THEORETISCHE INFORMATIK: EINE ANWENDUNGSORIENTIERTE EINFÜHRUNG
1. AUFLAGE, KAPITEL 6.4

9. Oktober 2001

Inhaltsverzeichnis

6 Automatentheorie und Formale Sprachen

6.4 Kontextfreie Analyse für kontextfreie Sprachen von speziellem Typ

Wir haben ein Verfahren für die kontextfreie Analyse kennengelernt, das für jede kontextfreie Sprache anwendbar ist. Dieses Verfahren hat allerdings den Nachteil, daß es bei einer Eingabe der Länge n den Platz $O(n^2)$ und $O(n^3)$ Zeit benötigt. Die benötigte Zeit reduziert sich auf $O(n^2)$, falls die zugrundeliegende Grammatik eindeutig ist.

Nun stellt sich die Frage, ob wir für die Syntaxanalyse wirklich einen derart allgemeinen Algorithmus benötigen oder ob die kontextfreien Grammatiken, die die Syntax einer Programmiersprache beschreiben, eine spezielle Struktur besitzen, die wesentlich effizientere Algorithmen für die kontextfreie Analyse erlauben. Zur Beantwortung dieser Frage ist es zunächst nützlich, die Ursachen für den hohen Zeitbedarf der Simulation des Kellerautomaten herauszuarbeiten. Die erhöhte Zeitkomplexität rührt daher, daß alle möglichen Kellerinhalte gleichzeitig simuliert werden. Dies tun wir, da wir nicht lokal entscheiden können, welcher Kellerinhalt der richtige ist, oder, anders ausgedrückt, welche Alternative bei einer Expansion gewählt werden sollte. Ein Algorithmus, der lokal, d.h., in konstanter Zeit und mit konstantem Speicherplatz, die richtige Alternative bestimmen kann, würde die Durchführung der kontextfreien Analyse in linearer Zeit mit linearem Platz ermöglichen. Wir werden Grammatiktypen kennenlernen, für die derartige Algorithmen existieren und die auch zur Definition der Syntax von Programmiersprachen ausreichen.

6.4.1 $LL(k)$-Grammatiken und $LL(k)$-Parser

Sei $G = (V, \Sigma, P, S)$ eine eindeutige, reduzierte kontextfreie Grammatik und $M_G = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ der durch G induzierte Kellerautomat. Betrachte eine Rechnung von M_G bei Eingabe $w = a_1 a_2 \dots a_n$. Sei $[X \to \beta.A\gamma]$ der aktuelle Zustand von M_G.

Die Variable A habe l Alternativen $\alpha_1, \alpha_2, \ldots, \alpha_l$. Das heißt,

$$\delta([X \to \beta.A\gamma], \varepsilon, Z) = \\ \{([A \to .\alpha_1], [X \to \beta.A\gamma]Z), \ldots, ([A \to .\alpha_l], [X \to \beta.A\gamma]Z)\}.$$

Da G eindeutig und reduziert ist, kann nur eine der l Alternativen die Ableitung eines gegebenen Wortes in $L(G)$ zulassen. Die Frage ist nun, wie diese Alternative auf effiziente Art und Weise ausgewählt werden kann.

Sei $w \in L(G)$ und $S \underset{\text{lm}}{\Longrightarrow} \gamma_0 \underset{\text{lm}}{\Longrightarrow} \gamma_1 \underset{\text{lm}}{\Longrightarrow} \ldots \underset{\text{lm}}{\Longrightarrow} \gamma_m = w$ die Linksableitung von w. Sei $\gamma_i = a_1 a_2 \ldots a_j A\beta$ und der Parser habe die ersten j Symbole der Eingabe gelesen. Unser Ziel ist nun die Konstruktion von kontextfreien Grammatiken, für die die richtige Alternative aufgrund der Kenntnis von

- $a_1 a_2 \ldots a_j$ bereits gelesene Eingabe
- A Variable, die expandiert werden soll
- $a_{j+1} a_{j+2} \ldots a_{j+k}$ nächsten k Eingabezeichen, k fest

ausgewählt werden kann.

Am Ableitungsbaum läßt sich diese Eigenschaft, wie in Abbildung 6.1 skizziert, verdeutlichen. Derartige Grammatiken heißen $LL(k)$-Grammatiken. Dabei symbolisiert das erste L, daß die Eingabe von links nach rechts gelesen wird, während das zweite L ausdrückt, daß eine Linksableitung zugrunde liegt.

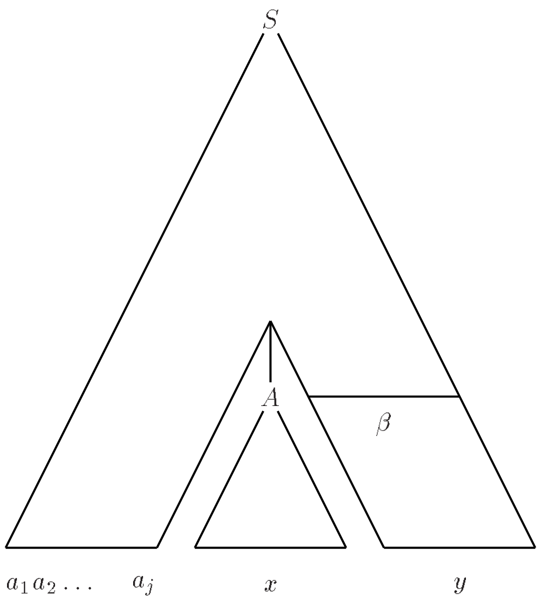

Abbildung 6.1: Ableitungsbaum für die Eingabe $w = a_1 a_2 \ldots a_j xy$.

Als nächstes werden wir $LL(k)$-Grammatiken formal definieren. Für $\alpha \in V^*$ enthält $\mathrm{FIRST}_k(\alpha)$ alle Präfixe der Länge k von Terminalstrings sowie alle Terminalstrings der Länge $\leq k$, die aus α ableitbar sind. Das heißt,

$$\mathrm{FIRST}_k(\alpha) = \{x \in \Sigma^* \mid \alpha \underset{\mathrm{lm}}{\overset{*}{\Longrightarrow}} xy, y \in \Sigma^*, |x| = k \text{ oder } y = \varepsilon, |x| < k\}.$$

Sei $G = (V, \Sigma, P, S)$ eine reduzierte kontextfreie Grammatik und $k \in \mathbb{N}_0$. G heißt *$LL(k)$-Grammatik*, falls G folgende Eigenschaft erfüllt: Für alle Linksableitungen

$$S \underset{\mathrm{lm}}{\overset{*}{\Longrightarrow}} wA\alpha \underset{\mathrm{lm}}{\Longrightarrow} w\beta\alpha \underset{\mathrm{lm}}{\overset{*}{\Longrightarrow}} wx \text{ und}$$
$$S \underset{\mathrm{lm}}{\overset{*}{\Longrightarrow}} wA\alpha \underset{\mathrm{lm}}{\Longrightarrow} w\gamma\alpha \underset{\mathrm{lm}}{\overset{*}{\Longrightarrow}} wy$$

mit $\mathrm{FIRST}_k(x) = \mathrm{FIRST}_k(y)$ gilt $\beta = \gamma$.

Eine kontextfreie Grammatik G ist LL, falls ein $k \in \mathbb{N}_0$ existiert, so daß G eine $LL(k)$-Grammatik ist.

Ausgehend von einer $LL(1)$-Grammatik erweitert folgendes Beispiel diese sukzessive, so daß sie zunächst nicht mehr eine $LL(1)$-, jedoch noch eine $LL(2)$-Grammatik ist, und schließlich sogar die LL-Eigenschaft verliert:

Beispiel 6.1 Sei G_1 eine kontextfreie Grammatik, die durch folgende Produktionen definiert ist:

$$\begin{aligned}\mathrm{STAT} \rightarrow\ & \textbf{if id then } \mathrm{STAT} \textbf{ else } \mathrm{STAT} \textbf{ fi} \mid \\ & \textbf{while id do } \mathrm{STAT} \textbf{ od} \mid \\ & \textbf{begin } \mathrm{STAT} \textbf{ end} \mid \\ & \textbf{id} := \textbf{id}.\end{aligned}$$

G_1 ist eine $LL(1)$-Grammatik. Tritt STAT als am weitesten links stehendes Nichtterminalzeichen auf, dann bestimmt das nächste Eingabesymbol, unabhängig von der bisher gelesenen Eingabe, die anzuwendende Alternative, da die Alternativen von STAT mit paarweise verschiedenen Terminalzeichen beginnen.

Durch Hinzufügen der folgenden Produktionen erweitern wir G_1 zu einer kontextfreien Grammatik G_2:

$$\mathrm{STAT} \rightarrow \textbf{id} : \mathrm{STAT} \mid \textbf{id}(\textbf{id}).$$

G_2 ist nicht mehr LL(1). Um dies einzusehen, betrachte folgenden Ableitungen:

$$\mathrm{STAT} \underset{\mathrm{lm}}{\overset{*}{\Longrightarrow}} w\,\mathrm{STAT}\,\alpha \underset{\mathrm{lm}}{\Longrightarrow} \begin{cases} w\,\underbrace{\mathbf{id} := \mathbf{id}}_{\beta} & \overset{*}{\Longrightarrow} wx \\ w\,\underbrace{\mathbf{id} : \mathrm{STAT}}_{\gamma} & \overset{*}{\Longrightarrow} wy \\ w\,\underbrace{\mathbf{id}(\mathbf{id})}_{\delta} & \overset{*}{\Longrightarrow} wz. \end{cases}$$

Es gilt $\mathrm{FIRST}_1(x) = \mathrm{FIRST}_1(y) = \mathrm{FIRST}_1(z) = \mathbf{id}$, aber β, γ und δ sind paarweise verschieden. G_2 ist jedoch eine $LL(2)$-Grammatik, was leicht nachprüfbar ist.

Wir erweitern nun G_1 durch Hinzufügen der folgenden Produktionen zu einer kontextfreien Grammatik G_3:

$$\begin{aligned} \mathrm{STAT} &\to \mathbf{id}(\mathrm{IDLIST}) \mid \mathrm{VAR} := \mathrm{VAR} \\ \mathrm{IDLIST} &\to \mathbf{id} \mid \mathbf{id}, \mathrm{IDLIST} \\ \mathrm{VAR} &\to \mathbf{id} \mid \mathbf{id}(\mathrm{IDLIST}). \end{aligned}$$

G_3 ist nicht in LL.

Übung 6.1: *Zeigen sie, daß G_3 keine LL-Grammatik ist.*

♦

Die $LL(k)$-Definition besagt, daß die nächste Alternative, die bei einer gegebenen Linkssatzform $wA\alpha$ gewählt werden muß, eindeutig durch w, A und die nächsten k Eingabesymbole bestimmt ist. Müßte man jedoch immer ganz w betrachten, dann würde dies nur zu quadratischen Analysealgorithmen führen. Mit Hilfe des folgenden Satzes können wir für $LL(1)$-Grammatiken zeigen, daß dies nicht notwendig ist.

Satz 6.1 *Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ ist genau dann $LL(k)$, wenn für alle $wA\alpha \in V^*$ mit $S \underset{lm}{\overset{*}{\Longrightarrow}} wA\alpha$ und alle paarweise verschiedenen Alternativen β und γ für A gilt $\mathrm{FIRST}_k(\beta\alpha) \cap \mathrm{FIRST}_k(\gamma\alpha) = \emptyset$.*

Beweis:
„$\Longrightarrow$“

Nehmen wir an, daß $\mathrm{FIRST}_k(\beta\alpha) \cap \mathrm{FIRST}_k(\gamma\alpha) \neq \emptyset$. Betrachte $x \in \mathrm{FIRST}_k(\beta\alpha) \cap \mathrm{FIRST}_k(\gamma\alpha)$. Aus der Definition von FIRST_k und der Reduzierbarkeit von

G folgt, daß die Ableitungen

$$S \underset{\text{lm}}{\overset{*}{\Longrightarrow}} wA\alpha \underset{\text{lm}}{\Longrightarrow} \begin{cases} w\beta\alpha & \underset{\text{lm}}{\overset{*}{\Longrightarrow}} \; wxy \\ w\gamma\alpha & \underset{\text{lm}}{\overset{*}{\Longrightarrow}} \; wxz \end{cases}$$

existieren. Falls $|x| < k$, dann gilt $y = z = \varepsilon$. Wegen $\beta \neq \gamma$ wäre dann G keine $LL(k)$-Grammatik.

„$\Longleftarrow$“
Nehmen wir an, daß G nicht $LL(k)$ ist. Dann existieren zwei Ableitungen

$$S \underset{\text{lm}}{\overset{*}{\Longrightarrow}} wA\alpha \underset{\text{lm}}{\Longrightarrow} \begin{cases} w\beta\alpha & \underset{\text{lm}}{\overset{*}{\Longrightarrow}} \; wx \\ w\gamma\alpha & \underset{\text{lm}}{\overset{*}{\Longrightarrow}} \; wy \end{cases}$$

mit $\mathrm{FIRST}_k(x) = \mathrm{FIRST}_k(y)$ und $\beta \neq \gamma$.

Dann sind β und γ verschiedene Alternativen von A und $\mathrm{FIRST}_k(x) = \mathrm{FIRST}_k(y) \in \mathrm{FIRST}_k(\beta\alpha) \cap \mathrm{FIRST}_k(\gamma\alpha)$.

■

Im Gegensatz zur Definition der $LL(k)$-Grammatik impliziert obiger Satz ein gutes Kriterium, um zu überprüfen, ob eine gegebene kontextfreie Grammatik $G = (V, \Sigma, P, S)$, die keine ε-Regeln enthält, $LL(1)$ ist. G ist genau dann $LL(1)$, wenn für alle $A \in N$

$$A \to \beta|\gamma \in P \implies \mathrm{FIRST}_1(\beta) \cap \mathrm{FIRST}_1(\gamma) = \emptyset.$$

Also genügt es, endlich viele FIRST-Mengen zu berechnen und endlich viele Durchschnitte zu bilden.

Falls die gegebene kontextfreie Grammatik $G = (V, \Sigma, P, S)$ ε-Regeln enthält, dann ist es etwas komplizierter, ein gutes Kriterium für die $LL(1)$-Eigenschaft herzuleiten. Zur Entwicklung eines solchen betrachten wir noch einmal die Aussage des obigen Satzes.

Falls sowohl $\beta \overset{*}{\Longrightarrow} \varepsilon$ als auch $\gamma \overset{*}{\Longrightarrow} \varepsilon$, dann ist G offensichtlich nicht LL. Sei o.B.d.A. $\beta \overset{*}{\Longrightarrow} \varepsilon$. Falls G eine $LL(1)$-Grammatik ist, dann folgt direkt aus Satz 6.1, daß $\mathrm{FIRST}_1(\alpha) \cap \mathrm{FIRST}_1(\gamma) = \emptyset$ für alle Linkssatzformen $wA\alpha$. Dies ist äquivalent zu

$$\underbrace{\bigcup \{\mathrm{FIRST}_1(\alpha) \mid S \underset{\text{lm}}{\overset{*}{\Longrightarrow}} wA\alpha\}}_{\mathrm{FOLLOW}_1(A)} \cap \mathrm{FIRST}_1(\gamma) = \emptyset.$$

Für eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ und für $\beta \in V^*$ definieren wir

$$\mathrm{FOLLOW}_k(\beta) = \{u \in \Sigma^* \mid S \underset{\text{lm}}{\overset{*}{\Longrightarrow}} \alpha\beta\delta \text{ und } u \in \mathrm{FIRST}_k(\delta)\}.$$

Für $L_1, L_2 \subseteq V^*$ und $k \in \mathbb{N}_0$ definieren wir

$$L_1 \cdot L_2 = \{\alpha\beta \mid \alpha \in L_1, \beta \in L_2\} \text{ und}$$
$$L_1 \oplus_k L_2 = \{u \in \Sigma^* \mid u \in \mathrm{FIRST}_k(\alpha\beta) \text{ für } \alpha \in L_1, \beta \in L_2\}.$$

Zusammen mit obiger Beobachtung impliziert Satz 6.1 direkt folgenden Satz:

Satz 6.2 *Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ ist genau dann $LL(1)$, wenn für alle $A \in N$ für alle verschiedenen Alternativen β und γ für A gilt*

$$\mathrm{FIRST}_1(\beta) \oplus_1 \mathrm{FOLLOW}_1(A) \cap \mathrm{FIRST}_1(\gamma) \oplus_1 \mathrm{FOLLOW}_1(A) = \emptyset.$$

Als Konsequenz erhalten wir folgendes Korollar:

Korollar 6.1 *Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ ist genau dann LL(1), wenn für alle $A \in N$ für alle Alternativen $\alpha_1, \alpha_2, \ldots, \alpha_r$ für A gilt*

1. $\mathrm{FIRST}_1(\alpha_1), \ldots, \mathrm{FIRST}_1(\alpha_r)$ *sind paarweise disjunkt.*
2. *Falls $\varepsilon \in \mathrm{FIRST}_1(\alpha_i)$ für ein i, dann gilt*

$$\mathrm{FOLLOW}_1(A) \cap \mathrm{FIRST}_1(\alpha_j) = \emptyset$$

für alle $j \in \{1, \ldots, r\} \setminus \{i\}$.

Satz 6.2 kann nicht für beliebiges $k > 1$ verallgemeinert werden. Es ist leicht zu sehen, daß, falls $\mathrm{FIRST}_k(\beta) \oplus_k \mathrm{FOLLOW}_k(A) \cap \mathrm{FIRST}_k(\gamma) \oplus_k \mathrm{FOLLOW}_k(A) = \emptyset$ für alle $A \in N$ für alle Alternativen β, γ für A, die Grammatik G eine $LL(k)$-Grammatik ist.

Übung 6.2: *Sei $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik. Ferner sei für alle $A \in N$ für alle Alternativen β und γ für A*

$$\mathrm{FIRST}_k(\beta) \oplus_k \mathrm{FOLLOW}_k(A) \cap \mathrm{FIRST}_k(\gamma) \oplus_k \mathrm{FOLLOW}_k(A) = \emptyset.$$

Zeigen Sie, daß G eine $LL(k)$-Grammatik ist.

Wie folgendes Beispiel zeigt, gilt jedoch die andere Richtung von Satz 6.2 schon für $k = 2$ nicht:

Beispiel 6.2 Sei die Grammatik G durch Produktionen $S \to aAaa \mid bAba$ und $A \to b \mid \varepsilon$ definiert.

