Kontext–freie Grammatik

Definition: Eine *kontext–freie* Grammatik G ist ein 4–Tupel der Form

$$G = \langle T, N, R, S \rangle$$

wobei die Bedeutung der Komponenten wie folgt ist:

1. T ist die Menge der *Terminale*

2. N ist die Menge der *Nicht–Terminale*

   Die Vereinigung von T und N wird mit V bezeichnet:

   $$V := T \cup N$$

   Die Menge V heißt auch das *Vokabular*.

3. $R \subseteq N \times V^*$

   ist die Menge der *Regeln*:

   (a) Die erste Komponente ist ein Nicht–Terminal.

   (b) Die zweite Komponente ist ein Wort aus Nicht–Terminalen und Terminalen.

   Statt $\langle X, \alpha \rangle \in R$ schreiben wir

   $$X \to \alpha.$$

4. $S \in N$ ist das Start–Symbol.

Bedeutung: Grammatiken werden zur Beschreibung von Programmier–Sprachen benutzt.

Beispiel: Arithmetische Ausdrücke

Beschreibung arithmetischer Ausdrücke möglich durch

$$G_{\texttt{arith}} := \langle T, N, R, \mathit{Expr} \rangle$$

1. $T = \{\mathbf{number}, \mathbf{variable}, \text{“+”}, \text{“−”}, \text{“*”}, \text{“/”}, \text{“(”}, \text{“)”}\}$
2. $N = \{\mathit{Expr}\}$
3. Die Menge R enthält folgende Regeln

$$\begin{array}{rcl} \mathit{Expr} & \rightarrow & \mathit{Expr}\ \text{“+”}\ \mathit{Expr} \\ & | & \mathit{Expr}\ \text{“−”}\ \mathit{Expr} \\ & | & \mathit{Expr}\ \text{“∗”}\ \mathit{Expr} \\ & | & \mathit{Expr}\ \text{“/”}\ \mathit{Expr} \\ & | & \text{“(”}\ \mathit{Expr}\ \text{“)”} \\ & | & \mathbf{number} \\ & | & \mathbf{variable} \end{array}$$

Es gibt zwei Arten von Terminalen

1. *Wörtliche Terminale* stehen für sich selbst.

   Beispiel: “**+**”, “**-**”, “*****”, “**-**”, “(”, “)”

   Werden in Regeln durch “ und ” abgegrenzt.

2. *Token–Terminale* beschreiben Klassen von Strings und werden durch reguläre Ausdrücke implementiert

   Beispiel: **number**, **variable**

   werden in Beispielen **fett** gesetzt, klein geschrieben

Nicht–Terminale werden *schräg* gesetzt, groß geschrieben

Beispiel: *Expr*

Sprache einer Grammatik

Gegeben: Grammatik $G = \langle T, N, R, S \rangle$ mit $V = T \cup N$.

Definition: Falls

1. $X \in N$,
2. $\alpha, \beta, \gamma \in V^*$,
3. $(X \to \beta) \in R$

ist, so sagen wir, dass

$$\alpha X \gamma \to \alpha \beta \gamma$$

ein *Ableitungs–Schritt* ist.

Die *transitive Hülle* von $\to$ bezeichen wir mit $\to^*$:
Für $\alpha, \beta, \gamma \in V^*$ gilt also:

1. $\alpha \to^* \alpha$
2. Falls $\alpha \to \beta$ und $\beta \to^* \gamma$, so folgt $\alpha \to^* \gamma$

Sprechweise: Falls $\alpha \to^* \gamma$ ist, sagen wir

α *wird zu* γ *reduziert*

Beispiel:

$$\begin{array}{lcl} \textit{Expr} & \to & \textit{Expr} \text{ “+” } \textit{Expr} \\ & \to & \textbf{number} \text{ “+” } \textit{Expr} \\ & \to & \textbf{number} \text{ “+” } \textbf{number} \end{array}$$

