9 Syntaxanalyse

Formale Grundlagen der Informatik I

Herbstsemester 2012

Robert Marti

Vorlesung teilweise basierend auf Unterlagen
von Prof. emer. Helmut Schauer

Einleitung: Definition und Struktur formaler Sprachen

Einer Sprache liegt ein **Vokabular** (*vocabulary*) [manchmal auch Alphabet oder Zeichenvorrat genannt] zugrunde:
Die Menge der Worte bzw. **Symbole** (*symbols*) [auch Zeichen genannt].

Aus den Symbolen können **Symbolfolgen** gebildet werden:

- korrekte bzw. **wohlgeformte** (well-formed) Folgen von Symbolen
- falsche bzw. missgebildete Folgen

Die **Grammatik** (*grammar*), auch **Syntax** genannt, legt fest,
welche Symbolfolgen korrekt sind.

⇒ Beschreibung der Menge der zulässigen Symbolfolgen einer Sprache

⇒ Definition einer Struktur der Sprache, die hilft, die Bedeutung (**Semantik**) zulässiger Symbolfolgen zu erkennen.

Eine Sprache ist durch Syntax und Semantik ihrer Texte definiert.

Beispiel einer einfachen Grammatik

Notation: eine Variante der Backus-Naur Form (BNF), die zur Definition der Programmiersprache Algol 60 verwendet wurde (Details später).

Satz = *Subjekt* *Praedikat* .

Subjekt = "Hunde" .

Subjekt = "Katzen" .

Praedikat = "essen" .

Praedikat = "schlafen" .

Zeile 1: 'Ein Satz besteht aus einem Subjekt gefolgt von einem Prädikat.'
Zeile 2: 'Das Subjekt besteht aus dem Wort "Hunde".'

Elemente einer Sprache

N Menge von **Nichtterminalsymbolen**
die syntaktischen Einheiten, z.B.
Sentence, Verb bzw. *Statement , Assignment, Expression*

T Menge von **Terminalsymbolen**
die Worte bzw. Symbole, die in einem Text vorkommen dürfen, z.B.
“woman”, “sing” bzw. "while", "if", "(", ")", ">=", "++")

P Menge von (kontextfreien) **Grammatikregeln** der Form

$$\alpha = \beta_1 \ldots \beta_n \quad (\alpha \in N,\ \beta_i \in N \cup T)$$

z.B.
Sentence = *NounPhrase VerbPhrase* .
Statement = "while" "(" *Condition* ")" *Statement* ";" .
[Grammatikregeln werden auch als **Produktionen** bezeichnet]

S **Startsymbol** ($S \in N$), die syntaktische Einheit der “höchsten Stufe”, z.B.
Sentence (bzw. Program, Class)

Beispiel: Nichtterminalsymbole für Englisch

Determiner	Artikel	the, some, most
Noun	Substantiv	
Verb	Verb	
Adjective	Adjektiv	big, fast
Adverb	Adverb	today, rarely, rather, very
Auxiliary	Hilfsverb	has, will, is, are, must, should
Conjunction	Konjunktion	and, or
Preposition	Präposition	to, on, with, in
Pronoun	Pronomen	he, who, which, his
Sentence	Satz	
NounPhrase	Nominalphrase (Subjekt, inkl. Artikel, Adjektive, ...)	
VerbPhrase	Verbphrase Prädikat, inkl. direkte/indirekte Objekte	
PrepositionalPhrase	Präposition gefolgt von Nominalphrase	

Herleitung von Symbolfolgen einer Sprache

Die Sprache $\boldsymbol{L}(\boldsymbol{T}, \boldsymbol{N}, \boldsymbol{P}, S)$ ist die Menge der Terminalsymbolfolgen $\boldsymbol{T}^*$, die aus dem Startsymbol S hergeleitet werden können:

$$\boldsymbol{L} = \{ \xi \in \boldsymbol{T}^* \mid S \rightarrow \xi \}$$

Eine Symbolfolge $\sigma_n \in (\boldsymbol{N} \cup \boldsymbol{T})^*$ kann aus einer Symbolfolge σ_0 **hergeleitet** werden, wenn und nur wenn es eine Folge $\sigma_1, \sigma_2, \ldots, \sigma_{n-1}$ gibt, so dass jedes σ_i aus σ_{i-1} $(1 \leq i \leq n)$ direkt hergeleitet werden kann.

