Vorlesung

Grundlagen der Theoretischen Informatik / Einführung in die Theoretische Informatik I

Bernhard Beckert

Institut für Informatik

UNIVERSITÄT KOBLENZ · LANDAU

Sommersemester 2007

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

Inhalt von Teil IV

- Die von **Kellerautomaten** (Push-Down-Automaten, PDAs) erkannten Sprachen sind genau die vom Typ 2 (**kontextfrei**).
- **Normalformen** für kontextfreie Grammatiken.
- **Pumping-Lemma** für kontextfreie Sprachen.
- Effiziente Algorithmen für Probleme über PDAs

Teil IV

Kellerautomaten und kontextfreie Sprachen

Zur Erinnerung: kontextfreie Grammatiken

Kontextfreie Grammatiken

- Kontextfreie Regel:
  Eine Variable wird durch ein Wort ersetzt,
  (egal in welchem Kontext die Variable steht)
- Es wird eine **einzelne** Variable ersetzt.
- Das Wort in der Conclusio kann Variablen und Terminale in **beliebiger Mischung** enthalten.

Zur Erinnerung: kontextfreie Sprachen

Beispiel 1.1 (kontextfreie Sprachen)

- $\{a^n b^n \mid n \in \mathbb{N}_0\}$
- $\{a^n b a^n \mid n \in \mathbb{N}_0\}$
- $\{ww^R \mid w \in \{a,b\}^*\}$

Ableitungsbäume

Definition 1.2 (Ableitungsbaum zu einer Grammatik)

Sei

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

eine kontextfreie Grammatik.

Ein **Ableitungsbaum (parse tree)** zu G ist ein angeordneter Baum

$$B = (W, E, v_0)$$

Ableitungsbäume

Definition 1.3 (Ableitungsbaum zu einer Grammatik, Fortsetzung)

Zudem muss gelten:

- Jeder Knoten $v \in W$ ist mit einem Symbol aus $V \cup T \cup \{\varepsilon\}$ markiert.
- Die Wurzel v_0 ist mit S markiert.
- Jeder innere Knoten ist mit einer Variablen aus V markiert.
- Jedes Blatt ist mit einem Symbol aus $T \cup \{\varepsilon\}$ markiert.
- Ist $v \in W$ ein innerer Knoten mit Söhnen $v_1, \ldots, v_k$ in dieser Anordnung und ist A die Markierung von v und A_i die Markierung von v_i, dann ist $A \rightarrow A_1 \ldots A_k \in R$.
- Ein mit ε markiertes Blatt hat keinen Bruder
  (denn das entspräche einer Ableitung wie $A \rightarrow ab\varepsilon Bc$).

Ableitungsbäume

Ablesen eines Wortes vom Ableitungsbaum

Wenn Wort w von Grammatik G erzeugt wird,
dann gibt es einen Ableitungsbaum mit den Buchstaben von w
als Blätter von links nach rechts.

Merke

Die Blätter eines Ableitungsbaumes sind angeordnet.
Es gibt eine Ordnung unter den Söhnen eines Knotens.

Ableitungsbäume

Definition 1.4

Seien b_1, b_2 Blätter. Dann:

$$\begin{aligned} b_1 < b_2 \ \underline{\textit{gdw}}\ & b_1, b_2 \text{ sind Brüder, und } b_1 \text{ liegt "links" von } b_2, \\ & \text{oder } \exists v, v_1, v_2 \in W \quad v \to v_1,\ v \to v_2,\ v_1 < v_2 \\ & \text{und } v_i \text{ ist Vorfahre von } b_i \text{ für } i \in \{1,2\}. \end{aligned}$$

Definition 1.5

Sei $\{b_1, \dots, b_k\}$ die Menge aller Blätter in B mit $b_1 < \dots < b_k$, und sei A_i die Markierung von b_i.
Dann heißt das Wort $A_1 \dots A_k$ die **Front** von B.

Ableitungsbäume

Theorem 1.6

Sei $G = (V, T, R, S)$ eine kontextfreie Grammatik.
Dann gilt für $w \in T^*$:

$$(S \Longrightarrow_G^* w) \quad \underline{\textit{gdw}} \quad \text{Es existiert ein Ableitungsbaum zu } G \text{ mit Front } w.$$

Beweis.

Einfach aus den Definitionen. □

Ableitungsbäume: Beispiel

Beispiel 1.7

Grammatik für die Menge aller aussagenlogischen Formeln über den Variablen $\{x, x_0, x_1, x_2, \dots\}$:

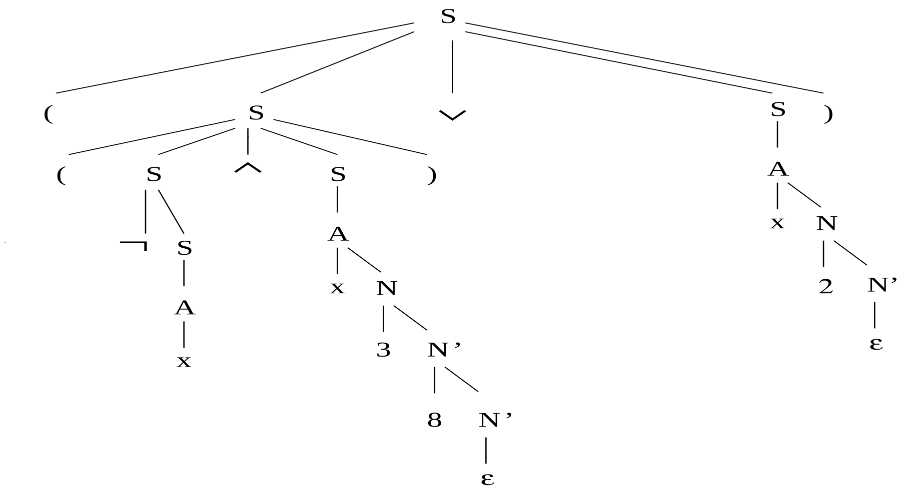
$$G = (\{S, A, N, N'\}, \{x, 0, \dots, 9, (,), \wedge, \vee, \neg\}, R, S)$$

mit der Regelmenge

$$\begin{aligned} R = \{ S &\to (S \wedge S) \mid (S \vee S) \mid \neg S \mid A \\ A &\to x \mid xN \\ N &\to 1N' \mid 2N' \mid \dots \mid 9N' \mid 0 \\ N' &\to 0N' \mid 1N' \mid \dots \mid 9N' \mid \varepsilon \} \end{aligned}$$

Ableitungsbäume: Beispiel

Ableitungsbaum für $((\neg x \land x38) \lor x2)$



Ableitungsbäume: Beispiel

Ableitung für $((\neg x \land x38) \lor x2)$

Der Ableitungsbaum steht für viele **äquivalente** Ableitungen, darunter diese:

$$\begin{array}{rcrc}
S & & (S \lor S) & \Rightarrow \\
((S \land S) \lor S) & \Rightarrow & ((\neg S \land S) \lor S) & \Rightarrow \\
((\neg A \land S) \lor S) & \Rightarrow & ((\neg x \land S) \lor S) & \Rightarrow \\
((\neg x \land A) \lor S) & \Rightarrow & ((\neg x \land xN) \lor S) & \Rightarrow \\
((\neg x \land x3N') \lor S) & \Rightarrow & ((\neg x \land x38N') \lor S) & \Rightarrow \\
((\neg x \land x38) \lor S) & \Rightarrow & ((\neg x \land x38) \lor A) & \Rightarrow \\
((\neg x \land x38) \lor xN) & \Rightarrow & ((\neg x \land x38) \lor x2N') & \Rightarrow \\
((\neg x \land x38) \lor x2) & & &
\end{array}$$

Links- und Rechtsableitung

Definition 1.8 (Linksableitung)

Eine Ableitung

$$w_1 \Longrightarrow_G w_2 \Longrightarrow_G \dots \Longrightarrow_G w_n$$

heißt **Linksableitung** falls w_{i+1} durch Ersetzen der linkesten Variable in w_i entsteht für alle $i < n$.

Die **Rechtsableitung** ist analog definiert.

Mehrdeutigkeit

Definition 1.9 (Mehrdeutigkeit)

Eine cf-Grammatik G heißt **mehrdeutig**

gdw

es gibt ein Wort $w \in L(G)$,
zu dem es in G **zwei verschiedene Linksableitungen** gibt.

Eine Sprache $L \in \mathbf{L}_2$ heißt **inhärent mehrdeutig**

gdw

alle kontextfreien Grammatiken für L sind mehrdeutig.

Bemerkung

Eine Grammatik G ist mehrdeutig, gdw :

es gibt zwei verschiedene Ableitungsbäume in G mit gleicher Front.