Definition: Die Sprache $\mathcal{L}(G)$ ist

$$\mathcal{L}(G) := \{\alpha \in T^* \mid S \to^* \alpha\}$$

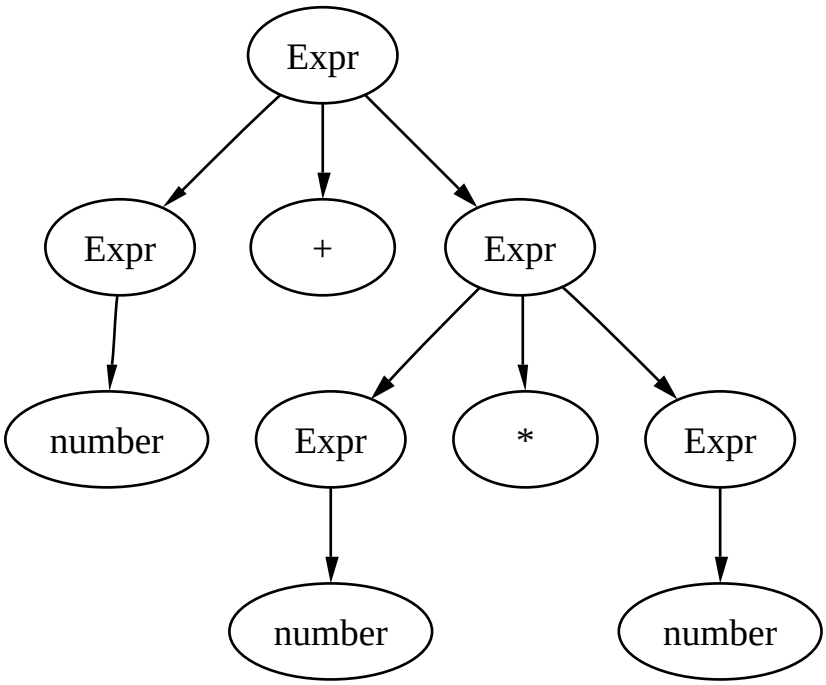
Parse–Bäume

Ein Wort aus $\mathcal{L}(G_{\texttt{arith}})$ kann auf mehrere Weisen abgeleitet werden:

1. "Richtige" Ableitung

$$
\begin{aligned}
\textit{Expr} &\rightarrow \textit{Expr}\ \text{“+”}\ \textit{Expr} \\
&\rightarrow \textbf{number}\ \text{“+”}\ \textit{Expr} \\
&\rightarrow \textbf{number}\ \text{“+”}\ \textit{Expr}\ \text{“}*\text{”}\ \textit{Expr} \\
&\rightarrow \textbf{number}\ \text{“+”}\ \textbf{number}\ \text{“}*\text{”}\ \textit{Expr} \\
&\rightarrow \textbf{number}\ \text{“+”}\ \textbf{number}\ \text{“}*\text{”}\ \textbf{number}
\end{aligned}
$$

   Zugehöriger Parse–Baum:



   Entspricht dem Abarbeiten eines arithmetischen Ausdrucks der Form

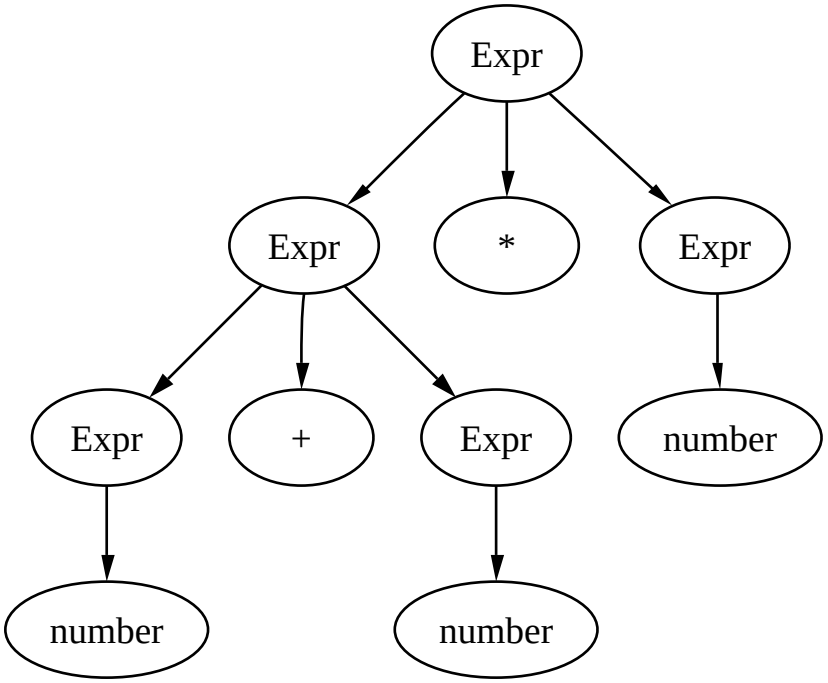
$$x + y * z$$

Parse–Bäume (Fortsetzung)

