

Vorlesung  
**Grundlagen der Theoretischen Informatik /  
Einführung in die Theoretische Informatik I**

**Bernhard Beckert**

Institut für Informatik



**Sommersemester 2007**

## Gleichmächtigkeit: PDAs und kontextfreie Grammatiken

### Lemma 5.15 (PDA $\rightarrow$ cf-Grammatik)

Zu jedem Push-Down-Automaten  $\mathcal{M}$  gibt es eine kontextfreie Grammatik  $G$  mit

$$L(G) = L(\mathcal{M})$$

### Beweis

Sei  $\mathcal{M}$  ein PDA, der eine Sprache  $L$  **über leeren Keller** akzeptiert.  
Wir konstruieren aus dem Regelsatz von  $\mathcal{M}$  eine kontextfreie Grammatik,  
die  $L$  erzeugt.

## Dank

Diese Vorlesungsmaterialien basieren ganz wesentlich auf den Folien zu den Vorlesungen von

**Katrin Erk** (gehalten an der Universität Koblenz-Landau)

**Jürgen Dix** (gehalten an der TU Clausthal)

Ihnen beiden gilt mein herzlicher Dank.

– *Bernhard Beckert, April 2007*

## Gleichmächtigkeit: PDAs und kontextfreie Grammatiken

### Beweis (Forts.)

**Idee:**

Die Variablen der Grammatik sind 3-Tupel der Form

$$[q, A, p]$$

**Bedeutung:**

Grammatik kann Wort  $x$  aus Variablen  $[q, A, p]$  ableiten

gdw

$\mathcal{M}$  kann vom Zustand  $q$  in den Zustand  $p$  übergehen,  
dabei  $A$  vom Keller entfernen (sonst den Keller unverändert lassen) und  
das Wort  $x$  lesen:

$$([q, A, p] \Longrightarrow^* x) \quad \underline{\text{gdw}} \quad ((q, x, A\gamma) \vdash^* (p, \varepsilon, \gamma))$$

## Gleichmächtigkeit: PDAs und kontextfreie Grammatiken

### Beweis (Forts.)

#### Formale Konstruktion:

Sei

$$\mathcal{M} = (K, \Sigma, \Gamma, \Delta, s_0, Z_0, F)$$

ein PDA.

Daraus konstruiert man die Grammatik

$$G = (V, T, R, S)$$

mit

$$V := \{[q, A, p] \mid q, p \in K, A \in \Gamma\} \cup \{S\}$$
$$T := \Sigma$$

und ...

## Gleichmächtigkeit: PDAs und kontextfreie Grammatiken

### Beweis (Forts.)

... folgenden Regeln in  $R$ :

- $S \rightarrow [s_0, Z_0, q]$  für alle  $q \in K$ ,
- $[q, A, q_{m+1}] \rightarrow a [q_1, B_1, q_2][q_2, B_2, q_3] \dots [q_m, B_m, q_{m+1}]$   
für jeden  $\Delta$ -Übergang  $(q, a, A) \Delta (q_1, B_1 \dots B_m)$  und  
für jede beliebige Kombination  $q_2, \dots, q_{m+1} \in K$ ,
- $[q, A, q_1] \rightarrow a$   
für jeden  $\Delta$ -Übergang  $(q, a, A) \Delta (q_1, \varepsilon)$

Dabei ist  $a \in \Sigma \cup \{\varepsilon\}$ .

Siehe Buch für Beweis, dass die Konstruktion das gewünschte Ergebnis liefert:

$$([q, A, p] \Longrightarrow^* x) \text{ gdw } ((q, x, A) \vdash^* (p, \varepsilon, \varepsilon))$$

woraus sofort  $L_\ell(\mathcal{M}) = L(G)$  folgt.