Zunächst werden wir mit Hilfe des Satzes 6.1 zeigen, daß G eine $LL(2)$-Grammatik ist. Wir unterscheiden zwei Fälle:

1. Eine Linksableitung beginnt mit der Produktion $S \to aAaa$.
   Dann gilt: $\mathrm{FIRST}_2(baa) \cap \mathrm{FIRST}_2(aa) = \emptyset$.
2. Eine Linksableitung beginnt mit der Produktion $S \to bAba$.
   Dann gilt $\mathrm{FIRST}_2(bba) \cap \mathrm{FIRST}_2(ba) = \emptyset$.

Also ist nach Satz 6.1 G eine $LL(2)$-Grammatik. Jedoch gilt $\mathrm{FOLLOW}_2(A) = \{aa, ba\}$ und somit

$$\mathrm{FIRST}_2(b) \oplus_2 \mathrm{FOLLOW}_2(A) \cap \mathrm{FIRST}_2(\varepsilon) \oplus_2 \mathrm{FOLLOW}_2(A) = \{ba\}.$$

♦

Obiges Beispiel impliziert, daß bereits für $k = 2$ die gelesene Eingabe für die Bestimmung der richtigen Alternative angeschaut werden muß. Interessant sind insbesondere $LL(k)$-Grammatiken, für die dies nicht notwendig ist.

Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ heißt *starke* $LL(k)$*-Grammatik* ($SLL(k)$*-Grammatik*), falls $\mathrm{FIRST}_k(\beta) \oplus_k \mathrm{FOLLOW}_k(A) \cap \mathrm{FIRST}_k(\gamma) \oplus_k \mathrm{FOLLOW}_k(A) = \emptyset$ für alle $A \in N$, für alle verschiedenen Alternativen β und γ für A.

Wir werden zunächst für $SLL(k)$-Grammatiken ein Verfahren zur Konstruktion eines Parsers entwickeln. Gemäß der Definition der $SLL(k)$-Grammatik sind hierfür die benötigten FIRST_k- und FOLLOW_k-Mengen von zentraler Bedeutung. Demzufolge ist deren effiziente Berechnung unser nächstes Anliegen.

Wir werden uns zunächst ein einfaches Verfahren zur Berechnung von $\mathrm{FIRST}_k(\alpha)$, $\alpha = A_1 A_2 \ldots A_r \in V^*$ überlegen. Das Verfahren ist ein Fixpunktverfahren. D.h., in einem Schritt werden der aktuellen Menge weitere Elemente hinzugefügt. Das Verfahren terminiert, sobald in einem Schritt die Elementanzahl der Menge sich nicht vergrößert. Wir gehen dabei davon aus, daß die zugrundeliegende kontextfreie Grammatik $G = (V, \Sigma, P, S)$ reduziert ist. Das Verfahren nutzt folgende Eigenschaft aus:

$$\mathrm{FIRST}_k(\alpha) = \mathrm{FIRST}_k(A_1) \oplus_k \mathrm{FIRST}_k(A_2) \oplus_k \ldots \oplus_k \mathrm{FIRST}_k(A_r).$$

Für $a \in \Sigma \cup \{\varepsilon\}$ gilt offensichtlich $\mathrm{FIRST}_k(a) = \{a\}$. Es genügt also, sich ein Verfahren zur Berechnung von $\mathrm{FIRST}_k(A)$, $A \in N$ zu überlegen. Hierzu definieren wir für alle $A \in V$ für wachsende $i \geq 0$ folgende Mengen $F_i(A)$:

1. $F_i(a) = \{a\}$ für alle $a \in \Sigma$ und $i \geq 0$.

2. $F_0(A) = \{x \in \Sigma^* \mid A \to x\beta \in P, |x| = k \text{ oder } |x| < k, \beta = \varepsilon\}$.
3. Nehmen wir an, daß $F_0, F_1, \ldots, F_{i-1}$ bereits für alle $A \in N$ definiert sind. Wir definieren dann

$$F_i(A) = F_{i-1}(A) \cup \{x \in \Sigma^* \mid A \to B_1B_2 \ldots B_s \in P \text{ und } x \in F_{i-1}(B_1) \oplus_k F_{i-1}(B_2) \oplus_k \ldots \oplus_k F_{i-1}(B_s)\}.$$

Obige Definition gibt uns eine Möglichkeit, $F_i(A)$ zu berechnen. Sobald $F_i(B) = F_{i-1}(B)$ für alle $B \in N$, gilt auch $F_{i+l}(A) = F_i(A)$ für alle $l \geq 0$. Aus der Definition folgt dann $\mathrm{FIRST}_k(A) = F_i(A)$ für dieses i.

Obigem Verfahren ist nicht anzusehen, wie es effizient implementiert werden kann. Unser Ziel ist es nun, unter Anwendung der uns bekannten Datenstrukturen und Programmierverfahren, obige Ideen derart zu verfeinern, daß dabei eine effiziente Implementierung des resultierenden Verfahrens herauskommt.

Für $\alpha \in V^*$ und $k \in \mathbb{N}_0$ bezeichne $\mathrm{EXACT}_k(\alpha)$ die Menge aller Strings $u \in \Sigma^k$, die aus α ableitbar sind. Das heißt,

$$\mathrm{EXACT}_k(\alpha) = \{u \in \Sigma^k \mid \alpha \overset{*}{\Longrightarrow} u\}.$$

$\mathrm{FIRST}_k(\alpha)$ enthält Strings der Länge $\leq k$, die aus α ableitbar sind. Entfernt man diese aus $\mathrm{FIRST}_k(\alpha)$, dann erhält man $\mathrm{first}_k(\alpha)$. Formal definieren wir

$$\mathrm{EXACT}_{\leq k}(\alpha) = \bigcup_{l=0}^{k} \mathrm{EXACT}_l(\alpha)$$

und

$$\mathrm{first}_k(\alpha) = \mathrm{FIRST}_k(\alpha) \setminus \mathrm{EXACT}_{\leq k}(\alpha).$$

Es gilt genau dann $\mathrm{EXACT}_0(\alpha) \neq \emptyset$, wenn $\alpha \overset{*}{\Longrightarrow} \varepsilon$. Unter der Verwendung der EXACT-Mengen möchten wir auf effiziente Art und Weise die FIRST- und FOLLOW-Mengen berechnen. Sei $\alpha = A_1A_2 \ldots A_r$. Dann können wir $\mathrm{EXACT}_k(\alpha)$ wie folgt schreiben:

$$\begin{aligned}
\mathrm{EXACT}_k(\alpha) &= \bigcup_{l=0}^{k} \mathrm{EXACT}_l(A_1) \cdot \mathrm{EXACT}_{k-l}(A_2 \ldots A_r) \\
&= \mathrm{EXACT}_k(A_1) \cdot \mathrm{EXACT}_0(A_2 \ldots A_r) \\
&\quad \cup \mathrm{EXACT}_0(A_1) \cdot \mathrm{EXACT}_k(A_2 \ldots A_r) \\
&\quad \cup \bigcup_{l=1}^{k-1} \mathrm{EXACT}_l(A_1) \cdot \mathrm{EXACT}_{k-l}(A_2 \ldots A_r).
\end{aligned}$$

Sei

$$\max_{\varepsilon}(\alpha) = \begin{cases} \max\{j \mid 1 \leq j \leq r \text{ und } A_1 \dots A_j \stackrel{*}{\Longrightarrow} \varepsilon\} & \text{falls } A_1 \stackrel{*}{\Longrightarrow} \varepsilon \\ 0 & \text{sonst.} \end{cases}$$

D.h., $\max_{\varepsilon}(\alpha)$ ist die Länge des maximalen Präfixes von α, aus dem das leere Wort abgeleitet werden kann. Demzufolge können wir $\mathrm{EXACT}_k(\alpha)$ wie folgt umformen:

$$\begin{aligned} \mathrm{EXACT}_k(\alpha) &= \bigcup_{j=0}^{\max_{\varepsilon}(\alpha)} \left[\bigcup_{l=1}^{k} \mathrm{EXACT}_l(A_{j+1}) \cdot \mathrm{EXACT}_{k-l}(A_{j+2} \dots A_r) \right] \\ &= \bigcup_{j=0}^{\max_{\varepsilon}(\alpha)} \Bigg[\mathrm{EXACT}_k(A_{j+1}) \cdot \mathrm{EXACT}_0(A_2 \dots A_r) \\ &\qquad \cup \bigcup_{l=1}^{k-1} \mathrm{EXACT}_l(A_{j+1}) \cdot \mathrm{EXACT}_{k-l}(A_{j+2} \dots A_r) \Bigg]. \end{aligned}$$

Unser Ziel ist nun, $\mathrm{FIRST}_k(\alpha)$ derart darzustellen, daß bei Kenntnis der EXACT- und der first-Mengen die Berechnung von $\mathrm{FIRST}_k(\alpha)$ einfach ist. Durch einfache Zerlegung erhalten wir:

$$\begin{aligned} \mathrm{FIRST}_k(\alpha) &= \mathrm{first}_k(A_1) \cup \bigcup_{l=0}^{k} \mathrm{EXACT}_l(A_1) \cdot \mathrm{FIRST}_{k-l}(A_2 \dots A_r) \\ &= \mathrm{first}_k(A_1) \cup \mathrm{EXACT}_k(A_1) \\ &\quad \cup \mathrm{EXACT}_0(A_1) \cdot \mathrm{FIRST}_k(A_2 \dots A_r) \\ &\quad \cup \bigcup_{l=1}^{k-1} \mathrm{EXACT}_l(A_1) \cdot \mathrm{FIRST}_{k-l}(A_2 \dots A_r) \\ &= \mathrm{EXACT}_0(\alpha) \cup \\ &\quad \bigcup_{j=0}^{\max_{\varepsilon}(\alpha)} \Bigg[\mathrm{first}_k(A_{j+1}) \cup \mathrm{EXACT}_k(A_{j+1}) \\ &\qquad \cup \bigcup_{l=1}^{k-1} \mathrm{EXACT}_l(A_{j+1}) \cdot \mathrm{FIRST}_{k-l}(A_{j+2} \dots A_r) \Bigg]. \end{aligned}$$

Also genügt es, für $0 \leq l \leq k$ für alle Variablen in N und alle Suffixe der rechten Seiten von Produktionen in P alle EXACT_l- und alle FIRST_l-Mengen zu berechnen.

Wir beginnen mit $l = 0$ und berechnen zunächst für wachsendes l die EXACT_l-Mengen.

Bei der Transformation einer beliebigen gegebenen kontextfreien Grammatik $G = (V, \Sigma, P, S)$ in ein äquivalente, reduzierte kontextfreie Grammatik haben wir uns implizit überlegt, wie wir in $O(|G|)$ Zeit für alle $A \in N$ entscheiden können, ob $A \stackrel{*}{\Longrightarrow} \varepsilon$. Also können wir in linearer Zeit

1. $\mathrm{EXACT}_0(A)$ für alle $A \in N$,
2. $\mathrm{EXACT}_0(A_j \ldots A_r)$, $1 \leq j \leq r$, für alle Alternativen $A_1 A_2 \ldots A_r$ der Variablen in N und
3. $\max_\varepsilon(\alpha)$ für alle rechten Seiten α der Produktionen in P

berechnen.

Zur Konstruktion der EXACT_1-Mengen berechnen wir für jede Variable $A \in N$ die Menge $W_\infty(A) = \{C \in V \mid A \stackrel{*}{\Longrightarrow} C\}$. Dann gilt $\mathrm{EXACT}_1(A) = W_\infty(A) \cap \Sigma$.

Sei $\beta = A_i A_{i+1} \ldots A_r$, $1 \leq i \leq r$ ein Suffix von $\alpha = A_1 \ldots A_r$. Ferner sei

$$W(\beta) = \{A_j \mid 1 \leq j \leq r, A_i \ldots A_{j-1} \stackrel{*}{\Longrightarrow} \varepsilon \text{ und } A_{j+1} \ldots A_r \stackrel{*}{\Longrightarrow} \varepsilon\}.$$

Unter der Voraussetzung, daß alle EXACT_0-Mengen bereits berechnet sind, können diese Mengen leicht in $O(|G|)$ Zeit berechnet werden.

Übung 6.3: *Gegeben sei eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$. Entwerfen Sie einen Algorithmus, der, gegeben alle EXACT_0-Mengen, für alle Suffixe β aller Alternativen von Variablen in N die Menge $W(\beta)$ in Gesamtzeit $O(|G|)$ berechnet.*

$W_\infty(A)$, $A \in N$ kann dann einfach durch folgendes Programmstück konstruiert werden:

(1) $W_{-1}(A) := \emptyset$; $W_0(A) := \{A\}$; $i := 0$;
(2) **while** $W_i(A) \neq W_{i-1}(A)$
    do
        $W_{i+1}(A) := W_i(A) \cup \{C \in W(\alpha) \mid \alpha$ ist Alternative einer Variablen in $W_i(A) \setminus W_{i-1}(A)\}$;
        $i := i + 1$
    od.

Man kann sich leicht davon überzeugen, daß obiges Programmstück nach maximal

$|N|$ Schleifendurchläufen mit $W_i(A) = W_\infty(A)$ terminiert. Nun gilt

$$\mathrm{EXACT}_1(\beta) = \bigcup_{B \in W(\beta)} \mathrm{EXACT}_1(B).$$

Ohne größere Anstrengung kann eine $O(|N| \cdot |G|)$-Implementierung des obigen Verfahrens entwickelt werden. Mit einiger Anstrengung kann die benötigte Zeit auf $O(|G|)$ reduziert werden.

Übung 6.4*:

a) *Entwickeln Sie eine $O(|N| \cdot |G|)$-Implementierung des Verfahrens zur Berechnung der EXACT_1-Mengen.*

b) *Versuchen Sie, Ihre Implementierung derart zu modifizieren, daß diese nur noch $O(|G|)$ Zeit benötigt.*

Sei $1 \leq l \leq k$ und seien alle EXACT_q-Mengen für $0 \leq q < l$ bereits berechnet. Unser Ziel ist nun die Berechnung der EXACT_l-Mengen. Sei $B \in N$. Dann gilt

$$\mathrm{EXACT}_l(B) = \bigcup_{B \to \alpha \in P} \mathrm{EXACT}_l(\alpha).$$

Also können wir $\mathrm{EXACT}_l(B)$ konstruieren, indem wir für alle Alternativen α von B $\mathrm{EXACT}_l(\alpha)$ berechnen und dann die Vereinigung dieser Mengen bilden. Wie wir uns bereits überlegt haben, erhalten wir für $\alpha = A_1 A_2 \dots A_r$

$$\mathrm{EXACT}_l(\alpha) = \bigcup_{j=0}^{\max_\varepsilon(\alpha)} \Big[\mathrm{EXACT}_l(A_{j+1}) \cdot \mathrm{EXACT}_0(A_{j+2} \dots A_r) \\ \cup \bigcup_{i=1}^{l-1} \mathrm{EXACT}_i(A_{j+1}) \cdot \mathrm{EXACT}_{l-i}(A_{j+2} \dots A_r) \Big].$$

Bis auf $\mathrm{EXACT}_l(A_{j+1})$ sind alle EXACT_q-Mengen auf der rechten Seite der obigen Gleichung berechnet, da für diese $q < l$ gilt. $\mathrm{EXACT}_l(A_{j+1})$ muß genau dann berechnet werden, wenn $\mathrm{EXACT}_0(A_{j+2} \dots A_r) = \{\varepsilon\}$.

Anscheinend sind wir in eine „Sackgasse “ geraten. Folgende Beobachtung zeigt jedoch, daß dies nicht der Fall ist:

Wir können obige Betrachtung bezüglich den Alternativen von A_{j+1} wiederholen. Nach maximal $|N|$ Wiederholungen ist die zu bearbeitende Variable bereits betrachtet, so daß das Verfahren abbricht.

Es stellt sich nun die Frage, wie wir all diese Betrachtungen effizient durchführen können. Hierzu konstruieren wir zunächst einen Graphen $H_{ex}^l = (V_1 \cup V_2^l, E^l)$. Die

Knoten in V_1 besitzen paarweise verschiedene Markierungen aus N. Im folgenden identifizieren wir jeden Knoten in V_1 mit seiner Markierung. Die Knoten in V_2^l haben paarweise verschiedene Markierungen vom Typ $\mathrm{EXACT}_i(C) \cdot \mathrm{EXACT}_{l-i}(\gamma)$, wobei $C \in V, \gamma \in V^*$, $1 \leq i < l$ und $\mathrm{EXACT}_i(C), \mathrm{EXACT}_{l-i}(\gamma) \neq \emptyset$. Für einen Knoten $u \in V_1^l$ bezeichne $M(u)$ die Menge seiner Markierungen.

Seien $B, A \in V_1$. Dann ist (B, A) genau dann eine Kante in E^l, wenn eine Produktion $A \to \delta_1 B \delta_2 \in P$ mit $\delta_1 \overset{*}{\Longrightarrow} \varepsilon$ und $\delta_2 \overset{*}{\Longrightarrow} \varepsilon$ existiert. Sei $u \in V_2^l$ ein Knoten mit der Markierung $M(u) = \mathrm{EXACT}_i(C) \cdot \mathrm{EXACT}_{l-i}(\gamma)$. Dann ist (u, A) genau dann eine Kante in E^l, wenn eine Produktion $A \to \delta C \gamma \in P$ mit $\delta \overset{*}{\Longrightarrow} \varepsilon$ existiert.

