Kontextfreie Grammatiken

Kontextfreie Grammatiken sind eine Erfindung Chomskys und wurden für die Beschreibung natürlicher Sprachen konzipiert. Eine kontextfreie Grammatik hat eine Menge *Umschreibungsregeln* (*rewriting rules*) und ein *Startsymbol*. Die Zeichenketten der beschriebenen Sprache werden generiert, indem man mit dem Startsymbol anfängt, und beliebige Umschreibungsregeln auf dieses Symbol anwendet, bis eine Kette aus Zeichen, die nicht weiter umschreibbar sind, entsteht.

Beispiel:

S → NP VP

NP → D N

D → *ein*

N → *Mann*

VP → *lacht*

Beachte: *ein*, *Mann* und *lacht* sind hier *Symbole* aus einem Alphabet, keine Wörter!

Mathematische Definition von kontextfreien Grammatiken

Eine kontextfreie Grammatik besteht aus

i) V: dem Alphabet

ii) $\Sigma \subseteq V$: den *Terminalsymbolen*

iii) $S \in V - \Sigma$: dem *Startsymbol*

und schließlich den *(Produktions-) Regeln*, die wir als eine (endliche) Relation definieren:

iv) $R \subseteq (V - \Sigma) \times V^*$

$V - \Sigma$ nennt man die Menge der *Nichtterminalen* (oder auch *Variablen*, *Kategorien*, *Metasymbole*).

Wenn $\langle A, u \rangle \in R$, schreiben wir $A \xrightarrow[G]{} u$, oder einfach $A \to u$.

Notation

- Große Buchstaben für die Nichtterminalen
- Kleine Buchstaben für die Terminalsymbole

Ableitungen

Für $u, v \in V^*$ definieren wir:

$u \overset{}{\underset{G}{\Rightarrow}} v$ *(v ist direkt ableitbar von u)* gdw es gibt $x, y, v' \in V^*$ und $A \in V - \Sigma$, so dass $u = xAy$, $v = xv'y$ und $A \xrightarrow[G]{} v'$.

Eine *Ableitung* ist eine Folge von direkten Ableitungen:

$w_0 \Rightarrow_G w_1 \Rightarrow_G \ldots \Rightarrow_G w_n$

Wir schreiben dann $w_0 \Rightarrow_G^n w_n$ *und sagen:* w_n *ist aus* w_0 *in n Schritten ableitbar*.

Wenn es ein $n \geq 0$ gibt, so dass $u \Rightarrow_G^n v$, heißt v *ableitbar aus* u, und wir schreiben $u \Rightarrow_G^* v$.

($\Rightarrow_G^*$ ist die reflexiv-transitive Hülle von $\Rightarrow_G$)

Die kontextfreien Sprachen

Die Sprache $L(G)$, die eine kontextfreie Grammatik G generiert, besteht aus den Zeichenketten über Σ, die vom Startsymbol ableitbar sind. Formal:

$L(G) = \{w \in \Sigma^* : S \Rightarrow_G^* w\}$

Wenn $L = L(G)$ für eine kontextfreie Grammatik G, heißt L *kontextfrei*.

Beispiel: Eine kontextfreie, nicht reguläre Sprache

$G = \langle V, \Sigma, R, S \rangle$, wobei

- $V = \{S, a, b\}$,
- $\Sigma = \{a, b\}$,
- $R = \{S \to aSb, S \to \varepsilon\}$

$L(G) = \{a^n b^n : n \geq 0\}$

Reguläre Grammatiken

Sei G eine kontextfreie Grammatik.

G ist eine *reguläre Grammatik* gdw
$R \subseteq (V - \Sigma) \times \Sigma^*((V - \Sigma) \cup \{\varepsilon\})$

D.h. auf der rechten Seite der Regeln einer regulären Grammatik kommt höchstens ein Nichtterminalsymbol vor; wenn ein Nichtterminalsymbol vorkommt, dann muss es ganz rechts stehen.

Beispiel für eine reguläre Grammatik:

$G = \langle V, \Sigma, R, S \rangle$, wobei

- $V = \{S, B, a, b\}$,
- $\Sigma = \{a, b\}$,
- $R = \{S \to aS, S \to B, B \to bB, B \to \varepsilon\}$

$L(G) = a^* b^*$

Theorem

Sei L eine kontextfreie Sprache. L ist regulär gdw $L = L(G)$ für eine reguläre Grammatik G.

Beweis

„⇒“ Angenommen, L ist regulär. Dann gibt es einen DEA $M = \langle K, \Sigma, \delta, s, F\rangle$, der L akzeptiert. Wir können annehmen, dass K und Σ keine gemeinsamen Elemente haben. Wir konstruieren jetzt die reguläre Grammatik $G = \langle V, \Sigma, R, S\rangle$, mit

- $V = \Sigma \cup K$,
- $S = s$,
- $R = \{q \to ap : \delta(q, a) = p\} \cup \{q \to \varepsilon : q \in F\}$

Weil $q \to ap$ gdw $\delta(q, a) = p$, gilt:

$$\begin{aligned}\langle q_0, \sigma_1 \ldots \sigma_n\rangle &\vdash_M \langle q_1, \sigma_2 \ldots \sigma_n\rangle \\ &\vdash_M \langle q_2, \sigma_3 \ldots \sigma_n\rangle \\ &\vdash_M \ldots \\ &\vdash_M \langle q_{n-1}, \sigma_n\rangle \\ &\vdash_M \langle q_n, \varepsilon\rangle\end{aligned}$$

gdw

$$\begin{aligned}q_0 &\overset{}{\underset{G}{\Rightarrow}} \sigma_1 q_1 \\ &\overset{}{\underset{G}{\Rightarrow}} \sigma_1\sigma_2 q_1 \\ &\overset{}{\underset{G}{\Rightarrow}} \ldots \\ &\overset{}{\underset{G}{\Rightarrow}} \sigma_1 \ldots \sigma_n q_n\end{aligned}$$

Deshalb gilt:

$w \in L(M)$

gdw es gibt ein $p \in F$ so dass $\langle s, w\rangle \vdash_M^* \langle p, \varepsilon\rangle$

gdw es gibt ein $p \in F$ so dass $s \overset{*}{\underset{G}{\Rightarrow}} wp$

gdw $w \in L(G)$

Zum letzten Schritt: „⇓“ gilt, weil es zu jedem $p \in F$ eine Regel $p \to \varepsilon$ gibt. „⇑“ gilt, weil eine Regel vom Typ $p \to \varepsilon$ immer die zuletzt angewandte Regel in einer Ableitung sein muss.

„⇐“ Angenommen, $L = L(G)$ für eine reguläre Grammatik $G = \langle V, \Sigma, R, S\rangle$. Wir konstruieren einen NEA $M = \langle K, \Sigma, \Delta, s, F\rangle$ mit

- $K = (V - \Sigma) \cup \{f\}$ (f neu)
- $s = S$
- $F = \{f\}$
- $\Delta = \{\langle A, w, B\rangle : A \overset{}{\underset{G}{\to}} wB \text{ und } B \in V - \Sigma\} \cup$
  $\{\langle A, w, f\rangle : A \overset{}{\underset{G}{\to}} w \text{ und } w \in \Sigma^*\}$

Es gilt:

$w \in L(G)$

gdw es gibt $A_1, \ldots, A_n \in V - \Sigma$ und $w_0, \ldots, w_n \in \Sigma^*$ so dass $w = w_0 \ldots w_n$ und $S \overset{*}{\underset{G}{\Rightarrow}} w_0 A_1 \overset{*}{\underset{G}{\Rightarrow}} \ldots \overset{*}{\underset{G}{\Rightarrow}} w_0 \ldots w_{n-1} A_n \overset{*}{\underset{G}{\Rightarrow}} w_0 \ldots w_n$

gdw es gibt $A_1, \ldots, A_n \in V - \Sigma$ und $w_0, \ldots, w_n \in \Sigma^*$ so dass $w = w_0 \ldots w_n$ und $\langle S, w_0 \ldots w_n \rangle \vdash_M \langle A_1, w_1 \ldots w_n \rangle \vdash_M \ldots \vdash_M \langle A_n, w_n \rangle \vdash_M \langle f, \varepsilon \rangle$

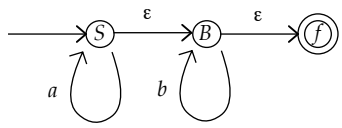
gdw $w \in L(M)$ □

Beispiel

Betrachten wir wieder $G = \langle V, \Sigma, R, S \rangle$ mit

- $V = \{S, B, a, b\}$,
- $\Sigma = \{a, b\}$,
- $R = \{S \to aS, S \to B, B \to bB, B \to \varepsilon\}$

Die Konstruktion aus dem Beweis gibt uns den nichtdeterministischen Automaten



Korrolar

Die regulären Sprachen bilden eine echte Teilmenge der Menge der kontextfreien Sprachen.

Beweis

Jede reguläre Sprache ist auch kontextfrei: Sie wird von einer regulären – und deshalb auch kontextfreien – Grammatik generiert. Auf der anderen Seite gibt es kontextfreie Sprachen, die nicht regulär sind. □

Ableitungsbäume

Ableitungsbäume stellen Ableitungen graphisch dar. Zum Beispiel können wir die beiden in gewissem Sinne äquivalenten Ableitungen

S ⇒ NP VP

⇒ *Juan* VP

⇒ *Juan* V NP

⇒ *Juan feeds* NP

⇒ *Juan feeds Pedro*

und

S ⇒ NP VP

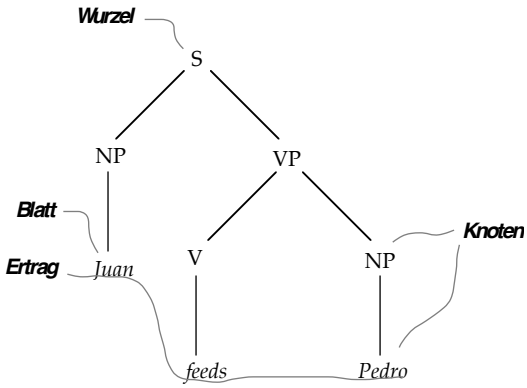
⇒ *Juan* VP

⇒ *Juan* V NP

⇒ *Juan* V *Pedro*

⇒ *Juan feeds Pedro*

mit diesem Ableitungsbaum darstellen:



Das Pumping-Lemma für kontextfreie Sprachen

Theorem

Sei G eine kontextfreie Grammatik. Dann gibt es eine Zahl K, so dass für jede Zeichenkette $w \in L(G)$ mit $|w| > K$ folgendes gilt:

Es gibt Zeichenketten $u, v, x, y, z \in \Sigma^s$ so dass:

1) $w = uvxyz$

2) mindestens eins von v und y ist nicht ε

3) Für jedes $n \geq 0$: $uv^nxy^nz \in L(G)$

Um das Pumping-Lemma beweisen zu können, brauchen wir erst noch einige Definitionen zu Ableitungsbäumen:

Ein *Pfad* in einem Ableitungsbaum ist eine Folge von Knoten, die mit der Wurzel anfängt und mit einem Blatt endet, und wo jedes Element in der Folge im Baum durch eine Kante mit dem vorigen Element verbunden ist.

