Theoretische Informatik
Kapitel 3 – Kontextfreie Sprachen

Sommersemester 2024

Dozentin: Mareike Mutz

im Wechsel mit

Prof. Dr. M. Leuschel

Prof. Dr. J. Rothe

REG versus CF

Theorem

REG *ist echt in* CF *enthalten.*

Beweis:

- Wir wissen: $L = \{a^m b^m \mid m \geq 1\}$ ist nicht regulär.
- Andererseits ist L kontextfrei, wie die einfache kontextfreie Grammatik $G = (\{a, b\}, \{S\}, S, P)$ mit den Regeln

$$P = \{S \to aSb \mid ab\}$$

  zeigt.
- Somit ist $L \in \mathrm{CF} - \mathrm{REG}$, also $\mathrm{REG} \subset \mathrm{CF}$. ❑

Normalformen

Ziel: Vereinfachung kontextfreier Grammatiken (kurz kfGs).

Ausnahmeregelung für das leere Wort:
Für kfGs, und nur für diese, erlauben wir λ-Regeln, d.h., Regeln der Form $A \to \lambda$, *auch wenn A nicht das Startsymbol ist*.

Dies kann manchmal wünschenswert sein.
Für kfGs kann man dies o.B.d.A. zulassen, denn λ-Regeln können ungestraft wieder entfernt werden.

Normalformen: λ-freie Grammatik

Definition (λ-freie Grammatik)

Eine kfG $G = (\Sigma, N, S, P)$ heißt *λ-frei*, falls in P keine Regel

$$A \to \lambda$$

mit $A \neq S$ auftritt.

Anmerkung: wir hatten die Umformung gesehen die eine Grammatik mit $S \to \lambda$ als einzige verkürzende Regel so umwandelt, dass S auf keiner rechten Seite auftaucht (Sonderregel für λ).

Theorem

Zu jeder kfG G (mit λ-Regeln) gibt es eine λ-freie kfG G' mit

$$L(G) = L(G').$$

Normalformen: λ-freie Grammatik

Beispiel: Wie kann man hier λ-Regeln löschen? Grammatik $G = (\Sigma, N, S, P)$ mit $\Sigma = \{a, b\}$, $N = \{S, A\}$ und der Regelmenge

$$\begin{array}{rcll} P & = & \{ & S \rightarrow bAAb, \\ & & & A \rightarrow aA \mid \lambda \quad \}. \end{array}$$

Normalformen: λ-freie Grammatik

Beispiel: Wie kann man hier λ-Regeln löschen? Grammatik $G = (\Sigma, N, S, P)$ mit $\Sigma = \{a, b, c\}$, $N = \{S, A, B, C\}$ und der Regelmenge

$$
\begin{aligned}
P \quad = \quad \{ \quad & S \rightarrow ABC, \\
& A \rightarrow aC \mid aBB, \\
& B \rightarrow CC \mid b, \\
& C \rightarrow c \mid \lambda \qquad \}.
\end{aligned}
$$

Normalformen: λ-freie Grammatik

Beweis: Der Beweis beruht auf folgender Konstruktion:

Gegeben: kfG $G = (\Sigma, N, S, P)$ mit λ-Regeln; o.B.d.A. sei $\lambda \notin L(G)$ (andernfalls ist die Sonderregel für λ anzuwenden).

Gesucht: λ-freie kfG G' mit $L(G) = L(G')$.

Konstruktion:

1. Bestimme die Menge

$$N_\lambda = \{A \in N \mid A \vdash_G^* \lambda\}$$

sukzessive wie folgt:

(a) Ist $A \rightarrow \lambda$ eine Regel in P, so ist $A \in N_\lambda$.

(b) Ist $A \rightarrow A_1 A_2 \cdots A_k$ eine Regel in P mit $k \geq 1$ und $A_i \in N_\lambda$ für alle i, $1 \leq i \leq k$, so ist $A \in N_\lambda$.

Normalformen: λ-freie Grammatik

Beispiel: Was ist $N_\lambda = \{A \in N \mid A \vdash_G^* \lambda\}$ hier?

$$
\begin{array}{rcll}
P & = & \{ & S \to ABC, \\
 & & & A \to aC \mid aBB, \\
 & & & B \to CC \mid b, \\
 & & & C \to c \mid \lambda \qquad \}.
\end{array}
$$

(a) Ist $A \to \lambda$ eine Regel in P, so ist $A \in N_\lambda$.

(b) Ist $A \to A_1 A_2 \cdots A_k$ eine Regel in P mit $k \geq 1$ und $A_i \in N_\lambda$ für alle i, $1 \leq i \leq k$, so ist $A \in N_\lambda$.

Normalformen: λ-freie Grammatik

2. Füge für jede Regel der Form

$$B \rightarrow uAv \quad \text{mit } B \in N,\ A \in N_\lambda \text{ und } uv \in (N \cup \Sigma)^+$$

   zusätzlich die Regel

$$B \rightarrow uv$$

   zu P hinzu.

3. Entferne alle Regeln $A \rightarrow \lambda$ aus P.

Schritt 2 muss auch für neu generierte Regeln iterativ angewendet werden.

Dies ergibt die gesuchte λ-freie kfG G' mit $L(G) = L(G')$. ❑

Normalformen: λ-freie Grammatik

Beispiel:

$$P = \{ \quad S \to bAAb, \\ A \to aA \mid \lambda \quad \}.$$

$$P = \{ \quad S \to ABC, \\ A \to aC \mid aBB, \\ B \to CC \mid b, \\ C \to c \mid \lambda \quad \}.$$

Wir haben $N_\lambda = \{A\}$ bzw $N_\lambda = \{B, C\}$. Füge für jede Regel der Form

$$B \to uAv \quad \text{mit } B \in N,\ A \in N_\lambda \text{ und } uv \in (N \cup \Sigma)^+$$

zusätzlich die Regel $B \to uv$ zu P hinzu und lösche alle Regeln $A \to \lambda$ aus P.

Normalformen: λ-freie Grammatik

Beispiel: Wir betrachten die Grammatik $G = (\Sigma, N, S, P)$ mit

- dem terminalen Alphabet $\Sigma = \{0, 1\}$,
- dem nichtterminalen Alphabet $N = \{S, A, B, C, D, E\}$ und
- der Regelmenge

$$
\begin{array}{rcll}
P & = & \{ & S \rightarrow ABD, \\
 & & & A \rightarrow ED \mid BB, \\
 & & & B \rightarrow AC \mid \lambda, \\
 & & & C \rightarrow \lambda, \\
 & & & D \rightarrow 0, \\
 & & & E \rightarrow 1 \qquad \}.
\end{array}
$$

Normalformen: λ-freie Grammatik

1. Bestimme für die Grammatik G nun gemäß dem Beweis des obigen Satzes die Menge N_λ:
   (a) $N_\lambda = \{B, C\}$, da $B \rightarrow \lambda \in P$ und $C \rightarrow \lambda \in P$;
   (b) $N_\lambda = \{A, B, C\}$, da $A \rightarrow BB \in P$.

2. Ergänze die Regeln gemäß dem zweiten Schritt der Konstruktion im Beweis dieses Satzes, der iterativ auch auf neue Regeln anzuwenden ist.

3. Entferne alle Regeln
$$A \rightarrow \lambda$$
mit $A \in N$ aus P.

Normalformen: λ-freie Grammatik

Wir erhalten eine kontextfreie Grammatik

$$G' = (\Sigma, \{S, A, B, C, D, E\}, S, P')$$

mit

$$\begin{array}{rcll} P' & = & \{ & S \to ABD \mid BD \mid AD \mid D, \\ & & & A \to ED \mid BB \mid B, \\ & & & B \to AC \mid A \mid C, \\ & & & D \to 0, \\ & & & E \to 1 \qquad \} \end{array}$$

und $L(G') = L(G)$.
(Die Regeln $B \to AC$ und $B \to C$ sind überflüssig, da C auf keiner linken Seite einer Regel steht, und können wieder entfernt werden.)

Normalformen: Eliminierung einfacher Regeln

Definition (Einfache Regel)

Regeln $A \rightarrow B$ heißen *einfach*, falls A und B Nichtterminale sind.

Theorem

Zu jeder kfG G gibt es eine kfG G' ohne einfache Regeln, so dass

$$L(G) = L(G').$$

Normalformen: Eliminierung einfacher Regeln

Beispiel: Wie kann man hier einfache Regeln eliminieren? Grammatik $G = (\Sigma, N, S, P)$ mit $\Sigma = \{a, b\}$, $N = \{S, A, B\}$ und der Regelmenge

$$
\begin{array}{rcll}
P & = & \{ & S \to A, \\
 & & & A \to aA \mid B, \\
 & & & B \to bB \mid b \quad \}.
\end{array}
$$

Normalformen: Eliminierung einfacher Regeln

Beispiel: Wie kann man hier einfache Regeln eliminieren? Grammatik $G = (\Sigma, N, S, P)$ mit $\Sigma = \{a, b\}$, $N = \{S, A, B\}$ und der Regelmenge

$$P = \{ \begin{array}{l} S \to A, \\ A \to aA \mid B, \\ B \to bB \mid b \quad \}. \end{array}$$

Ordnung der "Aufrufe": $S \mapsto A \mapsto B$. Zuerst alle einfachen Regeln mit B rechts ersetzen, dann alle mit A rechts.

Normalformen: Eliminierung einfacher Regeln

Beispiel: Wie kann man hier einfache Regeln eliminieren? Grammatik $G = (\Sigma, N, S, P)$ mit $\Sigma = \{a, b, c\}$, $N = \{S, A, B\}$ und der Regelmenge

$$
\begin{array}{rcll}
P & = & \{ & S \rightarrow A, \\
 & & & A \rightarrow aA \mid B \mid c, \\
 & & & B \rightarrow bB \mid A \qquad \}.
\end{array}
$$

Normalformen: Eliminierung einfacher Regeln

Beispiel: Wie kann man hier einfache Regeln eliminieren? Grammatik $G = (\Sigma, N, S, P)$ mit $\Sigma = \{a, b, c\}$, $N = \{S, A, B\}$ und der Regelmenge

$$\begin{array}{rcll} P & = & \{ & S \to A, \\ & & & A \to aA \mid B \mid c, \\ & & & B \to bB \mid A \qquad \}. \end{array}$$

Ordnung der “Aufrufe”: $S \mapsto A \mapsto B \mapsto A$. A und B fusionieren, dann in Regel für S ersetzen.

Normalformen: Eliminierung einfacher Regeln

Beweis: Der Beweis beruht auf folgender Konstruktion:

Gegeben: kfG $G = (\Sigma, N, S, P)$ (mit einfachen Regeln).

Gesucht: kfG G' ohne einfache Regeln, so dass $L(G) = L(G')$.

Konstruktion: 1. Entferne alle Zyklen

$$B_1 \rightarrow B_2, \quad B_2 \rightarrow B_3, \quad \ldots, \quad B_{k-1} \rightarrow B_k, \quad B_k \rightarrow B_1$$

mit $B_i \in N$ und ersetze alle B_i (in den verbleibenden Regeln) durch ein neues Nichtterminal B.

Normalformen: Eliminierung einfacher Regeln

Beweis: Der Beweis beruht auf folgender Konstruktion:

Gegeben: kfG $G = (\Sigma, N, S, P)$ (mit einfachen Regeln).

Gesucht: kfG G' ohne einfache Regeln, so dass $L(G) = L(G')$.

Konstruktion:

1. Entferne alle Zyklen

$$B_1 \rightarrow B_2, \quad B_2 \rightarrow B_3, \quad \ldots, \quad B_{k-1} \rightarrow B_k, \quad B_k \rightarrow B_1$$

mit $B_i \in N$ und ersetze alle B_i (in den verbleibenden Regeln) durch ein neues Nichtterminal B.

(Anmerkung: Hier sind wirklich nur Zyklen gemeint, die nur aus einfachen Regeln bestehen.)

Normalformen: Eliminierung einfacher Regeln

2. Topologische Sortierung: Nummeriere die Nichtterminale als $\{A_1, A_2, \ldots, A_n\}$ so, dass aus $A_i \to A_j$ folgt: $i < j$.

3. Für $k = n-1, n-2, \ldots, 1$ (rückwärts!) eliminiere die Regel $A_k \to A_\ell$ mit $k < \ell$ so:
   Sind die Regeln mit A_ℓ als linker Seite gegeben durch

$$A_\ell \to u_1 \mid u_2 \mid \cdots \mid u_m,$$

   so entferne $A_k \to A_\ell$ und füge die folgenden Regeln hinzu:

$$A_k \to u_1 \mid u_2 \mid \cdots \mid u_m.$$

Dies liefert die gesuchte kfG G' ohne einfache Regeln mit $L(G) = L(G')$. ❑

Normalformen: Eliminierung einfacher Regeln

Beispiel: Betrachte die Grammatik $G = (\Sigma, N, S, P)$ mit

- dem terminalen Alphabet $\Sigma = \{0, 1\}$,
- dem nichtterminalen Alphabet $N = \{S, A, B, C, D\}$ und
- der Regelmenge

$$
\begin{array}{rcll}
P & = & \{ & S \rightarrow A \mid 0C \mid 00 \mid 0000, \\
 & & & A \rightarrow B \mid 11, \\
 & & & B \rightarrow C \mid 1, \\
 & & & C \rightarrow D, \\
 & & & D \rightarrow B \qquad \qquad \qquad \}.
\end{array}
$$

Normalformen: Eliminierung einfacher Regeln

1. Entferne alle Zyklen über Nichtterminalsymbole gemäß dem Beweis des Satzes.
   $B \rightarrow C, \quad C \rightarrow D, \quad D \rightarrow B$ werden entfernt und alle Vorkommen von B, C, D in den restlichen Regeln werden durch B ersetzt:

$$
\begin{array}{rcll}
P_1 & = & \{ & S \rightarrow A \mid 0B \mid 00 \mid 0000, \\
 & & & A \rightarrow B \mid 11, \\
 & & & B \rightarrow 1 \qquad\qquad\qquad \}.
\end{array}
$$

2. Nummeriere die Nichtterminalsymbole: S (1), A (2), B (3).

Normalformen: Eliminierung einfacher Regeln

3. Eliminiere Regeln wie folgt:
   (a) $A \rightarrow B$ wird entfernt und dafür $A \rightarrow 1$ hinzugefügt;
   (b) $S \rightarrow A$ wird entfernt und dafür $S \rightarrow 11$ und $S \rightarrow 1$ hinzugefügt:

$$\begin{array}{rcll} P' & = & \{ & S \rightarrow 1 \mid 11 \mid 0B \mid 00 \mid 0000, \\ & & & A \rightarrow 1 \mid 11, \\ & & & B \rightarrow 1 \qquad\qquad\qquad\qquad \}. \end{array}$$

Die so erhaltene Grammatik $G = (\Sigma, N', S, P')$ erfüllt

$$L(G') = L(G).$$

❑

Wiederholung: Normalformen für CF-Sprachen

- Kontextfreie Grammatiken (kfGs) sind Grammatiken mit Regeln der Form $A \rightarrow A_1A_2$ und $A \rightarrow a$, wobei A, A_1, A_2 Nichtterminale und a ein Terminal sind.