Da die Knoten in V_1 und die Kanten zwischen Knoten in V_1 unabhängig von l sind, müssen wir diesen Teil des Graphen nur einmal konstruieren. Da $\mathrm{EXACT}_i(C)$ und $\mathrm{EXACT}_{l-i}(\gamma)$ jeweils bereits berechnet sind, kann der restliche Teil des Graphen H_{ex}^l vor der Berechnung der EXACT_l-Mengen konstruiert werden. Folgendes Lemma beschreibt einige Eigenschaften des Graphen H_{ex}^l:

Lemma 6.1 *Sei $H_{ex}^l = (V_1 \cup V_2^l, E^l)$ wie oben definiert. Dann hat H_{ex}^l folgende Eigenschaften:*

1. $|V_1| = |N|$ *und* $|V_2^l| \leq (l-1)|G|$.
2. $|E^l| \leq l \cdot |G|$.
3. *Alle Knoten in V_2^l haben keinen Vorgänger.*
4. *Sei $C_1, C_2, \ldots, C_t, C_1$ ein Kreis in H_{ex}^l. Dann gilt $C_i \in V_1$ für $1 \leq i \leq t$ und* $\mathrm{EXACT}_l(C_1) = \mathrm{EXACT}_l(C_2) = \cdots = \mathrm{EXACT}_l(C_t)$.
5. *Für alle $B \in N$ gilt*

$$\mathrm{EXACT}_l(B) = \{z \in \Sigma^l \mid \exists u \in V_2^l \textit{ mit } z \in M(u) \textit{ und es existiert ein Pfad von } u \textit{ nach } B \textit{ in } H_{ex}^l\}.$$

Beweis:

1. folgt direkt aus der Definition des Graphen. Man beachte, daß ein Knoten in V_2^l zu einem Suffix der rechten Seite einer Produktion in P korrespondiert. Zu jedem solchen Suffix gibt es maximal $l-1$ korrespondierende Knoten in V_2^l.
2. Jede Kante zwischen Knoten in V_1 entspricht einer Produktion und jede Produktion $A \to \alpha$ korrespondiert zu maximal $|\alpha|$ solcher Kanten. Also ist die Anzahl dieser Kanten durch $|G|$ begrenzt. Jede Kante von einem Knoten in V_2^l zu einem Knoten in V_1 entspricht einer Produktion in P. Jede Produktion $A \to \alpha$ korrespondiert zu maximal $(l-1)|\alpha|$ solcher Kanten. Also ist die Anzahl dieser Kanten durch $(l-1)|G|$ beschränkt.

3. folgt offensichtlich direkt aus der Konstruktion.
4. und 5. ergeben sich unmittelbar aus der Konstruktion des Graphen H^l_{ex} und aus obigen Betrachtungen bezüglich $\mathrm{EXACT}_l(B)$.

■

Nun können wir $\mathrm{EXACT}_l(B)$, $B \in N$ berechnen, indem wir alle Knoten $u \in V^l_2$, von denen aus B in H^l_{ex} erreicht werden kann, bestimmen. Sei $R(B) \subseteq V^l_2$ diese Menge von Knoten. Das heißt,

$$R(B) = \{u \in V^l_2 \mid \exists \text{ Pfad von } u \text{ nach } B \text{ in } H^l_{ex}\ \}.$$

Dann gilt

$$\mathrm{EXACT}_l(B) = \bigcup_{u \in R(B)} M(u).$$

Auf folgende Art und Weise können wir gleichzeitig für alle $B \in N$ die Menge $R(B)$ berechnen. Zur Definition der starken Zusammenhangskomponenten eines Graphen siehe Kapitel 7.

(1) Berechne alle starken Zusammenhangskomponenten von H^l_{ex}. Schrumpfe jede starke Zusammenhangskomponente CC zu einem Superknoten s_{CC}. Für alle Knoten $w \notin CC$ füge genau dann die Kante (s_{CC}, w) bzw. (w, s_{CC}) zu E hinzu, wenn es einen Knoten $v \in CC$ mit $(v, w) \in E^l$ bzw. $(w, v) \in E^l$ gibt. Entferne alle Kanten (x, y) mit mindestens einer der Knoten x, y ist in CC aus E^l. Sei $\tilde{H}^l_{ex} = (\tilde{V}_1 \cup V^l_2, \tilde{E}^l)$ der resultierende Graph.
(2) Berechne die transitive Hülle des durch $\tilde{V}_1$ induzierten Teilgraphen von $\tilde{H}^l_{ex}$. Für $v \in \tilde{V}_1$ sei $T(v) = \{x \in \tilde{V}_1 \mid \exists \text{ Pfad von } x \text{ nach } v \text{ in } \tilde{H}^l_{ex}\}$.
(3) Für alle $B \in N$ berechne $R(B)$ wie folgt: Sei $v \in \tilde{V}_1$ derjenige Knoten, der zur starken Zusammenhangskommponente, die B enthält, korrespondiert. Führe dann folgende Zuweisung durch:

$$R(B) := \{u \in V^l_2 \mid \exists x \in T(v) : (u, x) \in \tilde{E}^l\}.$$

Die Korrektheit des obigen Verfahrens ergibt sich unmittelbar aus der Konstruktion. Schritt (1) kann durch den Standardalgorithmus (siehe Kapitel 7.2) in Zeit $O(|G|)$ durchgeführt werden. Schritt (2) benötigt höchstens $O(|N| \cdot |G|)$ Zeit. Lemma 6.1 impliziert, daß Schritt (3) in $O(|N|^2 l |G|)$ durchgeführt werden kann.

Übung 6.5: *Arbeiten Sie eine $O(|N|^2 l |G|)$-Implementierung des Verfahrens zur Berechnung aller $R(B)$, $B \in N$, aus.*

Nachdem $R(B)$ vorliegt, kann nun $\mathrm{EXACT}_l(B) = \bigcup_{u \in R(B)} M(u)$ leicht berechnet werden. Da $|R(B)| \leq (l-1)|G|$ und da $|M(u)| \leq |\Sigma|^l$ für jeden Knoten $u \in R(B)$, kann $\mathrm{EXACT}_l(B)$ in Zeit $O((l-1)|G|l|\Sigma|^l)$ berechnet werden. Eine sinnvolle Datenstruktur zur Speicherung einer EXACT_q-Menge ist ein Trie bezüglich des Alphabets Σ.

Nehmen wir an, daß für alle $B \in N$ die Menge $\mathrm{EXACT}_l(B)$ in Form eines Tries vorliegt. Wir betrachten eine beliebige Alternative $\alpha = A_1 A_2 \ldots A_r$ einer Variablen in N. Wegen

$$\begin{aligned}\mathrm{EXACT}_l(\alpha) = \bigcup_{j=0}^{max_\varepsilon(\alpha)} \Big[& \mathrm{EXACT}_l(A_{j+1}) \cdot \mathrm{EXACT}_0(A_{j+2} \ldots A_r) \\ & \cup \bigcup_{i=1}^{l-1} \mathrm{EXACT}_i(A_{j+1}) \cdot \mathrm{EXACT}_{l-1}(A_{j+2} \ldots A_r) \Big]\end{aligned}$$

kann der zu $\mathrm{EXACT}_l(\alpha)$ korrespondierende Trie mittels Verschmelzung von maximal $|\alpha| \cdot l$ Tries der Höhe l berechnet werden. Dabei können wir, ohne die benötigte Zeit um mehr als einen konstanten Faktor zu erhöhen, gleichzeitig für jeden Suffix α' von α den zu $\mathrm{EXACT}_l(\alpha')$ korrespondierenden Trie berechnen. Hierfür benötigen wir $O(|\alpha|l|\Sigma|^l)$ Zeit, was eine Gesamtzeit von $O(|G|l|\Sigma|^l)$ ergibt. Somit haben wir folgendes Lemma bewiesen:

Lemma 6.2 *Sei $k \in \mathbb{N}$ und $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik. Dann können sämtliche Tries, die zu einem $\mathrm{EXACT}_l(\alpha)$ mit $0 \leq l \leq k$ und $\alpha \in N$ oder α ist Suffix der rechten Seite einer Produktion korrespondieren, in $O\left(k^2 \cdot |G| \left(|N|^2 + |\Sigma|^k\right)\right)$ Zeit konstruiert werden.*

Übung 6.6: *Arbeiten Sie den Algorithmus zur Konstruktion der zu den EXACT-Mengen korrespondierenden Tries aus.*

Unter der Voraussetzung, daß die zu den EXACT-Mengen korrespondierenden Tries bereits berechnet sind, werden wir nun ein Verfahren zur Berechnung der FIRST-Mengen entwickeln. Sei $B \in N$. Dann gilt

$$\mathrm{FIRST}_l(B) = \mathrm{EXACT}_{\leq l}(B) \cup \mathrm{first}_l(B)$$

und

$$\mathrm{first}_l(B) = \bigcup_{B \to \alpha \in P} \mathrm{first}_l(\alpha).$$

Für $\alpha = A_1 A_2 \dots A_r$ erhalten wir

$$\mathrm{first}_l(\alpha) = \bigcup_{j=0}^{\max_\varepsilon(\alpha)} \Bigg[\mathrm{first}_l(A_{j+1}) \cup \bigcup_{i=1}^{l-1} \mathrm{EXACT}_i(A_{j+1})\,\mathrm{first}_{l-i}(A_{j+2} \dots A_r) \\ \cup \mathrm{EXACT}_l(A_{j+1}) \setminus \mathrm{EXACT}_l(A_{j+1}A_{j+2} \dots A_r)\Bigg].$$

Vergleichen wir obige Gleichung für $\mathrm{first}_l(\alpha)$ mit der für $\mathrm{EXACT}_l(\alpha)$ hergeleiteten Gleichung, dann fällt die große Ähnlichkeit auf. Im Grunde genommen können wir nun analog zur Berechnung der EXACT_l-Mengen verfahren. Dementsprechend sieht die Berechnung der first_l-Mengen folgendermaßen aus:

Zunächst konstruieren wir einen Graphen $H^l_{\mathit{fi}} = (V_1 \cup V^l_2, E^l)$. Analog zu oben korrespondiert V_1 zu N. Die Knoten in V^l_2 haben nun Markierungen vom Typ $\mathrm{EXACT}_i(C)\,\mathrm{first}_{l-i}(\gamma)$, wobei $C \in V$, $\gamma \in V^*$, $1 \leq i < l$ und $\mathrm{EXACT}_i(C)$, $\mathrm{first}_{l-i}(\gamma) \neq \emptyset$. Seien $B, A \in V_1$. Dann gilt

$$(B, A) \in E^l \iff \exists A \to \beta B \gamma \in P \text{ mit } \beta \stackrel{*}{\Longrightarrow} \varepsilon.$$

Sei $u \in V^l_2$ ein Knoten mit der Markierung $M(u) = \mathrm{EXACT}_i(C) \cdot \mathrm{first}_{l-i}(\gamma)$. Dann gilt

$$(u, A) \in E^l \iff \exists A \to \beta C \gamma \in P \text{ mit } \beta \stackrel{*}{\Longrightarrow} \varepsilon.$$

Genauso wie Lemma 6.1 kann folgendes Lemma bewiesen werden:

Lemma 6.3 *Sei $H^l_{\mathit{fi}} = (V_1 \cup V^l_2, E^l)$ wie oben definiert. Dann hat H^l_{fi} folgende Eigenschaften:*

1. $|V_1| = N$ *und* $|V^l_2| \leq (l-1)|G|$.
2. $|E| \leq l \cdot |G|$.
3. *Alle Knoten in V^l_2 haben keinen Vorgänger.*
4. *Sei $C_1, C_2, \dots, C_t, C_1$ ein Kreis in H^l_{fi}. Dann gilt*

$$\mathrm{first}_l(C_1) = \mathrm{first}_l(C_2) = \dots = \mathrm{first}_l(C_t).$$

5. *Für alle $B \in N$ gilt*

$$\mathrm{first}_l(B) = \{z \in \Sigma^l \mid \exists u \in V^l_2 \text{ so daß } z \in M(u) \text{ und es einen Pfad von } u \text{ nach } B \text{ in } H^l_{\mathit{fi}} \text{ gibt}\}.$$

Sei nun $R(B) = \{u \in V_2^l \mid \exists$ Pfad von u nach B in $H_{fi}^l\}$. Dann gilt

$$\mathrm{first}_l(B) = \bigcup_{u \in R(B)} M(u).$$

$R(B)$ bezüglich H_{fi}^l wird genauso berechnet wie $R(B)$ bezüglich H_{ex}^l. Die Berechnung der korrespondierenden Tries zu $\mathrm{first}_l(B)$, $B \in N$ bzw. $\mathrm{first}_l(\alpha)$, α ist Suffix der rechten Seite einer Produktion in P ist nahezu identisch zur Berechnung der zu EXACT_l-Mengen korrespondierenden Tries. Genauso wie oben überlegen wir uns, daß die benötigte Zeit $O(k^2|G|(|N|^2 + |\Sigma|^k))$ ist.

Sei $\alpha = A_1 A_2 \ldots A_r$. Wegen

$$\begin{aligned}\mathrm{FIRST}_l(\alpha) = \mathrm{EXACT}_0(\alpha) \cup \bigcup_{j=0}^{\max_\varepsilon(\alpha)} \Big[& \mathrm{first}_l(A_{j+1}) \cup \mathrm{EXACT}_l(A_{j+1}) \cup \\ & \bigcup_{i=1}^{l-1} \mathrm{EXACT}_i(A_{j+1}) \cdot \mathrm{FIRST}_{l-i}(A_{j+1} \ldots A_r) \Big]\end{aligned}$$

kann der Trie für $\mathrm{FIRST}_l(\alpha)$, wobei α die rechte Seite einer Produktion in P ist, durch Verschmelzung von maximal $|\alpha| \cdot (l+1)$ Tries der Höhe $\leq l$ berechnet werden. Auch hier können wir, ohne die benötigte Zeit um mehr als einen konstanten Faktor zu erhöhen, für jeden Suffix α' von α den zu $\mathrm{FIRST}_l(\alpha')$ korrespondierenden Trie berechnen. Insgesamt erhalten wir folgendes Lemma:

Lemma 6.4 *Sei $k \in \mathbb{N}$ und $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik. Dann können sämtliche Tries, die zu einem $\mathrm{FIRST}_l(\alpha)$ mit $0 \leq l \leq k$ und $\alpha \in N$ oder α ist Suffix der rechten Seite einer Produktion korrespondieren, in $O(k^2 \cdot |G|(|N|^2 + |\Sigma|^k))$ Zeit konstruiert werden.*

Es verbleibt noch die Berechnung der FOLLOW-Mengen. Gemäß Definition gilt für $\beta \in V^*$

$$\mathrm{FOLLOW}_k(\beta) = \{u \in \Sigma^* \mid S \underset{\mathrm{lm}}{\overset{*}{\Longrightarrow}} \alpha\beta\delta \text{ und } u \in \mathrm{FIRST}_k(\delta)\}.$$

Für das Item $A \to \delta B.\gamma$ definieren wir

$$\begin{aligned}&\mathrm{FOLLOW}_k(A \to \delta B.\gamma) = \\ &\qquad \{u \in \Sigma^* \mid S \underset{\mathrm{lm}}{\overset{*}{\Longrightarrow}} wA\mu \Longrightarrow w\delta B.\gamma\mu \text{ und } u \in \mathrm{FIRST}_k(\gamma\mu)\}.\end{aligned}$$

Dann gilt

$$\mathrm{FOLLOW}_k(B) = \bigcup_{\substack{A \to \delta B.\gamma \\ \text{Item}}} \mathrm{FOLLOW}_k(A \to \delta B.\gamma).$$

Ferner gilt

$$\begin{aligned}\mathrm{FOLLOW}_k(A \to \delta B.\gamma) &= \mathrm{first}_k(\gamma) \cup \bigcup_{i=0}^{k} \mathrm{EXACT}_i(\gamma) \cdot \mathrm{FOLLOW}_{k-i}(A) \\ &= \mathrm{first}_k(\gamma) \cup \bigcup_{i=1}^{k-1} \mathrm{EXACT}_i(\gamma) \cdot \mathrm{FOLLOW}_{k-i}(A) \\ &\quad \cup \mathrm{EXACT}_k(\gamma) \cup \mathrm{EXACT}_0(\gamma) \cdot \mathrm{FOLLOW}_k(A).\end{aligned}$$

Auch hier ist es sinnvoll, beginnend mit $l = 1$, für wachsende l die FOLLOW_l-Mengen zu berechnen. Wiederum fällt die Ähnlichkeit der obigen Gleichung zu denen zur Berechnung der EXACT_l- bzw. first_l-Mengen hergeleiteten Gleichungen auf. Dementsprechend kann analog verfahren werden. Dies bleibt den Lesern als Übung überlassen.

Übung 6.7: *Entwickeln Sie einen Algorithmus zur Konstruktion der* FOLLOW-*Mengen. Welchen Platz und welche Zeit benötigt Ihr Algorithmus?*

Sei $G = (V, \Sigma, P, S)$ eine $SLL(k)$-Grammatik. Wir sind nun in der Lage, einen $SLL(k)$-Parser zu generieren. Üblicherweise kann dieser durch einen Kellerautomaten mit Ausgabe modelliert werden. Wir werden nun einen $SLL(k)$-Parser informell beschreiben. Abbildung 6.2 skizziert seine Struktur.

Dabei bezeichnet auf dem Eingabeband w die bereits gelesene Eingabe, u die nächsten k zu lesenden Symbole und v die restliche Eingabe. Auf das Ausgabeband schreibt der Parser eine Folge von Produktionennummern, die im Fall des Akzeptierens die Linksableitung ergeben.

Die *Parsertabelle* realisiert eine Abbildung

$$PT : N \times \Sigma^{\leq k} \to \{1, 2, \ldots, |P|\} \cup \{\text{error}\}.$$

Die nächste anzuwendende Produktion ergibt sich dann gemäß dem obersten Kelleritem Z und dem Lookahead u aus der Parsertabelle PT. Steht in Z vor der Variablen A der Bearbeitungspunkt, dann steht in $PT(A, u)$ die Nummer der anzuwendenden Produktion.

Nehmen wir an, daß wir bezüglich G für jede Alternative β den Trie für $\mathrm{FIRST}_k(\beta)$ und für jede Variable A den Trie für $\mathrm{FOLLOW}_k(A)$ berechnet haben. Da gemäß der

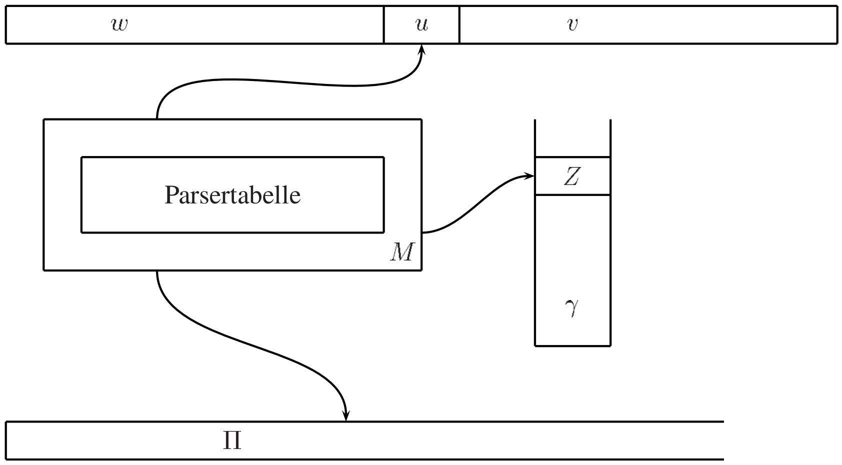

Abbildung 6.2: Struktur eines $SLL(k)$-Parsers.