Die *Länge* des Pfades ist gleich der Anzahl der Kanten, d.h. gleich der Anzahl der Knoten minus 1.

Die *Höhe* eines Baumes ist gleich der Länge des längsten Pfades.

Der *Ertrag* eines Baumes ist das Wort, das man erhält, wenn man die Blätter von links nach rechts betrachtet.

Beweis des Pumping-Lemmas:

Sei $G = \langle V, \Sigma, R, S \rangle$ eine kontextfreie Grammatik. Es genügt, folgendes zu zeigen:

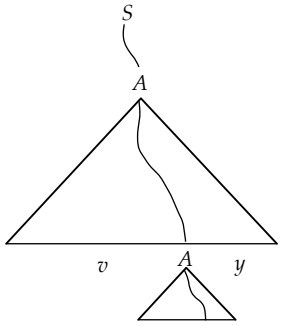
(*) Es gibt ein K, so dass jede Zeichenkette in $L(G)$ länger als K eine Ableitung

$S \overset{*}{\underset{G}{\Rightarrow}} uAz \overset{*}{\underset{G}{\Rightarrow}} uvAyz \overset{*}{\underset{G}{\Rightarrow}} uvxyz$

hat, wobei $u, v, x, y, z \in \Sigma^*$, $A \in V\text{–}\Sigma$, und $v \neq \varepsilon$ oder $y \neq \varepsilon$, weil die Ableitung $A \overset{*}{\underset{G}{\Rightarrow}} vAy$ dann beliebig viele Male (auch 0-mal) wiederholt werden kann.

Sei $p = max\{|\alpha| : A \underset{G}{\rightarrow} \alpha\}$. Dann hat ein Ableitungsbaum der Höhe m höchstens p^m Blätter. Oder umgekehrt: Wenn ein Ableitungsbaum einen Ertrag mit Länge $> p^m$, dann hat der Baum einen Pfad, der länger als m ist.

Sei $m = |V\text{–}\Sigma|$, p wie oben, $K = p^m$, w ein Wort mit Länge $> K$. Sei T ein Ableitungsbaum mit Wurzel S und Ertrag w. Dann hat T mindestens einen Pfad mit mehr als $|V\text{–}\Sigma|+1$ Knoten, und so ein Pfad muss mindestens zwei Knoten haben, die mit dem selben Nichtterminalen markiert sind:

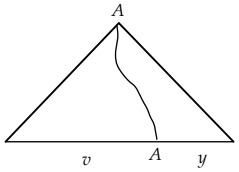


Wir haben also für jeden solchen Pfad eine Ableitung

$S \overset{*}{\underset{G}{\Rightarrow}} uAz \overset{*}{\underset{G}{\Rightarrow}} uvAyz \overset{*}{\underset{G}{\Rightarrow}} uvxyz,$

wie wir zeigen wollten. Nun muss nur noch gezeigt werden, dass es mindestens einen solchen Pfad gibt, wo mindestens eins von v und y nicht-leer ist.

Für jeden Fall, wo v und y beide leer sind, können wir den Teilbaum



einfach entfernen, ohne dass sich der Ertrag des Baumes ändert. Dies kann aber nicht für jeden Pfad mit Länge > m möglich sein, weil es sonst einen Baum mit Höhe ≤ m gäbe, der w als Ertrag hat, und dies ist nicht möglich da $|w| > K = p^m$. □

Beispiel

$L = \{a^n b^n c^n : n \geq 0\}$ ist nicht kontextfrei.

Beweis

Nehmen wir an, L ist kontextfrei, d.h. $L = L(G)$ für eine kontextfreie Grammatik G. Sei K wie im Pumping-Lemma, und sei $n > K / 3$. Dann erfüllt $w = a^n b^n c^n$ die Voraussetzung des Pumping-Lemmas ($|w| > K$). Also ist $w = uvxyz$, wo mindestens eins von v und y nicht-leer ist.

Es gibt zwei Möglichkeiten:

1) Mindestens eins von v oder y enthält zwei verschiedene Zeichen (a und b oder b und c), und deshalb die Teilzeichenkette ab oder bc. Dann kommt aber in uv^2xy^2z mindestens ein b vor einem a vor, oder ein c vor einem b, also $w \notin L(G)$ – Kontradiktion!

2) v oder y oder beide bestehen nur aus gleichen Zeichen (z.B. $v = bb$, $y = cc$). Dann hat aber uv^2xy^2z nicht die selbe Anzahl von a's, b's und c's (sondern z.B. 2 a's zu wenig, wenn $v = bb$ und $y = cc$) – Kontradiktion! □

Abschlußeigenschaften der Klasse der kontextfreien Sprachen

Wir fangen mit einem negativen Ergebnis an, das direkt aus dem vorigen Beispiel folgt:

Theorem

Die Klasse der kontextfreien Sprachen ist unter Schnittbildung *nicht* abgeschlossen.

Beweis

Die Sprachen L_1 und L_2 sind kontextfrei (die entsprechenden Grammatiken können leicht angegeben werden):

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

$L_2 = \{a^m b^m c^n : m, n \geq 0\}$

Aber $L_1 \cap L_2 = \{a^n b^n c^n : n \geq 0\}$, und diese Sprache ist, wie wir jetzt wissen, nicht kontextfrei. □

Theorem

Die Klasse der kontextfreien Sprachen ist unter Vereinigung, Konkatenation und Kleene'scher Hüllenbildung abgeschlossen.

Beweis

Seien $G_1 = \langle V_1, \Sigma_1, R_1, S_1 \rangle$ und $G_2 = \langle V_2, \Sigma_2, R_2, S_2 \rangle$. Wir können annehmen, dass $(V_1 - \Sigma_1) \cap (V_2 - \Sigma_2) = \varnothing$. (Man kann z.B. die Nichtterminalen umbenennen, indem man die Symbole mit 1, bzw. 2, indexiert.)

Vereinigung:

Sei S ein neues Symbol. $G = \langle V_1 \cup V_2, \Sigma_1 \cup \Sigma_2, R, S \rangle$ mit

$R = R_1 \cup R_2 \cup \{S \to S_1, S \to S_2\}$

generiert $L(G_1) \cup L(G_2)$

Dass $L(G) \supseteq L(G_1) \cup L(G_2)$ ist klar. Weil G_1 und G_2 keine gemeinsamen Nichtterminale haben, gilt auch die umgekehrte Richtung.

Konkatenation:

Ähnlich, mit neuer Regel $S \to S_1 S_2$.

Kleene'sche Hülle:

Sei $G = \langle V_1, \Sigma_1, R_1 \cup \{S_1 \to \varepsilon, S_1 \to S_1 S_1\}, S_1 \rangle$. Dann ist $L(G) = L(G_1)^*$. □

Korrolar

Die Klasse der kontextfreien Sprachen ist unter Komplementbildung nicht abgeschlossen.

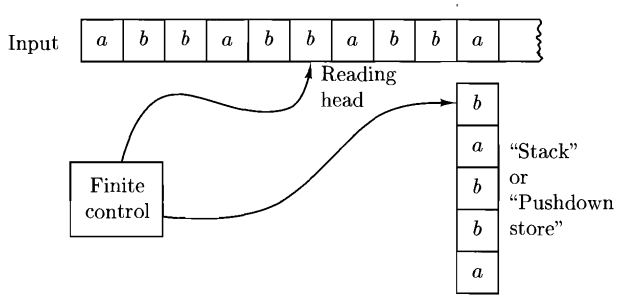
Beweis

Wenn sie das wäre, wäre sie auch unter Schnittbildung abgeschlossen, weil:

$L_1 \cap L_2 = \Sigma^* - ((\Sigma^* - L_1) \cup (\Sigma^* - L_2))$ □

Kellerautomaten

Kellerautomaten sind wie nicht-deterministische endliche Automaten, mit einem zusätzlichen Keller:



Der Keller ist wie ein „bodenloser“ Stapel von Zeichen. Der Automat darf bei jedem Schritt beliebig viele Zeichen von diesem Stapel entfernen (aber nur von oben), und beliebig viele Zeichen auf den Stapel legen.

Kellerautomaten formal definiert:

Ein Kellerautomat M besteht aus

- K – einer endlichen Menge von Zuständen
- Σ – einem Eingabealphabet
- Γ – einem Kelleralphabet
- s – einem Startzustand ($s \in K$)
- F – einer Menge von Endzuständen ($F \subseteq K$)
- Δ – einer Übergangsrelation: Eine endliche Teilmenge von $(K \times \Sigma^* \times \Gamma^*) \times (K \times \Gamma^*)$

$\langle\langle p, u, \beta\rangle, \langle q, \gamma\rangle\rangle \in \Delta$ bedeutet, dass, wenn M sich im Zustand p befindet und β der oberste Teil des Kellers ist, der Automat u lesen darf und dabei in den Zustand q übergeht, β vom Keller entfernt und γ auf den Keller legt.

Konfigurationen

Eine Konfiguration eines Kellerautomaten gibt den Zustand, die noch nicht gelesene Eingabe und den Inhalt des Kellers (von oben bis unten) an, z.B.

$\langle p, abba, 001110\rangle$

($\Sigma = \{a, b\}$, $\Gamma = \{0, 1\}$)

Die ergibt-Relation für KA

$\langle q, w, \alpha\rangle \vdash_M \langle q', w', \alpha'\rangle$

gdw

1) $w = uw'$ für ein $u \in \Sigma^*$,

2) $\alpha = \beta\gamma$ und $\alpha' = \beta'\gamma$ für Zeichenketten $\beta, \beta', \gamma \in \Gamma^*$ und

3) $\langle\langle q, u, \beta\rangle, \langle q', \beta'\rangle\rangle \in \Delta$.

