Die LR-Analysatoren arbeiten anders. Sie „vertagen" die Entscheidung, die der LL-Analysator bei der Expansion trifft, indem sie zu jedem Zeitpunkt der Analyse alle Möglichkeiten, die zu einer (Rechts-)Ableitung für das Eingabewort führen können, parallel verfolgen. Erst, wenn eine solche Situation zur Entscheidung, d.h. zur Auswahl oder zum Verwerfen von Möglichkeiten zwingt, treffen sie diese Entscheidung, und zwar aufgrund ihres Kellerinhaltes und ebenfalls einer beschränkten Zahl von Symbolen Vorausschau. LR-Analysatoren sind Rechtsparser.

8.2.4 Grammatikflußanalyse

Im weiteren benötigen wir häufig Informationen über Eigenschaften von kontextfreien Grammatiken, z.B. über die Menge aller Syntaxbäume zu einer kfG. Oft assoziieren wir diese Informationen mit Nichtterminalen der Grammatik. I.a. gehören zu einem Nichtterminal X unendlich viele Teilbäume mit X als Wurzelmarkierung. Außerdem gibt es i.a. unendlich viele sogenannte obere Baumfragmente für X; sie erhält man, indem man aus einem Syntaxbaum einen Teilbaum für X ohne die Wurzel X heraustrennt (siehe Abbildung 8.7). Mithilfe der Grammatikflußanalyse (GFA) versucht man, endliche bzw. endlich darstellbare Informationen über diese beiden unendlichen Mengen zu berechnen.

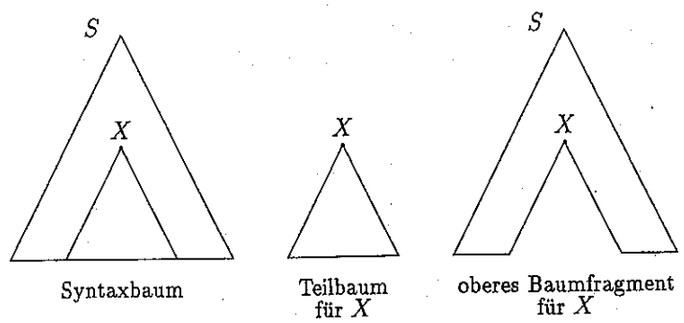

Abb. 8.7: Syntaxbaum, Teilbaum, oberes Baumfragment

Beispiel 8.2.11
Das Problem festzustellen, ob ein Nichtterminal X produktiv ist, ist äquivalent zu dem Problem festzustellen, ob die Menge der Teilbäume zu X nicht leer ist. X ist erreichbar, wenn es mindestens ein oberes Baumfragment für X gibt. Allerdings gilt hier nicht die Äquivalenz, wie wir später sehen werden. □

Informationen über die Menge aller Teilbäume für X können berechnet werden, indem man rekursiv Informationen über die Mengen aller Teilbäume für die Nichtterminale in Produktionen für X benutzt.

Da hier Informationen aus rechten Produktionenseiten - in bildlicher Darstellung meist unten - benutzt werden, um Informationen für linke Seiten - in bildlicher Darstellung meist oben - zu gewinnen, spricht man von einem **bottom up-Problem**. Ein bottom up-Parser setzt schließlich auch Bäume für eine rechte Produktionenseite zu einem Baum für das Nichtterminal der linken Seite zusammen.

Umgekehrt berechnet man Informationen über die Menge der oberen Baumfragmente von X unter Benutzung von Informationen über die Nichtterminale, in deren Produktionen X rechts auftritt. Da man hier Informationen über die linke Seite zur Berechnung von Informationen für die rechte Seite benutzt, spricht man von einem **top down-Problem**.

Zum Lösen dieser beiden Klassen von Problemen werden wir zwei Techniken, die **bottom up-** bzw. die **top down-Grammatikflußanalyse** einführen.

In den folgenden Abschnitten benutzen wir eine für die Darstellung von Grammatikflußproblemen geeignete Notation für kontextfreie Produktionen. Jede Produktion bekommt eine Bezeichnung. Die meist dafür benutzten Namen sind p, q, r usw. Die **Stelligkeit** einer Produktion, das ist die Zahl der Vorkommen von Nichtterminalen auf der rechten Seite, belegen wir immer mit den Namen n_p, n_q, n_r usw. Ist also $p = (X_0 \rightarrow u_0 X_1 u_1 \ldots X_{n_p} u_{n_p})$ mit $X_i \in V_N, u_i \in V_T^*$, so steht $p[i]$ für $X_i, 0 \leq i \leq n_p$. Ein **Vorkommen** eines Nichtterminals X in p an der Position i, d.h. $p[i] = X$, bezeichnen wir mit (p, i). Außerdem nehmen wir immer an, daß die Grammatiken erweitert sind. Somit tritt S nicht auf einer rechten Seite auf.

Betrachten wir wieder die Produktivität von Nichtterminalen. $X \in V_N$ ist **produktiv** genau dann, wenn es ein Terminalwort w gibt mit $X \stackrel{*}{\Longrightarrow} w$. Eine zu einem Algorithmus führende geeignete rekursive Definition ist die folgende:

(1) X ist **produktiv über die Produktion** p genau dann, wenn $p[0] = X$ ist, und wenn alle $p[i]$ für $1 \leq i \leq n_p$ produktiv sind. Insbesondere ist X produktiv über eine Produktion $X \rightarrow u$, $u \in V_T^*$, wenn eine solche existiert.

(2) X ist **produktiv**, wenn X über mindestens eine seiner Alternativen produktiv ist.

Diese Zweistufigkeit der Definition ist typisch für alle Probleme, die wir durch Grammatikflußanalyse lösen werden:

(1) stellt eine Abhängigkeit der Information zu einem Nichtterminal X von den Informationen über die Symbole in der rechten Seite einer Produktion für X dar; (2) gibt an, wie die aus den verschiedenen Alternativen für X erhaltene Information zu kombinieren ist.

Nun betrachten wir die Analyse der Erreichbarkeit von Nichtterminalen. Wieder hilft eine zweistufige rekursive Definition.

(1) Ein Nichtterminal X ist **erreichbar** bei seinem Vorkommen (p, i) für ein i mit $1 \leq i \leq n_p$, wenn $p[0]$ erreichbar ist.

(2) X ist erreichbar, wenn es bei einem seiner Vorkommen erreichbar ist.

(3) Das Startsymbol ist erreichbar.

Die letzte Festlegung sorgt für den richtigen Start der Analyse. (1) legt fest, daß sich die Erreichbarkeit von linken auf rechte Produktionsseiten überträgt. (2) bestimmt, daß schon ein Vorkommen von X auf einer rechten Seite mit erreichbarer linker Seite reicht. Hier geht es um eine Kombination der Informationen, die bei allen Vorkommen von X auf rechten Produktionsseiten angefallen sind.

Jetzt werden wir von diesen beiden Beispielen abstrahieren, um zu unseren Schemata für Algorithmen zu kommen. Als Parameter der Schemata kann man jetzt schon die folgenden identifizieren:

- Wertebereiche für die zu berechnende Information;
- **Transferfunktionen** zu allen Produktionen der kfG; sie beschreiben die Abhängigkeit zwischen Informationen auf der linken und rechten Seite von Produktionen;
- eine **Kombinationsfunktion**; sie beschreibt, wie eine Menge von Informationen für ein Nichtterminal zu einer Information zu kombinieren ist.

Die entsprechenden Komponenten in der Definition des bottom up-Grammatikflußanalyseproblems bekommen ein '↑' an ihre Bezeichnung gehängt, im top down-Fall ein '↓'. Ziel einer Grammatikflußanalyse ist es, eine Funktion $I : V_N \to D$ zu berechnen, die jedem Nichtterminal X Information aus dem Wertebereich D zuordnet, so daß I ein aus der Grammatik, den Transferfunktionen und der Kombinationsfunktion konstruiertes Gleichungssystem erfüllt.