Definition der $SLL(k)$-Grammatik für alle $A \in N$ für alle verschiedenen Alternativen β und γ für A

$$\mathrm{FIRST}_k(\beta) \oplus_k \mathrm{FOLLOW}_k(A) \cap \mathrm{FIRST}_k(\gamma) \oplus_k \mathrm{FOLLOW}_k(A) = \emptyset,$$

kann die Parsertabelle PT nun leicht berechnet werden.

Übung 6.8: *Entwickeln Sie einen Algorithmus, der für eine gegebene $SLL(k)$-Grammatik $G = (V, \Sigma, P, S)$ die Parsertabelle PT, bei gegebenen Tries für die FIRST_k- sowie die FOLLOW_k-Mengen, konstruiert. Wie groß kann PT in Abhängigkeit von $|G|$ im schlimmsten Fall werden? Analysieren Sie den benötigten Speicherplatz und die benötigte Rechenzeit Ihres Algorithmus.*

Gegeben eine $LL(k)$-Grammatik $G = (V, \Sigma, P, S)$ könnten wir analog zu oben mittels Konstruktion einer Parsertabelle einen $LL(k)$-Parser für G konstruieren. Dies kann jedoch zu sehr großen Parsertabellen und daher zu einem ineffizienten Parser führen. Es gibt $LL(k)$-Grammatiken, die notwendigerweise nur Parsertabellen, die in der Grammatikgröße exponentielle Größe haben, besitzen. $SLL(k)$-Grammatiken haben immer eine Parsertabelle polynomieller Größe. Dies liegt daran, daß im Gegensatz zu $SLL(k)$-Grammatiken die Parsertabelle einer beliebigen $LL(k)$-Grammatik auch von der bereits gelesenen Eingabe abhängen kann.

Unser Ziel ist es nun, unter Verwendung von etwas aufwendigeren Datenstrukturen als Tabellen und der Möglichkeit, die bereits konstruierte Linksableitung zu modifizieren, auch für $LL(k)$-Grammatiken Parser polynomieller Größe zu konstruieren. Derartige Parser nennen wir *erweiterte $LL(k)$-Parser*. Zur Erläuterung der Schwierigkeiten, die dabei auftreten, und deren effizienten Bewältigung betrachten wir zunächst ein Beispiel.

Beispiel 6.3 Sei $n \in \mathbb{N}$ und die kontextfreie Grammatik $G_n = (V_n, \Sigma_n, P_n, A_0)$ definiert durch:

$\Sigma_n = \{a_i, b_i, c_i, d_i \mid 1 \leq i \leq n\}$,
$V_n \setminus \Sigma_n = \{A_i, B_j, C_j \mid 0 \leq i \leq n, 1 \leq j \leq n\}$ und

$$\begin{aligned} P_n : A_i &\to a_{i+1}A_{i+1}B_{i+1}|d_{i+1}A_{i+1}C_{i+1} && (0 \leq i \leq n-1) \\ A_n &\to b_i|\varepsilon && (1 \leq i \leq n) \\ B_i &\to b_ic_i|\varepsilon && (1 \leq i \leq n) \\ C_i &\to c_i|\varepsilon && (1 \leq i \leq n). \end{aligned}$$

Mit Hilfe des Satzes 6.1 kann leicht gezeigt werden, daß G_n eine $LL(2)$-Grammatik ist.

Übung 6.9: *Zeigen Sie, daß G_n eine $LL(2)$-Grammatik ist.*

Wir betrachten nun die Linksableitung eines Strings $x_1x_2 \ldots x_nb_jc_j \ldots$, wobei $x_i \in \{a_i, d_i\}$, $1 \leq i \leq n$. In dem Augenblick, wenn der Parser die richtige Alternative für A_n bestimmen soll, benötigt er die Information, ob $X_j = B_j$, d.h. $x_j = a_j$ oder ob $X_j = C_j$ d.h. $x_j = d_j$. Falls $X_j = B_j$, dann wird b_jc_j aus der Variablen B_j abgeleitet und ε ist die Alternative für A_n. Andernfalls wird c_i aus C_j abgeleitet und b_j ist die korrekte Alternative für A_n.

Prinzipiell gibt es zwei Möglichkeiten, diese Information zu erhalten:

1. Der Parser hat diese Information im aktuellen Zustand q und dem obersten Kellersymbol kodiert. Dann muß die Anzahl der verschiedenen Paare (q, Z) exponentiell in n sein, da es 2^n verschiedene Strings $x_1x_2 \ldots x_n$ mit $x_i \in \{a_i, d_i\}$, $1 \leq i \leq n$ gibt. Dies würde zu einem Parser, dessen Größe exponentiell in der Grammatikgröße ist, führen.
2. Der Parser schaut im Keller nach, ob dort B_j oder C_j steht. Verlangt man nun, daß die zu A_n korrespondierende Produktion vor jeder zu einem B_l oder C_l korrespondierenden Produktion ausgedruckt wird und die über B_j bzw. C_j im Keller stehende Information im Zustand q und dem obersten Kellersymbol Z gespeichert wird, dann werden exponentiell viele verschiedene Paare (q, Z) benötigt. Dies folgt aus der Beobachtung, daß in der Linkssatzform $x_1x_2 \ldots x_nA_nX_n$ $X_{n-1} \ldots X_{j+1}X_j \ldots$ insgesamt 2^{n-j} verschiedene Strings $X_nX_{n-1} \ldots X_{j+1}$, $X_l \in \{B_l, C_l\}$, $n \geq l > j$, möglich sind. Demnach erhielten wir wiederum einen Parser, dessen Größe exponentiell in der Grammatikgröße ist.

Erlauben wir jedoch, daß wir den Ausdruck der zu A_n korrespondierenden Produktion auf später verschieben, d.h., uns lediglich merken, daß dieser noch zu erfolgen hat, und alle möglichen Berechnungen der Linksableitung parallel fortsetzen, dann erhalten wir auf einfache Art und Weise einen Parser, dessen Größe lediglich linear in der Grammatikgröße ist.

Die richtige Alternative für die ersten n Variablen $A_0, A_1, \ldots, A_{n-1}$ ergeben sich jeweils unmittelbar aus dem ersten noch nicht gelesenen Symbol der Eingabe. Die korrespondierenden Produktionen können auch direkt in die Linksableitung geschrieben werden.

Betrachte nun den Augenblick, in dem A_n expandiert werden soll. Falls das erste noch nicht gelesene Symbol in $\{c_1, c_2, \ldots, c_n\}$ liegt, ist klar, daß ε die richtige Alternative für A_n ist. Falls das erste noch nicht gelesene Symbol b_j, $1 \leq j \leq n$, und das zweite noch nicht gelesene Symbol $\neq c_j$ sind, dann muß b_j die richtige Alternative für A_n sein. In beiden Fällen kann die korrespondierende Produktion direkt in die Linksableitung geschrieben werden. Falls die beiden ersten noch nicht gelesenen Symbole b_jc_j, $1 \leq j \leq n$, sind, dann merkt sich der Parser, daß die zu A_n korrespondierende Produktion noch in die Linksableitung geschrieben werden muß und setzt parallel die Konstruktion der beiden möglichen Linksableitungen fort. Für jede Variable in $X_nX_{n-1}\ldots X_{j+1}$ weiß der Parser, daß ε die richtige Alternative ist und kann nacheinander die korrespondierenden Produktionen in die Linksableitung schreiben.

Betrachte nun den Augenblick, in dem X_j expandiert werden soll. Falls $X_j = B_j$, dann ist b_jc_j die richtige Alternative für X_j und ε die richtige Alternative für A_n. Andernfalls, d.h. $X_j = C_j$, ist c_j die richtige Alternative für X_j und b_j die richtige Alternative für A_n. In beiden Fällen können die korrespondierenden Produktionen in die Linksableitung geschrieben werden. Damit die zu A_n korrespondierende Produktion an die korrekte Stelle in der Linksableitung geschrieben werden kann, hat sich der Parser diese mittels eines Zeigers gemerkt.

Die richtigen Alternativen der restlichen Variablen ergeben sich wiederum jeweils unmittelbar aus dem ersten noch nicht gelesenen Symbol der Eingabe. Also können die korrespondierenden Produktionen direkt in die Linksableitung geschrieben werden.

Übung 6.10: *Konstruieren Sie einen erweiterten $LL(2)$-Parser für G_n, dessen Größe $O(n)$ ist.*

♦

Unser Ziel ist es nun, die oben anhand des Beispiels skizzierte Vorgehensweise derart auszuarbeiten, daß diese für beliebige $LL(k)$-Grammatiken anwendbar ist. Folgende Fragen müssen wir hierbei beantworten:

1. Wieviele Linksableitungen müssen im schlimmsten Fall gleichzeitig bearbeitet werden?
2. Welche Datenstrukturen unterstützen eine effiziente parallele Fortsetzung aller möglichen Linksableitungen?
3. Wie ist die Ausgabe zu behandeln?

Sei $G = (V, \Sigma, P, S)$ eine $LL(k)$-Grammatik. Sei M_G der in Kapitel 6.2.2 für die Grammatik G konstruierte Kellerautomat. Wir werden nun den Parser P_G für G konstruieren, indem wir M_G sukzessive erweitern. Nehmen wir hierfür an, daß w die bereits gelesene Eingabe, x derjenige Teilstring der Länge k der Eingabe, der auf w folgt und $[X \to \beta.Ay]$, $A \in N$ das oberste Kellersymbol sind. D.h., M_G will die Variable A expandieren. Abbildung 6.3 skizziert diese Situation. Nehmen wir

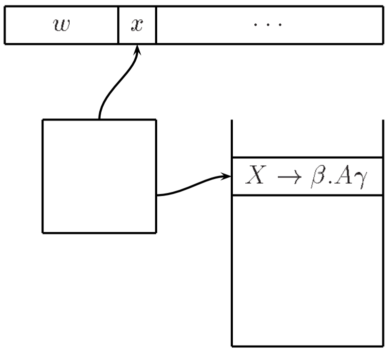

Abbildung 6.3: Kellerautomat.

ferner an, daß P_G kein Symbol von $x = x_1x_2\dots x_k$ kennt. Ziel ist es nun, diejenigen Teilstrings von x, die exakt aus A ableitbar sind, zu bestimmen.

Falls $x \in FIRST_k(A)$, dann impliziert die $LL(k)$-Eigenschaft, daß genau eine Alternative β von A mit $x \in FIRST_k(\beta)$ existiert und diese die korrekte Alternative für A ist.

Falls $x \notin FIRST_k(A)$, dann kann nur ein echter Präfix x' von x mit $x' \in FIRST_k(A)$ aus A abgeleitet sein. Die Idee ist nun, alle diese Präfixe parallel zu bearbeiten und die Betrachtung mit dem ersten Symbol von γ fortzusetzen.

Also ist es sinnvoll, daß P_G den maximalen Präfix u von x, der auch Präfix eines Elementes in $\mathrm{FIRST}_k(A)$ ist, zu bestimmen. Zur effizienten Lösung dieser Aufgabe nehmen wir an, daß P_G für jede Variable $X \in N$ den zu $\mathrm{FIRST}_k(X)$ korrespondierenden Trie $T_k(X)$ enthält. P_G liest nun parallel x und den dazu korrespondierenden String in $T_k(A)$, bis der maximale Präfix u von x in $T_k(A)$ bestimmt ist. Zwei Fälle können auftreten:

1. u ist ein echter Präfix von x, d.h. $|u| < k$ oder
2. $u = x$, d.h. $|u| = k$.

Fall 1: $|u| = k$.

Dann gilt $x \in \mathrm{FIRST}_k(A)$. Da G eine $LL(k)$-Grammatik ist, gilt für jeden echten Präfix u' von x mit $u' \in \mathrm{FIRST}_k(A)$, daß dieser in der Linksableitung für die Eingabe von P_G nicht der Terminalstring sein kann, der aus A abgeleitet wird. Wegen der $LL(k)$-Eigenschaft gibt es somit höchstens eine Alternative β von A mit $x \in \mathrm{FIRST}_k(\beta)$, so daß P_G den richtigen Expansionsschritt durchführen kann.

Fall 2: $|u| < k$.

Aus der Konstruktion folgt direkt, daß der Terminalstring, der in der Linksableitung für die Eingabe des Parsers aus A abgeleitet wird, ein Präfix u' von u ist. Aus der Definition von $\mathrm{FIRST}_k(A)$ folgt ferner, daß $u' \in \mathrm{FIRST}_k(A)$. Also ist es nützlich, wenn P_G auf derartige Präfixe direkten Zugriff hat. Hierzu enthält jeder Knoten $v \in T_k(A)$ für denjenigen Knoten $w \in T_k(A)$ mit

1. $s(w) \in \mathrm{FIRST}_k(A)$, wobei $s(w)$ den zu w korrespondierenden String bezeichnet und
2. w ist der zu v nächstliegende Knoten auf dem Pfad von der Wurzel zu v, der 1. erfüllt

einen Zeiger, der auf w zeigt. Da P_G noch nicht ganz x kennt, kann P_G nicht notwendigerweise entscheiden, welcher dieser $\leq k$ Präfixe derjenige ist, der aus A in der Linksableitung für die Eingabe abgeleitet wird. Also muß P_G sich all diese Möglichkeiten merken. Hierzu initialisiert P_G für jeden dieser Terminalstrings eine Schlange, bestehend aus einer Komponente. Diese enthält

- die Variable A,
- den Terminalstring u' und
- einen Zeiger auf diejenige Stelle in der Linksableitung, an die gegebenfalls die Ableitung von u' aus A geschrieben wird.

Daß die Schlange für unseren Zweck eine vernünftige Datenstruktur ist, werden wir gleich sehen.

Nun setzt P_G mit dem ersten noch nicht expandierten bzw. nicht gelesenen Zeichen im Keller seine Arbeit fort. Da für jedes Item im Keller alle Symbole vor dem Bearbeitungspunkt bereits abgearbeitet sind und das Symbol direkt hinter dem Bearbeitungspunkt eine bereits expandierte Variable ist, steht das gesuchte Symbol an zweiter Stelle hinter dem Bearbeitspunkt.

Also führt P_G solange Pop-Operationen durch, bis das oberste Kellerelement hinter dem Bearbeitungszeitpunkt mindestens zwei Symbole besitzt.

P_G arbeitet nun mit dem zweiten Symbol B hinter dem Bearbeitungspunkt des obersten Items im Keller weiter. Dieses ist entweder eine noch nicht expandierte Variable oder ein noch nicht durch einen Leseschritt verarbeitetes Terminalzeichen. In beiden Fällen arbeitet P_G nacheinander alle Schlangen ab.

Nehmen wir an, daß P_G gerade die Schlange S betrachtet und sei j die Länge desjenigen Präfixes von x, der bezüglich der Schlange S bereits abgeleitet ist. Wir diskutieren nun die Fälle $B \in N$ und $B \in \Sigma$ nacheinander.

a) $B \in \Sigma$

Falls $B = x_{j+1}$, dann erweitern wir S. Die neue Komponente enthält das Terminalzeichen x_{j+1}. Falls $B \neq x_{j+1}$, dann ist P_G bezüglich der Schlange S in eine Sackgasse geraten, so daß die Schlange S vollständig gelöscht werden kann.

b) $B \in N$

Auch hier ist es sinnvoll, daß P_G den maximalen Präfix der nächsten $k-j$ Symbole der Eingabe, der auch Präfix eines Elementes in $\mathrm{FIRST}_k(B)$ ist, bestimmt. Im Gegensatz zu oben sind jedoch mitunter einige dieser $k-j$ Symbole bekannt. Bereits gelesene Symbole von x sollen nicht erneut gelesen werden. Wünschenswert ist es vielmehr, einen effizienten Zugriff auf den „richtigen“ Knoten in $T_k(B)$ zu haben, um dann analog zu oben weiter zu verfahren.

Hierzu erweitern wir P_G um einen Trie T_G für Σ^k. P_G manipuliert einen Zeiger $P(T_G)$, der zu jedem Zeitpunkt auf denjenigen Knoten $u \in T_G$ zeigt, für den gilt: $s(u)$ ist der bisher gelesene Präfix von x.

Für $v \in T$ bezeichne $t(v)$ die Tiefe von v in T_G und $s_i(v)$, $0 \leq i < t(v)$, denjenigen Suffix von $s(v)$, der mit dem $(i+1)$-ten Symbol von $s(v)$ beginnt. Um jeweils einen effizienten Zugriff auf den richtigen Knoten in $T_k(B)$ zu gewährleisten, enthält jeder Knoten $v \in T_G$ für alle $A \in N$ für $1 \leq i < t(v)$ einen Zeiger $P_{i,A}(v)$, der auf denjenigen Knoten $w \in T_k(A)$ zeigt, für den gilt: $s(w)$ ist der maximale Präfix eines Elementes von $\mathrm{FIRST}_k(A)$, der auch Präfix von $s_i(v)$ ist.

Über den Zeiger $P_{j,B}(u)$ haben wir nun direkten Zugriff auf den richtigen Knoten w in $T_k(B)$.

Falls nun $s(w) \neq s_j(u)$, dann ist $s(w)$ der maximale Präfix von $s(u)$, der auch Präfix eines Elementes von $\mathrm{FIRST}_k(B)$ ist, und P_G verfährt analog zu oben. Im Gegensatz zu oben initialisiert P_G keine neuen Schlangen, sondern erweitert S geeignet. Im Fall, daß $|\{u' \mid u' \text{ ist Präfix von } s(w) \text{ und } u' \in \mathrm{FIRST}_k(B)\}| > 1$, ist es sinnvoll, S nicht zu kopieren, sondern alle Schlangen kompakt, ähnlich wie wir es mit den Kellern bei der deterministischen Simulation von M_G getan haben, zu realisieren. Hierzu erhält der zum Schwanz der Schlange S korrespondierende Knoten für jeden Präfix u' von $s(w)$ mit $u' \in \mathrm{FIRST}_k(B)$ einen neuen direkten Nachfolger, der zu u' korrespondiert. Falls kein Präfix von $s(w)$ in $\mathrm{FIRST}_k(B)$ existiert, dann ist P_G bezüglich der Schlange S in eine Sackgasse geraten, so daß die Schlange S vollständig gelöscht werden kann.

Übung 6.11: *Arbeiten Sie eine kompakte Realisierung der Schlangen aus.*

Falls $s(w) = s_j(u)$, dann liest P_G parallel x und den dazu korrespondierenden String in $T_k(B)$ weiter.

