zeichenkette ist ebenfalls ein Wort der Sprache L.)

Beweis des Lemmas 4.1
Der Beweis erfolgt durch Induktion über i. Wir nehmen an, daß eine Grammatik in Chomsky–Normalform $L - \{\epsilon\}$ erzeugt. Da Grammatiken in Chomsky–Normalform binär verzweigend sind, haben Ableitungsbäume die in Abbildung (28) angegebene Gestalt. Bäume der Höhe i können höchstens Wörter der Länge 2^i ableiten. Der Baum in Abbildung (28) hat die Höhe 3, so daß er höchstens Wörter der Länge $2^3 = 8$ ableiten kann (siehe die Anzahl der Blätter des Baums).

Sei i die Anzahl der nichtterminalen Symbole in Φ_G und sei $n = 2^{i+1}$, so daß ein nichtterminales Symbol mehr als einmal vorkommt. Sei

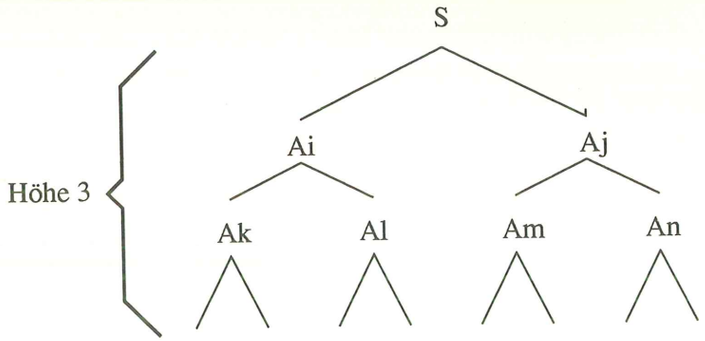

Abbildung 28: Ableitungsbaum–Schema für Grammatiken in Chomsky–Normalform

ferner $z = uvwxy \in L$ mit $|z| \geq n$. Der Ableitungsbaum für z hat dann die in Abbildung (29) angegebene Gestalt. Der Baum hat die Höhe $i + 1$, so daß eine Variable (in der Abbildung als A bezeichnet) zweimal vorkommt.

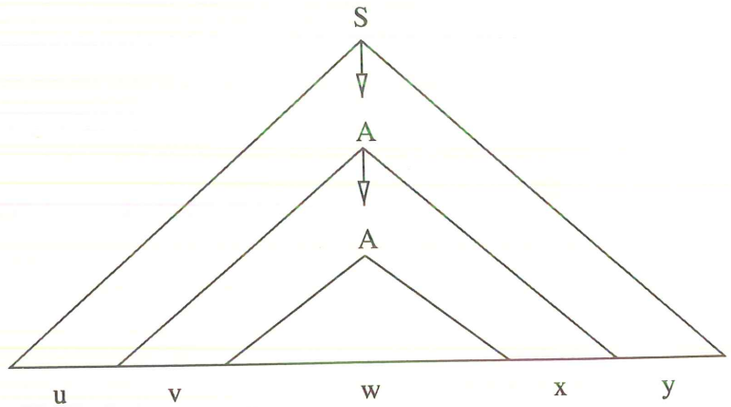

Abbildung 29: Ableitungsschema für $z = uvwxy$

Dann ist $vx \geq 1$ und uv^iwx^iy für alle $i \geq 0$ ein Wort aus L sowie $|vwx| \leq n$.

Mittels dieses Lemmas können wir jetzt, wie versprochen, den folgenden Satz beweisen:

Satz 4.3
Die Sprache $L = \{a^ib^ia^i \mid i \geq o\}$ ist nicht kontextfrei.

Beweis 4.3
Wir nehmen an, L sei kontextfrei und leiten daraus einen Widerspruch ab. Sei n die Längenschranke des Pumping Lemmas und sei $z = uvwxy = a^ib^ia^i$ mit $|vx| \geq 1$ und $|vwx| \leq n$.

Falls vwx kein a aus der zweiten a–Reihe enthält, enthält uwy zuviele a's. Falls vwx kein a aus der ersten a–Reihe enthält, enthält uwy zuviele a's.

Wie man es auch dreht und wendet, der Ableitungsbaum für L kann nicht der geforderten Gestalt entsprechen.

4.5 Nur eine andere Notation: Backus–Systeme

Wir haben bisher die kontextfreien Grammatiken als einen Mechanismus zur Erzeugung kontextfreier Sprachen kennengelernt. Eine weitere Form der Angabe kontextfreier Sprachen sind die sogenannten Backus–Systeme. Backus–Systeme wurden von J. Backus für die Beschreibung der Programmiersprache ALGOL entwickelt. Backus-Systeme werden häufig bei der Beschreibung von Programmier– und Wissensrepräsentationssprachen angewandt. Sie sind aber in ihrer Standardform nur andere Notationen für kontextfreie Grammatiken.

Dabei wird eine besondere Schreibweise für die Übergangsregeln eingeführt. Für $A \to w$ schreibt man $A ::= w$. Wenn es mehrere Regeln der Form $A \to w_1, A \to w_2, \ldots, A \to w_n$ gibt, kürzt man dies ab mit $A ::= w_1|w_2|\ldots|w_n$. Bei der Beschreibung der Syntax von Programmiersprachen wünscht man sich für die Benennung der Nichtterminalsymbole sinnvolle Namen anstelle von Zeichen. Der Name eines Nichtterminalsymbols wird in spitze Klammern geschrieben wie z.B. $\langle Ausdruck\rangle, \langle Programm\rangle$ usw.

Ein Beispiel hierzu ist die Beschreibung der Programmiersprache PROLOG II in Giannesini et al. (1986). Der Zeichensatz dieser Programmiersprache wird wie folgt beschrieben:

⟨Zeichen⟩ ::= ⟨Buchstabe⟩
⟨Zeichen⟩ ::= ⟨Ziffer⟩
⟨Zeichen⟩ ::= ⟨Sonderzeichen⟩
⟨Buchstabe⟩ ::= A|B|C|...|Z|a|b|c ...|z