Compilerbau: Kontextfreie Grammatiken und Syntaxanalyse

SS 2019

Michael Jäger

8. Mai 2019

Grammatiken, Sprachen und Kellerautomaten

Kontextfreie Grammatiken

- formale Beschreibungsmöglichkeit für die Syntax von Programmiersprachen (und natürlichen Sprachen)
- Kellerautomaten akzeptieren kontextfreie Sprachen.
- Die **Syntaxanalyse** (Parser)
  - basiert auf Kellerautomaten
  - prüft ob Programmquelltext der kontextfreien Grammatik entspricht

Definition

Eine kontextfreie Grammatik ist ein 4-Tupel $G = (V, \Sigma, R, S)$ mit folgenden Bestandteilen:

1. V ist die endliche Menge der Variablen (oder Nonterminalsymbole)
2. Σ ist die endliche Menge der Terminalsymbole, $\Sigma \cap V = \emptyset$.
3. R ist die endliche Menge der Ableitungsregeln (oder Ersetzungsregeln), $R \subset (V \times (V \cup \Sigma)^*$
4. $S \in V$ ist das Startsymbol.

Die Notation für eine Ableitungsregel ist

$$X \to w$$

wobei $X \in V, w \in (V \cup \Sigma)^*$. X heißt linke Regelseite, w rechte Regelseite.

Beispiel

$G = (\{S\}, \{0, 1\}, R, S)$, R besteht aus den beiden Regeln

$$\begin{aligned} S &\rightarrow 0S1 \\ S &\rightarrow \varepsilon \end{aligned}$$

Ableitungen

Mit den Ableitungsregeln lassen sich Ableitungen konstruieren, z.B.

$$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000S111 \Rightarrow 000111$$

Ausgehend vom Startsymbol wird in jedem Ableitungsschritt eine Ersetzung der linken Regelseite einer Regel durch die rechte Seite der Regel durchgeführt. Die Ableitung endet, wenn keine Variablen mehr vorhanden sind.

Regelformen

Besondere Formen von Ableitungsregeln

- Eine Regel der Form $X \to \varepsilon$ heißt ε-Regel.
- Eine Regel der Form $X \to Xw$ heißt direkt linksrekursiv.
- Eine Regel der Form $X \to wX$ heißt direkt rechtsrekursiv.

Anmerkungen

- Neben direkter Linksrekursion sind indirekte Rekursionformen möglich, z.B. $X \to Yw$, $Y \to X$
- Linksrekursive Regeln bereiten manchen Syntaxanalyseverfahren Probleme

Notation

Verzicht auf die explizite Definition von N, Σ, S

- Wir geben meist nur die Regeln an
- Nonterminalsymbole sind dann alle Symbole, die auf linken Regelseiten auftauchen
- Startsymbol ist die linke Seite der ersten Regel
- Terminalsymbole sind alle anderen in den Regeln vorkommenden Symbole

Notation

Nummerierung der Regeln

Bei nummerierten Regeln kann zur besseren Nachvollziehbarkeit für jeden Schritt einer Ableitung die Nummer der verwendeten Regel angeben und ggf. noch die zu ersetzende Variable unterstreichen.
Beispiel:

$$1.\ S \to 0S1$$
$$2.\ S \to \varepsilon$$

$$\underline{S} \overset{1}{\Rightarrow} 0\underline{S}1 \overset{1}{\Rightarrow} 00\underline{S}11 \overset{1}{\Rightarrow} 000\underline{S}111 \overset{2}{\Rightarrow} 000111$$

Notation

Mehrere Regeln mit gleicher linker Seite

- Falls die Regelmenge für eine Variable X mehrere Regeln enthält

$$\begin{aligned} X &\to w_1 \\ X &\to w_2 \\ &\dots \\ X &\to w_n \end{aligned}$$

schreiben wir auch

$$X \to w_1 \mid w_2 \mid \cdots \mid w_n$$

Beispiel: Grammatik für eine Programmiersprache

Zwischenräume dienen zur Abgrenzung der Symbole,
Terminalsymbole sind durch einfache Apostrophe gekennzeichnet:

```
program              →   globalDefinitions
globalDefinitions    →   ε  | globalDefinitions globalDefinition
globalDefinition     →   variableDef | functionDef | typeDef
variableDef          →   identifier ':' type optInitialiser
optInitialiser       →   ε  | initialiser
functionDef          →   functionHeader functionBody
                     ...
```

Beispiel: SPL-Grammatik / Notation für cup-Generator

```
                ...
type_declaration ::= TYPE IDENT EQ type SEMIC
                   ;
type             ::= IDENT
                   |   ARRAY LBRACK INTLIT RBRACK OF type
                   ;
                ...
var_declarations ::= /* empty */
                   |   var_declaration var_declarations
                   ;

var_declaration  ::= VAR IDENT COLON type SEMIC
                   ;
                ...
```

Formale Definition der Ableitbarkeit

Definition

Sei $G = (V, \Sigma, R, S)$ eine Grammatik, $u, x, y, w \in (V \cup \Sigma)^*$.
Aus uxw ist uyw direkt ableitbar (Notation: $uxw \Rightarrow uyw$), wenn $(x \to y) \in R$.
Aus uxw ist uyw ableitbar (Notation: $uxw \overset{*}{\Rightarrow} uyw$), wenn für ein $n \geq 0$ Wörter $v_0, \ldots, v_n$ existieren, so dass

$$uxw = uv_0w \Rightarrow uv_1w \ldots \Rightarrow uv_nw = uyw$$

(Für $n = 0$ ist $x = y$.)
Die Anzahl der Ableitungsschritte n ist die Länge der Ableitung.

Die Ableitbarkeitsrelation ist demzufolge der transitive und reflexive Abschluss der direkten Ableitbarkeit.

