Kontextfreie Sprachen (Typ 2)

Hans Jürgen Ohlbach

Keywords: Typ 2-Grammatiken, Chomsky Normalform, Typ 2-Pumping Lemma

Empfohlene Vorkenntnisse: Funktionen und Relationen, Formale Sprachen, Typ 3-Sprachen

Inhaltsverzeichnis

1 Einführung

Kontextfreie Grammatiken sind für viele Anwendung der Informatik eigentlich die brauchbarsten Grammatiken, ausdrucksstark genug, dass man interessante Sprachen damit spezifizieren kann, aber noch einigermaßen effizient parsbar. Insbesondere zur Spezifikation von Programmiersprachen eignen sie sich. Ein einfaches Beispiel ist:

Anweisung	$\rightarrow$	if-Anweisung | while-Anweisung | Zuweisung
if-Anweisung	$\rightarrow$	if(Bedingung) then {Anweisung} else {Anweisung}
while-Anweisung	$\rightarrow$	while(Bedingung) {Anweisung}
Zuweisung	$\rightarrow$	Variable := Term
Bedingung	$\rightarrow$	...
Term	$\rightarrow$	...

wobei man die Bedingungen und die Terme weiter definieren müsste.

Die Sprache $L = \{a^n b^n \mid n > 0\}$ die als Standardbeispiel dient, um mit dem Pumping Lemma zu beweisen, dass sie nicht regulär ist, kann durch die kontextfreie Grammatik

$$\text{S} \rightarrow \text{ab} \mid \text{aSb}$$

erzeugt werden.

In diesem Text werden nur allgemeine Eigenschaften von kontextfreien Sprachen besprochen. Die zwei Arten des Parsen, bottom-up mit dem CYK-Algorithmus, und top-down mit Kellerautomaten werden in separaten Texten besprochen. Ein konkreter Parsergenerator für kontextfreie Sprachen ist der ANTLR-Parsergenerator, der ebenfalls in einem separaten Text vorgestellt wird.

2 Chomsky Normalform

Jede Typ 2-Grammatik lässt sich in eine spezielle Normalform umformen, die *Chomsky Normalform* Hierbei gibt es nur zwei Typen von Produktionen:

$$\text{A} \rightarrow \text{a}$$
$$\text{A} \rightarrow \text{BC}$$

wobei A,B,C Variable (Nicht-Terminale) sind, und a eine Konstante (Terminalsymbol) ist.

Diese Normalform ist sehr ähnlich zu den Typ 3-Grammatiken mit den Produktionen

$$\text{A} \rightarrow \text{a}$$
$$\text{A} \rightarrow \text{bC}$$

Die Umformung einer beliebigen Typ 2-Grammatik in Chomsky Normalform geschieht in folgenden Schritten:

Separieren: Für jede Konstante a führe eine Variable A ein, mit der Produktion $A \rightarrow a$.
Alle Vorkommnisse von a werden durch A ersetzt.
Jetzt gibt es nur noch Variablen in den längeren rechten Seiten der Produktionen.

ϵ-Produktionen eliminieren:
Für jede Produktion der Art $A \rightarrow \epsilon$:
Kopiere alle Produktionen mit A auf der rechten Seite, und lasse das A weg.
z.B. für $B \rightarrow CAD$ wird zusätzlich $B \rightarrow CD$ hinzugefügt.

Zum Schluss bleibt höchstens noch $S \rightarrow \epsilon$ für die Startvariable S übrig.

Zerlegen der rechten Seite mit Länge > 2: Für alle Produktionen der Gestalt $A \rightarrow B_1 \ldots B_k$: führe neue Variablen ein und ersetze die Produktion durch

$$\begin{aligned} A &\rightarrow B_1 C_2 \\ C_2 &\rightarrow B_2 C_3 \\ \ldots &\rightarrow \ldots \\ C_{k-1} &\rightarrow B_{k-1} B_k \end{aligned}$$

Zyklen beseitigen: Bei Zyklen der Art $A_1 \to A_2$, $A_2 \to A_3 \ldots A_n \to A_1$ können alle Variablen A_i durch A_1 ersetzen.

Kettenproduktionen beseitigen:
Produktionen der Art $A_1 \to A_2$, $A_2 \to A_3 \ldots A_n \to x_1 \mid \ldots \mid x_k$, wobei die x_i Variablen oder Konstanten sein können, kann man ersetzen durch $A_i \to x_1 \mid \ldots \mid x_k$ für alle A_i.