Mehrdeutigkeit: Beispiele

Beispiel 1.10 (Mehrdeutigkeit)

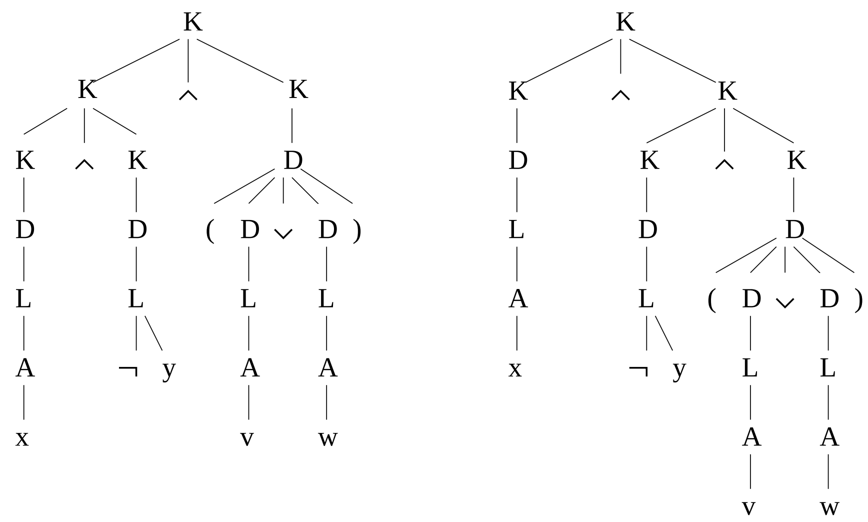
Eindeutige Grammatik für aussagenlogische Formeln:

$$\begin{aligned}
S &\to (S \wedge S) \mid (S \vee S) \mid \neg S \mid A \\
A &\to x \mid xN \\
N &\to 1N' \mid 2N' \mid \ldots \mid 9N' \mid 0 \\
N' &\to 0N' \mid 1N' \mid \ldots \mid 9N' \mid \varepsilon\}
\end{aligned}$$

Mehrdeutige Grammatik für aussagenlogische Formeln:

$$\begin{aligned}
K &\to K \wedge K \mid D \qquad \textbf{Regel mit Klammer-Ersparnis!} \\
D &\to (D \vee D) \mid L \\
L &\to \neg A \mid A \\
A &\to v \mid w \mid x \mid y \mid z
\end{aligned}$$

Mehrdeutigkeit: Beispiele



Mehrdeutigkeit: Beispiele

Beispiel 1.11 (Inhärente Mehrdeutigkeit)

Die Sprache

$$L := \{a^i b^j c^k \mid i = j \text{ oder } j = k\}$$

ist **inhärent mehrdeutig**.

Teil IV

Kellerautomaten und kontextfreie Sprachen

Startsymbol nur links

Einfache Annahme

Im folgenden soll für alle cf-Grammatiken gelten:

Das Startsymbol S kommt nie auf einer rechten Regelseite vor.

Umformung

Ist das bei einer Grammatik nicht gegeben, kann man es wie folgt erreichen:

- Führe ein neues Startsymbol S_{neu} ein
- Füge die Regel

  $$S_{neu} \to S$$

  hinzu.

Nutzlose Symbole

Nutzlose Symbole und Regeln: Intuition

- Variablen und Symbole, die vom Startsymbol aus unerreichbar sind.
- Variablen, von denen aus kein Terminalwort abgeleitet werden kann.
- Regeln, die solche Variablen und Symbole enthalten

Nutzlose Symbole

Definition 2.1 ((co-)erreichbare, nutzlose Symbole)

Sei $G = (V, T, R, S)$ eine Grammatik.

Ein Symbol $x \in (V \cup T)$ heißt

erreichbar: Es gibt $\alpha, \beta \in (V \cup T)^*$: $S \Longrightarrow_G^* \alpha x \beta$

co-erreichbar: Es gibt $w \in T^*$: $x \Longrightarrow_G^* w$

nutzlos: x ist nicht erreichbar oder nicht co-erreichbar.

Nutzlose Symbole

Theorem 2.2 (cf-Grammatik ohne nutzlose Symbole)

Ist $G = (V, T, R, S)$ eine cf-Grammatik mit $L(G) \neq \emptyset$,
dann existiert eine cf-Grammatik $G' = (V', T', R', S')$ mit:

- *G' ist äquivalent zu G.*
- *Jedes $x \in (V \cup T)$ ist erreichbar und co-erreichbar.*

Beweis

Man kann G' aus G effektiv konstruieren:

- Wie im folgenden beschrieben, die nutzlosen Symbole bestimmen.
- Diese Symbole und alle Regeln, die sie enthalten, entfernen.

Nutzlose Symbole

Algorithmus zur Berechnung der co-erreichbaren Variablen

Input: Grammatik $G = (V, T, R, S)$
Output: co-erreichbare Variablen

$\mathrm{Alt} := \emptyset$
$\mathrm{Neu} := \{A \in V \mid \exists w \in T^* \ (A \to w \in R)\}$

while $\mathrm{Alt} \neq \mathrm{Neu}$
{
  $\mathrm{Alt} := \mathrm{Neu}$
  $\mathrm{Neu} := \mathrm{Alt} \cup \{A \in V \mid \exists \alpha \in (T \cup \mathrm{Alt})^* \ (A \to \alpha \in R)\}$
}

output Neu

Nutzlose Symbole

Algorithmus zur Berechnung der erreichbaren Symbole

Input: Grammatik $G = (V, T, R, S)$
Output: erreichbare Symbole

$\mathrm{Alt} := \emptyset$
$\mathrm{Neu} := \{S\}$
while $\mathrm{Alt} \neq \mathrm{Neu}$
{
  $\mathrm{Alt} := \mathrm{Neu}$
  $\mathrm{Neu} := \mathrm{Alt} \cup \{x \in (V'' \cup T'') \mid \exists A \in \mathrm{Alt}$
  $\exists \alpha, \beta \in (V'' \cup T'')^*$
  $(A \to \alpha x \beta \in R)\}$
}

output Neu

Normalform für Regeln

Theorem 2.3 (Normalform)

Zu jeder Grammatik G (beliebigen Typs)
existiert eine äquivalente Grammatik G',
bei der für alle Regeln $P \to Q \in R'$ gilt:

- *$Q \in V^*$ und P beliebig*
- *$Q \in T$ und $P \in V$*

Für alle Typen außer den linearen hat G' denselben Typ wie G.

Normalform für Regeln

Beweis.

Für jedes Terminal $t \in T$ erzeuge man eine neue Variable V_t.

- $V' = V \cup \{V_t \mid t \in T\}$
- R' entsteht aus R, indem für jede Regel $P \to Q \in R$ in Q alle Vorkommen eines Terminals t durch die zugehörige Variable V_t ersetzt werden. Außerdem enthält R' für jedes $t \in T$ eine neue Regel $V_t \to t$.

Also $L(G') = L(G)$,
und für alle Sprachklassen außer $\mathbf{L}_3$ hat G' denselben Typ wie G. □

Elimination von ε-Regeln

Idee

Variablen, aus denen ε ableitbar ist, sollten eliminiert werden

Definition 2.4 (ε-Regel, nullbare Variablen)

Eine Regel der Form

$$P \to \varepsilon \qquad (P \text{ eine Variable})$$

heißt **ε-Regel**.

Eine Variable A heißt **nullbar**,
falls

$$A \Longrightarrow^* \varepsilon$$

Elimination von ε-Regeln

Theorem 2.5 (ε-Regeln sind eliminierbar)

Zu jeder cf-Grammatik G existiert eine äquivalente cf-Grammatik G'

- *ohne ε-Regeln und nullbare Variablen,*
  falls $\varepsilon \notin L(G)$,
- *mit der einzigen ε-Regel $S \to \varepsilon$ und der einzigen nullbaren Variablen S,*
  falls $\varepsilon \in L(G)$ und S das Startsymbol ist.

Elimination von ε-Regeln

Algorithmus zur Berechnung der nullbaren Variablen

Input: Grammatik $G = (V, T, R, S)$ $\qquad$ S o.B.d.A. in keiner Regel rechts
Output: nullbare Variablen

$Alt := \emptyset$
$Neu := \{A \in V \mid A \to \varepsilon \in R\}$
while $Alt \neq Neu$
$\{$ $Alt := Neu$
  für alle $(P \to Q) \in R$ **do**
  $\{$ **if** $Q = A_1 \dots A_n$ **and** $A_i \in Neu$ für $1 \leq i \leq n$ **and** $P \notin Neu$,
    then $Neu := Neu \cup \{P\}$
  $\}$
$\}$

output Neu

Elimination von ε-Regeln

Beweis (Forts.)

