Kapitel 5: Semantische Analyse

Prinzip

Operationen der Übersetzung werden an die Grammatik gebunden
–> Compiler–Generatoren

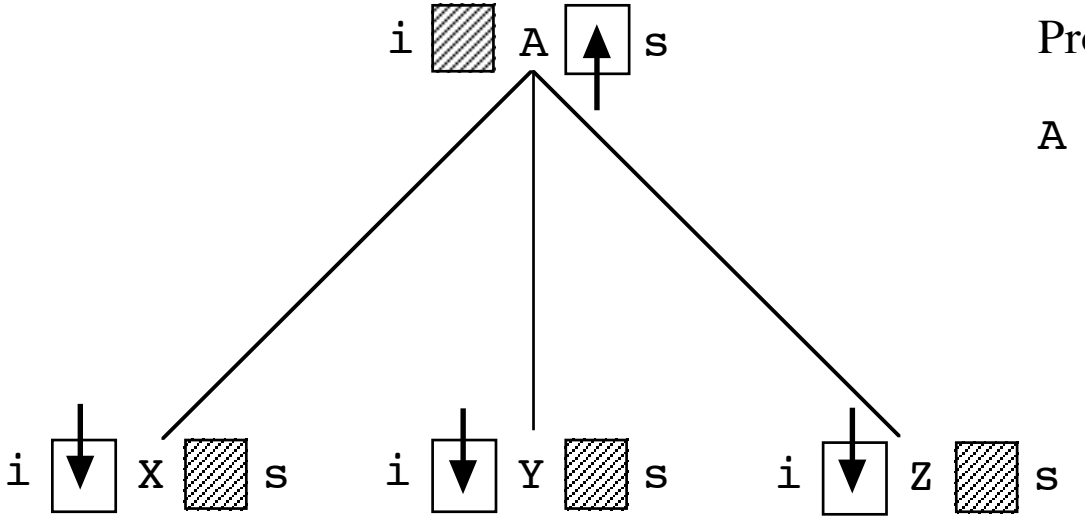
Themen

- **Attributierte Grammatik**
- **Mehrpass – Compiler – Generatoren**
- **Symboltabelle**
- **Typ – Überprüfung**
- **Ox Beispiel**

Attributierte Grammatik

... ist eine kontextfreie Grammatik mit folgenden Erweiterungen

- **jedes Grammatik-Symbol kann eine Menge von Attributen haben**
- **jeder Produktion werden Attribut-Berechnungsregeln zugeordnet**



Produktion	Regeln

`A -> X Y Z`	`A.s = ...`
	`X.i = ...`
	`Y.i = ...`
	`Z.i = ...`

Es gibt synthetisierte (aufsteigende) und ererbte (absteigende) Attribute

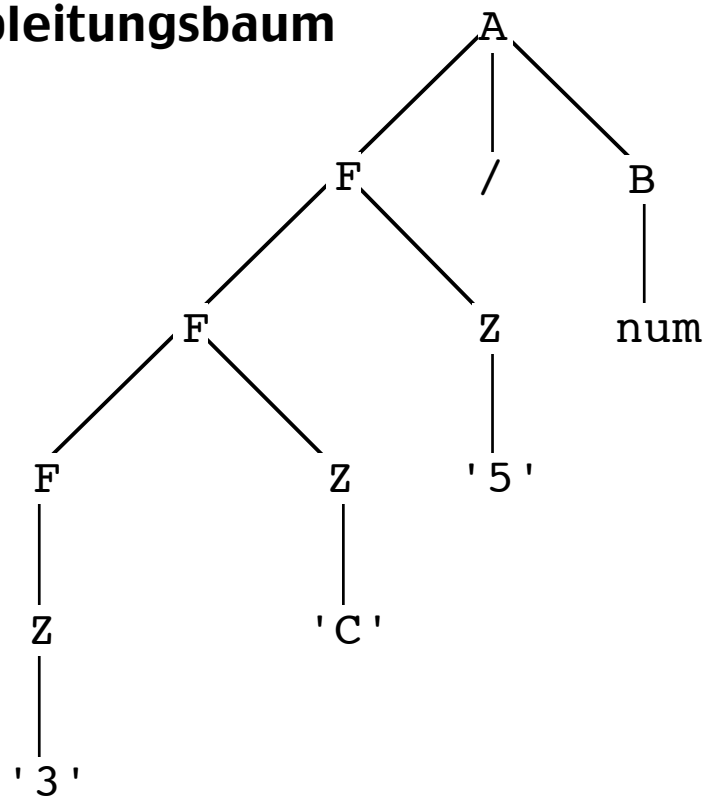
AG – Beispiel

Der Wert von Zahlen in verschiedenen Zahlensystemen soll berechnet werden
Schreibweise `<Ziffernfolge>/<Basis>`**, z.B.** `3C5/16`
Ziffern `0-9,A-Z`
Basis `2-35`

Grammatik

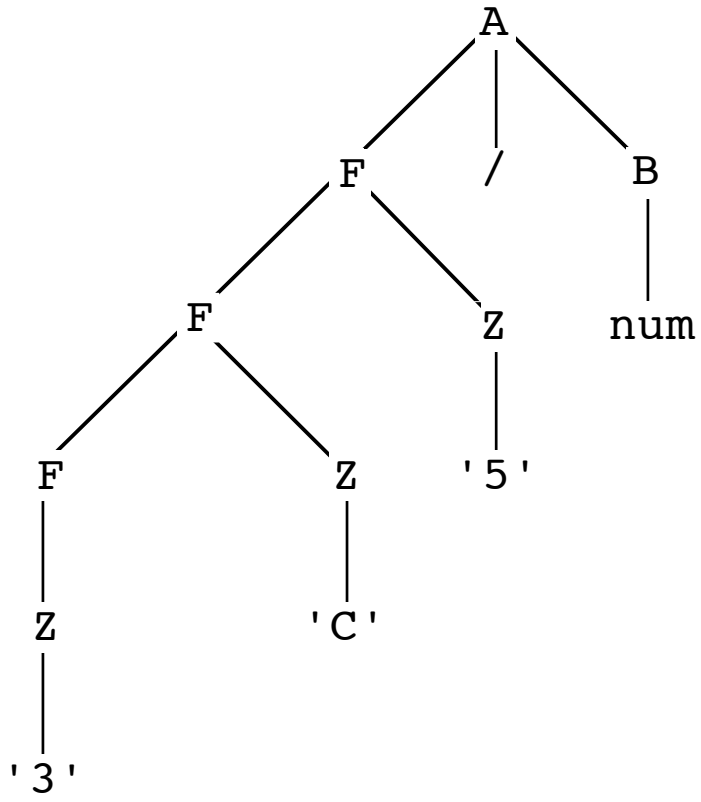
```
A -> F / B

F -> F Z
F -> Z

Z -> '0'
Z -> '1'
.
.
Z -> 'Y'
Z -> 'Z'

B -> num
```

Ableitungsbaum



AG – Beispiel: (1) Anfangswerte

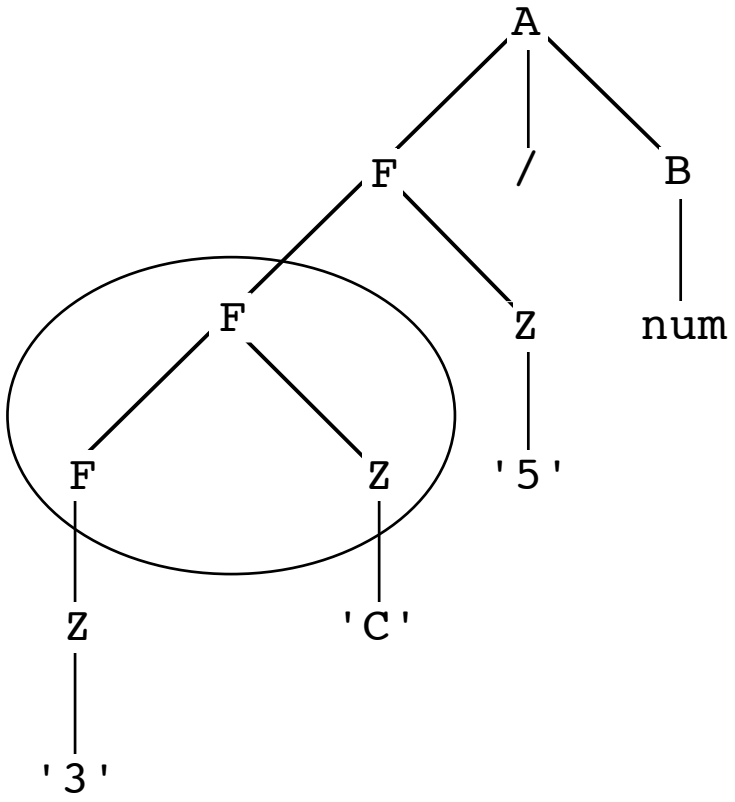
Terminale liefern die Anfangswerte für Attributberechnungen



Produktionen	Regeln
`Z -> '0'`	`Z.v = 0`
`Z -> '1'`	`Z.v = 1`
.	
.	
`Z -> 'Y'`	`Z.v = 34`
`Z -> 'Z'`	`Z.v = 35`
`B -> num`	`B.v = num.x`
`F -> Z`	`F.v = Z.v`

AG – Beispiel: (2) Berechnungen

Berechnungen erfolgen i.a. aufsteigend zur Wurzel hin (strukturelle Semantik)

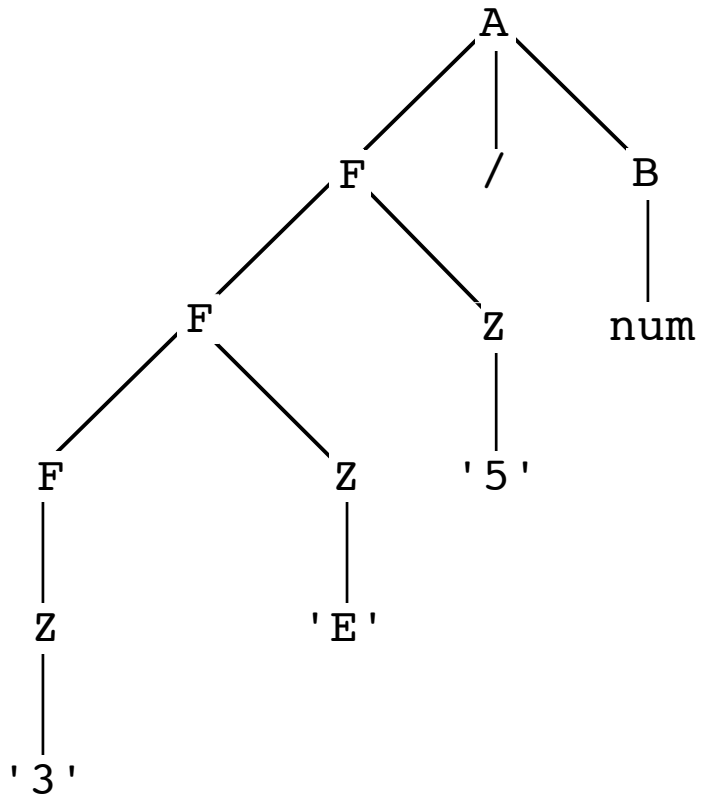


Produktionen	Regeln
F_1 -> F_2 Z	F_1.v = F_2.v*b+Z.v
A -> F / B	A.v = F.v

In den Berechnungsregeln einer AG gibt es keine globalen Variablen

AG – Beispiel: (3) Kontextinformation

Attributwerte können in benachbarte linke/rechte Teilbäume übertragen und von oben nach unten weitergegeben werden (ererbte Attribute)

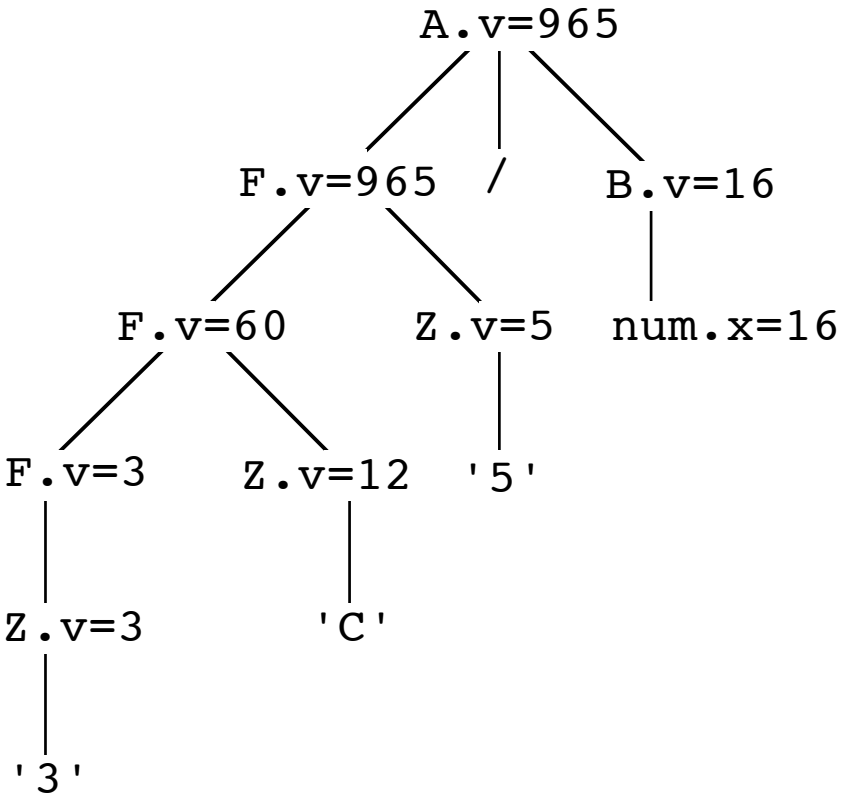


Produktionen	Regeln
`A -> F / B`	`F.b = B.v`
F_1 `->` F_2 `Z`	F_2`.b =` F_1`.b`

AG – Beispiel: (4) Attributierte Grammatik

Produktionen	Regeln
A -> F / B	A.v = F.v F.b = B.v
F_1 -> F_2 Z	F_1.v = F_2.v*F_1.b+Z.v F_2.b = F_1.b
F -> Z	F.v = Z.v
Z -> '0'	Z.v = 0
Z -> '1'	Z.v = 1
.	
.	
Z -> 'Y'	Z.v = 34
Z -> 'Z'	Z.v = 35
B -> num	B.v = num.x

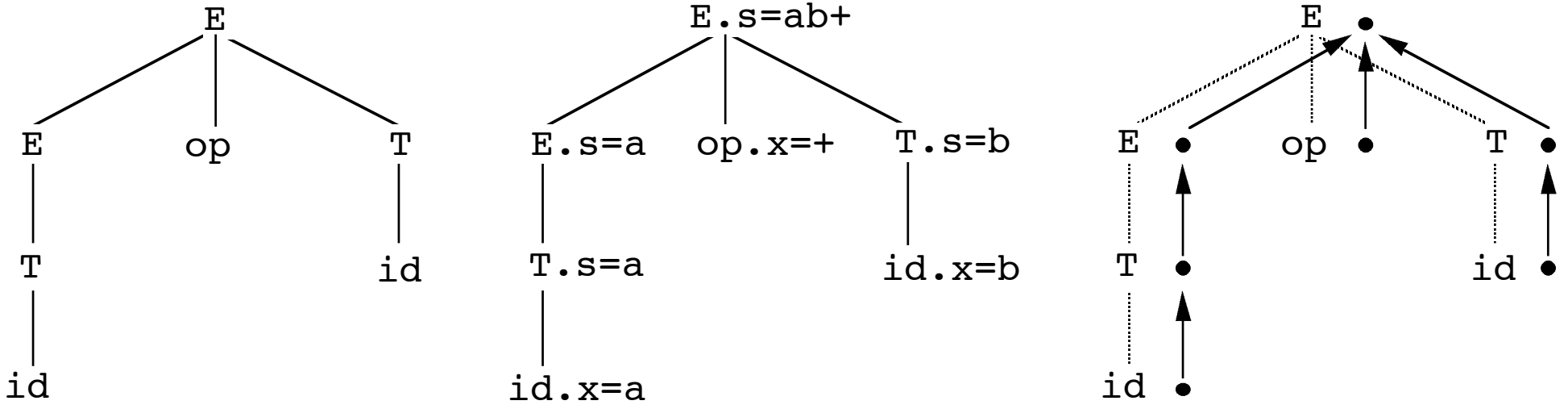
Attributierter Ableitungsbaum



Attributierte LR-Grammatik für Ausdrücke (Infix=>Postfix)

Produktion	Regeln
E -> E_1 op T	E.s = conc(E_1.s,T.s,op.x)
E -> T	E.s = T.s
T -> (E)	T.s = E.s
T -> id	T.s = id.x

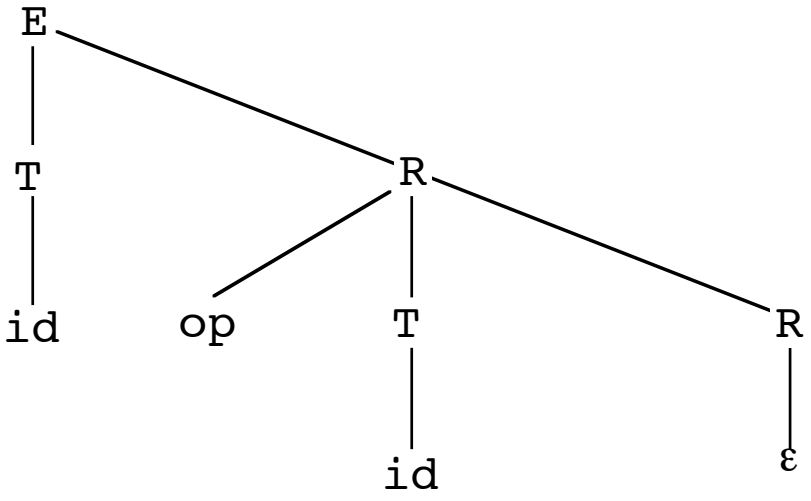
id op id

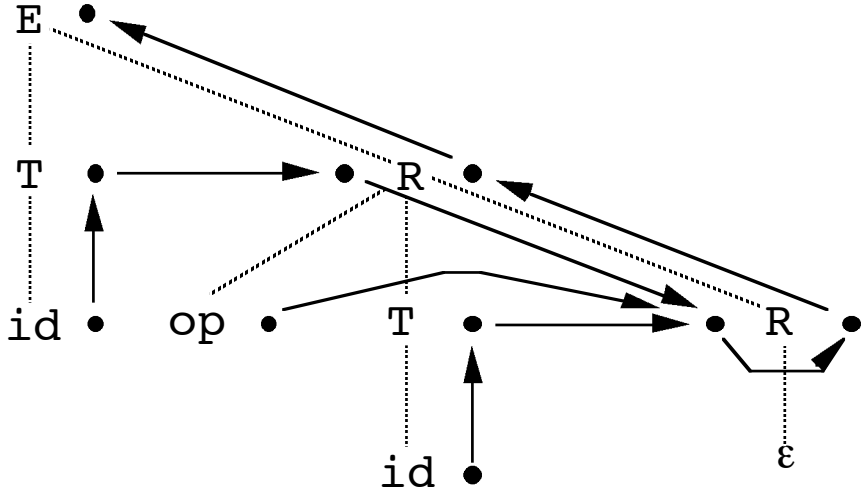


Eine AG, die nur synthetisierte Attribute hat, heißt S-attributiert

Attributierte LL-Grammatik für Ausdrücke (Infix=>Postfix)

Produktion	Regeln
`E -> T R`	`R.i = T.s` `E.s = R.s`
`R -> op T` R_1	R_1`.i = conc(R.i,T.s,op.x)` `R.s =` R_1`.s`
`R -> ε`	`R.s = R.i`
`T -> ( E )`	`T.s = E.s`
`T -> id`	`T.s = id.x`





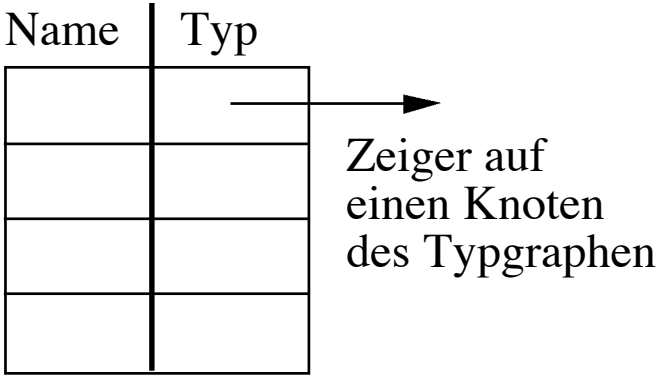
Mehrpass-Compiler-Generatoren

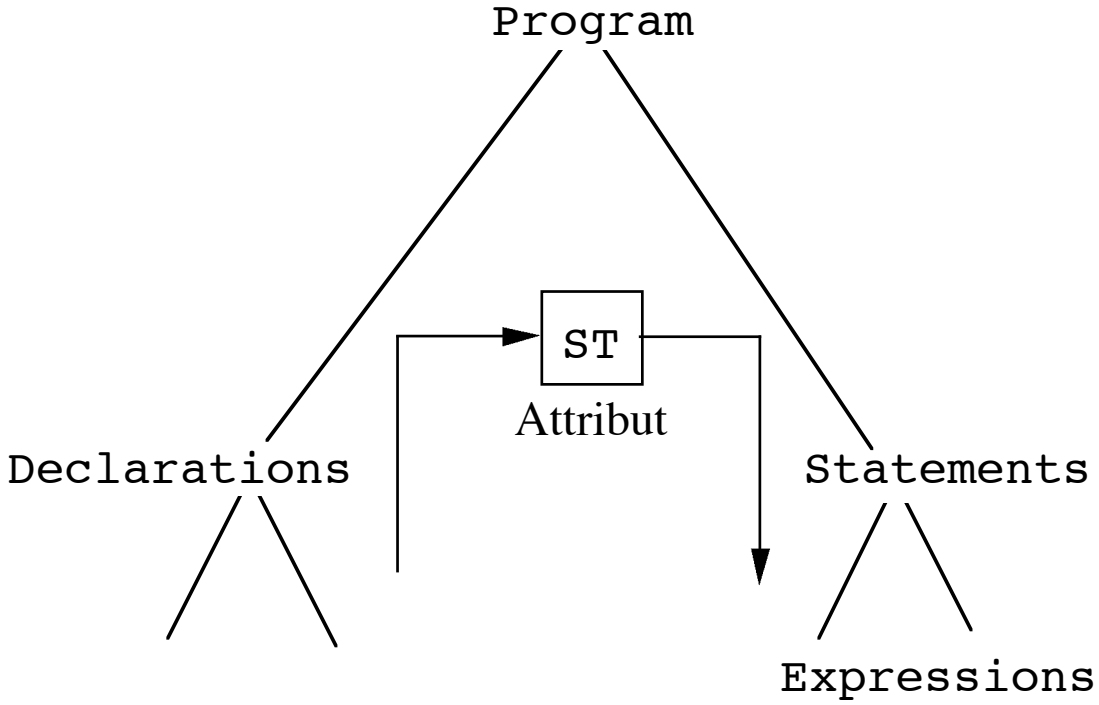
Allgemeine Lösung: Mehrpass-Übersetzung

a) Ableitungsbaum erzeugen
b) Attribute auswerten

Beispiel: Generator Ox

Symboltabelle





AG zur Erzeugung der Symboltabelle

Produktion	Regeln
P -> D S	D.i = () S.i = D.s
D -> D_1 ; D_2	D_1.i = D.i D_2.i = D_1.s D.s = D_2.s
D -> id : T	D.s = D.i ‖ (id.x,T.type)
T -> int	T.type = int
T -> real	T.type = real
D -> proc id P	h = D.i ‖ (id.x,proc) D.s = h P.i = h
P -> D S	D.i = P.i S.i = D.s

AG zur Typ-Überprüfung

Produktion	Regeln
P -> D S	D.i = () S.i = D.s
S -> S_1 ; S_2	S_1.i = S.i S_2.i = S.i
S -> id := E	E.i = S.i if (looktype(id.x,S.i)<>E.type) error
E -> E_1 + E_2	E_1.i = E.i E_2.i = E.i E.type = if (E_1.type==int && E_2.type==int) int else if (E_1.type==real && E_2.type==real) real else error
E -> id	E.type = looktype(id.x,E.i)

Ox-Lex Beispiel

```
%{
#include "beispiel1.h"
#include "oxout.tab.h"
%}

comment         \/\/.*
number          [0-9]+
register        %r([abcd]x|[sb]p|[sd]i|[89]|1[0-5])
whitespace      [\n\t ]

%%

"+="            return (PLUSASSIGNOP);
"="             return (ASSIGNOP);
"+"             return (PLUSOP);
";"             return (';');
{register}      return (REGISTER);
                        @{ @REGISTER.name@ = strdup(yytext); @}
{number}        return (NUMBER); @{ @NUMBER.val@ = atol(yytext); @}
{whitespace}+   ;
{comment}       ;

.               printf("Lexical error.\n"); exit(1);
```

Ox Beispiel

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

@attributes { char* name; } REGISTER
@attributes { long val; } NUMBER
@attributes { treenode *n; } stmt expr constexpr
@traversal @preorder codegen

%{
treenode *newOpNode(int op, treenode *left, treenode *right);
treenode *newRegNode(char* name);
treenode *newNumNode(long num);

extern void invoke_burm(NODEPTR_TYPE root);
%}

%start  stmt_list

%%

stmt_list:      /* empty */
        |       stmt ';' stmt_list
                @{
                        @codegen invoke_burm(@stmt.n@);
                @}
        ;
```

```
stmt    :   REGISTER ASSIGNOP expr
                @{ @i @stmt.n@ = newOpNode(ASSIGN,
                          newRegNode(@REGISTER.name@),@expr.n@); @}
        |       REGISTER PLUSASSIGNOP expr
                @{ @i @stmt.n@ = newOpNode(ADDASSIGN,
                          newRegNode(@REGISTER.name@),@expr.n@); @}
        ;

expr    :       REGISTER
                @{ @i @expr.n@ = newRegNode(@REGISTER.name@); @}
        |       constexpr
                @{ @i @expr.n@ = @constexpr.n@; @}
        ;

constexpr:      NUMBER
                @{ @i @constexpr.n@ = newNumNode(@NUMBER.val@); @}
        |       constexpr PLUSOP constexpr
                @{ @i @constexpr.n@ = newOpNode(ADD,
                              @constexpr.1.n@,@constexpr.2.n@); @}
        ;

%%

int yyerror(char *e) {...}
int main(void) {yyparse(); return 0;}
```