Kontextfreie Sprachen

Hans U. Simon (RUB)

Email: simon@lmi.rub.de

Homepage: http://www.ruhr–uni–bochum.de/lmi

Bekannte Beispiele

Die Regeln

$$E \to T \mid E + T \ , \ T \to F \mid T * F \ , \ F \to a \mid (E)$$

erzeugen die korrekt geklammerten arithmetischen Ausdrücke.

Die Regeln

$$S \to ab \mid aSb$$

erzeugen die einfache Klammersprache

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

Kontextfreie Regeln und Rekursion

Die Variablen in kontextfreien Regeln repräsentieren rekursiv definierbare Konzepte. Zum Beipiel:

- Lies Regel $E \to T \mid E + T$ wie folgt: Ein Expression ist ein Term oder die Summe aus einem Expression und einem Term.
- Lies Regel $S \to ab \mid aSb$ wie folgt: ein Mitglied der einfachen Klammersprache hat die Form ab oder ist ein von a und b eingeklammertes Mitglied der einfachen Klammersprache.

Kontextfreie Regeln und Programmiersprachen

Programmiersprachen enthalten

- eine Vielzahl rekursiv definierbarer Konzepte
- und diverse Klammerstrukturen

Betrachte zum Beispiel folgendes Fragment einer kontextfreien Grammatik für MODULA:

$$
\begin{aligned}
\langle \text{Anw} \rangle &\rightarrow \langle \text{While–Anw} \rangle \mid \langle \text{If–Anw} \rangle \\
\langle \text{While–Anw} \rangle &\rightarrow \text{WHILE } \langle \text{Bedingung} \rangle \text{ DO } \langle \text{Anw} \rangle \text{ END} \\
\langle \text{If–Anw} \rangle &\rightarrow \text{IF } \langle \text{Bedingung} \rangle \text{ THEN } \langle \text{Anw} \rangle \text{ END} \\
\langle \text{Bedingung} \rangle &\rightarrow \cdots
\end{aligned}
$$

Erläuterungen dazu in der Vorlesung!

Chomsky Normalform (CNF)

Definition: Eine kontextfreie Grammatik ist in Chomsky Normalform, wenn sie nur Regeln der Form

$$A \to BC \ , A \to a \text{ mit } A, B, C \in V, a \in \Sigma$$

besitzt.

Effekt: Binäre Syntaxbäume !

Transformation in Chomsky Normalform

Satz: Jede kontextfreie Grammatik G mit $\varepsilon \notin L(G)$ kann in eine äquivalente kontextfreie Grammatik G' in Chomsky Normalform transformiert werden.

Der Beweis erfolgt in 4 Phasen, die folgendes erreichen:

1. Nur Regeln der Form $A \to a$ dürfen überhaupt auf der rechten Seite ein Zeichen aus Σ verwenden. Die Grammatik heißt dann separiert.
2. Die rechte Seite einer Regel hat maximal die Länge 2.
3. Es gibt keine ε–Regeln mehr.
4. Es gibt keine „Kettenregeln“ (der Form $A \to B$) mehr.

Danach ist die Grammatik in Chomsky Normalform.

Ziel 1: Separierte Grammatik

- Führe für jedes Terminalzeichen a eine neue Variable X_a ein sowie die neuen Regeln $X_a \to a$.
  (Jedes Terminalzeichen bekommt eine „große Schwester“ in V.)
- Bei einer Regel, deren rechte Seite nicht nur aus einem Terminalzeichen bzw. einem String über V besteht, wird jedes Terminalzeichen a durch seine „große Schwester“ $X_a \in V$ ersetzt.

Danach ist die Grammatik separiert (ohne dass die Sprache abgeändert wurde).

Ziel 2: Verkürzung rechter Seiten von Regeln

Ersetze eine Regel R der Form

$$A \to B_1 B_2 \cdots B_k, \ k \geq 3$$

unter Verwendung neuer Variablen $R_1, \ldots, R_{k-2}$ durch das Regelsystem

$$A \to B_1 R_1, \ R_1 \to B_2 R_2, \ \cdots \ , \ R_{k-3} \to B_{k-2} R_{k-2}, \ R_{k-2} \to B_{k-1} B_k \ .$$

Danach hat jede rechte Seite einer Regel maximal die Länge 2 (ohne dass die Sprache abgeändert wurde).

Ziel 3: Elimination von ε–Regeln

- Bestimme nach folgendem Schema die Menge V_ε aller Variablen, aus denen ε ableitbar ist:
  1. Nimm anfangs in V_ε alle Variablen A auf, zu denen eine Regel der Form $A \to \varepsilon$ existiert.
  2. Solange noch Regeln der Form $B \to CD$ mit $C, D \in V_\varepsilon$, $B \notin V_\varepsilon$ bzw. $B \to C$ mit $C \in V_\varepsilon$, $B \notin V_\varepsilon$ existieren, nimm auch B in V_ε auf.
- Für jede Regel der Form $A \to BC$ mache folgendes:
  - Falls $B \in V_\varepsilon$, dann kreiere die zusätzliche Regel $A \to C$.
  - Falls $C \in V_\varepsilon$, dann kreiere die zusätzliche Regel $A \to B$.
- Eliminiere alle ε–Regeln.

Danach enthält die Grammatik keine ε–Regeln mehr (ohne das die Sprache abgeändert wurde).

Ziel 4: Elimination von Kettenregeln

- Betrachte die „Kettenregel–Relation“ K bestehend aus allen Paaren (A, B) zu denen eine Kettenregel $A \to B$ existiert und berechne ihre transitive Hülle K^+.
- Eliminiere alle Kettenregeln.
- Falls $(A, C) \in K^+$, dann kreiere für jede C–Regel $C \to \beta$ die zusätzliche Regel $A \to \beta$.
  A „erbt“ gewissermaßen alle rechten Seiten von C.

Danach enthält die Grammatik keine Kettenregeln mehr. Durch den Trick mit der „Vererbung“ rechter Seiten wurden die Kettenregeln überflüssig gemacht. Die Sprache wird durch die beschriebenen Transformationsschritte nicht abgeändert.

Details zur Berechnung der transitiven Hülle

Bemerkungen:

- $(A, C) \in K^+$ **gdw** A und C sind durch eine „Kette von Kettenregeln“ der Form

  $$A \to B_1 \to B_2 \to \cdots \to B_{k-1} \to B_k \to C$$

  (inklusive des Grenzfalles $A \to C$) miteinander verbunden.

- Berechnung von K^+ entspricht algorithmisch der Berechnung der transitiven Hülle in dem „Kettenregel–Hilfsgraphen“ (Algorithmus von Warshall).

Beispiel

Die kontextfreie Grammatik mit den Regeln

$$S \to aOb \ , \ O \to P \mid OO \mid aOb \ , \ P \to x \mid E \ , \ E \to \varepsilon$$

wird in Chomsky Normalform gebracht wie folgt:

1. Mit Hilfe der neuen Variablen A, B (die „großen Schwestern“ von a, b) erhalten wir die separierte Grammatik

$$S \to AOB \ , \ O \to P \mid OO \mid AOB \ , \ P \to x \mid E \ , \ E \to \varepsilon \ , A \to a \ , \ B \to b \ .$$

2. Mit Hilfe der neuen Variablen S', O' vermeiden wir zu lange rechte Seiten, indem die Regeln $S \to AOB$ bzw. $O \to AOB$ ersetzt werden durch

$$S \to AS' \ , \ S' \to OB \text{ bzw. } O \to AO' \ , \ O' \to OB \ .$$

Beispiel (fortgesetzt)

Zwischenergebnis nach Phase 2:

$$S \to AS' \ , \ S' \to OB \ , \ O \to P \mid OO \mid AO' \ , \ O' \to OB$$

$$P \to x \mid E \ , \ E \to \varepsilon \ , \ A \to a \ , \ B \to b$$

3. $V_\varepsilon = \{E, P, O\}$. Die Technik zur Elimination von ε–Regeln führt zu

$$S \to AS' \ , \ S' \to OB \mid B \ , \ O \to P \mid OO \mid AO' \ , \ O' \to OB \mid B \ ,$$

$$P \to x \mid E \ , A \to a \ , \ B \to b \ .$$

   Dabei wurde die überflüssige Regel $O \to O$ weggelassen.

4. Es gilt $K^+ = \{(S', B), (O, P), (O, O), (O', B), (P, E), (O, E)\}$. Die Technik zur Elimination von Kettenregeln führt zu

$$S \to AS' \ , \ S' \to OB \mid b \ , \ O \to OO \mid AO' \mid x \ , \ O' \to OB \mid b \ ,$$

$$P \to x \ , \ A \to a \ , \ B \to b \ .$$

Greibach Normalform (GNF)

Definition: Eine kontextfreie Grammatik ist in Greibach Normalform, wenn sie nur Regeln der Form

$$A \to aB_1 \cdots B_k \ , \ A \to a \text{ mit } k \geq 1, A, B_1, \ldots, B_k \in V, a \in \Sigma$$

besitzt (einzelnes Terminalzeichen gefolgt von einem String aus Variablen).