$\vdash_M^*$ (lies: *ergibt*) ist wieder die reflexiv-transitive Hülle von $\vdash_M$

Die von einem KA akzeptierte Sprache

Ein Kellerautomat akzeptiert die Eingabe, wenn es eine Berechnung gibt, bei deren Ende der Automat sich mit leerer Resteingabe und leerem Keller in einem Endzustand befindet:

$w \in L(M)$

gdw

es gibt ein $q \in F$ so dass $\langle s, w, \varepsilon\rangle \vdash_M^* \langle q, \varepsilon, \varepsilon\rangle$

Beispiel: Jeder NEA kann als ein KA mit leerem Kelleralphabet betrachtet werden.

Beispiel: Ein Kellerautomat, der die Sprache $\{ww^R: w \in \{a, b\}^*\}$ akzeptiert:

$M = (K, \Sigma, \Gamma, \Delta, s, F)$, wobei

- $K = \{s, f\}$,
- $\Sigma = \Gamma = \{a, b\}$,
- $F = \{f\}$,
- $\Delta = \{ \langle\langle s, a, \varepsilon\rangle, \langle s, a\rangle\rangle,$
  $\langle\langle s, b, \varepsilon\rangle, \langle s, b\rangle\rangle,$
  $\langle\langle s, \varepsilon, \varepsilon\rangle, \langle f, \varepsilon\rangle\rangle,$
  $\langle\langle f, a, a\rangle, \langle f, \varepsilon\rangle\rangle,$
  $\langle\langle f, b, b\rangle, \langle f, \varepsilon\rangle\rangle \}$

Im mittleren Übergang wird „geraten“, dass die Mitte der Eingabe erreicht wurde, und der Automat fängt an, die im Keller gespeicherten Zeichen mit dem Rest der Eingabe zu vergleichen.

Kontextfreie Sprachen und Kellerautomaten

Linksableitungen

Eine direkte Linksableitung ist eine direkte Ableitung, bei der das am weitesten links stehende Nichtterminalsymbol ersetzt wird. Eine Linksableitung ist eine Folge von direkten Linksableitungen.

Notation:

$w \overset{L}{\underset{G}{\Rightarrow}} w'$ für direkte Linksableitungen

$w \overset{L}{\underset{G}{\Rightarrow}}^n w'$ für Linksableitungen in n Schritten

$w \overset{L}{\underset{G}{\Rightarrow}}^* w'$ wenn für ein $n \geq 0$: $w \overset{L}{\underset{G}{\Rightarrow}}^n w'$.

Theorem

Sei G eine kontextfreie Grammatik und $w \in \Sigma^*$.

Dann gilt: $w \in L(G)$ gdw $S \overset{L}{\underset{G}{\Rightarrow}}{}^* w$.

Beweis: Jede Linksableitung ist eine Ableitung. Für die andere Richtung: Wenn $w \in L(G)$, dann gibt es eine Ableitung $S \underset{G}{\Rightarrow}{}^* w$. Zu dieser Ableitung gehört ein Ableitungsbaum, von dem man leicht eine Linksableitung ablesen kann. □

Theorem

Eine Sprache L ist kontextfrei gdw $L = L(M)$ für einen Kellerautomaten M.

Wir werden nur eine Richtung dieses Theorems beweisen. Die folgende Hälfte werden wir nicht beweisen:

Theorem

Wenn $L = L(M)$ für einen Kellerautomaten M, dann ist L kontextfrei.

Der Beweis ist ziemlich umständlich, und bringt für computerlinguistische Zwecke nicht besonders viel.

Zu zeigen bleibt also:

Theorem

Zu jeder kontextfreien Sprache L gibt es einen KA, der L akzeptiert.

Beweis

Sei L eine kontextfreie Sprache. Dann gibt es eine kontextfreie Grammatik $G = \langle V, \Sigma, R, S\rangle$ mit $L(G)=L$. Wir konstruieren einen KA M mit $L(M) = L(G)$ wie folgt:

$M = \langle K, \Sigma, \Gamma, \Delta, s, \{f\}\rangle$, wobei

$K = \{s, f\}$,

$\Gamma = V$,

$$\begin{aligned}\Delta = \quad & \{(\langle s, \varepsilon, \varepsilon\rangle, \langle f, S\rangle)\} \\ & \cup \{(\langle f, \varepsilon, A\rangle, \langle f, x\rangle) : A \underset{G}{\rightarrow} x\} \\ & \cup \{(\langle f, a, a\rangle, \langle f, \varepsilon\rangle) : a \in \Sigma\}\end{aligned}$$

Der Automat simuliert eine Linksableitung: Jede Anwendung einer Regel wird durch das Ersetzen des obersten Kellersymbols mit dem entsprechenden Zeichen aus der Regel simuliert, und wenn danach das oberste Kellersymbol ein Terminalsymbol ist, wird es gelöscht – falls es mit der Eingabe übereinstimmt.

Beispiel: Wir betrachten wieder

$G = \langle \{S, a, b\}, \{a, b\}, \{S \to aSb, S \to \varepsilon\}, S \rangle$

Dann ist $M = \langle \{s, f\}, \{a, b\}, \{S, a, b\}, \Delta, s, \{f\} \rangle$, wobei

$\Delta = \{ \langle \langle s, \varepsilon, \varepsilon \rangle, \langle f, S \rangle \rangle,$
$\langle \langle f, \varepsilon, S \rangle, \langle f, aSb \rangle \rangle,$
$\langle \langle f, \varepsilon, S \rangle, \langle f, \varepsilon \rangle \rangle,$
$\langle \langle f, a, a \rangle, \langle f, \varepsilon \rangle \rangle,$
$\langle \langle f, b, b \rangle, \langle f, \varepsilon \rangle \rangle \}$

und z.B.:

$\langle s, aabb, \varepsilon \rangle \vdash_M \langle f, aabb, S \rangle$
$\vdash_M \langle f, aabb, aSb \rangle$
$\vdash_M \langle f, abb, Sb \rangle$
$\vdash_M \langle f, abb, aSbb \rangle$
$\vdash_M \langle f, bb, Sbb \rangle$
$\vdash_M \langle f, bb, bb \rangle$
$\vdash_M \langle f, b, b \rangle$
$\vdash_M \langle f, \varepsilon, \varepsilon \rangle$

Wir beweisen die folgenden zwei Behauptungen:

(1) Wenn $S \overset{L}{\underset{G}{\Longrightarrow}}{}^* \alpha_1\alpha_2$, wo $\alpha_1 \in \Sigma^*$ und $\alpha_2 \in (V - \Sigma)V^* \cup \{\varepsilon\}$, dann $\langle f, \alpha_1, S \rangle \vdash_M^* \langle f, \varepsilon, \alpha_2 \rangle$.

(2) Wenn $\langle f, \alpha_1, S \rangle \vdash_M^* \langle f, \varepsilon, \alpha_2 \rangle$, wo $\alpha_1 \in \Sigma^*$ und $\alpha_2 \in V^*$, dann $S \overset{L}{\underset{G}{\Longrightarrow}}{}^* \alpha_1\alpha_2$.

(1) und (2) reichen zusammen aus, um das Theorem zu beweisen (setze $\alpha_2 = \varepsilon$ in beiden Fällen.)

Beweis von (1):

Induktion über Ableitungslänge:

Induktionsannahme:

(1) gilt für Ableitungen mit Länge $\leq n$.

Induktionsbasis:

$S \overset{L}{\underset{G}{\Longrightarrow}}{}^* \alpha_1\alpha_2$ ist eine Ableitung in 0 Schritten.

D.h.: $S = \alpha_1\alpha_2$, also $\alpha_1 = \varepsilon$, $\alpha_2 = S$.
$\langle f, \alpha_1, S \rangle \vdash_M^* \langle f, \varepsilon, \alpha_2 \rangle$ ist dann einfach
$\langle f, \varepsilon, S \rangle \vdash_M^* \langle f, \varepsilon, S \rangle$ – trivial!

Induktionsschritt:

Wir betrachten den letzten Schritt einer Linksableitung in $n+1$ Schritten:

Wenn $S \overset{L}{\underset{G}{\Longrightarrow}}{}^{n+1} \alpha_1\alpha_2$, dann $S \overset{L}{\underset{G}{\Longrightarrow}}{}^{n} \beta_1 A \beta_2 \overset{L}{\underset{G}{\Longrightarrow}} \beta_1\gamma\beta_2 = \alpha_1\alpha_2$, wo $\beta_1 \in \Sigma^*$, β_2, $\gamma \in V^*$ und $A \underset{G}{\rightarrow} \gamma$. Weil $\alpha_1 \in \Sigma^*$ ist $\alpha_1 = \beta_1\delta$, wo $\delta \in \Sigma^*$. Und dann ist $\delta\alpha_2 = \gamma\beta_2$.

Für die ersten n Schritte gilt die Induktionsannahme, also gilt

$\langle f, \beta_1, S\rangle \vdash_M^* \langle f, \varepsilon, A\beta_2\rangle$

und deshalb auch

(*) $\langle f, \beta_1\delta, S\rangle \vdash_M^* \langle f, \delta, A\beta_2\rangle$

Es gilt:

$$\begin{aligned}
\langle f, \alpha_1, S\rangle &= \langle f, \beta_1\delta, S\rangle && [\alpha_1 = \beta_1\delta] \\
&\vdash_M^* \langle f, \delta, A\beta_2\rangle && [(*)] \\
&\vdash_M \langle f, \delta, \gamma\beta_2\rangle && [A \underset{G}{\rightarrow} \gamma] \\
&\vdash_M^* \langle f, \varepsilon, \alpha_2\rangle && [\delta\alpha_2 = \gamma\beta_2]
\end{aligned}$$

und damit ist der Induktionsschritt vollständig.

Beweis für (2):

Induktion über die Länge einer M-Berechnung:

Induktionsannahme:

(2) gilt für Berechnungen in $\leq n$ Schritten.

Induktionsbasis:

$\langle f, \alpha_1, S\rangle \vdash_M^* \langle f, \varepsilon, \alpha_2\rangle$ in 0 Schritten, d.h. $\alpha_1 = \varepsilon$, $\alpha_2 = S$, und dann gilt $S \overset{L}{\underset{G}{\Rightarrow}}{}^* \alpha_1\alpha_2$

Induktionsschritt:

Wenn $\langle f, \alpha_1, S\rangle \vdash_M^{n+1} \langle f, \varepsilon, \alpha_2\rangle$, dann $\langle f, \alpha_1, S\rangle \vdash_M^n \langle f, \beta, \gamma\rangle \vdash_M \langle f, \varepsilon, \alpha_2\rangle$, wobei $\beta \in \Sigma^*$ und $\gamma \in \Gamma^*$.