ϵ-Produktionen können für die Formulierung einer Grammatik sehr nützlich sein. Für einige Algorithmen sind sie aber problematisch und müssen vorher eliminiert werden. Ihre Eliminierung zeigen wir an folgendem Beispiel:

Die Grammatik

Satz	$\to$	Subjekt Prädikat Objekt	Adjektiv	$\to$	kleine
Subjekt	$\to$	Artikel Attribut Substantiv	Adjektiv	$\to$	bissige
Artikel	$\to$	ϵ	Adjektiv	$\to$	große
Artikel	$\to$	der	Substantiv	$\to$	hund
Artikel	$\to$	die	Substantiv	$\to$	katze
Artikel	$\to$	das	Prädikat	$\to$	jagt
Attribut	$\to$	ϵ	Objekt	$\to$	Artikel Attribut Substantiv
Attribut	$\to$	Adjektiv Attribut			

wird zu

Satz	$\to$	Subjekt Prädikat Objekt	Adjektiv	$\to$	kleine
Subjekt	$\to$	Artikel Attribut Substantiv	Adjektiv	$\to$	bissige
Subjekt	$\to$	Attribut Substantiv	Adjektiv	$\to$	große
Subjekt	$\to$	Artikel Substantiv	Substantiv	$\to$	hund
Subjekt	$\to$	Substantiv	Substantiv	$\to$	katze
Artikel	$\to$	der	Prädikat	$\to$	jagt
Artikel	$\to$	die	Objekt	$\to$	Artikel Attribut Substantiv
Artikel	$\to$	das	Objekt	$\to$	Attribut Substantiv
Attribut	$\to$	Adjektiv Attribut	Objekt	$\to$	Substantiv
Attribut	$\to$	Adjektiv	Objekt	$\to$	Artikel Substantiv

Ein Beispiel für die CNF-Umformung ist die Grammatik $S \to ab|aSb$,
welche die Sprache $\{a^n b^b \mid n > 0\}$ erzeugt.

Separieren:

S	$\to$	AB
S	$\to$	ASB
A	$\to$	a
B	$\to$	b

Lange Produktionen verkürzen:

$$\begin{array}{lcl} S & \to & AB \\ S & \to & AC \\ C & \to & SB \\ A & \to & a \\ B & \to & b \end{array}$$

Die Chomsky Normalform hat u.a. die Eigenschaft, dass ihre Syntaxbäume Binärbäume sind. Das wird z.B. beim Beweis des Pumping Lemmas ausgenutzt. Der CYK-Algorithmus zum bottom-up Parsing basiert auch auf der Chomsky Normalform.

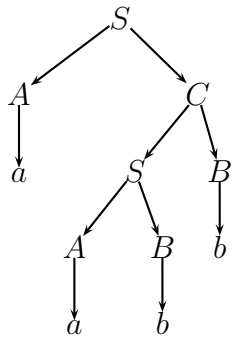
Neben der Chomsky Normalform für Typ 2-Grammatiken gibt es noch die *Greibach Normalform*, wo alle Produktionen die Gestalt $A \to aB_1 \dots B_k$, $(k \geq 0)$ haben. Diese scheint aber weniger nützlich zu sein als die Chomsky Normalform.