Bemerkung:

- Die Einschränkung auf $k = 1$ würde die regulären Grammatiken liefern.
- Greibach Normalform führt später zu Kellerautomaten, die in „Realzeit“ arbeiten (in jedem Schritt ein Eingabesymbol verarbeiten).

Transformation in Greibach Normalform

Satz: Jede kontextfreie Grammatik G mit $\varepsilon \notin L(G)$ kann in eine äquivalente kontextfreie Grammatik G' in Greibach Normalform transformiert werden.

Der Beweis geht **oBdA** davon aus, dass G separiert ist und keine ε–Regeln besitzt (was zum Beispiel bei Chomsky Normalform der Fall ist). Er besteht aus vier Teilen:

1. Technik zur Vermeidung von „Linksrekursion“
2. Erzwingen einer „Monotonie–Eigenschaft“ unter Verwendung von Hilfsvariablen
3. Transformation in Greibach–Normalform abgesehen von B–Regeln für eine Hilfsvariable B
4. Transformation der B–Regeln in Greibach Normalform für jede Hilfsvariable B

Eine Invarianzbedingung

Im Beweis wird das Regelsystem P dynamisch modifiziert werden. Dabei wird es jedoch stets nur Regeln enthalten, deren rechte Seiten

- zu V^+ gehören (String aus Variablen)
- oder der GNF genügen (einzelnes Terminalzeichen gefolgt von einem String aus Variablen).

Bemerkungen:

1. Anfangs ist dies der Fall, da wir von einer separierten kontextfreien Grammatik ohne ε–Regeln ausgehen.
2. Dass die Eigenschaft erhalten bleibt, könnte im Prinzip induktiv gezeigt werden.

Es foglt eine Beschreibung der vier Beweisteile.

Vermeidung von Linksrekursion

Die A–Regeln

$$A \to \overbrace{A\alpha_1 \mid \cdots \mid A\alpha_k}^{linksrekursiv} \mid \beta_1 \mid \cdots \mid \beta_l \text{ mit } \alpha_i \neq \varepsilon$$

erzeugen gerade die durch folgenden regulären Ausdruck beschreibbaren Satzformen:

$$(\beta_1|\cdots|\beta_l)\cdot(\alpha_1|\cdots|\alpha_k)^*$$

Diese Satzformen lassen sich genau so gut (ohne Linksrekursion) erzeugen durch

$$\begin{aligned} A &\to \beta_1 \mid \cdots \mid \beta_l \mid \beta_1 B \mid \cdots \mid \beta_l B \\ B &\to \alpha_1 \mid \cdots \mid \alpha_k \mid \alpha_1 B \mid \cdots \mid \alpha_k B \end{aligned}$$

B ist dabei eine neue Variable.

Erzwingen einer Monotonie–Eigenschaft

Ziel:

Für $V = \{A_1, \ldots, A_m\}$ und Regeln der Form $A_i \to A_j\alpha$ erzwinge, dass $i < j$.

Dies leistet folgende „Bootstrapping“–Methode:

FOR $i := 1$ TO m DO
    FOR $j := 1$ TO $i - 1$ DO
        FOR all $A_i \to A_j\alpha \in P$ DO
            Seien $A_j \to \beta_1 | \cdots | \beta_k$ alle A_j–Regeln
            Streiche $A_i \to A_j\alpha$ aus P
            Füge $A_i \to \beta_1\alpha | \cdots | \beta_k\alpha$ hinzu
        ENDFOR
    ENDFOR
    Vermeide linksrekursive Regeln der Form $A_i \to A_i\alpha$ (sofern vorhanden)
    mit der besprochenen Technik unter Einsatz der Hilfsvariable B_i
ENDFOR

Greibach Normalform für A_i–Regeln

Die Monotonie–Eigenschaft impliziert, dass die rechten Seiten der A_m–Regeln mit einem Terminalzeichen beginnen. Folgende Methode erreicht diese Eigenschaft iterativ auch für die Variablen $A_{m-1}, \ldots, A_1$:

FOR $i := m-1$ DOWNTO 1 DO
    FOR $j := i+1$ TO m DO
        FOR all $A_i \to A_j\alpha \in P$ DO
            Seien $A_j \to \beta_1 | \cdots | \beta_k$ alle A_j–Regeln
            Streiche $A_i \to A_j\alpha$ aus P
            Füge $A_i \to \beta_1\alpha | \cdots | \beta_k\alpha$ hinzu
        ENDFOR
    ENDFOR
ENDFOR

Greibach Normalform für B_i–Regeln

Die rechte Seite einer B_i–Regel beginnt

- mit einem Terminalzeichen (gut)
- oder einer Variable A_j (weniger gut)

Da alle A_j–Regeln bereits der Greibach–Normalform genügen, ist die folgende Maßnahme nun ausreichend:

FOR $i := 1$ TO m DO
    FOR $j := 1$ TO m DO
        FOR all $B_i \to A_j\alpha \in P$ DO
            Seien $A_j \to \beta_1 | \cdots | \beta_k$ alle A_j–Regeln
            Streiche $B_i \to A_j\alpha$ aus P
            Füge $B_i \to \beta_1\alpha | \cdots | \beta_k\alpha$ hinzu
        ENDFOR
    ENDFOR
ENDFOR

Beispiel

Aufgabe: Bringe die durch

$$A_1 \to A_2A_3 \ , \ A_2 \to A_3A_1 \mid b \ , \ A_3 \to A_1A_2 \mid a$$

gegebene Grammatik in Greibach Normalform.

Beim Erzwingen der Monotonie–Eigenschaft müsssen nur die A_3–Regeln modifiziert werden. Es entstehen folgende Zwischenergebnisse:

$$\begin{aligned}
&(1)\ A_3 \ \to \ A_2A_3A_2 \mid a \\
&(2)\ A_3 \ \to \ A_3A_1A_3A_2 \mid bA_3A_2 \mid a \\
&(3)\ A_3 \ \to \ bA_3A_2 \mid a \mid bA_3A_2B_3 \mid aB_3 \\
&(3)\ B_3 \ \to \ A_1A_3A_2 \mid A_1A_3A_2B_3
\end{aligned}$$

Beim Übergang von (2) nach (3) wurde die Technik zur Vermeidung von Linksrekursion eingesetzt.

Beispiel (fortgesetzt)

$$
\begin{aligned}
A_1 &\rightarrow A_2A_3 \\
A_2 &\rightarrow A_3A_1 \mid b \\
A_3 &\rightarrow bA_3A_2 \mid a \mid bA_3A_2B_3 \mid aB_3 \\
B_3 &\rightarrow A_1A_3A_2 \mid A_1A_3A_2B_3
\end{aligned}
$$

hat bereits A_3–Regeln in GNF. Die Technik zur Vererbung dieser Eigenschaft auf die Variablen A_2, A_1 liefert:

$$
\begin{aligned}
A_3 &\rightarrow bA_3A_2 \mid a \mid bA_3A_2B_3 \mid aB_3 \\
A_2 &\rightarrow bA_3A_2A_1 \mid aA_1 \mid bA_3A_2B_3A_1 \mid aB_3A_1 \mid b \\
A_1 &\rightarrow bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3A_2B_3A_1A_3 \mid aB_3A_1A_3 \mid bA_3 \\
B_3 &\rightarrow A_1A_3A_2 \mid A_1A_3A_2B_3
\end{aligned}
$$

Beispiel (fortgesetzt)

$$
\begin{aligned}
A_3 &\rightarrow bA_3A_2 \mid a \mid bA_3A_2B_3 \mid aB_3 \\
A_2 &\rightarrow bA_3A_2A_1 \mid aA_1 \mid bA_3A_2B_3A_1 \mid aB_3A_1 \mid b \\
A_1 &\rightarrow bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3A_2B_3A_1A_3 \mid aB_3A_1A_3 \mid bA_3 \\
B_3 &\rightarrow A_1A_3A_2 \mid A_1A_3A_2B_3
\end{aligned}
$$

hat bereits A_i–Regeln in GNF. Die Technik zur Vererbung dieser Eigenschaft auf die B_i–Regeln liefert dieselben A_i–Regeln und die folgenden B_3–Regeln:

$$
\begin{aligned}
B_3 \rightarrow\ & bA_3A_2A_1A_3A_3A_2 \mid aA_1A_3A_3A_2 \mid bA_3A_2B_3A_1A_3A_3A_2 \\
& \mid aB_3A_1A_3A_3A_2 \mid bA_3A_3A_2 \\
& \mid bA_3A_2A_1A_3A_3A_2B_3 \mid aA_1A_3A_3A_2B_3 \mid bA_3A_2B_3A_1A_3A_3A_2B_3 \\
& \mid aB_3A_1A_3A_3A_2B_3 \mid bA_3A_3A_2B_3
\end{aligned}
$$