$$\sigma_0 \rightarrow \sigma_n \quad \equiv \quad \sigma_{i-1} \rightarrow \sigma_i \ (1 \leq i \leq n)$$

Eine Symbolfolge $\eta = \alpha\eta'\beta$ kann aus einer Symbolfolge $\xi = \alpha\xi'\beta$ **direkt hergeleitet** werden, wenn und nur wenn

$$\xi' = \eta' \in \boldsymbol{P}$$

Bemerkung: Wenn $\eta = \eta'$ und $\xi = \xi'$ (d.h. wenn α und β die leere Folge ε sind), dann ist die Produktion kontextfrei.

Beispiel: Herleitung in einer einfachen Grammatik

Grammatik

Satz = *Subjekt* *Praedikat* .
Subjekt = "Hunde" .
Subjekt = "Katzen" .
Praedikat = "essen" .
Praedikat = "schlafen" .

Behauptung

"Hunde schlafen" ist ein Element der Sprache bzw. *Satz* → "Hunde" "schlafen":

Herleitung

Satz → *Subjekt* *Praedikat* → "Hunde" *Praedikat* → "Hunde" "schlafen" .

Alternative Herleitung:

Satz → *Subjekt* *Praedikat* → *Subjekt* "schlafen" → "Hunde" "schlafen" .

Weiteres Beispiel: Arithmetische Ausdrücke (1)

1 *Expr* = *Term* .
2 *Expr* = *Term* *AddOp* *Expr* .

3 *AddOp* = "+" .
4 *AddOp* = "−" .

5 *Term* = *Factor* .
6 *Term* = *Factor* *MulOp* *Term* .

7 *MulOp* = "*" .
8 *MulOp* = "/" .

9 *Factor* = *Var* .
10 *Factor* = *Const* .
11 *Factor* = "(" *Expr* ")" .

12 *Var* = *Letter* .

13 *Const* = *Digit* .
14 *Const* = *Digit* *Const* .

15 *Letter*: Die Buchstaben "a" bis "z".

16 *Digit*: Die Ziffern "0" bis "9".

Weiteres Beispiel: Arithmetische Ausdrücke (2)

Behauptung

"2*x" ist eine *Expr* .

Herleitung

	Expr
(1)	*Term*
(6)	*Factor MulOp Term*
(10)	*Const MulOp Term*
(13)	*Digit MulOp Term*
(16)	"2" *MulOp Term*
(7)	"2" "*" *Term*
(5)	"2" "*" *Factor*
(9)	"2" "*" *Var*
(12)	"2" "*" *Letter*
(15)	"2" "*" "x"

Elemente der Syntaxdefinition

Syntaxdefinitionen enthalten bekannte Elemente, die auch in Programmiersprachen vorkommen:

Sequenz
Beispiel:
11 *Factor* = "(" *Expr* ")" .

Alternative
Beispiel:
7 *MulOp* = "*" .
8 *MulOp* = "/" .

Iteration (als Paar von rekursiven Produktionen repräsentiert)
Beispiel:
5 *Term* = *Factor* .
6 *Term* = *Factor* *MulOp* *Term* .

EBNF: Extended Backus-Naur Form (1)

Die erweiterte Backus-Naur Form (Extended Backus-Naur Form, EBNF) führt zusätzliche Metasymbole zur kompakten Darstellung von **Alternativen** (siehe unten) und Iterationen (siehe nächste Folie).