## Gleichmächtigkeit: PDAs und kontextfreie Grammatiken

### Beispiel 5.16

Sprache:

$$L_{ab} = \{a^n b^n \mid n \in \mathbb{N}_0\}$$

$L_{ab}$  wird über leeren Keller akzeptiert von dem PDA

$$\mathcal{M} = (\{s_0, s_1\}, \{a, b\}, \{Z_0, A\}, s_0, Z_0, \emptyset)$$

mit den Regeln

1.  $(s_0, \varepsilon, Z_0) \Delta (s_0, \varepsilon)$
2.  $(s_0, a, Z_0) \Delta (s_0, A)$
3.  $(s_0, a, A) \Delta (s_0, AA)$
4.  $(s_0, b, A) \Delta (s_1, \varepsilon)$
5.  $(s_1, b, A) \Delta (s_1, \varepsilon)$

## Gleichmächtigkeit: PDAs und kontextfreie Grammatiken

### Beispiel (Forts.)

Die Transformation ergibt folgende Grammatik-Regeln:

- $$S \rightarrow [s_0, Z_0, s_0] \mid [s_0, Z_0, s_1]$$
1.  $[s_0, Z_0, s_0] \rightarrow \varepsilon$
  2.  $[s_0, Z_0, s_0] \rightarrow a[s_0, A, s_0]$   
 $[s_0, Z_0, s_1] \rightarrow a[s_0, A, s_1]$
  3.  $[s_0, A, s_0] \rightarrow a[s_0, A, s_0][s_0, A, s_0]$   
 $[s_0, A, s_0] \rightarrow a[s_0, A, s_1][s_1, A, s_0]$   
 $[s_0, A, s_1] \rightarrow a[s_0, A, s_0][s_0, A, s_1]$   
 $[s_0, A, s_1] \rightarrow a[s_0, A, s_1][s_1, A, s_1]$
  4.  $[s_0, A, s_1] \rightarrow b$
  5.  $[s_1, A, s_1] \rightarrow b$

## Beispiel (Forts.)

Lesbarer haben wir damit folgende Grammatik:

$$\begin{aligned} S &\rightarrow A \mid B \\ A &\rightarrow aC \mid \varepsilon \\ B &\rightarrow aD \\ C &\rightarrow aCC \mid aDE \\ D &\rightarrow aCD \mid aDF \mid b \\ F &\rightarrow b \end{aligned}$$

Man sieht jetzt:

- Variable  $E$  ist nutzlos, und damit auch die Variable  $C$ .
- Die Grammatik enthält Kettenproduktionen und nullbare Variablen.

## Teil I

- 1 Ableitungsbäume
- 2 Umformung von Grammatiken
- 3 Normalformen
- 4 Pumping-Lemma für kontextfreie Sprachen
- 5 Pushdown-Automaten (PDAs)
- 6 Abschlusseigenschaften**
- 7 Wortprobleme
- 8 Der CYK-Algorithmus

## Beispiel (Forts.)

Nach Entfernung der überflüssigen Elemente:

$$\begin{aligned} S &\rightarrow \varepsilon \mid aD \\ D &\rightarrow aDF \mid b \\ F &\rightarrow b \end{aligned}$$

Mit dieser Grammatik kann man z.B. folgende Ableitung ausführen:

$$\begin{aligned} S &\Rightarrow aD \Rightarrow aaDF \Rightarrow aaaDFF \Rightarrow aaabFF \\ &\Rightarrow aaabbF \Rightarrow aaabbb \end{aligned}$$

## Abschlusseigenschaften

### Theorem 6.1 (Abschlusseigenschaften von $L_2$ )

$L_2$  ist abgeschlossen gegen:

- Vereinigung  $\cup$
- Konkatenation  $\circ$
- Kleene-Stern  $*$

### Beweis

Seien

$$G_i = (V_i, T_i, R_i, S_i) \quad (i \in \{1, 2\})$$

zwei cf-Grammatiken mit  $V_1 \cap V_2 = \emptyset$ .