Definition 8.2.16 (bottom up – Grammatikflußanalyseproblem)
Sei G eine kfG. Ein **bottom up – Grammatikflußanalyseproblem** für G und I besteht aus

- einem Wertebereich $D\uparrow$; dieser Wertebereich ist die Menge der möglichen Informationen zu Nichtterminalen;
- einer **Transferfunktion** $F_p\uparrow$: $D\uparrow^{n_p} \to D\uparrow$ für jede Produktion $p \in P$,
- einer **Kombinationsfunktion** $\nabla\uparrow$: $2^{D\uparrow} \to D\uparrow$.

Durch diese Festlegung wird für eine gegebene kfG ein rekursives Gleichungssystem definiert:

$$\boxed{I(X) \quad = \nabla\uparrow \{F_p\uparrow (I(p[1]), \ldots, I(p[n_p])) \mid p[0] = X\} \ \forall X \in V_N} \qquad (I\uparrow)$$

□

Beispiel 8.2.12 (Produktivität von Nichtterminalen)

$D\uparrow$	$\{true, false\}$	*true* für produktiv
$F_p\uparrow$	$\bigwedge$	(*true* für $n_p = 0$, d.h. für terminale Produktionen)
$\nabla\uparrow$	$\bigvee$	(*false* für Nichtterminale ohne Alternativen)

Das Gleichungssystem für das Problem der Produktivität von Nichtterminalen ist:

$$\boxed{Pr(X) \quad = \bigvee\{\bigwedge_{i=1}^{n_p} Pr(p[i]) \mid p[0] = X\} \text{ für alle } X \in V_N} \qquad (Pr)$$

□

Definition 8.2.17 (top down – Grammatikflußanalyseproblem)
Sei G eine kfG. Ein **top down – Grammatikflußanalyseproblem** für G und I besteht aus

- einem Wertebereich $D\downarrow$;
- n_p **Transferfunktionen** $F_{p,i}\downarrow$: $D\downarrow \to D\downarrow$, $1 \leq i \leq n_p$, für jede Produktion $p \in P$,
- einer **Kombinationsfunktion** $\nabla\downarrow$: $2^{D\downarrow} \to D\downarrow$,
- einem Wert I_0 für S unter der Funktion I.

Dadurch wird wiederum für eine gegebene kfG ein rekursives Gleichungssystem für I definiert; der Verständlichkeit halber definieren wir I für Nichtterminale und für Vorkommen von Nichtterminalen:

$$\boxed{\begin{array}{lcl} I(S) & = & I_0 \\ I(p,i) & = & F_{p,i}\downarrow (I(p[0])) \text{ für alle } p \in P,\ 1 \leq i \leq n_p \\ I(X) & = & \nabla\downarrow \{I(p,i) \mid p[i] = X\}, \text{ für alle } X \in V_N - \{S\} \end{array}} \qquad (I\downarrow)$$

□

Beispiel 8.2.13 (Erreichbare Nichtterminale)

$D\downarrow$	$\{true, false\}$	*true* für erreichbar
$F_{p,i}\downarrow$	*id*	identische Abbildung
$\nabla\downarrow$	$\bigvee$	Boolesches Oder
		(*false*, falls es kein Vorkommen für ein Nichtterminal gibt)
I_0	*true*	

Daraus ergibt sich für Er das rekursive Gleichungssystem

$$\boxed{\begin{array}{lcl} Er(S) & = & true \\ Er(X) & = & \bigvee\{Er(p[0]) \mid p[i] = X,\ 1 \leq i \leq n_p\} \ \ \forall X \in V_N - \{S\} \end{array}} \qquad (Er)$$

□

Sowohl für die beiden Schemata als auch für die beiden Beispiele ergeben sich rekursive Gleichungen, von deren Lösungen wir die gewünschte Information über die Nichtterminale der gegebenen Grammatik erwarten. Natürlich stellen sich jetzt mehrere Fragen:

- Gibt es überhaupt Lösungen für die sich ergebenden Gleichungssysteme?
- Welche Lösung ist die gesuchte, wenn es zu einem Gleichungssystem mehrere gibt?
- Wie berechnet man Lösungen bzw. gewünschte Lösungen?

Zur Beantwortung dieser Fragen diskutieren wir erst einmal einen naheliegenden Weg und studieren dann die notwendigen Voraussetzungen für diese Gleichungssysteme.
Sei ein rekursives Gleichungssystem

$$\begin{aligned} x_1 &= f_1(x_1, \ldots, x_n) \\ x_2 &= f_2(x_1, \ldots, x_n) \\ &\vdots \\ x_n &= f_n(x_1, \ldots, x_n) \end{aligned}$$

gegeben. Es solle über einem Bereich D gelöst werden. Enthält D ein kleinstes Element – bezeichnen wir dieses wie üblich mit $\bot$, so könnten wir folgendes versuchen. Wir setzen für alle $x_1, \ldots, x_n$ den Wert $\bot$ ein und wenden alle f_i auf dieses $(\bot, \ldots, \bot)$-Tupel, welches wir mit $\overline{x}^0$ bezeichnen, an. Dann erhalten wir ein neues Tupel $\overline{x}^1$, auf welches wir wiederum alle f_i anwenden, usw. Es ergibt sich eine Folge $(\overline{x}^0, \overline{x}^1, \overline{x}^2, \ldots)$. I.a. führt das zu nichts. In allen Fällen, die wir betrachten, können wir allerdings garantieren, daß dieses Verfahren gegen eine von uns gewünschte Lösung konvergiert. Erstens wissen wir einiges über die im folgenden auftretenden Bereiche:

- alle sind endlich;
- auf allen ist eine partielle Ordnung definiert, dargestellt durch das Symbol $\sqsubseteq$;
- alle enthalten das oben geforderte, eindeutig bestimmte kleinste Element $\bot$, lies bottom;
- die kleinste obere Schranke, i.Z. $\sqcup$, ist für je zwei Bereichselemente definiert.

Diese Bereiche bilden also jeweils zusammen mit ihrer Operation $\sqcup$ und ihrer partiellen Ordnung $\sqsubseteq$ endliche, **vollständige $\sqcup$-Halbverbände**, geschrieben als $(D, \sqsubseteq, \bot, \sqcup)$.

Zweitens sind alle Funktionen monoton. Weil $\overline{x}^0$ kleinstes Element ist, gilt sicher, daß $\overline{x}^0 \sqsubseteq \overline{x}^1$. Aus der Monotonie der Funktionen f_i folgt daraus mittels Induktion $\overline{x}^0 \sqsubseteq \overline{x}^1 \sqsubseteq \overline{x}^2 \sqsubseteq \cdots$. Da der Bereich D endlich ist, gibt es ein n, sodaß $\overline{x}^n = \overline{x}^{n+1}$. Dann gilt aber auch für alle $m > n$, daß $\overline{x}^m = \overline{x}^n$. Das so erhaltene $\overline{x}^n$ ist eine Lösung des Gleichungssystems. Es läßt sich leicht zeigen, daß es die kleinste Lösung ist, genannt der kleinste Fixpunkt.

In GFA-Problemen setzen sich die Funktionen f_i aus den Transfer- und den Kombinationsfunktionen zusammen und zwar so, daß die Monotonie der

Transfer- und Konbinationsfunktionen die Monotonie der zusammengesetzten Funktionen impliziert.

Wir müssen also für jedes Flußanalyseproblem nachweisen, daß der Wertebereich ein endlicher vollständiger $\sqcup$-Halbverband ist, und daß die Transferfunktionen und die Kombinationsfunktion monoton sind.