3 Das Pumping Lemma

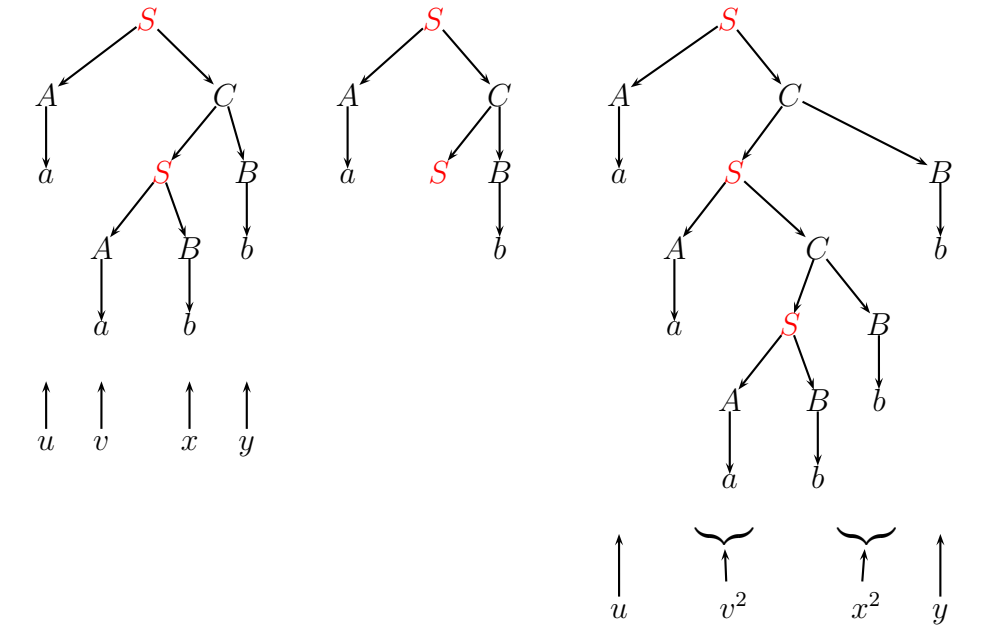
Analog zum Pumping Lemma für Typ 3-Sprachen, besagt das Pumping Lemma für Typ 2-Sprachen ebenfalls, dass man in langen Wörtern bestimmte Teile eines Wortes „aufpumpen“ kann. Wir illustrieren das an der Chomsky Normalform der Grammatik für die Sprache $\{a^n b^n \mid n > 0\}$:

$$\begin{array}{lcl} S & \to & AB \\ S & \to & AC \\ C & \to & SB \end{array} \qquad \begin{array}{lcl} A & \to & a \\ B & \to & b \end{array}$$

Der Syntaxbaum für das Wort $aabb$ sieht folgendermaßen aus.



In diesem Baum kommt die Variable S zweimal vor. Wir könnten also den unteren Teilbaum unter der Variablen S herausschneiden, und durch den ganzen Baum ersetzen.



Aus dem Wort aabb kann man also durch herausschneiden und hineinkopieren das Wort aaabbb erzeugen. Wie man an dem Syntaxbaum sieht, lässt sich das beliebig wiederholen.

Das war jetzt ein Spezialfall. Im allgemeinen Fall kann man auf ganz analoge Weise ein genügend großes Wort z in 5 Teile zerlegen: $z = uvwxy$, und dann v und x synchron aufpumpen: uv^iwx^iy. (Im Beispiel oben war der Mittelteil w leer.)

Theorem 1 (Pumping Lemma für Typ 2-Sprachen) *Für jede kontextfreie Sprache L gibt es eine Pumpingzahl n, so dass alle Wörter $z \in L$ mit $|z| \geq n$ sich zerlegen lassen in*

$$z = uvwxy$$

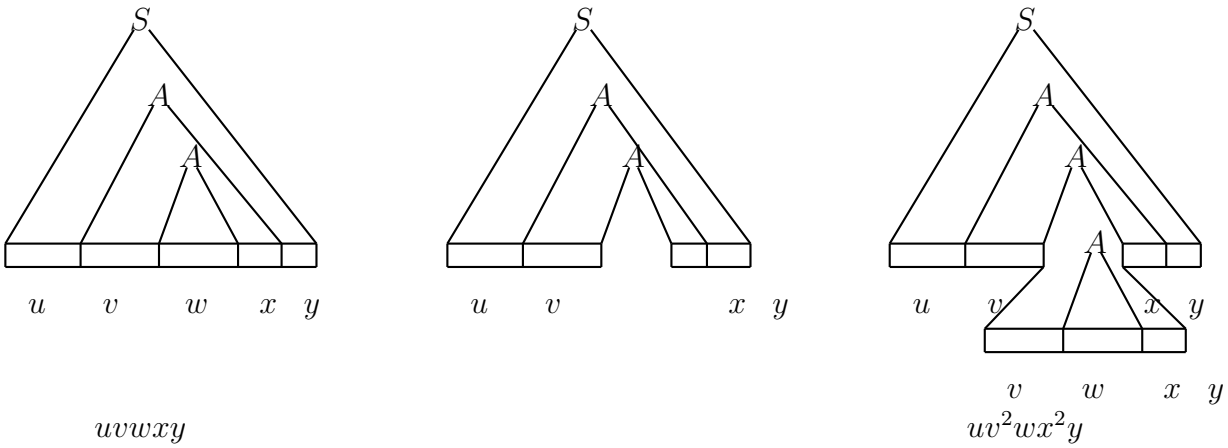
mit folgenden Eigenschaften:

- $|vx| \geq 1$
- $|vwx| \leq n$
- $\forall i : uv^iwx^iy \in L$

Der Beweis geht über die Syntaxbäume für die Chomsky Normalform der Sprache L. Die Syntaxbäume sind Binärbäume. Ein solcher Baum der Tiefe k hat in der Größenordnung 2^k Blätter. Die Blätter sind die Wörter der Sprache. Wählt man die Pumpingzahl $n = 2^{\text{Anzahl Variablen der Grammatik}+1}$, dann ist die Baumtiefe für ein Wort dieser Länge größer oder gleich der Anzahl Variablen der Grammatik + 1. Das bedeutet, dass mindestens eine Variable A doppelt vorkommen muss (wie die Variable S im Beispiel oben). Den Teilbaum unterhalb des unteren Vorkommnisses von A kann man dann herausschneiden, und durch den Teilbaum des oberen Vorkommnisses ersetzen. Das lässt sich beliebig oft wiederholen.

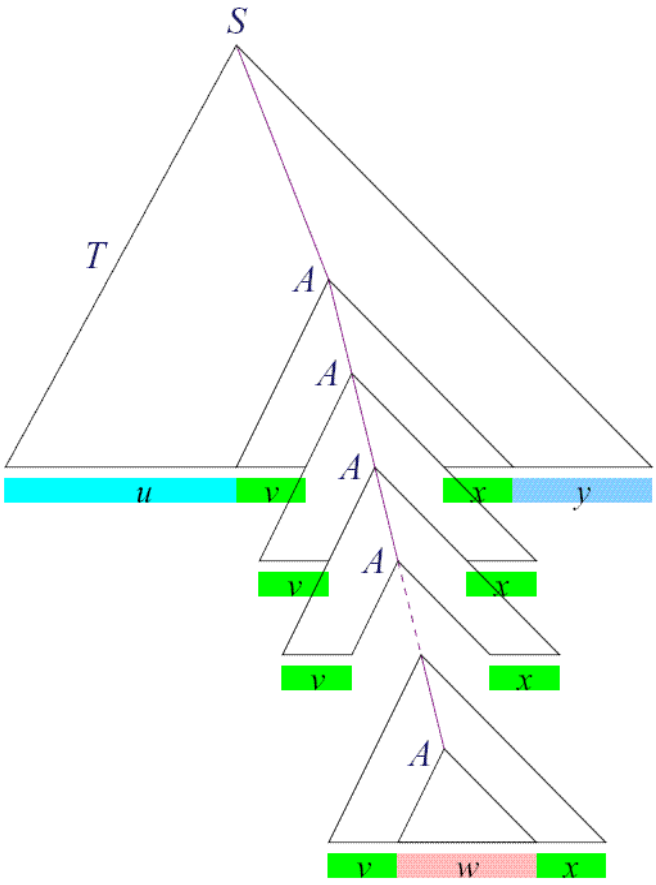
Schematisch sieht das so aus:

Original	A herausgeschnitten	A reinkopiert



Nur die v- und y-Teile werden synchron aufgepumpt. Der w-Teil, welcher auch leer sein kann, wird nicht aufgepumpt. Die Anfangs- und End-Teile u und y werden auch nicht gepumpt.

Das folgende Bild illustriert den „Pumpeffekt“ noch deutlicher.



Wie beim Pumping Lemma für Typ 3-Sprachen, kann man auch mit diesem Pumping Lemma *nicht folgern*, dass eine Sprache vom Typ 2 ist, sondern nur *widerlegen*, dass eine Sprache vom Typ 2 ist. Für die Sprache $L = \{a^n b^m c^m d^m \mid n, m > 0\}$ gilt nämlich das Pumping Lemma, aber sie ist nicht vom Typ 2.

Der Widerlegungsbeweis, dass eine Sprache L nicht vom Typ 2 ist, geschieht nach dem gleichen Schema wie bei Typ 3:

- Angenommen, die Sprache ist vom Typ 2.
- dann gilt das Pumping Lemma.
- Sei n die Pumping Zahl.
- Wir wählen ein geeignetes genügend langes Wort.
- Wir zeigen, dass *jede* Zerlegung nach dem Pumping Lemma beim Aufpumpen aus der Sprache L herausführt.
- Das ist ein Widerspruch zur Annahme, dass das Pumping Lemma gilt.
- Also ist die Sprache nicht vom Typ 2.