Das Pumping–Lemma (uvwxy–Theorem)

Satz: Für jede kontextfreie Sprache $L \subseteq \Sigma^*$ gilt:

$$\exists n \geq 1,$$
$$\forall z \in L \text{ mit } |z| \geq n,$$
$$\exists u, v, w, x, y \in \Sigma^* \text{ mit } z = uvwxy, 1 \leq |vx| \leq |vwx| \leq n, \quad .$$
$$\forall i \geq 0 \ :$$
$$uv^i wx^i y \in L$$

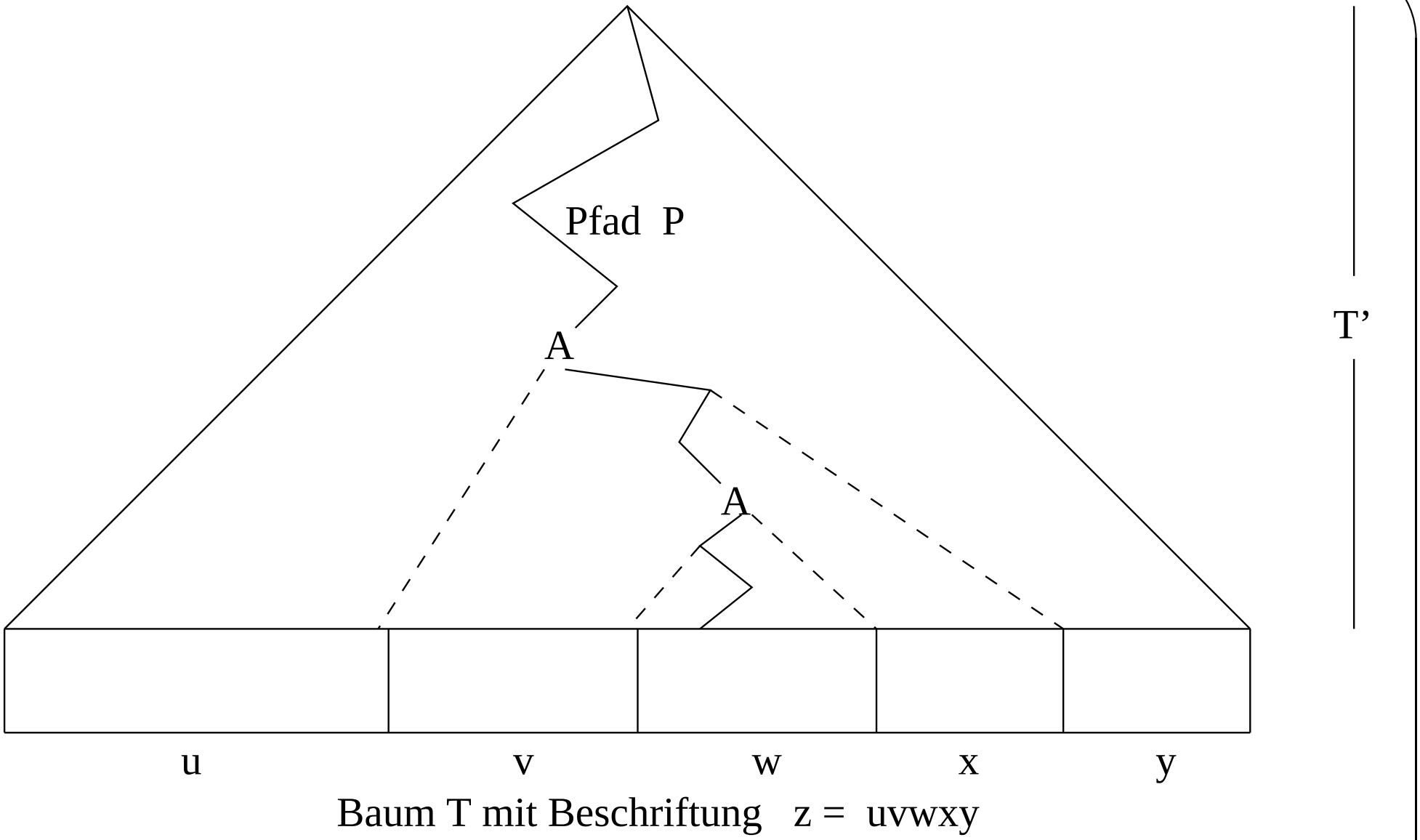
Es folgt:

- der formale Beweis
- plus eine Veranschaulichung

- Benutze eine kontextfreie Grammatik G für $L\setminus\{\varepsilon\}$ in Chomsky Normalform und wähle $k := |V|$ (Anzahl der Variablen) und $n := 2^k$.
- Zu einem Wort $z = a_1 \cdots a_s \in L$ mit $a_i \in \Sigma$ und $s \geq n$ betrachte den Syntaxbaum T mit Beschriftung z und den binären Teilbaum T', der von den inneren Knoten von T induziert wird.
- Da T' (wie auch T) $s \geq 2^k$ Blätter hat, folgt $h := (\text{Höhe von } T') \geq k$.
- Betrachte einen Pfad P der Länge $h \geq k$ (also mit $h+1 \geq k+1$ Knoten) in T'. Gemäß Schubfachprinzip sind zwei der $k+1$ letzten Knoten von P mit derselben Variable, sagen wir A, markiert.
- Die Ableitung von z aus dem Startsymbol S hat demnach die folgende Form:

$$S \Rightarrow_G^* \overbrace{uAy \Rightarrow_G^* uvAxy}^{zyklischer\ Teil} \Rightarrow_G^* uvwxy = z$$

  für geeignet gewählte Wörter u, v, w, x, y mit $1 \leq |vx| \leq |vwx| \leq n$.
- Durch i–fache Iteration (oder Weglassen) des zyklischen Ableitungsteils sehen wir, dass $uv^iwx^iy \in L$ für alle $i \geq 0$.



Baum T mit Beschriftung z = uvwxy

Anwendung: Nachweis der Nichtkontextfreiheit

Folgerung: Falls für eine Sprache $L \subseteq \Sigma^*$ die Bedingung

$\forall n \geq 1$,

$\exists z \in L$ mit $|z| \geq n$,

$\forall u, v, w, x, y \in \Sigma^*$ mit $z = uvwxy, 1 \leq |vx| \leq |vwx| \leq n$,

$\exists i \geq 0$:

$uv^i wx^i y \notin L$

erfüllt ist, dann ist L nicht kontextfrei.

Beispiel für eine nicht–kontexfreie Sprache

Die folgende Sprache $L \subseteq \{a, b\}^*$ ist nicht kontexfrei:

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

Begründung:

1. Zu beliebig vorgegebenem $n \geq 1$ wähle $z = a^n b^n c^n$.
   Offensichtlich gilt $z \in L$ und $|z| \geq n$.
2. Zu beliebig vorgegebener Zerlegung $z = uvwxy$ mit $1 \leq |vx| \leq |vwx| \leq n$ wähle $i = 0$. Beachte, dass vx wegen $|vwx| \leq n$ das Zeichen a oder das Zeichen c nicht enthält.

Nun gilt

$$z' = uv^0wx^0y = uwy \notin L \ ,$$

da die notwendige Bedingung

$$|z'|_a = |z'|_b = |z'|_c$$

nicht erfüllt sein kann.

Der Fall unärer Sprachen

Vereinfachte Form des Pumping Lemmas: Für jede kontextfreie Sprache $L \subseteq \{0\}^*$ gilt:

$$\begin{aligned}
&\exists n \geq 1, \\
&\forall m \geq n \text{ mit } 0^m \in L \\
&\exists 1 \leq l \leq n \qquad\qquad . \\
&\forall i \geq 0 \ : \\
&0^{m-l}0^{il} \in L
\end{aligned}$$

denn: Für die Zerlegung $0^m = uvwxy$ aus dem Pumping Lemma wähle $l := |vx|$ so dass $vx = 0^l$.

Der Fall unärer Sprachen (fortgesetzt)

Satz: Jede kontextfreie Sprache über einem einelementigen Alphabet ist sogar regulär.

Folgerung: Die folgenden Sprachen sind nicht kontextfrei:

$$L = \{0^p | \ p \text{ ist Primzahl}\}$$

$$L = \{0^m | \ m \text{ ist Quadratzahl}\}$$

Beweis des Satzes

- Wähle n als die betreffende Konstante aus dem Pumping Lemma.
- $l \geq 1$ heißt „Pumpindex“ **gdw** $\exists m \geq n, \forall i \geq 0 : 0^{m-l}0^{il} \in L$.
- $I \subseteq \{1, \ldots, n\}$ bezeichne die Menge aller Pumpindizes.
- Die Zahl $q := n! \geq n$ ist dann ein Vielfaches aller Pumpindizes.
- r heißt „Pumprest“ **gdw** $\exists m \geq q : 0^m \in L$ und $r = m \bmod q$.
  m_r bezeichne das kleinste m mit dieser Eigenschaft.
- $R \subseteq \{0, \ldots, q-1\}$ bezeichne die Menge aller Pumpreste.
- Die Sprache

$$L' := \overbrace{\{x \in L |\ |x| < q\}}^{endlich} \cup \bigcup_{r \in R} \overbrace{\{0^{m_r+iq} |\ i \geq 0\}}^{0^{m_r}(0^q)^*}$$

  ist regulär und es gilt $L' = L$.