Wenn $V_N = \{X_1, \ldots, X_n\}$ ist, so ordnen wir die Nichtterminale in der Reihenfolge $X_1, X_2, \ldots, X_n$ an und konstruieren in naheliegender Weise aus dem Gleichungssystem $(I\uparrow)$ eine Funktion $F\uparrow\colon D\uparrow^n \to D\uparrow^n$, die sogenannte **Gesamtschrittfunktion** (analog für top down-Probleme); $F\uparrow$ berechnet in einem (Gesamt-) Schritt neue Informationen für alle Nichtterminale und benutzt dabei jeweils die den Produktionen zugeordneten Funktionen $F_p\uparrow$. $F\uparrow$ ist definiert durch $F\uparrow(d_1, \ldots, d_n) = (d_1', \ldots, d_n')$ mit

$$d_i' = \nabla\uparrow \{F_p\uparrow(d_{i_1}, \ldots, d_{i_{n_p}}) \mid p[0] = X_i, p[j] = X_{i_j} \text{ für } 1 \le j \le n_p\}.$$

Sind die $F_p\uparrow$ und auch $\nabla\uparrow$ monoton, so ist auch $F\uparrow$ monoton. Ist $D\uparrow$ endlich, so auch $D\uparrow^n$. Das bottom-Element $\bot^n$ von $D\uparrow^n$ ist gleich $(\bot, \ldots, \bot)$. Somit beginnen wir die Iteration unter $F\uparrow$ mit $\bot^n$. Bei Konvergenz der Iteration liegt der kleinste Fixpunkt vor.

Jetzt greifen wir unsere beiden Beispiele wieder auf und zeigen, daß für sie die hinreichenden Bedingungen für die Fixpunktberechnung erfüllt sind.

Beispiel 8.2.14 (produktive Nichtterminale)
$D = \{true, false\}$ mit $false \sqsubseteq true$, also $\bot = false$ und $\top = true$, ist ein endlicher vollständiger $\sqcup$-Verband. Dabei wird $Pr(X) = true$ interpretiert als „X wurde schon als produktiv erkannt“. $Pr(X) = false$ während der Iteration heißt „Produktivität noch unbekannt“; nach der Konvergenz zum kleinsten Fixpunkt heißt $Pr(X) = false$, daß X unproduktiv ist. Die den Produktionen zugeordneten Transferfunktionen, also die Konjunktionen verschiedener Stelligkeit, sind monoton; ebenso die Kombinationsfunktion, die Disjunktion. Also ist auch die aus ihnen konstruierte Funktion $pr : D^n \to D^n$ monoton. Die iterative Berechnung des kleinsten Fixpunkts von pr beginnt mit dem bottom-Element in D^n, also $(false, \ldots, false)$. □

Beispiel 8.2.15 (produktive Nichtterminale)
Sei folgende Grammatik gegeben:

$$G = (\{S', S, X, Y, Z\}, \{a, b\}, \left\{ \begin{array}{lcl} S' & \to & S \\ S & \to & aX \\ X & \to & bS \mid aYbY \\ Y & \to & ba \mid aZ \\ Z & \to & aZX \end{array} \right\}, S')$$

Das Gleichungssystem (Pr) war gegeben durch:

$$Pr(A) = \bigvee_{\{p \mid p[0]=A\}} \left(\bigwedge_{i=1}^{n_p} Pr(p[i]) \right) \text{ (für alle } A \in V_N)$$

Angewandt auf obige Grammatik ergibt sich das folgende Gleichungssystem: (Wir betrachten S' nicht, da $Pr(S') = Pr(S)$ gilt)

$$\begin{array}{lcl} Pr(S) & = & Pr(X) \\ Pr(X) & = & Pr(S) \vee Pr(Y) \\ Pr(Y) & = & true \vee Pr(Z) = true \\ Pr(Z) & = & Pr(Z) \wedge Pr(X) \end{array}$$

Wir wählen auf V_N die Anordnung S, X, Y, Z und erhalten als Gesamtschrittfunktion $pr : \{true, false\}^4 \to \{true, false\}^4$ definiert durch

$$pr(s, x, y, z) = (x, s \vee y, true, z \wedge x).$$

Die Fixpunktiteration startet mit $p = (false,\ false,\ false,\ false)$.

$$\begin{array}{lcl} pr(p) & = & (false,\ false,\ true,\ false). \\ pr^2(p) & = & (false,\ true,\ true,\ false). \\ pr^3(p) & = & (true,\ true,\ true,\ false). \\ pr^4(p) & = & (true,\ true,\ true,\ false). \end{array}$$

Das Ergebnis hat damit die Interpretation, daß Z unproduktiv ist, während alle anderen Nichtterminale produktiv sind. □

Beispiel 8.2.16 (erreichbare Nichtterminale)
Der allen Nichtterminalen zugeordnete $\sqcup$-Halbverband ist wieder $D = \{false, true\}$ mit $false \sqsubseteq true$. Die Transferfunktionen sind die Identität, die Kombinationsfunktion ist die Disjunktion. Beide sind monoton. Also ist auch die aus ihnen konstruierte Gesamtschrittfunktion $er : D^n \to D^n$ monoton. Die Iteration startet wieder mit $(false, \ldots, false)$. □

Beispiel 8.2.17 (erreichbare Nichtterminale)
Sei folgende Grammatik gegeben:

$$G = (\{S, U, V, X, Y, Z\}, \{a, b, c, d\}, \left\{ \begin{array}{lcl} S & \to & Y \\ Y & \to & YZ \mid Ya \mid b \\ U & \to & V \\ X & \to & c \\ V & \to & Vd \mid d \\ Z & \to & ZX \end{array} \right\}, S)$$

Das Gleichungssystem (Er) war gegeben durch:
$Er(X) = \bigvee\{Er(p[0]) \mid p[i] = X,\ 1 \le i \le n_p\}$ für alle $X \in V_N - \{S\}$ und $Er(S) = true$.

Für die Beispielgrammatik ergibt sich:

$$\begin{array}{lcl} Er(S) & = & true \\ Er(U) & = & false \\ Er(V) & = & Er(U) \vee Er(V) \\ Er(X) & = & Er(Z) \\ Er(Y) & = & Er(S) \vee Er(Y) \\ Er(Z) & = & Er(Y) \vee Er(Z) \end{array}$$

Wir wählen die Anordnung S, U, V, X, Y, Z auf den Nichtterminalen und erhalten die Gesamtschrittfunktion

$$er(s, u, v, x, y, z) = (true, false, v, z, s \vee y, y \vee z).$$

Die Iteration startet mit dem Wert $e = (true,\ false,\ false,\ false,\ false,\ false)$

$$\begin{array}{lcl} er(e) & = & (true,\ false,\ false,\ false,\ true,\ false). \\ er^2(e) & = & (true,\ false,\ false,\ false,\ true,\ true). \\ er^3(e) & = & (true,\ false,\ false,\ true,\ true,\ true). \\ er^4(e) & = & (true,\ false,\ false,\ true,\ true,\ true). \end{array}$$

Das Ergebnis kann man so interpretieren, daß U, V nicht erreichbar, alle anderen Nichtterminale erreichbar sind. Man beachte jedoch, daß Z unproduktiv ist, und daß nach Eliminierung von Z und den Produktionen, in denen Z auftritt, X nicht mehr erreichbar ist. Will man durch Eliminieren von unproduktiven und nicht erreichbaren Nichtterminalen eine Grammatik reduziert machen, so empfiehlt es sich, erst die unproduktiven und dann die nicht erreichbaren Nichtterminale zu eliminieren. □

Dies können wir auch dadurch ausdrücken, daß wir eine neue, schärfere Definition der Erreichbarkeit und damit ein neues top down-GFA-Problem definieren. Demnach ist ein Nichtterminal X bei seinem Vorkommen (p, i) nur dann erreichbar, wenn $p[0]$ erreichbar ist, und wenn alle $p[j]$ mit $1 \le j \le n_p$ und $i \ne j$ produktiv sind. Das sich ergebende top down-Problem basiert also auf vorher berechneten Informationen des bottom up-Problems, Produktivität von Nichtterminalen. Dieses ist bei den meisten top down-GFA-Problemen der Fall.