Insgesamt ist die Datenstruktur zur Repräsentation der Schlangen ein Wald. Der Pfad von der Wurzel eines Baumes im Wald zu einem Blatt korrespondiert zu einer aktuellen Schlange und umgekehrt. Die Wurzel ist der Kopf und das Blatt der Schwanz der Schlange. Jeder Knoten auf dem Pfad vom Kopf zum Schwanz der Schlange korrespondiert zu einem Teilstring von x. Wir erhalten den zur Schlange korrespondierenden Präfix von x, indem wir auf dem Pfad vom Kopf zum Schwanz die zu den Knoten korrespondierenden Teilstrings konkatenieren.

Als nächstes werden wir uns überlegen, daß die Anzahl der Schlangen stets $\leq k$ ist. Dabei wird wesentlich verwendet, daß eine $LL(k)$-Grammatik eindeutig ist. Dies ist leicht zu zeigen und bleibt den Lesern als Übung überlassen.

Übung 6.12: *Zeigen Sie, daß eine $LL(k)$-Grammatik eindeutig ist.*

Lemma 6.5 *Die Anzahl der Schlangen ist stets $\leq k$.*

Beweis: Wir betrachten nun immer diejenigen Zeitpunkte, in denen alle Schlangen bezüglich des aktuellen Symbols initialisiert bzw. abgearbeitet sind. Jeder dieser Zeitpunkte korrespondiert zu dem bis dahin abgearbeiteten String $\gamma = A\gamma' \in NV^*$, wobei für A die richtige Alternative noch nicht ermittelt ist. Die Schlangen korrespondieren zu paarweise verschiedenen Ableitungen eines echten Präfixes von x aus γ. Es gibt k verschiedene echte Präfixe von x. Beachte, daß ε ein Präfix der Länge 0 von x ist.

Nehmen wir an, daß zwei verschiedene Schlangen zur Ableitung desselben Präfixes von x aus α korrespondieren. Dann könnten wir, da G reduziert ist, ein Element von $L(G)$ auf unterschiedliche Art und Weise aus dem Startsymbol ableiten. Dies widerspricht der Eindeutigkeit der Grammatik G. Also korrespondiert zu jedem Präfix von x maximal eine Schlange. Somit ist die Anzahl der Schlangen $\leq k$.

■

Somit haben wir auch bewiesen, daß die Anzahl der parallel zu bearbeitenden Ableitungen stets $\leq k$ ist.

Als nächstes möchten wir die Länge der Schlangen begrenzen. Wenn wir dafür sorgen, daß, bis auf den Schwanz einer Schlange, jeder Knoten zu einem String der Länge

> 0 korrespondiert, dann enthält jede Schlange höchstens k Knoten. Um dies zu erreichen, schreibt der Parser stets, wenn ε derjenige String sein kann, der aus der gerade betrachteten Variable C abgeleitet wird, die eindeutige Ableitung von ε aus C in die zu konstruierende Linksableitung. Falls aus C ein anderer String u abgeleitet wird, dann wird später die falsche Ableitung von ε aus C durch die korrekte Ableitung von u aus C überschrieben werden. Andernfalls hat der Parser in die Linksableitung die korrekten Produktionen geschrieben. Wir benötigen die Möglichkeit, daß der Schwanz einer Schlange zum Terminalstring ε korrespondieren kann, da der Vater des korrespondierenden Knotens auch noch einen weiteren Sohn besitzen kann und da ε auch derjenige Präfix von x, der durch die Schlange repräsentiert wird, sein kann. Gemäß unserer Konstruktion benötigen Knoten, die zu ε korrespondieren, keinen Pointer auf eine Position in der bisher konstruierten Linksableitung. Falls der Schwanz einer betrachteten Schlange zu ε korrespondiert, dann erhält nicht der Schwanz, sondern sein Vater im Baum neue direkte Nachfolger.

Insgesamt haben wir gezeigt, daß die Anzahl der Knoten innerhalb einer Schlange maximal k ist, woraus direkt folgendes Lemma folgt:

Lemma 6.6 *Die Anzahl der Knoten im Wald und somit auch die Anzahl der Zeiger aus dem Wald in die bereits konstruierte Linksableitung ist stets* $\leq k^2$.

Der früheste Zeitpunkt, zu dem der Präfix u' von x, der in der Linksableitung aus A abgeleitet wird, feststeht, ist dann, wenn der Wald aus genau einem Baum besteht. Dieser Präfix korrespondiert dann zur Wurzel r dieses Baumes. Da die zugrundeliegende Grammatik G eine $LL(k)$-Grammatik ist, ist dies spätestens dann der Fall, wenn das letzte Symbol x_k des Lookaheads gelesen ist. P_G modifiziert nun die bereits konstruierte Linksableitung sowie den Wald auf folgende Art und Weise:

(1) Über den in der Wurzel stehenden Zeiger ermittelt P_G die richtige Position in der bisher konstruierten Linksableitung. Dort wird die eindeutig bestimmte Linksableitung von u' aus A hingeschrieben. Wir können annehmen, daß wir diese im voraus berechnet und gespeichert haben. Dabei wird eventuell die falsche Ableitung von ε aus A überschrieben.

(2) Die Wurzel r des Baumes wird entfernt, wodurch dieser eventuell in mehrere Bäume zerfällt.

Da wir den Präfix u' von x entfernt haben, verschiebt sich der Anfang des Lookaheads um $|u'|$ Positionen nach rechts. Dementsprechend müssen wir den Zeiger $P(T_G)$ umsetzen. Um dies in konstanter Zeit durchführen zu können, enthält jeder Knoten $v \in T_G$ für $1 \leq i < t(v)$ einen Zeiger $P_i(v)$, der auf denjenigen Knoten $w \in T_G$ mit $s(w) = s_i(v)$ zeigt. Über den Zeiger $P_{|u'|}(v)$ haben wir nun direkten Zugriff auf denjenigen Knoten, auf den wir $P(T_G)$ setzen müssen. Nach Umsetzen des Zeigers

$P(T_G)$ setzt P_G die Berechnung fort. Zu analysieren verbleibt noch der für die Tries benötigte Speicherplatz sowie die vom Parser für die Konstruktion der Linksableitung benötigte Zeit.

Jeder der $|N| + 1$ Tries besteht aus weniger als $2|\Sigma|^k$ Knoten. Wie man sich leicht überlegt, ist die Anzahl der Zeiger pro Knoten in T_G maximal $(|N| + 1)k$ und in $T_k(A)$, $A \in N$, maximal k. Somit genügt $O(k|N||\Sigma|^k)$ Platz zur Speicherung aller Tries. Während jedem Lese- bzw. Expansionsschritt werden maximal k Schlangen abgearbeitet. Da hierdurch die Anzahl der Schlangen nicht größer als k wird, ist die Gesamtzeit für den Aufbau der Schlangen pro Schritt durch $O(k)$ beschränkt. Für den Abbau von Schlangen kann insgesamt nicht mehr Zeit als für den Aufbau benötigt werden. Somit ergibt sich eine Laufzeit von $O(k \cdot \text{Länge der Ableitung})$. Insgesamt haben wir folgenden Satz bewiesen:

Satz 6.3 *Sei $G = (V, \Sigma, P, S)$ eine $LL(k)$-Grammatik. Dann kann ein Parser P_G für G konstruiert werden, der folgende Eigenschaften hat:*

1. *Die Eingabe wird einmal von links nach rechs gelesen.*
2. *Der für die Spezifikation des Parsers benötigte Platz ist $O(k|N||\Sigma|^k)$.*
3. *P_G benötigt Platz für maximal k^2 Zeiger.*
4. *Die für eine gegebene Eingabe zur Konstruktion der Linksableitung benötigte Zeit ist $O(k \cdot \text{Länge der Linksableitung})$.*

Übung 6.13*:

a) Erweitern Sie den Algorithmus zur Berechnung der EXACT-Mengen derart, daß die vom Parser P_G benötigten Ableitungen $A \underset{lm}{\overset{*}{\Longrightarrow}} u$ mit $|u| < k$ auch berechnet werden. Wie groß ist die zusätzlich benötigte Zeit?

b) Arbeiten Sie einen Parsergenerator für $LL(k)$-Grammatiken aus. Welche Zeit benötigt Ihr Generator für die Generierung eines erweiterten Parsers P_G für eine gegebene $LL(k)$-Grammatik $G = (V, \Sigma, P, S)$?

6.4.2 $LR(k)$-Grammatiken und $LR(k)$-Parser

Sei $G = (V, \Sigma, P, S)$ eine eindeutige, reduzierte kontextfreie Grammatik. Betrachte für $x = x_1x_2 \dots x_n \in L(G)$ den zur Ableitung von x aus S korrespondierenden eindeutigen Ableitungsbaum $T(x)$. Alle bisherigen Verfahren zur kontextfreien Analyse lesen die Eingabe von links nach rechts und arbeiten dabei $T(x)$ top down von links nach rechts, also in Präordnung, ab. Dies entspricht der Vorwärtskonstruktion der eindeutigen Linksableitung von x aus S.

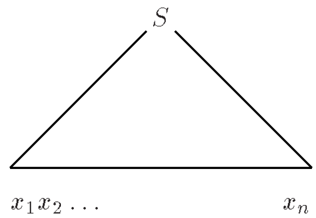

Abbildung 6.4: Ableitungsbaum $T(x)$ für ein $x \in L(G)$.

Denkbar ist auch folgende Vorgehensweise: Die Eingabe wird von links nach rechts gelesen und gleichzeitig $T(x)$ bottom up von links nach rechts, also in Postordnung, abgearbeitet. Dies entspricht der Rückwärtskonstruktion der eindeutigen Rechtableitung von x aus S.

Es stellt sich nun die Frage, wie ein Kellerautomat, der die Syntaxanalyse auf diese Art und Weise durchführt, aussieht. Wir geben eine informelle Beschreibung dieses Kellerautomaten an und überlassen die formale Definition den Lesern als Übung.

Nehmen wir an, daß

$$S \underset{\mathrm{rm}}{\Longrightarrow} \alpha_0 \underset{\mathrm{rm}}{\Longrightarrow} \alpha_1 \underset{\mathrm{rm}}{\Longrightarrow} \ldots \underset{\mathrm{rm}}{\Longrightarrow} \alpha_{m-1} \underset{\mathrm{rm}}{\Longrightarrow} \alpha_m = x$$

die Rechtsableitung für x aus S ist. Der Kellerautomat startet mit der Rechtssatzform $\alpha_m = x$ als Eingabe und konstruiert nacheinander die Rechtssatzformen α_{m-1}, α_{m-2}, $\ldots$, α_1, α_0, S. Dabei ist er derart organisiert, daß der Kellerinhalt, von unten nach oben gelesen, konkateniert mit der noch zu lesenden Eingabe gerade die aktuelle Rechtssatzform ergibt. Demnach ist zu Beginn der Keller leer. Seien y die noch zu lesende Eingabe und $\alpha_i = \gamma y$ die aktuelle Rechtssatzform. Dann steht γ von unten nach oben im Keller. Unser Ziel ist nun, aus α_i die Satzform α_{i-1} zu konstruieren. Abbildung 6.5 skizziert diese Situation.

Falls $\alpha_i = \gamma_1\gamma_2 y$ und $\alpha_{i-1} = \gamma_1 A y$, dann steht die Alternative der in α_{i-1} expandierten Variable A oben auf dem Keller. Falls $\alpha_i = \gamma_1\gamma_2 y_1 y_2$ und $\alpha_{i-1} = \gamma_1 A y_2$, dann ist ein Teil der Alternative der in α_{i-1} expandierten Variable A noch zu lesende Eingabe. Ziel des Kellerautomaten ist es, dafür zu sorgen, daß die Alternative der in α_{i-1} expandierten Variable A oben auf dem Keller steht, um dann diese durch A zu ersetzen. Hierfür stehen ihm folgende Operationen zur Verfügung:

1. Das nächste Eingabesymbol wird gelesen und auf den Keller geshiftet.
2. Der Kellerautomat erkennt, daß die Alternative einer Variablen A oben auf dem Keller steht und ersetzt diese durch A. D.h., der Kellerautomat führt einen Reduktionsschritt durch.

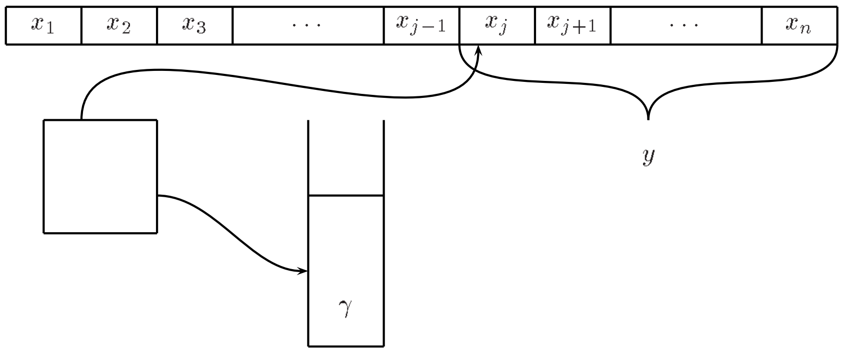

Abbildung 6.5: Konfiguration $\alpha_i = \gamma y$ des Kellerautomaten.

Der Kellerautomat kann in jedem Schritt jede der beiden Operationen ausführen. Also ist der oben beschriebene Kellerautomat nichtdeterministisch. In Anbetracht seiner Operationen heißt er *Shift-Reduce-Parser*.

Übung 6.14*:

a) *Geben Sie eine formale Definition des oben skizzierten nichtdeterministischen Kellerautomaten an.*

b) *Entwickeln Sie einen deterministischen Algorithmus, der alle Kellerinhalte des oben beschriebenen Kellerautomaten gleichzeitig simuliert. Analysieren Sie, in Abhängigkeit von der Länge der Eingabe, die Laufzeit Ihres Algorithmus.*

Unser Ziel ist nun, den Typ der dem Shift-Reduce-Parser zugrundeliegenden kontextfreien Grammatik derart einzuschränken, daß dieser deterministisch wird. Dabei soll, ähnlich zur $LL(k)$-Analyse, die Entscheidung, welcher der nächste durchzuführende Schritt des Shift-Reduce-Parsers ist, eindeutig durch den aktuellen Kellerinhalt und die nächsten k Zeichen der Eingabe festgelegt werden. Wir nehmen nachfolgend stets an, daß bei den betrachteten kontextfreien Grammatiken das Startsysmbol S nicht auf der rechten Seite einer Produktion vorkommt. Da eine beliebige kontextfreie Grammatik mittels Hinzunahme eines neuen Startsymbols S' und der Produktion $S' \to S$ in eine derartige äquivalente kontextfreie Grammatik transformierbar ist, stellt dies keine Einschränkung dar.

Sei $G = (V, \Sigma, P, S)$ eine reduzierte kontextfreie Grammatik und $k \in \mathbb{N}_0$. G heißt *$LR(k)$-Grammatik*, falls die drei Bedingungen

1. $S \underset{\text{rm}}{\overset{*}{\Longrightarrow}} \alpha A w \underset{\text{rm}}{\Longrightarrow} \alpha\beta w$,

2. $S \overset{*}{\underset{\text{rm}}{\Longrightarrow}} \gamma Bx \underset{\text{rm}}{\Longrightarrow} \alpha\beta y$ und
3. $\mathrm{FIRST}_k(w) = \mathrm{FIRST}_k(y)$

$\alpha = \gamma$, $A = B$ und $x = y$ implizieren. G heißt LR-*Grammatik*, falls G für ein $k \in \mathbb{N}_0$ eine $LR(k)$-Grammatik ist. Folgende Intuition führte zur Definition von $LR(k)$-Grammatiken:

Falls $\alpha\beta w$ und $\alpha\beta y$ Rechtssatzformen der $LR(k)$-Grammatik G mit $\mathrm{FIRST}_k(w) = \mathrm{FIRST}_k(y)$ sind und $A \to \beta$ die letzte Produktion ist, die in einer Rechtsableitung von $\alpha\beta w$ aus S verwendet wurde, dann muß auch bei der Rückwärtskonstruktion der Rechtsableitung von $\alpha\beta y$ aus S die Produktion $A \to \beta$ verwendet und somit auch $\alpha\beta y$ zu αAy reduziert werden.

Beispiel 6.4 Sei die kontextfreie Grammatik G_1 durch folgende Produktionen definiert:

$$S \to C|D,\ C \to aC|b \text{ und } D \to aD|c.$$

G_1 ist eine $LR(1)$-Grammatik. Um dies einzusehen, untersuchen wir die Rechtsableitungen in G_1. Jede Rechtsableitung in G_1 hat eine der folgenden beiden Formen:

1. $S \Longrightarrow C \overset{i}{\Longrightarrow} a^iC \Longrightarrow a^ib$ für $i \geq 0$
2. $S \Longrightarrow D \overset{i}{\Longrightarrow} a^iD \Longrightarrow a^ic$ für $i \geq 0$

Betrachte die Ableitungen

$$\begin{array}{llll} S \overset{*}{\underset{\text{rm}}{\Longrightarrow}} & \alpha Aw & \Longrightarrow \alpha\beta w & \text{und} \\ S \overset{*}{\underset{\text{rm}}{\Longrightarrow}} & \gamma Bx & \Longrightarrow \alpha\beta y & \end{array}$$

in G_1. Aus der Definition von G_1 folgt direkt, daß in einer Rechtssatzform rechts von einer Variablen kein Terminalzeichen stehen kann. Also gilt $w = x = \varepsilon$. Sei $\mathrm{FIRST}_1(w) = \mathrm{FIRST}_1(y)$. Dann gilt auch $y = \varepsilon$.

Zu zeigen ist noch $\alpha = \gamma$ und $A = B$. Sei $B \to \delta$ diejenige Produktion, die angewandt wurde, um $\alpha\beta$ aus γB abzuleiten.

Falls $A = C$, dann gilt $\beta = aC$ oder $\beta = b$.

$\beta = aC$ impliziert, daß $B = S$ oder $B = C$, da nur Alternativen von S und C mit C enden. $B = S$ impliziert $\gamma = \varepsilon$ und scheidet wegen $S \not\Longrightarrow aC$ aus. Also gilt $B = C$, $\delta = aC$ und $\gamma = \alpha$.

$\beta = b$ impliziert $B = C$, da nur C eine Alternative, die mit b endet, besitzt. Also gilt $A = B$ und $\gamma = \alpha$.

Der Fall $A = D$ wird analog zum Fall $A = C$ behandelt.

Falls $A = S$, dann gilt $\alpha = \varepsilon$ und $\beta = C$ oder $\beta = D$. Somit gilt auch $\gamma = \varepsilon$ und $B = S$.