1. "Falsche" Ableitung

$$
\begin{aligned}
\textit{Expr} &\rightarrow \textit{Expr}\ \text{“}*\text{”}\ \textit{Expr} \\
&\rightarrow \textit{Expr}\ \text{“}+\text{”}\ \textit{Expr}\ \text{“}*\text{”}\ \textit{Expr} \\
&\rightarrow \textit{number}\ \text{“}+\text{”}\ \textit{Expr}\ \text{“}*\text{”}\ \textit{Expr} \\
&\rightarrow \textit{number}\ \text{“}+\text{”}\ \textit{number}\ \text{“}*\text{”}\ \textit{Expr} \\
&\rightarrow \textit{number}\ \text{“}+\text{”}\ \textit{number}\ \text{“}*\text{”}\ \textit{number}
\end{aligned}
$$

Zugehöriger Parse–Baum:



Abgeleitet wurde $x + y * z$

Parse–Baum entspricht aber

$(x + y) * z$

Problem: Grammatik $G_{\texttt{arith}}$ ist *mehrdeutig*!

Grammatik für reguläre Ausdrücke

Beschreibung regulärer Ausdrücke möglich durch

$$G_{\texttt{RegExp}} = \langle T, N, R, \mathit{Expr} \rangle$$

1. $T = \{\mathbf{letter}, \mathbf{escape}, \text{“+”}, \text{“}*\text{”}, \text{“(”}, \text{“)”}\}$
2. $N = \{\mathit{Expr}, \mathit{Term}, \mathit{Factor}\}$
3. Die Menge R enthält folgende Regeln

$$\begin{array}{lcl}
\mathit{Expr} & \rightarrow & \mathit{Term}\ \text{“+”}\ \mathit{Expr} \\
& | & \mathit{Term} \\
\mathit{Term} & \rightarrow & \mathit{Factor}\ \mathit{Term} \\
& | & \mathit{Factor} \\
\mathit{Factor} & \rightarrow & \text{“(”}\ \mathit{Expr}\ \text{“)”}\ \text{“}*\text{”} \\
& | & \text{“(”}\ \mathit{Expr}\ \text{“)”} \\
& | & \mathbf{letter}\ \text{“}*\text{”} \\
& | & \mathbf{letter} \\
& | & \mathbf{escape}\ \text{“}*\text{”} \\
& | & \mathbf{escape}
\end{array}$$

Grammatik eindeutig! Wurde erreicht durch *Prioritäten*:

$$\mathit{Factor} > \mathit{Term} > \mathit{Expr}$$

entspricht Priorität der Operations–Symbole

1. Abschluß “*****”
2. Konkatenation
3. Alternative “**+**”

Beispiele für $\mathcal{L}(G_{\mathtt{RegExp}})$

Interpretation der Terminale

Sei Σ Menge aller Ascii–Zeichen.

1. **letter**: Alle Zeichen außer "+", "∗", "(", ")", "\", also

   $$\mathcal{L}(\mathbf{letter}) = \Sigma \setminus \{\text{“+”}, \text{“}*\text{”}, \text{“(”}, \text{“)”}, \text{“\textbackslash”}\}$$

2. **escape**: Wörter, die aus zwei Zeichen bestehen, wobei das erste Zeichen ein Backslash "\" ist:

   $$\mathcal{L}(escape) = \{c_1c_2 \in \Sigma^* \mid c_1 = \text{“\textbackslash”}\}$$

   Semantik:

   (a) \e steht für ε

   (b) $\backslash c$ mit $c \neq$ e steht für c
   notwendig, um mit regulären Ausdrücken nach

   "+", "∗", "(", ")", "\"

   suchen zu können

Beispiele für reguläre Ausdrücke

1. (a + \e)b∗

   Konventionelle Schreibweise: `a?b*`

2. /\ ∗ \\

   Slash "/, gefolgt von "∗", gefolgt von Backslash "\"

Recursive–Descent–Parser für $G_{\texttt{RegExp}}$

Die Regeln für *Expr*

$$\begin{array}{rcl} \textit{Expr} & \rightarrow & \textit{Term} \text{ “+” } \textit{Expr} \\ & | & \textit{Term} \end{array}$$

Vorgehen:

1. Parse *Term*, erhalte FSM `f1`.
2. Falls dananch “+” in Eingabe, parse `Expr`, erhalte FSM `f2`.

   Gebe `alternative(f1, f2)` zurück.
3. Sonst: Gebe `f1` zurück.