Das Gleichungssystem für das neue Erreichbarkeitsproblem ist:

$$\boxed{\begin{array}{lcl} Er(S) & = & true \\ Er(X) & = & \bigvee\{Er(p[0]) \wedge \bigwedge_{1 \le j \le n_p, i \ne j} Pr(p[j]) \mid p[i] = X,\ 1 \le i \le n_p\} \\ & & \text{für alle } X \in V_N - \{S\} \end{array}} \qquad (Er)$$

Statt der Iteration über das Gesamtschrittverfahren kann man verschiedene Einzelschrittverfahren zur Berechnung des kleinsten Fixpunkts benutzen. In jedem Einzelschrittverfahren berechnet man in jedem Schritt nur die Information für ein Nichtterminal neu. Ansonsten wird die Information übernommen. Dabei ist es eine vernünftige Strategie, ein Nichtterminal erst dann zu besuchen, wenn vorher alle seine Vorgänger, d.h. Nichtterminale auf einer seiner rechten

Seiten (seit dem letzten Durchgang durch diesen Knoten) besucht worden sind, und im Falle rekursiver Nichtterminale bei den „obersten“ in dieser Rekursion zu beginnen. Außerdem muß darüber Buch geführt werden, ob sich Informationen noch geändert haben, so daß weitere Berechnungen erforderlich sind. Sobald alle Nichtterminale einmal konsekutiv besucht wurden, ohne daß sich Information geändert hat, ist eine stabile Lösung, der kleinste Fixpunkt erreicht.

Im weiteren werden wir die Elemente, die ein GFA-Problem charakterisieren, jeweils in einem Kasten der folgenden Art zusammenfassen:

bottom up-/top down-GFA-Problem <*Name*>
H-Verband
Transferfunktion
Kombinationsfunktion
Anfangsbelegung für S

Das Problem der Produktivität wird folgendermaßen beschrieben:

bottom up-GFA-Problem PRODUKTIV	
H-Verband	$(\{false, true\}, \{false \sqsubseteq true\}, false, \vee)$
Transferfunktion	$\wedge$
Kombinationsfunktion	$\vee$

8.2.5 Ein lineares Verfahren

„Produktive Nichtterminale“ und „Erreichbare Nichtterminale“ gehören zu den einfachsten Problemen, die sich sinnvoll als GFA-Probleme beschreiben lassen. Deshalb nimmt es nicht wunder, daß es für sie viel effizientere Spezialverfahren als das Gesamtschrittverfahren oder die Iteration über die Grammatik gibt. Es reicht für beide Probleme nämlich eine geschickte Durchmusterung der Grammatik, die jedes Nichtterminal höchstens einmal bearbeitet.

Algorithmus: PRODUKTIV
Eingabe: Eine kontextfreie Grammatik G
Ausgabe: Lösung des Gleichungssystems (Pr) für G
Methode: Bottom-up-Durchmusterung der Grammatik, beginnend mit den terminalen Produktionen. Nach einer Initialisierungsphase werden in der Hauptschleife die Elemente von *zuverarbeiten* nacheinander betrachtet. Dabei bleibt folgende Invariante erhalten:

(a) Wenn $Pr[X] = true$, so ist X produktiv.

(b) Wenn $restNT[p] = 0$, so ist $Pr[p[0]] = true$.

(c) $restNT[p]$ ist die Anzahl der Nichtterminalvorkommen $p[i] = X \quad (i > 0)$ mit $Pr[X] = false$ oder $X \in zuverarbeiten$.

Bei Termination mit $zuverarbeiten = \emptyset$ folgt dann $Pr[X] = true$ gdw. X produktiv. (Zusätzlich sind durch die Bedingung $restNT[p] > 0$ die unproduktiven Produktionen p gekennzeichnet.) Die Linearität des Algorithmus ergibt sich, wenn die Vorkommen jedes Nichtterminals in Listen verwaltet werden, die effizient aus der Grammatik abgelesen werden können.


```
var   Pr: array[V_N] of boolean initially false;
      restNT: array[P] of integer;
      zuverarbeiten: set of V_N initially ∅;

begin
      for each Produktion p do
            (* n_p Vorkommen rechts in p noch nicht als produktiv erkannt *)
            restNT[p] := n_p;
            if restNT[p] = 0 then    (* terminale Produktion *)
                  Pr[p[0]] := true;
                  zuverarbeiten := zuverarbeiten ∪ {p[0]}
            fi
      od;
      while zuverarbeiten ≠ ∅ do
            wähle X ∈ zuverarbeiten;
            for each Vorkommen von p[i] = X   (i > 0) do
                  restNT[p] := restNT[p] − 1;
                  if restNT[p] = 0 then
                        (* p[0] als produktiv entdeckt *)
                        if not Pr[p[0]] then
                              Pr[p[0]] := true;
                              zuverarbeiten := zuverarbeiten ∪ {p[0]}
                        fi
                  fi
            od;
            zuverarbeiten := zuverarbeiten − {X}
      od
end;
```

Was sind die Besonderheiten des Problems „produktive Nichtterminale“, die diesen effizienten Algorithmus ermöglichen?

1. Der vorliegende Verband hat nur die Höhe zwei, d.h. die längste aufsteigende Kette hat die Länge zwei. Monotonie sorgt dafür, daß ein auf *true* gesetztes $Pr(X)$ nicht wieder auf *false* fallen kann. Jedes auf *true* gesetzte $Pr(X)$ hat also seinen endgültigen Wert bekommen.

2. Weshalb muß nicht über Zyklen iteriert werden? Der Algorithmus PRODUKTIV betritt Zyklen nur über nichtrekursive Produktionen bereits als produktiv erkannter Nichtterminale oder gar nicht. Im ersten Fall kann der Zyklus an dieser Stelle aufgetrennt werden. Tritt der zweite Fall nie auf, so sind alle Nichtterminale auf diesem Zyklus unproduktiv.

8.2.6 *FIRST* und *FOLLOW*

Betrachten wir den Item-Kellerautomaten K_G zu einer kfG G bei einer Expansion, einem (E)-Übergang. Sein aktueller Zustand sei $[X \to \alpha.Y\beta]$. K_G muß nichtdeterministisch aus den Alternativen $Y \to \alpha_1 \mid \ldots \mid \alpha_n$ auswählen. Eventuell könnte er die Entscheidung deterministisch treffen, wenn er die Menge der von den α_i produzierten Wörter oder zumindest ihre Anfänge kennen würde. Wenn der Anfang der restlichen Eingabe nur von einem der α_i produziert werden kann, etwa von α_{i_0}, so wird die Alternative $Y \to \alpha_{i_0}$ auszuwählen sein. Wenn zumindest eines der α_i auch kurze Wörter, z.B. ε, produzieren kann, so ist auch die Menge der Wörter oder ihrer Anfänge interessant, die auf Y folgen können.

Deshalb ist es bei der Generierung von Parsern für kontextfreie Grammatiken häufig notwendig, die Menge der Präfixe bestimmter Länge aller Wörter für jedes Nichtterminal zu kennen. Die erzeugten Parser gründen dann Entscheidungen auf einem Vergleich eines Präfixes der restlichen Eingabe mit den Elementen dieser vorberechneten Menge. Wenn ein Nichtterminal kurze Wörter, z.B. ε, produzieren kann, so ist ein Präfix der restlichen Eingabe auch dann für die gesuchte Entscheidung „günstig", wenn es in einer Satzform auf dieses Nichtterminal folgen kann. Diese beiden Begriffe werden durch die Funktionen $FIRST_k$ und $FOLLOW_k$ formalisiert.

Wir schreiben $V^{\leq k}$ für $\bigcup_{i=0}^{k} V^i$ und $V_{\#}^{\leq k}$ für $V^{\leq k} \cup V^{\leq k-1}\{\#\}$, wobei $\#$ ein Symbol ist, das nicht in V enthalten ist.

Definition 8.2.18 (k-Präfix, k-Konkatenation)
Sei V ein Alphabet und sei $w = a_1 \ldots a_n$, $a_i \in V$ für $(1 \leq i \leq n)$, $n \geq 0$.

$$k : w = \begin{cases} a_1 \ldots a_n & \text{falls} \quad n \leq k \\ a_1 \ldots a_k & \text{sonst} \end{cases} \qquad \text{ist der } \mathbf{k\text{-Präfix}} \text{ von } w.$$

$\oplus_k : V^* \times V^* \to V^{\leq k}$, definiert durch $u \oplus_k v = k : uv$,
heißt die **k-Konkatenation.**
Wir erweitern beide Operationen auf Mengen von Wörtern.