Wiederholung: Normalformen für CF-Sprachen

- Kontextfreie Grammatiken (kfGs) sind Grammatiken mit Regeln der Form $A \rightarrow A_1A_2$ und $A \rightarrow a$, wobei A, A_1, A_2 Nichtterminale und a ein Terminal sind.
- Wir haben einen Algorithmus kennen gelernt, der äquivalente λ-freie Grammatiken erzeugen kann zu einer kfG mit Regeln $A \rightarrow \lambda$.
- Darum erlauben wir in kfG auch Lambda-Regeln von anderen Nichtterminalen aus als dem Startsymbol

Wiederholung: Normalformen für CF-Sprachen

- Kontextfreie Grammatiken (kfGs) sind Grammatiken mit Regeln der Form $A \rightarrow A_1 A_2$ und $A \rightarrow a$, wobei A, A_1, A_2 Nichtterminale und a ein Terminal sind.
- Wir haben einen Algorithmus kennen gelernt, der äquivalente λ-freie Grammatiken erzeugen kann zu einer kfG mit Regeln $A \rightarrow \lambda$.
- Darum erlauben wir in kfG auch Lambda-Regeln von anderen Nichtterminalen aus als dem Startsymbol
- Wir können genauso auch "einfache Regeln" entfernen. Also Regeln der Form $A \rightarrow B$ mit $A, B \in N$

Chomsky-Normalform

Definition (Chomsky-Normalform)

Eine kfG $G = (\Sigma, N, S, P)$ mit $\lambda \notin L(G)$ ist in *Chomsky-Normalform* (kurz CNF), falls alle Regeln in P eine der folgenden Formen haben:

- $A \rightarrow BC$ mit $A, B, C \in N$;
- $A \rightarrow a$ mit $A \in N$ und $a \in \Sigma$.

Theorem

Zu jeder kfG G mit $\lambda \notin L(G)$ gibt es eine kfG G' in CNF, so dass gilt:

$$L(G) = L(G').$$

Normalformen: Chomsky-Normalform

Beispiel: Wie könnte man hier die CNF (Regeln $A \to BC$ oder $A \to a$) erreichen? Grammatik $G = (\Sigma, N, S, P)$ mit $\Sigma = \{a\}$, $N = \{S, A\}$ und der Regelmenge

$$P = \{ \quad S \to AAA, \quad A \to aA \mid a \quad \}.$$

Chomsky-Normalform

Beweis: Der Beweis beruht auf folgender Konstruktion:

Gegeben: λ-freie kfG $G = (\Sigma, N, S, P)$ ohne einfache Regeln; $\lambda \notin L(G)$.

Gesucht: kfG G' in CNF mit $L(G) = L(G')$.

Konstruktion:

1. Regeln $A \rightarrow a$ mit $A \in N$ und $a \in \Sigma$ sind in CNF und werden übernommen. Betrachte im Folgenden nur noch die restlichen Regeln; diese sind von der Form:

$$A \rightarrow x \quad \text{mit } x \in (N \cup \Sigma)^* \text{ und } |x| \geq 2.$$

2.
   - Füge für jedes $a \in \Sigma$ ein neues Nichtterminal B_a zu N hinzu,
   - ersetze jedes Vorkommen von $a \in \Sigma$ durch B_a und
   - füge zu P die Regel $B_a \rightarrow a$ hinzu.

Chomsky-Normalform

3. Nicht in CNF sind nun nur noch Regeln der Form

   $A \to B_1 B_2 \cdots B_k$, wobei $k \geq 3$ und jedes B_i ein Nichtterminal ist.

   Jede solche Regel wird ersetzt durch die Regeln:

$$
\begin{array}{rcl}
A & \to & B_1 C_2, \\
C_2 & \to & B_2 C_3, \\
& \vdots & \\
C_{k-2} & \to & B_{k-2} C_{k-1}, \\
C_{k-1} & \to & B_{k-1} B_k,
\end{array}
$$

   wobei $C_2, C_3, \ldots, C_{k-1}$ neue Nichtterminale sind.

Dies liefert die gesuchte Grammatik G' in CNF mit $L(G) = L(G')$. ❑

Chomsky-Normalform: Beispiel

Beispiel: Wir betrachten die transformierte Grammatik aus dem vorigen Beispiel. O.B.d.A. entfernen wir die Regeln

$$A \rightarrow 1 \mid 11$$

und das Nichtterminal A und erhalten die Grammatik $G = (\Sigma, N, S, P)$ mit

- dem terminalen Alphabet $\Sigma = \{0, 1\}$,
- dem nichtterminalen Alphabet $N = \{S, B\}$ und
- der Regelmenge

$$\begin{array}{rcll} P & = & \{ & S \rightarrow 1 \mid 11 \mid 0B \mid 00 \mid 0000, \\ & & & B \rightarrow 1 \qquad\qquad\qquad\qquad \}. \end{array}$$

Chomsky-Normalform: Beispiel

1. Regeln $A \to a$ mit $A \in N$ und $a \in \Sigma$ sind in CNF und werden übernommen. Diese zwei Regeln werden also übernommen:

$$S \to 1 \quad \text{und} \quad B \to 1.$$

2. Wir führen zwei neue Nichtterminalsymbole ein, X_1 und X_0, und erhalten die folgende Regelmenge:

$$\begin{aligned} P_1 \quad = \quad \{ \quad & S \to 1 \mid X_1X_1 \mid X_0B \mid X_0X_0 \mid X_0X_0X_0X_0, \\ & B \to 1, \\ & X_1 \to 1, \\ & X_0 \to 0 \qquad \}. \end{aligned}$$

Chomsky-Normalform: Beispiel

3. Noch nicht in Chomsky-Normalform ist die Regel

$$S \rightarrow X_0 X_0 X_0 X_0.$$

Diese Regel ersetzen wir durch die drei Regeln

$$S \rightarrow X_0 C_2, \quad C_2 \rightarrow X_0 C_3, \quad C_3 \rightarrow X_0 X_0,$$

wobei C_2 und C_3 neue Nichtterminalsymbole sind.

Chomsky-Normalform: Beispiel

Wir erhalten die Grammatik $G' = (\Sigma, N', S, P')$ mit

- $\Sigma = \{0, 1\}$,
- $N' = \{S, B, X_0, X_1, C_2, C_3\}$ und
- Regelmenge

$$
\begin{array}{rcll}
P' & = & \{ & S \rightarrow 1 \mid X_1 X_1 \mid X_0 B \mid X_0 X_0 \mid X_0 C_2, \\
 & & & C_2 \rightarrow X_0 C_3, \\
 & & & C_3 \rightarrow X_0 X_0, \\
 & & & B \rightarrow 1, \\
 & & & X_1 \rightarrow 1, \\
 & & & X_0 \rightarrow 0 \qquad \}
\end{array}
$$

in CNF mit $L(G) = L(G')$.

Chomsky-Normalform: Beispiel

Gibt es eine andere kfG G'' in CNF, die nicht isomorph zu G' ist und so dass $L(G'') = L(G) = \{1, 11, 01, 00, 0000\}$?

Chomsky-Normalform: Beispiel

Gibt es eine andere kfG G'' in CNF, die nicht isomorph zu G' ist und so dass $L(G'') = L(G) = \{1, 11, 01, 00, 0000\}$?

Ja, unendlich viele. Zum Beispiel: $G'' = (\Sigma, N'', S, P'')$ wo wir X_1 entfernt haben:

- $\Sigma = \{0, 1\}$, $N'' = \{S, B, X_0, C_2, C_3\}$ und
- Regelmenge

$$
\begin{array}{rcll}
P'' & = & \{ & S \rightarrow 1 \mid BB \mid X_0B \mid X_0X_0 \mid X_0C_2, \\
 & & & C_2 \rightarrow X_0C_3, \\
 & & & C_3 \rightarrow X_0X_0, \\
 & & & B \rightarrow 1, \\
 & & & X_0 \rightarrow 0 \qquad \}
\end{array}
$$

Chomsky-Normalform: Beispiel

Gibt es eine andere kfG G'' in CNF, die nicht isomorph zu G' ist und so dass $L(G'') = L(G) = \{1, 11, 01, 00, 0000\}$?

Anderes Beispiel: $G''' = (\Sigma, N'', S, P''')$:

- $\Sigma = \{0, 1\}$, $N'' = \{S, B, X_0, C_2, C_3\}$ und
- Regelmenge

$$
\begin{array}{rcll}
P' & = & \{ & S \rightarrow 1 \mid BB \mid X_0 B \mid X_0 X_0 \mid C_2 X_0, \\
 & & & C_2 \rightarrow C_3 X_0, \\
 & & & C_3 \rightarrow X_0 X_0, \\
 & & & B \rightarrow 1, \\
 & & & X_0 \rightarrow 0 \qquad \}
\end{array}
$$

Chomsky-Normalform

Bemerkung: Es sei G eine Grammatik in Chomsky-Normalform, $w \in L(G)$ und $w \neq \lambda$.

- Jede Ableitung von w in G besteht aus genau $2|w| - 1$ Schritten. Es wird $(|w| - 1)$-mal eine Regel der Form

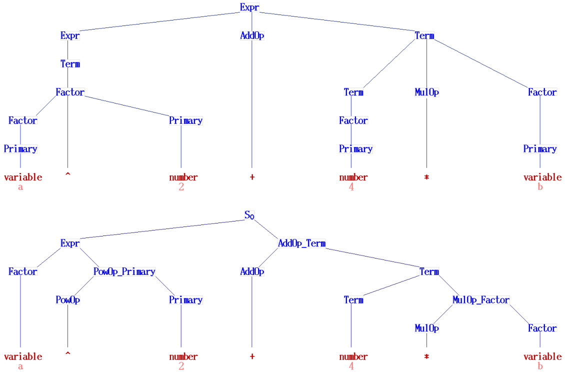
$$A \rightarrow BC$$

angewandt und $|w|$-mal eine Regel der Form

$$A \rightarrow a.$$

- Jeder Syntaxbaum für w in G ist ein Binärbaum.

CNF - Binärbaum.



Quelle: https://en.wikipedia.org/wiki/Chomsky_normal_form

“Abstract syntax tree of the arithmetic expression $a^2 + 4 * b$ wrt. the example grammar (top) and its Chomsky normal form (bottom)”

Greibach-Normalform

Definition (Greibach-Normalform)

Eine kontextfreie Grammatik $G = (\Sigma, N, S, P)$ mit $\lambda \notin L(G)$ ist in *Greibach-Normalform* (kurz GNF), falls jede Regel in P die folgende Form hat:

$$A \to aB_1B_2 \cdots B_k \quad \text{mit } k \geq 0 \text{ und } A, B_i \in N \text{ und } a \in \Sigma.$$

Theorem

Zu jeder kfG G mit $\lambda \notin L(G)$ gibt es eine kfG G' in GNF, so dass

$$L(G) = L(G').$$

ohne Beweis

Greibach-Normalform: Beispiel

Beispiel: Die Grammatik $G = (\Sigma, N, S, P)$ mit

- $\Sigma = \{0, 1\}$,
- $N = \{S, A\}$ und
- Regelmenge

$$P = \{ \begin{array}{l} S \to 0A \mid 0SA, \\ A \to 1 \end{array} \}$$

ist offensichtlich in Greibach-Normalform, und es gilt

$$L(G) = \{0^n 1^n \mid n \geq 1\}.$$

Greibach-Normalform: Beispiel

Beispiel: Die Grammatik $G = (\Sigma, N, S, P)$ mit

- $\Sigma = \{0, 1\}$,
- $N = \{S, A, B\}$ und
- Regelmenge

$$P = \{ \quad S \rightarrow 0A \mid 1B \mid 0SA \mid 1SB, \quad A \rightarrow 0, \quad B \rightarrow 1 \quad \}$$

ist offensichtlich in Greibach-Normalform, und es gilt

$$L(G) = \{x\, sp(x) \mid x \in \{0, 1\}^{+}\}.$$

Greibach-Normalform

Bemerkung: Grammatiken in GNF unterscheidet man bezüglich der Längen der rechten Seiten der Produktionen.

- Man kann zeigen, dass sich jede kontextfreie Grammatik in eine äquivalente kontextfreie Grammatik in Greibach-Normalform transformieren lässt, so dass für alle Regeln

  $$A \rightarrow aB_1 B_2 \cdots B_k$$

  stets $k \leq 2$ gilt.

- Für den Spezialfall $k \in \{0, 1\}$ erhalten wir gerade die Definition rechtslinearer Grammatiken.

Greibach-Normalform

Bemerkung: Es sei G eine Grammatik in Greibach-Normalform, $w \in L(G)$ und $w \neq \lambda$. Dann besteht jede Ableitung von w in G aus genau $|w|$ Schritten.
(Es wird in jedem Ableitungsschritt genau ein Terminalsymbol von w abgeleitet.)

Pumping-Lemma

Wie könnte ein Pumping Lemma für kontextfreie Sprachen aussehen?

Beispiel: $G = (\{a, b\}, \{S\}, S, P)$ mit den Regeln

$$P = \{S \rightarrow aSb \mid ab\}$$

Wort $aaabbb \in L(G)$.

Pumping-Lemma

Wie könnte ein Pumping Lemma für kontextfreie Sprachen aussehen?

Beispiel: $G = (\{a, b\}, \{S\}, S, P)$ mit den Regeln

$$P = \{S \to aSb \mid ab\}$$

Wort $aaabbb \in L(G)$.

Zerlegung $uvxy = aaabbb$.