Der zuletzt benützte Übergang war entweder

Fall (1): $\langle\langle f, \beta, \beta\rangle, \langle f, \varepsilon\rangle\rangle$, wobei $\beta \in \Sigma$

Dann ist $\alpha_1 = \delta\beta$ (β ist das letzte Zeichen der Eingabe) für $\delta \in \Sigma^*$, $\gamma = \beta\alpha_2$, und:

$\langle f, \alpha_1, S\rangle = \langle f, \delta\beta, S\rangle \vdash_M^n \langle f, \beta, \beta\alpha_2\rangle \vdash_M \langle f, \varepsilon, \alpha_2\rangle$

Dann gilt aber auch:

$\langle f, \delta, S\rangle \vdash_M^n \langle f, \varepsilon, \beta\alpha_2\rangle$,

und für diese Berechnung gilt die Induktionsannahme:

$S \overset{L}{\underset{G}{\Rightarrow}}{}^* \delta\beta\alpha_2 = \alpha_1\alpha_2$.

Fall (2): $\langle\langle f, \varepsilon, A\rangle, \langle f, \delta\rangle\rangle$, wobei $A \rightarrow_G \delta$

Im letzten Schritt wird keine Eingabe gelesen, deshalb ist $\beta = \varepsilon$, $\gamma = A\gamma_1$ für ein $\gamma_1 \in \Gamma^*$, und $\alpha_2 = \delta\gamma_1$.

Weil $\beta = \varepsilon$, kann die Induktionsannahme direkt auf die n ersten Berechnungsschritte angewendet werden, und es gilt:

$S \overset{L}{\underset{G}{\Rightarrow}}{}^* \alpha_1 A\gamma_1 \Rightarrow \alpha_1\delta\gamma_1 = \alpha_1\alpha_2$. □

Theorem

Sei L kontextfrei und R regulär.

Dann ist $L \cap R$ kontextfrei.

Beweis

Zu L gibt es einen KA M_1, der L akzeptiert.

Zu R gibt es einen DEA M_2, der R akzeptiert.

Wir werden aus diesen Automaten einen KA M konstruieren, der $L \cap R$ akzeptiert. Weil jede von einem KA akzeptierte Sprache kontextfrei ist (nicht bewiesen), ist dann auch $L \cap R$ kontextfrei: M ist wie M_1, speichert aber gleichzeitig, in welchem Zustand M_2 sich bei gleich viel gelesener Eingabe befindet (es ist wichtig, dass M_2 deterministisch ist!):

Sei $M = \langle K, \Sigma, \Gamma_1, \Delta, s, F\rangle$, wobei

$K = K_1 \times K_2$,

$\Sigma = \Sigma_1 \cup \Sigma_2$,

$s = \langle s_1, s_2\rangle$,

$F = F_1 \times F_2$,

und

$\langle\langle q_1, q_2\rangle, u, \beta\rangle, \langle\langle p_1, p_2\rangle, \gamma\rangle\rangle \in \Delta$

gdw

$\langle\langle q_1, u, \beta\rangle, \langle p_1, \gamma\rangle\rangle \in \Delta_1$ und $\langle q_2, u\rangle \vdash_{M2}^{*} \langle p_2, \varepsilon\rangle$ □

Beispiel

$L = \{w \in \{a, b, c\}^* : \#a \text{ in } w = \#b \text{ in } w = \#c \text{ in } w\}$

ist nicht kontextfrei.

Beweis: Wenn L kontextfrei wäre, dann auch $L \cap a^*b^*c^* = \{a^n b^n c^n : n \geq 0\}$. □

Entscheidbarkeit

Sei G eine kontextfreie Grammatik.

Können wir die Frage „Ist w ∈ L(G)?“ beantworten? Statt eine direkte Antwort zu geben, werden wir zeigen das es zu G einen *Parser* gibt. Die Antwort ja folgt dann unmittelbar.

Parsing

Was ist Parsing?

> **Analyse, Syntaktische.** (Parsing) Verfahren zur Strukturanalyse von Sätzen einer formalen oder natürlichen Sprache mit Hilfe der zugrundeliegenden Grammatik. (…)

Bei Parsing interessiert uns also nicht nur die Frage, ob eine Zeichenkette zu einer Sprache gehört, sondern auch welchen Ableitungsbaum (oder, wenn die Grammatik mit Absicht mehrdeutig ist, wie es bei natürlichen Sprachen oft der Fall ist, Ableitungsbäume) die Zeichenkette in der gegebenen Grammatik hat.

Im Folgenden werden wir uns der Frage widmen, inwieweit Kellerautomaten, die von Grammatiken abgeleitet sind, als Parser benutzt werden können.

Wir haben schon gezeigt, dass wir zu jeder Grammatik einen KA erzeugen können, der die von der Grammatik generierte Sprache akzeptiert. Wir betrachten jetzt einen Automaten, der durch diese Methode entstanden ist.

Beispiel

Von der Grammatik

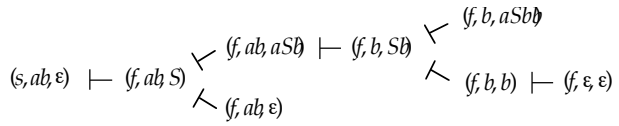
$G = \langle \{a, b, S\}, \{a, b\}, \{S \to aSb, S \to \varepsilon\}, S\rangle$,

die $L = \{a^n b^n : n \geq 0\}$ erzeugt, wird folgender Automat konstruiert:

$M = \langle \{s, f\}, \{a, b\}, \{S, a, b\}, \Delta, s, \{f\}\rangle$ mit

$\Delta = \{ \langle (s, \varepsilon, \varepsilon), (f, S)\rangle,$
$\langle (f, \varepsilon, S), (f, aSb)\rangle,$
$\langle (f, \varepsilon, S), (f, \varepsilon)\rangle,$
$\langle (f, a, a), (f, \varepsilon)\rangle,$
$\langle (f, b, b), (f, \varepsilon)\rangle \}$

Die beiden Transitionen, die den Regeln entsprechen, sind kompatibel, also ist M nicht deterministisch. Trotzdem ist M ein anwendbarer Parser für L: Für jede Eingabe gibt es nur endlich viele, endlich lange Berechnungen. Z.B. sind bei Eingabe ab nur die folgenden Berechnungen möglich:



Streng genommen ist M kein Parser, weil er keine Ausgabe gibt. Eine Linksableitung – und damit ein Ableitungsbaum – kann aber von den Schritten der Berechnung, die Regelanwendungen entsprechen, direkt abgelesen werden ($S \Rightarrow aSb \Rightarrow ab$). Einige Lehrbücher führen Transducer-Kellerautomaten ein, die Regelanwendungen auf ein Ausgabeband schreiben.

Top-down-Parser

Den Automaten aus dem Beweis „k.f.G. → KA“ nennen wir den *Top-down-Kellerautomaten* zu G. Für die Beispielgrammatik war dieser KA direkt als Top-down-Parser anwendbar. Im allgemeinen können wir aber nicht sicher sein, dass der konstruierte Automat bei jeder Eingabe hält.

Wir stellen jetzt fest, dass der Automat genau dann in unendliche Schleifen gerät, wenn die Grammatik *linksrekursiv* ist:

Definition

Eine Grammatik ist *linksrekursiv* gdw sie ein vom Startsymbol erreichbares linksrekursives Nichtterminalsymbol hat, d.h. gdw

es gibt ein $A \in V - \Sigma$ und $x, y, u \in V^*$ so dass:

$$S \underset{G}{\Rightarrow}^* xAy \underset{G}{\Rightarrow}^+ xAu$$

Beispiel

Wir werden folgende Beispielgrammatik

$G = \langle \{S, L, (,), a, \_\}, \{(,), a, \_\}, R, S \rangle$

benutzen, wobei R die folgenden Regeln enthält:

$R = \{\ S \to (),$
$\quad S \to a,$
$\quad S \to (L),$
$\quad L \to S,$
$\quad L \to L\_S\ \}$

G stellt die Syntax von „S-Expressions“ dar, die der Hauptbestandteil der Syntax der Programmiersprache LISP sind. (Hier verwenden wir a stellvertretend für Symbole und _ für “whitespace.”)

Wir betrachten jetzt die Linksrekursion in der zweiten L-Regel ($L \to L\_S$). Linksrekursion heißt hier, dass das Zeichen der linken Seite einer Regel als am weitesten links stehendes Zeichen auf der rechten Seite vorkommt.

Der von unserer Beispielgrammatik direkt konstruierte Parser hat die folgende Transition:

$\langle (f, \varepsilon, L), (f, L\_S) \rangle$

Diese Transition ist beliebig viele Male nacheinander anwendbar, deshalb ist dieser Automat kein Parser.

Allgemein können wir beweisen:

Theorem

Sei G eine Grammatik, und sei M der entsprechende Top-down-Kellerautomat. Dann gibt es zu jeder Eingabe nur endlich viele, endlich lange Berechnungen gdw G nicht linksrekursiv ist.

Beweis

Wenn G linksrekursiv ist, hat der Automat Teilberechnungen, die folgendermaßen aussehen:

$$\langle f, x, A\rangle \vdash_M^* \langle f, x, A\alpha\rangle$$

wobei A vom S aus erreichbar ist. M hat dann die Möglichkeit, diese Teilberechnung beliebig viele Male durchzuführen, und kann deshalb in eine unendliche Schleife geraten.

Umgekehrt: M legt zuerst das Startsymbol S auf den Keller, und tauscht nachher das oberste Kellersymbol um, bis ein Terminalsymbol als oberstes Kellersymbol vorkommt. Dann wird das entsprechende Zeichen von der Eingabe gelesen. Wenn G nicht linksrekursiv ist, ist jede solche Teilberechnung höchstens $|V - \Sigma|$ lang, also hat keine Berechnung mehr als $|w| \times |V - \Sigma|$ Schritte. Deshalb gibt es auch nur endlich viele mögliche Berechnungen. □

Zurück zum Beispiel:

Es ist klar, dass die folgenden Regeln genau die selben Ausdrücke generieren wie die in der Grammatik gegebenen L-Regeln:

$$L \to SL'$$
$$L' \to \_SL'$$
$$L' \to \varepsilon$$

Im allgemeinen läßt sich Linksrekursion innerhalb von Regeln folgendermaßen entfernen:

Methode zum Entfernen von Linksrekursion:

Sei G eine Grammatik mit den folgenden A-Regeln (wir nehmen an, die Regel $A \to A$ kommt nicht vor):

$$A \to A\alpha_1, \ldots, \quad A \to A\alpha_n,$$

$$A \to \beta_1, \ldots, \quad A \to \beta_m,$$

wobei $n>0$ und kein β fängt mit einem A an. Ersetze dann diese Regeln durch die Regeln

$$A \to \beta_1 A', \ldots, A \to \beta_m A'$$

$$A' \to \alpha_1 A', \ldots, A' \to \alpha_n A'$$

$$A' \to \varepsilon$$

wobei A' ein neues Nichtterminalsymbol ist.

