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 → 6
2. 12/6 → 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 → Xa | Xb	StatementList → statement ';' statementList
	| statement
A → X A'	StatementList → statement statementList2
A' → a | b	StatementList2 → ';' statement statementList2
	| ε

Grammatik für einfache Ausdrücke

EBNF (3)

Beseitigung der Linksrekursion durch Iteration

N : faktor, term, termx, expression, exprx
T : Zahl,'+','*','(',')','-','/'
s : expression
R : expression ::= term exprx
exprx ::= {('+' | '-') term}
term ::= faktor termx
termx ::= {('*' | '/') faktor}
faktor ::= Zahl | '(' expression ')'

Unterschiedliche Terminale Anfänge der Alternativen

{} wiederholung (Iteration), durch Schleife umgesetzt.

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 → A a | b	Expr → Expr '+' Term | Term	A=Expr
		a='+' Term
		b=Term
A → b A'	Expr → Term Rexpr	
A' → a A'|ε	Rexpr → '+' Term Rexpr | ε	

Grammatik für einfache Ausdrücke (G4)

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

First/Follow

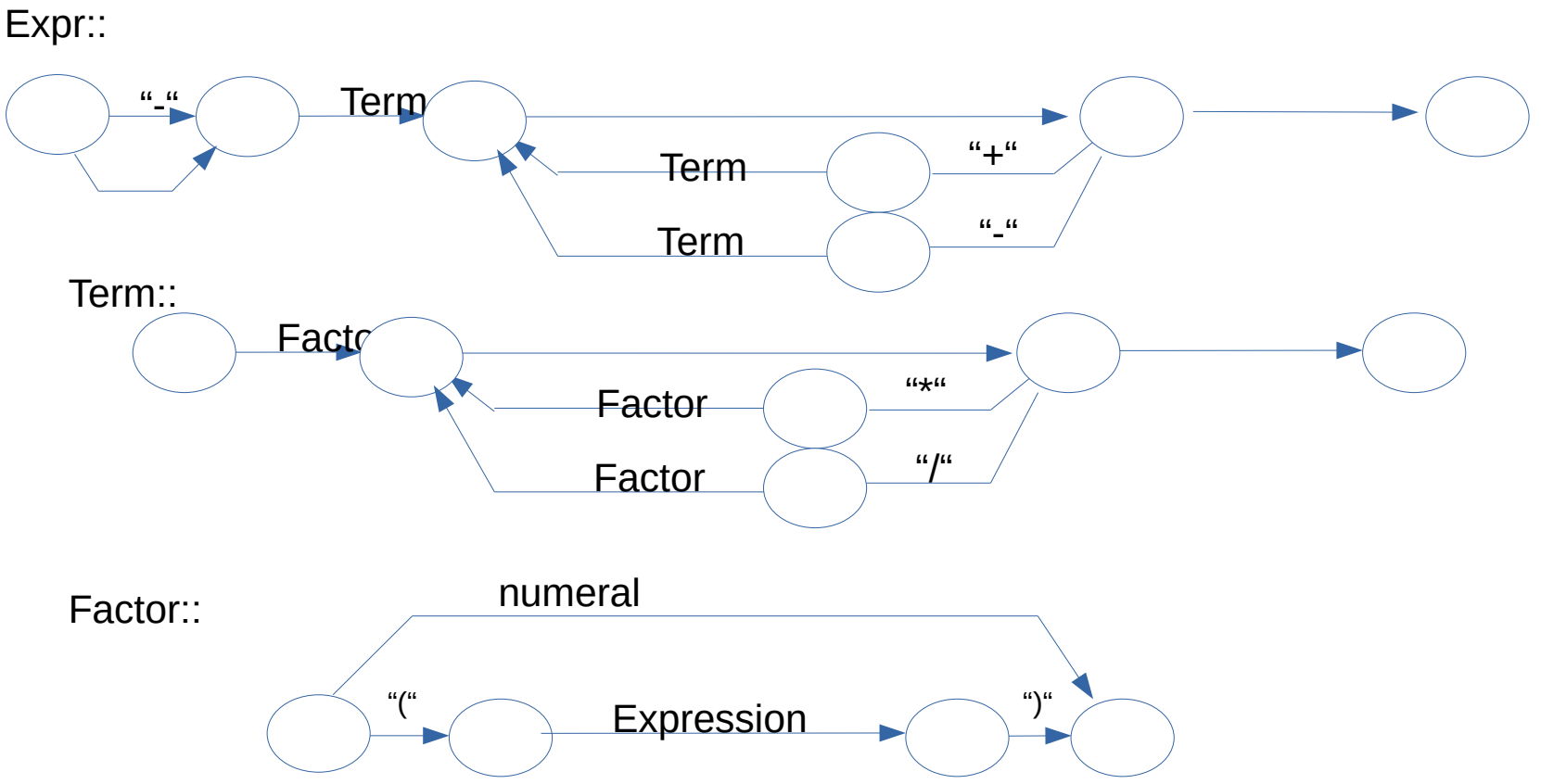
	First	Follow
expression	Zahl, (	$,)
rexpr	+, ε	$,)
term	Zahl, (	$,), +
rterm	*, ε	$,), +
factor	Zahl, (	$,), +, *

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=>t, wenn s->s1->s2-> … sn->t
direkte Abl.	s= s1 s2 s3, t= s1 t2 s3, S2 -> t2
Grammatik	G = (T,N,s,R)

12