$k : L = \{k : w \mid w \in L\}$
Seien $L_1, L_2 \subseteq V^*$. $L_1 \oplus_k L_2 = \{x \oplus_k y \mid x \in L_1, y \in L_2\}$.

□

Definition 8.2.19 ($FIRST_k$, $FOLLOW_k$)
Sei $G = (V_N, V_T, P, S)$ eine kontextfreie Grammatik.
$FIRST_k : (V_N \cup V_T)^* \to 2^{V_T^{\leq k}}$ mit
$FIRST_k(\alpha) = \{k : u \mid \alpha \overset{*}{\Longrightarrow} u\}$

ist die Menge der Präfixe der Länge k von Terminalworten für α bzw. der Wörter der Länge kleiner oder gleich k für α.
$FOLLOW_k : V_N \to 2^{V_{T\#}^{\leq k}}$ mit
$FOLLOW_k(X) = \{w \mid S \overset{*}{\Longrightarrow} \beta X \gamma \text{ und } w \in FIRST_k(\gamma)\}$

ist die Menge der Terminalworte der Länge kleiner oder gleich k, die in einer Satzform direkt auf X folgen können. $\#$ ist ein nicht zum Satz gehörendes Endesymbol, etwa das Dateiendesymbol **EOF**.

$FIRST_k$ läßt sich leicht auf Mengen von Wörtern verallgemeinern:
Sei $L \subseteq (V_N \cup V_T)^*$. Dann definieren wir $FIRST_k(L) = \bigcup_{\alpha \in L} FIRST_k(\alpha)$ □

Einige Eigenschaften der k-Konkatenation und von $FIRST_k$ beschreibt das folgende

Lemma 8.2.2
Seien $L_1, L_2, L_3 \subseteq V^*$ und $k \geq 1$. Dann gelten

(a) $L_1 \oplus_k (L_2 \oplus_k L_3) = (L_1 \oplus_k L_2) \oplus_k L_3$

(b) $L_1 \oplus_k \{\varepsilon\} = \{\varepsilon\} \oplus_k L_1 = k : L_1$

(c) $L_1 \oplus_k L_2 = \emptyset$, gdw. $L_1 = \emptyset$ oder $L_2 = \emptyset$

(d) $\varepsilon \in L_1 \oplus_k L_2$, gdw. $\varepsilon \in L_1$ und $\varepsilon \in L_2$

(e) $FIRST_k(X_1 \ldots X_n) = FIRST_k(X_1) \oplus_k \ldots \oplus_k FIRST_k(X_n)$

Die Beweise zu (b), (c) und (d) sind trivial. (a) ergibt sich aus Fallunterscheidungen über die Länge von Wörtern $x \in L_1$, $y \in L_2$, $z \in L_3$. Der Beweis zu (e) benutzt mehrere Hilfsaussagen, nämlich:

(1) $X_1 \ldots X_n \overset{*}{\Longrightarrow} u$ gdw. $\exists u_1, \ldots, u_n : X_1 \overset{*}{\Longrightarrow} u_1, \ldots, X_n \overset{*}{\Longrightarrow} u_n$ und $u = u_1 \ldots u_n$.

(2) $k : u_1 \ldots u_n = u_1 \oplus_k \ldots \oplus_k u_n$.

□

Die Berechnungen der Funktionen $FIRST_k(X)$ und $FOLLOW_k(X)$ für Nichtterminale X lassen sich wieder als GFA-Probleme formulieren. $FIRST_k(X)$ besteht aus den k-Präfixen der Blattwörter aller Bäume für X und $FOLLOW_k(X)$ aus den k-Präfixen des zweiten Teils der Blattwörter aller oberen Baumfragmente für X, wie in Abbildung 8.8 gezeigt.

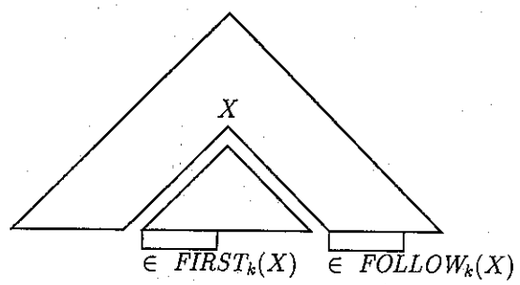

Abb. 8.8: $FIRST_k$ und $FOLLOW_k$ in Syntaxbäumen

Um das Grammatikflußanalyseproblem $FIRST_k$ formulieren zu können, muß man erst die Rekursion über die Grammatik herausfinden, die hinter der Definition von $FIRST_k$ steckt. Zur $FIRST_k$-Menge für ein Nichtterminal X tragen alle Produktionen für X bei. Die Kombinationsfunktion ist die Mengenvereinigung. In einer Produktion $p = (X \to u_0 X_1 u_1 \ldots X_{n_p} u_{n_p})$ für X müssen die Terminalworte $u_0, u_1, \ldots, u_{n_p}$ und die Wörter aus den $FIRST_k$-Mengen der Nichtterminale der rechten Seite zu Wörtern von $FIRST_k(X)$ konkateniert werden. Dabei konkateniert man von links nach rechts, bis die Länge k erreicht ist.

Abbildung 8.9 definiert das bottom up-GFA-Problem $FIRST_k$ sowie das zugehörige Gleichungssystem für eine gegebene kontextfreie Grammatik G. Der zugrundeliegende Halbverband ist $(2^{V_T^{\leq k}}, \subseteq, \emptyset, \cup)$. $V_T^{\leq k}$ ist endlich, und $\{Fir_p\}_{p \in P}$ ist offensichtlich eine Menge monotoner Funktionen.

bottom up-GFA-Problem $FIRST_k$	
H-Verband	$(2^{V_T^{\leq k}}, \subseteq, \emptyset, \cup)$
Transf. Fkt.	$Fir_p(d_1, \ldots, d_{n_p}) = \{u_0\} \oplus_k d_1 \oplus_k \{u_1\} \oplus_k d_2 \oplus_k \ldots \oplus_k d_{n_p} \oplus_k \{u_{n_p}\}$, falls $p = (X_0 \to u_0 X_1 u_1 X_2 \ldots X_{n_p} u_{n_p})$; $Fir_p = k : u$ für eine terminale Produktion $X \to u$
Komb. Fkt.	$\cup$

Das rekursive Gleichungssystem für $FIRST_k$ ist

$$Fi_k(X) = \bigcup_{\{p \mid p[0] = X\}} Fir_p(Fi_k(p[1]), \ldots, Fi_k(p[n_p])) \; \forall X \in V_N \qquad (Fi_k)$$

Abb. 8.9: GFA-Problem $FIRST_k$

Beispiel 8.2.18
Sei G_2 die kontextfreie Grammatik mit den Produktionen:

0 :	S	$\to$	E	3 :	E'	$\to$	$+E$	6 :	T'	$\to$	$*T$
1 :	E	$\to$	TE'	4 :	T	$\to$	FT'	7 :	F	$\to$	(E)
2 :	E'	$\to$	ϵ	5 :	T'	$\to$	ϵ	8 :	F	$\to$	**id**

G_2 erzeugt die gleiche Sprache der arithmetischen Ausdrücke wie G_0 und G_1.

Sei $k = 1$, d.h. wir betrachten das bottom up-GFA-Problem $FIRST_1$ für G_2. Die den Produktionen 0 – 8 zugeordneten Transferfunktionen sind dann:

$Fir_0(d) = d$
$Fir_1(d_1, d_2) = Fir_4(d_1, d_2) = d_1 \oplus_1 d_2$
$Fir_2 = Fir_5 = \{\epsilon\}$, $Fir_3(d) = \{+\}$, $Fir_6(d) = \{*\}$, $Fir_7(d) = \{(\}$, $Fir_8 = \{\mathbf{id}\}$

Eine iterative Berechnung der kleinsten Lösung des $FIRST_1$-Problems für diese kontextfreie Grammatik folgt. Mit allen Nichtterminalen wird anfänglich die Information $\emptyset$ assoziiert. □

Beispiel 8.2.19 (Fortführung von Beispiel 8.2.18.)
Die folgenden drei Spalten beschreiben die Berechnung von Fi_1 für die Nichtterminale der Grammatik. Die Berechnungsschritte sind spaltenweise aufgelistet.