- $uv^0x^0y = aabb \in L$
- $uv^2x^2y = aaaabbbb \in L$
- $uv^3x^3y = aaaaabbbbb \in L$

Pumping-Lemma

Wie könnte ein Pumping Lemma für kontextfreie Sprachen aussehen?
Zweites Beispiel: $G = (\{a, b, c\}, \{S\}, S, P)$ mit den Regeln

$$P = \{S \rightarrow aSb \mid c\}$$

Wort $aacbb \in L(G)$.

Pumping-Lemma

Wie könnte ein Pumping Lemma für kontextfreie Sprachen aussehen?

Zweites Beispiel: $G = (\{a, b, c\}, \{S\}, S, P)$ mit den Regeln

$$P = \{S \to aSb \mid c\}$$

Wort $aacbb \in L(G)$.

Zerlegung $uvwxy = aacbb$.

- $uv^0wx^0y = acb \in L$
- $uv^2wx^2y = aaacbbb \in L$
- $uv^3wx^3y = aaaacbbbb \in L$

Pumping-Lemma

Ziel: Nachweis, dass bestimmte Sprachen nicht kontextfrei sind.

Theorem (Pumping-Lemma für kontextfreie Sprachen)

Sei L eine kontextfreie Sprache. Dann existiert eine (von L abhängige) Zahl $n \geq 1$, so dass sich alle Wörter $z \in L$ mit $|z| \geq n$ zerlegen lassen in

$$z = uvwxy,$$

wobei gilt:

1. $|vx| \geq 1$,
2. $|vwx| \leq n$,
3. $(\forall i \geq 0)\,[uv^iwx^iy \in L]$.

Pumping-Lemma: Beweis

Beweis:

- Es sei L eine kontextfreie Sprache. Wir setzen voraus, dass $\lambda \notin L$.
- Es sei $G = (\Sigma, N, S, P)$ eine kfG für L in CNF mit k Nichtterminalen.
- Wir wählen $n = 2^{k+1}$.
- Sei $z \in L$ ein beliebiges Wort mit $|z| \geq n$.
- Der Syntaxbaum B der Ableitung

$$S \vdash_G^* z$$

ist (bis auf den letzten Ableitungsschritt) ein Binärbaum mit

$$|z| \geq n = 2^{k+1}$$

Blättern.

Pumping-Lemma: Beweis: Lemma über Binärbäume

Lemma

Jeder Binärbaum B_k mit mindestens 2^k Blättern besitzt mindestens einen Pfad der Länge[a] *mindestens k.*

[a] Die Länge eines Pfades ist die Anzahl seiner Kanten.

Pumping-Lemma: Beweis: Lemma über Binärbäume

Beweis: Der Beweis wird durch Induktion über k geführt.

Induktionsanfang: $k = 0$. Jeder Binärbaum B_0 mit mindestens $2^0 = 1$ Blatt besitzt mindestens einen Pfad der Länge mindestens 0.

Induktionsschritt: $k \mapsto k + 1$. Die Behauptung gelte für k.

- Betrachte einen beliebigen Binärbaum B_{k+1} mit mindestens 2^{k+1} Blättern.
- Mindestens einer seiner Teilbäume hat mindestens $2^{k+1}/2$ Blätter (sonst hätte B_{k+1} weniger als
$$\frac{2^{k+1}}{2} + \frac{2^{k+1}}{2} = 2^{k+1}$$
Blätter); sei B_k dieser Teilbaum.

Pumping-Lemma: Beweis: Lemma über Binärbäume

- Nach Induktionsvoraussetzung gibt es in B_k einen Pfad α der Länge mindestens k.
- Verlängert man α zur Wurzel von B_{k+1}, so ergibt sich ein Pfad der Länge mindestens $k+1$ in B_{k+1}.

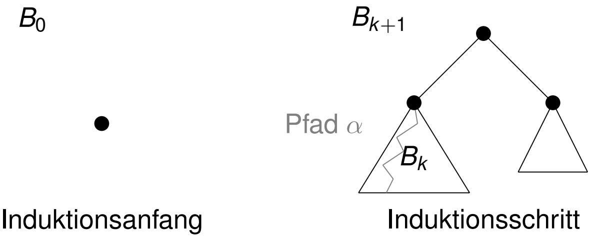


Abbildung: Beweis des Lemmas über Binärbäume ❑

Pumping-Lemma: Beweis

Weiter im Beweis des Pumping-Lemmas.

- Im Syntaxbaum B der Ableitung

$$S \vdash_G^* z$$

  gibt es nach obigem Lemma einen Pfad der Länge mindestens $k+1$.

- Fixiere einen solchen Pfad α maximaler Länge.
- Nach Wahl von

$$n = 2^{k+1}$$

  muss es wegen $\|N\| = k$ auf einem solchen Pfad α maximaler Länge ein Nichtterminal A geben, das doppelt vorkommt.

Pumping-Lemma: Beweis

- Denn:
  - Ein Pfad mit Länge $k+1$ hat $k+2$ Knoten, davon darf der letzte ein Blatt sein.
  - Die restlichen $k+1$ Knoten sind mit Nichtterminalen beschriftet.
- Diese beiden Vorkommen von A können so gewählt werden, dass die ersten beiden Eigenschaften des Pumping-Lemmas erfüllt sind.
- Dazu bestimmen wir dieses doppelte Vorkommen von A auf α *von unten nach oben so, dass das obere A höchstens $k+1$ Schritte von der Blattebene entfernt ist*.
- Die Teilbäume unter diesen A induzieren eine Zerlegung von

$$z = uvwxy.$$

Pumping-Lemma: Beweis

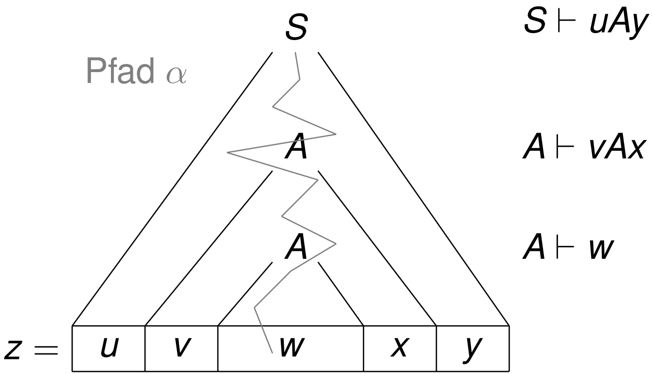


Abbildung: Auf einem Pfad α der Länge mindestens $k + 1$ muss es ein Nichtterminal A geben, das doppelt vorkommt.

Pumping-Lemma: Beweis

Wir verifizieren die drei Aussagen des Pumping-Lemmas.

1. Da G in CNF ist, muss das obere A mittels einer Regel

$$A \to BC$$

weiter abgeleitet werden. Also ist

$$|vx| \geq 1.$$

2. Da das obere A höchstens $k+1$ Schritte von der Blattebene entfernt ist, gilt

$$|vwx| \leq 2^{k+1} = n.$$

Dies folgt wieder aus dem Lemma über Binärbäume:
Hätte der Teilbaum unter dem oberen A mehr als 2^{k+1} Blätter, so gäbe es unter ihm einen Pfad der Länge $> k+1$, Widerspruch.

Pumping-Lemma: Beweis

3. Dies folgt daraus, dass stets ein Wort in L abgeleitet wird, wenn bei der Ableitung von z die Schritte zwischen den beiden Vorkommen von A ($A \vdash vAx$)
   - entweder weggelassen ($i = 0$):
     $S \vdash uAy \vdash uwy$
   - oder beliebig oft wiederholt ($i \geq 1$) werden:
     $S \vdash uAy \vdash uvAxy \vdash uvvAxxy \vdash \dots uv^iAx^iy \vdash uv^iwx^iy$

   Also gilt:

$$(\forall i \geq 0)\,[uv^iwx^iy \in L].$$

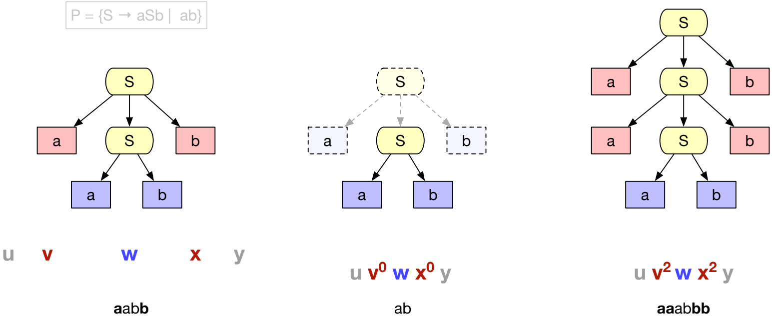
Das Pumping-Lemma für kontextfreie Sprachen ist bewiesen. ❑

Pumping-Lemma: Illustration

Beispiel: $G = (\{a, b\}, \{S\}, S, P)$ mit den Regeln

$$P = \{S \to aSb \mid ab\}$$

Wort $aabb \in L(G)$.

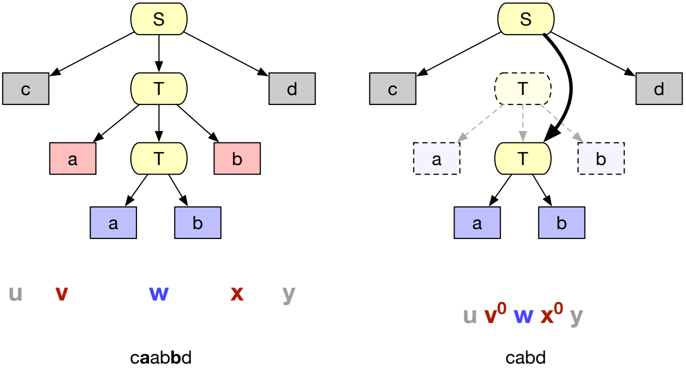


Pumping-Lemma: Illustration

Beispiel: $G = (\{a, b, c, d\}, \{S\}, S, P)$ mit den Regeln

$$P = \{S \to cTd, T \to aTb \mid ab\}$$

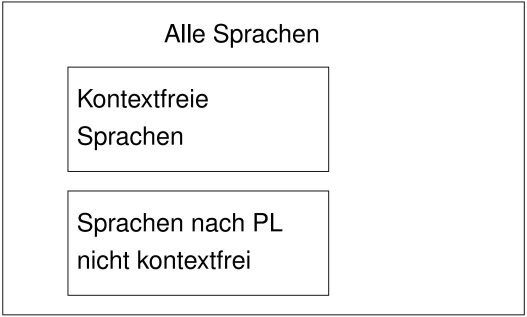
Wort $caabbd \in L(G)$.



Pumping-Lemma

Bemerkung: Man beachte, dass das Pumping-Lemma keine Charakterisierung von CF liefert, sondern lediglich eine Implikation:

$$L \in \mathrm{CF} \quad \Rightarrow \quad (\exists n \geq 1)\,(\forall z \in L, |z| \geq n)\,(\exists u, v, w, x, y \in \Sigma^*)$$
$$[z = uvwxy \wedge (1) \wedge (2) \wedge (3)].$$



Pumping-Lemma: Anwendungen

Behauptung: $L = \{a^m b^m c^m \mid m \geq 1\}$ ist nicht kontextfrei.

Beweis: Der Beweis wird indirekt geführt.

- Angenommen, $L \in \mathrm{CF}$.
- Sei n die Zahl, die nach dem Pumping-Lemma für L existiert.
- Betrachte das Wort

$$z = a^n b^n c^n$$

mit $|z| = 3n > n$.

Pumping-Lemma: Anwendungen

- Da $z \in L$, lässt sich z so in

$$z = uvwxy = a^n b^n c^n$$

zerlegen, dass gilt:

1. $|vwx| \leq n$, d.h., vx kann nicht aus a-, b- *und* c-Symbolen bestehen (vx kann also höchstens zwei der drei Symbole a, b, c enthalten);
2. $|vx| \geq 1$, d.h., $vx \neq \lambda$;
3. $(\forall i \geq 0)\,[uv^i wx^i y \in L]$.

- Insbesondere gilt für $i = 0$:

$$uv^0 wx^0 y = uwy \in L,$$

im Widerspruch zur Definition von L, denn wegen der obigen Eigenschaften kann uwy nicht die Form $a^m b^m c^m$ haben.

- Also ist die Annahme falsch und L nicht kontextfrei. ❑

Pumping-Lemma: Anwendungen

Behauptung: $L = \{0^p \mid p \text{ ist Primzahl}\}$ ist nicht kontextfrei.

Beweis: Der Beweis wird indirekt geführt.

- Angenommen, $L \in \mathrm{CF}$.
- Sei n die Zahl, die nach dem Pumping-Lemma für L existiert und $p \geq n$ eine Primzahl.
- Betrachte das Wort

$$z = 0^p$$

  mit $|z| = p \geq n$.

Pumping-Lemma: Anwendungen

- Da $z \in L$, lässt sich z so in

$$z = uvwxy = 0^p$$

  zerlegen, dass gilt:
  1. $|vwx| \leq n$,
  2. $|vx| \geq 1$ und
  3. $(\forall i \geq 0)\,[uv^i wx^i y \in L]$.
- Insbesondere gilt für $i = p + 1$:

$$\begin{aligned} |uv^{p+1}wx^{p+1}y| &= |uvwxy| + |v^p| + |x^p| = p + p(|v| + |x|) \\ &= p + p(|vx|) = p(1 + |vx|). \end{aligned}$$

  Da $|vx| \geq 1$, ist $|uv^{p+1}wx^{p+1}y|$ keine Primzahl, im Widerspruch zur Definition von L.
- Also ist die Annahme falsch und L nicht kontextfrei. ❑

Pumping-Lemma: Anwendungen

Behauptung: $L = \{0^m \mid m \text{ ist Quadratzahl}\}$ ist nicht kontextfrei.

Beweis: Siehe Übungen. ❑

Abschlusseigenschaften von CF

Theorem

CF *ist abgeschlossen unter*

- *Vereinigung,*
- *Konkatenation,*
- *Iteration und*
- *Spiegelung,*

ist jedoch nicht *abgeschlossen unter*

- *Schnitt,*
- *Komplement und*
- *Differenz.*

Abschlusseigenschaften von CF: Beweis

Beweis: Seien L_1 und L_2 kontextfreie Sprachen und

$$G_i = (\Sigma, N_i, S_i, P_i),$$

wobei $i \in \{1, 2\}$, zwei kontextfreie Grammatiken mit $N_1 \cap N_2 = \emptyset$ und

$$L(G_i) = L_i.$$

Im Folgenden sei $S \notin N_1 \cup N_2$ ein neues Nichtterminal.