Theorem

Sei G eine kontextfreie Grammatik, und G' die Grammatik, die von G durch Anwendung der obigen Methode entsteht. Dann ist $L(G') = L(G)$.

Beweis

$L(G') \supseteq L(G)$: Betrachte folgende Linksableitung von einem beliebigen Wort w in G:

$$S \overset{L}{\underset{G}{\Rightarrow}} \gamma_0 \overset{L}{\underset{G}{\Rightarrow}} \gamma_1 \overset{L}{\underset{G}{\Rightarrow}} \ldots \overset{L}{\underset{G}{\Rightarrow}} \gamma_k = w$$

Ein Ableitungsschritt, in dem eine der "β-Regeln" angewandt wurde, d.h., wo

$$\gamma_j = xA\gamma' \text{ und } \gamma_{j+1} = x\beta_i\gamma',\ i \leq m,$$

läßt sich durch die beiden Schritte

$$xA\gamma' \overset{L}{\underset{G'}{\Rightarrow}} x\beta_iA'\gamma' \overset{L}{\underset{G'}{\Rightarrow}} x\beta_i\gamma'$$

ersetzen.

Eine Folge von l Ableitungsschritten, in denen linksrekursive Regeln angewandt wurden, sieht im allgemeinen folgendermaßen aus:

$$\begin{aligned}(1)\ xA\gamma' &\overset{L}{\underset{G}{\Rightarrow}} xA\alpha_{j1}\gamma' \\ &\overset{L}{\underset{G}{\Rightarrow}} \ldots \\ &\overset{L}{\underset{G}{\Rightarrow}} xA\alpha_{jl}\ldots\alpha_{j1}\gamma' \\ &\overset{L}{\underset{G}{\Rightarrow}} x\beta_p\alpha_{jl}\ldots\alpha_{j1}\gamma'\end{aligned}$$

und kann mit

$$\begin{aligned}(2)\ xA\gamma' &\overset{L}{\underset{G'}{\Rightarrow}} x\beta_pA'\gamma' \\ &\overset{L}{\underset{G'}{\Rightarrow}} x\beta_p\alpha_{jl}A'\gamma' \\ &\overset{L}{\underset{G'}{\Rightarrow}} \ldots \\ &\overset{L}{\underset{G'}{\Rightarrow}} x\beta_p\alpha_{jl}\ldots\alpha_{j1}\gamma'\end{aligned}$$

ersetzt werden.

$L(G') \subseteq L(G)$: Wenn w aus S G'-ableitbar ist, müssen wir Teilableitungen, die A' enthalten, entfernen. Solche sehen aber wie (2) (möglicherweise mit $l=0$) aus, und wir können sie mit Ableitungen wie (1) ersetzen.□

Indirekte Linksrekursivität

Linksrekursivität kommt nicht nur in einzelnen Regeln vor. Z.B. ist A in den folgenden Regeln linksrekursiv:

$$A \to Bc,\ B \to Aa$$

Es gibt aber eine Erweiterung des obigen Algorithmus, die jede Art von Linksrekursion beseitigt (wird nicht vorgestellt). Also gilt:

Theorem

Zu jeder kontextfreien Sprache gibt es eine Grammatik ohne Linksrekursion.

Bottom-up-Verfahren: Shift-reduce-Parser

Statt zur Eingabe w von S aus nach Linksableitungen zu suchen, können wir die umgekehrte Strategie wählen: Von w aus Rechtsableitungen rekonstruieren. Dies nennt man Bottom-up-Parsing, weil man den Ableitungsbaum von unten, vom Ertrag aus, konstruiert.

Wir können statt den schon vertrauten Top-down-Kellerautomaten einen Bottom-up-Kellerautomaten konstruieren:

Von der Grammatik $G = \langle V, \Sigma, R, S\rangle$ konstruieren wir den Automaten $M = \langle \{s, f\}, \Sigma, V, \Delta, s, \{f\}\rangle$, wo Δ die folgenden Transitionen hat:

Für jede $\sigma \in \Sigma$: $\langle\langle s, \sigma, \varepsilon\rangle, \langle s, \sigma\rangle\rangle$ (“shift”)

Für jede $A \to \alpha \in R$: $\langle\langle s, \varepsilon, \alpha^R\rangle, \langle s, A\rangle\rangle$ (“reduce”)

und schließlich: $\langle\langle s, \varepsilon, S\rangle, \langle f, \varepsilon\rangle\rangle$

M benutzt ein sogenanntes shift-reduce-Verfahren: Er hat zwei Typen von Bewegungen zur Wahl: Eine Möglichkeit ist, dass M ein Zeichen von der Eingabe liest und auf den Keller legt (wenn nur dies gemacht wird, enthält der Keller schließlich die ganze Eingabe in umgekehrter Reihenfolge). Die andere Möglichkeit ist, dass die obersten Zeichen des Kellers der rechten Seite einer Regel (gespiegelt) entsprechen. Dann kann M diese Zeichen mit der entsprechenden linken Seite austauschen.

Beispiel

Von der LISP-Grammatik (in der ursprünglichen Version) erhalten wir folgenden Automaten:

$M = \langle \{s, f\}, \{(,), a, \_\}, \{S, L, (,), a, \_\}, \Delta, s, \{f\}\rangle$, wobei

$\Delta = \{$ $\langle\langle s, (, \varepsilon\rangle, \langle s, (\rangle\rangle$, *[shift ( ]*
$\langle\langle s,), \varepsilon\rangle, \langle s,)\rangle\rangle$, *[shift)]*
$\langle\langle s, a, \varepsilon\rangle, \langle s, a\rangle\rangle$, *[shift a]*
$\langle\langle s, \_, \varepsilon\rangle, \langle s, \_\rangle\rangle$, *[shift _]*
(5) $\langle\langle s, \varepsilon,)(\rangle, \langle s, S\rangle\rangle$, *[reduce: () ← S]*
(6) $\langle\langle s, \varepsilon, a\rangle, \langle s, S\rangle\rangle$, *[reduce: a ← S]*
(7) $\langle\langle s, \varepsilon,)L(\rangle, \langle s, S\rangle\rangle$, *[reduce: (L) ← S]*
(8) $\langle\langle s, \varepsilon, S\rangle, \langle s, L\rangle\rangle$, *[reduce: S ← L]*

(9) $\langle\langle s, \varepsilon, S\_L\rangle, \langle s, L\rangle\rangle$, $\langle\langle s, \varepsilon, S\rangle, \langle f, \varepsilon\rangle\rangle$ } [reduce: L_S ← L]

Z.B. macht M u.a. die folgende Berechnung bei Eingabe *(()_a)*, d.h. mit Startkonfiguration $\langle s, (()\_a), \varepsilon\rangle$:

Erst dreimal Shift:

$\vdash_M \langle s, ()\_a), ()\rangle$
$\vdash_M \langle s,)\_a), (()\rangle$
$\vdash_M \langle s, \_a),)(()\rangle$

Dann zweimal Reduce, mit den Transitionen (5), (8):

$\vdash_M \langle s, \_a), S()\rangle$
$\vdash_M \langle s, \_a), L()\rangle$

Zweimal Shift:

$\vdash_M \langle s, a), \_L()\rangle$
$\vdash_M \langle s,), a\_L()\rangle$

Und zweimal Reduce, mit den Transitionen (6), (9):

$\vdash_M \langle s,), S\_L()\rangle$
$\vdash_M \langle s,), L()\rangle$

Schließlich ein Shift, ein Reduce und fertig:

$\vdash_M \langle s, \varepsilon,)L()\rangle$
$\vdash_M \langle s, \varepsilon, S\rangle$
$\vdash_M \langle f, \varepsilon, \varepsilon\rangle$

Hält der Automat bei jeder Eingabe?

Man sieht, dass M bei jeder Eingabe nur endlich viele, endlich lange Berechnungen macht: Es besteht zwar die Möglichkeit zwischen einem Shift und einem Reduce zu wählen. Aber man kann nur so viel Shifts machen wie es Zeichen in der Eingabe gibt. Ein Problem gäbe es, wenn es möglich wäre, beliebig viele Male zu reduzieren, ohne dass die Zahl der Zeichen auf dem Keller geringer würde. Bei dieser Grammatik ist das kein Problem: Die einzigen Transitionen, in denen der Kellerinhalt nicht kürzer wird, sind (6) und (8), aber ein L kann danach nicht weiter reduziert werden.

Im allgemeinen müssen wir verlangen, dass G nicht zirkulär ist, d.h. $A \Rightarrow_G^+ A$ soll für kein $A \in V - \Sigma$ gelten.

Ein anderes Problem besteht, wenn G ε-Regeln enthält. Die kann man ja immer anwenden, und deshalb den Keller beliebig vergrößern. Deshalb werden wir die hier nicht zulassen. (Es ist aber möglich, ε-Regeln bis zu einem gewissen Grad zuzulassen!)

Wir können jetzt folgendes feststellen:

Theorem

Wenn eine Grammatik ε-frei und nicht-zirkulär ist, macht der entsprechende Bottom-up-Kellerautomat bei jeder Eingabe nur endlich viele, endlich lange Berechnungen. □

Theorem

Sei G eine kontextfreie Grammatik, und M der entsprechende Bottom-up-Kellerautomat. Dann gilt $L(M) = L(G)$.

Beweis

Wie fast immer bei Induktionsbeweisen, beweisen wir eine etwas stärkere Aussage:

Für $x \in \Sigma^*$ und $\gamma \in \Gamma^*$ gilt:

$$\langle s, x, \gamma\rangle \vdash_M^* \langle s, \varepsilon, S\rangle \text{ gdw } S \overset{R}{\underset{G}{\Longrightarrow}}{}^* \gamma^R x.$$

„⇒“ Induktion über Berechnungslänge:

Induktionsannahme:

Die ⇒-Richtung der Behauptung gilt für Berechnungen mit bis zu k Schritten.

Induktionsbasis:

$k = 0$ bedeutet, dass $x = \varepsilon$ und $\gamma = S$, und $S \overset{R}{\underset{G}{\Longrightarrow}}{}^* \gamma^R x$ gilt.