Übung 6.15: *Zeigen Sie, daß die kontextfreie Grammatik G_1 nicht LL ist.*

♦

Gegeben eine $LR(k)$-Grammatik $G = (V, \Sigma, P, S)$ möchten wir einen deterministischen Shift-Reduce-Parser für G generieren. Sei für $x \in L(G)$

$$S \underset{\text{rm}}{\Longrightarrow} \alpha_0 \underset{\text{rm}}{\Longrightarrow} \alpha_1 \underset{\text{rm}}{\Longrightarrow} \dots \underset{\text{rm}}{\Longrightarrow} \alpha_{m-1} \underset{\text{rm}}{\Longrightarrow} \alpha_m = x$$

die Rechtsableitung von x aus dem Startsymbol S. Seien y die noch zu lesende Eingabe und $\alpha_i = \gamma y$ die aktuelle Rechtssatzform. Abbildung 6.6 illustriert diese Situation anhand des Ableitungsbaumes.

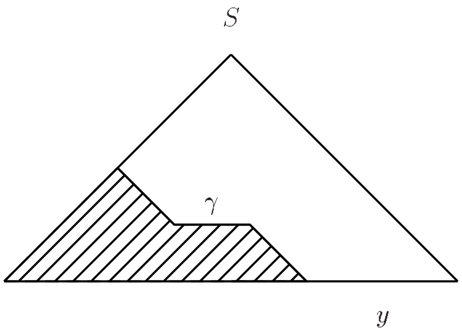

Abbildung 6.6: Lage der aktuellen Rechtssatzform $\alpha_i = \gamma y$ im Ableitungsbaum.

Falls α_i aus α_{i-1} durch Anwendung der Produktion $A \to \beta$ entstanden ist, dann gilt $\beta = \beta_1\beta_2$, wobei β_1 ein Suffix von γ und β_2 ein Präfix von y ist. β_2 ist genau dann ε, wenn die Alternative, die zu A reduziert wird, oben auf dem Keller steht. Dies ist insbesondere auch dann der Fall, wenn $\beta = \varepsilon$. Dann kann der Reduktionsschritt durchgeführt und oben auf dem Keller β_1 durch A ersetzt werden. Nehmen wir an, daß $\beta_2 \neq \varepsilon$, d.h., der Reduktionsschritt kann noch nicht durchgeführt werden.

Falls $\beta_1 \neq \varepsilon$, dann verarbeitet die Reduktion von $\beta_1\beta_2$ zu A einen nichtleeren Suffix des Kellerinhaltes. Es muß lediglich der Präfix β_2 von y auf den Keller geshiftet und dann der Reduktionsschritt durchgeführt werden.

Falls $\beta_1 = \varepsilon$, dann verarbeitet der Reduktionsschritt keinen Suffix des Kellerinhaltes. Wir werden nun untersuchen, wie der nächste Reduktionsschritt, der dies tut, aussieht.

Da wir die Rechtsableitung von x aus S rückwärts konstruieren, muß ein Präfix y_1 von y existieren, der aus einem $\delta \in NV^*$ abgeleitet wurde, so daß ein echter Suffix

γ_2 von γ konkateniert mit δ diejenige Alternative einer Variablen C, zu der sie in der Ableitung expandiert worden ist, ergeben. D.h., $\gamma = \gamma_1\gamma_2$ und $y = y_1y_2$, so daß gilt:

1. $\gamma_2 \neq \varepsilon$.
2. Es existiert eine Produktion $C \to \gamma_2\delta \in P$ für die gilt:
   - $\delta \in NV^*$ und $\delta \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} y_1$.
   - Der Shift-Reduce-Parser muß y_1 zu δ reduzieren. D.h., nach dem Lesen von y_1 steht δ oberhalb γ im Keller.
   - Anschließend wird $\gamma_2\delta$ zu C reduziert.

Abbildung 6.7 illustriert diese Situation.

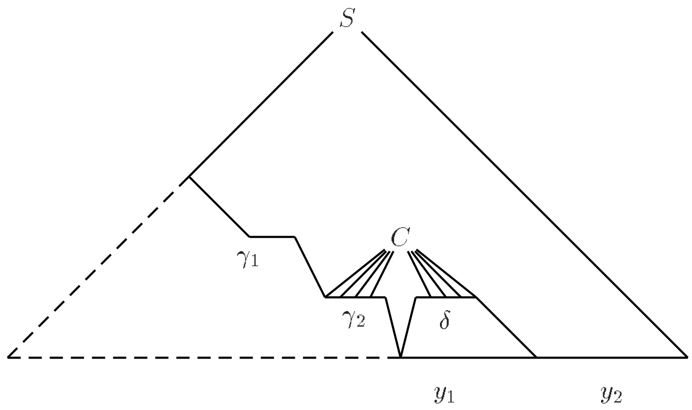

Abbildung 6.7: Lage der Produktion $C \to \gamma_2\delta$ im Ableitungsbaum.

Möglicherweise gilt $y_1 = \varepsilon$. Falls $y_1 \neq \varepsilon$, dann ist der zur Produktion $C \to \gamma_2\delta$ korrespondierende Reduktionsschritt der nächste, der einen nichtleeren Suffix von γ mit einem nichtleeren Präfix von y verbindet.

Nützlich wäre es also, wenn der Shift-Reduce-Parser zu jedem Zeitpunkt alle Produktionen $A \to \beta_1\beta_2$ kennt, für die gilt:

1. β_1 ist Suffix des aktuellen Kellerinhalts γ,
2. möglicherweise wird ein Präfix y_1 von y zu β_2 und dann
3. $\beta_1\beta_2$ mittels Anwendungen der Produktion $A \to \beta_1\beta_2$ zu A reduziert.

Falls $\beta_2 = y_1$, dann bedeutet y_1 zu β_2 zu reduzieren, den Präfix y_1 von y auf den Keller zu shiften.

Die Reduktion von $\beta_1\beta_2$ zu A ist nach der Reduktion von y_1 zu β_2 nur dann möglich, wenn zu diesem Zeitpunkt bezüglich des aktuellen Lookaheads $u = \mathrm{FIRST}_k(y_2)$, wobei $y = y_1y_2$, folgende Eigenschaft erfüllt ist:

Es existiert eine Ableitung

$$S \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} \gamma_1 A w \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} \gamma_1\gamma_2\delta w \quad \text{mit } u = \mathrm{FIRST}_k(w).$$

Also ist es sinnvoll, bezüglich einer solchen Produktion $A \to \beta_1\beta_2$ auch alle möglichen Lookaheads von y_2, die diese Reduktion erlauben, zu kennen. Obige Betrachtungen führen zu folgenden Definitionen:

$[A \to \beta_1.\beta_2, u]$ ist ein $LR(k)$-*Item* für G, falls $A \to \beta_1\beta_2 \in P$ und $u \in \Sigma^{\leq k}$. Ein $LR(k)$-Item $[A \to \beta_1.\beta_2, u]$ ist *gültig* für $\alpha\beta_1 \in V^*$, falls eine Ableitung

$$S \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} \alpha A w \underset{\mathrm{rm}}{\Longrightarrow} \alpha\beta_1\beta_2 w \quad \text{mit } u = \mathrm{FIRST}_k(w)$$

existiert.

Zum Zeitpunkt, wenn $\alpha\beta_1 y$ die aktuelle Rechtssatzform ist; d.h., $\alpha\beta_1$ ist der aktuelle Kellerinhalt und y ist die noch zu lesende Eingabe, dann ist die Produktion $A \to \beta_1\beta_2$ ein möglicher Kandidat für eine spätere Reduktion und u ein möglicher Lookahead zum Zeitpunkt der Reduktion.

Beispiel 6.4 (Fortführung) Betrachte wieder die $LR(k)$-Grammatik G_1. Das $LR(k)$-Item $[C \to a.C, \varepsilon]$ ist wegen $S \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} aaC \underset{\mathrm{rm}}{\Longrightarrow} aaaC$ gültig für aaa.

♦

Als nächstes werden wir zeigen, daß die Menge der für den aktuellen Kellerinhalt gültigen $LR(k)$-Items stets die notwendige Information enthält, um den nächsten Schritt des Shift-Reduce-Parsers eindeutig festzulegen. Für den Beweis des zentralen Satzes hierzu benötigen wir folgendes Lemma, das Nicht-$LR(k)$-Grammatiken etwas genauer charakterisiert als die Definition der $LR(k)$-Grammatik.

Lemma 6.7 *Sei $k \in \mathbb{N}_0$ und $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik, die keine $LR(k)$-Grammatik ist. Dann existieren Ableitungen*

$$\begin{array}{lllll} S \overset{*}{\underset{rm}{\Longrightarrow}} \alpha A w & \underset{rm}{\Longrightarrow} & \alpha\beta w & & \textit{und} \\ S \overset{*}{\underset{rm}{\Longrightarrow}} \gamma B x & \underset{rm}{\Longrightarrow} & \gamma\delta x = \alpha\beta y & & \end{array}$$

mit $\mathrm{FIRST}_k(w) = \mathrm{FIRST}_k(y)$ *und* $|\gamma\delta| \geq |\alpha\beta|$, *aber* $\alpha A y \neq \gamma B x$.

Beweis: Aus der Definition von $LR(k)$-Grammatiken folgt, daß Ableitungen

$$\begin{array}{lll} S \underset{\mathrm{rm}}{\overset{*}{\Longrightarrow}} \alpha A w \underset{\mathrm{rm}}{\Longrightarrow} \alpha\beta w & & \text{und} \\ S \underset{\mathrm{rm}}{\overset{*}{\Longrightarrow}} \gamma B x \underset{\mathrm{rm}}{\Longrightarrow} \gamma\delta x = \alpha\beta y & & \end{array}$$

mit $\mathrm{FIRST}_k(w) = \mathrm{FIRST}_k(y)$ und $\gamma Bx \neq \alpha Ay$ existieren. Falls $|\gamma\delta| \geq |\alpha\beta|$, dann ist nichts mehr zu beweisen. Nehmen wir also an, daß $|\gamma\delta| < |\alpha\beta|$. Wir werden nun ein anderes Gegenbeispiel zur $LR(k)$-Eigenschaft, in dem $\gamma\delta$ die Rolle von $\alpha\beta$ spielt, konstruieren.

Wegen $\gamma\delta x = \alpha\beta y$ und $|\gamma\delta| < |\alpha\beta|$ existiert ein $z \in \Sigma^+$ mit $\gamma\delta z = \alpha\beta$. Also existieren die Ableitungen

$$\begin{array}{lll} S \underset{\mathrm{rm}}{\overset{*}{\Longrightarrow}} \gamma B x \underset{\mathrm{rm}}{\Longrightarrow} \gamma\delta x & & \text{und} \\ S \underset{\mathrm{rm}}{\overset{*}{\Longrightarrow}} \alpha A w \underset{\mathrm{rm}}{\Longrightarrow} \alpha\beta w = \gamma\delta z w. & & \end{array}$$

Da z derart definiert ist, daß $x = zy$, folgt aus $\mathrm{FIRST}_k(w) = \mathrm{FIRST}_k(y)$ auch $\mathrm{FIRST}_k(x) = \mathrm{FIRST}_k(zw)$.

Zu zeigen ist noch, daß obige Ableitungen die $LR(k)$-Eigenschaft verletzen. Nehmen wir hierzu an, daß dies nicht der Fall ist. Dann gilt $\alpha Aw = \gamma Bzw$ und somit auch $\gamma Bz = \alpha A$. Dies ist ein Widerspruch zu $z \in \Sigma^+$.

Da wir im Grunde genommen nur die Rolle der beiden Ausgangsableitungen vertauscht haben, folgt nach einer Umbenennung obiger Ableitungen direkt, daß diese die Behauptung des Lemmas erfüllen.

■

Sei γy die aktuelle Rechtssatzform. D.h., γ ist der aktuelle Kellerinhalt und y die noch zu lesende Eingabe. Sei $u = FIRST_k(y)$ der aktuelle Lookahead. Ferner bezeichne $\mathfrak{a}(\gamma)$ die *Menge der für γ gültigen $LR(k)$-Items*. Betrachte $[A \to \beta_1.\beta_2, v] \in \mathfrak{a}(\gamma)$. Dies bedeutet insbesondere, daß β_1 Suffix von γ ist. Falls $\beta_2 = \varepsilon$, dann gilt $v = u$ und das $LR(k)$-Item $[A \to \beta_1., u]$ korrespondiert zu einem durchführbaren Reduktionsschritt, bei dem die Produktion $A \to \beta_1$ angewandt wird. Falls $\beta_2 \in \Sigma V^*$ und $u \in FIRST_k(\beta_2 v)$, dann korrespondiert das Item $[A \to \beta_1.\beta_2, v]$ zu einem durchführbaren Leseschritt. Der folgende Satz besagt, daß $LR(k)$-Items in $\mathfrak{a}(\gamma)$ zu höchstens einem durchführbaren Schritt korrespondieren.

Satz 6.4 *Seien $k \in \mathbb{N}_0$ und $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik. G ist genau dann eine $LR(k)$-Grammatik, wenn für alle $u \in \Sigma^{\leq k}$ und alle $\alpha\beta \in V^*$ folgende Eigenschaft erfüllt ist:*

Falls ein $LR(k)$-Item $[A \to \beta., u]$ für $\alpha\beta$ gültig ist, dann existiert kein anderes für $\alpha\beta$ gültiges $LR(k)$-Item $[C \to \beta_1.\beta_2, v]$ mit $\beta_2 \in \Sigma V^* \cup \{\varepsilon\}$ und $u \in FIRST_k(\beta_2 v)$.

Beweis:
„$\Longrightarrow$“
Seien $[A \to \beta., u]$ und $[C \to \beta_1.\beta_2, v]$ mit $\beta_2 \in \Sigma V^* \cup \{\varepsilon\}$ und $u \in \mathrm{FIRST}_k(\beta_2 v)$ zwei verschiedene für $\alpha\beta$ gültige $LR(k)$-Items. Dann gibt es Ableitungen

$$\begin{array}{lllll} S \underset{\mathrm{rm}}{\overset{*}{\Longrightarrow}} & \alpha A w & \underset{\mathrm{rm}}{\Longrightarrow} & \alpha\beta w & \text{mit } \mathrm{FIRST}_k(w) = u \\ S \underset{\mathrm{rm}}{\overset{*}{\Longrightarrow}} & \alpha_1 C x & \underset{\mathrm{rm}}{\Longrightarrow} & \alpha_1\beta_1\beta_2 x & \text{mit } \mathrm{FIRST}_k(x) = v. \end{array}$$

Wir zeigen nun, daß G keine $LR(k)$-Grammatik ist. In Abhängigkeit von β_2 unterscheiden wir drei Fälle:

Fall 1: $\beta_2 = \varepsilon$.

Dann gilt $u = v$. Somit sehen die beiden Ableitungen folgendermaßen aus:

$$\begin{array}{lllll} S \underset{\mathrm{rm}}{\overset{*}{\Longrightarrow}} & \alpha A w & \underset{\mathrm{rm}}{\Longrightarrow} & \alpha\beta w & \text{und} \\ S \underset{\mathrm{rm}}{\overset{*}{\Longrightarrow}} & \alpha_1 C x & \underset{\mathrm{rm}}{\Longrightarrow} & \alpha_1\beta_1 x = \alpha\beta x, & \end{array}$$

wobei $\mathrm{FIRST}_k(w) = \mathrm{FIRST}_k(x) = u$.

Da beide $LR(k)$-Items gemäß unserer Annahme verschieden sind, gilt $A \neq C$ oder $\beta \neq \beta_1$. In beiden Fällen ist dann die $LR(k)$-Eigenschaft verletzt.

Fall 2: $\beta_2 = z$ für ein $z \in \Sigma^+$.

Dann sehen die beiden Ableitungen folgendermaßen aus:

$$\begin{array}{lllll} S \underset{\mathrm{rm}}{\overset{*}{\Longrightarrow}} & \alpha A w & \underset{\mathrm{rm}}{\Longrightarrow} & \alpha\beta w & \text{und} \\ S \underset{\mathrm{rm}}{\overset{*}{\Longrightarrow}} & \alpha_1 C x & \underset{\mathrm{rm}}{\Longrightarrow} & \alpha_1\beta_1 z x, & \end{array}$$

wobei $\alpha\beta = \alpha_1\beta_1$ und $\mathrm{FIRST}_k(w) = \mathrm{FIRST}_k(zx) = u$. Dann ist G keine $LR(k)$-Grammatik, da ansonsten $x = zx$ erfüllt sein müßte, was wegen $z \in \Sigma^+$ unmöglich ist.

Fall 3: $\beta_2 = a\beta_2'$ für ein $a \in \Sigma$ und $\beta_2' \in V^*NV^*$.

Dann gilt $\beta_2 = a\beta_2' \underset{\mathrm{rm}}{\overset{*}{\Longrightarrow}} au_1Bu_3 \underset{\mathrm{rm}}{\Longrightarrow} au_1u_2u_3$ mit $\mathrm{FIRST}_k(au_1u_2u_3x) = u$.

Somit sehen die beiden Ableitungen folgendermaßen aus:

$$\begin{array}{lllllll} S \underset{\mathrm{rm}}{\overset{*}{\Longrightarrow}} & \alpha A w & \underset{\mathrm{rm}}{\Longrightarrow} & \alpha\beta w & & \\ S \underset{\mathrm{rm}}{\overset{*}{\Longrightarrow}} & \alpha_1 C x & \underset{\mathrm{rm}}{\Longrightarrow} & \alpha_1\beta_1 a\beta_2' x & \underset{\mathrm{rm}}{\overset{*}{\Longrightarrow}} & \alpha_1\beta_1 a u_1 B u_3 x \\ & & \underset{\mathrm{rm}}{\Longrightarrow} & \alpha_1\beta_1 a u_1 u_2 u_3 x, & & \end{array}$$

wobei $\alpha_1\beta_1 = \alpha\beta$ und $au_1u_2u_3x = uz$ für ein $z \in \Sigma^*$. Falls G eine $LR(k)$-Grammatik ist, dann gilt mit $\gamma = \alpha_1\beta_1 au_1$ und $y = u_3x$:

$\alpha Aau_1u_2u_3x = \alpha_1\beta_1 au_1 Bu_3x$ und somit $\alpha Aau_1u_2 = \alpha_1\beta_1 au_1 B$. Durch Substitution von $\alpha\beta$ für $\alpha_1\beta_1$ erhalten wir $Aau_1u_2 = \beta au_1 B$. Wegen $a \in \Sigma$ und $u_1u_2 \in \Sigma^*$ endet Aau_1u_2 mit einem Symbol in Σ. Also gilt wegen $B \in N$ auch $Aau_1u_2 \neq \beta au_1 B$. Also ist G keine $LR(k)$-Grammatik.