Ausgangsgrammatik G habe die Normalform, bei der für jede Regel $P \to Q$: $Q \in V^*$ oder $Q \in T$.

Für jede Regel $P \to A_1 \dots A_n$ generiere alle möglichen Kombinationen

$$P \to \alpha_1 \dots \alpha_n$$

mit

- $\alpha_i \in \{\varepsilon, A_i\}$ falls A_i nullbar
- $\alpha_i = A_i$ falls A_i nicht nullbar

Dann

- Füge alle diese neuen Regeln zur Grammatik hinzu
- Entferne alle Regeln der Form $A \to \varepsilon$ mit $A \neq S$

Elimination von ε-Regeln

Beweis (Forts.)

Zu zeigen:
Für die neue Grammatik G' gilt: $L(G') = L(G)$

Vorgehen:

- G hat die Normalform:
  Für jede Regel $P \to Q$ gilt $Q \in V^*$ oder $Q \in T$.
- Wir beweisen die etwas stärkere Behauptung

  $$\text{für alle } A \in V \ \text{ für alle } w \in (V \cup T)^* - \{\varepsilon\}$$
  $$\left((A \Longrightarrow_G^* w) \ \underline{\text{gdw}} \ (A \Longrightarrow_{G'}^* w)\right),$$
- Daraus folgt sofort $L(G') = L(G)$.

Elimination von ε-Regeln

Beweis (Forts.)

"⇒" Wir zeigen: Aus $A \Longrightarrow_G^* w$ folgt $A \Longrightarrow_{G'}^* w$ (Induktion über Länge einer Ableitung von A nach w in G).

Induktionsanfang: Länge = 0.
Dann ist $w = A$, und $A \Longrightarrow_{G'}^* A$ gilt immer.

Induktionsschritt: Es sei schon gezeigt: Wenn in G in n Schritten eine Ableitung $B \Longrightarrow_G^* u$ durchgeführt werden kann, dann folgt, daß in G' die Ableitung $B \Longrightarrow_{G'}^* u$ möglich ist.

Elimination von ε-Regeln

Beweis (Forts.)

Außerdem gelte in der Ausgangsgrammatik G: $A \Longrightarrow_G^* w \neq \varepsilon$ in $n+1$ Schritten.
Dann gilt:

- $A \Longrightarrow_G w' \Longrightarrow_G^* w$,
- $w' = A_1 \dots A_\ell \Longrightarrow_G^* w_1 \dots w_\ell = w$,
- und es wird jeweils A_i zu w_i in höchstens n Schritten für geeignete $w', A_1, \dots, A_\ell, w_1, \dots, w_\ell$.
- Per Induktionsvoraussetzung gilt also schon:
  - Entweder $A_i \Longrightarrow_{G'}^* w_i$
  - oder $w_i = \varepsilon$ für $1 \le i \le \ell$.

Elimination von ε-Regeln

Beweis (Forts.)

Fall 1: $w_i = \varepsilon$, A_i ist nullbar.
Dann gibt es in G' eine Regel $A \to A_1 \dots A_{i-1} A_{i+1} \dots A_\ell$ nach der obigen Konstruktionsvorschrift für G', falls $A_1 \dots A_{i-1} A_{i+1} \dots A_\ell \neq \varepsilon$. Das ist der Fall, denn sonst hätten wir: $A \Longrightarrow w' = \varepsilon \Longrightarrow^* w = \varepsilon$ (aus nichts wird nichts), aber $w = \varepsilon$ ist ausgeschlossen.

Fall 2: $w_i \neq \varepsilon$. Dann gilt nach Induktionsvoraussetzung $A_i \Longrightarrow_{G'}^* w_i$.

Elimination von ε-Regeln

Beweis (Forts.)

Wir haben also folgendes gezeigt:

Sei $I = \{i \in \{1 \dots \ell\} \mid w_i \neq \varepsilon\} \neq \emptyset$.

Dann gibt es in R' eine Regel $A \to A_{i_1} \dots A_{i_m}$ mit $I = \{i_1, \dots, i_m\}$, und die A_i sind so angeordnet wie in der ursprünglichen Regel $A \to A_1 \dots A_\ell$.

Mit dieser neuen Regel können wir w so ableiten:

$$A \Longrightarrow_{G'} A_{i_1} \dots A_{i_m} \Longrightarrow^*_{G'} w_{i_1} \dots w_{i_m} = w$$

Elimination von ε-Regeln

Beweis (Forts.)

"$\Leftarrow$" Wir zeigen: Aus $A \Longrightarrow^*_{G'} w$ folgt $A \Longrightarrow^*_{G} w$ (Induktion über Länge einer Ableitung von A nach w in G'):

Induktionsanfang: Länge = 0. Dann ist $w = A$, und $A \Longrightarrow^*_G A$ gilt immer.

Induktionsschritt: Es gelte für alle Ableitungen $A \Longrightarrow^*_{G'} w$ einer Länge von höchstens n, daß $A \Longrightarrow^*_G w$.

Ist $A \Longrightarrow^*_{G'} w$ eine Ableitung der Länge $n+1$, so gibt es ein ℓ, Wörter $w_1, \dots, w_\ell$ und Variablen $A_1, \dots, A_\ell$ mit $A \Longrightarrow_{G'} A_1 \dots A_\ell \Longrightarrow^*_{G'} w = w_1 \dots w_\ell$. Es gilt jeweils $A_i \Longrightarrow^*_{G'} w_i$ in höchstens n Schritten, und $w_i \neq \varepsilon$.

Elimination von ε-Regeln

Beweis (Forts.)

Nach der Induktionsvoraussetzung folgt daraus:

- für die Originalgrammatik G gibt es Ableitungen $A_i \Longrightarrow^*_G w_i$
- damit gibt es auch eine Ableitung $A_1 \dots A_\ell \Longrightarrow^*_G w$.

Da es in G' eine Ableitung $A \Longrightarrow_{G'} A_1 \dots A_\ell$ gibt, gibt es in R' eine Regel $A \to A_1 \dots A_\ell$. Wie ist diese Regel aus R entstanden?
Eine Regel in R' entsteht aus einer Regel in R, indem einige nullbare Variablen gestrichen werden. Es gab also in G nullbare Variablen B_1 bis B_m, so daß R die Regel

$$A \to A_1 \dots A_{\ell_1} B_1 A_{\ell_1+1} \dots A_{\ell_2} B_2 \dots A_m B_m A_{m+1} \dots A_\ell$$

enthält. (m kann auch 0 sein, dann war die Regel selbst schon in R.)

Elimination von ε-Regeln

Beweis (Forts.)

Also gilt in G:

$$\begin{aligned} A &\Longrightarrow_G A_1 \dots A_{\ell_1} B_1 A_{\ell_1+1} \dots A_{\ell_2} B_2 \dots A_m B_m A_{m+1} \dots A_\ell \\ &\Longrightarrow^*_G A_1 \dots A_{\ell_1} A_{\ell_1+1} \dots A_{\ell_2} \dots A_m A_{m+1} \dots A_\ell \Longrightarrow^*_G w \end{aligned}$$

da ja $B_i \Longrightarrow^*_G \varepsilon$ möglich ist. □

Elimination von ε-Regeln: Beispiel

Beispiel 2.6

R:	R':
$S \longrightarrow ABD$	$S \longrightarrow ABD \mid AD \mid BD \mid D$
$A \longrightarrow ED \mid BB$	$A \longrightarrow ED \mid BB \mid B$
$B \longrightarrow AC \mid \varepsilon$	$B \longrightarrow AC \mid A \mid C$
$C \longrightarrow \varepsilon$	
$D \longrightarrow d$	$D \longrightarrow d$
$E \longrightarrow e$	$E \longrightarrow e$

Für die Regelmenge R in der linken Spalte sind die Variablen A, B, C nullbar.

Der obige Algorithmus erzeugt aus R die rechts aufgeführte Regelmenge R'.

Elimination von ε-Regeln

Beobachtung

- Der Algorithmus lässt nutzlose Variablen zurück, die nicht in Prämissen auftauchen (und deshalb nicht co-erreichbar sind). Hier: C.
- Der Algorithmus lässt nutzlose Regeln zurück. Hier: $B \longrightarrow AC \mid C$.

Elimination von ε-Regeln

Korollar

$$\mathbf{L}_2 \subseteq \mathbf{L}_1$$

Das heißt, jede kontextfreie Sprache ist auch kontextsensitiv

Beweis

Regeln einer kontextsensitiven Grammatik müssen folgende Form haben:

- entweder $uAv \longrightarrow u\alpha v$
  mit $u, v, \alpha \in (V \cup T)^*, |\alpha| \geq 1, A \in V$
- oder $S \longrightarrow \varepsilon$
  und S kommt in keiner Regelconclusio vor.