- *Vereinigung*: Die Grammatik

  $$G = (\Sigma, N_1 \cup N_2 \cup \{S\}, S, P_1 \cup P_2 \cup \{S \to S_1 \mid S_2\})$$

  leistet

  $$L(G) = L_1 \cup L_2.$$

Abschlusseigenschaften von CF: Beweis

- *Konkatenation*: Die Grammatik

$$G = (\Sigma, N_1 \cup N_2 \cup \{S\}, S, P_1 \cup P_2 \cup \{S \to S_1 S_2\})$$

leistet

$$L(G) = L_1 L_2.$$

- *Iteration*: Die Grammatik

$$G = (\Sigma, N_1 \cup \{S, S'\}, S, P_1 \cup \{S \to \lambda \mid S', S' \to S' S' \mid S_1\}$$

leistet

$$L(G) = (L_1)^*.$$

Abschlusseigenschaften von CF: Beweis

- *Spiegelung:* Es sei $G = (\Sigma, N, S, P)$ eine kontextfreie Grammatik in CNF, d.h., alle Regeln in G sind von der Form

  $$A \to BC \quad \text{oder} \quad A \to a,$$

  wobei $A, B, C \in N$ und $a \in \Sigma$.

  Um die gespiegelten Wörter zu erzeugen, drehen wir die rechten Seiten in Regeln der Form

  $$A \to BC$$

  um.

  Wir erhalten durch $G' = (\Sigma, N, S, P')$ mit

  $$P' = \{A \to CB \mid A \to BC \in P\} \cup \{A \to a \mid A \to a \in P\}$$

  eine kontextfreie Grammatik in CNF mit $L(G') = sp(L(G))$.

Abschlusseigenschaften von CF: Beweis

- *Schnitt*: Die Sprachen

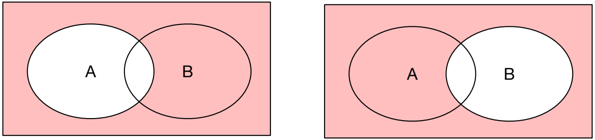
$$A = \{a^i b^i c^j \mid i, j \geq 1\} \quad \text{und} \quad B = \{a^i b^j c^j \mid i, j \geq 1\}$$

sind beide kontextfrei, jedoch (nach dem Pumping-Lemma) nicht ihr Durchschnitt

$$A \cap B = \{a^i b^i c^i \mid i \geq 1\}.$$

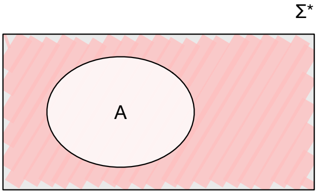
- *Komplement*: folgt nach de Morgan aus dem Abschluss unter Vereinigung und dem Nicht-Abschluss unter Schnitt:

$$A \cap B = \overline{\overline{A} \cup \overline{B}}.$$



Abschlusseigenschaften von CF: Beweis

- *Differenz:* folgt nach $\overline{A} = \Sigma^* - A$ aus dem Nicht-Abschluss unter Komplement. ❑



Abschluss von CF unter Schnittbildung mit REG

Theorem

CF *ist abgeschlossen unter Schnittbildung mit* REG*: Es seien* L_1 *eine kontextfreie Sprache und* L_2 *eine reguläre Sprache. Dann ist*

$$L_1 \cap L_2$$

eine kontextfreie Sprache.

Beweis:

- Es sei $G = (\Sigma, N, S, P)$ eine kontextfreie Grammatik in Chomsky-Normalform mit $L_1 = L(G)$.
- $M = (\Sigma, Z, \delta, z_0, F)$ sei ein DFA mit $Z = \{z_0, \ldots, z_n\}$, so dass $L_2 = L(M)$.

Abschluss von CF unter Schnittbildung mit REG

Beispiel zum Nachdenken: $G = (\{a, b, c\}, \{S, A, B, C, D\}, S, \{S \rightarrow AD \mid CC \mid c, A \rightarrow a, D \rightarrow SB, B \rightarrow b, C \rightarrow CC \mid c\})$ und Automat für regulären Ausdruck a^*cb^*.

Abschluss von CF unter Schnittbildung mit REG

- Wir definieren die Grammatik $\hat{G} = (\Sigma, \hat{N}, \hat{S}, \hat{P})$ mit
  - $\hat{N} = \{A_{ij} \mid \text{für alle } A \in N, 0 \leq i, j \leq n\} \cup \{\hat{S}\}$ und
  - der Regelmenge

$$
\begin{array}{rcll}
\hat{P} & = & \{ & A_{ij} \rightarrow B_{ik} C_{kj} \quad \text{für alle } A \rightarrow BC \in P,\ 0 \leq i, j, k \leq n \\
 & & & A_{ij} \rightarrow a \quad \text{für alle } A \rightarrow a \in P,\ 0 \leq i, j \leq n \\
 & & & \qquad\qquad \text{und } \delta(z_i, a) = z_j \\
 & & & \hat{S} \rightarrow S_{0j} \quad \text{für alle } z_j \in F \quad \}
\end{array}
$$

Abschluss von CF unter Schnittbildung mit REG

$G = (\{a, b, c\}, \{S, A, B, C, D\}, S, \{S \to AD \mid CC \mid c, A \to a, D \to SB, B \to b, C \to CC \mid c\})$ und Automat für regulären Ausdruck a^*cb^*.

$Z = \{z_0, z_1\}$, $\delta(z_0, a) = z_0$, $\delta(z_0, c) = z_1$, $\delta(z_1, b) = z_1$, $F = \{z_1\}$

$$
\begin{aligned}
\hat{P} = \{ & \hat{S} \to S_{01}, \\
& A_{00} \to a,\ B_{11} \to b,\ C_{01} \to c, \\
& S_{01} \to A_{00} D_{01},\ S_{01} \to A_{01} D_{11}, \\
& D_{01} \to S_{00} B_{01},\ D_{01} \to S_{01} B_{11}, \\
& D_{01} \to S_{00} C_{01},\ D_{01} \to S_{01} C_{11}, \\
& S_{00} \to A_{00} D_{00},\ S_{00} \to A_{01} D_{10}, \\
& D_{00} \to S_{00} B_{00},\ D_{00} \to S_{01} B_{10}, \\
& D_{00} \to S_{00} C_{00},\ D_{00} \to S_{01} C_{10},\ \ldots \}
\end{aligned}
$$

Abschluss von CF unter Schnittbildung mit REG

- Es gilt:

$$L(\hat{G}) = L_1 \cap L_2.$$

$\subseteq$ Es sei $x \in L(\hat{G})$, d.h., $\hat{S} \vdash_{\hat{G}} S_{0j} \vdash_{\hat{G}}^{*} x$.
Durch Ignorieren der Indizierung der Nichtterminale in der Ableitung $S_{0j} \vdash_{\hat{G}}^{*} x$, erhalten wir eine Ableitung für x in der Grammatik G, also $x \in L_1$
Die Indizierung impliziert, dass $\widehat{\delta}(z_0, x) \in F$, also

$$x \in L_2.$$

Abschluss von CF unter Schnittbildung mit REG

- zu zeigen: $L(\hat{G}) = L_1 \cap L_2$

  $\supseteq$ Nach Konstruktion von $\hat{P}$.
  ($x \in L_2 \Rightarrow \widehat{\delta}(z_0, x) = z_j \in F \Rightarrow$ es gibt eine Folge von Indizes, so dass eine Ableitung von $S \vdash_G^* x$ in eine Ableitung $\hat{S} \vdash_{\hat{G}} S_{0j} \vdash_{\hat{G}}^* x$ umgewandelt werden kann)

Fertig. ❑

Abschluss von CF unter Schnittbildung mit REG

Lemma

Sei $\Sigma = \{a, b, c\}$ *und* $L' = \{w \in \Sigma^*$ *so dass w gleich viele a, b und c's enthält* $\}$. *Die Sprache* L' *ist nicht kontextfrei.*

Abschluss von CF unter Schnittbildung mit REG

Lemma

Sei $\Sigma = \{a, b, c\}$ und $L' = \{w \in \Sigma^*$ so dass w gleich viele a, b und c's enthält $\}$. Die Sprache L' ist nicht kontextfrei.

Beweis: Wir wissen, dass $L = \{a^m b^m c^m \mid m \geq 1\}$ nicht kontextfrei ist. Deshalb führt die Annahme, dass L' kontextfrei ist zum Widerspruch, da $L = L' \cap \{a^i b^j c^k | i, j, k \geq 1\}$.

Der Algorithmus von Cocke, Younger und Kasami

Ziel: Ein effizienter Algorithmus für das Wortproblem für kontextfreie Grammatiken.

Definition (Wortproblem)

Für $i \in \{0, 1, 2, 3\}$ definieren wir das *Wortproblem für Typ-i-Grammatiken* wie folgt:

$$\mathrm{Wort}_i = \{(G, x) \mid G \text{ ist Typ-}i\text{-Grammatik und } x \in L(G)\}.$$

Der Algorithmus von Cocke, Younger und Kasami

Bemerkung:

1. Später werden wir sehen:
   (a) Wort_0 ist algorithmisch nicht lösbar.

   (b) Wort_1 ist algorithmisch lösbar, allerdings nur mit einem exponentiellen Aufwand.

2. Wort_2 ist sogar effizient lösbar, sofern die entsprechende kfG in Chomsky-Normalform gegeben ist.

3. Der Algorithmus von Cocke, Younger und Kasami beruht auf der Idee des dynamischen Programmierens.

Parsing: Einfaches Beispiel

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

$$R = \{S \to AS \mid b,$$
$$A \to a\}$$

Ist $aab \in L(G)$?

Parsing: Einfaches Beispiel

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

$$R = \{S \to AS \mid b,$$
$$A \to a\}$$

Ist $aab \in L(G)$?

Top-down parsing: wir starten mit S und versuchen eine Ableitung von *aab* zu finden:

$$\begin{aligned} S &\vdash AS & &\vdash aS \\ &\vdash aAS & &\vdash aaS \\ &\vdash aab \end{aligned}$$

Bottom-Up Parsing: Einfaches Beispiel

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

$$R = \{S \to AS \mid b,\quad A \to a\}$$

Ist $aab \in L(G)$?

Bottom-Up parsing: wir starten mit *aab* und versuchen rückwärts zu S zu kommen:

Bottom-Up Parsing: Einfaches Beispiel

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

$$R = \{S \to AS \mid b, \\ A \to a\}$$

Ist $aab \in L(G)$?

Bottom-Up parsing: wir starten mit aab und versuchen rückwärts zu S zu kommen:

$aab \dashv Aab \dashv AAb \dashv AAS \dashv AS \dashv S$.

Parsing: Einfaches Beispiel

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

$$R = \{S \to AS \mid b,$$
$$A \to a\}$$

Der CYK Algorithmus benutzt zum Bottom-Up Parsing eine Tabelle um mit beliebigen Grammatiken in CNF effizient umgehen zu können.

i:	1	2	3
T(i,2)		-	-
T(i,1)			-
T(i,0)			

Wort	a	a	b

Parsing: Einfaches Beispiel

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

$$R = \{S \to AS \mid b,$$
$$A \to a\}$$

Der CYK Algorithmus benutzt eine Tabelle um mit beliebigen Grammatiken in CNF effizient umgehen zu können.

i:	1	2	3
T(i,2)	{S}	-	-
T(i,1)	∅	{S}	-
T(i,0)	{A}	{A}	{S}
Wort	a	a	b

Algorithmenentwurf mit dynamischer Programmierung

1. Charakterisiere den Lösungsraum und die Struktur einer erwünschten optimalen Lösung.

2. Definiere rekursiv, wie sich eine optimale Lösung (und der ihr zugeordnete Wert) aus kleineren optimalen Lösungen (und deren Werten) zusammensetzt. Dabei wird das Bellmansche Optimalitätsprinzip angewandt.

3. Konzipiere den Algorithmus bottom-up so, dass für $n = 1, 2, 3, \ldots$ tabellarisch optimale Teillösungen (und deren Werte) gefunden werden. Beim Finden einer optimalen Teillösung $k > 1$ hilft dabei, dass bereits alle optimalen Teillösungen der Größe $< k$ bereitstehen.

Das Bellmansche Optimalitätsprinzip

Die optimale Lösung eines Problems der Größe n setzt sich aus den optimalen Teillösungen der kleineren Teilprobleme zusammen.

Bemerkung: Auch wenn das Wortproblem eigentlich nicht ein Optimierungsproblem ist, lässt sich dieses Prinzip hier anwenden.

Idee des CYK-Algorithmus

Gegeben seien eine kfG

$$G = (\Sigma, N, S, P)$$

in CNF und ein Wort $x \in \Sigma^*$.

- Hat

  $$x = a \in \Sigma$$

  die Länge 1, so kann es aus einem Nichtterminal A nur abgeleitet werden, indem eine Regel der Form

  $$A \to a$$

  angewandt wird.

Idee des CYK-Algorithmus

- Ist dagegen

$$x = a_1 a_2 \cdots a_n$$

  ein Wort der Länge $n \geq 2$, wobei $a_i \in \Sigma$, so ist x aus A nur ableitbar, weil

  - zunächst eine Regel der Form

    $$A \rightarrow BC$$

    angewandt wurde,

  - aus B dann das Anfangsstück von x und

  - aus C das Endstück von x abgeleitet wurde.

Idee des CYK-Algorithmus

- Es gibt also ein k mit $1 \leq k < n$, so dass gilt: die Regel

$$A \to BC$$

ist in P und

$$B \vdash_G^* a_1 a_2 \cdots a_k \quad \text{und} \quad C \vdash_G^* a_{k+1} a_{k+2} \cdots a_n.$$

- Folglich kann das Wortproblem für ein Wort x der Länge n zurückgeführt werden auf die Lösungen des Wortproblems für zwei kleinere Wörter der Länge k und $n - k$.
- Der Wert von k steht dabei nicht fest, sondern es ist jeder Wert mit $1 \leq k < n$ möglich.

Idee des CYK-Algorithmus

- Mit dynamischer Programmierung
  - beginnen wir also bei der Länge 1 und untersuchen systematisch alle Teilwörter von x auf ihre mögliche Ableitbarkeit aus einem Nichtterminal von G.
  - All diese Information werden in einer Tabelle gespeichert.
  - Wird nun ein Teilwort der Länge $m \leq n$ von x untersucht, so stehen die Teilinformationen über alle kürzeren Teilwörter bereits zur Verfügung.