Der CYK–Algorithmus

Das spezielle Wortproblem für eine kontextfreie Sprache L (mit $\varepsilon \notin L$):

Eingabe: $w = a_1 \cdots a_n$ mit $a_i \in \Sigma$

Frage: $w \in L$?

Der Algorithmus von Cocke, Younger und Kasami benutzt eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ in Chomsky Normalform für L und die Methode des dynamischen Programmierens zur Berechnung einer Tabelle T:

1. Für $1 \leq i, j \leq n$ berechne

$$T[i,j] := \{A \in V |\ A \Rightarrow_G^* a_i a_{i+1} \cdots a_{i+j-1}\} \ .$$

2. Akzeptiere w **gdw** $S \in T[1,n]$.

Wegen

$$S \in T[1,n] \textbf{ gdw } S \Rightarrow_G^* a_1 a_2 \cdots a_n = w$$

akzeptiert der Algorithmus exakt die Eingaben aus L.

Berechnung der Tabelle

- Das Teilwort

$$w_{i,j} := a_i a_{i+1} \cdots a_{i+j-1}$$

  ist das ab Position i beginnende Teilwort von w der Länge j.

- $T[i,j]$ soll die Menge aller Variablen sein, aus denen das Teilwort $w_{i,j}$ ableitbar ist.

- Die Ableitung eines Teilwortes a_i der Länge 1 aus einer Variablen A ist nur über eine Regel der Form $A \to a_i$ möglich ist.

- Die Ableitung eines Teilwortes $w_{i,j} = a_i \cdots a_{i+j-1}$ für $j \geq 2$ hat wegen der binären Syntaxbäume eine spezielle Form (s. folgende Illustration).

Berechnung der Tabelle (fortgesetzt)

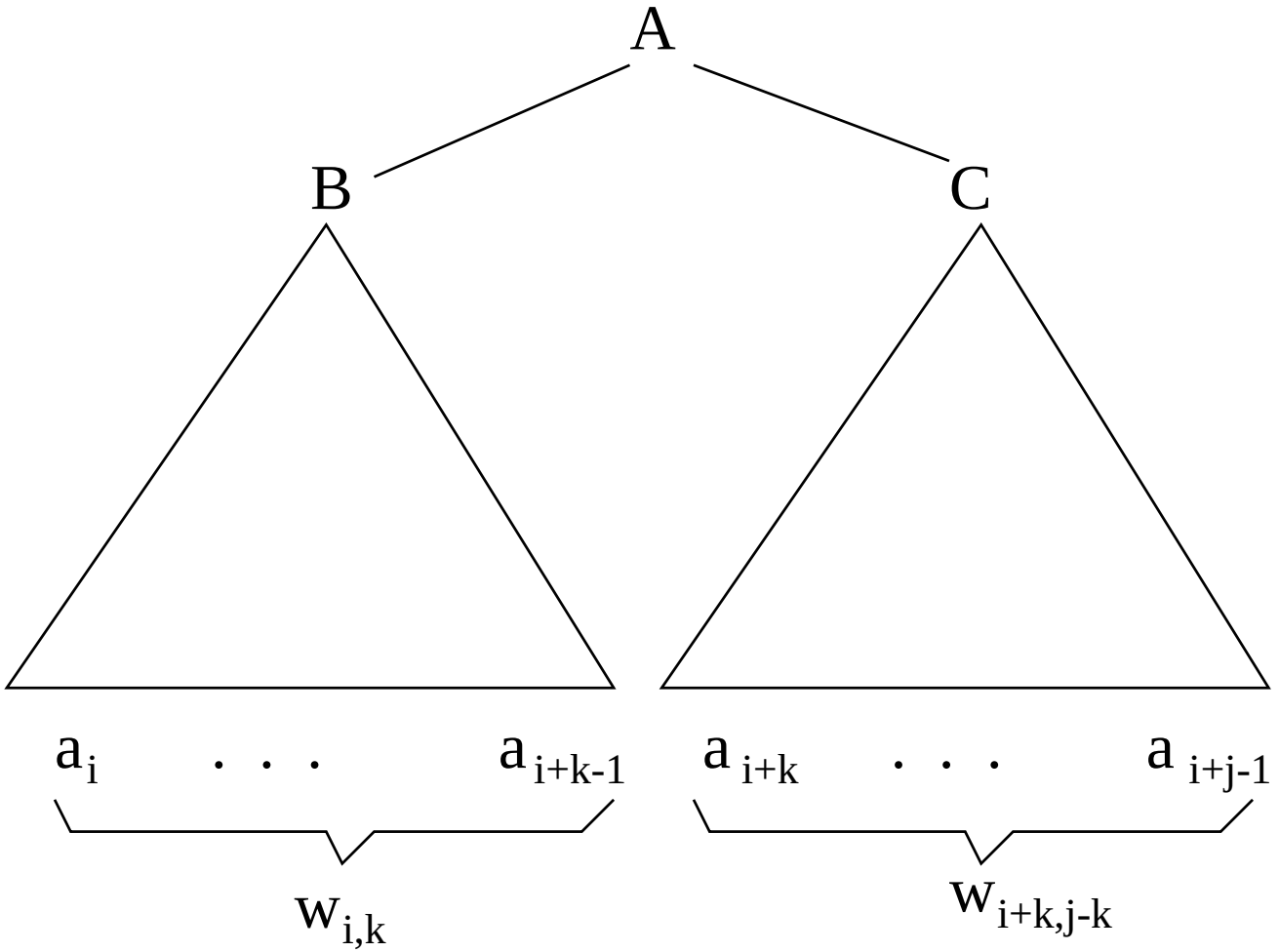


Abbildung 1: Syntaxbaum zur Ableitung $A \Rightarrow_G^* w_{i,j}$.

Berechnung der Tabelle (fortgesetzt)

Tabelle T kann spaltenweise nach folgendem Schema berechnet werden:

$$
\begin{aligned}
T[i,1] &= \{X \in V | \quad X \to a_i \in P\} \\
T[i,j] &= \{X \in V | \quad \exists 1 \le k \le j-1, \exists X \to BC \in P : \\
&\qquad\qquad B \Rightarrow_G^* w_{i,k} \ , \ C \Rightarrow_G^* w_{i+k,j-k}\} \\
&= \{X \in V | \quad \exists 1 \le k \le j-1, \exists X \to BC \in P : \\
&\qquad\qquad B \in T[i,k] \ , \ C \in T[i+k, j-k]\}
\end{aligned}
$$

berechnet werden.

Beachte, dass die Tabelleneinträge $T[i,k]$ und $T[i+k, j-k]$ zum Zeitpunkt der Berechnung von $T[i,j]$ schon bekannt sind.

Beispiel

Die Sprache $L = \{a^n b^n c^m | \ n, m \geq 1\}$ wird durch folgende kontextfreie Grammatik mit Startvariable S generiert:

$$S \to AB \ , \ A \to ab \mid aAb \ , \ B \to c \mid cB$$

Umformen in Chomsky Normalform ergibt:

$$S \to AB \ , \ A \to CD \mid CF \ , \ B \to c \mid EB \ , \ C \to a \ , \ D \to b \ , \ E \to c \ , \ F \to AD$$

Beispiel (fortgesetzt)

$S \to AB$, $A \to CD \mid CF$, $B \to c \mid EB$, $C \to a$, $D \to b$, $E \to c$, $F \to AD$

Eingabe *aaabbbcc* führt zu folgender Tabelle:

$i_{\downarrow} \mid j \to$	1	2	3	4	5	6	7	8
a	C					A	S	S
a	C			A	F			-
a	C	A	F				-	-
b	D					-	-	-
b	D				-	-	-	-
b	D			-	-	-	-	-
c	E,B	B	-	-	-	-	-	-
c	E,B	-	-	-	-	-	-	-

Statischer versus dynamischer Speicher

Statischer Speicher: ein im „Programm“ festgelegter Speicherblock konstanter Größe; kann nicht dynamisch (zur Laufzeit) und damit auch nicht in Abhängigkeit von der Länge der Eingabe wachsen.

Dynamischer Speicher: kann dynamisch (zur Laufzeit) wachsen und daher auch von der Länge der Eingabe abhängig sein.