„$\Longleftarrow$"
Nehmen wir an, daß G keine $LR(k)$-Grammatik ist. Dann impliziert Lemma 6.7, daß zwei Ableitungen

$$\begin{array}{lllll} S & \underset{\text{rm}}{\overset{*}{\Longrightarrow}} \alpha Aw & \underset{\text{rm}}{\Longrightarrow} \alpha\beta w & & \text{und} \\ S & \underset{\text{rm}}{\overset{*}{\Longrightarrow}} \gamma Cx & \underset{\text{rm}}{\Longrightarrow} \gamma\delta x = \alpha\beta y & & \end{array}$$

mit $\mathrm{FIRST}_k(w) = \mathrm{FIRST}_k(y) = u$, $|\gamma\delta| \geq |\alpha\beta|$ und $\alpha Ay \neq \gamma Cx$ existieren.

Dann ist das $LR(k)$-Item $[A \to \beta.u]$ gültig für $\alpha\beta$. Zu konstruieren ist noch ein weiteres für $\alpha\beta$ gültiges $LR(k)$-Item $[C \to \beta_1.\beta_2, v]$ mit $\beta_2 \in \Sigma V^* \cup \{\varepsilon\}$ und $u \in FIRST_k(\beta_2 v)$.

Wegen $|\gamma\delta| \geq |\alpha\beta|$ und $\gamma\delta x = \alpha\beta y$ gilt $\gamma\delta = \alpha\beta z$ für ein $z \in \Sigma^*$. Je nachdem, ob z ein Suffix von δ oder δ ein Suffix von z ist, unterscheiden wir zwei Fälle:

1. $\delta = \delta' z$ für ein $\delta' \in V^*$.

   Dann impliziert $y = zx$, daß das $LR(k)$-Item $[C \to \delta'.z, v]$ gültig für $\alpha\beta$ ist. Nehmen wir an, daß $[C \to \delta'.z, v] = [A \to \beta., u]$. Dann gilt $A = C$, $\beta = \delta'$, $z = \varepsilon$ und $u = v$. Dann gilt aber auch $\delta = \beta$ und somit $\alpha = \gamma$ und $x = y$, was wegen $\alpha Ay \neq \gamma Cx$ nicht möglich ist. Also gilt $[C \to \delta'.z, v] \neq [A \to \beta., u]$.

2. $z = z'\delta$ für ein $z' \in \Sigma^+$.

   Betrachte die Rechtsableitung $S \underset{\text{rm}}{\overset{*}{\Longrightarrow}} \alpha\beta z' Cx$. Sei αBy_1 die letzte Rechtssatzform in ihr mit $y_1 \in \Sigma^*$, $B \in N$ und $|\alpha_1 B| \leq |\alpha\beta| + 1$. Wegen $|S| = 1$ existiert diese Rechtssatzform. Wir können nun die Ableitung von $\alpha\beta y$ aus S auf folgende Art und Weise schreiben:

   $$S \underset{\text{rm}}{\overset{*}{\Longrightarrow}} \alpha_1 By_1 \underset{\text{rm}}{\Longrightarrow} \alpha_1\beta_1\beta_2 y_1 \underset{\text{rm}}{\overset{*}{\Longrightarrow}} \alpha_1\beta_1 y,$$

   wobei $\alpha_1\beta_1 = \alpha\beta$. Die Wahl der Rechtssatzform $\alpha_1 By_1$ impliziert, daß $|\alpha_1| \leq |\alpha\beta|$ und daß $\beta_2 \in \Sigma V^*$. Demnach ist das $LR(k)$-Item $[B \to \beta_1.\beta_2, v]$ gültig für $\alpha\beta$. Wegen $FIRST_k(y) = u$ gilt $u \in FIRST_k(\beta_2 v)$. $\beta_2 \in \Sigma V^*$ impliziert, daß $[A \to \beta.]$ und $[B \to \beta_1.\beta_2]$ verschieden sind.

■

Falls zu einem Zeitpunkt während der Konstruktion der Rechtsableitung ein $LR(k)$-Item $[A \to \beta., u]$ für den aktuellen Kellerinhalt γ gültig ist und u mit dem aktuellen Lookahead übereinstimmt, dann steht β oben auf dem Keller und der zur Produktion $A \to \beta$ korrespondierende Reduktionsschritt ist der einzige durchführbare Schritt. Falls ein $LR(k)$-Item $[C \to \beta_1.\beta_2, v]$ mit $\beta_2 \in \Sigma V^*$ gültig und der aktuelle Lookahead u in $\mathrm{FIRST}_k(\beta_2 v)$ enthalten ist, dann ist der zu $FIRST_1(u)$ korrespondierende Leseschritt der einzige durchführbare Schritt; d.h., das nächste Eingabesymbol kann auf den Keller geshiftet werden.

Seien $\mathfrak{a}$ die aktuelle Menge der gültigen $LR(k)$-Items und u der aktuelle Lookahead. Der nächste Schritt des Shift-Reduce-Parsers ist dann:

$$\begin{cases} \text{Reduziere bzgl. der Produktion } A \to \beta & \text{falls } [A \to \beta., u] \in \mathfrak{a} \\ \text{Führe einen Shift durch} & \text{falls } [C \to \beta_1.\beta_2, v] \in \mathfrak{a} \\ & \text{ mit } \beta_2 \in \Sigma V^* \text{ und} \\ & u \in FIRST_k(\beta_2 v) \\ \text{Akzeptiere} & \text{falls } [S \to \alpha.\varepsilon] \in \mathfrak{a} \\ & \text{ und } u = \varepsilon \\ \text{Fehlermeldung} & \text{sonst.} \end{cases}$$

Also ist es nützlich, wenn der Parser zu jedem Zeitpunkt die Menge der für den aktuellen Kellerinhalt gültigen $LR(k)$-Items kennt. Bei der Vorausberechnung der möglichen Mengen von gültigen $LR(k)$-Items tritt für nichtendliche Sprachen das Problem auf, daß die Anzahl der Rechtsableitungen, die die Gültigkeit von $LR(k)$-Items implizieren, unendlich ist. Dieses Problem wird durch die Beobachtung, daß die Anzahl der verschiedenen Mengen von $LR(k)$-Items endlich ist, entschärft.

Sei

$$\mathfrak{S} = \{\mathfrak{a} \mid \mathfrak{a} = \mathfrak{a}(\gamma) \text{ für ein } \gamma \in V^*\}.$$

Unser Ziel ist nun die Berechnung von $\mathfrak{S}$. Hierzu entwickeln wir zunächst einen Algorithmus, der für ein gegebenes $\gamma = X_1X_2\ldots X_t$ die Menge $\mathfrak{a}(\gamma)$ berechnet. Wir konstruieren $\mathfrak{a}(\gamma)$, indem wir nacheinander $\mathfrak{a}(\varepsilon), \mathfrak{a}(X_1), \mathfrak{a}(X_1X_2), \ldots, \mathfrak{a}(X_1X_2\ldots X_t)$ berechnen.

Algorithmus: BERECHNUNG VON $\mathfrak{a}(\gamma)$

Eingabe: kontextfreie Grammatik $G = (V, \Sigma, P, S)$, $k \in \mathbb{N}_0$ und $\gamma = X_1X_2\ldots X_t$.

Ausgabe: $\mathfrak{a}(\gamma)$.

Methode:

(1) Konstruktion von $\mathfrak{a}(\varepsilon)$:
   (a) $\mathfrak{a}(\varepsilon) := \emptyset$;
   (b) **for** alle $S \to \alpha \in P$
      do
         $\mathfrak{a}(\varepsilon) := \mathfrak{a}(\varepsilon) \cup \{[S \to .\alpha, \varepsilon]\}$
      od;
   (c) **if** $[A \to .B\alpha, u] \in \mathfrak{a}(\varepsilon)$ **and** $B \to \beta \in P$
      then
         $\mathfrak{a}(\varepsilon) := \mathfrak{a}(\varepsilon) \cup \{[B \to .\beta, x] \mid x \in FIRST_k(\alpha u)\}$
      fi;
   (d) Wiederhole Schritt (c) solange, bis kein neues Item mehr zu $\mathfrak{a}(\varepsilon)$ hinzugefügt wird.

(2) Konstruktion von $\mathfrak{a}(X_1X_2 \dots X_t)$, $t \geq 1$:
   $i := 0$;
   while $i < t$
    do

   (a) $\mathfrak{a}(X_1X_2 \dots X_{i+1}) := \emptyset$;
   (b) **for** alle $[A \to \alpha.X_{i+1}\beta, u] \in \mathfrak{a}(X_1X_2 \dots X_i)$
         do
            $\mathfrak{a}(X_1X_2 \dots X_{i+1}) := \mathfrak{a}(X_1X_2 \dots X_{i+1}) \cup \{[A \to \alpha X_{i+1}.\beta, u]\}$
         od;
   (c) **if** $[A \to \alpha.B\beta, u]$ wurde zu $\mathfrak{a}(X_1 \dots X_{i+1})$ hinzugefügt
      and $B \to \delta \in P$
         then
            $\mathfrak{a}(X_1X_2 \dots X_{i+1}) := \mathfrak{a}(X_1X_2 \dots X_{i+1}) \cup \{[B \to .\delta, x] \mid x \in \mathrm{FIRST}_k(\beta u)\}$
      fi;
   (d) Wiederhole Schritt (c) solange, bis kein neues Item mehr zu $\mathfrak{a}(X_1X_2 \dots X_{i+1})$ hinzugefügt wird.
   (e) $i := i + 1$.
    od.

Sei $\mathfrak{a} = \mathfrak{a}(\gamma)$ für ein $\gamma \in V^*$. Für $X \in V$ definieren wir

$$GOTO(\mathfrak{a}, X) = \mathfrak{a}(\gamma X).$$

Da Schritt 2 des obigen Algorithmus unabhängig von $X_1X_2 \dots X_i$ ist, ist obige Definition konsistent. Die Berechnung von $\mathfrak{a}(X_1X_2 \dots X_{i+1})$ hängt wegen

$\mathfrak{a}(X_1X_2\ldots X_{i+1}) = GOTO(\mathfrak{a}(X_1X_2\ldots X_i), X_{i+1})$ nur von X_{i+1} und der Menge $\mathfrak{a}(X_1X_2\ldots X_i)$ ab.

Bevor wir die Korrektheit des obigen Algorithmus beweisen, führen wir diesen anhand eines Beispieles durch.

Beispiel 6.5 Sei G_2 die kontextfreie Grammatik, die durch folgende Produktionen definiert ist:

$$
\begin{aligned}
&S \to A\\
&A \to aAaAb|\varepsilon.
\end{aligned}
$$

Sei $k = 1$. Unser Ziel ist nun die Berechnung von $\mathfrak{a}(\varepsilon)$, $\mathfrak{a}(A)$ und $\mathfrak{a}(Aa)$.

Berechnung von $\mathfrak{a}(\varepsilon)$: (1b) $\mathfrak{a}(\varepsilon) := \{[S \to .A, \varepsilon]\}$.

(1c) $\cup\{[A \to .AaAb, \varepsilon], [A \to \varepsilon., \varepsilon]\}$.

(1d) Wegen $FIRST_1(aAb) = \{a\}$ ergibt sich nachfolgende Wiederholung von Schritt (1c):
$\cup\{[A \to .AaAb, a], [A \to \varepsilon., a]\}$.

Also erhalten wir $\mathfrak{a}(\varepsilon) = \{[S \to .A, \varepsilon], [A \to .AaAb, \varepsilon], [A \to \varepsilon., \varepsilon], [A \to .AaAb, a], [A \to \varepsilon., a]\}$.

Berechnung von $\mathfrak{a}(A)$: Hierzu berechnen wir $GOTO(\mathfrak{a}(\varepsilon), A)$.

(2b) $\mathfrak{a}(A) := \{[S \to A., \varepsilon], [A \to A.aAb, \varepsilon], [A \to A.aAb, a]\}$.

Die Durchführung von (2c) erweitert $\mathfrak{a}(A)$ nicht, so daß Schritt (2b) bereits $\mathfrak{a}(A)$ berechnet.

Berechnung von $\mathfrak{a}(Aa)$: Hierzu berechnen wir $GOTO(\mathfrak{a}(A), a)$ und erhalten:

$$
\begin{aligned}
\mathfrak{a}(Aa) = \{\ &[A \to Aa.Ab, \varepsilon], [A \to Aa.Ab, a], [A \to .AaAb, b], [A \to \varepsilon., b],\\
&[A \to .AaAb, a], [A \to \varepsilon., a]\}.
\end{aligned}
$$

♦

Die Korrektheit des Algorithmus BERECHNUNG VON $\mathfrak{a}(\gamma)$ ergibt sich aus folgendem Satz:

Satz 6.5 *Ein $LR(k)$-Item ist nach der Durchführung von Schritt (2) im obigen Algorithmus genau dann in $\mathfrak{a}(\gamma)$, wenn dieses für γ gültig ist.*

Beweis: Es ist leicht zu sehen, daß obiger Algorithmus terminiert und auch $\mathfrak{a}(\varepsilon)$ korrekt berechnet wird.

Übung 6.16: *Zeigen Sie, daß der Algorithmus* BERECHNUNG VON $\mathfrak{a}(\gamma)$ $\mathfrak{a}(\varepsilon)$ *korrekt berechnet.*

Nehmen wir an, daß $\mathfrak{a}(X_1X_2\ldots X_i)$ genau die für $X_1X_2\ldots X_i$ gültigen $LR(k)$-Items enthält. Zu zeigen ist, daß nach der Durchführung von Schritt (2) genau die für $X_1X_2\ldots X_{i+1}$ gültigen $LR(k)$-Items in $\mathfrak{a}(X_1X_2\ldots X_{i+1})$ sind.

„$\Longrightarrow$ "

Nehmen wir an, daß $[A \to \beta_1.\beta_2, u]$ gültig für $X_1X_2\ldots X_{i+1}$ ist. Dann existiert eine Ableitung

$$S \underset{\text{rm}}{\overset{*}{\Longrightarrow}} \alpha A w \underset{\text{rm}}{\Longrightarrow} \alpha_1\beta_1\beta_2 w$$

mit $\alpha_1\beta_1 = X_1X_2\ldots X_{i+1}$ und $\mathrm{FIRST}_k(w) = u$. Wir unterscheiden zwei Fälle:

Falls $\beta_1 = \beta_1{}'X_{i+1}$, dann ist $[A \to \beta_1{}'.X_{i+1}\beta_2, u]$ gültig für $X_1X_2\ldots X_i$ und somit gemäß unserer Annahme in $\mathfrak{a}(X_1X_2\ldots X_i)$. Also fügt Schritt (2b) das $LR(k)$-Item $[A \to \beta_1.\beta_2, u]$ in $\mathfrak{a}(X_1X_2\ldots X_{i+1})$ ein.

Falls $\beta_1 = \varepsilon$, dann gilt $\alpha = X_1X_2\ldots X_{i+1}$. Da die Ableitung $S \underset{\text{rm}}{\overset{*}{\Longrightarrow}} \alpha A w$ eine Rechtsableitung ist, gilt

$$S \underset{\text{rm}}{\overset{*}{\Longrightarrow}} \alpha' B y \underset{\text{rm}}{\Longrightarrow} \alpha'\gamma X_{i+1}\delta y \underset{\text{rm}}{\overset{*}{\Longrightarrow}} \alpha A w,$$

wobei $\alpha'\gamma = X_1X_2\ldots X_i$ und in jedem Schritt der Ableitung $\alpha'\gamma X_{i+1}\delta y \underset{\text{rm}}{\overset{*}{\Longrightarrow}} \alpha A w$ eine Variable rechts von X_{i+1} ersetzt wird.

Also ist das $LR(k)$-Item $[B \to \gamma.X_{i+1}\delta, v]$ mit $v = \mathrm{FIRST}_k(y)$ für $X_1X_2\ldots X_i$ gültig und somit gemäß unserer Annahme in $\mathfrak{a}(X_1X_2\ldots X_i)$. Folglich fügt Schritt (2b) das Item $[B \to \gamma X_{i+1}.\delta, v]$ in $\mathfrak{a}(X_1X_2\ldots X_{i+1})$ ein.

Wegen $\delta y \underset{\text{rm}}{\overset{*}{\Longrightarrow}} Aw$ existiert eine Folge $D_1, D_2, \ldots, D_m$ von Variablen und eine Folge $\Theta_1, \Theta_2, \ldots, \Theta_m \in V^*$ von Strings, so daß

1. δ mit D_1 beginnt,
2. $A = D_m$ und
3. $D_i \to D_{i+1}\Theta_{i+1} \in P$ für $1 \leq i < m$.

Durch wiederholte Anwendung von Schritt (2c) wird somit das $LR(k)$-Item $[A \to .\beta_2, u]$ in $\mathfrak{a}(X_1X_2\ldots X_{i+1})$ eingefügt.

„$\Longleftarrow$ "

Nehmen wir an, daß das $LR(k)$-Item $[A \to \beta_1.\beta_2, u]$ in $\mathfrak{a}(X_1X_2\ldots X_{i+1})$ eingefügt wird.

Falls $[A \to \beta_1.\beta_2, u]$ während Schritt (2b) in $\mathfrak{a}(X_1X_2\ldots X_{i+1})$ eingefügt wird, dann gilt $\beta_1 = \beta_1{}'X_{i+1}$ und $[A \to \beta_1{}'.X_{i+1}\beta_2, u] \in \mathfrak{a}(X_1X_2\ldots X_i)$. Also existiert eine Ableitung

$$S \overset{*}{\underset{\text{rm}}{\Longrightarrow}} \alpha A w \overset{*}{\underset{\text{rm}}{\Longrightarrow}} \alpha\beta_1{}'X_{i+1}\beta_2 w,$$

wobei $\alpha\beta_1{}' = X_1X_2\ldots X_i$ und $\text{FIRST}_k(w) = u$. Also ist das $LR(k)$-Item $[A \to \beta_1.\beta_2, u]$ gültig für $X_1X_2\ldots X_{i+1}$.

Sei $[A \to \beta_1.\beta_2, u]$ das erste für $X_1X_2\ldots X_{i+1}$ nicht gültige $LR(k)$-Item, das während Schritt (2c) in $\mathfrak{a}(X_1X_2\ldots X_{i+1})$ eingefügt wird.