Induktionsschritt:

Eine Berechnung in $k + 1$ Schritten sieht so aus:

$$\langle s, x, \gamma\rangle \vdash_M \langle s, w, \beta\rangle \vdash_M^k \langle s, \varepsilon, S\rangle$$

Nach der Induktionsannahme gilt $S \overset{R}{\underset{G}{\Longrightarrow}}{}^* \beta^R w$.

Für den ersten Schritt müssen wir 2 Fälle unterscheiden:

1) Er ist ein Shift-Schritt: Dann ist $x = \sigma w$ und $\beta = \sigma\gamma$ für ein Terminalsymbol σ, und $\gamma^R x = \gamma^R \sigma w = \beta^R w$.

2) Er ist ein Reduce-Schritt: Dann ist $x = w$ und es gibt eine Regel $A \to \alpha$ und ein $\delta \in \Gamma^*$ so dass $\beta = A\delta$ und $\gamma = \alpha^R \delta$. Es gilt dann

$$S \overset{R}{\underset{G}{\Longrightarrow}}{}^* \beta^R w = \delta^R A w = \delta^R A x \overset{R}{\underset{G}{\Longrightarrow}} \delta^R \alpha x = \gamma^R x.$$

„⇐“ Induktion über die Länge einer Rechtsableitung von $\gamma^R x$:

Induktionsannahme:

Die ⇐-Richtung der Behauptung gilt für Rechtsableitungen mit bis zu k Schritten.

Induktionsbasis:

k = 0: Dann ist $x = \varepsilon$ und $\gamma = S$, und $\langle s, x, \gamma\rangle \vdash_M^* \langle s, \varepsilon, S\rangle$.

Induktionsschritt:

Eine Ableitung in $k + 1$ Schritten sieht so aus:

$$S \overset{R}{\underset{G}{\Longrightarrow}}{}^{k} \beta^R A w \overset{R}{\underset{G}{\Longrightarrow}} \beta^R \alpha w = \gamma^R x$$

Die Induktionsannahme gibt uns, dass

$$\langle s, w, A\beta\rangle \vdash_M^* \langle s, \varepsilon, S\rangle$$

Weil $|x| \geq |w|$ (Rechtsableitung!) gibt es ein u so dass $\beta^R\alpha = \gamma^R u$ und $uw = x$. Ein Reduce-Schritt gibt uns $\langle s, w, \alpha^R\beta\rangle \vdash_M \langle s, w, A\beta\rangle$, und $|u|$ Shift-Schritte geben uns:

$$\langle s, x, \gamma\rangle = \langle s, uw, \gamma\rangle \vdash_M^* \langle s, w, u^R\gamma\rangle = \langle s, w, \alpha^R\beta\rangle$$

Also gilt $\langle s, x, \gamma\rangle \vdash_M^* \langle s, \varepsilon, S\rangle$.

Jetzt wissen wir, dass der Bottom-up Parser *korrekt* ist. Wir stellen nun fest, dass jede kontextfreie Sprache eine Grammatik hat, die sich für Bottom-up Parsing eignet:

Lemma 1

Es gibt einen Algorithmus, der jede kontextfreie Grammatik G in eine kontextfreie Grammatik G' transformiert, so dass $L(G') = L(G)$ und G' die folgenden Eigenschaften hat:

1) für jedes Wort außer ε gibt es Ableitungen, in denen Regeln mit leeren rechten Seiten nicht angewendet werden.

2) Falls $\varepsilon \in L(G)$ hat G' die Regel $S \to \varepsilon$.

Beweis

Der Algorithmus läuft folgendermaßen:

Initialisierung: $G' = G$.

1. Finde zwei Regeln der Typen $A \to \varepsilon$ bzw. $B \to uAv$ aus R', wobei $B \to uv$ nicht in R' ist.
2. Wenn kein Regelpaar in Schritt 1 gefunden wurde, sind wir fertig. Sonst setzte $R' := R' \cup \{B \to uv\}$ und wiederhole Schritt 1.

Der Algorithmus terminiert, weil jede neue Regel auf der rechten Seite eine Zeichenkette hat, die durch Entfernung von einem Zeichen aus einer schon vorhandenen Regel gebildet wurde – der Algorithmus terminiert spätestens, wenn jede mögliche solche Zeichenkette gebildet worden ist (und das sind endlich viele).

Der Algorithmus ist korrekt, weil jede G-Ableitung, in der eine ε-Regel verwendet wird, folgendermaßen aussieht (für $w \neq \varepsilon$):

$$S \underset{G}{\Rightarrow}^* xBy \underset{G}{\Rightarrow} xuAvy \underset{G}{\Rightarrow} xuvy \underset{G}{\Rightarrow}^* w$$

In G' wird es die Regel $B \to uv$ geben, also können wir auf die ε-Regel verzichten. Für den Fall $w = \varepsilon$ ist es klar, dass G' die Regel $S \to \varepsilon$ haben wird.

Umgekehrt ist auch jede G'-Ableitung durch eine G-Ableitung ersetzbar – nichts neues wird generiert. □

Lemma 2

Es gibt einen Algorithmus, der jede kontextfreie Grammatik G in eine kontextfreie Grammatik G' transformiert, so dass $L(G') = L(G)$, und wo G' die folgende Eigenschaft hat:

Für jedes Wort gibt es Ableitungen, in denen Regeln vom Typ $A \to B$, wobei $A, B \in V\text{–}\Sigma$, nicht angewandt werden.

Beweis

Erstens, es ist für $A, B \in V\text{–}\Sigma$ entscheidbar ob $A \underset{G}{\Rightarrow}^* B$, weil solche Ableitungen immer in weniger als $|V\text{–}\Sigma|$ Schritten durchführbar sind.

Wir wenden dann den folgenden einfachen Algorithmus an:

Für jede Regel $A \to_{G'} u$:
    Für jedes B so dass $B \Rightarrow^*_{G'} A$:
        Führe $B \to_{G'} u$ als neue Regel ein. □

Es ist klar, dass wir die oben erzeugten Grammatiken wieder „abspecken“ können, indem wir die $A \to \varepsilon$ Regeln (außer $S \to \varepsilon$) und die $A \to B$ Regeln entfernen.

Bottom-up Parsing vs. Top-down Parsing

Jetzt wissen wir dass der Bottom-up-Kellerautomat *korrekt* ist. Außerdem wissen wir, dass es zu jeder kontextfreien Grammatik möglich ist, eine äquivalente nicht-zirkuläre, ε-freie Grammatik zu finden (außer eventuell $S \to \varepsilon$, man kann dies als Sonderfall behandeln). Also ist das Bottom-up Verfahren für jede Grammatik anwendbar.

Tatsächlich hat das Bottom-up Verfahren einen kleinen Vorteil: Wenn ein G' aus G durch das Entfernen von ε-Regeln und Zirkularität entstanden ist, kann man von einem G'-Ableitungsbaum immer einen G-Ableitungsbaum rekonstruieren. Für Grammatiken,

die linksrekursionsfrei gemacht worden sind, ist dies nicht immer möglich (wird hier nicht gezeigt).

Bessere Parsingalgorithmen

Die generellen Top-down- und Bottom-up-Verfahren haben beide exponentielle Komplexität: die Parsingzeit beträgt c^n, wobei n die Eingabelänge und c irgendeine Konstante ist.

In natürlich-sprachlichen kontextfreien Parsern werden oft allgemein einsetzbare Algorithmen mit Komplexität cn^3 (oder noch besser), wie z.B. Earleys Algorithmus, eingesetzt. Für Programmiersprachen werden meistens deterministische Bottom-up-Parser mit linearer Komplexität benutzt.

Kontextfreie Grammatiken für natürliche Sprachen

Bei Grammatiken für natürliche Sprachen teilt man die Nichtterminalsymbole oft in *nichtlexikalische Kategorien* und *Präterminale* oder *lexikalische Kategorien* auf. D.h., man begrenzt sich auf Regeln, die entweder nur Nichtterminalsymbole enthalten (oft nur die Regeln genannt) oder ein einziges Terminalsymbol auf der rechten Seite (diese Teilmenge der Regeln nennt man das Lexikon).

Beispiel:

G = ⟨V, Σ, R, S⟩, wobei:

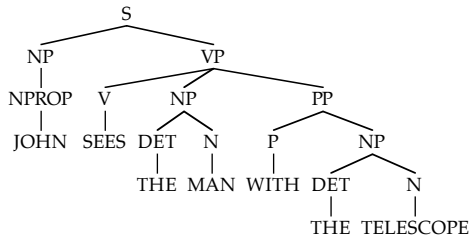
$V = \{S, NP, VP, PP, N, NPROP, V, P, Det\} \cup \Sigma$.

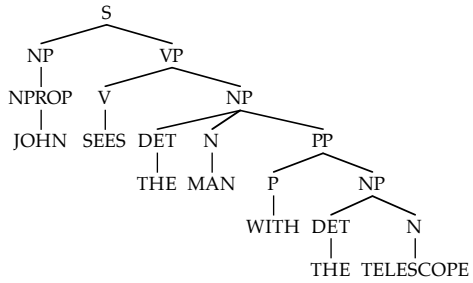
$\Sigma = \{John, the, man, telescope, sees, with, ...\}$

und R enthält die folgenden Regeln

$S \rightarrow NP\ VP$	$PP \rightarrow P\ NP$	$N \rightarrow man$
$VP \rightarrow V$	$NP \rightarrow Nprop$	$N \rightarrow telescope$
$VP \rightarrow V\ PP$	$NP \rightarrow Det\ N$	$V \rightarrow sees$
$VP \rightarrow V\ NP$	$NP \rightarrow Det\ N\ PP$	$P \rightarrow with$
$VP \rightarrow V\ NP\ PP$	$NPROP \rightarrow John$	$Det \rightarrow the$

Beispiel: Zwei Ableitungsbäume für *John sees the man with the telescope:*





Hier hat Mehrdeutigkeit in der Grammatik einen Sinn!

Ein weiteres entscheidbares Problem

Wir wissen jetzt, dass die Frage „Ist $w \in L(G)$?“ für kontextfreie G entscheidbar ist.

Hier kommt noch ein entscheidbares Problem:

Theorem

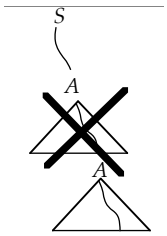
Es gibt einen Algorithmus, der für jede G die Frage

Ist $L(G) = \varnothing$?

beantwortet.