Alternative

2 verschiedene Symbolfolgen

$\alpha \mid \beta$ entweder Symbolfolge α oder Symbolfolge β

Beispiel:

MulOp = "*" | "/" .

1 optionale Symbolfolge

[α] Symbolfolge α kommt 0 oder 1 vor

Abkürzung für: $\alpha \mid \varepsilon$, wobei ε die leere Symbolfolge ist.

Beispiel:

Factor = *Identifier* ["(" *ParameterList* ")"] .

Bem: Die obige Produktion war nicht Teil des Beispiels auf Folie 8.

EBNF: Extended Backus-Naur Form (2)

Zusätzliche Metasymbole zur kompakten Darstellung von **Iterationen**.

Iteration

{ α } Symbolfolge α kommt 0, 1 oder mehrmals vor

Beispiele:

Term = *Factor* { *MulOp* *Factor* } .

ParameterList = [*Expr* { "," *Expr* }] .

Bem: Die zweite Produktion war nicht Teil des Beispiels auf Folie 8.

Ausserdem werden Klammern benötigt:

α β | γ δ bedeutet: Sequenz α β oder Sequenz γ δ

α (β | γ) δ bedeutet: Sequenz α β δ oder Sequenz α γ δ

Zuasmmenfassung: Syntax und Semantik der EBNF

x	Nonterminal
"x"	Terminal
x y	Sequenz
x | *y*	Alternative
{ *x* }	Iteration
(*x*)	Bindung
[*x*]	Option
y = *x* .	Definition

Arithmetische Ausdrücke in EBNF definiert

1 *Expr* = *Term* { ("+" | "−") *Term* } .

2 *Term* = *Factor* { ("*" | "/") *Factor* } .

3 *Factor* = *Var* | *Const* | "(" *Expr* ")" .

4 *Var* = *Letter* . *)

5 *Const* = *Digit* { *Digit* } .

6 *Letter* = "a" | "b" | "c" | … | "z" .

7 *Digit* = "0" | "1" | "2" | … | "9" .

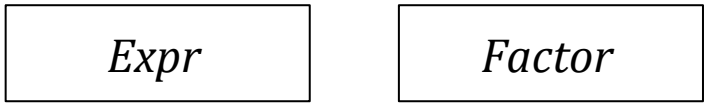
*) Bezeichner für Variablem (Var) können in den meisten Sprachen aus mehreren Zeichen besstehen, z.B.:

Var = *Letter* { *Letter* | *Digit* } .

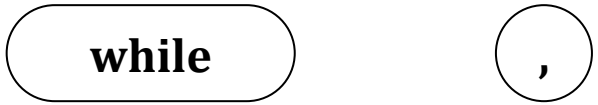
Syntaxdiagramme

Syntaxdiagramme dienen der *graphischen Beschreibung* der Syntax formaler Sprachen und sind eine leichter verständliche Alternative zu Produktionen in Formalismen wie EBNF und ähnlichen.

Nichtterminalsymbole werden als Rechtecke dargestellt:



Terminalsymbole werden als Ovale (respektive Kreise) dargestellt:



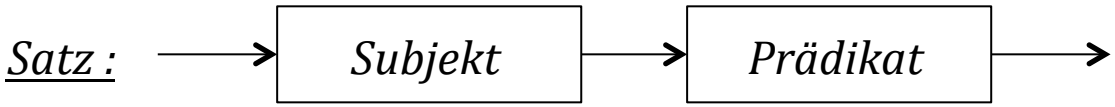
Grundelemente von Syntaxdiagrammen sind Sequenz, Alternative und Iteration.

Sequenz

Sequenzen werden durch Verbinden der einzelnen Nichtterminal- und Terminalsymbole dargestellt.

Beispiele:

Satz = *Subjekt Prädikat*.