Diesen Bedingungen genügt die kontextfreie Grammatik nach Elimination der ε-Regeln.

Elimination von Kettenproduktionen

Definition 2.7 (Kettenproduktion)

Eine Regel der Form

$$A \longrightarrow B \qquad \text{mit } A, B \in V$$

heißt **Kettenproduktion**.

Theorem 2.8 (Kettenproduktionen sind eliminierbar)

Zu jeder cf-Grammatik existiert eine äquivalente cf-Grammatik ohne Kettenproduktionen.

Elimination von Kettenproduktionen

Beweis

Sei $G = (V, T, R, S)$ eine kontextfreie Grammatik ohne ε-Regeln, außer ggf. $S \to \varepsilon$.

Konstruiere neue Grammatik wie folgt:

1. Für alle
   - Variablenpaare $A, B \in V, \quad A \neq B \quad$ mit $A \Longrightarrow^* B$
   - Regeln $B \to \alpha \in R, \quad \alpha \notin V$

   füge zu R hinzu:

   $$A \to \alpha$$

2. Lösche alle Kettenproduktionen

Normalform für cf-Grammatiken

Theorem 2.9 (Normalform für cf-Grammatiken)

Zu jeder cf-Grammatik existiert eine äquivalente cf-Grammatik

- *ohne ε-Regeln*
  (bis auf $S \to \varepsilon$, falls ε zur Sprache gehört;
  in diesem Fall darf S in keiner Regelconclusio vorkommen),
- *ohne nutzlose Symbole,*
- *ohne Kettenproduktionen,*
- *so daß für jede Regel $P \to Q$ gilt: entweder $Q \in V^*$ oder $Q \in T$.*

Normalform für cf-Grammatiken

Beweis

1. Man teste zunächst, ob S nullbar ist. Falls ja, dann verwende man S_{neu} als neues Startsymbol und füge die Regeln $S_{neu} \to S \mid \varepsilon$ zum Regelsatz hinzu.
2. Man eliminiere nutzlose Symbole.
3. Man eliminiere alle ε-Regeln außer $S_{neu} \to \varepsilon$.
4. Man bringe die Grammatik in die Normalform, bei der für jede Regel $P \to Q$ gilt: entweder $Q \in V^*$ oder $Q \in T$.
5. Man eliminiere Kettenproduktionen.
6. Zum Schluss eliminiere man noch einmal alle nutzlosen Symbole (wg. Schritt 3)

Teil IV

Kellerautomaten und kontextfreie Sprachen

Normalformen

Unterschied: Grammatiktypen und Normalformen

Gemeinsamkeit: Sowohl Grammatiktypen als auch Normalformen schränken die Form von Grammatikregeln ein.

Unterschied:

- Grammatiktypen (rechtslinear, kontextfrei usw.) führen zu **unterschiedlichen Sprachklassen**
- Normalformeln führen zu **den selben Sprachklassen**

Normalformen

Wozu dann Normalformen?

- Weniger Fallunterscheidungen bei Algorithmen, die mit Grammatiken arbeiten.
- Struktur von Grammatiken einfacher zu „durchschauen"

Zwei Normalformen

Chomsky-Normalform: Baut auf den Umformungen des vorigen Teils auf.

Greibach-Normalform: Ähnlich den rechtslinearen Grammatiken.

Chomsky-Normalform

Definition 3.1 (Chomsky-Normalform)

Eine cf-Grammatik $G = (V, T, R, S)$ ist in **Chomsky-Normalform (CNF)**, wenn gilt:

- G hat nur Regeln der Form

$$\begin{array}{lll} A \to BC & \text{mit } A, B, C \in V \text{ und} & \\ A \to a & \text{mit } A \in V,\ a \in T & (\text{nicht } \varepsilon!) \end{array}$$

- Ist $\varepsilon \in L(G)$, so darf G zusätzlich die Regel $S \to \varepsilon$ enthalten. In diesem Fall darf S in keiner Regelconclusio vorkommen.
- G enthält keine nutzlosen Symbole.

Chomsky-Normalform

Theorem 3.2 (Chomsky-Normalform)

Zu jeder cf-Grammatik existiert eine äquivalente cf-Grammatik in Chomsky-Normalform.

Beweis

Schritt 1: Wende auf G die Umformungen des letzten Abschnitts an.

Ergebnis:

- G hat keine nutzlosen Symbole
- Alle Regeln haben die Form
  1. $A \to \alpha$ mit $A \in V$ und $\alpha \in V^*$, $|\alpha| \geq 2$, und
  2. $A \to a$ mit $A \in V$, $a \in T$

Chomsky-Normalform

Beweis (Forts.)

Schritt 2: Regeln so umformen, daß keine Conclusio eine Länge größer 2 hat.

Ersetze jede Regel

$$A \to A_1 \ldots A_n \text{ mit } A, A_i \in V, n \geq 3$$

durch:

$$\begin{array}{ll} A & \to A_1 C_1 \\ C_1 & \to A_2 C_2 \\ & \vdots \\ C_{n-2} & \to A_{n-1} A_n \end{array}$$

Dabei sind die C_i neue Variablen in V.

□

Greibach-Normalform

Definition 3.3 (Greibach-Normalform)

Eine cf-Grammatik $G = (V, T, R, S)$ ist in **Greibach-Normalform (GNF)**, wenn gilt:

- G hat nur Regeln der Form
  $$A \to a\alpha \textit{ mit } A \in V \textit{ und } a \in T \textit{ und } \alpha \in V^*$$
- Ist $\varepsilon \in L(G)$, so darf G zusätzlich die Regel $S \to \varepsilon$ enthalten. In diesem Fall darf S in keiner Regelconclusio vorkommen.
- G enthält keine nutzlosen Symbole.

Teil IV

Kellerautomaten und kontextfreie Sprachen

Wiederholung: Pumping-Lemma für reguläre Sprachen

Theorem 4.1 (Pumping-Lemma für L_3-Sprachen)

*Sei $L \in$ **RAT**.*
Dann existiert ein $n \in \mathbb{N}$, so dass:

Für alle

$$x \in L \quad \textit{mit} \quad |x| \geq n$$

existiert eine Zerlegung

$$x = uvw \qquad u, v, w \in \Sigma^*$$

mit

- $|v| \geq 1$
- $|v| < n$
- $uv^m w \in L \quad \textit{für alle } m \in \mathbb{N}$

Pumping-Lemma für kontextfreie Sprachen

Theorem 4.2 (Pumping-Lemma für kontextfreie Sprachen)

Sei L kontextfrei
Dann existiert ein $n \in \mathbb{N}$, so dass:

Für alle

$$z \in L \quad \text{mit} \quad |x| \geq n$$

existiert eine Zerlegung

$$z = uvwxy \qquad u, v, w, x, y \in \Sigma^*$$

mit

- $|vx| \geq 1$
- $|vwx| < n$
- $uv^m wx^m y \in L$ *für alle* $m \in \mathbb{N}$

Pumping-Lemma für kontextfreie Sprachen

Beweisidee

Bei der Ableitung eines hinreichend langen Wortes muss es eine Variable geben, die mehr als einmal auftaucht.

Dies führt zu einer Schleife in der Ableitung, die aufgepumpt werden kann.

Pumping-Lemma für kontextfreie Sprachen

Anwendung des Pumping-Lemmas für cf-Sprachen

Wenn das cf-Pumping-Lemma für eine Sprache nicht gilt,
dann kann sie nicht kontextfrei sein.

Beispiel 4.3 (Sprachen, die nicht kontextfrei sind)

Für folgende Sprachen kann man mit Hilfe des cf-Pumping-Lemmas zeigen, dass sie nicht kontextfrei sind:

- $\{a^p \mid p \text{ prim}\}$
- $\{a^n b^n c^n \mid n \in \mathbb{N}\}$
- $\{zzz \mid z \in \{a, b\}^*\}$d.

Teil IV

Kellerautomaten und kontextfreie Sprachen

Erzeugende Grammatiken – akzeptierende Automaten

Erinnerung: Reguläre Sprachen

- werden erzeugt von rechtslinearen Grammatiken
- werden akzeptiert von endlichen Automaten

Jetzt: Kontextfreie Sprachen

- werden erzeugt von kontextfreien Grammatiken
- werden akzeptiert von **Pushdown-Automaten**

Idee des Push-Down-Automaten

Beispiel 5.1

Die „prototypische“ cf-Sprache

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

- Endliche Automaten reichen nicht aus.
- Sie können sich nicht merken, wie oft sie einen Zustand durchlaufen haben.
- Für $a^n b^n$ muss man aber **mitzählen**.

Idee des Push-Down-Automaten

Idee: Wie kann man diese Sprache akzeptieren?