- Endliche Automaten verfügen (in Form ihrer Zustandsmenge) nur über einen statischen Speicher.
- Ihre beschränkte Potenz (zum Beispiel die Unfähigkeit Sprachen wie $\{a^n b^n | \ n \geq 1\}$ zu erkennen) ist diesem Umstand geschuldet.

Kellerspeicher

Eine eingeschränkte aber durchaus wirkungsvolle Form des dynamischen Speichers ist der Kellerspeicher (auch einfach „Keller“ genannt);

- In einem Keller können Symbole „gestapelt“ werden.
- Nur oben auf dem Stapel darf ein Symbol draufgelegt (Schreib– bzw. Push–Operation) oder entfernt werden (Lösch– bzw. Pop–Operation).
- Der Stapel darf dynamisch (theoretisch unbegrenzt) wachsen.

Ein Lob der „Hochstapelei“

Das Erkennen der nicht–regulären Sprache

$$\{a^n b^n | \; n \geq 1\}$$

ist unter Verwendung eines Kellerspeichers babyleicht:

Push–Phase: Solange a's gelesen werden, lege diese auf den Stapel

Pop–Phase: Beim Lesen des ersten b's schalte von der Push– auf die Pop–Phase um: pro gelesenem b entferne von dem Stapel ein a.

Akzeptieren/Verwerfen: Akzeptiere **gdw** die Eingabe die Form a^+b^+ hat (was mit der „endlichen Kontrolle“ so nebenbei getestet wird) und der Stapel nach Verarbeitung der Eingabe leer ist.

Beispiel

Bei Eingabe

$$aaaabbbb$$

verändert sich der Stapel wie folgt:

Eingabe	a	a	a	a	b	b	b	b
				a				
			a	a	a			
		a	a	a	a	a		
Stapel	a	a	a	a	a	a	a	

Kellerautomat (informelle Definition)

Ein (nichtdeterministischer) Kellerautomat (englisch: pushdown automaton), kurz PDA ist ein um einen Kellerspeicher erweiterter NFA.

Die formale Definition wird zudem folgende technische Details berücksichtigen:

- Neben dem Eingabealphabet gibt es ein (zum Eingabealphabet nicht notwendig disjunktes) Kelleralphabet (inklusive einem „untersten Kellerzeichen“).
- Ein PDA darf „ε–Transitionen“ vollziehen, in denen kein weiteres Eingabesymbol verarbeitet wird.
- Es gibt nur einen Startzustand (wegen der ε–Transitionen kein echtes „handicap“).
- Die Eingabe wird „mit leerem Keller“ (wie in dem eben gezeigten illustrierenden Beipiel) akzeptiert. (Endzustände sind daher überflüssig.)

Kellerautomat (formale Definition)

$\mathcal{P}_e(U)$ bezeichnet die Menge aller endlichen Teilmengen einer (evtl. unendlichen) Menge U.

Ein PDA M besteht aus den folgenden Komponenten:

- Z, die Zustandsmenge (eine endliche Menge)
- Σ, das Eingabealphabet (ebenfalls endlich)
- Γ, das Kelleralphabet (ebenfalls endlich)
- $\delta : Z \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \mathcal{P}_e(Z \times \Gamma^*)$, die Überführungsfunktion
- $z_0 \in Z$, der Startzustand
- $\# \in \Gamma$, das unterste Kellerzeichen

Konvention: Wir werden des öfteren (ohne dies explizit immer wieder zu erwähnen und hauptsächlich zur Vermeidung lästiger Fallunterscheidungen) mit a ein Element von $\Sigma \cup \{\varepsilon\}$ bezeichnen.

Konventionen

- Der String $B_1 \cdots B_k$ wird im Grenzfall $k = 0$ mit dem leeren Wort identifiziert.
- Der Kellerinhalt wird als String notiert, wobei das „links“ im String „oben“ im Keller bedeutet.

Beachte, dass die Ersetzung von A durch $B_1 \cdots B_k$ im Keller die folgenden Grenzfälle mit einschließt:

Pop–Operation Im Falle $k = 0$ wird A aus dem Keller entfernt.

Ersetzung Im Falle $k = 1$ wird A durch B_1 ersetzt.

Falls $k \geq 1$ ist im Anschluss B_1 das oberste Kellersymbol.

Arbeitsweise des PDA

- Anfangs befindet sich M im Startzustand z_0 und der Keller enthält nur das unterste Kellersymbol #.
- Falls $(z', B_1 \cdots B_k) \in \delta(z, \varepsilon, A)$, dann darf M im Zustand z und bei oberstem Kellersymbol A eine ε–Transition vollziehen, wobei ein Wechsel auf den Zustand z' erfolgt und im Keller das oberste Symbol A durch $B_1 \cdots B_k$ ersetzt wird.
- Falls $(z', B_1 \cdots B_k) \in \delta(z, a, A)$ mit $a \in \Sigma$, dann darf M im Zustand z, bei oberstem Kellersymbol A und beim Lesen des Eingabesymbols a auf den Zustand z' wechseln und im Keller das oberste Symbol A durch $B_1 \cdots B_k$ ersetzen. Anschließend rückt der Lesekopf des Eingabebandes auf das nächste Symbol.
- Die Eingabe wird akzeptiert **gdw** M durch geeignete Wahl der jeweils möglichen Rechenschritte erreichen kann, dass der Keller nach Verarbeitung des letzten Zeichens der Eingabe (+ evtl. weiteren ε–Transitionen) leer ist.

Exkurs: Konfiguration (Momentaufnahme)

In eine Konfiguration für einen (evtl. nichtdeterministischen) Prozess packen wir alle Informationen hinein, die nötig sind, um

- den Prozess zu suspendieren (temporär zu stoppen)
- und ihn später wieder weiterlaufen zu lassen.

Beispiele:

- Informationen die Betriebssysteme von laufenden Prozessen protokollieren (s. Grundvorlesung zur Informatik oder Vorlesung über Betriebssysteme)
- Schachspiel mit Konfiguration bestehend aus der aktuellen Schachstellung und einem Zusatzbit, welches angibt, ob WEISS oder SCHWARZ am Zug ist (+ weitere Details wegen komplizierter Remis–Regeln)

Konfigurationen eines PDA

Die Konfiguration eines PDA besteht aus

- dem aktuellen Zustand $z \in Z$,
- dem noch nicht verarbeiteten Teil $v \in \Sigma^*$ der Eingabe,
- dem aktuellen Kellerinhalt $\alpha \in \Gamma^*$.

Notation: $(z, v, \alpha) \in Z \times \Sigma^* \times \Gamma^*$.

Anfangskonfiguration bei Eingabe w: $(z_0, w, \#)$

Akzeptierende Endkonfiguration: $(z, \varepsilon, \varepsilon)$ für jedes $z \in Z$

`Macht das Sinn ?? Erläutere !!`

Folgekonfigurationen

Eine „Rechnung“ eines PDA lässt sich als Folge von Konfigurationen beschreiben.

Definition

1. (z', v', α') heißt unmittelbare Folgekonfiguration von (z, v, α) **gdw** (z', v', α') aus (z, v, α) durch einen „Rechenschritt“ (einmalige Verwendung der Überführungsfunktion) resultieren kann.
   Notation: $(z, v, \alpha) \vdash (z', v', \alpha')$.

2. (z', v', α') heißt Folgekonfiguration von (z, v, α) **gdw** (z', v', α') aus (z, v, α) durch eine (evtl. leere) Folge von Rechenschritten resultieren kann.
   Notation: $(z, v, \alpha) \vdash^* (z', v', \alpha')$.
   Formal ist „$\vdash^*$” die reflexive–transitive Hülle von „$\vdash$“.

Sprache eines PDA

Die folgende Definition der vom PDA M erkannten Sprache $N(M)$ entspricht unserer Vereinbarung über das Akzeptieren mit leerem Keller:

$$N(M) := \{w \in \Sigma^* | \ \exists z \in Z : (z_0, w, \#) \vdash^* (z, \varepsilon, \varepsilon)\}$$

In Worten: Wort w gehört zur Sprache $N(M)$ **gdw** M durch Verarbeitung der Eingabe w aus der Anfangskonfiguration in eine akzeptierende Endkonfiguration gelangen kann.

Konvention: Im Folgenden schreiben wir (der besseren Lesbarkeit zuliebe)

$$zA \xrightarrow{a} z'x \text{ statt } (z', x) \in \delta(z, a, A) \ .$$

Beispiel

Die Sprache

$$L = \{a_1 a_2 \cdots a_n \$ a_n \cdots a_2 a_1 |\ a_i \in \{a, b\}\}$$

kann durch einen PDA erkannt werden, der

- in einer „Push–Phase“ $a_1, \ldots, a_n$ in den Keller legt (a_n befindet sich dann oben),
- beim Lesen des Trennzeichens „$“ von der „Push–“ in die „Pop-Phase“ umschaltet,
- in der Pop-Phase beim Lesen von a_i und oberstem Kellersymbol a_i (Match!) das Symbol a_i vom Keller entfernt,
- nach Abarbeitung der Eingabe und bei oberstem Kellersymbol „#“, das Symbol # vom Keller entfernt (um diesen vollständig zu leeren).