Beweis

Wenn $L(G) \neq \varnothing$ gibt es einen Ableitungsbaum mit Höhe $\leq |V–\Sigma|$, und ein Wort in der Sprache als Ertrag, weil höhere Bäume einen Pfad haben, in dem das selbe Nichtterminalsymbol mehr als einmal vorkommt. Das Teilbäumchen das zwischen diesen beiden Vorkommen erzeugt wird, läßt sich entfernen, und der Ertrag bleibt ein Wort der Sprache:



Der Algorithmus ist dann einfach: Generiere jeden Ableitungsbaum der Grammatik, der nicht höher als $|V–\Sigma|$ ist, und überprüfe, ob dessen Ertrag ein Wort über Σ ist. □

Unentscheidbare Probleme der kontextfreien Sprachen

Es gibt zu den kontextfreien Sprachen erstaunlich viele Probleme, für die es keinen allgemeinen Entscheidungsalgorithmus gibt:

Theorem

Die folgenden Probleme sind unentscheidbar (für L kontextfreie Sprache, G, G_1, G_2 Grammatiken, R reguläre Sprache):

- Ist $L(G) = \Sigma^*$?
- Ist $L(G_1) = L(G_2)$?
- Ist $L(G_1) \subseteq L(G_2)$?
- Ist $L(G) = R$?
- Ist $R \subseteq L(G)$?
- Ist $L(G)$ regulär?
- Ist L mehrdeutig?
- Ist $\Sigma^* - L(G)$ kontextfrei?
- Ist $L(G_1) \cap L(G_2)$ kontextfrei?

Beweis

Nicht durchgeführt, da er viel Turingmaschinen-Theorie voraussetzt.

Deterministische Kellerautomaten

Ein Kellerautomat heißt *deterministisch* (DKA) wenn er zu jeder Konfiguration *höchstens* (nicht *genau*, wie bei DEA) eine mögliche Transition hat.

Für die formale Definition der DKA brauchen wir einige Hilfsdefinitionen:

Zwei Zeichenketten sind *konsistent* gdw die eine ein Präfix der anderen ist. (Also ist w konsistent mit wu, und ε ist mit jeder Zeichenkette konsistent.)

Zwei Transitionen

$\langle\langle p, w_1, \gamma_1\rangle, \langle q_1, \delta_1\rangle\rangle$ und

$\langle\langle p, w_2, \gamma_2\rangle, \langle q_2, \delta_2\rangle\rangle$

sind *kompatibel* wenn w_1 und w_2 konsistent sind und γ_1 und γ_2 konsistent sind.

Definition

Ein Kellerautomat M ist *deterministisch* gdw M keine zwei unterschiedlichen Transitionen hat, die kompatibel sind.

Deterministisch kontextfreie Sprachen

Eine kontextfreie Sprache L heißt *deterministisch* gdw $L\$ = L(M)$ für einen DKA M, wobei $\$$ ein neues Zeichen ist.[2]

Der Grund für die kleine Komplikation bei der Definition ist, dass man gerne Sprachen wie

$$a^* \cup \{a^n b^n : n \geq 0\}$$

als deterministisch bezeichnen möchte, ein DKA nach unserer Definition kann aber nicht bei Eingabe *aaa* den Keller leeren, bevor die ganze Eingabe gesehen wurde, weil ja die Eingabe auch *aaabbb* hätte sein können, und dann ist es „zu spät." Deshalb führen wir die Eingabe-Ende-Markierung $\$$ ein.

Theorem

Die regulären Sprachen bilden eine echte Teilmenge der deterministisch kontextfreien Sprachen.

Beweis

Jeder DEA kann als ein DKA betrachtet werden, der seinen Keller nicht benutzt. Auf der anderen Seite gibt es Sprachen wie $\{a^n b^n : n \geq 0\}$, die deterministisch, aber nicht regulär sind. □

Theorem

Die deterministisch kontextfreien Sprachen sind unter Komplementbildung abgeschlossen.

Beweis-Skizze

Intuitiv kann man wie bei den DEAs einfach einen Automaten konstruieren, in dem Endzustände und Nicht-Endzustände umgetauscht wurden. Folgende mögliche Eigenschaften des DKAs macht dies nicht ohne weiteres möglich:

1. Zustände, von denen aus es keine gültigen Übergänge mehr gibt.
2. Schleifen, wo keine Eingabe gelesen wird.
3. ε-Bewegungen zwischen End- und nicht-Endzuständen.

Wir werden hier nicht beweisen, dass diese Probleme sich überwinden lassen.

---

2 Die Definition stammt von Lewis & Papadimitriou. Eine andere mögliche Definition ist: Eine kontextfreie Sprache L heißt deterministisch gdw $L = L_f(M)$ für einen DKA M. Hier ist $L_f(M)$ die Sprache, die M im Endzustand – aber nicht notwendigerweise mit leerem Keller – akzeptiert. Diese Definition wird von Hopcroft & Ullman benutzt; ihr $L(M)$ (für Kellerautomaten im Allgemeinen) ist unser $L_f(M)$.

Theorem

Die deterministisch kontextfreien Sprachen bilden eine echte Teilmenge der kontextfreien Sprachen.

Beweis

Jede deterministisch kontextfreie Sprache ist auch kontextfrei. Aber nicht jede kontextfreie Sprache ist deterministisch kontextfrei, es gibt ja kontextfreie Sprachen mit einem Komplement das nicht kontextfrei – und dadurch auch nicht deterministisch kontextfrei – ist. □

Die letzte Aussage läßt sich auch konstruktiv bestätigen, wir werden folgendes zeigen:

Lemma

Das Komplement der Sprache $L_{abc} = \{a^n b^n c^n : n \geq 0\}$ ist kontextfrei.

Beweis

$\Sigma^* - L_{abc} = (\Sigma^* - a^*b^*c^*) \cup (a^*b^*c^* - L_{abc})$

$\Sigma^* - a^*b^*c^*$ ist regulär.

$a^*b^*c^* - L_{abc} =$
$\{a^m b^n c^p : m>n\} \cup \{a^m b^n c^p : n>m\} \cup$
$\{a^m b^n c^p : n>p\} \cup \{a^m b^n c^p : p>n\} \cup$
$\{a^m b^n c^p : m>p\} \cup \{a^m b^n c^p : p>m\}$

$= aa^*\{a^n b^n : n \geq 0\}c^* \cup \{a^n b^n : n \geq 0\}bb^*c^* \cup$
$a^*bb^*\{b^n c^n : n \geq 0\} \cup a^*\{b^n c^n : n \geq 0\}cc^* \cup$
$aa^*\{a^n b^m c^n : n,m \geq 0\} \cup \{a^n b^m c^n : n,m \geq 0\}cc^*.$

(Die letzten zwei Sprachen lassen sich einfach von kontextfreien Grammatiken erzeugen.) □

Theorem

Die deterministisch kontextfreien Sprachen sind unter Vereinigung nicht abgeschlossen.

Beweis

Im vorigen Beweis haben wir die nichtdeterministische Sprache $\Sigma^* - L_{abc}$ als eine endliche Vereinigung von deterministischen Sprachen dargestellt: $\Sigma^* - a^*b^*c^*$ ist regulär, und damit deterministisch kontextfrei; für die anderen Sprachen lassen sich leicht deterministische Automaten konstruieren. □

Theorem

Die deterministisch kontextfreien Sprachen sind unter Schnittbildung nicht abgeschlossen.

Beweis

$L_1 \cup L_2 = \Sigma^* - ((\Sigma^* - L_1) \cap (\Sigma^* - L_2))$,

und die deterministisch kontextfreien Sprachen sind abgeschlossen unter Komplementbildung, d.h.: Wären sie unter Schnittbildung abgeschlossen, wären sie auch unter Vereinigung abgeschlossen. □

Theorem

Die deterministisch kontextfreien Sprachen sind unter Schnittbildung und Vereinigung mit regulären Sprachen abgeschlossen.

Beweis

Der Beweis, dass die kontextfreien Sprachen unter Schnittbildung mit regulären Sprachen abgeschlossen sind, läßt sich auch ohne sonstige Änderung mit einem DKA statt einem KA durchführen.

Weil $L \cup R = \Sigma^* - ((\Sigma^* - L) \cap (\Sigma^* - R))$, sind die deterministisch kontextfreien Sprachen auch unter Vereinigung mit regulären Mengen abgeschlossen. □

Theorem

Die deterministisch kontextfreien Sprachen sind unter Konkatenation und Kleene'scher Hüllenbildung nicht abgeschlossen.

Beweis

$L_1 = \{a^m b^n c^p : m=n\}$ und $L_2 = \{a^m b^n c^p : n=p\}$ sind beide deterministisch kontextfrei, aber $L = L_1 \cup L_2$ ist es nicht, weil das Komplement von L nicht kontextfrei ist. Dies läßt sich folgendermaßen zeigen: Nehmen wir an, dass $\Sigma^* - L$ kontextfrei ist. Dann muss aber $(\Sigma^* - L) \cap a^*b^*c^*$ auch kontextfrei sein. Die letzte Sprache ist aber $\{a^m b^n c^p : m \neq n \text{ und } n \neq p\}$. Mit einer starken Version des Pumping-Lemmas (das Ogden-Lemma) läßt sich zeigen, dass diese Sprache nicht kontextfrei ist.

Jetzt betrachten wir $L_3 = dL_1 \cup L_2$. Diese Sprache ist deterministisch: Ein Automat kann nach dem Lesen vom ersten Zeichen schon entscheiden, ob die Eingabe zu L_1 oder zu L_2 gehört.

Jetzt bilden wir die Konkatenation von den beiden deterministisch kontextfreien Sprachen d^* und L_3. Die dadurch entstandene Sprache ist nicht deterministisch kontextfrei, weil sonst auch

$L_4 = d^*L_3 \cap da^*b^*c^* = dL_1 \cup dL_2 = dL$

deterministisch kontextfrei wäre, was offensichtlich nicht der Fall sein kann.

Für Kleene'sche Hüllenbildung betrachten wir die Sprache $L_5 = d \cup L_3$. L_5 ist eine deterministisch kontextfreie Sprache, weil L_3 es ist.

Aber L_5^* kann nicht deterministisch sein, weil

$L_5^* \cap da^*b^*c^* = dL_1 \cup dL_2 = dL$. □

Prädiktiver Parser mit Lookahead

Wir betrachten wieder die Grammatik

$G = \langle \{a, b, S\}, \{a, b\}, \{S \to aSb, S \to \varepsilon\}, S\rangle$,

die $L = \{a^n b^n : n \geq 0\}$ erzeugt.