CYK-Algorithmus mit dynamischer Programmierung

Schritt 1: Lösungsraum und Struktur der Lösung: Sei $G = (\Sigma, N, S, P)$ eine gegebene kfG in CNF, und sei

$$x = a_1 a_2 \cdots a_n$$

ein gegebenes Wort. Der Algorithmus von Cocke, Younger und Kasami baut eine zweidimensionale Tabelle T der Größe $n \times n$ auf, so dass $T(i, j)$ genau die Nichtterminale A von G enthält, so dass

$$A \vdash_G^* a_i a_{i+1} \cdots a_{i+j}$$

gilt. Dabei werden nicht alle Matrixelemente von T benötigt; es genügt eine untere Dreiecksmatrix.

Motivierendes Beispiel

$G = (\{a, b\},\ \{S, A, B\},\ S,\ R)$ mit

$$
\begin{aligned}
R = \{S &\to AS \mid a, \\
A &\to BS, \\
B &\to b\}
\end{aligned}
$$

$$
\begin{aligned}
S &\vdash AS & &\vdash BSS \\
&\vdash bSS & &\vdash bASS \\
&\vdash bBSSS & &\vdash bbSSS \\
&\vdash bbaSS & &\vdash bbaaS \\
&\vdash bbaaa & &
\end{aligned}
$$

Motivierendes Beispiel

$G = (\{a, b\}, \{S, A, B\}, S, R)$ mit $R = \{S \to AS \mid a, A \to BS, B \to b\}$.

$T(i, 0) = \{A \in N \mid A \to a_i \text{ ist Regel in } R\}$

i:	1	2	3	4	5
T(i,1)					
T(i,0)	B	B	S	S	S
Wort	b	b	a	a	a

Motivierendes Beispiel

$G = (\{a, b\}, \ \{S, A, B\}, \ S, \ R)$ mit $R = \{S \to AS \mid a, A \to BS, B \to b\}$.

$T(i,1) = \{A \in N \mid A \to BC \in R \wedge B \in T(i,0) \wedge C \in T(i+1,0)\}$

$A \in T(i,1)$ gdw $A \vdash BC \vdash a_i C \vdash a_i a_{i+1}$ gdw $A \vdash^* a_i a_{i+1}$

i:	1	2	3	4	5
T(i,2)					
T(i,1)		A			
T(i,0)	B	B	S	S	S
Wort	b	b	a	a	a

Motivierendes Beispiel

$G = (\{a, b\},\ \{S, A, B\},\ S,\ R)$ mit $R = \{S \rightarrow AS \mid a,\ A \rightarrow BS,\ B \rightarrow b\}$.

$A \in T(i, j)$ gdw $A \vdash^* a_i a_{i+1} \ldots a_{i+j}$

i:	1	2	3	4	5
T(i,4)					
T(i,3)					
T(i,2)					
T(i,1)		A			
T(i,0)	B	B	S	S	S
Wort	b	b	a	a	a

Motivierendes Beispiel

$G = (\{a, b\}, \{S, A, B\}, S, R)$ mit $R = \{S \to AS \mid a, A \to BS, B \to b\}$.

$A \in T(i, j)$ gdw $A \vdash^* a_i a_{i+1} \ldots a_{i+j}$

$S \in T(1, 4)$ gdw $S \vdash^* a_1 \ldots a_5$ gdw $a_1 \ldots a_5 \in L(G)$

i:	1	2	3	4	5
T(i,4)	S				
T(i,3)	A				
T(i,2)		S			
T(i,1)		A			
T(i,0)	B	B	S	S	S
Wort	b	b	a	a	a

CYK-Algorithmus mit dynamischer Programmierung

Schritt 2: Herleitung der Rekursion:

- Für $1 \leq i \leq n$ ist
$$T(i,0) = \{A \in N \mid A \to a_i \text{ ist Regel in } P\}.$$
- Die weiteren Einträge in die Tabelle werden Zeile für Zeile, von unten nach oben, bestimmt.
Für $j = 1, 2, \ldots, n-1$ enthält $T(i,j)$ genau die Nichtterminale $A \in N$, für die es eine Regel
$$A \to BC$$
in P und ein $k \in \{0, 1, \ldots, j-1\}$ (also in den darunterliegenden Zeilen von T) gibt mit
$$B \in T(i,k) \quad \text{und} \quad C \in T(i+k+1, j-k-1).$$

CYK-Algorithmus mit dynamischer Programmierung

- Inhaltlich heißt das, dass es eine Ableitung

$$A \vdash_G BC$$

gibt sowie darauf folgende Ableitungen

$$B \vdash_G^* a_i a_{i+1} \cdots a_{i+k} \quad \text{und} \quad C \vdash_G^* a_{i+k+1} a_{i+k+2} \cdots a_{i+j},$$

insgesamt also

$$\begin{aligned} A \vdash_G BC \quad & \vdash_G^* \quad a_i a_{i+1} \cdots a_{i+k} C \\ & \vdash_G^* \quad a_i a_{i+1} \cdots a_{i+k} a_{i+k+1} a_{i+k+2} \cdots a_{i+j}. \end{aligned}$$

CYK-Algorithmus mit dynamischer Programmierung

- In der obersten Tabellenposition $T(1, n-1)$ steht nach Definition das Startsymbol S genau dann, wenn gilt

$$S \vdash_G^* a_1 a_2 \cdots a_{1+(n-1)} = a_1 a_2 \cdots a_n = x.$$

Also kann man die Entscheidung, ob $x \in L(G)$, daran ablesen, ob S in der Position

$$T(1, n-1)$$

der Tabelle enthalten ist.

CYK-Algorithmus in Pseudocode (Schritt 3)

$\texttt{CYK}(G,x)$ { // $G = (\Sigma, N, S, P)$ ist kfG in CNF und $x = a_1 a_2 \cdots a_n \in \Sigma^*$

  $\texttt{for}$ $(i = 1, 2, \ldots, n)$ { $T(i,0) = \{A \in N \mid A \to a_i$ ist Regel in $P\}$; }

  $\texttt{for}$ $(j = 1, 2, \ldots, n-1)$ {

    $\texttt{for}$ $(i = 1, 2, \ldots, n-j)$ {

      $T(i,j) = \emptyset$;

      $\texttt{for}$ $(k = 0, 1, \ldots, j-1)$ {

        $T(i,j) = T(i,j) \cup \{A \in N \mid$ es gibt eine Regel $A \to BC$ in P

          und $B \in T(i,k)$ und $C \in T(i+k+1, j-k-1)\}$;

      }

    }

  }

  $\texttt{if}$ $(S \in T(1, n-1))$ $\texttt{return}$ "$x \in L(G)$";

  $\texttt{else return}$ "$x \notin L(G)$";

}

CYK-Algorithmus

Bemerkung:

- Die Komplexität des Algorithmus von Cocke, Younger und Kasami ist wegen der drei ineinander verschachtelten for-Schleifen

$$\mathcal{O}(n^3).$$

CYK-Algorithmus

Bemerkung:

- Die Komplexität des Algorithmus von Cocke, Younger und Kasami ist wegen der drei ineinander verschachtelten for-Schleifen

$$\mathcal{O}(n^3).$$

- Indem man rückverfolgt, *weshalb* das Startsymbol S schließlich in die Tabellenposition $T(1, n-1)$ kommt, erhält man den Syntaxbaum der entsprechenden Ableitung.

CYK-Algorithmus: Beispiel

Beispiel: Gegeben sei die Grammatik G' mit den folgenden Regeln:

$$S \rightarrow ab \mid aSb \mid aSbb.$$

Umformen in CNF ergibt zunächst die Grammatik G'' mit den Regeln:

$$S \rightarrow AB \mid ASB \mid ASBB, \quad A \rightarrow a, \quad B \rightarrow b$$

und schließlich die Grammatik G in CNF mit den Regeln:

$$\begin{array}{lcl} S & \rightarrow & AB \mid AC \mid DE, \\ C & \rightarrow & SB, \\ D & \rightarrow & AS, \\ E & \rightarrow & BB, \\ A & \rightarrow & a, \\ B & \rightarrow & b. \end{array}$$

CYK-Algorithmus: Beispiel

Offenbar ist

$$L(G) = \{a^m b^n \mid 1 \leq m \leq n < 2m\}.$$

Betrachte das Wort

$$x = aaabbbb$$

in Σ^*, wobei $\Sigma = \{a, b\}$. Dieses Wort x gehört zu $L(G)$, und eine Ableitung (eine so genannte Linksableitung) ist:

$$\begin{array}{llllllll} S & \vdash_G & DE & \vdash_G & ASE & \vdash_G & aSE & \vdash_G \quad aACE \\ & \vdash_G & aaCE & \vdash_G & aaSBE & \vdash_G & aaABBE & \\ & \vdash_G & aaaBBE & \vdash_G & aaabBE & \vdash_G & aaabbE & \\ & \vdash_G & aaabbBB & \vdash_G & aaabbbB & \vdash_G & aaabbbb. & \end{array}$$

CYK-Algorithmus: Beispiel

Der Algorithmus von Cocke, Younger und Kasami erzeugt dann Zeile für Zeile, von unten nach oben, die folgende Tabelle:

i	1	2	3	4	5	6	7
6	**S**, C						
5	S, D	C					
4	D	S, C					
3		S					
2		D	C				
1			S	E	E	E	
0	A	A	A	B	B	B	B
j	a	a	a	b	b	b	b

Der Eintrag S an der Position $T(1, n-1)$ signalisiert, dass $x = aaabbbb$ in $L(G)$ ist.


MEMENTO
MANCHMAL IST ES BESSER, ZU VERGESSEN.
GUY PEARCE
CARRIE-ANNE MOSS
JOE PANTOLIANO
MEMENTO

Kellerautomaten

Ziel: Automatenmodell zur Charakterisierung von CF.

Dazu erweitern wir das NFA-Modell um einen Speicher (Keller bzw. Stack), der nach dem Lifo-Prinzip („Last in – first out“) arbeitet.

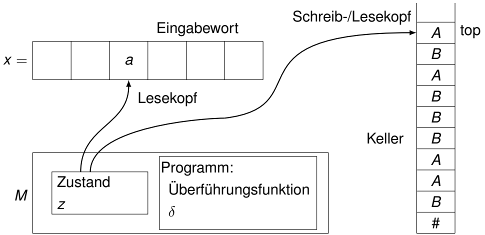


Abbildung: Ein Kellerautomat

Kellerautomaten

Definition (Kellerautomat (PDA))

Ein *(nichtdeterministischer) Kellerautomat* (kurz PDA für *push-down automaton*) ist ein 6-Tupel $M = (\Sigma, \Gamma, Z, \delta, z_0, \#)$, wobei

- Σ das Eingabe-Alphabet ist,
- Γ das Kelleralphabet,
- Z eine endliche Menge von Zuständen,
- $\delta : Z \times (\Sigma \cup \{\lambda\}) \times \Gamma \to \mathfrak{P}_e(Z \times \Gamma^*)$ die Überführungsfunktion ($\mathfrak{P}_e(Z \times \Gamma^*)$ ist die Menge aller *endlichen* Teilmengen von $Z \times \Gamma^*$),
- $z_0 \in Z$ der Startzustand,
- $\# \in \Gamma$ das Bottom-Symbol im Keller.

Kellerautomaten

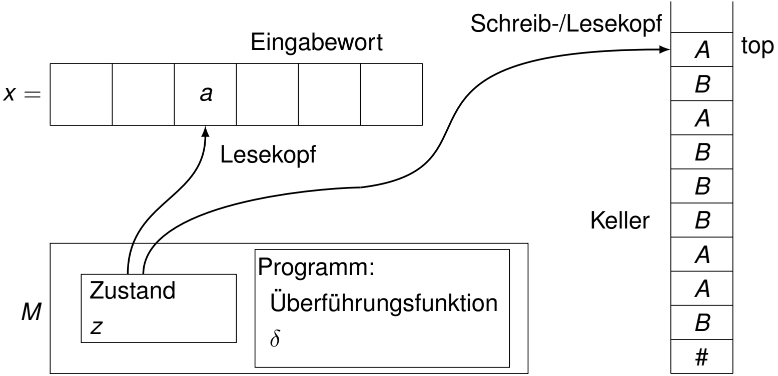


Abbildung: Arbeitsweise eines Kellerautomaten

Kellerautomaten: Arbeitsweise

- δ-Übergänge $(z', B_1 B_2 \cdots B_k) \in \delta(z, a, A)$ schreiben wir auch so:

$$zaA \to z' B_1 B_2 \cdots B_k.$$

  Dies heißt: Wird
  - im Zustand $z \in Z$
  - das Eingabesymbol $a \in \Sigma$ gelesen und
  - ist $A \in \Gamma$ das Top-Symbol im Keller,

  so geht M über
  - in den Zustand z',
  - der Lesekopf rückt ein Feld nach rechts auf dem Eingabeband und
  - das Top-Symbol A im Keller wird ersetzt durch die Kellersymbole

$$B_1 B_2 \cdots B_k,$$

    wobei B_1 das neue Top-Symbol ist.

Kellerautomaten: Arbeitsweise

- POP-Operation: Ist $k = 0$ in

$$zaA \to z' B_1 B_2 \cdots B_k,$$

so wird A aus dem Keller „gePOPt“.

- PUSH-Operation: Ist $k = 2$ und $B_1 B_2 = BA$ in

$$zaA \to z' B_1 B_2 \cdots B_k,$$

so wird B in den Keller „gePUSHt“.

- $z\lambda A \to z' B_1 B_2 \cdots B_k$ heißt dasselbe wie oben, nur dass hier kein Eingabesymbol gelesen wird und der Lesekopf stehen bleibt.

Unterschiede zwischen PDA und NFA

- Akzeptierung erfolgt durch *leeren Keller* statt durch Endzustand (obwohl dies äquivalent auch möglich wäre).

- Takte (d.h. Rechenschritte) ohne Lesen eines Eingabesymbols sind beim PDA mit Regeln der Form

  $$z\lambda A \to z' B_1 B_2 \cdots B_k$$

  möglich. (Für NFAs gibt es auch ein äquivalentes Modell mit λ-Übergängen, so genannte λ-NFAs.)

- Daher ist auch nur ein Startzustand nötig (man kann, wenn gewünscht, von diesem in jeden anderen Zustand gelangen, ohne ein Symbol der Eingabe zu verarbeiten).