- Weitere Informationen auf dem Stack sichern
- Später zurückholen
- Ähnlich einem „Prozeduraufruf“
- Grammatikregel wie $S \to aAb$ entspricht Aufruf einer Prozedur für das A.

Stack, Stapel, Keller

- Last in, first out
- Zuletzt gespeicherte Information liegt immer „obenauf“
- Beliebig viel Information kann gespeichert werden
  (Aber kein beliebiger Zugriff!)

Push-Down-Automat

Push-Down-Automat (PDA): Informell

- Wie endlicher Automat, aber **zusätzlicher** einen Stack
- Übergangsrelation bezieht das oberste Stacksymbol in den Übergang ein
- Bei Zustandsübergang: lesen und schreiben auf Stack

Push-Down-Automat

Definition 5.2 (Push-Down-Automat)

Ein **Push-Down-Automat (PDA)** ist ein Tupel

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

Dabei ist

K	eine endliche Menge von Zuständen
Σ	das Eingabealphabet
Γ	das Stack- oder Kelleralphabet
$s_0 \in K$	der Startzustand
$Z_0 \in \Gamma$	das Anfangssymbol im Keller
$F \subseteq K$	eine Menge von finalen Zuständen
Δ	die Zustandsübergangsrelation, eine endliche Relation:

$$\Delta \subset \quad (K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma) \times (K \times \Gamma^*)$$

Push-Down-Automat

Arbeitsschritt eines PDA

In Abhängigkeit

- vom aktuellen Zustand
- vom nächsten Eingabezeichen (oder auch unabhängig davon)
- vom obersten Kellersymbol

geschieht folgendes

- nächstes **Eingabezeichen** wird **gelesen oder nicht** (bei ε),
- das oberste **Kellersymbol** wird **entfernt**,
- der **Zustand** wird **geändert**,
- es werden null oder mehr **Zeichen auf den Keller** geschoben
  Bei neuen Keller-Wort $\gamma = A_1 \dots A_n$ wird A_n zuerst auf den Keller geschoben usw., so daß am Schluss A_1 obenauf liegt.

Push-Down-Automat

Notation

- a, b, c für Buchstaben aus Σ
- u, v, w für Wörter aus Σ^*
- A, B für Stacksymbole aus Γ
- γ, η für Stackinhalte aus Γ^*

Push-Down-Automat: Konfiguration

Konfiguration eines PDA: Informell

- Konfiguration beschreibt die aktuelle Situation des PDA **komplett**
- Bestandteile:
  - **aktueller Zustand**
  - **noch zu lesendes Restwort**
  - **kompletter Stackinhalt**
- Für Konfigurationen C_1, C_2 bedeutet

$$C_1 \vdash C_2$$

daß der PDA in einem Schritt von C_1 nach C_2 gelangen kann.

Push-Down-Automat: Konfiguration

Definition 5.3 (Konfiguration eines PDA, $\vdash$)

Eine **Konfiguration** C eines PDA $\mathcal{M} = (K, \Sigma, \Gamma, \Delta, s_0, Z_0, F)$ ist ein Tripel

$$(q, w, \gamma) \in K \times \Sigma^* \times \Gamma^*.$$

- q der aktuelle Zustand
- w der noch zu lesendes Restwort
- γ der komplette Stackinhalt

Definition 5.4 (Startkonfiguration)

Bei Eingabewort w ist die **Startkonfiguration**:

$$(s_0, w, Z_0)$$

Push-Down-Automat: Konfiguration

Definition 5.5 (Nachfolgekonfiguration)

C_2 heißt **Nachfolgekonfiguration** von C_1,

$$C_1 \vdash C_2$$

falls

$$\exists a \in \Sigma \ \exists A \in \Gamma \ \exists w \in \Sigma^* \ \exists \gamma, \eta \in \Gamma^*$$

so dass

entweder $C_1 = (q_1, aw, A\gamma)$, $C_2 = (q_2, w, \eta\gamma)$, und $(q_1, a, A) \ \Delta \ (q_2, \eta)$,

oder $C_1 = (q_1, w, A\gamma)$, $C_2 = (q_2, w, \eta\gamma)$, und $(q_1, \varepsilon, A) \ \Delta \ (q_2, \eta)$,

Push-Down-Automat: Rechnung

Definition 5.6 (Rechnung eines PDA)

Sei $\mathcal{A}$ ein Push-Down-Automat.

$$C \vdash_A^* C'$$

gdw es eine Reihe von Konfigurationen

$$C_0, C_1, \ldots, C_n \qquad (n \geq 0)$$

so daß

- $C = C_0$,
- $C' = C_n$,
- $C_i \vdash_A C_{i+1}$ für alle $0 \leq i < n$

Dann heißt $C_0, C_1, \ldots, C_n$ eine **Rechnung** von A

Push-Down-Automat: Akzeptierte Sprache

Definition 5.7 (von PDA akzeptierte Sprache)

Ein PDA $\mathcal{M}$ kann auf zwei verschiedene Arten eine Sprache akzeptieren:

- über **finale Zustände**
- über **leeren Keller**

$$L_f(\mathcal{M}) = \{w \in \Sigma^* \mid \exists q \in F \ \exists \gamma \in \Gamma^* \ \big((s_0, w, Z_0) \vdash_{\mathcal{M}}^* (q, \varepsilon, \gamma)\big)\}$$

$$L_l(\mathcal{M}) = \{w \in \Sigma^* \mid \exists q \in K \ \big((s_0, w, Z_0) \vdash_{\mathcal{M}}^* (q, \varepsilon, \varepsilon)\big)\}$$

Bemerkung

Das zu akzeptierende Wort w muss von $\mathcal{M}$ ganz gelesen werden:

$$(s_0, w, Z_0) \vdash^* (q, \varepsilon, \cdot)$$

ist gefordert.

Push-Down-Automat

Bemerkung

- Das unterste Symbol im Keller kann gelöscht werden.
- Dann aber **hängt** der PDA
- Er kann nicht mehr weiter rechnen
- **Es gibt keineNachfolgekonfiguration**

Push-Down-Automat: Beispiel

Beispiel 5.8

Sprache der Palindrome über $\{a, b\}$:

$$L = \{w \in \{a,b\}^* \mid w = w^R\}$$

L wird über leeren Keller akzeptiert von dem PDA

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

mit ...

Push-Down-Automat: Beispiel

Beispiel (Forts.)

Idee:

- Ein Palindrom $w = w^R$ hat die Form

  vv^R oder vav^R oder vbv^R

  für ein $v \in \{a, b\}^*$
- Der Automat $\mathcal{M}$ liest v und merkt sich jeden Buchstaben.
- **Er rät indeterminiert die Wortmitte.**
  Falls das Wort eine ungerade Anzahl von Buchstaben hat,
  also $w = vav^R$ oder $w = vbv^R$,
  dann muss dabei ein Buchstabe überlesen werden.
- Der Stack enthält nun v^R.
  $\mathcal{M}$ muss jetzt nur noch jeden weiteren gelesenen Buchstaben mit dem jeweils obersten Kellersymbol vergleichen.

Push-Down-Automat: Beispiel

Beispiel (Forts.)

$$(s_0, \varepsilon, Z_0)\ \Delta\ (s_1, \varepsilon) \quad \Big\}\ \varepsilon \text{ akzeptieren}$$

$$\left.\begin{array}{ll}
(s_0, a, Z_0) & \Delta\ (s_0, A) \\
(s_0, a, A) & \Delta\ (s_0, AA) \\
(s_0, a, B) & \Delta\ (s_0, AB) \\
(s_0, b, Z_0) & \Delta\ (s_0, B) \\
(s_0, b, A) & \Delta\ (s_0, BA) \\
(s_0, b, B) & \Delta\ (s_0, BB)
\end{array}\right\}\ \text{Stack aufbauen}$$

Push-Down-Automat: Beispiel

Beispiel (Forts.)

$$\left.\begin{array}{ll}(s_0, \varepsilon, A) & \Delta\,(s_1, \varepsilon) \\ (s_0, \varepsilon, B) & \Delta\,(s_1, \varepsilon)\end{array}\right\}\ \text{Richtungswechsel für Palindrome mit ungerader Buchstabenanzahl}$$

$$\left.\begin{array}{ll}(s_0, a, A) & \Delta\,(s_1, \varepsilon) \\ (s_0, b, B) & \Delta\,(s_1, \varepsilon)\end{array}\right\}\ \text{Richtungswechsel für Palindrome mit gerader Buchstabenanzahl}$$

$$\left.\begin{array}{ll}(s_1, a, A) & \Delta\,(s_1, \varepsilon) \\ (s_1, b, B) & \Delta\,(s_1, \varepsilon)\end{array}\right\}\ \text{Stack abbauen}$$

Push-Down-Automat: Beispiel

Beispiel (Forts.)