Beispiel (fortgesetzt)

Auf diese Weise werden auch Eingaben $w \notin L$ entlarvt. Beim Auftreten von „Störungen“ wie

- Eingabe w hat nicht die Form $w = u\$v$ mit $u, v \in \{a, b\}^*$,
- Auftreten eines „Mismatches“ in der Pop–Phase (gelesenes Zeichen stimmt nicht überein mit dem obersten Kellerzeichen),

wird der Keller (zur Strafe) nicht geleert.

Beachte, dass der Fall des Mismatches die Fälle

$$w = u\$v \ , |u| \neq |v|$$

mit einschließt.

Beispiel (fortgesetzt)

M hat die folgenden Komponenten:

- Zustandsmenge $\{z_0, z_1\}$
- Eingabealphabet $\{a, b, \$\}$
- Kelleralphabet $\{a, b, \#\}$
- Startzustand z_0
- unterstes Kellerzeichen $\#$

Intuition:

- Während der Push–Phase ist M im Zustand z_0.
- Während der Pop–Phase ist M im Zustand z_1.

Beispiel (fortgesetzt)

Die folgenden Transitionen realisieren die Push–Phase:

$$z_0\# \xrightarrow{a} z_0a\# \quad , \quad z_0a \xrightarrow{a} z_0aa \quad , \quad z_0b \xrightarrow{a} z_0ab$$
$$z_0\# \xrightarrow{b} z_0b\# \quad , \quad z_0a \xrightarrow{b} z_0ba \quad , \quad z_0b \xrightarrow{b} z_0\ bb$$

Das Umschalten von Push auf Pop wird realisiert durch

$$z_0\# \xrightarrow{\$} z_1\# \ , \ z_0a \xrightarrow{\$} z_1a \ , \ z_0b \xrightarrow{\$} z_1b \ .$$

Die folgenden Transitionen realisieren die Pop–Phase:

$$z_1a \xrightarrow{a} z_1\varepsilon \ , \ z_1b \xrightarrow{b} z_1\varepsilon \ , \ z_1\# \xrightarrow{\varepsilon} z_1\varepsilon$$

Auf der Eingabe $ba\$ab$ ergibt sich die Rechnung:

$$\begin{aligned}(z_0, ba\$ab, \#) &\vdash (z_0, a\$ab, b\#) \vdash (z_0, \$ab, ab\#) \vdash \\ (z_1, ab, ab\#) &\vdash (z_1, b, b\#) \vdash (z_1, \varepsilon, \#) \vdash (z_1, \varepsilon, \varepsilon)\end{aligned}$$

Beispiel (fortgesetzt)

Unser PDA für die Sprache

$$L = \{a_1 a_2 \cdots a_n \$ a_n \cdots a_2 a_1 |\ a_i \in \{a, b\}\}$$

ist deterministisch:

- Jede Konfiguration hat nur eine unmittelbare Folgekonfiguration.

Beispiel (fortgesetzt)

Die Sprache

$$L' = \{a_1 a_2 \cdots a_n a_n \cdots a_2 a_1 |\ a_i \in \{a, b\}\}$$

(L abzüglich des Trennzeichens) kann von einem PDA M' erkannt werden, der ein nicht–deterministischer Verwandter von M ist:

- Transitionen der Push- und Pop-Phase wie zuvor plus folgende Transitionen zum „Umschalten“:

$$z_0 a \xrightarrow{a} z_1 \varepsilon \ , \ z_0 b \xrightarrow{b} z_1 \varepsilon \ , \ z_0 \# \xrightarrow{\varepsilon} z_1 \varepsilon \ .$$

M' hat (sofern gelesenes Zeichen mit dem obersten Kellerzeichen übereinstimmt) die Wahl in der Push–Phase zu verweilen oder (mit Hilfe der neuen Transitionen) in die Pop–Phase umzuschalten. Bei Eingaben aus L' wird die akzeptierende Rechnung genau zum richtigen Zeitpunkt diese Umschaltung vornehmen (Raten der Nahtstelle an der Mitte des Eingabewortes).

- Man kann zeigen, dass kein deterministischer PDA für L' existiert.

Von der kontextfreien Grammatik zum PDA

Satz: Jede kontextfreie Grammatik G kann in einen äqivalenten PDA M transformiert werden.

Beweis: Gegeben $G = (V, \Sigma, P, S)$, wähle die Komponenten von M wie folgt:

- Zustandsmenge $\{z\}$ (**ein** Zustand)
- Eingabealphabet Σ
- Kelleralphabet $V \cup \Sigma$
- Überführungsfunktion δ (durch Transitionen weiter unten spezifiziert)
- Startzustand z
- unterstes Kellerzeichen S

Intuition: Zu Eingabe $w \in \Sigma^*$ versucht M eine Linksableitung $S \Rightarrow_G^* w$ herzustellen. Dabei soll zu jedem Zeitpunkt im Keller eine Satzform stehen, aus welcher der noch nicht gelesene Teil der Eingabe ableitbar ist (sofern $w \in L$).

Anfangs richtig: gesamte Eingabe w noch nicht gelesen, S im Keller.

Beweis (fortgesetzt)

Abkürzung: CFG = Contextfree Grammar = Kontextfreie Grammatik

Überführungsfunktion δ wird nach folgendem Schema entworfen:

CFG	PDA	
$A \to \alpha$	$zA \xrightarrow{\varepsilon} z\alpha$	(ε-Transition)
	$za \xrightarrow{a} z\varepsilon$	(a-Transition für jedes $a \in \Sigma$)

`--- In Worten ?! ---`

Beobachtung am Rande: Falls G in Greibach Normalform ist, dann hat jede Regel die Form $A \to \alpha$ mit $\alpha = a\alpha'$, $a \in \Sigma$, $\alpha' \in V^*$ und wir können wir nach dem Schema

CFG	PDA
$A \to a\alpha'$	$zA \xrightarrow{a} z\alpha'$

verfahren. Der resultierende PDA (ohne ε–Transitionen!) arbeitet nichtdeterministisch in Realzeit.

Beweis (fortgesetzt)

Der PDA M ist gerade so entworfen, dass seine Rechnungen Linksableitungen der Grammatik G entsprechen:

CFG	PDA
$S \Rightarrow_{lm}^* u\alpha \Rightarrow_{lm}^* uv$	$(z, uv, S) \vdash^* (z, v, \alpha) \vdash^* (z, \varepsilon, \varepsilon)$

`--- In Worten ?! ---`

Hierbei ist $w = uv \in \Sigma^*$ und α ist ein entweder leerer oder ein mit einer Variablen beginnender String. Index „lm“ steht für „leftmost“ und kennzeichnet Linksableitungsschritte.

Es folgt direkt: $L(G) = N(M)$.

Bleibt zu zeigen, dass sich Grammatik und PDA wechselseitig nach obigem Schema „simulieren“ können. Wir konzentrieren uns dabei auf die „Simulation“ eines Ableitungsschrittes bezüglich G durch Rechenschritte von M. (Die umgekehrte Überlegung funktioniert aber analog.)

Beweis (fortgesetzt)

Wir nehmen an, dass α von der Form $\alpha = X\alpha'$ ist und in der Linksableitung $u\alpha \Rightarrow_{lm}^{*} w = uv$ eine Regel der Form

$$X \to a_1 \cdots a_k \alpha''$$

zur Anwendung kommt (mit $a_1 \cdots a_k$ als maximalem Präfix aus Σ^* auf der rechten Seite). Dann muss natürlich v von der Form

$$v = a_1 \cdots a_k v'$$

sein und es muss $\alpha''\alpha' \Rightarrow_{lm}^{*} v'$ gelten. Der PDA simuliert diesen Ableitungsschritt (illustriert für $k = 1$) nach folgendem Schema:

CFG	PDA
$u\alpha = uX\alpha' \Rightarrow_{lm} ua_1\alpha''\alpha'$	$(z, a_1v', X\alpha') \vdash (z, a_1v', a_1\alpha''\alpha')$
	$\vdash (z, v', \alpha''\alpha')$

Vom PDA zur kontextfreien Grammatik

Satz: Jeder PDA M kann in eine äquivalente kontextfreie Grammatik G transformiert werden.

Beweisstruktur:

- Bringe M zunächst in eine „Normalform“ (ohne die zugehörige Sprache abzuändern).
- Entwerfe eine kontextfreie Grammatik G, deren Linksableitungen gerade den akzeptierenden Rechnungen von M entsprechen.

