Äquivalente Grammatiken / attributierte Grammatik

- Linksfaktorisierung
- Elimination von Linksrekursion
- Umwandlung von EBNF in BNF
- Attributierte Grammatik
- Semantikfunktionen und Übersetzungsschema
- Synthetisierte, ererbte und L-attributierte Grammatik

Linksfaktorisierung

Eingabe: Eine Grammatik G.

Ausgabe: Eine äquivalente links-faktorisierte Grammatik.

Methode: Bestimme für jedes Nichtterminal A dasjenige α, das längstes gemeinsames Präfix zweier oder mehrerer Alternativen von A ist. Wenn es ein nichttriviales gemeinsames Präfix gibt ($\alpha \neq \varepsilon$,), ersetze alle A-Produktionen

$$A \rightarrow \alpha\,\beta 1 \mid \alpha\,\beta 2 \mid \ldots \mid \alpha\,\beta n \mid \gamma$$

(wobei γ für alle nicht mit α beginnenden Alternativen steht), durch

$$A \rightarrow \alpha\,A' \mid \gamma$$
$$A' \rightarrow \beta 1 \mid \beta 2 \mid \ldots \mid \beta n$$

Dabei ist A' ein neues Nichtterminal. Wiederhole diese Transformation so lange, bis es für kein Nichtterminal mehr Alternativen mit gemeinsamem Präfix gibt.

Beispiel für Linksfaktorisierung

```
stmt    →       if expr then stmt
        |       if expr then stmt else stmt
        |       other
```

Faktorisierung:

```
stmt    →       if expr then stmt else-teil
        |       other
else-teil → else stmt | ε
```

(In diesem Fall ist die faktorisierte Grammatik immer noch mehrdeutig)

Elimination von Links-Rekursion

ersetze Grammatik-Regeln der Form

$A \rightarrow A\ \alpha$

$A \rightarrow \beta$

durch:

$A \rightarrow \beta\ A'$

$A' \rightarrow \alpha\ A'$

$A' \rightarrow \varepsilon$

Beispiel:

$E \rightarrow E + T$ $\quad\quad$ $A = E,\ \alpha = + T$

$E \rightarrow T$ $\quad\quad$ $\beta = T$

ergibt:

$E \rightarrow T\ R$ $\quad\quad$ $A' = R$

$R \rightarrow + T\ R$

$R \rightarrow \varepsilon$

Vergleich EBNF / BNF

Die etwas einfacher aussehenden Regeln der EBNF lassen sich einfach in BNF umsetzen, indem einfach für komplexe EBNF-Konstrukte neue Regeln definiert werden (das zeigt auch den formalen Zusammenhang von EBNF und BNF):

$E \rightarrow T\ (+\ T\ )^*$

ergibt:

$E \rightarrow T\ R$
$R \rightarrow +\ T\ R$
$R \rightarrow \varepsilon$

Idee: $R \rightarrow (\ +\ T\ )^*$

Attributierte Grammatiken

- Ordne jedem Grammatiksymbol ein *Attribut* zu
- Die Attribute der Terminalsymbole werden durch den Scanner definiert
- Die Attribute der Nichtterminalsymbole werden durch *Semantikfunktionen* definiert.
- Semantische Funktionen mit Seiteneffekt heißen *Übersetzungsschema*

Beispiel 1 (synthetisierte Attribute):

		// Übersetzungsschema
E:a $\rightarrow$ E:b + T:c	// a = b + c	`// System.out.println("+");`
E:a $\rightarrow$ T:b	// a = b	`// System.out.println(b);`

Beispiel 2 (ererbte Attribute):

E:a $\rightarrow$ T:b R:{c,d}	// a = d; c = b
R:{a,b} $\rightarrow$ + T:c R:{d,e}	// b = e; d = a + c
R:{a,b} $\rightarrow \varepsilon$	// b = a;

Weitere Beispiele

1. L-attributiert (ererbt von links nach rechts)

```
decl → type:a varlist:b                  // b = a
varlist:a → var:b                                  // declare(b, a)
varlist:a → var:b COMMA varlist:c   // declare(b, a); c = a
type:a → INT                                // a = INT_TYPE
type:a → FLOAT                                   // a = FLOAT_TYPE
```

2. Nicht L-attributiert

```
decl → VAR varlist:a COLON type:b                // a = b
```

3. S-attributiert (synthetisiert)

```
decl → type:a varlist        :b                  // declareAll(b, a)
varlist:a → var:b                                  // a = new List(b)
varlist:a → var:b COMMA varlist:c   // a = prepend(b, c)
```

Mögliche Übersetzungsaktionen

- Direkte Auswertung (Kommandosprache, arithmetische Ausdrücke)
- Direkte Übersetzung (einfacher Bytecode, einfache Quelltextänderung)
- Speicherung der Information in Datenbank (z.B. XML-Parser)
- Speicherung von Information in Symboltabellen (Deklarationen)
- Erzeugung eines abstrakten Syntaxbaums

Ein abstrakter Syntaxbaum (+ Symboltabelle) erfordert eine nachfolgende Weiterverarbeitung (Typprüfung, Optimierung, Auswertung, Codegenerierung)

EBNF-erweitert um Auswertung von Ausdrücken

Die Grundidee einer Erweiterung einer Syntaxerkennung um Übersetzungsaktionen besteht darin, zunächst den Syntaxsymbolen Attribute, d.h. Werte zuzuordnen (String, Zahl, Objekt) und dann in die Syntaxregeln Programmiersprachkonstrukture einzubetten. Bei einem LL(1) Parser ist die richtige Abfolge der Aktionen immer gewährleistet, da die Abarbeitung der Regeln streng deterministisch (innerhalb einer Regel von links nach rechts) verläuft.

Die genaue Form der Formulierung hängt bei LL(1) Generatoren vom verwendeten Werkzeug ab. Bei rekursivem Abstieg, wird einfach, das Programm zur Syntaxerkennung geeignet erweitert.

Da die Symbolerkennung beim Scanner ihren Ausgangspunkt nimmt, muss natürlich bereits der Scanner die nötigen Informationen (Zahlenwerte, Strings) weitergeben.

Auf einer etwas theoretischeren Ebene bezeichnet man eine entsprechend erweiterte Grammatik als attributierte Grammatik (BNF+Regeln am Ende) bzw. in der gerade besprochenen relativ freien Form als Übersetzungsschema.

In dem folgenden Beispiel werden Attrribute (im rekursiven Abstieg) rekursiv verarbeitet.

EBNF-erweitert um Auswertung von Ausdrücken

```
expression → term:s ( + term:t {s+=t;} )* {RESULT=s;}
term → z:z {RESULT=z;} | ( expression:e ) {RESULT=e;}

double expression() {
    double sum = term();
    while (lookahead() == PLUS) {     // ( + term )*
        match(PLUS); sum += term();
    }
    return sum;
}

double term() {
    if (lookahead() == ZAHL) {
        match(ZAHL); return tokenValue();
    } else if (lookahead() == KLAUF) {
        match(KLAUF);
        double returnValue = expression();
        match(KLZU);
        return returnValue;
    } else
        throw new SyntaxException();
}
```