Sprache eines Kellerautomaten

Definition

Sei $M = (\Sigma, \Gamma, Z, \delta, z_0, \#)$ ein PDA.

- $\mathfrak{K}_M = Z \times \Sigma^* \times \Gamma^*$ ist die Menge aller *Konfigurationen* von M.

- Ist $k = (z, \alpha, \gamma)$ eine Konfiguration aus $\mathfrak{K}_M$, so ist im aktuellen Takt der Rechnung von M:
  - $z \in Z$ der aktuelle Zustand von M;
  - $\alpha \in \Sigma^*$ der noch zu lesende Teil des Eingabeworts;
  - $\gamma \in \Gamma^*$ der aktuelle Kellerinhalt.

Für jedes Eingabewort $w \in \Sigma^*$ ist $(z_0, w, \#)$ die entsprechende Startkonfiguration von M.

Sprache eines Kellerautomaten

- Auf $\mathfrak{K}_M$ definieren wir wie folgt eine binäre Relation
$$\vdash_M \ \subseteq \mathfrak{K}_M \times \mathfrak{K}_M.$$
- *Intuitiv*: Für $k, k' \in \mathfrak{K}_M$ gilt $k \vdash_M k'$ genau dann, wenn k' aus k durch eine Anwendung von δ hervorgeht.
- *Formal*: Für Zustände $z, z' \in Z$, Symbole $a_1, a_2, \ldots, a_n \in \Sigma$ und Kellersymbole $A_1, A_2, \ldots, A_m, B_1, B_2, \ldots, B_k \in \Gamma$ ist

$$(z, a_1 a_2 \cdots a_n, A_1 A_2 \cdots A_m) \vdash_M \begin{cases} (z', a_2 \cdots a_n, B_1 B_2 \cdots B_k A_2 \cdots A_m) \\ \qquad \text{falls } (z', B_1 B_2 \cdots B_k) \in \delta(z, a_1, A_1) \\ (z', a_1 \cdots a_n, B_1 B_2 \cdots B_k A_2 \cdots A_m) \\ \qquad \text{falls } (z', B_1 B_2 \cdots B_k) \in \delta(z, \lambda, A_1). \end{cases}$$

Sprache eines Kellerautomaten

- Sei $\vdash_M^*$ die reflexive und transitive Hülle von $\vdash_M$, d.h., für $k, k' \in \mathfrak{K}_M$ gilt
$$k \vdash_M^* k'$$
genau dann, wenn k' aus k durch endlich-fache Anwendung von δ hervorgeht.

- Die *vom PDA M akzeptierte Sprache* ist definiert durch
$$L(M) = \{w \in \Sigma^* \mid (z_0, w, \#) \vdash_M^* (z, \lambda, \lambda) \text{ für ein } z \in Z\}.$$
Wir bezeichnen für jedes $z \in Z$ die Konfiguration (z, λ, λ) als Endkonfiguration für den PDA M.

Kellerautomaten: Beispiel

Beispiel: Die Sprache

$$L = \{a^m b^m \mid m \geq 1\}$$

ist kontextfrei. Ein Kellerautomat, der L erkennt, ist definiert durch

$$M = (\{a, b\}, \{A, \#\}, \{z_0, z_1\}, \delta, z_0, \#),$$

wobei die Überführungsfunktion δ durch die folgenden Regeln gegeben ist:

$z_0 a\# \to z_0 A\#$	$z_1 \lambda\# \to z_1 \lambda$
$z_0 aA \to z_0 AA$	$z_1 bA \to z_1 \lambda$
$z_0 bA \to z_1 \lambda$	

Kellerautomaten: Beispiel

- Beispielsweise ist $aabb \in L(M) = L$, denn:

$$
\begin{aligned}
(z_0, aabb, \#) &\vdash_M (z_0, abb, A\#) \\
&\vdash_M (z_0, bb, AA\#) \\
&\vdash_M (z_1, b, A\#) \\
&\vdash_M (z_1, \lambda, \#) \\
&\vdash_M (z_1, \lambda, \lambda)
\end{aligned}
$$

Kellerautomaten: Beispiel

- Andererseits ist $abb \notin L(M) = L$, denn:

$$\begin{aligned}(z_0, abb, \#) &\vdash_M (z_0, bb, A\#)\\ &\vdash_M (z_1, b, \#)\\ &\vdash_M (z_1, b, \lambda)\end{aligned}$$

  und keine weitere Regel ist anwendbar, da der Keller leer ist, also kein Top-Symbol existiert. Da die Eingabe jedoch nicht vollständig verarbeitet wurde, wird nicht akzeptiert.

- Die Überführungsfunktion von M ist sogar „deterministisch“ (die formale Definition kommt später), denn jede Konfiguration hat nur eine mögliche Folgekonfiguration.

Kellerautomaten: Beispiel

Beispiel: Der folgende PDA ist jedoch „echt nichtdeterministisch“:

$$M' = (\{a,b\}, \{A,B,\#\}, \{z_0,z_1\}, \delta, z_0, \#),$$

wobei die Überführungsfunktion δ gegeben ist durch:

$z_0a\#$	$\rightarrow$	$z_0A\#$	z_0aA	$\rightarrow$	$z_1\lambda$
z_0aA	$\rightarrow$	z_0AA	z_0bB	$\rightarrow$	$z_1\lambda$
z_0aB	$\rightarrow$	z_0AB	$z_0\lambda\#$	$\rightarrow$	$z_1\lambda$
$z_0b\#$	$\rightarrow$	$z_0B\#$	z_1aA	$\rightarrow$	$z_1\lambda$
z_0bA	$\rightarrow$	z_0BA	z_1bB	$\rightarrow$	$z_1\lambda$
z_0bB	$\rightarrow$	z_0BB	$z_1\lambda\#$	$\rightarrow$	$z_1\lambda$

Kellerautomaten: Beispiel

Beispiel: Der folgende PDA ist jedoch „echt nichtdeterministisch“:

$$M' = (\{a, b\}, \{A, B, \#\}, \{z_0, z_1\}, \delta, z_0, \#),$$

wobei die Überführungsfunktion δ gegeben ist durch:

$z_0 a\#$	$\rightarrow$	$z_0 A\#$	$z_0 aA$	$\rightarrow$	$z_1 \lambda$
$z_0 aA$	$\rightarrow$	$z_0 AA$	$z_0 bB$	$\rightarrow$	$z_1 \lambda$
$z_0 aB$	$\rightarrow$	$z_0 AB$	$z_0 \lambda\#$	$\rightarrow$	$z_1 \lambda$
$z_0 b\#$	$\rightarrow$	$z_0 B\#$	$z_1 aA$	$\rightarrow$	$z_1 \lambda$
$z_0 bA$	$\rightarrow$	$z_0 BA$	$z_1 bB$	$\rightarrow$	$z_1 \lambda$
$z_0 bB$	$\rightarrow$	$z_0 BB$	$z_1 \lambda\#$	$\rightarrow$	$z_1 \lambda$

Kellerautomaten: Beispiel

- Es gilt

$$L(M') = \{w\, sp(w) \mid w \in \{a,b\}^*\},$$

wobei $sp(w)$ die Spiegelung des Wortes w ist.

- M' hat nichtdeterministische Übergänge:

$$z_0 aA \begin{array}{l} \to \; z_0 AA \\ \\ \to \; z_1 \lambda \end{array} \quad \text{und} \quad z_0 bB \begin{array}{l} \to \; z_0 BB \\ \\ \to \; z_1 \lambda \end{array}$$

- Der Nichtdeterminismus wird hier benötigt, um die Wortmitte zu raten. Es gibt keinen deterministischen PDA, der $L(M')$ erkennt.

Kellerautomaten: Beispiel

- Drei mögliche Folgen von Konfigurationen bei Eingabe *abba* sind:

$$
\begin{array}{lll}
 & \vdash & (z_1, abba, \lambda) \\
(z_0, abba, \#) & \vdash & (z_0, bba, A\#) \vdash (z_0, ba, BA\#) \vdash (z_0, a, BBA\#) \vdash (z_0, \lambda, ABBA\#) \\
 & & \qquad \vdash (z_1, a, A\#) \vdash (z_1, \lambda, \#) \vdash (z_1, \lambda, \lambda)
\end{array}
$$

  - Der obere Pfad führt zur Nicht-Akzeptierung, da das Eingabeband nicht leer, aber ohne Top-Symbol im leeren Keller keine weitere Regel anwendbar ist.
  - Der mittlere Pfad führt zur Nicht-Akzeptierung, da der Keller nicht leer, aber keine weitere Regel anwendbar ist.
  - Der untere Pfad führt zur Akzeptierung. (*M* wechselt nach Abarbeitung der Hälfte des Wortes in den Zustand z_1 und vergleicht den Kellerinhalt zeichenweise mit der zweiten Hälfte der Eingabe.)

Kellerautomaten und kontextfreie Grammatiken

Theorem

L ist kontextfrei $\iff$ es gibt einen Kellerautomaten M mit $L(M) = L$.

Beweis: **(⇒)** Sei $L \in \mathrm{CF}$, und sei $G = (\Sigma, N, S, P)$ eine kfG mit

$$L(G) = L.$$

Idee:

- Wir konstruieren einen Kellerautomaten M mit $L(M) = L$:

$$M = (\Sigma, N \cup \Sigma, \{z\}, \delta, z, S).$$

- Der Kellerautomat M simuliert bei Eingabe x auf dem Keller eine Linksableitung (d.h., es wird stets das am weitesten links stehende Nichtterminal ersetzt) von x in G.

Kellerautomaten und kontextfreie Grammatiken

Die Überführungsfunktion δ ist wie folgt definiert:

- Ist $A \rightarrow q$ eine Regel in P mit $A \in N$ und $q \in (N \cup \Sigma)^*$, so sei
$$(z, q) \in \delta(z, \lambda, A).$$
  Das heißt, lässt sich eine Regel der Grammatik auf das Top-Symbol im Keller anwenden, so tue dies, ohne ein Eingabesymbol zu lesen.
- Für jedes $a \in \Sigma$ sei
$$(z, \lambda) \in \delta(z, a, a).$$
  Das heißt, ist das Top-Symbol im Keller ein Terminalzeichen a, das auch links in der Eingabe steht, so wird a aus dem Keller „gePOPt".

Kellerautomaten und kontextfreie Grammatiken

Beispiel: $G = (\{a, b\},\ \{S\},\ S,\ \{S \to aSb, S \to ab\})$.
Die Überführungsfunktion δ für M = $(\{a, b\}, \{a, b, S\}, \{z\}, \delta, z, S)$ ist daher wie folgt:

- $\delta(z, a, a) = \{(z, \lambda)\}$, $\delta(z, b, b) = \{(z, \lambda)\}$
- $\delta(z, \lambda, S) = \{(z, aSb), (z, ab)\}$

Der PDA ist ein nicht-deterministischer “Top-Down Recursive Descent Parser” für die Grammatik.

Kellerautomaten und kontextfreie Grammatiken

Originalbeweis im Skript:

$L(M) = L(G) = L$ da für alle $x \in \Sigma^*$ gilt:

$$
\begin{aligned}
x \in L(G) \quad & \Longleftrightarrow \quad S \vdash_G^* x \\
& \Longleftrightarrow \quad \text{es gibt eine Folge von Konfigurationen von } M \text{ mit} \\
& \qquad\quad (z, x, S) \vdash_M \cdots \vdash_M (z, \lambda, \lambda) \\
& \Longleftrightarrow \quad (z, x, S) \vdash_M^* (z, \lambda, \lambda) \\
& \Longleftrightarrow \quad x \in L(M).
\end{aligned}
$$

Somit ist: L(M) = L(G) = L.

Kellerautomaten und kontextfreie Grammatiken

Originalbeweis im Skript:

$L(M) = L(G) = L$ da für alle $x \in \Sigma^*$ gilt:

$$
\begin{aligned}
x \in L(G) \quad &\iff \quad S \vdash_G^* x \\
&\iff \quad \text{es gibt eine Folge von Konfigurationen von } M \text{ mit} \\
&\qquad\quad (z, x, S) \vdash_M \cdots \vdash_M (z, \lambda, \lambda) \\
&\iff \quad (z, x, S) \vdash_M^* (z, \lambda, \lambda) \\
&\iff \quad x \in L(M).
\end{aligned}
$$

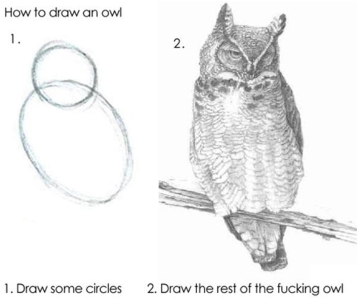
Somit ist: L(M) = L(G) = L.

Dieser Schritt ist nicht ganz so einfach:

$S \vdash_G^* x \iff (z, x, S) \vdash_M \cdots \vdash_M (z, \lambda, \lambda)$

(es gibt zB λ und normale Regeln in M).

- *Writing is nature's way of letting you know how sloppy your thinking is.*
  Dick Guindon
- *Mathematics is nature's way of letting you know how sloppy your writing is. Formal mathematics is nature's way of letting you know how sloppy your mathematics is.*
  Leslie Lamport, Specifying Systems



http://knowyourmeme.com/memes/how-to-draw-an-owl

Kellerautomaten und kontextfreie Grammatiken

Wir wollen ausführlicher beweisen, dass $L(M) = L(G)$. (Im Buch von Hopcroft et al. sind dies über zwei Seiten mit zwei Beweisrichtungen.)

Definition (Linksableitung $\vdash_{lm}$)

Sei $G = (\Sigma, N, S, P)$. Wir haben $vA\gamma \vdash_{lm} v\alpha\gamma$ gdw. $A \to \alpha \in P$, $A \in N$, $v \in \Sigma^*$ und $\gamma \in (\Sigma \cup N)^*$.

Ist G in CNF, und $S \vdash_{lm}^* vA\gamma$ so haben wir $\gamma \in N^*$.

Lemma

$x \in L(G) \iff S \vdash_{lm}^* x.$

Kellerautomaten und kontextfreie Grammatiken

Der Beweis wird etwas einfacher, wenn wir annehmen, dass G in CNF ist.

Lemma

Sei $G = (\Sigma, N, S, P)$ eine Grammatik in CNF und M der konstruierte PDA. Sei $A, B, C \in N$, dann gilt:

$$\begin{aligned} A \vdash_{lm} BC &\iff A \to BC \in P \\ &\iff (z, BC) \in \delta(z, \lambda, A) \\ &\iff (z, w, A) \vdash_M (z, w, BC) \end{aligned}$$

Ein Ableitungsschritt in G mit $A \to BC$ entspricht genau einem Rechenschritt im PDA M. (Anm.: Eingabe $w \in \Sigma^*$ ist beliebig.)