Sei

$$L_i = L(G_i)$$

## Abschlusseigenschaften

### Beweis (Forts.)

zu  $\cup$ :

$$(V_1 \cup V_2 \cup \{S_{neu}\}, T_1 \cup T_2, R_1 \cup R_2 \cup \{S_{neu} \rightarrow S_1 \mid S_2\}, S_{neu})$$

erzeugt gerade  $L_1 \cup L_2$

zu  $\circ$ :

$$(V_1 \cup V_2 \cup \{S_{neu}\}, T_1 \cup T_2, R_1 \cup R_2 \cup \{S_{neu} \rightarrow S_1 S_2\}, S_{neu})$$

erzeugt gerade  $L_1 \circ L_2$

zu  $*$ :

$$(V_1 \cup \{S_{neu}\}, T_1, R_1 \cup \{S_{neu} \rightarrow S_1 S_{neu} \mid \varepsilon\}, S_{neu})$$

erzeugt gerade  $L_1^*$ . □

## Abschlusseigenschaften

### Theorem 6.2 (Abschlusseigenschaften von $L_2$ )

$L_2$  ist *nicht* abgeschlossen gegen:

- Durchschnitt  $\cap$
- Komplement  $\neg$

## Abschlusseigenschaften

### Beweis

Zu „ $\cap$ “:

$$L_1 = \{a^n b^n c^m \mid n, m \in \mathbb{N}\}$$

$$L_2 = \{a^m b^n c^n \mid n, m \in \mathbb{N}\}$$

wird erzeugt von  $G_i = (\{S, S', T\}, \{a, b, c\}, R_i, S)$  mit

$$R_1 = \{ \begin{array}{l} S \rightarrow S' T \\ S' \rightarrow a S' b \mid ab \\ T \rightarrow c T \mid c \end{array} \}$$

$$R_2 = \{ \begin{array}{l} S \rightarrow T S' \\ S' \rightarrow b S' c \mid bc \\ T \rightarrow a T \mid a \end{array} \}$$

Sowohl  $L_1$  als auch  $L_2$  sind cf, **nicht** aber  $L_1 \cap L_2 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ .

## Abschlusseigenschaften

### Beweis

Zu „ $\neg$ “:

Angenommen,  $L_2$  wäre abgeschlossen gegen  $\neg$ .

Wegen

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

wäre  $L_2$  dann auch abgeschlossen gegen  $\cap$  – **Widerspruch** □

## Teil I

- 1 Ableitungsbäume
- 2 Umformung von Grammatiken
- 3 Normalformen
- 4 Pumping-Lemma für kontextfreie Sprachen
- 5 Pushdown-Automaten (PDAs)
- 6 Abschlusseigenschaften
- 7 Wortprobleme**
- 8 Der CYK-Algorithmus

## Wortproblem

### Problem

**Gegeben:** eine cf-Grammatik  $G$ , so daß  $L(G)$  eine Sprache ist über  $\Sigma$ , und ein Wort  $w \in \Sigma^*$

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

## Wortproblem

### Lösung des Wortproblems für $L_3$

Gegeben eine rechtslineare Grammatik  $G$ , so daß  $L(G)$  eine Sprache ist über  $\Sigma$ , und ein Wort  $w \in \Sigma^*$ .

- Konstruiere aus  $G$  einen  $\varepsilon$ -NDEA  $A_1$ .
- Konstruiere aus  $A_1$  einen NDEA  $A_2$ .
- Konstruiere aus  $A_2$  einen DEA  $A_3$ .
- Probiere aus, ob  $A_3$  das Wort  $w$  akzeptiert.  
Dazu braucht der Automat  $A_3$  genau  $|w|$  Schritte.

## Wortproblem

### Das Wortproblem für $L_2$

- Zu jeder cf-Grammatik  $G$  kann man einen PDA konstruieren
- Aber ein Pushdown-Automat kann  $\varepsilon$ -Übergänge machen, in denen er das Wort nicht weiter liest.
- **Wie kann man dann garantieren, daß der Automat in endlich vielen Schritten das Wort  $w$  zu Ende gelesen hat?**
- Deshalb: verwende anderes Verfahren:  
**Cocke-Younger-Kasami-Algorithmus** (CYK-Algorithmus)  
Auch: Chart-Parsing

Gegeben: Ein Wort

$$w = a_1 \dots a_n$$

## Idee

- Prinzip der dynamischen Programmierung
- 1.: Ermittle woraus sich die einstelligen Teilworte ableiten lassen
- 2.: Ermittle woraus sich die zweistelligen Teilworte ableiten lassen
- ...
- $n$ .: Ermittle woraus sich die  $n$ -stelligen Teilworte ( $w$  selbst) ableiten lassen