Für das Eingabewort *abbabba* rechnet $\mathcal{M}$ so:

$$\begin{array}{ccccc}(s_0, abbabba, Z_0) \vdash & (s_0, bbabba, A) & \vdash & (s_0, babba, BA) & \vdash \\ (s_0, abba, BBA) \vdash & (s_0, bba, ABBA) & \vdash & (s_1, bba, BBA) & \vdash \\ (s_1, ba, BA) \vdash & (s_1, a, A) & \vdash & (s_1, \varepsilon, \varepsilon) & \end{array}$$

Push-Down-Automat: Beispiel

Beispiel 5.9

Die Sprache

$$L = \{w \in \{a, b\}^* \mid \#_a(w) = \#_b(w)\}$$

wird über finalen Zustand akzeptiert von dem PDA

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

mit ...

Push-Down-Automat: Beispiel

Beispiel (Forts.)

Idee:

- auf dem Stack mitzählen, wieviel A-Überhang oder B-Überhang momentan besteht
- Der Stack enthält zu jedem Zeitpunkt
  - entweder nur $A/\underline{A}$ (A-Überhang)
  - oder nur $B/\underline{B}$ (B-Überhang)
  - oder nur das Symbol Z_0 (Gleichstand).
- Das unterste A bzw. B auf dem Stack ist durch einen Unterstrich gekennzeichnet.
  So weiß $\mathcal{M}$, wenn er dies Stacksymbol löscht, daß dann bis zu diesem Moment gleichviel as wie bs gelesen wurden.

Push-Down-Automat: Beispiel

Beispiel (Forts.)

$$\begin{array}{ll}
(s_0, a, Z_0)\ \Delta\ (s_1, \underline{A}) & (s_0, b, Z_0)\ \Delta\ (s_1, \underline{B}) \\
(s_1, a, \underline{A})\ \Delta\ (s_1, A\underline{A}) & (s_1, b, \underline{B})\ \Delta\ (s_1, B\underline{B}) \\
(s_1, a, A)\ \Delta\ (s_1, AA) & (s_1, b, B)\ \Delta\ (s_1, BB) \\
(s_1, a, \underline{B})\ \Delta\ (s_0, Z_0) & (s_1, b, \underline{A})\ \Delta\ (s_0, Z_0) \\
(s_1, a, B)\ \Delta\ (s_1, \varepsilon) & (s_1, b, A)\ \Delta\ (s_1, \varepsilon)
\end{array}$$

Push-Down-Automat: Finaler Zustand / leerer Keller

Theorem 5.10 (finale Zustände $\longrightarrow$ leerer Keller)

Zu jedem PDA $\mathcal{M}_1$ existiert ein PDA $\mathcal{M}_2$ mit

$$L_f(\mathcal{M}_1) = L_l(\mathcal{M}_2)$$

Beweisidee

- Wir simulieren die Maschine $\mathcal{M}_1$, die über finale Zustände akzeptiert, durch die Maschine $\mathcal{M}_2$, die über leeren Keller akzeptiert.
- $\mathcal{M}_2$ arbeitet wie $\mathcal{M}_1$, mit dem Unterschied:
  Wenn ein Zustand erreicht wird, der in $\mathcal{M}_1$ final war, kann $\mathcal{M}_2$ seinen Keller leeren.

Push-Down-Automat: Finaler Zustand / leerer Keller

Theorem 5.11 (leerer Keller $\longrightarrow$ finale Zustände)

Zu jedem PDA $\mathcal{M}_1$ existiert ein PDA $\mathcal{M}_2$ mit

$$L_l(\mathcal{M}_1) = L_f(\mathcal{M}_2)$$

Beweisidee

- Wir simulieren die Maschine $\mathcal{M}_1$, die über leeren Keller akzeptiert, durch die Maschine $\mathcal{M}_2$, die über finale Zustände akzeptiert.
- $\mathcal{M}_2$ arbeitet wie $\mathcal{M}_1$,
  legt aber ein zusätzliches Symbol ganz unten in den Keller.
  Wenn $\mathcal{M}_1$ seinen Keller geleert hätte (also das neue unterste Symbol sichtbar wird),
  kann $\mathcal{M}_2$ in einen finalen Zustand gehen.

Gleichmächtigkeit: PDAs und kontextfreie Grammatiken

Theorem 5.12 (PDA akzeptieren $\mathbf{L}_2$)

Die Klasse der PDA-akzeptierten Sprachen ist $\mathbf{L}_2$.

Beweis

Dazu beweisen wir die folgenden zwei Lemmata, die zusammen die Aussage des Satzes ergeben.

Gleichmächtigkeit: PDAs und kontextfreie Grammatiken

Lemma 5.13 (cf-Grammatik → PDA)

Zu jeder kontextfreien Grammatik G gibt es einen PDA $\mathcal{M}$ mit

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

Beweis

O.B.d.A. sei die kontextfreie Grammatik $G = (V, T, R, S)$ in Greibach-Normalform: Alle Grammatikregeln haben die Form

$$A \to au \qquad \text{mit } A \in V,\ a \in T,\ u \in V^*$$

Wir konstruieren zu G einen PDA $\mathcal{M}$, der $L(G)$ akzeptiert.

Gleichmächtigkeit: PDAs und kontextfreie Grammatiken

Beweis (Forts.)

Idee: Der Automat $\mathcal{M}$

- vollzieht die Grammatikregeln nach, die angewendet worden sein könnten, um das aktuelle Eingabewort zu erzeugen und
- merkt sich das aktuelle Wort in der Ableitung bzw. dessen Rest
- merkt sich auf dem Keller alle Variablen, die im gedachten Ableitungswort noch vorkommen und noch ersetzt werden müssen.
- Die linkeste Variable liegt zuoberst: $\mathcal{M}$ arbeitet mit der Linksableitung.

Gleichmächtigkeit: PDAs und kontextfreie Grammatiken

Beweis (Forts.)

Genauer:

- Erzeugung eines Wortes mit G beginnt beim Startsymbol S.
  Deshalb S bei $\mathcal{M}$ in Startkonfiguration oben auf dem Keller.
- Angenommen, G hat 2 Regeln mit S auf der linken Seite:
  $S \to aA_1A_2$ und $S \to bB_1B_2$
  Angenommen, der erste Buchstabe des Input-Wortes w ist ein a.

  Wenn w von G erzeugt wurde, hat G die erste der zwei S-Produktionen angewendet.
  Entsprechend: Der Automat $\mathcal{M}$ schiebt A_1A_2 auf den Stack.

Gleichmächtigkeit: PDAs und kontextfreie Grammatiken

Beweis (Forts.)

Genauer:

- Der zweite Buchstabe des Eingabeworts muss durch Anwendung einer Regel $A_1 \to a_1\alpha$ erzeugt worden sein.
  Angenommen, der zweite Buchstabe des Eingabeworts ist a_1. Dann müssen die nächsten Buchstaben des Wortes aus den Variablen in α entstehen.
  Der Automat entfernt A_1 vom Stack und legt α auf den Stack.
- Wenn es zwei Regeln $A_1 \to a_1\alpha_1$ und $A_1 \to a_1\alpha_2$ gibt, dann wählt $\mathcal{M}$ indeterminiert eine der Regeln aus.
- Der PDA hat nur einen einzigen Zustand und akzeptiert über den leeren Keller.

Gleichmächtigkeit: PDAs und kontextfreie Grammatiken

Beweis (Forts.)

Formal:

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

mit

$$K := \{s_0\}$$
$$\Sigma := T$$
$$\Gamma := V$$
$$Z_0 := S$$
$$F := \emptyset$$
$$\Delta := \{((s_0, a, A), (s_0, \alpha)) \mid A \to a\alpha \in R\}$$

Gleichmächtigkeit: PDAs und kontextfreie Grammatiken

Beweis (Forts.)

Damit gilt (Beweis s. Buch):

Es gibt eine Linksableitung $S \Longrightarrow_G^* x\alpha$ mit $x \in T^*, \alpha \in V^*$

gdw

$\mathcal{M}$ rechnet $(s_0, x, S) \vdash_{\mathcal{M}}^* (s_0, \varepsilon, \alpha)$

Daraus folgt unmittelbar:

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

Gleichmächtigkeit: PDAs und kontextfreie Grammatiken

Beispiel 5.14

Die Sprache

$$L = \{ww^R \mid w \in \{a, b\}^+\}$$

wird generiert von der GNF-Grammatik $G = (\{S, A, B\}, \{a, b\}, R, S)$ mit

$$R = \{\ S \to aSA \mid bSB \mid aA \mid bB$$
$$A \to a$$
$$B \to b\ \}$$

Daraus kann man einen PDA mit den folgenden Regeln konstruieren:

$(s_0, a, S)\ \Delta\ (s_0, SA)$
$(s_0, a, S)\ \Delta\ (s_0, A)$
$(s_0, b, S)\ \Delta\ (s_0, SB)$
$(s_0, b, S)\ \Delta\ (s_0, B)$
$(s_0, a, A)\ \Delta\ (s_0, \varepsilon)$
$(s_0, b, B)\ \Delta\ (s_0, \varepsilon)$

Gleichmächtigkeit: PDAs und kontextfreie Grammatiken

Lemma 5.15 (PDA $\to$ 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.

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 unveränder lassen) und das Wort x lesen:

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

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:

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

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

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:

$$
\begin{array}{lrcl}
 & S & \rightarrow & [s_0, Z_0, s_0] \mid [s_0, Z_0, s_1] \\
1. & [s_0, Z_0, s_0] & \rightarrow & \varepsilon \\
2. & [s_0, Z_0, s_0] & \rightarrow & a[s_0, A, s_0] \\
 & [s_0, Z_0, s_1] & \rightarrow & a[s_0, A, s_1] \\
3. & [s_0, A, s_0] & \rightarrow & a[s_0, A, s_0][s_0, A, s_0] \\
 & [s_0, A, s_0] & \rightarrow & a[s_0, A, s_1][s_1, A, s_0] \\
 & [s_0, A, s_1] & \rightarrow & a[s_0, A, s_0][s_0, A, s_1] \\
 & [s_0, A, s_1] & \rightarrow & a[s_0, A, s_1][s_1, A, s_1] \\
4. & [s_0, A, s_1] & \rightarrow & b \\
5. & [s_1, A, s_1] & \rightarrow & b
\end{array}
$$

Gleichmächtigkeit: PDAs und kontextfreie Grammatiken

Beispiel (Forts.)

Lesbarer haben wir damit folgende Grammatik:

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

Man sieht jetzt:

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

Gleichmächtigkeit: PDAs und kontextfreie Grammatiken

Beispiel (Forts.)

Nach Entfernung der überflüssigen Elemente:

$$
\begin{array}{l}
S \rightarrow \varepsilon \mid aD \\
D \rightarrow aDF \mid b \\
F \rightarrow b
\end{array}
$$

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

$$
\begin{array}{l}
S \Longrightarrow aD \Longrightarrow aaDF \Longrightarrow aaaDFF \Longrightarrow aaabFF \\
\Longrightarrow aaabbF \Longrightarrow aaabbb
\end{array}
$$

Teil IV

Kellerautomaten und kontextfreie Sprachen

Abschlusseigenschaften

Theorem 6.1 (Abschlusseigenschaften von $\mathbf{L}_2$)

$\mathbf{L}_2$ *ist abgeschlossen gegen:*

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

Beweis

Seien

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

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

$$L_i = L(G_i)$$

Abschlusseigenschaften

Beweis (Forts.)

zu $\cup$:

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

erzeugt gerade $L_1 \cup L_2$

zu $\circ$:

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

erzeugt gerade $L_1 \circ L_2$

zu *:

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

erzeugt gerade L_1^*. $\square$

Abschlusseigenschaften

Theorem 6.2 (Abschlusseigenschaften von $\mathbf{L}_2$)

$\mathbf{L}_2$ *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

$$R_1 = \{ \begin{array}{l} S \to S'T \\ S' \to aS'b \mid ab \\ T \to cT \mid c\} \end{array}$$

$$R_2 = \{ \begin{array}{l} S \to TS' \\ S' \to bS'c \mid bc \\ T \to aT \mid a\} \end{array}$$

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, $\mathbf{L}_2$ wäre abgeschlossen gegen $\neg$.

Wegen

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

wäre $\mathbf{L}_2$ dann auch abgeschlossen gegen $\cap$ – **Widerspruch** $\square$

Teil IV

Kellerautomaten und kontextfreie Sprachen

Wortproblem

Problem

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

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

Wortproblem

Lösung des Wortproblems für $\mathbf{L}_3$

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

- Konstruiere aus G einen ε-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 ε-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

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

Chart-Parsing

Beispiel 7.1

Die Grammatik $G = (\{S\}, \{a, b\}, R, S)$ mit

$$R = \{\ S \to aSa \mid bSb \mid aa \mid bb\ \}$$

erzeugt die Sprache $\{vv^R \mid v \in \{a, b\}^+\}$

Betrachten wir das Wort $w = abbaabba$.

Was sind mögliche letzte Schritte von Ableitungen, die zu w geführt haben können?

Wir merken uns alle möglichen einzelnen Ableitungsschritte in einer Chart, um Mehrfacharbeit zu vermeiden.

Wenn das Wort w in der Sprache $L(G)$ ist, enthält am Ende der Chart eine mit S markierte Kante, die vom ersten bis zum letzten Knoten reicht.

Chart-Parsing

Beispiel (Forts.)

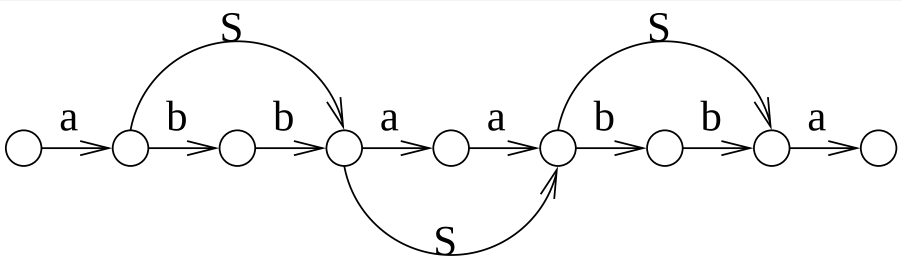


Chart-Parsing

Beispiel (Forts.)

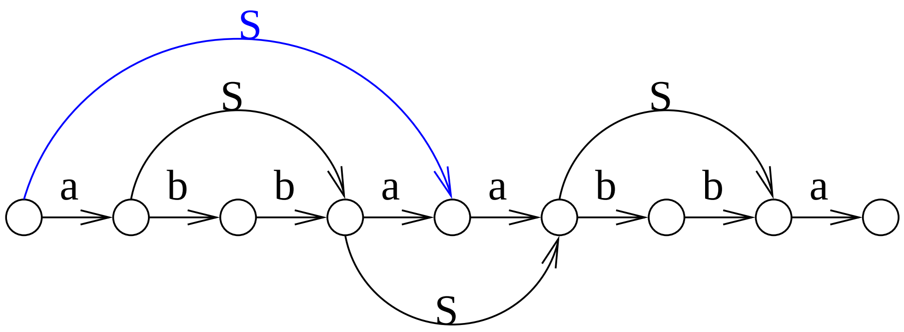


Chart-Parsing

Beispiel (Forts.)

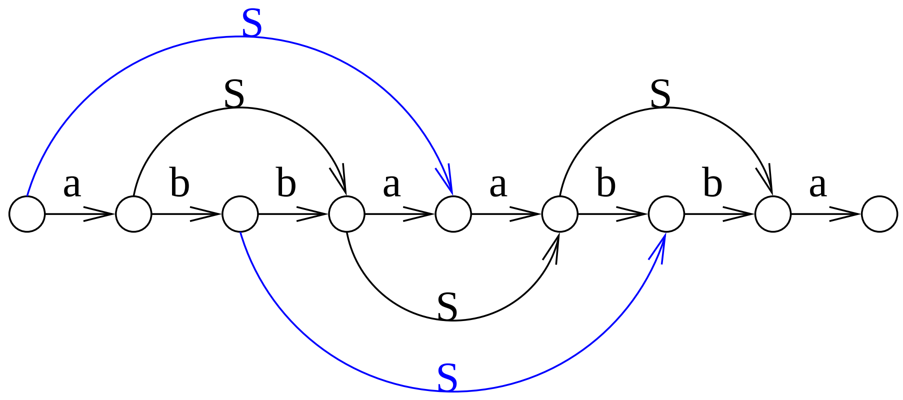


Chart-Parsing

Beispiel (Forts.)

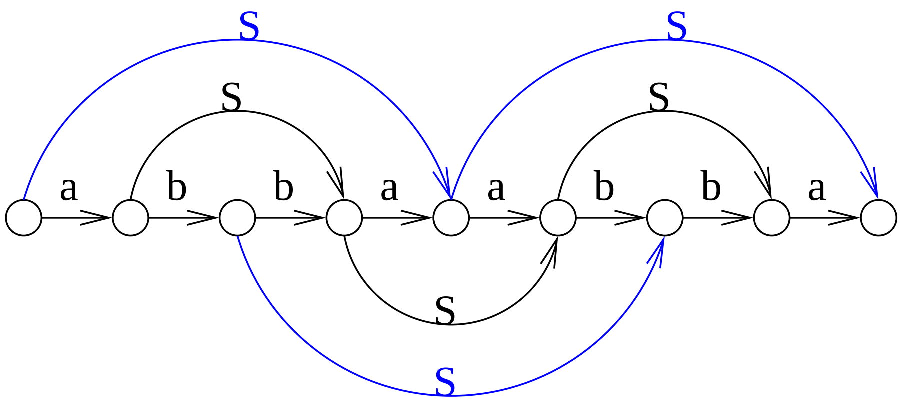


Chart-Parsing

Beispiel (Forts.)

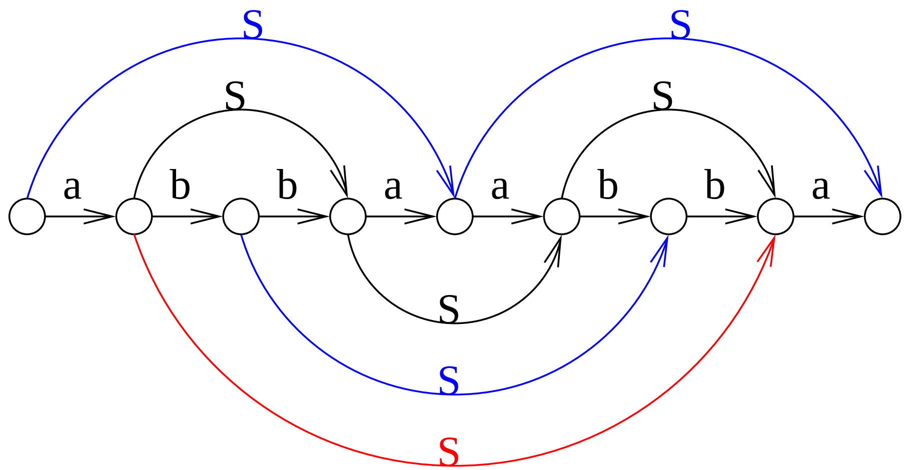


Chart-Parsing

Beispiel (Forts.)

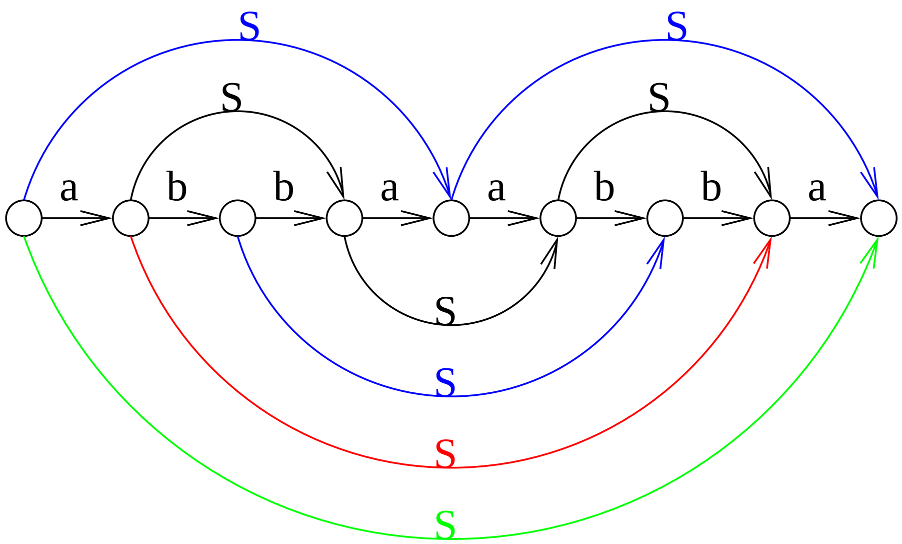


Chart-Parsing

Zur Vereinfachung

Wir fordern:

Grammatik ist in **Chomsky-Normalform**.

Dann:
Immer nur **zwei** benachbarte Kanten betrachten, um herauszufinden, ob darüber eine neue Kante eingefügt werden kann.

Chart-Parsing

Beispiel (Forts.)

Grammatik in CNF, die dieselbe Sprache wie oben erzeugt:
$G = (\{S, S_a, S_b, A, B\}, \{a, b\}, R, S)$ mit

$$
\begin{aligned}
R = \{ \quad & S \to AS_a \mid BS_b \mid AA \mid BB \\
& S_a \to SA \\
& S_b \to SB \\
& A \to a \\
& B \to b\}
\end{aligned}
$$

Chart-Parsing

Darstellung als Array

Für eine Kante, die den i. bis j. Buchstaben überspannt und mit A markiert ist, steht im $[i, j]$-Element des Arrays die Eintragung A.

Definition 7.2 ($M * N$)

Sei $L = L(G)$ kontextfrei, und $G = (V, T, R, S)$ in Chomsky-Normalform. Mit $M, N \subseteq V$ sei

$$M * N := \{A \in V \mid \exists B \in M, \exists C \in N: \; A \to BC \in R\}$$

Chart-Parsing

Definition 7.3 ($w_{i,j}$, $V_{i,j}$)

Sei $w = a_1 \dots a_n$ mit $a_i \in \Sigma$.
Dann:

- $w_{i,j} := a_i \dots a_j$ ist das Infix von w vom i-ten bis zum j-ten Buchstaben
- $V_{i,j} := \{A \in V \mid A \Longrightarrow_G^* w_{i,j}\}$

Chart-Parsing

Lemma 7.4

Sei $w = a_1 \dots a_n$, $a_i \in \Sigma$, d.h. $|w| = n$. Dann gilt:

1. $V_{i,i} = \{A \in V \mid A \to a_i \in R\}$
2. $V_{i,k} = \bigcup_{j=i}^{k-1} V_{i,j} * V_{j+1,k}$ *für* $1 \leq i < k \leq n$

Beachte:
Die Grammatik muss in Chomsky-Normalform sein!

Chart-Parsing

Beweis

1. $V_{i,i} = \{A \in V \mid A \Longrightarrow_G^* a_i\} = \{A \in V \mid A \to a_i \in R\}$, da G in CNF ist.

$A \in V_{i,k}$ mit $1 \leq i < k \leq n$
gdw $A \Longrightarrow_G^* a_i \dots a_k$
gdw $\exists j,\ i \leq j < k:\ \exists B, C \in V : A \Longrightarrow BC$, und
$B \Longrightarrow_G^* w_{i,j} \neq \varepsilon$
und $C \Longrightarrow_G^* w_{j+1,k} \neq \varepsilon$ (da G in CNF ist)
gdw $\exists j,\ i \leq j < k:\ \exists B, C \in V : A \Longrightarrow BC$
und $B \in V_{i,j}$ und $C \in V_{j+1,k}$

2. gdw $\exists j,\ i \leq j < k : A \in V_{i,j} * V_{j+1,k}$

$\square$

Teil IV

Kellerautomaten und kontextfreie Sprachen

CYK-Algorithmus (Cocke-Younger-Kasami)

Algorithmus

Input sei eine Grammatik G in CNF und ein Wort $w = a_1 \dots a_n \in \Sigma^*$.

(i) **for** $i := 1$ *to* n **do** $\quad$ `/* Regeln A → a eintragen */`
$\quad V_{i,i} := \{A \in V \mid A \to a_i \in R\}$

(ii) **for** $h := 1$ *to* $n - 1$ **do**
$\quad$ **for** $i := 1$ *to* $n - h$ **do**

$$V_{i,i+h} = \bigcup_{j=i}^{i+h-1} V_{i,j} * V_{j+1,i+h}$$

(iii) **if** $S \in V_{1,n}$ **then return** `Ausgabe` $w \in L(G)$
$\quad$ **else return** `Ausgabe` $w \notin L(G)$

CYK-Algorithmus (Cocke-Younger-Kasami)

Eigenschaften

Für Wörter der Länge $|w| = n$ entscheidet der CYK-Algorithmus in der Größenordnung von n^3 Schritten, ob $w \in L(G)$ ist.

CYK-Algorithmus (Cocke-Younger-Kasami)

Beispiel 8.1 (CYK)

Eine Grammatik in CNF, die dieselbe Sprache wie oben erzeugt:
$G = (\{S, S_a, S_b, A, B\}, \{a, b\}, R, S)$

$$\begin{aligned} R = \{ \quad & S \to AS_a \mid BS_b \mid AA \mid BB \\ & S_a \to SA \\ & S_b \to SB \\ & A \to a \\ & B \to b\} \end{aligned}$$

Die Sprache ist: $L(G) = \{vv^R \mid v \in \{a, b\}^+\}$

Auführlich an der Tafel.