

# Kapitel 5: 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
- Top Down Analyse
- Bottom Up Analyse

# 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
```

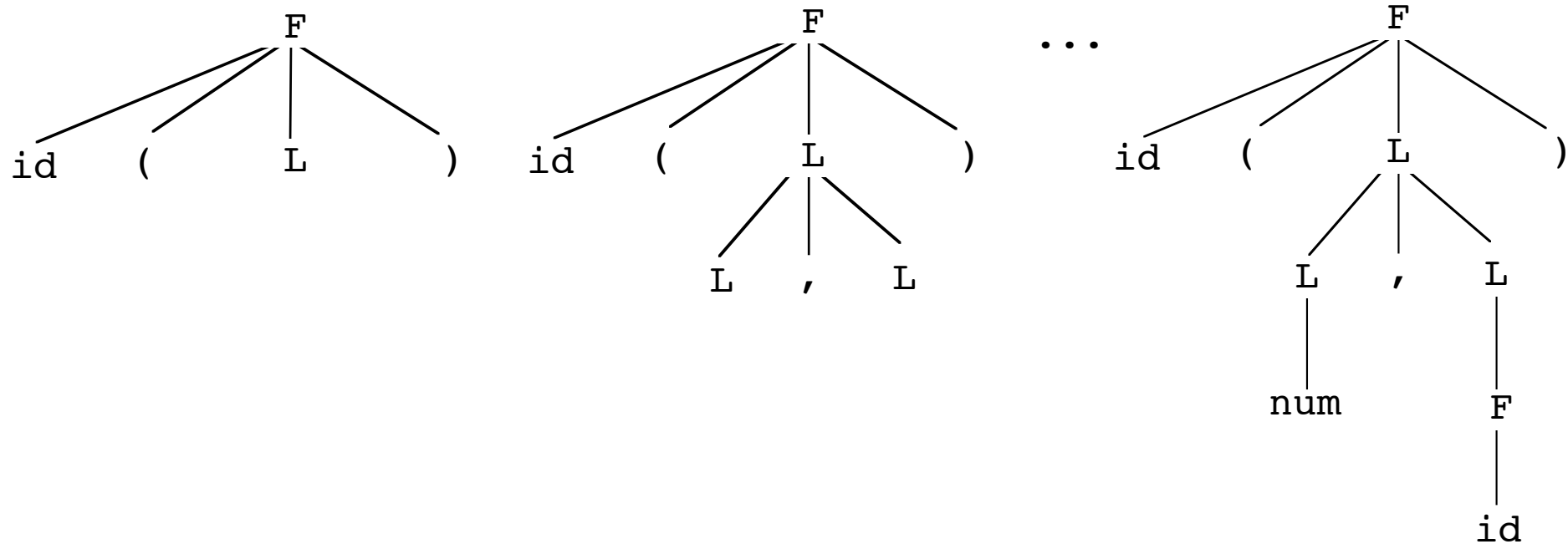
## Eine Ableitung

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

**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**  
 $L_r \rightarrow , \text{num } L_r$   
 $L_r \rightarrow \square$

---

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

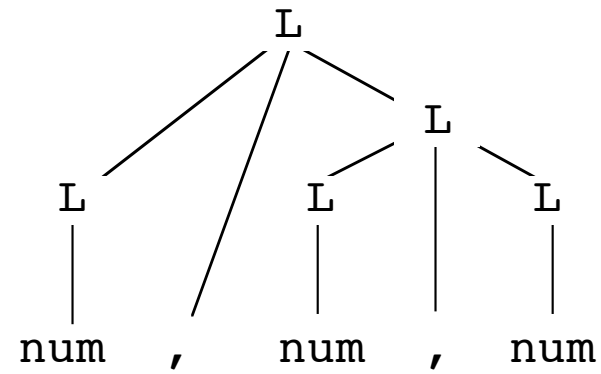
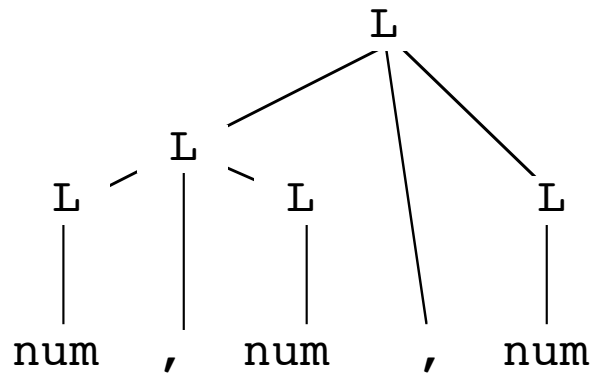
G6:  $L \rightarrow \text{num}\{, \text{num}\}$             **Extended BNF**

# Links- und Rechtsrekursion

## Beispiel

$L \rightarrow L, L$

$L \rightarrow \text{num}$



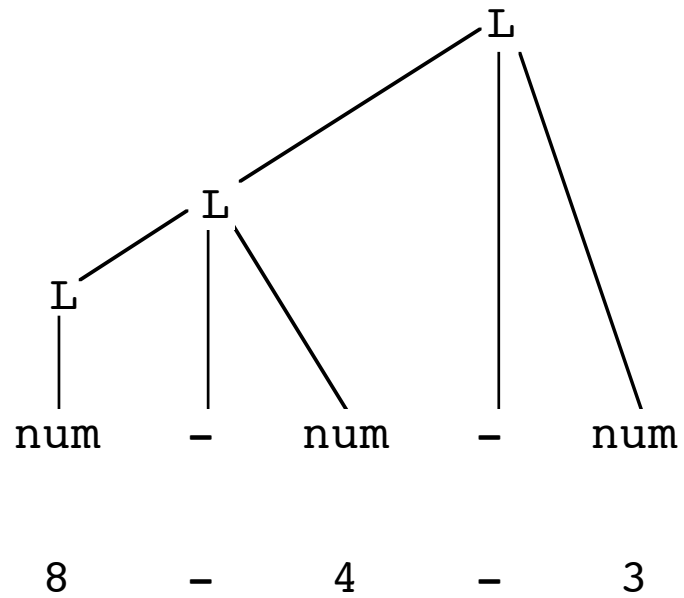
Die Grammatik ist mehrdeutig,  
weil es zu einer Satzform verschiedene Ableitungsbäume gibt

# Links- vs. Rechtsrekursion

## Beispiel

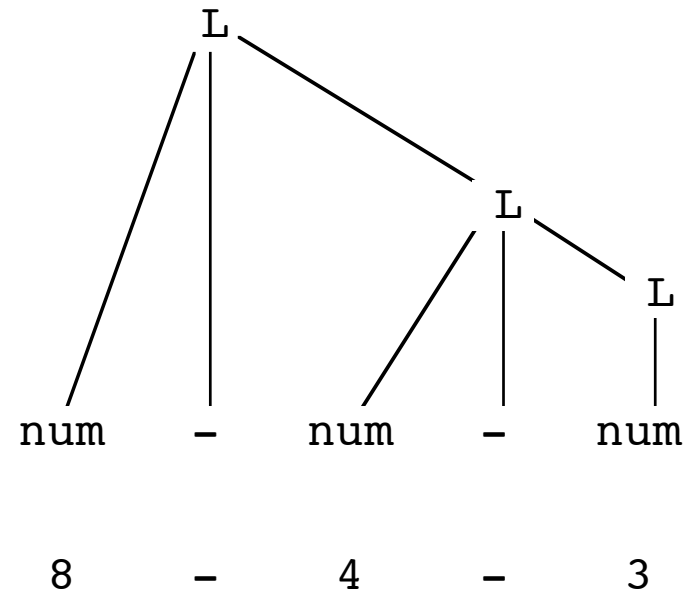
$L \rightarrow L - \text{num}$

$L \rightarrow \text{num}$



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

$L \rightarrow \text{num}$



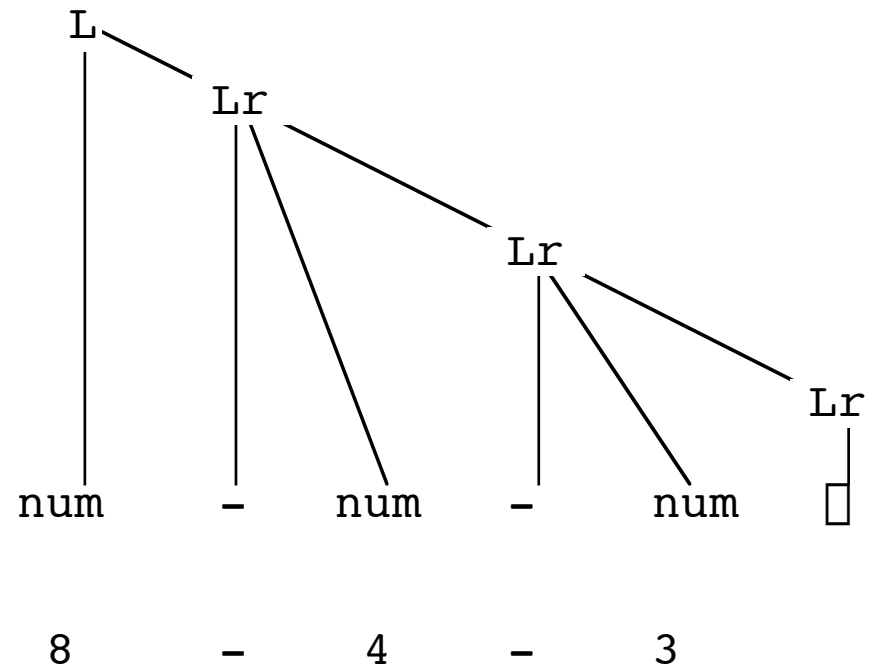
# Restrekursion

## Beispiel

L  $\rightarrow$  num Lr

Lr  $\rightarrow$  - num Lr

Lr  $\rightarrow$   $\square$



# 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

$S \rightarrow E$

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

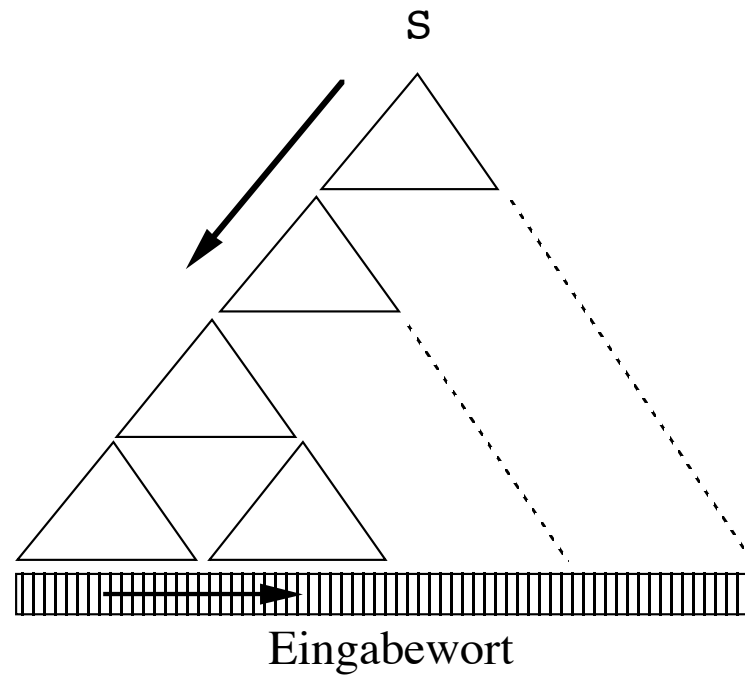
$T \rightarrow F T_r$   
 $T_r \rightarrow * F T_r \mid \square$

$F \rightarrow ( E ) \mid id$

**geeignet für  
TOP DOWN Analyse**

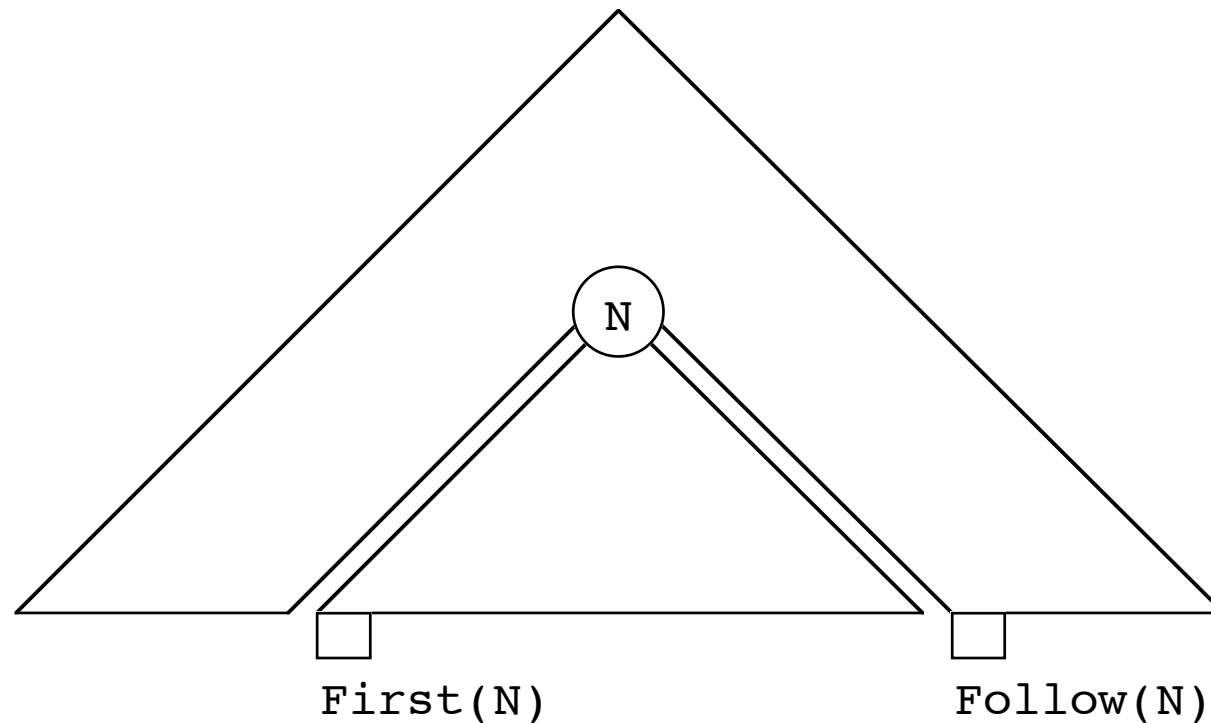
# TOP DOWN Syntax-Analyse

Vom Startsymbol ausgehend wird versucht, das Eingabewort abzuleiten



**Entscheidungsproblem: Welche Produktion ist bei Alternativen anzuwenden ?**

# FIRST- und FOLLOW-Mengen



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

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

# TOP DOWN Analyse-Tabelle

## Grammatik

S	->	E \$	First(S)={id, (}	
E	->	T Er	First(E)={id, (}	
Er	->	+ T Er   $\square$	First(Er)={+, $\square$ }	Follow(Er)={), \$}
T	->	F Tr	First(T)={id, (}	
Tr	->	* F Tr   $\square$	First(Tr)={*, $\square$ }	Follow(Tr)={), +, \$}
F	->	( E )   id	First(F)={id, (}	

## Analyse-Tabelle: (Ziel, Eingabesymbol) -> neue Ziele

	id	(	)	+	*	\$
S	E	E	.	.	.	.
E	T Er	T Er	.	.	.	.
Er	.	.	$\square$	+ T Er	.	$\square$
T	F Tr	F Tr	.	.	.	.
Tr	.	.	$\square$	$\square$	* F Tr	$\square$
F	id	( E )	.	.	.	.

# Ablauf einer TOP DOWN Analyse

	id	(	)	+	*	\$
S	E	E	.	.	.	.
E	T Er	T Er	.	.	.	.
Er	.	.	□	+ T Er	.	□
T	F Tr	F Tr	.	.	.	.
Tr	.	.	□	□	* F Tr	□
F	id	( E )	.	.	.	.

## Keller

## Eingabe

## Produktion (neue Ziele)

\$ S	id + id * id \$	S -> E
\$ E	id + id * id \$	E -> T Er
\$ Er T	id + id * id \$	T -> F Tr
\$ Er Tr F	id + id * id \$	F -> id
\$ Er Tr id	id + id * id \$	match id
\$ Er Tr	+ id * id \$	Tr -> □ (nicht: Tr -> * F Tr)
\$ Er	+ id * id \$	Er -> + T Er (nicht: Er -> □)
\$ Er T +	+ id * id \$	match +
\$ Er T	id * id \$	T -> F Tr
\$ ...		

# Rekursive Syntax-Prozeduren

Für jede Zeile der Tabelle (Nonterminal) wird eine Prozedur gebildet

	id	(	)	+	*	\$
S	E	E	.	.	.	.
E	T Er	T Er	.	.	.	.
Er	.	.	□	+ T Er	.	□
T	F Tr	F Tr	.	.	.	.
Tr	.	.	□	□	* F Tr	□
F	id	( E )	.	.	.	.

```
void E() {
    if (sym=id||sym='(') {T();Er();}
    else Error();
}
```

```
void Er() {
    if (sym='+'){nextsym();T();Er();}
    else if (sym=')'||sym='$'){
    else Error();
}
```

# LL(1)–Grammatik

Eine LL(1)– Grammatik erlaubt eine deterministische TOP DOWN Analyse  
[von Links nach rechts mittels Linksableitung und 1 Symbol Vorausschau]

Jede LL(1)–Grammatik hat eine eindeutige Analyse–Tabelle

Jede LL(1)–Grammatik erfüllt folgende Bedingung:

Für jede Produktion  $N \rightarrow \alpha_1 \mid \alpha_2 \mid \dots$  gilt

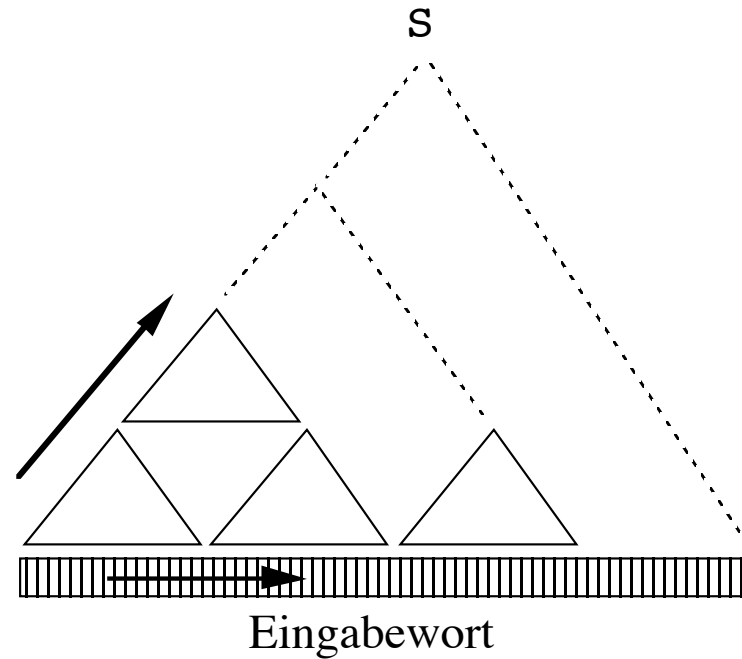
1.  $\text{First}(\alpha_i) \cap \text{First}(\alpha_j) = \{\}$  f.a.  $i, j$  ( $i \neq j$ )
2. höchstens ein  $\alpha_i$  läßt sich nach  $\epsilon$  ableiten
3. falls  $\alpha_i \Rightarrow^* \epsilon$ , dann  $\text{First}(\alpha_j) \cap \text{Follow}(N) = \{\}$  f.a.  $j \neq i$

**Gegenbeispiel:**  $A \rightarrow A a \mid b$        $\text{First}(A a) \cap \text{First}(b) = \{b\} \neq \{\}$

**Linksrekursive Produktionen sind nicht zur TOP DOWN Analyse geeignet**

# BOTTOM UP Syntax-Analyse

Vom Eingabewort ausgehend wird versucht, zum Startsymbol zu reduzieren



Entscheidungsproblem: Reduzieren oder Weiterlesen?

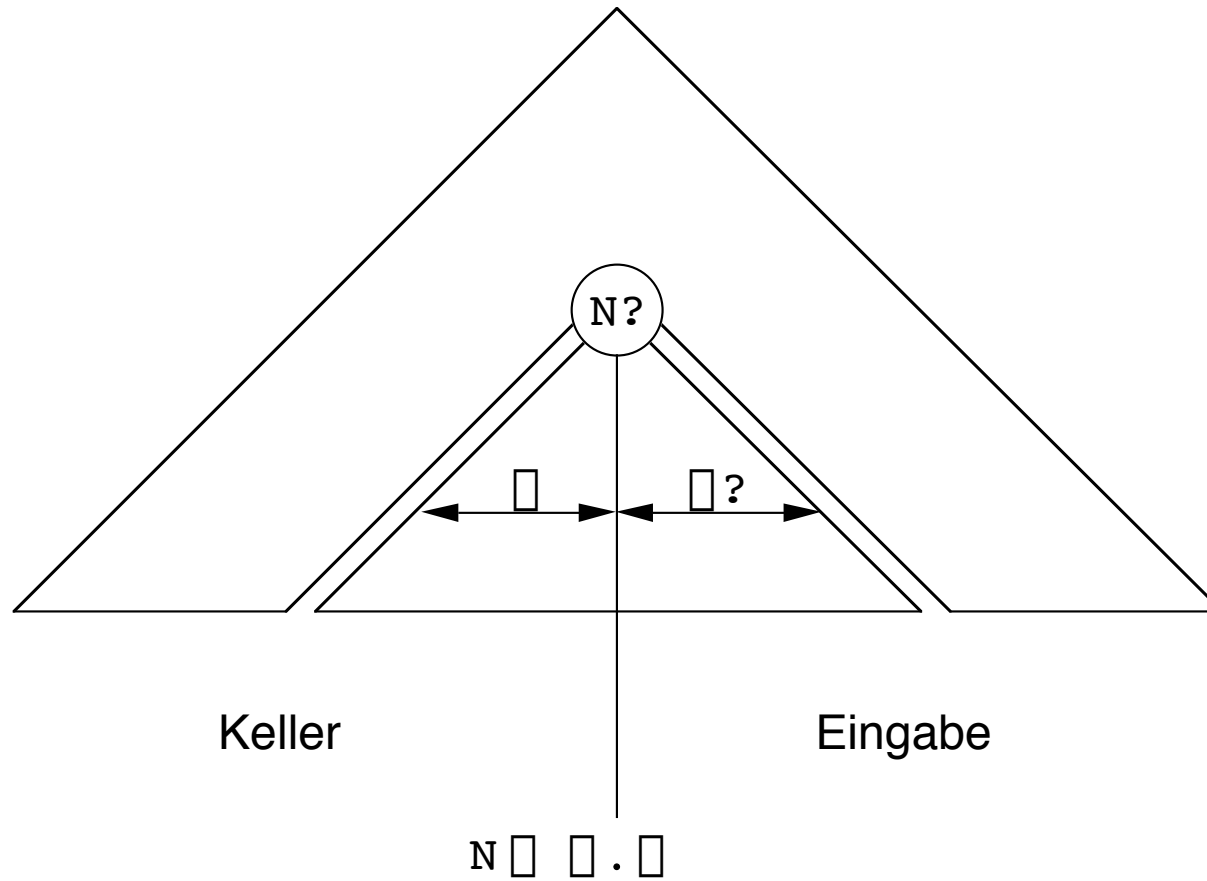
# Ablauf einer BOTTOM UP Analyse

## 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  $N \alpha \beta . \gamma$  ist eine Produktion, die einem Analysezustand zugeordnet ist:  
Um  $N$  abzuleiten, wurde bereits  $\alpha$  abgeleitet und  $\beta$  muß noch abgeleitet werden

Fallunterscheidungen für  $\beta$

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

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

(2)  $N \alpha \beta .$

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

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

Um  $A$  abzuleiten, muß  $\gamma$  abgeleitet werden,  
d.h.  $A \alpha \beta . \gamma$  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

---

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

---

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

---

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

---

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

# 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)

LR–Verfahren sind mächtiger als LL–Verfahren in Bezug auf die Grammatiken,  
die sie verarbeiten können

LR(1) LL(1)