Z.B. besagt der zweite Eintrag in der ersten Spalte: Nach dem Besuch der Produktionsknoten 8 und daraufhin 7 erhält die Fi_1-Menge am Nichtterminal F die Symbole **id** und (.

Prod.	Fi_1								
8	F :	{**id**}	5	T' :	$\{*, \epsilon\}$	2	E' :	$\{+, \epsilon\}$
7	F :	{**id**, (}	4	T :	{**id**, (}	1	E :	{**id**, (}
6	T' :	$\{*\}$	3	E' :	$\{+\}$	0	S :	{**id**, (}

□

Abbildung 8.10 definiert $FOLLOW_k$, das top down-GFA-Problem für eine kontextfreie Grammatik G und das zugehörige rekursive Gleichungssystem. Der Halbverband ist ähnlich wie bei $FIRST_k$. Betrachten wir den Beitrag des Vorkommens (p, i) von X zu $FOLLOW_k(X)$. Hinter diesem Vorkommen von X in der Produktion $p = (X_0 \to u_0 X_1 u_1 \ldots X_{n_p} u_{n_p})$ steht $u_i X_{i+1} \ldots X_{n_p} u_{n_p}$. Die Menge der Wörter, die auf dieses Vorkommen von X folgen kann, erhalten wir also durch die Konkatenation, in der richtigen Reihenfolge, der $\{u_i\}, \{u_{i+1}\}, \ldots, \{u_{n_p}\}$ mit den vorher berechneten Mengen $FIRST_k(X_{i+1}), \ldots, FIRST_k(X_{n_p})$, wobei die entstehenden Wörter nach dem k-ten Symbol abgeschnitten werden. Ist ein auf diese Weise berechnetes Wort weniger als k Symbole lang, so fügen wir hinten noch Wörter aus $FOLLOW_k(X_0)$ an. Dies ist die „Arbeitsweise" der Funktion $Fol_{p,i}$. $\{Fol_{p,i}\}_{p \in P, 1 \leq i \leq n_p}$ ist, wie man leicht sieht, eine Menge von monotonen Funktionen.

top down-GFA-Problem $FOLLOW_k$	
H-Verband	$(2^{V_{T\#}^{\leq k}}, \subseteq, \emptyset, \cup)$
Transf. Fkt.	$Fol_{p,i}(d) = \{u_i\} \oplus_k Fi_k(X_{i+1}) \oplus_k \{u_{i+1}\} \oplus_k \ldots \oplus_k Fi_k(X_{n_p}) \oplus_k \{u_{n_p}\} \oplus_k d$ falls $p = (X_0 \to u_0 X_1 u_1 X_2 \ldots X_{n_p} u_{n_p})$;
Komb. Fkt.	$\cup$
Startwert	$\{\#\}$

Das rekursive Gleichungssystem für $FOLLOW_k$ ist

$$\begin{array}{l} Fo_k(X) = \bigcup_{\{p \mid p[i] = X, 1 \leq i \leq n_p\}} Fol_{p,i}(Fo_k(p[0])) \; \forall X \in V_N - \{S\} \\ Fo_k(S) = \{\#\} \end{array} \qquad (Fo_k)$$

Abb. 8.10: GFA-Problem $FOLLOW_k$

Beispiel 8.2.20 (Fortführung von Beispiel 8.2.18)
Wir betrachten das top down-GFA-Problem $FOLLOW_1$ für die kontextfreie Grammatik G_2 aus Beispiel 8.2.18. Jedem Vorkommen (p, i) eines Nichtterminals X auf einer rechten Seite, d.h. $p[i] = X$, ist eine Transferfunktion zugeordnet.

Wir listen sie auf:

$Fol_{0,1}(d) = d$
$Fol_{1,1}(d) = Fi_1(E')\oplus_1 d = \{+,\varepsilon\}\oplus_1 d,\ Fol_{1,2}(d) = d$
$Fol_{3,1}(d) = d$
$Fol_{4,1}(d) = Fi_1(T')\oplus_1 d = \{*,\varepsilon\}\oplus_1 d,\ Fol_{4,2}(d) = d$
$Fol_{6,1}(d) = d$
$Fol_{7,1}(d) = \{)\}$

Zur Berechnung von Fo_1 für alle Nichtterminale wird S mit $\{\#\}$ initialisiert, alle anderen Nichtterminale mit $\emptyset$. Eine mögliche Iteration berechnet die $FOLLOW_1$-Information in der folgenden Reihenfolge, wieder spaltenweise zu lesen.

Pos.	Fo_1							
(0,1)	E:	$\{\#\}$	(4,1)	F:	$\{*,+,\#\}$	(4,2)	T':	$\{+,\#,)\}$
(1,2)	E':	$\{\#\}$	(7,1)	E:	$\{\#,)\}$	(4,1)	F:	$\{*,+,\#,)\}$
(1,1)	T:	$\{+,\#\}$	(1,2)	E':	$\{\#,)\}$			
(4,2)	T':	$\{+,\#\}$	(1,1)	T:	$\{+,\#,)\}$			

□

8.2.7 Der Spezialfall $FIRST_1$ und $FOLLOW_1$

Ein für die Praxis relevanter Spezialfall ist die Berechnung von $FIRST_1$ und $FOLLOW_1$ in der Generierung von LL(1)-Parsern (siehe Abschnitt 8.3).

Das folgende Lemma ist die Basis für unser weiteres Vorgehen:

Lemma 8.2.3
Für zwei Sprachen $L_1, L_2 \subseteq V^{\leq 1}$ gelten:

(a) $L_1\oplus_1 L_2 = \begin{cases} \emptyset & \text{falls } L_2 = \emptyset \\ L_1 & \text{falls } L_2 \neq \emptyset \text{ und } \varepsilon \notin L_1 \\ (L_1 - \{\varepsilon\}) \cup L_2 & \text{falls } L_2 \neq \emptyset \text{ und } \varepsilon \in L_1 \end{cases}$

(b) $L_1\oplus_1 L_2 \subseteq L_1 \cup L_2$

Da nach impliziter Voraussetzung unsere Grammatiken reduziert sind, enthalten sie keine unproduktiven Nichtterminale. Deshalb gilt für alle $X \in V_N \cup V_T$: $FIRST_1(X) \neq \emptyset$. Wenn wir bei der Lösung von $FIRST_1$-Problemen garantieren können, daß bei Anwendung der Fir_p-Funktion die Argumente nie leer sind, können wir die Funktionen Fir_p für $FIRST_1$ so umschreiben, daß wir nur die letzten beiden Fälle von Lemma 8.2.3(a) benutzen. Mehrfache Anwendung ergibt eine neue Form für Fir_p:
Für $p = (X_0 \to u_0X_1u_1 \ldots X_{n_p}u_{n_p})$ ist

$$Fir_p(d_1,\ldots,d_{n_p}) = \bigcup_{1 \leq i < j_p} (d_i - \{\varepsilon\}) \cup \begin{cases} \{1 : u_{j_p-1}\}, & \text{falls } u_{j_p-1} \neq \varepsilon \\ d_{j_p} & \text{sonst} \end{cases},$$

wobei j_p maximal ist mit der Eigenschaft, daß $\varepsilon \in d_i$ und $u_{i-1} = \varepsilon$ für alle $(1 \leq i < j_p)$. Für $p = (X_0 \to u)$ ist $Fir_p = \{1 : u\}$ eine konstante Funktion.

Wie man sieht, hängt j_p, die Zahl der zu berücksichtigenden Parameter davon ab, in wievielen der ersten d_i ε enthalten ist. In unserem Fall sind die d_i $FIRST_1$-Mengen der Nichtterminale der rechten Seite. Wir spalten jetzt die Berechnung in zwei Phasen auf: In der ersten Phase wird berechnet, ob ein Nichtterminal ε produziert, also ε in seiner $FIRST_1$-Menge hat. In der zweiten Phase wird dann der ε-freie Rest der $FIRST_1$-Mengen berechnet.