Dann gilt $\beta_1 = \varepsilon$ und es existiert ein $LR(k)$-Item $[B \to \gamma.A\delta, v]$ mit $u \in \text{FIRST}_k(\delta v)$, das vorher zu $\mathfrak{a}(X_1X_2\ldots X_{i+1})$ hinzugefügt wurde. Unsere Annahme impliziert, daß $[B \to \gamma.A\delta, v]$ für $X_1X_2\ldots X_{i+1}$ gültig ist. Also existiert eine Ableitung

$$S \overset{*}{\underset{\text{rm}}{\Longrightarrow}} \alpha' B y \underset{\text{rm}}{\Longrightarrow} \alpha'\gamma A\delta y$$

mit $v = \text{FIRST}_k(y)$ und $\alpha'\gamma = X_1X_2\ldots X_{i+1}$. Also erhalten wir

$$S \overset{*}{\underset{\text{rm}}{\Longrightarrow}} X_1X_2\ldots X_{i+1}A\delta y \overset{*}{\underset{\text{rm}}{\Longrightarrow}} X_1X_2\ldots X_{i+1}Az \underset{\text{rm}}{\Longrightarrow} X_1X_2\ldots X_{i+1}\beta_2 z,$$

wobei $u = \text{FIRST}_k(z)$. Also ist das $LR(k)$-Item $[A \to .\beta_2, u]$ gültig für $X_1X_2\ldots X_{i+1}$, was ein Widerspruch zu unserer Annahme ist.

■

Die Berechnung der Menge $\mathfrak{S}$ ist nun einfach.

Algorithmus BERECHNUNG VON $\mathfrak{S}$

Eingabe: kontextfreie Grammatik $G = (V, \Sigma, P, S)$ und $k \in \mathbb{N}_0$.

Ausgabe: $\mathfrak{S} = \{\mathfrak{a} \mid \mathfrak{a} = \mathfrak{a}(\gamma)$ für ein $\gamma \in V^*\}$.

Methode: (1) $\mathfrak{S} := \emptyset$; $H := \emptyset$;

(2) Berechne $\mathfrak{a}(\varepsilon)$;
$\mathfrak{S} := \mathfrak{S} \cup \{\mathfrak{a}(\varepsilon)\}$;
$H := H \cup \{\mathfrak{a}(\varepsilon)\}$;

(3) **while** $H \neq \emptyset$
do
wähle $\mathfrak{a} \in H$ beliebig;
$H := H \setminus \{\mathfrak{a}\}$;
for alle $X \in V$
do
berechne $\text{goto}(\mathfrak{a}, X)$;
if $\mathfrak{a}' = \text{goto}(\mathfrak{a}, X) \neq \emptyset$ **and** $\mathfrak{a}' \notin \mathfrak{S}$

then
$\mathfrak{S} := \mathfrak{S} \cup \{\mathfrak{a}'\}$;
$H := H \cup \{\mathfrak{a}'\}$
fi
od
od.

Die von obigem Algorithmus berechnete Menge $\mathfrak{S}$ heißt *kanonische Kollektion* der Menge der $LR(k)$-Items.

Übung 6.17: *Beweisen Sie die Korrektheit des Algorithmus* BERECHNUNG VON $\mathfrak{S}$.

Mit Hilfe der kanonischen Kollektion $\mathfrak{S}$ von $LR(k)$-Items können wir nun testen, ob die zugrundeliegende kontextfreie Grammatik $G = (V, \Sigma, P, S)$ eine $LR(k)$-Grammatik ist.

$\mathfrak{a} \in \mathfrak{S}$ heißt *konsistent*, falls $\mathfrak{a}$ keine zwei Items $[A \to \beta., u]$ und $[C \to \beta_1.\beta_2, v]$ mit $u \in \mathrm{FIRST}_k(\beta_2 v)$ enthält.

Satz 6.5 impliziert, daß G genau dann eine $LR(k)$-Grammatik ist, wenn jedes $\mathfrak{a} \in \mathfrak{S}$ konsistent ist. Also muß jedes $\mathfrak{a} \in \mathfrak{S}$ auf Konsistenz überprüft werden, um zu entscheiden, ob G eine $LR(k)$-Grammatik ist.

Sei $\mathfrak{a} \in \mathfrak{S}$. $T(\mathfrak{a}) = \langle \mathrm{action}_{\mathfrak{a}}, \mathrm{goto}_{\mathfrak{a}} \rangle$ heißt *die zu* $\mathfrak{a}$ *assozierte* $LR(k)$*-Tafel*, wobei $\mathrm{action}_{\mathfrak{a}}$ bzw. $\mathrm{goto}_{\mathfrak{a}}$ folgende zwei Funktionen sind:

$$\begin{aligned}\mathrm{action}_{\mathfrak{a}} : \Sigma^{\leq k} \to\ & \{\mathrm{error}, \mathrm{shift}, \mathrm{accept}\} \\ & \cup \{\mathrm{reduce}\ i \mid i \text{ ist Nummer einer Produktion in } P\}\end{aligned}$$

und

$$\mathrm{goto}_{\mathfrak{a}} : V \to \mathfrak{S} \cup \{\mathrm{error}\}.$$

Dabei gilt:

$$\mathrm{action}_{\mathfrak{a}}(u) = \begin{cases} \mathrm{shift} & \text{falls } [A \to \beta_1.\beta_2, v] \in \mathfrak{a} \text{ mit } \beta_2 \in \Sigma V^* \text{ und} \\ & u \in \mathrm{FIRST}_k(\beta_2 v) \\ \mathrm{reduce}\ l & \text{falls } [A \to \beta., u] \in \mathfrak{a} \\ & \text{und } a \to \beta \text{ die } l\text{-te Produktion ist} \\ \mathrm{accept} & \text{falls } [S \to \alpha.\varepsilon] \in \mathfrak{a} \text{ und } u = \varepsilon \\ \mathrm{error} & \text{sonst} \end{cases}$$

$$\mathrm{goto}_{\mathfrak{a}}(X) = \begin{cases} \mathfrak{a}' & \text{falls } \mathrm{goto}(\mathfrak{a}, X) = \mathfrak{a}' \\ \mathrm{error} & \text{falls } \mathrm{goto}(\mathfrak{a}, X) = \emptyset. \end{cases}$$

Alle $LR(k)$-Items $[C \to \beta_1.\beta_2, v]$, die zur Definition von $\mathrm{action}_{\mathfrak{a}}(u)$ benötigt werden, erfüllen $\beta_2 \in \Sigma V^* \cup \{\varepsilon\}$. Jedoch können $LR(k)$-Items $[C \to \beta_1.\beta_2, v]$ mit $\beta_2 \notin \Sigma V^* \cup \{\varepsilon\}$ zur Unterscheidung zweier Mengen $\mathfrak{a}$ und $\mathfrak{a}'$ notwendig sein.

Sei $\gamma \in V^*$. Falls $\mathfrak{a} = \mathfrak{a}(\gamma)$, dann sagen wir, daß $T(\mathfrak{a})$ *zu* γ *assoziiert.*

Sei $\mathfrak{T}$ die Menge der $LR(k)$-Tafeln, die zur kanonischen Kollektion der Menge von $LR(k)$-Items assoziiert ist. Das Paar $(\mathfrak{T}, T(\mathfrak{a}(\varepsilon)))$ heißt *kanonische Menge von* $LR(k)$*-Tafeln* der $LR(k)$-Grammatik G.

Der kanonische $LR(k)$-Parser ist ein Shift-Reduce-Parser. Er besitzt ein Eingabeband, ein Ausgabeband und einen Keller. Mit Hilfe einer Tabelle, deren Zeilen $LR(k)$-Tafeln sind, steuert er die Berechnung. Zu Beginn enthält der Keller die Starttafel $T(\mathfrak{a}(\varepsilon))$ und sonst nichts. Beachte, daß $T(\mathfrak{a}(\varepsilon)) \neq \emptyset$. Das Eingabeband enthält den String, für den die Rechtsableitung konstruiert werden soll. Das Ausgabeband ist leer. Der $LR(k)$-Parser führt dann folgenden Algorithmus aus:

Algorithmus KANLRPARS

Eingabe: $(\mathfrak{T}, T(\mathfrak{a}(\varepsilon)))$ für eine $LR(k)$-Grammatik $G = (V, \Sigma, P, S)$, Eingabestring $z \in \Sigma^*$.

Ausgabe: Rechtsableitung für z, falls $z \in L(G)$ und eine Fehlermeldung sonst.

Methode: Führe die Schritte (1) und (2) solange durch, bis z akzeptiert wird oder ein Fehler auftritt. Falls z akzeptiert wird, dann ist der String in der Ausgabe die Rechtsableitung für z.

(1) Bestimme den Lookahead u, bestehend aus den ersten k noch nicht gelesenen Symbolen der Eingabe.

(2) Sei $T(\mathfrak{a}) = \langle \mathrm{action}_{\mathfrak{a}}, \mathrm{goto}_{\mathfrak{a}} \rangle$ diejenige Tafel, die oben auf dem Keller liegt. Berechne $\mathrm{action}_{\mathfrak{a}}(u)$.

(a) Falls $\mathrm{action}_{\mathfrak{a}}(u) = \mathrm{shift}$, dann

$$\begin{cases} \text{Lese nächstes Eingabesymbol } a & \text{falls } a \text{ existiert} \\ \text{und lege es auf den Keller} & \\ \text{Fehlermeldung} & \text{falls } goto_{\mathfrak{a}}(a) = error \end{cases}$$

Berechne $\mathrm{goto}_{\mathfrak{a}}(a)$.

$$\begin{cases} \text{Lege } T(\mathfrak{a}') \text{ auf den Keller und} & \text{falls } \mathrm{goto}_{\mathfrak{a}}(a) = \mathfrak{a}' \\ \text{springe zu Schritt (1)} & \\ \text{Fehlermeldung} & \text{sonst} \end{cases}$$

(b) Falls $\mathrm{action}_{\mathfrak{a}}(u) = \mathrm{reduce}\ l$, dann
- führe $2|\beta|$ Pop-Operationen durch, wobei $A \to \beta$ die l-te Produktion ist,
- lege A auf den Keller,
- lege $\mathrm{goto}_{\mathfrak{a}}(A)$ auf den Keller,
- schreibe l auf das Ausgabeband und
- springe zu Schritt (1).

(c) Falls $\mathrm{action}_{\mathfrak{a}}(u) = \mathrm{accept}$, dann
- akzeptiere.

(d) Falls $\mathrm{action}_{\mathfrak{a}}(u) = \mathrm{error}$, dann
- gib eine Fehlermeldung aus.

Folgende Beobachtung ist für das Verständnis von $LR(k)$-Parsern von zentraler Bedeutung:

- Eine Tafel im Keller assoziiert stets zu demjenigen String $\gamma \in V^*$, der sich aus den Grammatiksymbolen unterhalb der Tafel im Keller ergibt.

Diese Beobachtung kann leicht mittels Induktion über die Anzahl der Rechenschritte bewiesen werden.

Übung 6.18: *Beweisen Sie die Korrektheit des Algorithmus* KANLRPARS.

Eine $LR(k)$-Tafel kann sehr groß sein. Daher ist es häufig besser, jede $LR(k)$-Tafel einmal abzuspeichern und anstatt die Tafel selbst einen Zeiger auf die abgespeicherte Tafel auf den Keller abzulegen.

Satz 6.6 *Die Anzahl der Schritte, die der Algorithmus* KANLRPAS *bei einem Eingabestring der Länge n durchführt, ist $O(n)$.*

Beweis: Es ist klar, daß der Algorithmus genau einmal eine Fehlermeldung ausgibt bzw. akzeptiert. Es werden maximal n Shiftoperationen durchgeführt. Zu zeigen ist also noch, daß die Anzahl der Reduktionsschritte $O(n)$ ist.

Betrachte einen Reduktionsschritt $\mathrm{reduce}\ l$. Sei $A \to \beta$ die l-te Produktion. In Abhängigkeit, wie die bereits konstruierte Rechtsableitung bezüglich β aussieht, erhält $\mathrm{reduce}\ l$ einen von drei möglichen Typen:

1. $\beta \underset{\mathrm{rm}}{\overset{*}{\Longrightarrow}} \varepsilon$.
   Dann sagen wir, $\mathrm{reduce}\ l$ ist eine *ε-Reduktion.*
2. $\beta = \beta_1\beta_2$ mit $\beta_1 \underset{\mathrm{rm}}{\overset{*}{\Longrightarrow}} z_1$ und $\beta_2 \underset{\mathrm{rm}}{\overset{*}{\Longrightarrow}} z_2$, wobei $z_1, z_2 \in \Sigma^+$.
   Dann sagen wir, daß $\mathrm{reduce}\ l$ *zusammenführend* ist.

3. $\beta = \beta_1 B \beta_2$ mit $\beta_1 \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} \varepsilon$, $\beta_2 \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} \varepsilon$ und $B \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} z$, wobei $z \in \Sigma^*$, $B \in N$.
   Dann sagen wir, daß $\mathrm{reduce}\, l$ *kettenbildend* ist.

Eine ε-Reduktion $\mathrm{reduce}\, l$ ist *terminierend*, falls die Variable A, die β ersetzt, Teil der rechten Seite einer Produktion ist, die zu einem zusammenführenden oder kettenbildenden Reduktionsschritt korrespondiert. Bezeichne

Z	die Anzahl der zusammenführenden Reduktionsschritte,
K	die Anzahl der kettenbildenden Reduktionsschritte,
T	die Anzahl der terminierenden ε-Reduktionen,
U	die Anzahl der nichtterminierenden ε-Reduktionen und
r	die maximale Länge der rechten Seite einer Produktion.

Folgende Fakten sind leicht einzusehen:

1. $T < r(Z + K)$
2. $Z \leq n - 1$.

Da die zugrundeliegende Grammatik G eindeutig ist, gibt es für jede Variable maximal eine Möglichkeit, ε aus ihr abzuleiten. Bezeichne l die maximale Länge einer solchen Ableitung. Dann gilt offensichtlich

3. $U \leq l \cdot T$.

Somit ist nur noch $K = O(n)$ zu beweisen. Wegen $Z \leq n - 1$ genügt es zu zeigen, daß zwischen zwei zusammenführenden Reduktionsschritten nur konstant viele kettenbildende Reduktionsschritte sein können. Aus der Eindeutigkeit von G folgt unmittelbar, daß diese Anzahl durch $|N| - 1$ beschränkt ist.

■

Übung 6.19*: *Ermitteln Sie die bestmögliche Konstante, die sich bezüglich des Satzes 6.6 hinter dem $O(n)$ verbirgt.*

Satz 6.7 *Die Anzahl der Tafeln in der kanonischen Menge von $LR(k)$-Tafeln einer $LR(k)$-Grammatik $G = (V, \Sigma, P, S)$ ist $\leq 2^{2|G||\Sigma|^k}$.*

Beweis: Da es maximal $|G|$ verschiedene Items $A \to \beta_1.\beta_2$ gibt und $|\Sigma^{\leq k}| \leq 2 \cdot |\Sigma|^k$, ist die Anzahl von verschiedenen $LR(k)$-Items durch $2 \cdot |G| \cdot |\Sigma^k|$ beschränkt. Die Tafeln korrespondieren zu paarweise verschiedenen Teilmengen der Menge aller $LR(k)$-Items. Also ist die Anzahl der Tafeln maximal $2^{2|G||\Sigma|^k}$.

■

6.5 Ergänzende Übungsaufgaben

Übung 6.20: *Zeigen Sie, daß die durch das Produktionensystem* $P = \{S \to AB, A \to a, B \to CD \mid aE, C \to ab, D \to bb, E \to bba\}$ *definierte kontextfreie Grammatik* G *nicht* $LR(1)$ *ist.*

Übung 6.21: *Entscheiden Sie, welche der folgenden Grammatiken* $LR(1)$ *sind, und konstruieren Sie für die* $LR(1)$*-Grammatiken die zugehörigen Parsertafeln:*

a) $S \to AB, A \to 0A1 \mid \varepsilon, B \to 1B \mid 1$

b) $S \to 0S1 \mid A, a \to 1A \mid 1$

c) $S \to S + A \mid A, A \to (S) \mid a(S) \mid a$

Welche der obigen Grammatiken sind $LR(0)$*?*

Übung 6.22: *Berechnen Sie die kanonische Kollektion von* $LR(1)$*-Items für die durch das Produktionensystem* $P = \{S \to A, A \to AaAb \mid \varepsilon\}$ *definierte kontextfreie Grammatik.*

6.6 Literaturhinweise

$LL(k)$-Grammatiken wurden Mitte der 60er Jahre erstmals von LEWIS und STEARNS [LS68] definiert. Das einfache Fixpunktverfahren zur Berechnung von $\mathrm{FIRST}_k(\alpha)$ kann zum Beispiel bereits in [AU72] nachgelesen werden. Die hier vorgestellte effiziente Implementierung ist von BLUM und ansonsten nicht publiziert. Der erweiterte $LL(k)$-Parser wurde 1996 von BLUM entwickelt [Blu98]. 1965 hat KNUTH $LL(k)$-Grammatiken eingeführt [Knu65]. Eine sehr gute Beschreibung von $LR(k)$-Parser enthält bereits das gute Buch [AU72] von AHO und ULLMAN. Die dort definierte *e-free-first function ist nicht notwendig, was in der vorliegenden Beschreibung von* $LR(k)$*-Parser berücksichtigt worden ist. 1987 veröffentlichte* CHAPMAN *[Cha87] ein sehr gutes Buch über* LR*-Parser. In der Abhandlung [SSS90] beschreiben* SIPPU *und* SOISALON-SOININEN *den Stand der Forschung von 1990.*

Literaturverzeichnis

[AU72] *A. V. Aho and J. D. Ullman.* The Theory of Parsing, Translation, and Compiling, Vol. I: Parsing. *Prentice-Hall, 1972.*

[Blu98] *N. Blum. On parsing ll-languages. In* 3. WIA, LNCS 1660, 13–21, *1998.*

[Cha87] *N. P. Chapman.* LR Parsing: Theory and Practice. *Cambridge University Press, 1987.*

[Knu65] *D. E. Knuth. On the translation of languages from left to right.* Information and Control 8, 607–639, *1965.*

[LS68] *P. M. Lewis II and R. E. Stearns. Syntax directed transduction.* Journal of the ACM 15, 464–488, *1968.*

[SSS90] *S. Sippu and E. Soisalon-Soininen.* Parsing Theory, Vol. II: $LR(k)$ and $LL(k)$ Parsing. *Springer, 1990.*