Als Beispiel zeigen wir, dass die Sprache $L = \{a^m b^m c^m \mid m > 0\}$ nicht vom Typ 2 ist.

- Angenommen, L ist vom Typ 2
- dann gilt das Pumping Lemma
- Sei n die Pumping Zahl
- Sei $z = a^n b^n c^n$
- Für die Zerlegung $z = uvwxy$ nutzen wir die Bedingung $|vwx| \leq n$ aus:
  Fall 1: vwx liegt völlig im a-Teil von z. Dann werden nur a's gepumpt, aber keine b's und c's.
  Fall 2: vwx liegt völlig im b-Teil von z. Dann werden nur b's gepumpt, aber keine a's und cs.
  Fall 3: vwx liegt völlig im c-Teil von z. Dann werden nur c's gepumpt, aber keine a's und bs.
  Fall 4: vwx überlappt den a und b-Teil von z. Dann werden nur a's und b's gepumpt, aber keine cs.
  Fall 5: vwx überlappt den b und c-Teil von z. Dann werden nur b's und c's gepumpt, aber keine a's.
  Mehr Fälle gibt es nicht. Jeder führt zum Widerspruch.
- Das ist ein Widerspruch zur Annahme, dass das Pumping Lemma gilt.
- Also ist L nicht vom Typ 2.

Hinweis: Ein häufige Fehlerquelle bei solchen Beweisen, ist es, eine Zerlegungsmöglichkeit zu übersehen, und damit einen falschen Beweis zu produzieren.

Ein Versuch, für die Sprache $L = \{a^{2m}b^{m+2} \mid m > 0\}$ einen Widerlegungsbeweis mit dem Pumping Lemma zu führen fängt ganz richtig mit der Wahl des Wortes $z = a^{2n}b^{n+2}$ für die Pumpingzahl n an. Eine Reihe von Zerlegungsmöglichkeiten führen auch tatsächlich zu einem Widerspruch. Es gibt aber eine Zerlegung, die sich aufpumpen lässt, nämlich

$$z = \overbrace{\underbrace{a\ldots}_{u}\underbrace{aa}_{v}}^{2n}\overbrace{\underbrace{b}_{x}\underbrace{\ldots b}_{y}}^{n+2}$$

Hier kann man synchron je zwei a's und ein b pumpen, und bleibt in der Sprache L.

4 Das Wortproblem bei Typ 2-Sprachen

Zur Lösung des Wortproblems für Typ 2-Sprachen gibt es zwei Ansätze: bottom-up mit dem CYK-Algorithmus, und top-down mit Kellerautomaten. Beide werden in separaten Dokumenten vorgestellt.

Die Anwendung der Kellerautomaten auf das Wortproblem wird, ähnlich wie bei Typ 3-Sprachen, wiederum das Problem aufwerfen, ob die Automaten deterministisch oder nichtdeterministisch arbeiten. Leider ist es bei den Kellerautomaten nicht möglich, nichtdeterministische Automaten deterministisch zu machen.

Man sieht das sehr deutlich an den Palindromsprachen:
$L_1 = \{ww^{-1} \mid w \in \Sigma^+\}$ und (w^{-1} ist w rückwärts)
$L_2 = \{w\$w^{-1} \mid w \in \Sigma^+\}$.

L_1 definiert Palindrome ohne Mittensymbol, z.B. otto.
L_2 definiert dagegen Palindrome mit Mittensymbol, z.B. ot$to.

Um diese Palindrome zu Parsen, muss man sich das Anfangsstück merken, um es mit dem Endstück vergleichen zu können. Das geht aber nur, wenn man die Mitte erkennen kann. Bei L_2 sieht man das am $-Zeichen. Bei L_1 muss man die Mitte raten (wenn man keine Vorausschau auf das Gesamtwort hat).

Daher unterscheidet man die Typ-2 Sprachen in
Deterministisch kontextfreie Sprachen und *Nichtdeterministisch kontextfreie Sprachen*.

5 Abschlusseigenschaften

Man bezeichnet einen Sprachtyp als *Abgeschlossen unter der Operation* $X(L_1, \ldots, L_n)$, falls die Operation X den Typ der Sprachen erhält. Im gegenwärtigen Fall geht es um den Sprachtyp *kontextfrei*, und darum ob verschiedene Kombinationen von kontextfreien Sprachen wieder kontextfreie Sprachen ergeben.

