

Kapitel 4: Syntax-Analyse

Aufgabe

Die Token-Folge wird strukturiert in Anweisungen, Ausdrücke etc., um die Semantische Analyse und Code-Erzeugung zu ermöglichen

Themen

- Kontextfreie Grammatik
- Äquivalente Grammatiken
- Bottom Up Analyse
- Bison Beispiel

Kontextfreie Grammatik

- Grundlage für die Definition von Programmiersprachen
- Grundlage für die Syntax-Analyse im Übersetzer

Beispiel: Grammatik für Funktionsaufrufe

```
F -> id
F -> id(L)
L -> L,L
L -> num
L -> F
```

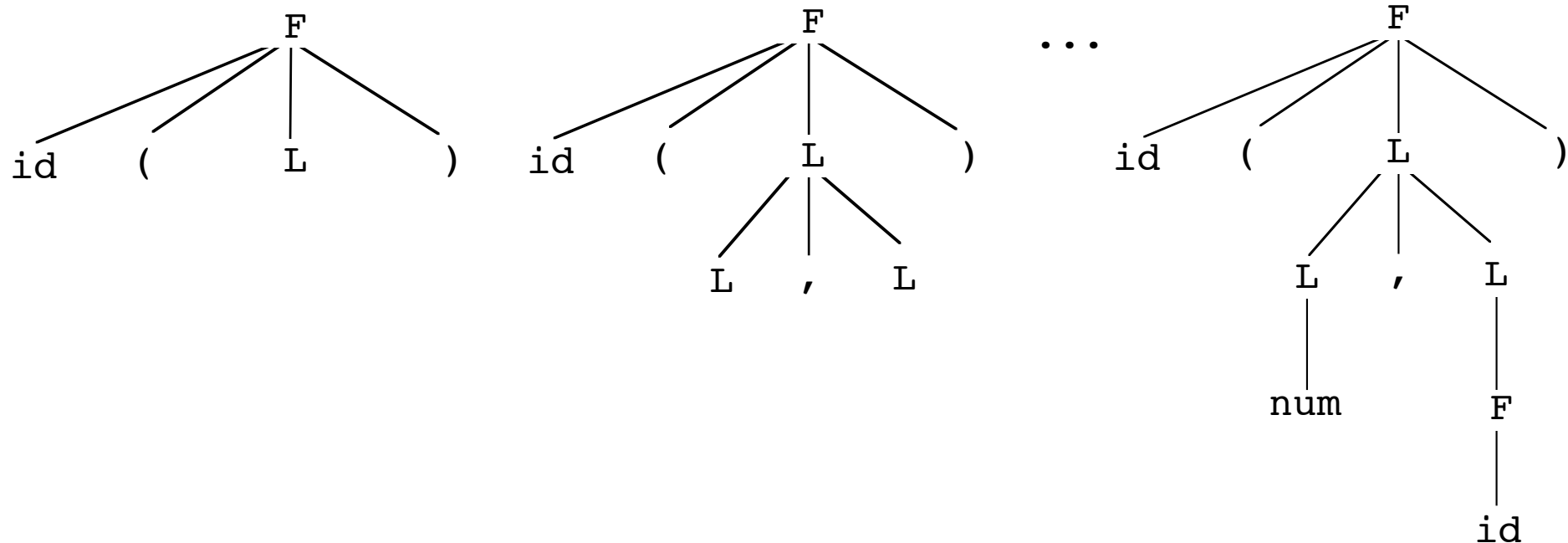
Eine Ableitung

$F \Rightarrow id(L) \Rightarrow id(L,L) \Rightarrow id(num,L) \Rightarrow id(num,F) \Rightarrow id(num,id)$

In jedem Ableitungsschritt wird eine Produktion der Grammatik angewandt, d.h. ein Nonterminal wird durch eine seiner rechten Seiten ersetzt

Ableitungsbaum

$F \Rightarrow id(L) \Rightarrow id(L,L) \Rightarrow id(num,L) \Rightarrow id(num,F) \Rightarrow id(num,id)$



**Der Ableitungsbaum zeichnet die Ableitung auf,
d.h. jede angewandte Produktion wird als Teilbaum eingetragen
(Nonterminal als Wurzel, rechte Seite als Nachfolger)**

Äquivalenz von Grammatiken

Zwei Grammatiken sind äquivalent, wenn sie dieselbe Sprache beschreiben

Folgende Grammatiken beschreiben z.B. Listen von Zahlen $\text{num}, \text{num}, \dots$

G1: $L \rightarrow L, L$ **links- und rechtsrekursiv**
 $L \rightarrow \text{num}$

G2: $L \rightarrow L, \text{num}$ **linksrekursiv**
 $L \rightarrow \text{num}$

G3: $L \rightarrow \text{num}, L$ **rechtsrekursiv**
 $L \rightarrow \text{num}$

G4: $L \rightarrow \text{num } L_r$ **restrekursiv**
 $L_r \rightarrow, \text{num } L_r$
 $L_r \rightarrow \varepsilon$

G5: $L \rightarrow \text{num}(, \text{num})^*$ **Regular Right Part**

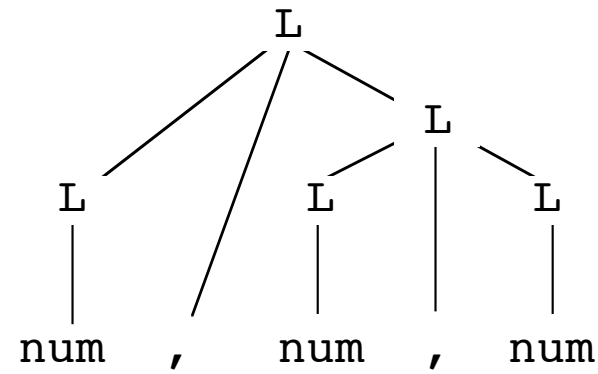
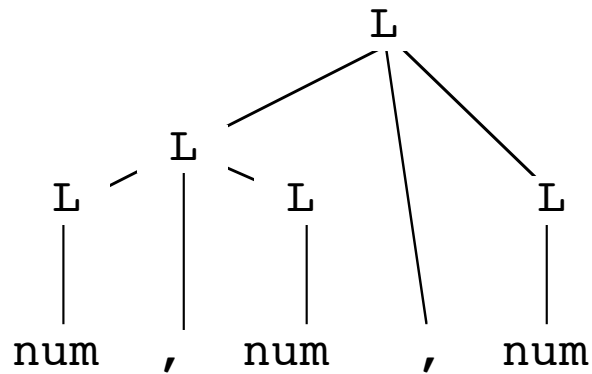
G6: $L \rightarrow \text{num}\{, \text{num}\}$ **Extended BNF**

Links- und Rechtsrekursion

Beispiel

$L \rightarrow L, L$

$L \rightarrow \text{num}$



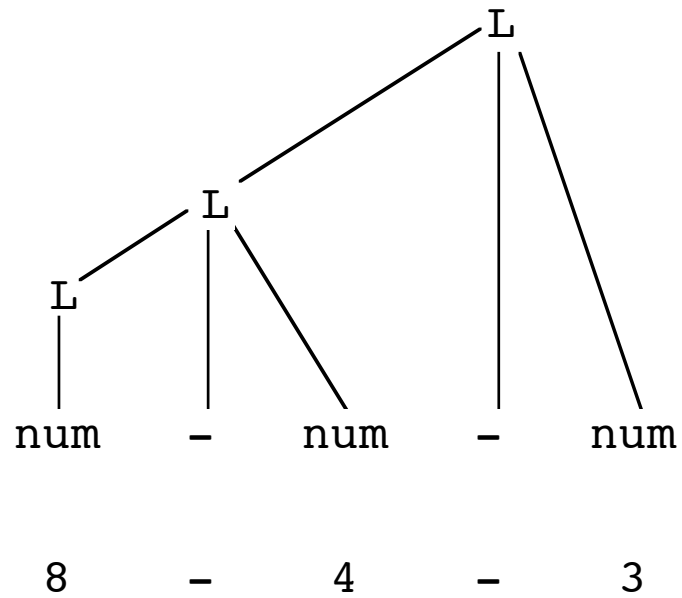
Die Grammatik ist mehrdeutig,
weil es zu einer Satzform verschiedene Ableitungsbäume gibt

Links- vs. Rechtsrekursion

Beispiel

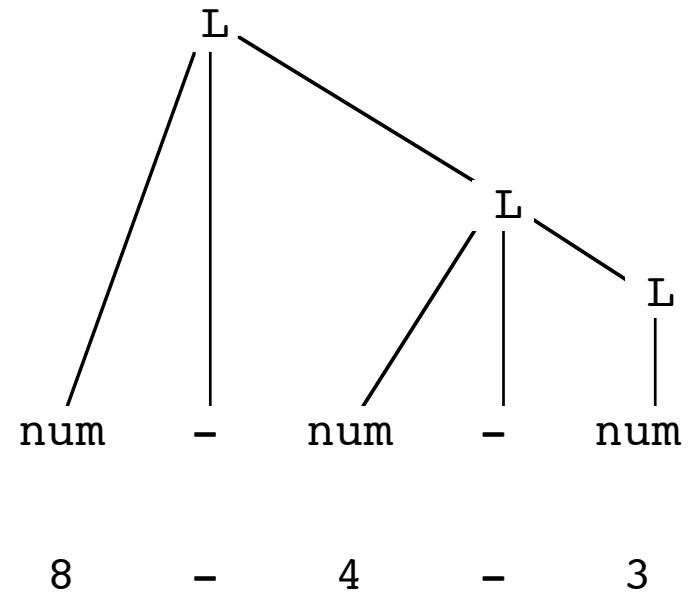
L -> L - num

L -> num



L -> num - L

L -> num



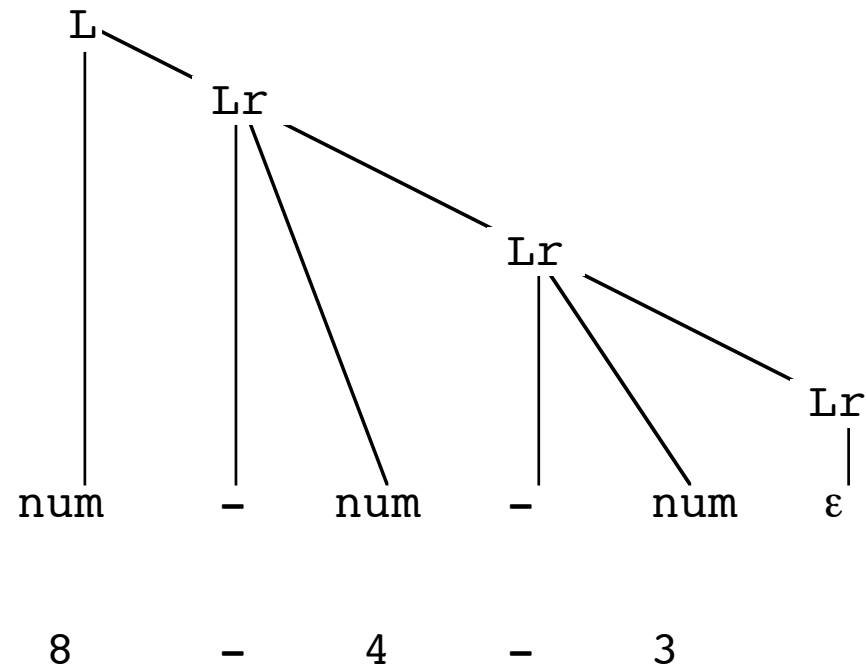
Restrekursion

Beispiel

$L \rightarrow \text{num } Lr$

$Lr \rightarrow - \text{ num } Lr$

$Lr \rightarrow \varepsilon$



Zwei äquivalente Grammatiken für Ausdrücke

Linksrekursion

$S \rightarrow E$ \Leftrightarrow

$E \rightarrow E + T \mid T$ \Leftrightarrow

$T \rightarrow T * F \mid F$ \Leftrightarrow

$F \rightarrow (E) \mid id$ \Leftrightarrow

**geeignet für
BOTTOM UP Analyse**

Restrekursion

$S \rightarrow E$

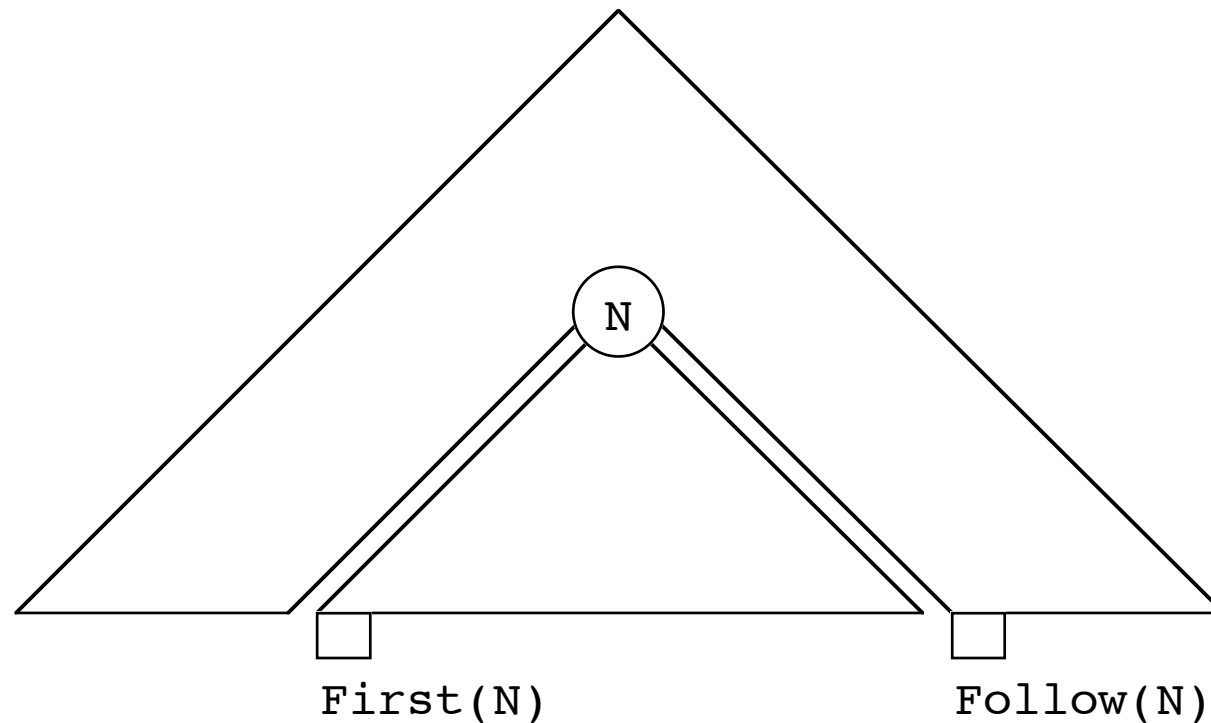
$E \rightarrow T E_r$
 $E_r \rightarrow + T E_r \mid \epsilon$

$T \rightarrow F T_r$
 $T_r \rightarrow * F T_r \mid \epsilon$

$F \rightarrow (E) \mid id$

**geeignet für
TOP DOWN Analyse**

FIRST- und FOLLOW-Mengen

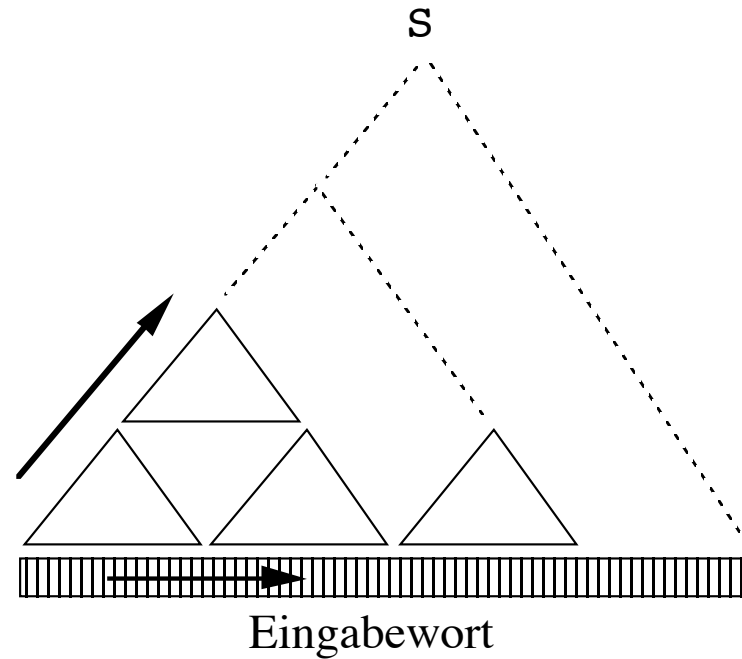


First(N) gibt an, mit welchen Terminalen die aus N ableitbaren Satzformen beginnen können, bzw. ob N nach ϵ ableitbar ist

Follow(N) gibt an, welche Terminalsymbole in beliebigen (aus S ableitbaren) Satzformen unmittelbar nach N folgen können, bzw. ob \$ folgen kann

BOTTOM UP Syntax-Analyse

Vom Eingabewort ausgehend wird versucht, zum Startsymbol zu reduzieren



Entscheidungsproblem: Reduzieren oder Weiterlesen?

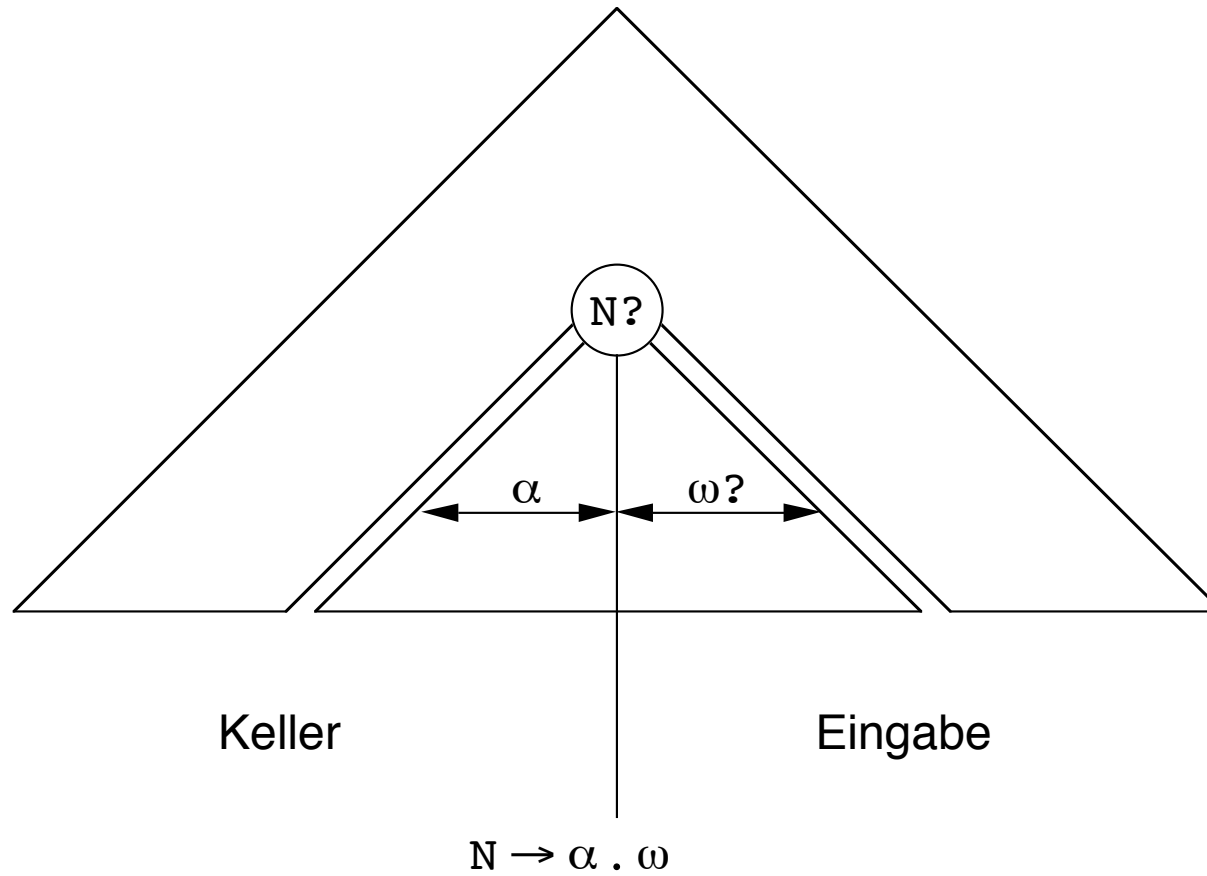
Ablauf einer BOTTOM UP Analyse

Grammatik (linksrekursiv)

$$\begin{array}{l}
 S \rightarrow E \\
 E \rightarrow E + T \quad | \quad T \\
 T \rightarrow T * F \quad | \quad F \\
 F \rightarrow (E) \quad | \quad id
 \end{array}$$

Keller	Eingabe	Aktion
\$	id + id * id \$	shift id
\$ id	+ id * id \$	reduce F \rightarrow id
\$ F	+ id * id \$	reduce T \rightarrow F
\$ T	+ id * id \$	reduce E \rightarrow T
\$ E	+ id * id \$	shift + (nicht: reduce S \rightarrow E)
\$ E +	id * id \$	shift id
\$ E + id	* id \$	reduce F \rightarrow id
\$ E + F	* id \$	reduce T \rightarrow F
\$ E + T	* id \$	shift * (nicht: reduce E \rightarrow E + T)
\$ E + T *	id \$	shift id
\$ E + T * id	\$	reduce F \rightarrow id
\$ E + T * F	\$	reduce T \rightarrow T * F
\$ E + T	\$	reduce E \rightarrow E + T
\$ E	\$	reduce S \rightarrow E
\$ S	\$	accept

Analysezustand und Item



Items

Ein Item $N \rightarrow \alpha . \omega$ ist eine Produktion, die einem Analysezustand zugeordnet ist:
Um N abzuleiten, wurde bereits α abgeleitet und ω muß noch abgeleitet werden

Fallunterscheidungen für ω

(1) $N \rightarrow \alpha . a\beta$

Aktion **shift**, falls a in der Eingabe

(2) $N \rightarrow \alpha .$

Aktion **reduce**, falls ein Element aus $\text{Follow}(N)$ in der Eingabe

(3) $N \rightarrow \alpha . A\beta$ mit $A \rightarrow \gamma$

Um A abzuleiten, muß γ abgeleitet werden,
d.h. $A \rightarrow . \gamma$ ist dem gleichen Zustand zugeordnet (Zustand = Item-Menge)

Konstruktion der Item-Mengen

Grammatik

- (0) $S \rightarrow E$
- (1) $E \rightarrow E \text{ op } T$
- (2) $E \rightarrow T$
- (3) $T \rightarrow (E)$
- (4) $T \rightarrow \text{id}$

Item-Mengen

$$I_0 = \left\{ \begin{array}{l} S \rightarrow \cdot E \\ E \rightarrow \cdot E \text{ op } T \\ E \rightarrow \cdot T \\ T \rightarrow \cdot (E) \\ T \rightarrow \cdot \text{id} \end{array} \right\}$$

$$I_1 = \left\{ \begin{array}{l} S \rightarrow E \cdot \\ E \rightarrow E \cdot \text{op } T \end{array} \right\} \quad I_1 = \text{goto}(I_0, E)$$

$$I_2 = \{ E \rightarrow T \cdot \} \quad I_2 = \text{goto}(I_0, T)$$

$$I_3 = \left\{ \begin{array}{l} T \rightarrow (\cdot E) \\ E \rightarrow \cdot E \text{ op } T \\ \dots \end{array} \right\} \quad I_3 = \text{goto}(I_0, ()$$

BOTTOM UP Analyse-Tabellen

Item-Mengen (Ausschnitt)

$I_1 = \{ S \rightarrow E \cdot \quad \quad \quad I_1 = \text{goto}(I_0, E)$
 $\quad \quad E \rightarrow E \cdot \text{op } T \}$

$I_2 = \{ E \rightarrow T \cdot \} \quad \quad \quad I_2 = \text{goto}(I_0, T)$

state	action-Tabelle					goto-Tabelle					
	id	op	()	\$	id	op	()	E	T
0	s	.	s	.	.	4	.	3	.	1	2
1	.	s	.	.	r0	.	5
2	.	r2	.	r2	r2
3	s	.	s	.	.	4	.	3	.	6	2
4	.	r4	.	r4	r4
5	s	.	s	.	.	4	.	3	.	.	7
6	.	s	.	s	.	.	5	.	8	.	.
7	.	r1	.	r1	r1
8	.	r3	.	r3	r3

Ablauf einer BOTTOM UP Analyse

state	action-Tabelle					goto-Tabelle					
	id	op	()	\$	id	op	()	E	T
0	s	.	s	.	.	4	.	3	.	1	2
1	.	s	.	.	r0	.	5
2	.	r2	.	r2	r2
3	s	.	s	.	.	4	.	3	.	6	2
4	.	r4	.	r4	r4
5	s	.	s	.	.	4	.	3	.	.	7
6	.	s	.	s	.	.	5	.	8	.	.
7	.	r1	.	r1	r1
8	.	r3	.	r3	r3

Keller

Eingabe

Aktion

```

0           id op id op id $   s
0 4         op id op id $   r4 (T -> id)
0 2         op id op id $   r2 (E -> T)
0 1         op id op id $   s
0 1 5       id op id $   s
0 1 5 4     op id $   r4 (T -> id)
0 1 5 7     op id $   r1 (E -> E op T)
0 1         op id $   s
...

```


LR(1)–Grammatik

Eine LR(1)– Grammatik erlaubt eine deterministische BOTTOM UP Analyse
[von Links nach rechts mittels Rechtsableitung und 1 Symbol Vorausschau]

Jede LR(1)–Grammatik hat eine eindeutige Analyse–Tabelle (action/goto)

$$\text{LR} \supset \text{LALR} \supset \text{SLR}$$

Bison Beispiel

```
%token    ASSIGNOP PLUSASSIGNOP
%left    PLUSOP
%token    REGISTER NUMBER ';'
%start    stmt
%%

stmt      : REGISTER ASSIGNOP expr ';'
          | REGISTER PLUSASSIGNOP expr ';'

expr      : REGISTER
          | constexpr

constexpr: NUMBER
          | constexpr PLUSOP constexpr

%%

int yyerror(char *e) {
    printf("Parser error: '%s'...\n", e);
    exit(2);
}

int main(void) {
    yyparse();
    return 0;
}
```