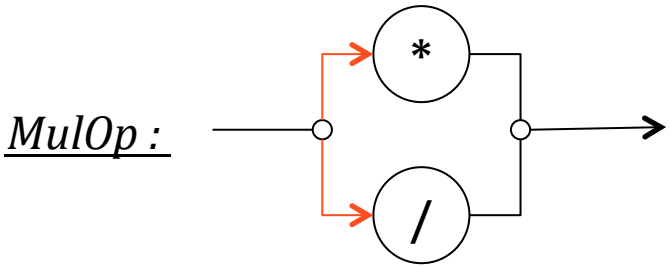
Factor = "(" *Expr* ")" .

Alternative

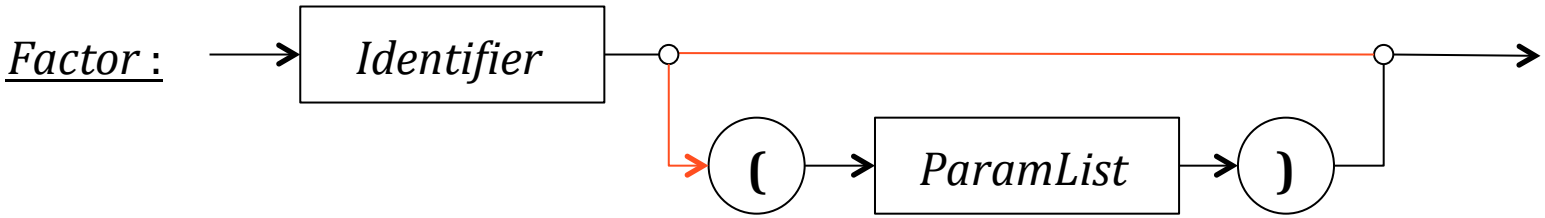
Alternativen werden durch Verzweigungen dargestellt.

Beispiele:

MulOp = "*" | "/" .



Factor = *Identifier* ["(" *ParamList* ")"] .

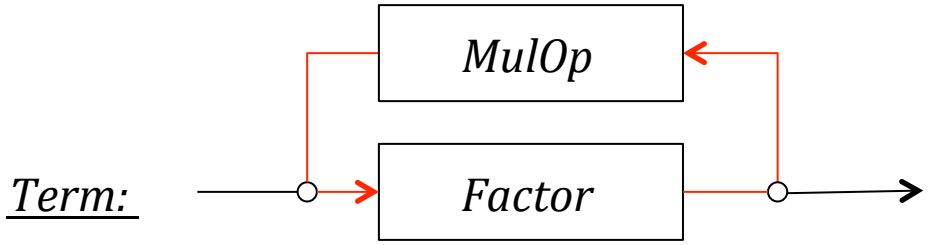


Iteration

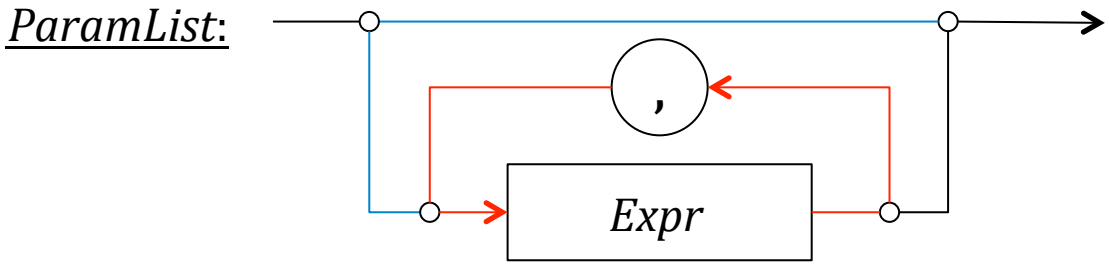
Iterationen werden durch Schleifen dargestellt.