Kontextfreie Sprachen sind abgeschlossen unter

- Vereinigung
- Produkt (Aneinanderhängen der Wörter)
- Stern (Beliebige Wiederholungen)

Sie sind nicht abgeschlossen unter

- Schnitt
- Komplement

Die Abschlüsse bekommt man meist durch Manipulation der beteiligten Grammatiken.

Abschluss unter Vereinigung: Zwei Grammatiken $G_1 = (V_1, \Sigma, P_1, S_1)$ und $G_1 = (V_1, \Sigma, P_1, S_1)$ mit disjunkten Variablen (kann man immer machen) vereinigt man, indem man eine neue Startvariable S einführt, mit der Produktion $S \to S_1 \mid S_2$. Damit werden sowohl Wörter aus G_1 als auch aus G_2 erzeugt. Die Vereinigungsgrammatik ist dann

$$G = (V_1 \cup V_2 \cup S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1 \mid S_2\}, S)$$

und die ist wieder vom Typ 2.

Beispiel:

S_1	$\to$	$ab \mid aS_1b$	(erzeugt $\{a^nb^n \mid n > 0\}$)
S_2	$\to$	$aa \mid bb \mid aS_2a \mid bS_2b$	(erzeugt$\{ww^{-1} \mid w \in \{a,b\}+\}, w^{-1}$ ist w rückwärts)
S	$\to$	$S_1 \mid S_2$	

Abschluss unter Produkt: Aus zwei Grammatiken $G_1 = (V_1, \Sigma, P_1, S_1)$ und $G_1 = (V_1, \Sigma, P_1, S_1)$ mit disjunkten Variablen kann men eine Sprache machen, deren Wörter aus einem G_1-Teil gefolgt von einem G_2-Teil besteht, indem man eine neue Startvariable S mit der Produktion $S \to S_1S_2$ hinzufügt. Die Produktgrammatik ist dann

$$G = (V_1 \cup V_2 \cup S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1S_2\}, S)$$

und die ist wieder vom Typ 2.

Beispiel:

S_1	$\to$	$ab \mid aS_1b$	(erzeugt $\{a^nb^n \mid n > 0\}$)
S_2	$\to$	$aa \mid bb \mid aS_2a \mid bS_2b$	(erzeugt$\{ww^{-1} \mid w \in \{a,b\}+\}, w^{-1}$ ist w rückwärts)
S	$\to$	S_1S_2	(erzeugt $\{a^nb^nww^{-1} \mid n > 0, w \in \{a,b\}+\}$)

Abschluss unter Stern: Die Wörter einer Grammatik $G = (V, \Sigma, P, S)$ kann man mit folgender Typ 2-Grammatik beliebig oft aneinanderhängen,

$$G = (V \cup \{S'\}, \Sigma, P \cup \{S' \to \epsilon, S' \to S, S' \to S'S'\}, S')$$

Beispiel:

S	$\to$	$ab \mid aSb$	(erzeugt $\{a^n b^n \mid n > 0\}$)
S'	$\to$	ϵ	erzeugt das leere Wort
S'	$\to$	S	einfache Wiederholung
S'	$\to$	$S'S'$	mehrfache Wiederholung

Falls die Ausgangsgrammatik das leere Wort produziert, sollte man $S \to \epsilon$ vorher entfernen.

Gegenbeispiel für Abschluss unter Schnitt: Die beiden Sprachen $L_1 = \{a^i b^j c^j \mid i, j > 0\}$ und $L_1 = \{a^i b^i c^j \mid i, j > 0\}$ sind kontextfrei. Deren Schnitt $L_1 \cap L_2 = \{a^i b^i c^i \mid i > 0\}$ ist jedoch nicht mehr kontextfrei, wie wir mit dem Pumping Lemma bewiesen haben.

Kein Abschluss unter Komplement: Rein mit mengentheoretischen Mitteln kann man jetzt zeigen, dass kontextfreie Sprachen nicht unter Komplement abgeschlossen sein können.

Es ist nämlich $L_1 \cap L_2 = (L_1' \cup L_2')'$

Wenn die Sprachen unter Komplement abgeschlossen wären, dann wären sie durch diese Umformung auch abgeschlossen unter Schnitt, was aber nicht der Fall ist. Also können sie nicht abgeschlossen unter Komplement sein.

Stichwortverzeichnis