„Normalform“ des PDA

Transitionen der Form

$$zA \xrightarrow{a} z'B_1 \cdots B_k \text{ mit } k > 2$$

könnten mit Hilfe neuer Zustände $z_1, \ldots, z_{k-2}$ simuliert werden durch die Transitionen

$$zA \xrightarrow{a} z_1 B_{k-1} B_k \ , \ z_1 B_{k-1} \xrightarrow{\varepsilon} z_2 B_{k-2} B_{k-1} \ , \ \ldots \ , \ z_{k-2} B_2 \xrightarrow{\varepsilon} z' B_1 B_2 \ .$$

Daher können wir **annehmen**, dass der gegebene PDA M nur Transitionen der Form

$$zA \xrightarrow{a} z'B_1 \cdots B_k \text{ mit } k \in \{0, 1, 2\}$$

verwendet.

Entwurf einer Grammatik (Tripelkonstruktion)

Gegeben der (normalisierte) PDA $M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$, wähle die Komponenten von G wie folgt:

- Variablenmenge $V = \{S\} \cup (Z \times \Gamma \times Z)$
- Terminalalphabet Σ
- Produktionensystem P (durch Regeln weiter unten spezifiziert)
- Startsymbol S

Wegen der Variablen der Form $(z, A, z') \in Z \times \Gamma \times Z$ spricht man von der „Tripelkonstruktion“.

Intuition: Interpretiere das Tripel (z, A, z') als „Wette“ auf folgendes Ereignis: nachdem M, gestartet im Zustand z mit Kellerinhalt A, den Keller geleert hat, befindet er sich im Zustand z'.

- Nimm für jedes $z \in Z$ die Regel $S \to (z_0, \#, z)$ in P auf.
- Für jede Transition von M der Form $zA \xrightarrow{a} z'\varepsilon$ nimm die Regel $(z, A, z') \to a$ in P auf.
- Für jede Transition von M der Form $zA \xrightarrow{a} z_1 B$ und jeden Zustand $z' \in Z$, nimm die Regel der Form $(z, A, z') \to a(z_1, B, z')$ in P auf.
- Für jede Transition von M der Form $zA \xrightarrow{a} z_1 BC$ und jede Wahl von $z', z_2 \in Z$, nimm die Regel $(z, A, z') \to a(z_1, B, z_2)(z_2, C, z')$ in P auf.

Die folgende Äquivalenz ist mit Induktion (nach der Länge der Rechnung von M bzw. nach der Länge der Linksableitung) beweisbar:

$$(z, x, A) \vdash^* (z', \varepsilon, \varepsilon) \textbf{ gdw } (z, A, z') \Rightarrow_{lm}^* x$$

$L(G) = N(M)$ ergibt sich jetzt wie folgt:

$$\begin{aligned} w \in N(M) &\Leftrightarrow \exists z \in Z : (z_0, w, \#) \vdash^* (z, \varepsilon, \varepsilon) \\ &\Leftrightarrow \exists z \in Z : S \Rightarrow (z_0, \#, z) \Rightarrow_{lm}^* w \\ &\Leftrightarrow w \in L(G) \end{aligned}$$

Induktionsbeweis (Skizze)

Wir beschränken uns hier auf den Induktionsschritt für die Behauptung

Wenn $(z, x, A) \vdash^* (z', \varepsilon, \varepsilon)$ **dann** $(z, A, z') \Rightarrow_{lm}^* x$

und betrachten auch nur den folgenden kompliziertesten Fall:

- $x = auv$ ist eine Zerlegung von x, die die folgende Beschreibung erleichtert.
- M ersetzt zunächst mit einer a-Transition Kellersymbol A durch BC.
- u ist das Teilwort nach dessen Verarbeitung zum ersten Male nur noch das unterste Symbol (nämlich C) im Keller steht.
- Nach Verarbeitung des Teilwortes v ist der Keller zum ersten Male leer.

Induktionsbeweis (fortgesetzt)

Die geschilderte Rechnung von M auf $x = auv$ hat die Form

$$(z, auv, A) \vdash (z_1, uv, BC) \vdash^* (z_2, v, C) \vdash^* (z', \varepsilon, \varepsilon)$$

und kann von G „simuliert“ werden wie folgt:

$$(z, A, z') \Rightarrow_{lm} a(z_1, B, z_2)(z_2, C, z') \Rightarrow^*_{lm} au(z_2, C, z') \Rightarrow^*_{lm} auv$$

Bei den mit „$\Rightarrow^*_{lm}$“ gekennzeichneten Passagen der Linksableitung wurde die Induktionsvoraussetzung eingesetzt.

Folgerungen

1. Die Klasse der von PDAs erkennbaren Sprachen stimmt mit der Klasse der kontextfreien Sprachen überein.

2. Jede kontextfreie Sprache kann von einem PDA erkannt werden, der nur einen einzigen Zustand besitzt, ohne ε–Transitionen auskommt und daher nichtdeterministisch in Realzeit arbeitet.

Deterministische Kellerautomaten (DPDAs)

Intuitiv nennen wir einen Kellerautomaten deterministisch, wenn er über Endzustände (statt mit leerem Keller) akzeptiert und wenn er in jeder Konfiguration maximal eine mögliche Transition (inklusive der ε–Transitionen) zur Verfügung hat.

Formal: Ein Kellerautomat M heißt deterministisch oder kurz DPDA **gdw** gilt:

- Neben den üblichen Komponenten eines Kellerautomaten ist noch eine Menge $E \subseteq Z$ von Endzuständen spezifiziert.
- Für alle $z \in Z$, $a \in \Sigma$ und $A \in \Gamma$ gilt

$$|\delta(z, a, A)| + |\delta(z, \varepsilon, A)| \leq 1 \ .$$

Deterministisch kontextfreie Sprachen

Die von einem DPDA $M = (Z, \Sigma, \Gamma, \delta, z_0, \#, E)$ erkannte Sprache ist dann gegeben durch

$$\{w \in \Sigma^* | \ \exists z \in E, \exists \alpha \in \Gamma^* : (z_0, w, \#) \vdash^* (z, \varepsilon, \alpha)\} \ .$$

Eine Sprache L heißt deterministisch kontexftfrei **gdw** ein DPDA existiert, der L erkennt.

Bemerkung (s. Vorlesung über Compilerbau):

DPDAs $\leftrightarrow$ LR(k)–Grammatiken

Deterministisch kontextfreie Sprachen (fortgesetzt)

Satz: Jede reguläre Sprache ist deterministisch kontextfrei.

denn: Jeder DFA kann als DPDA aufgefasst werden, der weder von seinem Keller noch von ε–Transitionen Gebrauch macht.

Satz (ohne Beweis): Jeder PDA, der mit leerem Keller akzeptiert, kann in einen äquivalenten PDA transformiert werden, der mit Endzuständen akzeptiert (und umgekehrt).

Folgerung: Jede deterministisch kontextfreie Sprache ist kontextfrei.

denn: Jeder DPDA kann als Spezialfall eines mit Endzuständen akzeptierenden PDA angesehen werden.

Beispiele

Die Sprache

$$L_1 = \{a^m b^m c^n |\ m \geq 1, n \geq 0\}$$

kann von einem DPDA mit $E = \{z_2\}$ und den folgenden Transitionen erkannt werden:

$z_0\# \xrightarrow{a} z_0A\#$, $z_0A \xrightarrow{a} z_0aA$, $z_0a \xrightarrow{a} z_0aa$		Push–Phase
$z_0A \xrightarrow{b} z_2\varepsilon$, $z_0a \xrightarrow{b} z_1\varepsilon$		Umschalten auf
$z_1A \xrightarrow{b} z_2\varepsilon$, $z_1a \xrightarrow{b} z_1\varepsilon$		die Pop–Phase
$z_2\# \xrightarrow{c} z_2\#$		Endroutine

Kellersymbol A repräsentiert das erste gelesene a-Zeichen. Mit Hilfe dieses Kellersymbols kann der DPDA deterministisch erkennen, wann er in den Endzustand z_2 wechseln muss.

Beispiele (fortgesetzt)

Die folgenden Sprachen sind deterministisch kontextfrei:

$$\begin{aligned} L_1 &= \{a^m b^m c^n \mid m, n \geq 1\} \\ L_2 &= \{a^n b^m c^m \mid m, n \geq 1\} \\ L_3 &= \{a_1 \cdots a_n \$ a_n \cdots a_1 \mid n \geq 1, a_i \in \{a, b\}\} \end{aligned}$$

- DPDA für L_2 analog zum DPDA für L_1.
- L_3 erkennbar mit einer DPDA–Variante des PDA, der früher schon mal für diese Sprache angegeben wurde.

Die Sprache

$$L_1 \cap L_2 = \{a^m b^m c^m \mid m \geq 1\}$$

ist (wie wir bereits wissen) nicht kontextfrei.

Folgerung: Weder deterministisch kontextfreie Sprachen noch kontextfreie Sprachen sind abgeschlossen unter „$\cap$“.