Beispiele:

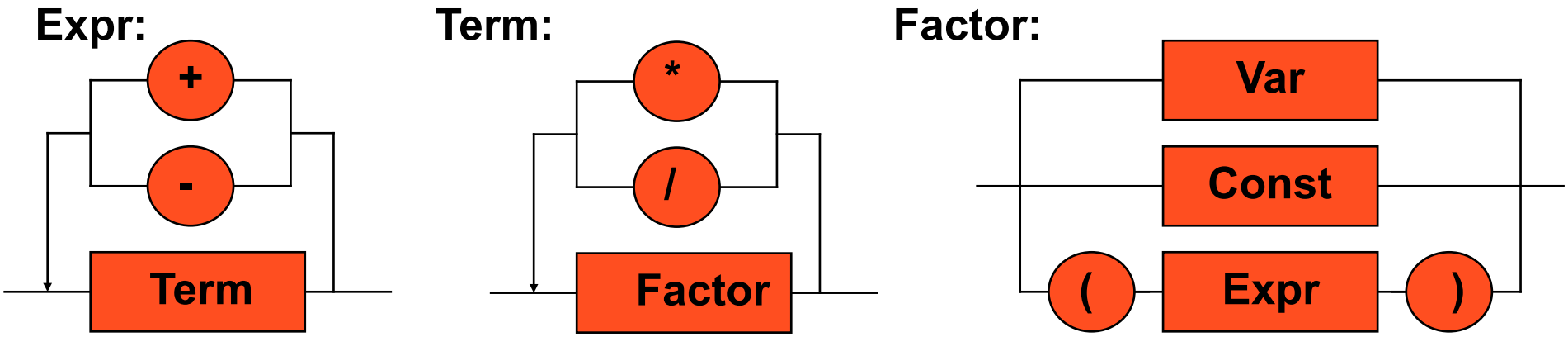
Term = Factor { MulOp Factor } .



ParamList = [Expr { "," Expr }] .



Arithmetische Ausdrücke als Syntaxdiagramme



zB (a+1)*b

Syntaxanalyse und Syntaxbaum (engl. Parse Tree)

Die **Syntaxanalyse** stellt fest, ob ein Text syntaktisch richtig ist oder nicht.

z.B. $-1 / (x+y)$ ist gemäss Grammatik auf der vorherigen Seite keine syntaktisch richtige *Expr*

Die Aussage, dass ein natürlichsprachlicher Text oder ein Programm (bzw. Programmfragment) syntaktisch korrekt ist, reicht im allgemeinen nicht:

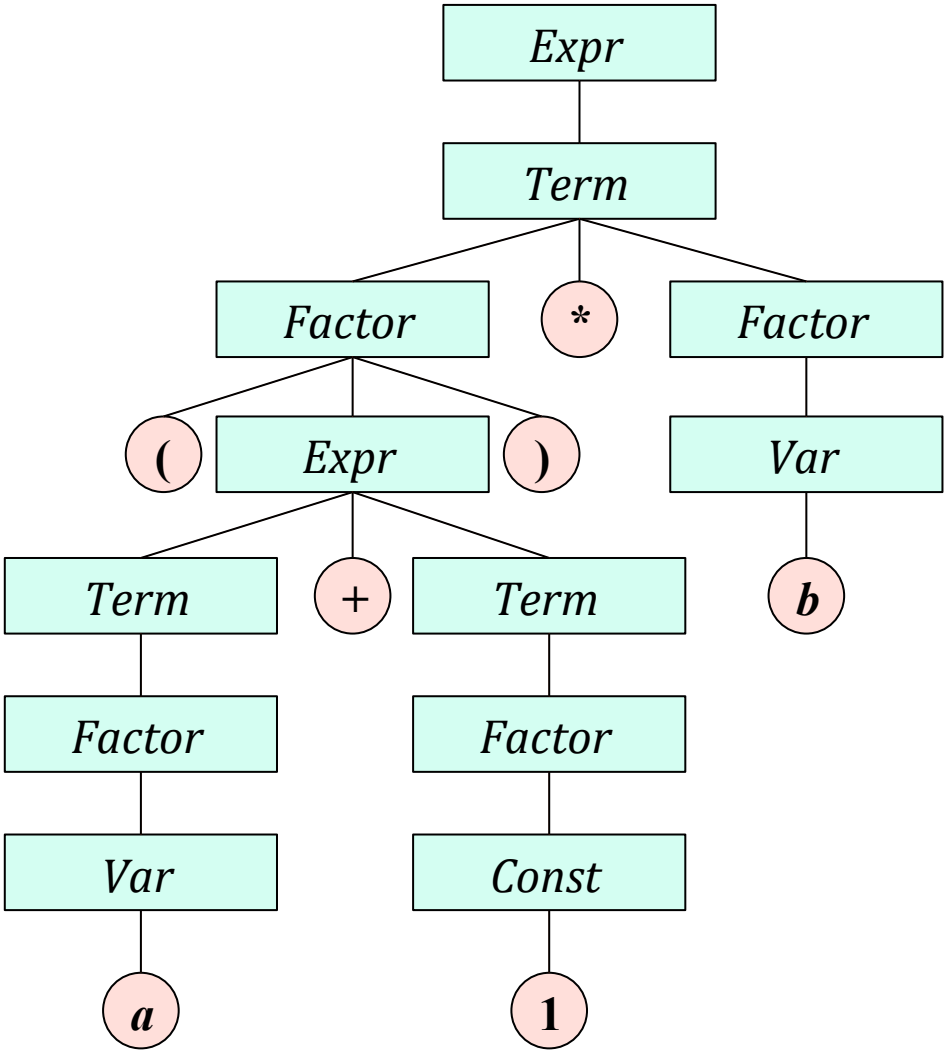
Ein Programm soll auch ausgeführt werden!