Die erste Phase besteht in der Lösung des bottom up-GFA-Problems ε-PROD. ε-PROD beschreibt das Problem herauszufinden, welche Nichtterminale ε produzieren. Es stellt sich heraus, daß ε-PROD sehr ähnlich zu PRODUKTIV ist. Ein Nichtterminal ist **ε-produktiv**, wenn es linke Seite einer ε-Produktion ist, oder wenn es linke Seite einer Produktion ohne Terminale ist, in der alle Nichtterminale der rechten Seite ε-produktiv sind. Abbildung 8.11 beschreibt den GFA-Rahmen ε-PROD. Der zugrundeliegende Verband ($\{true, false\}, \sqsubseteq$, $false$, $true$, $\vee$, $\wedge$) hat diesmal sogar besonders kurze Ketten, nämlich die eine Kette $false \sqsubseteq true$. Die Transferfunktionen, nämlich die Konjunktion in $\mathcal{E}ps$ sind natürlich monoton, und die Kombinationsfuktion, die Disjunktion, ist monoton; auch für die beiden nullstelligen Konstantenfunktionen c_{true} und c_{false} gilt dies.

bottom up-GFA-Problem ε-PROD	
H-Verband	$(\{true, false\}, \{false \sqsubseteq true\}, false, \vee)$
Transf. Fkt.	$Eps_p(d_1,\ldots,d_{n_p}) = \bigwedge_{i=1}^{n_p} d_i,$
	falls $p = (X_0 \to X_1 \ldots X_{n_p}),\ n_p \geq 1,\ X_i \in V_N$ für alle $1 \leq i \leq n_p$
	$Eps_p = \begin{cases} c_{true}, & \text{falls } p = (X \to \varepsilon) \\ c_{false}, & \text{falls } p = (X \to \alpha), \alpha \notin V_N^* \end{cases}$
Komb. Fkt.	$\vee$

Das rekursive Gleichungssystem für ε-PROD ist

$$eps(X) = \bigvee_{\{p \mid p[0]=X\}} Eps_p(eps(p[1]),\ldots,eps(p[n_p]))\ \forall X \in V_N \qquad (eps)$$

Abb. 8.11: GFA-Problem und rekursives Gleichungssystem (eps)

Einen effizienten Algorithmus zur Lösung von (eps) erhält man aus einer einfachen Modifikation des Algorithmus PRODUKTIV.

Nach der ersten Phase wissen wir, welche Nichtterminale ε produzieren. Damit können wir anstelle unserer $FIRST_1$-Funktion die ε-freie first-Funktion ε-ffi berechnen. ε-$ffi(X)$ ist identisch zu $FIRST_1(X)$, außer daß das in $FIRST_1$ eventuell enthaltene ε fehlt.

ε-$ffi(X) = \bigcup\{\varepsilon$-$ffi(Y) | X \to \alpha Y \beta \in P, Y \in (V_N \cup V_T)$ und $\alpha \stackrel{*}{\Longrightarrow} \varepsilon\}$,
dazu muß noch für $a \in V_T$ ε-$ffi(a) = \{a\}$ gesetzt werden.

Im folgenden Abschnitt wird ein Verfahren vorgestellt, das Probleme, welche die speziellen Eigenschaften von $FIRST_1$ und $FOLLOW_1$ haben, sehr effizient löst. Nach der Beschreibung des Verfahrens wird es auf $FIRST_1$ und $FOLLOW_1$ angewendet.

8.2.8 Reine Vereinigungsprobleme

Sei $R \subseteq A \times A$ eine Relation auf einer Menge A. Sei f eine mengenwertige Funktion mit der Eigenschaft, daß für alle $a \in A$ gilt

$$f(a) = g(a) \cup \bigcup\{f(b) \mid a\ R\ b\} \tag{8.1}$$

wobei g eine bekannte mengenwertige Funktion ist. Die Relation R drückt aus, daß b zu der für a zu berechnenden Menge beiträgt, wenn $a\ R\ b$ gilt. Sei $G_R = (A, E)$ der von R induzierte gerichtete Graph, d.h. $a \to b \in E$ genau dann, wenn $a\ R\ b$ gilt. Dann kann f durch geschicktes Durchlaufen von G_R berechnet werden. Für die nichtzyklischen Teile von G_R ist das einfach zu sehen. Man durchläuft sie rekursiv und berechnet f für ein Argument a erst, wenn es für alle Nachfolger von a bereits berechnet ist.

Wie behandelt man die Berechnung von f für Zyklen von G_R? Betrachten wir eine Folge $a_1, \ldots, a_n$ mit $a_i\ R\ a_{i+1}$ für $1 \le i < n$. Aus Gleichung 8.1 folgt $f(a_1) \supseteq f(a_2) \supseteq \ldots \supseteq f(a_n)$ und $f(a_1) \supseteq \bigcup_{1 \le i \le n} g(a_i)$.
Jetzt schließen wir einen Zyklus bezüglich R, indem wir $a_n\ R\ a_1$ hinzunehmen. Dann muß offensichtlich zusätzlich $f(a_n) \supseteq f(a_1)$ gelten. Zusammen ergibt sich, daß f auf allen Knoten des Zyklus den gleichen Wert hat. Deshalb bietet sich eine Vorgehensweise an, die auf der Kenntnis der starken Zusammenhangskomponenten von G_R beruht.

Definition 8.2.20 (stark zusammenhängend)
Sei $G = (A, E)$ ein gerichteter Graph. Zwei Knoten $a, b \in A$ heißen **stark zusammenhängend**, wenn es einen gerichteten Weg von a nach b und einen von b nach a gibt. Diese Eigenschaft definiert eine Äquivalenzrelation auf A. Die Klassen dieser Relation heißen die **starken Zusammenhangskomponenten** (SZK) von G. Eine SZK, die nur aus einem Knoten a ohne Kante von a nach a besteht, heißt eine **triviale** SZK. □

Starke Zusammenhangskomponenten sind Verallgemeinerungen von Zyklen. Es gilt ebenfalls: sei $S \subseteq A$ eine SZK, $S = \{a_1, \ldots, a_n\}$, dann ist $f(a_1) = \ldots = f(a_n)$ und $\bigcup_{1 \le i \le n} g(a_i) \subseteq f(a_1)$.

Deshalb kann der Graph G_R folgendermaßen in einen Graphen G'_R transformiert werden:

Ersetze jede SZK S durch einen neuen Knoten s mit $g(s) = \bigcup_{a \in S} g(a)$. Man sagt, S wird kollabiert zu s. Übernehme alle Kanten aus S heraus bzw. in S hinein als Kanten mit Anfangspunkt bzw. Endpunkt s. Der entstehende Graph ist azyklisch. Somit kann f mit dem oben geschilderten Verfahren für G'_R berechnet

werden. Anschließend wird für alle Knoten a einer SZK S, die zu einem Knoten s kollabiert wurde, $f(a) = f(s)$ gesetzt.

Dieses Verfahren muß erst die starken Zusammenhangskomponenten des Graphen G_R finden. Es zeigt sich, daß der dazu üblicherweise verwendete Algorithmus, die Tiefensuche, so modifiziert werden kann, daß die Berechnung von f parallel zu dieser rekursiven Durchmusterung des Graphen vorgenommen werden kann. Die oben geschilderte Reduktion des Graphen durch Ersetzen der SZK ist also gar nicht nötig. Der Algorithmus SZK enthält eine Prozedur *dfs*, die rekursiv die von einem Knoten erreichbaren Teilgraphen durchläuft und bei jedem erreichten Knoten prüft, ob er bereits einmal erreicht worden ist. Dann wurde nämlich ein Zyklus entdeckt.