Satzform und Sprache

Definition

- Sei $G = (V, \Sigma, R, S)$ eine kontextfreie Grammatik. Ein Wort $w \in (V \cup \Sigma)^*$, das sich aus S ableiten lässt, $S \overset{*}{\Rightarrow} w$, heißt Satzform (zu G).
- Die von G erzeugte Sprache ist die Menge aller nur aus Terminalsymbolen bestehenden Satzformen:

$$L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$$

- Zwei Grammatiken G_1 und G_2 heißen äquivalent, wenn $L(G_1) = L(G_2)$.

Beispiel

Die von der folgenden Grammatik G

$$\begin{aligned} S &\rightarrow 0S1 \\ S &\rightarrow \varepsilon \end{aligned}$$

definierte Sprache ist

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

Dies sieht man leicht aus der Form der möglichen Ableitungen: Jede Ableitung der Länge $k, n \geq 1$, besteht aus $k-1$ Anwendungen der 1. Regel gefolgt von einer Anwendung der 2. Regel:

$$S \overset{1}{\Rightarrow} 0S1 \overset{1}{\Rightarrow} 00S11 \overset{1}{\Rightarrow} \ldots \overset{1}{\Rightarrow} 0^{k-1}S1^{k-1} \overset{2}{\Rightarrow} 0^{k-1}1^{k-1}$$

Reihenfolge der Ableitungsschritte

Falls in einer Satzform mehrere Variablen vorkommen, ist die Reihenfolge der Ableitungsschritte nicht mehr eindeutig festgelegt.

Kanonische Ableitungen

Eine Ableitung, bei der in jedem Ersetzungsschritt die äußerst linke (rechte) Variable ersetzt wird, heißt Linksableitung (bzw. Rechtsableitung).

Ableitungsbaum

Ein Ableitungsbaum ist ein geordneter Baum, der aus einer Ableitung eines Worts w bezüglich einer Grammatik $G = (V, \Sigma, R, S)$ wie folgt konstruiert wird:

- Der Wurzelknoten wird mit dem Startsymbol S markiert
- Zu jeder in der Ableitung angewandten Regel $X \to x_1 ... x_n$ werden dem mit X markierten Knoten im Baum, der die ersetzte Variable repräsentiert, n neue, mit $x_1, ..., x_n$ markierte Knoten als Nachfolgeknoten zugeordnet.

Eigenschaften

- Die inneren Knoten des Baums sind mit Variablen markiert.
- Die Blätter sind mit Terminalsymbolen markiert.
- Die Blätter ergeben von links nach rechts angeordnet das abgeleitete Wort w.

Beispiel

Betrachte die Grammatik G:

1. $A \to aBC$
2. $B \to Bb$
3. $B \to \varepsilon$
4. $C \to c$

Das Wort $w = abbc$ liegt in $L(G)$. Die Linksableitung für w ist

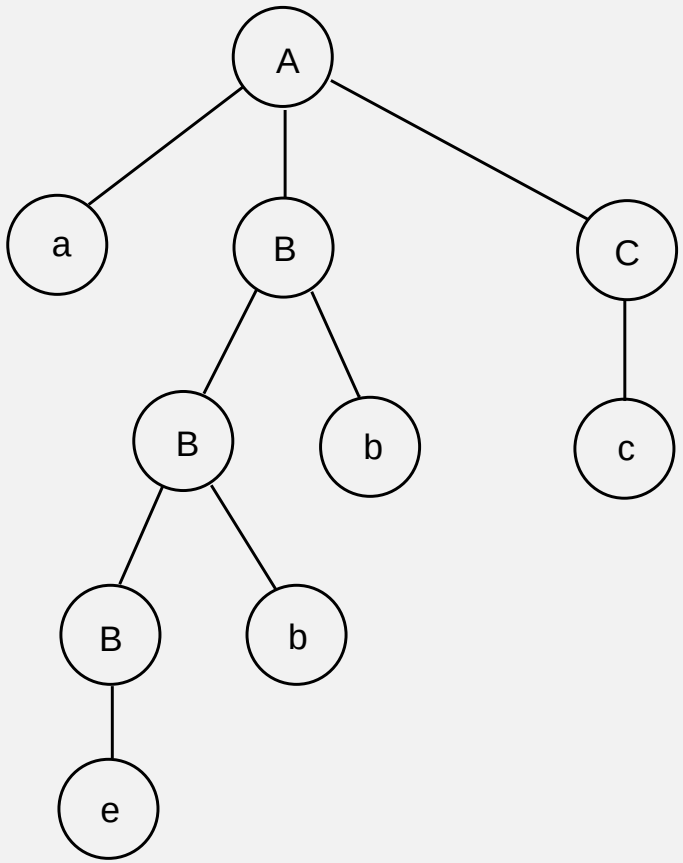
$$A \overset{1}{\Rightarrow} aBC \overset{2}{\Rightarrow} aBbC \overset{2}{\Rightarrow} aBbbC \overset{3}{\Rightarrow} abbC \overset{4}{\Rightarrow} abbc$$

Die Rechtsableitung:

$$A \overset{1}{\Rightarrow} aBC \overset{4}{\Rightarrow} aBc \overset{2}{\Rightarrow} aBbc \overset{2}{\Rightarrow} aBbbc \overset{3}{\Rightarrow} abbc$$

Ableitungsbaum zum Beispiel

Der (eindeutig bestimmte) Ableitungsbaum:



Mehrdeutigkeit

Definition

- Ein Wort w heißt mehrdeutig ableitbar bezüglich einer Grammatik G, wenn es für w mehrere Linksableitungen gibt.
- Eine Grammatik G heißt mehrdeutig, wenn ein Wort $w \in L(G)$ mehrdeutig ableitbar ist.
- Eine Grammatik G heißt inhärent mehrdeutig, wenn es keine zu G äquivalente nicht mehrdeutige Grammatik gibt.

Beispiel:

Die folgende Grammatik G mit $L(G) = \{a^n \mid n \geq 0\}$ ist mehrdeutig

$$S \to SS \mid a \mid \varepsilon$$

Mit der rekursiven Regel $S \to SS$ lassen sich beliebig viele S-Variablen erzeugen, die mit der ε-Regel dann wieder gelöscht werden können. Daher gibt es zu jedem Wort der Sprache beliebig viele Linksableitungen, z.B.

$$\underline{S} \Rightarrow \underline{S}S \Rightarrow \underline{S}SS \Rightarrow \underline{S}S \Rightarrow \underline{S} \Rightarrow a$$

Eine äquivalente nicht mehrdeutige Grammatik ist

$$S \to Sa \mid \varepsilon$$

Tabellen-gesteuerter LL(1)-Parser

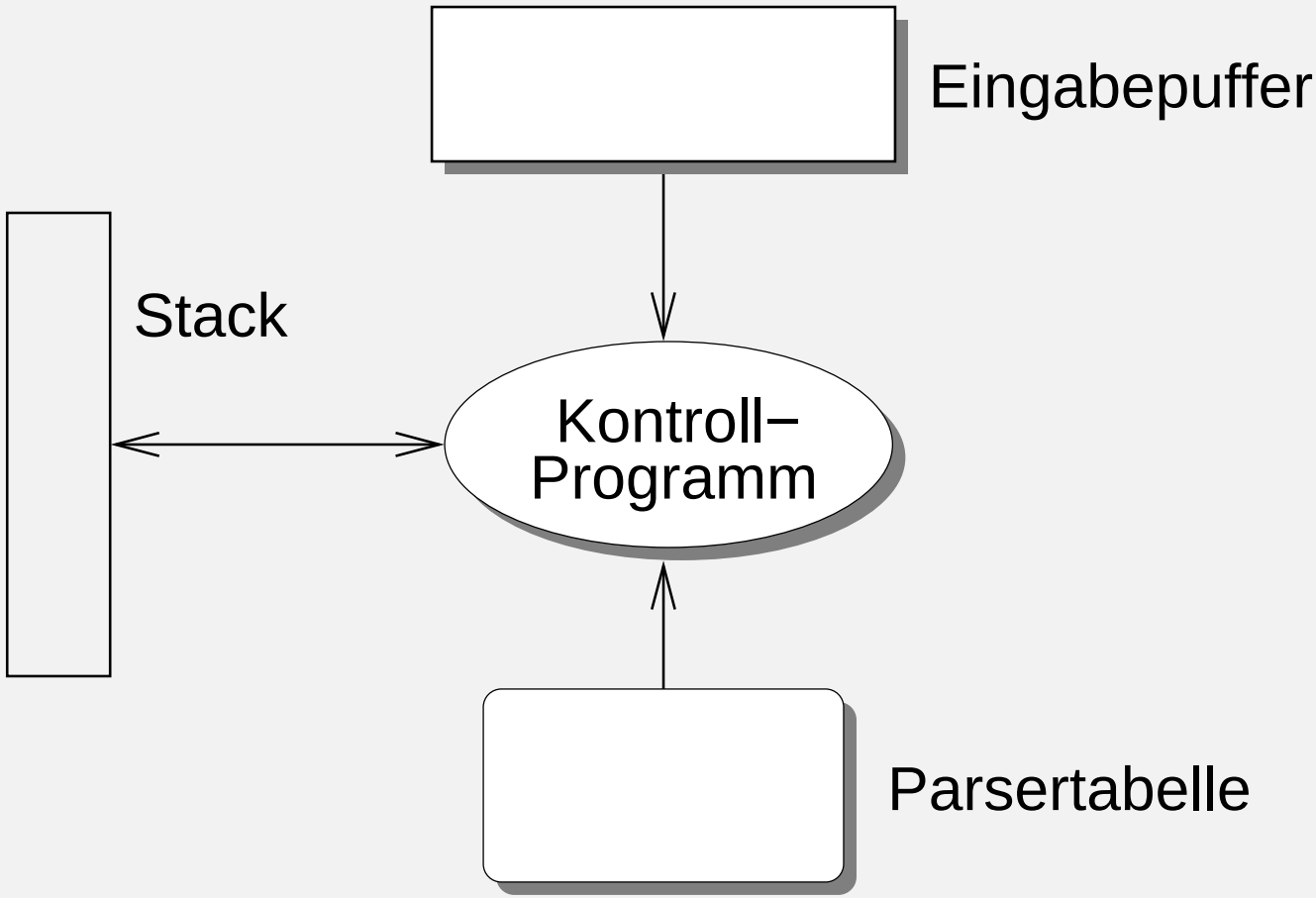
Top-Down-Analyse

- Ableitung von links nach rechts konstruieren
- Ableitungsbaum von der Wurzel ausgehend berechnen

Arbeitsweise

- Konstruktion einer Linksableitung vom Startsymbol ausgehend
- Verarbeitung der Eingabe von links nach rechts
- Satzform-Suffixe im Keller repräsentieren restliche Eingabe

Modell eines tabellengesteuerten LL(k)-Parsers



Algorithmus des Kontrollprogramms

- Die Grammatik sei $G = (V, \Sigma, R, S)$.
- In der Tabelle TAB steht zu dem gerade bearbeiteten Nonterminal $X \in V$ und dem aktuellen Eingabesymbol $a \in \Sigma$ die zu verfolgende Ableitungsregel aus R:
  $TAB[X, a] = X \to r$
  (oder, falls es diese nicht gibt, eine Fehlermarkierung *error*)
- X sei immer das oberste Keller-Element

Kontrollprogramm als Pseudocode

push(S) /* Startsymbol S auf Keller */
a:=nexttoken /* Eingabesymbol := erstes Symbol */
repeat
    if *Keller leer* **and** a=Eingabeende
    then akzeptiere
    elseif X = a **then** /* "Match"eines Terminalsymbols */
        pop
        a:=nexttoken
    elseif $X \in V$ **and** $TAB[X, a] = (X \to Y_1 Y_2 \ldots Y_k)$
    then
        print($X \to Y_1 Y_2 \ldots Y_k$) /* Ausgabe Ableitungsschritt*/
        pop(X)
        push Y_k, ..., push Y_2, push Y_1 /*Y_1 wird als nächstes bearbeitet */
    else
        Fehlerbehandlung
    end if
end repeat

Voraussetzung: LL(1)-Eigenschaft der Grammatik

Informale Definition

- LL(1)-Grammatik: Ermöglicht deterministische Regelauswahl abhängig vom nächsten Eingabesymbol
- LL(k)-Grammatik: Ermöglicht deterministische Regelauswahl abhängig von den nächsten k Eingabesymbolen

Beispiel

Folgende Grammatik ist LL(2) nicht aber LL(1)

$$S \rightarrow abb|acd$$

LL(1)-Transformation

Durch Linksfaktorisierung ergibt sich äquivalente LL(1)-Grammatik

$$S \rightarrow aS',\ S' \rightarrow bb|cd$$

Konstruktion der LL(1)-Parsertabelle

Ziel

Zur Grammatik $G = (\Sigma, N, P, S)$ wird die LL(1)-Parsertabelle bestimmt, in der zu jedem Nonterminalsymbol X und zu jedem Vorausschautoken a die passende Ableitungsregel

$$X \to w$$

steht. Falls keine solche Regel existiert, steht in der Tabelle ein Fehlereintrag.

Vorausschautoken und Regelauswahl

Problem

Parser soll für X die korrekte Ableitungsregel $X \rightarrow w$ bestimmen.

Auswahlkriterium

Vorausschautoken T (nächstes Terminalsymbol im Eingabestrom) bestimmt die zu verwendende Regel

Fallunterscheidung nötig

1. X ist nicht leer, X beginnt mit T

   formal: $T \epsilon FIRST(X)$
2. X ist leer, T folgt auf X

   formal: $T \epsilon FOLLOW(X)$

Fall 1: Anfangssymbole in der Vorausschau

X nicht leer

Wenn mit der Ableitungsregel $X \to w$ ein **nicht leeres** Wort w erzeugt wird, dann ist das Vorausschautoken das erste Symbol von w. Die Ableitungsregel passt also für alle Terminalsymbole mit denen w beginnen kann ($FIRST(w)$).

Beispiel

Regelauswahl für U:	$U \to cV$	bei Vorausschau c
	$U \to de$	bei Vorausschau d
Regelauswahl für V:	$V \to aY$	bei Vorausschau a
	$V \to bZ$	bei Vorausschau b
	$V \to U$	bei Vorausschau c oder d

Beachte: Die Regelauswahl für V hängt von den Regeln für U ab.

Definition: *FIRST*-Mengen

Informal

Zu jedem Wort $w \in (\Sigma \cup V)^*$ enthält $FIRST(w)$ alle Terminalsymbole, mit denen w beginnen kann. Zusätzlich enthält $FIRST(w)$ das leere Wort, wenn w gelöscht werden kann.

Formal

$$FIRST(w) = \begin{cases} \{x \in \Sigma \mid w \overset{*}{\Rightarrow} xu, \text{für ein } u \in (\Sigma \cup V)^*\} \cup \{\varepsilon\}, & \text{falls } w \overset{*}{\Rightarrow} \varepsilon \\ \{x \in \Sigma \mid w \overset{*}{\Rightarrow} xu, \text{für ein } u \in (\Sigma \cup V)^*\}, & \text{sonst} \end{cases}$$

Verwendung für die Regelauswahl

- Die Regel $X \to w$ wird gewählt, wenn das Vorausschautoken aus $FIRST(w)$ ist.

Auszug aus der Parsertabelle:

Nonterminal-Symbol	Vorausschau				
	a	b	c	d	e
U	Fehler	Fehler	U→cV	U→de	Fehler
V	V→aY	V→bZ	V→U	V→U	Fehler

Fall 2: Folgesymbole in der Vorausschau

Szenario

- $X = \varepsilon$, mit der auszuwählenden Ableitungsregel $X \to w$ wird das leere Wort erzeugt: $X \Rightarrow w \overset{*}{\Rightarrow} \varepsilon$.
- Das Vorausschausymbol gehört nicht zu X, sondern zu einem auf X folgenden anderen Bestandteil der Eingabe.

FOLLOW-Menge

- Ein *Folgesymbol* von X ist ein Terminalsymbol, das in der Eingabe direkt hinter X steht.

  $FOLLOW(X)$ ist die Menge der möglichen Folgesymbole von X

Formal

- $FOLLOW(X) = \{t \in \Sigma \mid S \overset{*}{\Rightarrow} uXtv \text{ für irgendwelche } u, v \in (\Sigma \cup V)^*\}$
- Das Eingabeende wird durch eine spezielles Token \$ repräsentiert. $\$ \in FOLLOW(X)$, wenn X am Ende der Eingabe auftreten kann, also $S \overset{*}{\Rightarrow} uX$ für irgendein $u \in (\Sigma \cup V)^*$.

Verwendung für die Regelauswahl:

$X \to w$ wird ausgewählt, wenn $\varepsilon \in FIRST(w)$ und das Vorausschausymbol aus $FOLLOW(X)$ ist.

$FIRST(X)$ für ein Symbol X bestimmen

Sei $X \in V \cup \Sigma$.

1. Falls $X \in \Sigma$, $FIRST(X) = X$
2. Falls $(X \to \varepsilon) \in P$, übernehme ε in $FIRST(X)$
3. Für jede Regel $(X \to Y_1 \dots Y_k) \in P$ verfahre wie folgt:
   Falls $\varepsilon \in FIRST(Y_1) \wedge \dots \wedge \varepsilon \in FIRST(Y_{i-1})$ für ein i, $1 \leq i < k$, übernehme alle Terminalzeichen aus $FIRST(Y_i)$ in $FIRST(X)$
   Falls $\varepsilon \in FIRST(Y_1) \wedge \dots \wedge \varepsilon \in FIRST(Y_k)$ übernehme ε in $FIRST(X)$

(2) und (3) müssen iteriert werden, bis sich an den $FIRST$-Mengen nichts mehr ändert

$FIRST(w)$ für ein Wort w bestimmen

Sei $w \in (V \cup \Sigma)^*$.

- $w = \epsilon$
  Dann ist $FIRST(w) = \epsilon$
- w beginnt mit einem Terminalsymbol, also $w = xv, x \in \Sigma$
  $FIRST(w) = FIRST(x) = \{x\}$

- w beginnt mit einem Nonterminalsymbol, also
  $w = Xv, X \in V$
  Fallunterscheidung: Kann X leer sein?
  - Wenn nicht, beginnt Xv mit einem Symbol aus $FIRST(X)$ und v ist nicht weiter von Interesse.
  - Falls X leer sein kann, lässt sich aus w über Xv jedoch auch v ableiten, so das $FIRST(v)$ auch bestimmt werden muss.

  ϵ ist aus X genau dann ableitbar, wenn $\epsilon \in FIRST(X)$.
  Es gilt also:

$$FIRST(Xv) = \begin{cases} FIRST(X) & \text{falls } \epsilon \notin FIRST(X) \\ FIRST(X) \backslash \{\epsilon\} \cup FIRST(v) & \text{falls } \epsilon \in FIRST(X) \end{cases}$$

Bestimmung der Folgesymbolmengen

Vorüberlegung

Um $FOLLOW(X)$ für ein $X \in V$ zu bestimmen, betrachtet man jedes Auftreten von X auf einer *rechten* Regelseite $Y \to uXv$. Von v hängt es ab, welche Terminalsymbole hinter X stehen können.

Beispiel

Aus der Betrachtung der Regel

$$U \to aYbZcYd$$

ergibt sich beispielsweise:

- $b \in FOLLOW(Y)$
- $c \in FOLLOW(Z)$
- $d \in FOLLOW(Y)$

Algorithmus

1. Übernehme das Eingabeende-Token \$ in $FOLLOW(S)$
   Das Startsymbol S der Grammatik steht im ersten Ableitungsschritt für die gesamte Eingabe, dahinter kommt das Eingabeende.
2. Für jedes Auftreten eines Nonterminalsymbols X in einer Regel $(A \to uXw) \in P$ übernehme alle Terminalzeichen aus $FIRST(w)$ in $FOLLOW(X)$.
3. Für jede Regel $(A \to uB) \in P$ sowie jede Regel $(A \to uBv) \in P$ mit $\varepsilon \in FIRST(v)$, übernehme alle Terminalzeichen aus $FOLLOW(A)$ in $FOLLOW(B)$.
   Die Folgezeichen von A sind Folgezeichen von B, weil B in diesem Fall der letzte Bestandteil von A ist.

(2) und (3) sind zu iterieren

Beispiel für Schritt 2:

Betrachte bei der Bestimmung von $FOLLOW(X)$ die Regel $Z \to uXYZv$: Aus der Regel ergibt sich:
Jedes Terminal, mit dem YZv beginnen kann, ist Folgesymbol von X, formal

$$(FIRST(YZv) \setminus \{\varepsilon\}) \subseteq FOLLOW(X)$$

$FIRST(YZv)$ muss bestimmt werden:

- Zunächst sind alle Terminalzeichen aus $FIRST(Y)$ darin.
- Wenn $\varepsilon \in FIRST(Y)$, kommen die Terminalsymbole aus $FIRST(Z)$ hinzu, denn aus $uXYZv$ kann $uXZv$ abgeleitet werden.
- Falls $\varepsilon \in FIRST(Y)$ **und** $\varepsilon \in FIRST(Z)$, kommt v auch noch hinzu, denn nach Löschen von Y und Z steht v direkt hinter X: $uXYZv \stackrel{*}{\Rightarrow} uXv$.

Konstruktion der Parsertabelle (*TAB*)

TAB enthält für jedes Nonterminalsymbol A eine Zeile und für jedes Terminalsymbol x eine Spalte. $TAB(A, x)$ enthält die Regel(n) für A, falls x das Vorausschau-Token ist.

Algorithmus

Konstruiere die Parsertabelle durch Ausführung der folgenden Schritte (1) und (2) für jede Ableitungsregel $A \rightarrow w$:

1. Für jedes Terminalsymbol $a \in FIRST(w)$ übernehme die Regel in $TAB[A, a]$
2. Falls $\varepsilon \in FIRST(w)$, übernehme die Regel für jedes $b \in FOLLOW(A)$ in $TAB[A, b]$.

   Falls $\epsilon \in FIRST(w)$ und $\$ \in FOLLOW(A)$, übernehme die Regel in $TAB[A, \$]$.

Alle leeren Tabelleneinträge werden mit *error* markiert.

LL(1)-Grammatiken

Definition

Eine Grammatik hat die LL(1)-Eigenschaft, g.d.w. kein Tabelleneintrag $TAB[X, a]$ mehrere Regeln enthält.

Beispiel zur Tabellenkonstruktion

Zur nachfolgenden Grammatik soll die LL(1)-Tabelle bestimmt werden. A sei das Startsymbol.

1. $A \rightarrow a$
2. $A \rightarrow BBC$
3. $B \rightarrow b$
4. $B \rightarrow \epsilon$
5. $C \rightarrow cc$

Schritt 1: Bestimme FIRST-Mengen der rechten Regelseiten

Vorüberlegung

- Nur Regel 2 erfordert eine nicht triviale Berechnung: $FIRST(BBC)$.
- Die Bestimmung von $FIRST(BBC)$ benötigt $FIRST(B)$. Falls $\epsilon \in FIRST(B)$ wird auch noch $FIRST(C)$ gebraucht.
- $FIRST(B)$ und $FIRST(C)$ sind von anderen $FIRST$-Mengen unabhängig, mit jeder der beiden Mengen kann man die Berechnung beginnen.
- $FIRST(A)$ wird für die Tabelle nicht benötigt und nur zu Übungszwecken berechnet.

FIRST-Mengen der rechten Regelseiten

$$
\begin{aligned}
FIRST(a) &= \{a\} \\
FIRST(b) &= \{b\} \\
FIRST(\epsilon) &= \{\epsilon\} \\
FIRST(B) &= \{b, \epsilon\} \\
FIRST(cc) &= \{c\} \\
FIRST(C) &= \{c\} \\
FIRST(BBC) &= \{b, c\} \\
FIRST(A) &= \{a, b, c\}
\end{aligned}
$$

Aus den FIRST-Mengen bestimmbare Tabelleneinträge

	a	b	c	$
A	$A \to a$	$A \to BBC$	$A \to BBC$	
B		$B \to b$		
C			$C \to cc$	

Bestimmung der *FOLLOW*-Mengen im Beispiel

Bestimmung der Reihenfolge

- Für die Tabelle wird nur $FOLLOW(B)$ benötigt, die anderen Mengen werden zu Übungszwecken berechnet
- $FOLLOW(C)$ benötigt $FOLLOW(A)$ weil C in Regel 2 letzter Bestandteil von A ist.
- Eine Abhängigkeit zwischen $FOLLOW(B)$ und $FOLLOW(A)$ ergibt sich dagegen aus Regel 2 nicht, weil C nicht auf ε abgeleitet werden kann. Damit ist B nicht letzter Bestandteil von A.
- Weitere Abhängigkeiten sind nicht vorhanden, da andere Nonterminalsymbole nicht als letzte Bestandteile rechter Regelseiten auftauchen.
- Wir wählen die Reihenfolge: $FOLLOW(A), FOLLOW(B), FOLLOW(C)$

Bestimmung der *FOLLOW*-Mengen:

- $FOLLOW(A) = \{\$\}$
  A ist Startsymbol und kommt in keiner Regel auf der rechten Seite vor.
- $FOLLOW(B)$ vereinigt alle Terminalzeichen aus $FIRST(B)$ (erstes Auftreten in Regel 2) und alle Terminalzeichen aus $FIRST(C)$ (zweites Auftreten in Regel 2). Also:
  $FOLLOW(B) = \{b, c\}$
- $FOLLOW(C) = FOLLOW(A)$, denn C kommt nur in Regel 2 auf der rechten Seite vor und zwar als letzter Bestandteil von A.

Vollständige Tabelle

Die sich aus den FIRST-Mengen ergebenden Einträge werden ergänzt:

- Regel 4 für die Folgesymbole von B
- *error*-Einträge für nicht korrekte Vorausschau-Tokens

	a	b	c	$
A	$A \to a$	$A \to BBC$	$A \to BBC$	*error*
B	*error*	$B \to b$ $B \to \epsilon$	$B \to \epsilon$	*error*
C	*error*	error	$C \to cc$	*error*

Anmerkungen zur Tabelle

- $TAB(B, b)$ enthält zwei Regeln.
- Die erste steht dort, weil b Anfangssymbol der rechten Regelseite ist.
  Die zweite, weil ϵ in der $FIRST$-Menge der rechten Regelseite steht und b Folgesymbol von B ist.
- Die Grammatik hat demnach nicht die LL(1)-Eigenschaft.
- Bei genauer Betrachtung erkennt man, dass die Grammatik mehrdeutig ist. Man sieht dies an der Eingabe bcc. Aus A wird BBC abgeleitet. Jetzt lässt sich das b sowohl aus dem ersten als auch aus dem zweiten B erzeugen, das verbleibende B wird gelöscht!

Beispiel für Ablauf einer Berechnung

Zur folgender LL(1)-Grammatik

1. $S \rightarrow ABC$
2. $A \rightarrow aaA$
3. $A \rightarrow C$
4. $B \rightarrow bBd$
5. $B \rightarrow \epsilon$
6. $C \rightarrow c$
7. $C \rightarrow d$

gehört die LL(1)-Parsertabelle

	a	**b**	**c**	**d**	**$**
S	S→ABC	*error*	S→ABC	S→ABC	*error*
A	A→aaA	*error*	A→C	A→C	*error*
B	*error*	B→bBd	B→ϵ	B→ϵ	*error*
C	*error*	*error*	C→c	C→d	*error*

Eingabe: *aadbdc*

Schritt	**Kellerinhalt** (rechts=oben)	**Resteingabe** (Vorausschausymbol fett)	**Regel** (laut Tabelle)
0		**a**adbdc	
1	S	**a**adbdc	S→ABC
2	CBA	**a**adbdc	A→aaA
3	CBAaa	**a**adbdc	–
4	CBAa	**a**dbdc	–
5	CBA	**d**bdc	A→C
6	CBC	**d**bdc	C→d
7	CBd	**d**bdc	–
8	CB	**b**dc	B→bBd
9	CdBb	**b**dc	–
10	CdB	**d**c	B→ϵ
11	Cd	**d**c	–
12	C	**c**	C→c

LL(k) - Verfahren bei mehrdeutigen Grammatiken

Mehrdeutigkeit führt bei der Syntaxanalyse zu Situationen, in denen die Auswahl einer Ableitungsregel nicht möglich ist. Da die Mehrdeutigkeit natürlich auch nicht durch Vorausschau aufgelöst werden kann, hat eine mehrdeutige Grammatik niemals die LL(k)-Eigenschaft für irgendein k.
Trotz allem werden mehrdeutige Grammatiken in Verbindung mit LL(k)-Verfahren zur Syntaxanalyse eingesetzt, denn sie sind oft wesentlich einfacher als entsprechende eindeutige Grammatiken.
Vorgehensweise bei mehrdeutigen Grammatiken:

- Man bestimme die LL(k)-Parsertabelle.
- Die Tabelle enthält an mindestens einer Stelle mehrere Ableitungsregeln
- Man mache die Tabelle durch Entfernen der „unerwünschten“ Regeln eindeutig

Grammatik-Transformationen für Top-Down-Analyse: Entfernung linksrekursiver Regeln

Linksrekursive Regeln führen zu LL(1)-Konflikten. Man betrachte dazu:

$$X \to \beta \mid X\alpha$$

(X möge in β nicht vorkommen)
Die Linksableitungen sind von der Form:

$$X \Rightarrow X\alpha \Rightarrow X\alpha\alpha \Rightarrow \cdots \Rightarrow X\alpha^n \Rightarrow \beta\alpha^n$$

Mit den Regeln lässt sich jede Satzform $\beta\alpha^i, i \geq 0$ erzeugen. Da β immer am Anfang steht, wäre eine Vorausschau über β hinaus notwendig, um zu entscheiden, ob die linksrekursive Regel zu wählen ist oder die andere.

Mit den nachfolgenden Regeln wird dagegen β im ersten Ableitungsschritt erzeugt, die α-Satzformen mit einer rechtsrekursiven Regel anschließend. Der LL(1)-Konflikt ist beseitigt:

$$\begin{aligned} X &\rightarrow \beta X' \\ X' &\rightarrow \varepsilon \mid \alpha X' \end{aligned}$$

Die Linksableitungen haben hier folgende Form

$$X \Rightarrow \beta X' \Rightarrow \beta\alpha X' \Rightarrow \beta\alpha\alpha X' \Rightarrow \cdots \Rightarrow \beta\alpha^n X' \Rightarrow \beta\alpha^n$$

Das Transformationsschema lässt sich sehr leicht auf Regeln der Form

$$X \rightarrow \beta_1 \mid \cdots \mid \beta_k \mid X\alpha_1 \mid \cdots \mid X\alpha_i$$

und (nicht ganz so einfach) auf indirekte Linksrekursion erweitern.

Linksfaktorisierung

Wenn mehrere Regeln für eine Variable X auf der rechten Seite mit dem gleichen Präfix u beginnen, führt dies (für $u \neq \varepsilon$) zu einem LL(1)-Konflikt.

$$X \to uv_1 \mid uv_2 \mid \cdots \mid uv_k$$

Durch eine einfache Transformation sorgt man dafür, dass u eindeutig in einem ersten Schritt erzeugt wird und der Rest (irgendein v_i) in einem weiteren Ableitungsschritt:

$$\begin{aligned} X &\to uX' \\ X' &\to v_1 \mid v_2 \mid \cdots \mid v_k \end{aligned}$$

Grammatiken für Audrücke

- In gängigen Programmiersprachen lassen sich aus Konstanten, Variablen und Operatoren komplexe Ausdrücke konstruieren.
- Bei der Implementierung der Sprachen muss deren Syntax in einer Grammatik spezifiziert werden.
- Problem: Wie berücksichtigt man Operatoreigenschaften wie Präzedenz und Assoziativität bei den Ableitungsregeln?

Operator-Eigenschaften

Operator-Assoziativität

Ein Operator $\circ$ heißt

- linksassoziativ, falls $a \circ b \circ c = (a \circ b) \circ c$
- rechtsassoziativ, falls $a \circ b \circ c = a \circ (b \circ c)$
- nichtassoziativ, falls $a \circ b \circ c$ nicht erlaubt ist.

Beispiel: $a - b - c = (a - b) - c$, denn Subtraktionsoperator ist linksassoziativ.

Operator-Präzedenz

Unter den Operatoren wird eine Präzedenzfolge festgelegt, ein Operator höherer Präzedenz bindet stärker.

Beispiel:

$a + b * c = a + (b * c)$, denn Multiplikationsoperator hat höhere Präzedenz als Additionsoperator.

Mehrdeutige Ausdrucksgrammatik

Folgende Ausdrucksgrammatik ist in zweifacher Hinsicht mehrdeutig

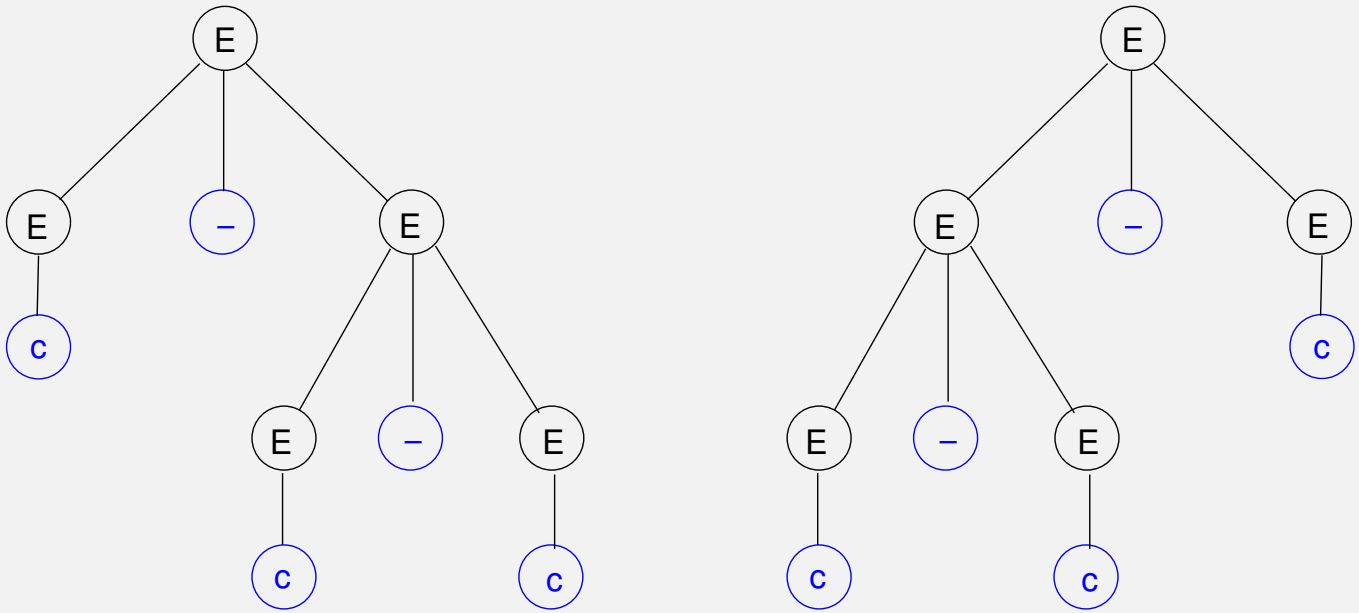
$$E \to c \mid E - E \mid E * E$$

E steht für „Expression“, c für eine beliebige Konstante.

1. Da die Assoziativität der Operatoren nicht in den Regeln berücksichtigt ist, gibt es z.B. für $c - c - c$ zwei Linksableitungen.
2. Da die Präzedenz der Operatoren nicht in den Regeln berücksichtigt ist, gibt es z.B. für $c * c - c$ zwei Linksableitungen.

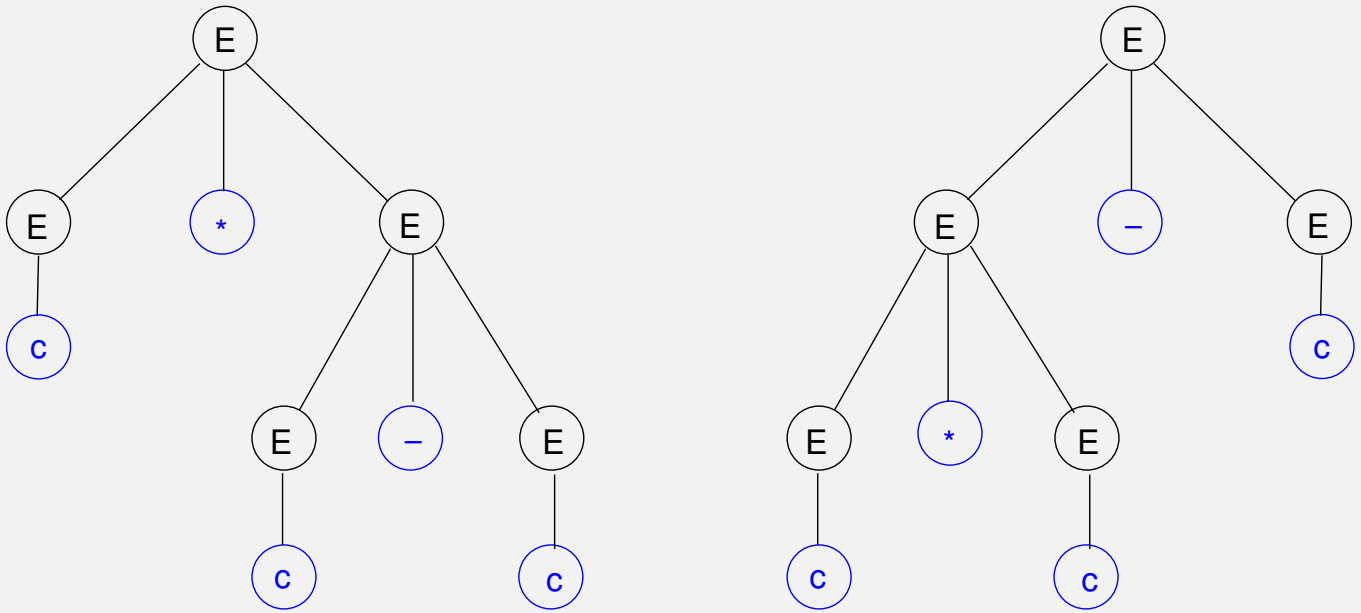
Mehrdeutigkeit bezüglich Assoziativität

Ableitungsbäume für $c - c - c$:



Mehrdeutigkeit bezüglich Präzedenzen

Ableitungsbäume für $c * c - c$:



Operatoreigenschaften in der Grammatik

Folgende Grammatik für Ausdrücke mit den linksassoziativen Operatoren $*$ und $-$ (geringe Präzedenz) sowie $*$ und $/$ (hohe Präzedenz) ist eindeutig:

$$\begin{aligned} E &\rightarrow E - T \mid E + T \mid T \\ T &\rightarrow T * F \mid T/F \mid F \\ F &\rightarrow c \mid (E) \end{aligned}$$

Man sieht an den Ableitungsbäumen leicht, dass ein Vertauschen von E und T in den ersten beiden Regeln

$$E \rightarrow T - E \mid T + E \mid T$$

für rechtsassoziative Operatoren und Regeln der Form

$$E \rightarrow T - T \mid T + T \mid T$$

für nichtassoziative Operatoren geeignet wären.

Notation: EBNF

- EBNF = Erweiterte Backus-Naur-Form
- Meta-Sprache zur Syntaxdefinition von Programmiersprachen
- Gegenüber kontextfreien Grammatiken kompaktere Notation durch
  - Wiederholungsoperator: `{...}`
  - Operator für optionale Satzformen: `[...]`
  - Operator für Alternativen: `|`
- EBNF lässt sich automatisch in kontextfreie Grammatik konvertieren