Die Struktur eines syntaktisch richtigen Textes kann durch einen **Syntaxbaum** (engl. *parse tree*) dargestellt werden.

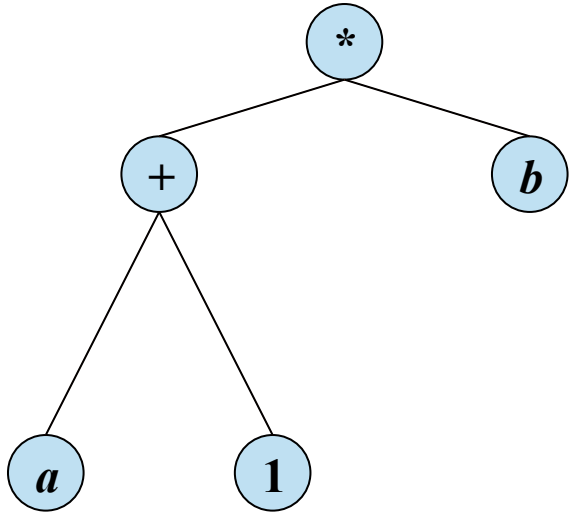
Der Parsebaum kann z.B. als Input für eine ausführende Maschine verwendet werden.

Beispiel eines Parse Trees

Parsebaum für $(a+1)*b$

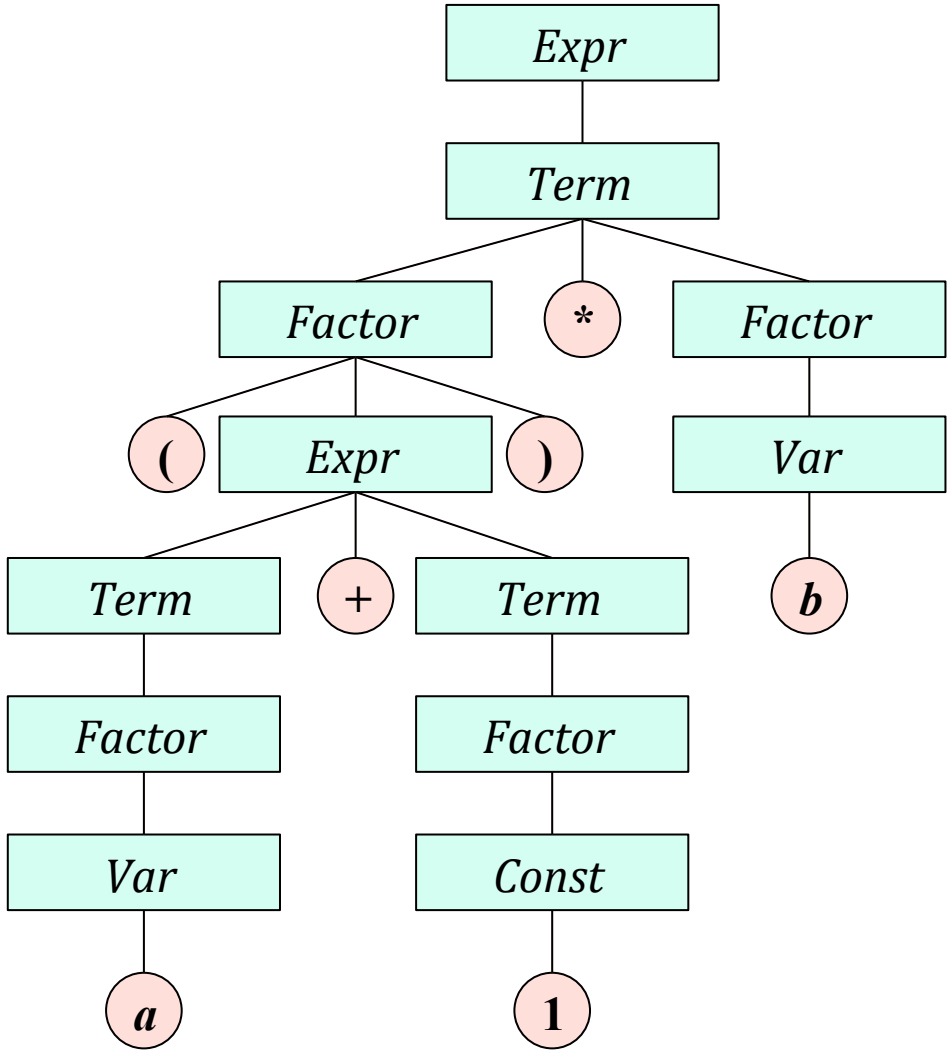


Operatorbaum für $(a+1)*b$



Aufbau des Parse Trees

Parsebaum für $(a+1)*b$



Syntax

```
1  Expr = Term { ( "+" | "−" ) Term } .
2  Term = Factor { ( "*" | "/" ) Factor } .
3  Factor = Var | Const | "(" Expr ")" .
4  Var = Letter .
5  Const = Digit { Digit } .
6  Letter = "a" | "b" | "c" | … | "z" .
7  Digit = "0" | "1" | "2" | … | "9" .
```

Schreiben einer Parsers (Rough Sketch)

```
class Parser {
static Symbol sym;                          // current symbol
static Symbol getsym() { ... }              // func returning next symbol
static void Expr(...) {
    Term(...);
    while ( sym ∈ {+, −} ) {
        sym = getsym();
        Term(...);
    }
}
static void Factor(...) {
    if ( sym ∈ {a, b, c, ... , z} ) {       // a var!
        ...
    } else if ( sym ∈ {0, 1, 2, ... , 9} ) {  // a const!
        ...
    } else if ( sym == ´(´ ) {              // a parenthesized expr!
        sym = getsym();
        Expr(...);
        if ( sym == ´)´ ) sym = getsym();
        else {  syntax error, raise hell!  }
    }
}
}
```

Syntax

1 *Expr* = *Term* { ("+" | "−") *Term* } .
3 *Factor* = *Var* | *Const* | "(" *Expr* ")" .
4 *Var* = *Letter* .
5 *Const* = *Digit* { *Digit* } .
6 *Letter* = "a" | "b" | "c" | … | "z" .
7 *Digit* = "0" | "1" | "2" | … | "9" .