Abschlusseigenschaften

Satz: Kontextfreie Sprachen sind abgeschlossen unter den Operationen „$\cup, \cdot, *$“.

denn: Gehe aus von den Grammatiken

$$G_1 = (V_1, \Sigma, P_1, S_1) \ , \ G_2 = (V_2, \Sigma, P_2, S_2) \ , \ V_1 \cap V_2 = \emptyset$$

für kontextfreie Sprachen L_1, L_2. Übernahme der Regeln aus P_1 und P_2 und Hinzufügen der Regeln

$$S \to S_1 \mid S_2 \text{ bzw. } S \to S_1 S_2 \text{ bzw. } S \to \varepsilon \mid S_1 S$$

mit neuem Startsymbol S führt zu einer kontextfreien Grammatik für $L_1 \cup L_2$ bzw. für $L_1 \cdot L_2$ bzw. $(L_1)^*$.

Abschlusseigenschaften (fortgesetzt)

Kontextfreie Sprachen sind nicht abgeschlossen unter den Operationen „$\cap, \neg$“.

denn:

- Nicht–Abgeschlossenheit unter „$\cap$“ wurde bereits gezeigt.
- Mit deMorgan folgt die Nicht–Abgeschlossenheit unter „$\neg$“.
  Genauer: Abschluss unter „$\neg$“ hätte wegen

$$L_1 \cap L_2 = \overline{\overline{L_1 \cap L_2}} = \overline{\bar{L}_1 \cup \bar{L}_2}$$

  den Abschluss unter „$\cap$“ zur Folge (und ist daher unmöglich).

Abschlusseigenschaften (fortgesetzt)

Satz: Die deterministisch kontextfreien Sprachen sind abgeschlossen unter „$\neg$“, aber nicht abgeschlossen unter „$\cap, \cup, \cdot, *$“.

- Abschluss unter „$\neg$“ ist nicht ganz einfach zu beweisen.
  (Hier: ohne Beweis)
- Nicht–Abgeschlossenheit unter „$\cap$“ wurde bereits gezeigt.
- Nicht–Abgeschlossenheit unter „$\cup$“ folgt dann direkt mit deMorgan.
- Nicht–Abgeschlossenheit unter „$\cdot, *$“ ist einfacher zu beweisen, wenn das Konzept des „Homomorphismus“ $\varphi : \Sigma^* \to \Sigma^*$ zur Verfügung steht.
  (Hier: ohne Beweis)

Abschlusseigenschaften (fortgesetzt)

Satz: Der Durchschnitt einer (deterministisch) kontextfreien Sprache mit einer regulären Sprache liefert wieder eine (deterministisch) kontextfreie Sprache.

Zu einem gegeben PDA $M_1 = (Z_1, \Sigma, \Gamma_1, \delta_1, z_{01}, E_1, \#)$ — Akzeptieren mit Endzuständen ! — und einem gegebenen DFA $M_2 = (Z_2, \Sigma, \delta_2, z_{02}, E_2)$ betrachte den „Produktautomaten“

$$M = (Z_1 \times Z_2, \Sigma, \Gamma_1, \delta, (z_{01}, z_{02}), \#, E_1 \times E_2)$$

mit $((z_1', z_2'), B_1 \cdots B_k) \in \delta((z_1, z_2), a, A)$ **gdw**

$$(z_1', B_1 \cdots B_k) \in \delta_1(z_1, a, A) \text{ und } z_2' = \delta_2(z_2, a) \ .$$

Intuition: M führt die „Rechnungen“ von M_1 und M_2 parallel aus.

Dann gilt:

- M ist ein PDA und erkennt die Sprache $N(M_1) \cap T(M_2)$.
- Wenn M_1 ein DPDA ist, dann ist auch M ein DPDA.

Das Wortproblem

Spezielles Wortproblem für eine kontextfreie Sprache $L \subseteq \Sigma^*$

Eingabe: $x \in \Sigma^*$

Frage: $x \in L$?

Komplexität: lösbar in „kubischer“ Zeit ($O(n^3)$ Rechenschritte).

Methode: Verwende den CYK–Algorithmus: n^2 Tabelleneinträge; jeder Eintrag kann in Linearzeit ($O(n)$ Rechenschritte) bestimmt werden.

Bemerkung: Für eine deterministisch kontextfreie Sprachen kann das Wortproblem (unter Verwendung einer LR(k)–Grammatik) in Linearzeit gelöst werden.

Nützliche und nutzlose Variable

Eine Variable $A \in V$ einer kontextfreien Grammatik $G = (V, \Sigma, P, S)$ heißt

- generierend **gdw** $\exists w \in \Sigma^* : A \Rightarrow_G^* w$
  (d.h., aus A kann ein Terminalstring abgeleitet werden),
- erreichbar **gdw** $\exists \alpha, \beta \in (V \cup \Sigma)^* : S \Rightarrow_G^* \alpha A \beta$,
- nützlich **gdw** A generierend und erreichbar ist,
- nutzlos **gdw** A nicht nützlich ist.

Säubern einer kontextfreien Grammatik

Eine kontextfreie Grammatik G heißt gesäubert **gdw** G keine nutzlosen Variablen enthält.

Säubern von G (ohne die Sprache abzuändern):

1. Berechne (s. Übung) die Menge aller generierenden Variablen.
2. Entferne aus G die nicht–generierenden Variablen und die Regeln, welche nicht-generierende Variablen verwenden.
3. **Kommentar:** Jetzt sind alle Variablen generierend.
4. Berechne (s. Übung) die Menge der erreichbaren Variablen.
5. Entferne aus G die nicht–erreichbaren Variablen und die Regeln, welche nicht–erreichbare Variablen enthalten.
6. **Kommentar:** Jetzt sind alle Variablen generierend und erreichbar und somit nützlich.

Bei geschickter Implementierung erfolgt Säubern in Linearzeit !

Das „Leerheitsproblem“

Eingabe: kontextfreie Grammatik G.

Frage: $L(G) = \emptyset$?

Komplexität: Linearzeit

Methode: Nutze aus:

$$L(G) \neq \emptyset \Leftrightarrow \ S \text{ ist generierend}$$

Das „Endlichkeitsproblem“

Eingabe: Kontextfreie Grammatik G

Frage: $|L(G)| < \infty$?

Komplexität: Linearzeit

Methode:
- Säubere G.
- Berechne einen Digraphen D mit Knotenmenge V, der eine Kante (A, B) enthält **gwd** es gibt eine Regel mit A auf der linken und B auf der rechten Seite.
- Nutze aus:

$$|L(G)| = \infty \Leftrightarrow D \text{ enthält einen Zyklus}$$

Ausblick: Nicht–entscheidbare Probleme

Im Kapitel über Berechenbarkeitstheorie
werden wir sehen:
Es gibt algorithmisch nicht entscheidbare Probleme.

Beispiele für nicht–entscheidbare Probleme

Für keines der folgenden Probleme gibt es einen Entscheidungsalgorithmus:

Schnittproblem: Gegeben sind zwei kontextfreie Grammatiken (oder zwei PDAs).

Frage: Exisitiert ein Terminalstring, der von beiden Grammatiken erzeugt (von beiden PDAs akzeptiert) werden kann ?

Kommentar: Sogar das Schnittproblem für deterministisch kontextfreie Sprachen (etwa gegeben durch DPDAs) ist unentscheidbar.

Äquivalenzproblem Erzeugen zwei gegebene kontextfreie Grammatiken dieselbe Sprache?

Das Äquivalenzproblem für deterministisch kontextfreie Sprachen

— Erkennen zwei gegebene DPDAs dieselbe Sprache ? —

ist hingegen entscheidbar.

Kommentar: 1997 fand Senizergues einen Entscheidungsalgorithmus.

Lernziele zum Level der kontextfreien Sprachen

- Kenntnis der wesentlichen Konzepte und Resultate auf dem Level der (deterministisch) kontextfreien Sprachen besitzen und Zusammenhänge verstehen
- Syntaxbäume und grammatische (Links-)Ableitungen angeben können
- zu einer (evtl. deterministisch) kontextfreien Sprache die passende kontextfreie Grammatik bzw. den passenden PDA (DPDA) angeben können
- zu einer gegebenen kontextfreien Grammatik (bzw. einem gegebenen PDA oder DPDA) die zugehörige Sprache ableiten können
- eine kontextfreie Grammatik in Chomsky oder Greibach Normalform transformieren können
- Nachweis der Nicht–Kontextfreiheit einer Sprache mit Hilfe des Pumping–Lemmas (evtl. auch unter Ausnutzung von Abschlusseigenschaften) führen können

Lernziele (fortgesetzt)

- den CYK–Algorithmus auf Instanzen des Wortproblems anwenden können
- Rechnungen von PDAs oder DPDAs als Konfigurationsfolgen ausdrücken können
- eine kontextfreie Grammatik in einen äquivalenten PDA transformieren können und umgekehrt
- die besprochenen Syntheseprobleme lösen können
- die besprochenen Entscheidungsprobleme lösen können