Algorithmus SZK
Eingabe: gerichteter Graph $G = (A, E)$, g mengenwertige Funktion auf A.
Ausgabe: mengenwertige Funktion f, die die Gleichung 8.1 erfüllt.
Methode: G wird mithilfe eines Kellers durchlaufen; dabei werden die starken Zusammenhangskomponenten gefunden; parallel dazu wird f berechnet.

```
S: stack of A init empty;
N: array [A] of integer init 0;
foreach a ∈ A do if N[a] = 0 then dfs(a) od;
proc dfs(a : A):
  push (S, a);
  let d = depth(S) in
    N[a] := d;                          (* anfänglich: Kellertiefe *)
    f(a) := g(a);                       (* (1) *)
    foreach b ∈ A where a → b ∈ E do
      if N[b] = 0 then dfs(b) fi;       (* Knoten noch unbesucht *)
      N[a] := min(N[a], N[b]);          (* N[b] < N[a] : *)
                                        (* Zyklus gefunden *)
      f(a) := f(a) ∪ f(b)               (* (2) *)
    od;
    if N[a] = d                         (* a Eintrittspunkt einer *)
                                        (* erledigten Komponente *)
    then
      repeat N[top(S)] := ∞;            (* Knoten erledigt *)
        f(top(S)) := f(a);              (* (3) *)
      until pop(S) = a
    fi
  tel
end dfs;
```

Etwas eingerückt stehen die Anweisungen (1), (2) und (3) zur Berechnung von f. Für jeden Knoten a aus A wird genau einmal *dfs* aufgerufen. Dabei wird

in (1) der Wert von f für a mit dem von g initialisiert. Jeweils nach der Rückkehr von einem Nachfolger b von a wird in (2) dessen f-Wert zu dem bisherigen Wert von $f(a)$ hinzuvereinigt. In der *repeat*-Schleife wird durch die Anweisung (3) der für den Eintrittspunkt a einer starken Zusammenhangskomponente gefundene Wert $f(a)$ den anderen Knoten in der SZK zugewiesen.

Die Komplexität des Algorithmus SZK ergibt sich folgendermaßen: insgesamt $|A|$ Aufrufe von *dfs* und $O(|A|)$ *push*-, *pop*- und *top*- Operationen und Berechnungen von f. $|E|$-maliges Durchlaufen der *foreach*-Schleife. Insgesamt ergibt sich die Zeitkomplexität $O(|A| + |E|)$.

Anwendungen auf $FIRST_1$ und $FOLLOW_1$

Jetzt werden wir zum Ausgangspunkt, nämlich der Berechnung der $FIRST_1$- und $FOLLOW_1$-Funktionen, zurückkehren. Für diese ist zu zeigen, wie sie in das Schema $f(X) = g(X) \cup \bigcup\{f(Y) \mid Y \in V_N \text{ und } X\ R\ Y\},\ X \in V_N$ hineinpassen. Dabei ist g eine anzugebende Funktion von V_N in 2^{V_T}.

Beginnen wir mit dem $FIRST_1$-Problem. Das f aus dem obigen Schema ist die Funktion ε-*ffi*. Die Relation R_{Fi} ist folgendermaßen definiert:

$$X\ R_{Fi}\ Y, \text{ gdw. es} X \to \alpha Y \beta \in P \text{gibt mit} \alpha \stackrel{*}{\Longrightarrow} \varepsilon.$$

Ein in einer Produktion für X rechts auftretendes Nichtterminal Y trägt zur $FIRST_1$-Menge von X bei, wenn alles, was links von Y steht, ε ist oder ε produziert. $g_{Fi}(X)$ ist die Menge aller Terminalsymbole an „Anfängen" von Produktionen für X. Die Funktion g_{Fi} ergibt sich aus der Grammatik als

$$g_{Fi}(X) = \{a | X \to \alpha a \beta \in P \text{ und } \alpha \stackrel{*}{\Longrightarrow} \varepsilon \text{ und } a \in V_T\}.$$

Nun formulieren wir $FOLLOW_1$ so, daß es in das obige Schema paßt. Die Funktion f des Schemas ist die Funktion Fo_1. Die Relation R_{Fo} ist definiert durch:

$$Y\ R_{Fo}\ X, \qquad \text{gdw. es } X \to \alpha Y \beta \in P \text{ gibt mit } \beta \stackrel{*}{\Longrightarrow} \varepsilon$$

Die linke Seite X trägt zur $FOLLOW_1$-Menge des auf der rechten Seite auftretenden Y genau dann bei, wenn alles, was rechts von Y steht, ε ist oder ε produziert. $g_{Fo}(Y)$ faßt alle möglichen Folgesymbole für Y der verschiedenen rechten Seiten, in denen Y auftritt, zusammen. Die Funktion g_{Fo} ist deshalb

$$\begin{aligned} g_{Fo}(Y) &= \bigcup\{\varepsilon\text{-}ffi(\gamma) | X \to \alpha Y \beta \gamma \in P \text{ und } \beta \stackrel{*}{\Longrightarrow} \varepsilon\}. \\ g_{Fo}(S) &= \{\#\} \end{aligned}$$

Zur Formulierung der $FIRST_1$- und $FOLLOW_1$-Berechnung verallgemeinern wir *eps* auf $(V_N \cup V_T)^*$:

$$\begin{aligned} eps(a) &= \textit{false} \text{ für } a \in V_T \\ eps(X_1 \ldots X_n) &= \bigwedge_{1 \le i \le n} eps(X_i), X_i \in V_T \cup V_N (1 \le i \le n) \end{aligned}$$

In der Definition von R_{Fi}, R_{Fo} und g_{Fi}, g_{Fo} setzten wir dann statt $\alpha \stackrel{*}{\Longrightarrow} \varepsilon$ bzw. $\beta \stackrel{*}{\Longrightarrow} \varepsilon$ die Werte $eps(\alpha) = \textit{true}$ bzw. $eps(\beta) = \textit{true}$ ein.

Beispiel 8.2.21
Wir betrachten wieder die Grammatik aus Beispiel 8.2.18

S	$\to$	E	E'	$\to$	$+E$	T'	$\to$	$*T$
E	$\to$	TE'	T	$\to$	FT'	F	$\to$	(E)
E'	$\to$	ε	T'	$\to$	ε	F	$\to$	**id**

Zuerst berechnen wir das zugehörige ε-PROD-Problem. Das Ergebnis ist:
$eps(E') = eps(T') = \textit{true},\ \ eps(S) = eps(E) = eps(T) = eps(F) = \textit{false}$
Die Relation R_{Fi} ergibt sich in diesem Beispiel zu:
$(S,E),\ (E,T),\ (T,F)$
Die Funktion g_{Fi} sieht folgendermaßen aus:

$$\begin{array}{lll} S \mapsto \emptyset & E \mapsto \emptyset & T \mapsto \emptyset \\ E' \mapsto \{+\} & T' \mapsto \{*\} & F \mapsto \{(, \mathbf{id}\} \end{array}$$

Der von R_{Fi} induzierte gerichtete Graph ist der folgende:

$$S \to E \to T \to F \qquad E' \qquad T'$$

Die Berechnung der Funktion Fi mittels Algorithmus SZK geschieht wie folgt:

Keller S	Fkt.	Berechnung der Funktion Fi					Anweisung
	S	E	T	F	E'	T'	
$S\|E\|T\|F$	$\emptyset$	$\emptyset$	$\emptyset$	{(, id}	$\emptyset$	$\emptyset$	(1)
$S\|E\|T$	$\emptyset$	$\emptyset$	{(, id}	{(, id}	$\emptyset$	$\emptyset$	(2)
$S\|E$	$\emptyset$	{(, id}	{(, id}	{(, id}	$\emptyset$	$\emptyset$	(2)
S	{(, id}	{(, id}	{(, id}	{(, id}	$\emptyset$	$\emptyset$	(2)
E'	{(, id}	{(, id}	{(, id}	{(, id}	{+}	$\emptyset$	(1)
T'	{(, id}	{(, id}	{(, id}	{(, id}	{+}	{*}	(1)

□

Beispiel 8.2.22 (Fortführung von Beispiel 8.2.21)
Die Relation R_{Fo} ergibt sich für diese Grammatik zu:
$(E',E),\ (T',T),\ (F,T),\ (T,E),\ (T,T'),\ (E,E')$

Der von R_{Fo} induzierte gerichtete Graph ist also:

$$\begin{array}{lcll} & E' & \leftrightarrows & E \\ S & & & \uparrow \\ & T' & \leftrightarrows & T \quad \leftarrow F \end{array}$$

Die Funktion g_{Fo} für diese Grammatik ist: