

Problem: Umformung der Grammatik

Die Umformung der Grammatik liefert eine Grammatik, die die Syntax korrekt erkennt, aber die Semantik verletzt.

Durch Einführung der Rechtsrekursion werden die Ausdrücke nun auch von rechts nach links bewertet und ausgerechnet.

$12/2*3$

1. $2*3 \rightarrow 6$

2. $12/6 \rightarrow 2$

Die Umformung der Grammatik ist also auf diese Weise unzulässig

EBNF

- Erweiterung der BNF durch nachfolgende Konstrukte:
- {...} Die in geschweiften Klammern angegebene Zeichenkette kommt 0 mal, einmal, oder mehrfach vor.
- [...] Die in eckigen Klammern angegebene Zeichenkette kommt optional einmal vor

Linksfaktorisierung (Ausklammern gleicher Anfänge in alternativen Regeln)

$A \rightarrow Xa \mid Xb$	StatementList \rightarrow statement ';' statementList statement
$A \rightarrow X A'$	StatementList \rightarrow statement statementList2
$A' \rightarrow a \mid b$	StatementList2 \rightarrow ';' statement statementList2 ϵ

Grammatik für einfache Ausdrücke

EBNF (3)

Beseitigung der Linksrekursion durch Iteration

N : faktor, term, expression

T : Zahl, '+', '*', '(', ')'

s : expression

R : expression ::= term exprx

exprx ::= { ('+' | '-') term }

term ::= faktor termx

termx ::= { ('*' | '/') faktor }

faktor ::= Zahl | '(' expression ')'

Unterschiedliche
Terminale Anfänge
der Alternativen

First/Follow

	First	Follow
expression	Zahl, (\$,)
exprx	+, '-', ϵ	\$,)
term	Zahl, (\$,), +, -
termx	*, /, ϵ	\$,), +, -
factor	Zahl, (\$,), +, -, *, /

Die hier angeführte Implementierung enthält wieder sowohl `expression`, als auch `exprx`. Die `while` Schleife implementiert dabei `exprx`.

```
double expression(void)
{
    double tmp;
    tmp=term();

    while (m.mc==mop && (m.cval=='+' || m.cval=='-'))
    {
        if (m.cval=='-' ) {lex(NULL); tmp -=term();} else
        if (m.cval=='+') {lex(NULL); tmp +=term();}
    }
    return tmp;
}
```

Umbau von Linksrekursionen zu Rechtsrekursionen

$A \rightarrow A a \mid b$	$\text{Expr} \rightarrow \text{Expr '+' Term} \mid \text{Term}$	$A = \text{Expr}$
		$a = \text{'+' Term}$
		$b = \text{Term}$
$A \rightarrow b A'$	$\text{Expr} \rightarrow \text{Term Rexpr}$	
$A' \rightarrow a A' \mid \varepsilon$	$\text{Rexpr} \rightarrow \text{'+' Term Rexpr} \mid \varepsilon$	

Grammatik für einfache Ausdrücke (G4)

N : faktor, term, expression
T : Zahl, '+', '*', '(', ')'
s : expression
R : expression ::= term rexpr
 texpr ::= '+' term rexpr
 | ϵ
 term ::= faktor rterm
 rterm ::= '*' faktor rterm
 | ϵ
 faktor ::= Zahl
 | '(' expression ')'

First/Follow

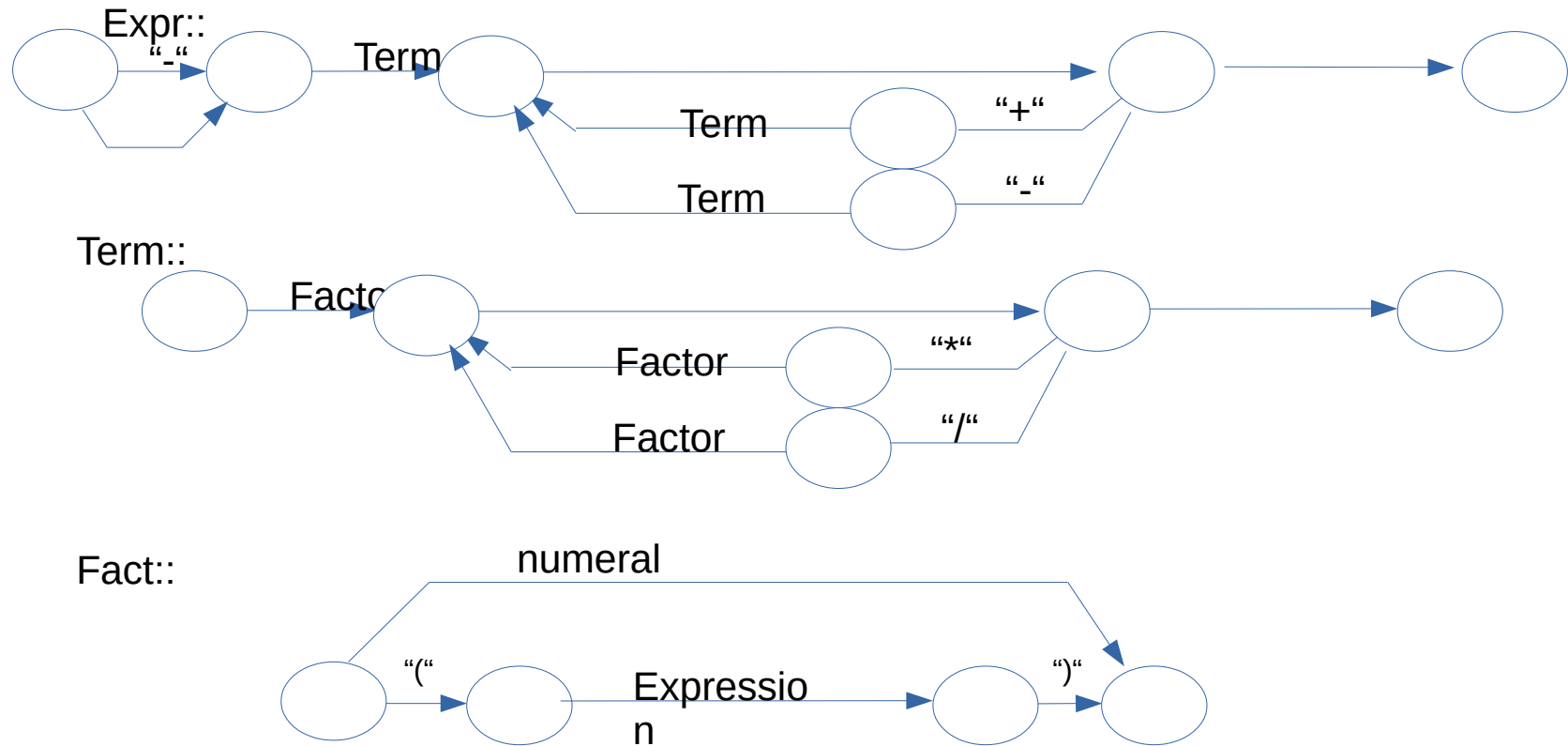
	First	Follow
expression	Zahl, (\$,)
rexpr	+, ϵ	\$,)
term	Zahl, (\$,), +
rterm	*, ϵ	\$,), +
factor	Zahl, (\$,), +, *

Implementation dazu

```
double expr(void)
{
    double tmp=term();
    return Rexpr(tmp);
}
```

```
double Rexpr(double tmp)
{
    if (m.cval=='-' && m.mc==mop )
        {lex(NULL); tmp -=term();tmp=Rexpr(tmp);} else
    if (m.cval=='+' && m.mc==mop )
        {lex(NULL); tmp +=term();tmp=Rexpr(tmp);}
    return tmp;
}
```

Darstellung durch Syntaxgraphen



Begriffe (Wiederholung)

Sprache	Menge aller gültigen Sätze zu einem Alphabet
Alphabet	terminales / nicht terminales
terminales A.	Zeichen aus denen die Sätze der Sprache bestehen
nicht terminales A.	Hilfszeichen zum Bilden von Regeln
Zeichenkette	Aneinanderreihung von Zeichen aus A
#s	Länge der Zeichenkette
Ableitung	$s \Rightarrow t$, wenn $s \rightarrow s_1 \rightarrow s_2 \rightarrow \dots \rightarrow s_n \rightarrow t$
direkte Abl.	$s = s_1 s_2 s_3$, $t = s_1 t_2 s_3$, $s_2 \rightarrow t_2$
Grammatik	$G = (T, N, s, R)$