Kellerautomaten und kontextfreie Grammatiken

Der Beweis wird etwas einfacher, wenn wir annehmen, dass G in CNF ist.

Lemma 2 über einen Linkableitungsschritt für G in CNF:

Lemma

Sei $G = (\Sigma, N, S, P)$ eine Grammatik in CNF und M der konstruierte PDA. Sei $A \in N$, $a \in \Sigma$ dann gilt:

$$\begin{aligned} A \vdash_{lm} a \quad &\Longleftrightarrow \quad A \rightarrow a \in R \\ &\Longleftrightarrow \quad (z, a) \in \delta(z, \lambda, A) \wedge (z, \lambda) \in \delta(z, a, a) \\ &\Longleftrightarrow \quad (z, a, A) \vdash_M (z, a, a) \vdash_M (z, \lambda, \lambda) \end{aligned}$$

Ein Ableitungsschritt in G mit $A \rightarrow a$ entspricht genau zwei Rechenschritten im PDA M.

Kellerautomaten und kontextfreie Grammatiken

Lemma

Sei $v \in \Sigma^*$, und $A \in N$, $a \in \Sigma$ dann gilt:

$A \vdash_{lm}^* v \iff (z, v, A) \vdash_M^* (z, \lambda, \lambda)$.

Beweis ($\Rightarrow$) per Induktion über die Anzahl n der Ableitungsschritte $\vdash_{lm}$.

Induktionsanfang $n = 1$. Es muss eine Regel $R \to a$ verwendet werden und wir haben $v = a$.

Induktionsschritt $(n-1) \mapsto n$. $A \vdash_{lm} BC \vdash_{lm}^* \beta\gamma$, wobei $B \vdash_{lm}^* \beta$ und $C \vdash_{lm}^* \gamma$. Per Induktionsannahme haben wir

- $B \vdash_{lm}^* \beta \iff (z, \beta, B) \vdash_M^* (z, \lambda, \lambda) \Rightarrow (z, \beta\sigma, B\rho) \vdash_M^* (z, \sigma, \rho)$
- $C \vdash_{lm}^* \gamma \iff (z, \gamma, A) \vdash_M^* (z, \lambda, \lambda)$

Daraus folgt: $(z, \beta\gamma, A) \vdash_M (z, \beta\gamma, BC) \vdash_M^* (z, \gamma, C) \vdash_M^* (z, \lambda, \lambda)$

Kellerautomaten und kontextfreie Grammatiken

Lemma

Sei $v \in \Sigma^*$, und $A \in N$, $a \in \Sigma$ dann gilt:

$A \vdash_{lm}^* v \iff (z, v, A) \vdash_M^* (z, \lambda, \lambda)$.

Anmerkung: der Beweis in die Rückrichtung ($\Leftarrow$) erfolgt auch per Induktion mit gleicher Zerlegung, ist aber etwas subtiler:
$(z, \beta, B\rho) \vdash_M^* (z, \lambda, \rho) \Leftarrow (z, \beta\sigma, B\rho) \vdash_M^* (z, \sigma, \rho)$ gilt, (d.h., wir können unverbrauchte Eingabe σ einfach entfernen, da Symbole nie auf die Eingabe zurück-gepusht werden) aber
$(z, \beta, B) \vdash_M^* (z, \lambda, \lambda) \Leftarrow (z, \beta, B\rho) \vdash_M^* (z, \lambda, \rho)$ gilt nicht (da man noch λ Regeln auf ρ anwenden kann).

Kellerautomaten und kontextfreie Grammatiken

Lemma

Sei $v \in \Sigma^*$, und $A \in N$, $a \in \Sigma$ dann gilt:
$A \vdash_{lm}^* v \iff (z, v, A) \vdash_M^* (z, \lambda, \lambda)$.

Anmerkung: der Beweis in die Rückrichtung ($\Leftarrow$) ist etwas subtiler: Deshalb verlangen wir, dass bei $(z, \beta\gamma, A) \vdash_M (z, \beta\gamma, BC) \vdash_M \dots (z, \beta_i\gamma, \beta_i'C) \dots \vdash_m (z, \gamma, C) \vdash_M^* (z, \lambda, \lambda)$ folgendes gilt: $\forall i : \beta_i \neq \lambda$, d.h. wir nehmen für γ den ersten Zeitpunkt wo B komplett vom Stack verschwunden ist. Dies garantiert, dass $(z, \beta, B) \vdash_M^* (z, \lambda, \lambda)$.

Aus dem Lemma folgt für $A = S$ dass $L(M) = L(G)$.

Kellerautomaten und kontextfreie Grammatiken

(⇐) Im Beweis der Rückrichtung von

$$\text{L ist kontextfrei} \Leftrightarrow \text{es gibt einen Automaten } M \text{ mit } L(M) = L$$

wird eine kfG

$$G = (\Sigma, N, S, P)$$

aus einem gegebenen PDA

$$M = (\Sigma, \Gamma, Z, \delta, z_0, \#)$$

konstruiert.

O.B.d.A. gelte $k \leq 2$ für alle δ-Regeln der Form

$$zaA \to z' B_1 B_2 \cdots B_k.$$

Kellerautomaten und kontextfreie Grammatiken

Denn gilt etwa $k > 2$ in einer δ-Regel der Form

$$zaA \to z' B_1 B_2 \cdots B_k,$$

dann wähle neue Zustände

$$z_1, z_2, \ldots, z_{k-2}$$

und ersetze diese δ-Regel durch die folgenden neuen δ-Regeln:

$$\begin{array}{rcl} zaA & \to & z_1 B_{k-1} B_k \\ z_1 \lambda B_{k-1} & \to & z_2 B_{k-2} B_{k-1} \\ & \vdots & \\ z_{k-2} \lambda B_2 & \to & z' B_1 B_2. \end{array}$$

Kellerautomaten und kontextfreie Grammatiken

Beispiel: Sei PDA M = $(\{a, b\}, \{S\}, \{z_1\}, \delta, z_1, S)$ mit

- $\delta(z_1, a, S) = \{(z_1, SSS)\}$ (also $z_1 aS \to z_1 SSS$)
- $\delta(z_1, b, S) = \{(z_1, \lambda)\}$ (also $z_1 bS \to z_1 \lambda$)

$(z_1, abbb, S) \vdash_M (z_1, bbb, SSS) \vdash_M (z_1, bb, SS) \vdash_M (z_1, b, S) \vdash_M (z_1, \lambda, \lambda)$

Wie könnte eine Grammatik G aussehen, so dass $\vdash_G$ genau $\vdash_M$ entspricht?

Kellerautomaten und kontextfreie Grammatiken

Beispiel: Sei PDA M = $(\{a, b, c\}, \{S\}, \{z_1, z_2\}, \delta, z_1, S)$ mit

- $\delta(z_1, a, S) = \{(z_1, SS)\}$ (also $z_1 aS \to z_1 SS$)
- $\delta(z_1, b, S) = \{(z_2, \lambda)\}$ (also $z_1 bS \to z_2 \lambda$)
- $\delta(z_2, c, S) = \{(z_2, \lambda)\}$ (also $z_2 cS \to z_2 \lambda$)

$(z_1, abc, S) \vdash_M (z_1, bc, SS) \vdash_M (z_2, c, S) \vdash_M (z_2, \lambda, \lambda)$

Wie könnte eine Grammatik G aussehen, so dass $\vdash_G$ genau $\vdash_M$ entspricht?

Kellerautomaten und kontextfreie Grammatiken

Um die Arbeitsweise des PDA M auf einem Wort x mittels einer Grammatik G zu simulieren, verwenden wir in G Variablen (Nichtterminale), die (bis auf das Startsymbol S) Tripel aus $Z \times \Gamma \times Z$ sind, d.h.,

$$N = \{S\} \cup Z \times \Gamma \times Z.$$

Ist etwa $(z_\ell, \gamma, z_r) \in N$ mit $z_\ell, z_r \in Z$ und $\gamma \in \Gamma$, so ist

- $z_\ell \in Z$ der Zustand vor einer Folge von Rechenschritten des PDA M,
- $\gamma \in \Gamma$ das dabei verarbeitete Kellersymbol,
- $z_r \in Z$ der Zustand, wenn γ aus dem Keller „gePOPt“ wird.

Kellerautomaten und kontextfreie Grammatiken

Aus der Variablen (z_ℓ, γ, z_r) sollen alle Zeichenreihen erzeugt werden können, die bewirken, dass γ vom Stack entfernt wird, während der Automat vom Zustand z_ℓ in den Zustand z_r übergeht.

Das heißt, wir simulieren Zustands- und Kelleränderung von M in den Nichtterminalen von G.

Kellerautomaten und kontextfreie Grammatiken

P besteht aus genau den folgenden Regeln, wobei stets
$z, z', z_1, z_2 \in Z, \quad A, B, C \in \Gamma \quad$ und $\quad a \in \Sigma \cup \{\lambda\}$ gilt:

1. $S \rightarrow (z_0, \#, z)$ für jedes $z \in Z$.
   Vom Startsymbol in G wollen wir alle Nichtterminale erreichen, die bedeuten, dass der PDA M seinen Keller leert, wenn er mit der Startkonfiguration beginnt.
   (Der dabei erreichte Zustand z ist beliebig.)

2. $(z, A, z') \rightarrow a$, falls $(z', \lambda) \in \delta(z, a, A)$.
   Falls der Kellerautomat die Möglichkeit hat, im Zustand z das Zeichen a aus der Eingabe zu lesen, A aus dem Keller zu entfernen und in Zustand z' überzugehen, so realisieren wir dies durch die Regel $(z, A, z') \rightarrow a$ in der Grammatik.

Kellerautomaten und kontextfreie Grammatiken

3. $(z, A, z') \to a(z_1, B, z')$, falls $(z_1, B) \in \delta(z, a, A)$.
   Falls der Kellerautomat die Möglichkeit hat, im Zustand z das Zeichen a aus der Eingabe zu lesen, A aus dem Keller zu entfernen, B auf den Keller zu schreiben und in Zustand z_1 überzugehen, so realisieren wir dies durch die Regeln $(z, A, z') \to a(z_1, B, z')$ in der Grammatik.

4. $(z, A, z') \to a(z_1, B, z_2)(z_2, C, z')$, falls $(z_1, BC) \in \delta(z, a, A)$.
   Falls der Kellerautomat die Möglichkeit hat, im Zustand z das Zeichen a aus der Eingabe zu lesen, A aus dem Keller zu entfernen, BC auf den Keller zu schreiben und in Zustand z_1 überzugehen, so realisieren wir dies durch die Regel $(z, A, z') \to a(z_1, B, z_2)(z_2, C, z')$ in der Grammatik.

Kellerautomaten und kontextfreie Grammatiken

Beispiel: Sei PDA M = $(\{a, b, c\}, \{S\}, \{z_1, z_2\}, \delta, z_1, S)$ mit

- $\delta(z_1, a, S) = \{(z_1, SS)\}$ (also $z_1 aS \rightarrow z_1 SS$)
- $\delta(z_1, b, S) = \{(z_2, \lambda)\}$ (also $z_1 bS \rightarrow z_2\lambda$)
- $\delta(z_2, c, S) = \{(z_2, \lambda)\}$ (also $z_2 cS \rightarrow z_2\lambda$)

$(z_1, abc, S) \vdash_M (z_1, bc, SS) \vdash_M (z_2, c, S) \vdash_M (z_2, \lambda, \lambda)$

Wir haben $G = (\{a, b, c\}, N, \{S\}, P)$ mit $N = \{S\} \cup Z \times \{S\} \times Z$ und P:

- $S \rightarrow (z_1, S, z_1) \mid (z_1, S, z_2)$
- $(z_1, S, z_2) \rightarrow b$
- $(z_2, S, z_2) \rightarrow c$
- $(z_1, S, y) \rightarrow a(z_1, S, x)(x, S, y)$ für $x, y \in \{z_1, z_2\}$ also
  - $(z_1, S, z_1) \rightarrow a(z_1, S, z_1)(z_1, S, z_1) \mid a(z_1, S, z_2)(z_2, S, z_1)$
  - $(z_1, S, z_2) \rightarrow a(z_1, S, z_2)(z_2, S, z_2) \mid a(z_1, S, z_1)(z_1, S, z_2)$

$S \vdash_G (z_1, S, z_2) \vdash_G a(z_1, S, z_2)(z_2, S, z_2) \vdash_G ab(z_2, S, z_2) \vdash_G abc$

Kellerautomaten und kontextfreie Grammatiken

Beispiel: Sei PDA M = $(\{a, b, c\}, \{S\}, \{z_1, z_2\}, \delta, z_1, S)$ mit

- $z_1 aS \to z_1 SS$
- $z_1 bS \to z_2 \lambda$ und $z_2 cS \to z_2 \lambda$

$(z_1, abc, S) \vdash_M (z_1, bc, SS) \vdash_M (z_2, c, S) \vdash_M (z_2, \lambda, \lambda)$

Version von G die besser zu lesen ist: $G = (\{a, b, c\}, N, \{S\}, P)$ mit

$N = \{S, S_{(1,1)}, S_{(1,2)}, S_{(2,1)}, S_{(2,2)}\}$ und P:

- $S \to S_{(1,1)} \mid S_{(1,2)}$
- $S_{(1,2)} \to b$
- $S_{(2,2)} \to c$
- $S_{(1,1)} \to aS_{(1,1)}S_{(1,1)} \mid aS_{(1,2)}S_{(2,1)}$
- $S_{(1,2)} \to aS_{(1,2)}S_{(2,2)} \mid aS_{(1,1)}S_{(1,2)}$

$S \vdash_G S_{(1,2)} \vdash_G aS_{(1,2)}S_{(2,2)} \vdash_G abS_{(2,2)} \vdash_G abc$

Kellerautomaten und kontextfreie Grammatiken

Die Regeln von G sind so beschaffen, dass eine Rechnung von M bei Eingabe x durch eine Linksableitung von x simuliert wird.

Die Korrektheit der Regeln liefert das folgende Lemma:

Lemma

Für alle $(z, A, z') \in N$ und alle $x \in \Sigma^*$ gilt:

$$(z, A, z') \vdash_G^* x \iff (z, x, A) \vdash_M^* (z', \lambda, \lambda).$$

Beweis des Lemmas

(⇐) Der Beweis wird durch Induktion über die Anzahl n der Rechenschritte von M geführt.