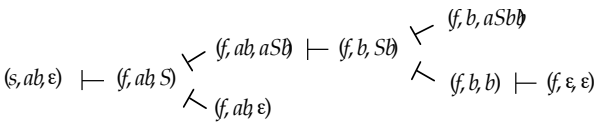
Hier der Top-down-Kellerautomat zu G wiederholt:

$M = \langle \{s, f\}, \{a, b\}, \{S, a, b\}, \Delta, s, \{f\}\rangle$, wobei

$\Delta = \{ \langle\langle s, \varepsilon, \varepsilon\rangle, \langle f, S\rangle\rangle,$
$\langle\langle f, \varepsilon, S\rangle, \langle f, aSb\rangle\rangle,$
$\langle\langle f, \varepsilon, S\rangle, \langle f, \varepsilon\rangle\rangle,$
$\langle\langle f, a, a\rangle, \langle f, \varepsilon\rangle\rangle,$
$\langle\langle f, b, b\rangle, \langle f, \varepsilon\rangle\rangle \}$

M ist nicht deterministisch, L ist aber eine deterministisch kontextfreie Sprache.

Wir betrachten wieder die möglichen Berechnungen bei Eingabe ab:



Wir sehen, dass der Automat immer die richtige Regel hätte wählen können, wenn er Information über das nächste Zeichen der Eingabe hätte.

Lookahead

Wir konstruieren aus M einen DKA M' mit Lookahead: M' liest immer das nächste Zeichen, um zwischen zwei anwendbaren Regeln unterscheiden zu können. Das gelesene Zeichen wird als Zustand gespeichert:

$M' = \langle \{s, q, q_a, q_b, q_\$\}, \{a, b, \$\}, \{S, a, b\}, \Delta', s, \{q_\$\}\rangle$,

wobei

$\Delta = \{\langle\langle s, \varepsilon, \varepsilon\rangle, \langle q, S\rangle\rangle,$ *[Start]*
$\langle\langle q, a, \varepsilon\rangle, \langle q_a, \varepsilon\rangle\rangle,$ *[Lese: a]*
$\langle\langle q_a, \varepsilon, a\rangle, \langle q, \varepsilon\rangle\rangle,$ *[a generiert, weiterlesen]*
$\langle\langle q, b, \varepsilon\rangle, \langle q_b, \varepsilon\rangle\rangle,$ *[Lese: b]*
$\langle\langle q_b, \varepsilon, b\rangle, \langle q, \varepsilon\rangle\rangle,$ *[b generiert, weiterlesen]*
$\langle\langle q, \$, \varepsilon\rangle, \langle q_\$, \varepsilon\rangle\rangle,$ *[Lese: $ – halt]*
$\langle\langle q_a, \varepsilon, S\rangle, \langle q_a, aSb\rangle\rangle,$ *[Nur S → aSb möglich]*
$\langle\langle q_b, \varepsilon, S\rangle, \langle q_b, \varepsilon\rangle\rangle \}$ *[Nur S → ε möglich]*

In den Zuständen q_a und q_b wird überprüft, ob das entsprechende Zeichen schon mit einer Regel erzeugt wurde. Wenn dies zutrifft, wird das Zeichen

vom Keller weggenommen und ein neues Zeichen wird gelesen. Wenn ein Terminalzeichen noch nicht erzeugt worden ist, werden die anwendbaren Regeln ausprobiert (in diesem Fall nur eine Regel je Zustand).

Wir schauen uns jetzt an, wie der Automat die Eingabe *aabb$* akzeptiert.

Zustand	Unverbrauchte Eingabe	Keller	Regel, die angewandt wurde bzw. Kommentar
s	*aabb$*	ε	*Start*
q	*aabb$*	*S*	
q_a	*abb$*	*S*	*a gelesen (Lookahead)*
q_a	*abb$*	*aSb*	$S \rightarrow aSb$ (1)
q	*abb$*	*Sb*	*a korrekt*
q_a	*bb$*	*Sb*	*a gelesen (Lookahead)*
q_a	*bb$*	*aSbb*	$S \rightarrow aSb$ (2)
q	*bb$*	*Sbb*	*a korrekt*
q_b	*b$*	*Sbb*	*b gelesen (Lookahead)*
q_b	*b$*	*bb*	$S \rightarrow \varepsilon$ (3)
q	*b$*	*b*	*b korrekt*
q_b	*$*	*b*	*b gelesen (Lookahead)*
q	*$*	ε	*b korrekt*
$q_\$$	ε	ε	*$ gelesen, akzeptiert!*

Bei Eingabe *aab$* hält der Automat mit *b* auf dem Keller. Bei Eingabe *aabbb$* bleibt der Automat in q_b hängen.

Linksfaktorisieren

Wir betrachten jetzt wieder die LISP-Grammatik

$G = \langle \{S, L, (,), a, \_\}, \{(,), a, \_\}, R, S \rangle$

wobei

$R = \{ S \rightarrow (), S \rightarrow a\ S \rightarrow (L)\ L \rightarrow S, L \rightarrow L\_S \}$

Es gibt zwei *S*-Regeln, die mit demselben Symbol anfangen, $S \rightarrow ()$ und $S \rightarrow (L)$.

Dies heißt, dass ein Automat mit einem Lookahead von nur einem Zeichen nicht zwischen diesen beiden Regeln entscheiden kann. Wir können aber die beiden Regeln mit den folgenden drei ersetzen:

$S \rightarrow (S'$

$S' \rightarrow L)$

$S' \to)$

Wir wenden folgende heuristische Methode an:

Linksfaktorisieren:

Wenn $A \to \alpha\beta$ und $A \to \alpha\gamma$ Regeln sind, wo $\alpha \neq \varepsilon$, dann ersetze sie mit $A \to \alpha A'$, $A' \to \beta$, und $A' \to \gamma$.

Theorem

Linksfaktorisieren ändert nicht die generierte Sprache.

Beweis: Einleuchtend! □

LL(k)-Grammatiken

Eine Grammatik heißt *LL(k)* wenn ein Lookahead von k Symbolen genügt, um, für jedes A, immer nur eine von den ganzen A-Regeln wählen zu können. Durch entfernen von Linksrekursion und Links-Faktorisieren lassen sich viele Grammatiken in *LL(1)*-Grammatiken umwandeln. Aus der LISP-Grammatik können wir folgende *LL(1)*-Grammatik ableiten:

$S \to (S'$
$S' \to L)$
$L \to SL'$
$S \to a$
$S' \to)$
$L' \to \_SL'$
$L' \to \varepsilon$

Der entsprechende Automat ist deterministisch und hat die folgenden 22 Übergänge:

$\langle\langle s, \varepsilon, \varepsilon\rangle, \langle q, S\rangle\rangle$	
$\langle\langle q, \sigma, \varepsilon\rangle, \langle q_\sigma, \varepsilon\rangle\rangle$	für $\sigma =$ *(,), _, a, $ (lookahead)*
$\langle\langle q_\sigma, \varepsilon, \sigma\rangle, \langle q, \varepsilon\rangle\rangle$	für $\sigma =$ *(,), _, a*
$\langle\langle q_($, ε, $S\rangle$, $\langle q_($, $(S')\rangle\rangle$	$S \to (S'$
$\langle\langle q_a, \varepsilon, S\rangle, \langle q_a, a\rangle\rangle$	$S \to a$
$\langle\langle q_\sigma, \varepsilon, S'\rangle, \langle q_\sigma, L\rangle\rangle$	für $\sigma =$ *(, a;* $S' \to L$
$\langle\langle q_), \varepsilon, S'\rangle, \langle q_),)\rangle\rangle$	$S' \to)$
$\langle\langle q_\sigma, \varepsilon, L\rangle, \langle q_\sigma, SL'\rangle\rangle$	für $\sigma =$ *(, a;* $L \to SL'$
$\langle\langle q\_, \varepsilon, L'\rangle, \langle q\_, \_SL'\rangle\rangle$	$L' \to \_SL'$
$\langle\langle q_\sigma, \varepsilon, L'\rangle, \langle q_\sigma, \varepsilon\rangle\rangle$	für $\sigma =$ *(,), a, $;* $L' \to \varepsilon$

Z.B. gibt es zu der Regel $L \to SL'$ eine Transition aus q_a und eine aus $q_($, weil *a* und *(* die möglichen Anfangssymbole von S sind.

Parsen des S-Expressions *(()_a):*

Zustand	Rest-eingabe	Keller	Regel, die angewandt wurde/Kommentar
s	**(()**_a**)**$	ε	*Start*
q	**(()**_a**)**$	S	
$q_($	**()**_a**)**$	S	**(** *gelesen (Lookahead)*
$q_($	**()**_a**)**$	**(**S'	$S \rightarrow$ **(**S'
q	**()**_a**)**$	S'	**(** *korrekt*
$q_($	**)**_a**)**$	S'	**(** *gelesen (Lookahead)*
$q_($	**)**_a**)**$	L**)**	$S' \rightarrow L$**)**
$q_($	**)**_a**)**$	SL'**)**	$L \rightarrow SL'$
$q_($	**)**_a**)**$	**(**$S'L'$**)**	$S \rightarrow$ **(**S'
q	**)**_a**)**$	$S'L'$**)**	**(** *korrekt*
$q_)$	_a**)**$	$S'L'$**)**	**)** *gelesen (Lookahead)*
$q_)$	_a**)**$	**)**L'**)**	$S' \rightarrow$ **)**
q	_a**)**$	L'**)**	**)** *korrekt*
$q_\_$	a**)**$	L'**)**	_ *gelesen (Lookahead)*
$q_\_$	a**)**$	_SL'**)**	$L' \rightarrow$ _SL'
q	a**)**$	SL'**)**	_ *korrekt*
q_a	**)**$	SL'**)**	a *gelesen (Lookahead)*
q_a	**)**$	aL'**)**	$S \rightarrow a$
q	**)**$	L'**)**	a *korrekt*
$q_)$	$	L'**)**	**)** *gelesen (Lookahead)*
$q_)$	$	**)**	$L' \rightarrow$ ε
q	$	ε	**)** *korrekt*
$q_\$$	ε	ε	$ *gelesen, akzeptiert!*

Die Hierarchie der LL-Sprachen

Es fragt sich jetzt: Gibt es zu jeder deterministisch kontextfreien Sprache eine *LL(k)*-Grammatik (für irgendein *k*)? Die Antwort ist *nein*, es kann sogar gezeigt werden, dass die Klasse der Sprachen, die von *LL(k+1)*-Grammatiken generiert werden, eine echte Obermenge der Sprachen ist, die von *LL(k)*-Grammatiken generiert werden.