<u>**Induktionsanfang:**</u> $\boldsymbol{n = 1}$**.** Es gilt

$$
\begin{array}{rcl}
(z, A, z') \vdash_G a & \iff & (z, A, z') \rightarrow a \text{ ist Regel in } P \\
 & \iff & (z', \lambda) \in \delta(z, a, A) \quad \text{wegen (2)} \\
 & & \text{in der Konstruktion von } P \\
 & \iff & (z, a, A) \vdash_M (z', \lambda, \lambda).
\end{array}
$$

Beweis des Lemmas

<u>**Induktionsschritt**</u>**: $(n-1) \mapsto n$.** Sei $n > 1$, und die Behauptung gelte für $n-1$.

Folglich ist $x = ay$ mit $a \in \Sigma \cup \{\lambda\}$, und es gilt:

$$(z, ay, A) \vdash_M (z_1, y, \alpha) \vdash_M^{n-1} (z', \lambda, \lambda)$$

für einen geeigneten Zustand $z_1 \in Z$ und Kellerinhalt $\alpha \in \{\lambda, B, BC\}$ und der Regel $zaA \to z_1\alpha$ in δ.

Unterscheide die folgenden drei Fälle.

Fall 1: $\alpha = \lambda$. Dieser Fall kann nicht eintreten, da (z_1, y, λ) keine Folgekonfiguration hat.

Beweis des Lemmas

Fall 2: $\alpha = B$. Nach Induktionsvoraussetzung gilt:

$$(z_1, y, \alpha) \vdash_M^{n-1} (z', \lambda, \lambda),$$

was $(z_1, B, z') \vdash_G^* y$ impliziert. Außerdem muss es wegen

$$(z, ay, A) \vdash_M (z_1, y, \alpha)$$

eine δ-Regel der Form $(z_1, B) \in \delta(z, a, A)$ geben.

Nach (3) in der Konstruktion von P gibt es in P eine Regel der Form

$$(z, A, z') \rightarrow a(z_1, B, z').$$

Somit ergibt sich insgesamt:

$$(z, A, z') \vdash_G a(z_1, B, z') \vdash_G^* ay = x,$$

also $(z, A, z') \vdash_G^* x$.

Beweis des Lemmas

Fall 3: $\alpha = BC$. Die Konfigurationenfolge

$$(z_1, y, BC) \vdash_M^* (z', \lambda, \lambda)$$

kann in zwei Teile zerlegt werden:

$$(z_1, y, BC) \vdash_M \ldots (z_i, \alpha_i, \gamma_i C) \ldots \vdash_M (z_2, y_2, C) \vdash_M^* (z', \lambda, \lambda),$$

wobei $y = y_1 y_2$ für ein geeignetes y_1 und $\forall i : \alpha_i \neq \lambda$. Für dieses y_1 gilt:

$$(z_1, y_1, B) \vdash_M^* (z_2, \lambda, \lambda).$$

Außerdem muss es wegen des ersten Schritts

$$(z, ay, A) \vdash_M (z_1, y, BC)$$

eine δ-Regel der Form $(z_1, BC) \in \delta(z, a, A)$ geben.

Beweis des Lemmas

Nach (4) in der Konstruktion von P gibt es in P also eine Regel der Form

$$(z, A, z') \rightarrow a(z_1, B, z_2)(z_2, C, z').$$

Insgesamt ergibt sich:

$$\begin{array}{rcl} (z, A, z') & \vdash_G & a(z_1, B, z_2)(z_2, C, z') \\ & \vdash_G^* & ay_1(z_2, C, z') \\ & \vdash_G^* & ay_1 y_2 = ay = x, \end{array}$$

also $(z, A, z') \vdash_G^* x$.

Beweis des Lemmas

(⇒) Der Beweis wird durch Induktion über k geführt, die Länge der Linksableitung von x.

Induktionsanfang: $k = 1$. Siehe den Induktionsanfang ($n = 1$) im Beweis von (⇐) des Lemmas.

Induktionsschritt: $(k - 1) \mapsto k$. Sei $k > 1$, und die Behauptung gelte für $k - 1$. Wir unterscheiden wieder drei Fälle.

Fall 1: $(z, A, z') \vdash_G a \vdash_G^* x$. Dann gilt $x = a$, im Widerspruch zu $k > 1$.

Beweis des Lemmas

Fall 2: $\boldsymbol{(z, A, z') \vdash_G a(z_1, B, z') \vdash_G^{k-1} ay = x}$**.**

Dann ist $(z_1, B) \in \delta(z, a, A)$.

Nach Induktionsvoraussetzung gilt:

$$(z_1, y, B) \vdash_M^* (z', \lambda, \lambda).$$

Somit ergibt sich:

$$(z, x, A) = (z, ay, A) \vdash_M (z_1, y, B) \vdash_M^* (z', \lambda, \lambda).$$

Beweis des Lemmas

Fall 3: $\boldsymbol{(z, A, z') \vdash_G a(z_1, B, z_2)(z_2, C, z') \vdash_G^{k-1} ay = x.}$
Dann ist $(z_1, BC) \in \delta(z, a, A)$.

Nach Induktionsvoraussetzung gilt für $y = y_1 y_2$:

$$\begin{aligned} (z_1, y_1, B) &\vdash_M^* (z_2, \lambda, \lambda); \\ (z_2, y_2, C) &\vdash_M^* (z', \lambda, \lambda). \end{aligned}$$

Somit ergibt sich:

$$\begin{aligned} (z, x, A) = (z, ay_1 y_2, A) &\vdash_M (z_1, y_1 y_2, BC) \\ &\vdash_M^* (z_2, y_2, C) \\ &\vdash_M^* (z', \lambda, \lambda). \end{aligned}$$

Das Lemma ist bewiesen. ❑

Kellerautomaten und kontextfreie Grammatiken

Aus dem Lemma ergibt sich $L(G) = L(M)$ so:

$$
\begin{array}{rcll}
x \in L(M) & \iff & (z_0, x, \#) \vdash_M^* (z, \lambda, \lambda) \text{ für ein } z \in Z & \\
& \iff & (z_0, \#, z) \vdash_G^* x \text{ für ein } z \in Z, & \text{nach dem Lemma} \\
& \iff & S \vdash_G^* x \quad \text{wegen (1) in der Konstruktion von } P & \\
& \iff & x \in L(G). &
\end{array}
$$

Der Satz ist bewiesen. ❑

Beispiel: kfG $\Rightarrow$ PDA

Beispiel: Wir betrachten die Grammatik

$$G = (\Sigma, N, S, R)$$

aus einem früheren Beispiel mit

- $\Sigma = \{*, +, (,), a\}$,
- $N = \{S\}$ und
- Regeln
$$R = \{S \to S + S \mid S * S \mid (S) \mid a\}.$$

G erzeugt die Sprache aller verschachtelten Klammerausdrücke mit den Operationen $+$ und $*$ und einem Zeichen a.

Beispiel: kfG $\Rightarrow$ PDA

Konstruiere einen PDA M mit $L(M) = L(G)$ wie folgt:

$$M = (\Sigma, N \cup \Sigma, \{z\}, \delta, z, S)$$

mit der folgenden Überführungsfunktion δ:

$z\lambda S \rightarrow zS+S$	$z((\rightarrow z\lambda$
$z\lambda S \rightarrow zS*S$	$z)) \rightarrow z\lambda$
$z\lambda S \rightarrow z(S)$	$z** \rightarrow z\lambda$
$z\lambda S \rightarrow za$	$z++ \rightarrow z\lambda$
	$zaa \rightarrow z\lambda$

Beispiel: kfG $\Rightarrow$ PDA

Eine Ableitung von $w = (a * a) + a$ in G und M:

kfG G			PDA M		
S	$\vdash_G$	$S + S$	$(z, (a * a) + a, S)$	$\vdash_M$	$(z, (a * a) + a, S + S)$
	$\vdash_G$	$(S) + S$		$\vdash_M$	$(z, (a * a) + a, (S) + S)$
				$\vdash_M$	$(z, a * a) + a, S) + S)$
	$\vdash_G$	$(S * S) + S$		$\vdash_M$	$(z, a * a) + a, S * S) + S)$
	$\vdash_G$	$(a * S) + S$		$\vdash_M$	$(z, a * a) + a, a * S) + S)$
				$\vdash_M$	$(z, *a) + a, *S) + S)$
				$\vdash_M$	$(z, a) + a, S) + S)$
	$\vdash_G$	$(a * a) + S$		$\vdash_M$	$(z, a) + a, a) + S)$
		$\dots$			$\dots$

Beispiel: kfG $\Rightarrow$ PDA

kfG G	PDA M
$\dots$	$\dots$
$\vdash_G \quad (a * a) + S$	$\vdash_M \quad (z, a) + a, a) + S)$
	$\vdash_M \quad (z,) + a,) + S)$
	$\vdash_M \quad (z, +a, +S)$
	$\vdash_M \quad (z, a, S)$
$\vdash_G \quad (a * a) + a$	$\vdash_M \quad (z, a, a)$
	$\vdash_M \quad (z, \lambda, \lambda)$

Beispiel: PDA $\Rightarrow$ kfG

Beispiel: Wir betrachten den Kellerautomaten $M = (\Sigma, \Gamma, Z, \delta, z_0, \#)$ mit

- Eingabealphabet $\Sigma = \{a, b\}$,
- Kelleralphabet $\Gamma = \{A, B, \#\}$,
- Zustandsmenge $Z = \{z_0, z_1\}$ und
- Überführungsfunktion δ mit den folgenden Regeln:

$z_0 a\# \to z_0 A\#$	$z_0 aA \to z_1 \lambda$	$z_0 aA \to z_0 AA$	$z_0 bB \to z_1 \lambda$
$z_0 aB \to z_0 AB$	$z_0 \lambda\# \to z_1 \lambda$	$z_0 b\# \to z_0 B\#$	$z_1 aA \to z_1 \lambda$
$z_0 bA \to z_0 BA$	$z_1 bB \to z_1 \lambda$	$z_0 bB \to z_0 BB$	$z_1 \lambda\# \to z_1 \lambda$

Es gilt

$$L(M) = \{w\, sp(w) \mid w \in \{a, b\}^*\}.$$

Beispiel: PDA $\Rightarrow$ kfG

Konstruiere eine Grammatik $G = (\Sigma, \{S\} \cup Z \times \Gamma \times Z, S, P)$ mit $L(G) = L(M)$ wie folgt. P besteht aus genau den folgenden Regeln:

1. $S \rightarrow (z_0, \#, z)$ für jedes $z \in Z$; d.h.,
   - $S \rightarrow (z_0, \#, z_0)$
   - $S \rightarrow (z_0, \#, z_1)$
2. $(z, A, z') \rightarrow a$, falls $(z', \lambda) \in \delta(z, a, A)$; d.h.,
   - $(z_0, A, z_1) \rightarrow a$, da $(z_1, \lambda) \in \delta(z_0, a, A)$
   - $(z_0, B, z_1) \rightarrow b$, da $(z_1, \lambda) \in \delta(z_0, b, B)$
   - $(z_0, \#, z_1) \rightarrow \lambda$, da $(z_1, \lambda) \in \delta(z_0, \lambda, \#)$
   - $(z_1, A, z_1) \rightarrow a$, da $(z_1, \lambda) \in \delta(z_1, a, A)$
   - $(z_1, B, z_1) \rightarrow b$, da $(z_1, \lambda) \in \delta(z_1, b, B)$
   - $(z_1, \#, z_1) \rightarrow \lambda$, da $(z_1, \lambda) \in \delta(z_1, \lambda, \#)$
3. $(z, A, z') \rightarrow a(z_1, B, z')$, falls $(z_1, B) \in \delta(z, a, A)$; d.h., hier keine

Beispiel: PDA $\Rightarrow$ kfG

4. $(z, A, z') \to a(z_1, B, z_2)(z_2, C, z')$, falls $(z_1, BC) \in \delta(z, a, A)$; d.h.,
   - $(z_0, \#, z') \to a(z_0, A, z_2)(z_2, \#, z'), \quad z', z_2 \in \{z_0, z_1\}$,
     da $(z_0, A\#) \in \delta(z_0, a, \#)$
   - $(z_0, A, z') \to a(z_0, A, z_2)(z_2, A, z'), \quad z', z_2 \in \{z_0, z_1\}$,
     da $(z_0, AA) \in \delta(z_0, a, A)$
   - $(z_0, B, z') \to a(z_0, A, z_2)(z_2, B, z'), \quad z', z_2 \in \{z_0, z_1\}$,
     da $(z_0, AB) \in \delta(z_0, a, B)$
   - $(z_0, \#, z') \to b(z_0, B, z_2)(z_2, \#, z'), \quad z', z_2 \in \{z_0, z_1\}$,
     da $(z_0, B\#) \in \delta(z_0, b, \#)$
   - $(z_0, A, z') \to b(z_0, B, z_2)(z_2, A, z'), \quad z', z_2 \in \{z_0, z_1\}$,
     da $(z_0, BA) \in \delta(z_0, b, A)$
   - $(z_0, B, z') \to b(z_0, B, z_2)(z_2, B, z'), \quad z', z_2 \in \{z_0, z_1\}$,
     da $(z_0, BB) \in \delta(z_0, b, B)$

Da $z', z_2 \in \{z_0, z_1\}$, sind hier $6 \cdot 4 = 24$ Regeln angegeben.

Beispiel: PDA $\Rightarrow$ kfG

Eine Ableitung von $w = abba$ in M und G:

PDA M			kfG G		
			S	$\vdash_G$	$(z_0, \#, z_1)$
$(z_0, abba, \#)$	$\vdash_M$	$(z_0, bba, A\#)$		$\vdash_G$	$a(z_0, A, z_1)(z_1, \#, z_1)$
	$\vdash_M$	$(z_0, ba, BA\#)$		$\vdash_G$	$ab(z_0, B, z_1)(z_1, A, z_1)(z_1, \#, z_1)$
	$\vdash_M$	$(z_1, a, A\#)$		$\vdash_G$	$abb(z_1, A, z_1)(z_1, \#, z_1)$
	$\vdash_M$	$(z_1, \lambda, \#)$		$\vdash_G$	$abba(z_1, \#, z_1)$
	$\vdash_M$	(z_1, λ, λ)		$\vdash_G$	$abba$