Implementierung:

```
// Globale Variable
char* charPtr;

// Vorab--Deklaration, notwendig wegen
// wechselseitiger Rekursion
FSM* parseTerm();
FSM* parseFactor();

FSM* parseExpr() {
    FSM* f1 = parseTerm();
    if (*charPtr == '+') {
        ++charPtr;
        FSM* f2 = parseExpr();
        return alternative(f1, f2);
    }
    return f1;
}
```

Parsen von *Term*

Die Regeln für *Term*:

$$\begin{array}{rcl} \textit{Term} & \rightarrow & \textit{Factor Term} \\ & | & \textit{Factor} \end{array}$$

Vorgehen:

1. Parse *Factor*, erhalte FSM `f1`

2. Falls Zeichen danach **erstes** Zeichen von *Term* sein kann, parse *Term*, erhalte FSM `f2`.

   Gebe `concat(f1, f2)` zurück.

3. Sonst: Gebe `f1` zurück

Implementierung:

```
FSM* parseTerm()
{
    FSM* f1 = parseFactor();
    if (*charPtr !=  0  &&
        *charPtr != '+' &&
        *charPtr != ')' &&
        *charPtr != '*'   )
    {
        FSM* f2 = parseTerm();
        return concat(f1, f2);
    }
    return f1;
}
```

Parsen von *Factor*

Die Regeln für *Factor*:

$$
\begin{array}{lcl}
\textit{Factor} & \rightarrow & \text{“(” } \textit{Expr} \text{ “)” “}*\text{”} \\
& | & \text{“(” } \textit{Expr} \text{ “)”} \\
& | & \textbf{letter “}*\text{”} \\
& | & \textbf{letter}
\end{array}
$$

Vorgehen:

1. Falls erstes Zeichen “(”:
   (a) Parse *Expr*, erhalte FSM `f`
   (b) Falls nächstes Zeichen “`*`”:
   Gebe `closure(f)` zurück
   (c) Sonst: Gebe `f` zurück.
2. Falls erstes Zeichen “\”:
   Erhöhe `charPtr`
   (a) Falls nächstes Zeichen “`e`”:
   erzeuge FSM `f` zum Erkennen von ε
   (b) Sonst: Gehe zu 3.
3. Sonst:
   (a) erzeuge FSM `f` zum
   Erkennen des Buchstabens `*charPtr`
   (b) Falls nächstes Zeichen “`*`”:
   Gebe `closure(f)` zurück
   (c) Sonst: Gebe `f` zurück.

Parsen von *Factor*: Implementierung

Implementierung des Parsens von *Factor*:

```
FSM* parseFactor()
{
    FSM* f;
    if (*charPtr == '(') {
        ++charPtr;
        f = parseExpr();
        ++charPtr;
    } else if (*charPtr == '\\') {
        ++charPtr;
        if (*charPtr == 'e') {
            f = createEmptyString();
            ++charPtr;
        } else {
            f = createCharacter(*charPtr);
            ++charPtr;
        }
    } else {
        f = createCharacter(*charPtr);
        ++charPtr;
    }
    if (*charPtr == '*') {
        ++charPtr;
        return closure(f);
    }
    return f;
}
```

Aufgaben

Aufgabe: Es sei das Alphabet $\Sigma = \{\text{“a”}, \text{“b”}\}$ gegeben. Geben Sie eine Grammatik G an, so dass gilt:

$$\mathcal{L}(G) = \{\mathtt{a}^m\mathtt{b}^{m+n}\mathtt{a}^n \mid n, m \in \mathbb{N} \wedge m \geq 1 \wedge n \geq 1\}$$

Lösung: $G = \left\langle \{\text{“a”}, \text{“b”}\}, \{S, A, B\}, R, S \right\rangle$ wobei R durch folgende Regeln gegeben ist

$$\begin{array}{lcl} S & \rightarrow & A\,B \\ A & \rightarrow & \text{“a”}\ A\ \text{“b”} \\ A & \rightarrow & \text{“a”}\ \text{“b”} \\ B & \rightarrow & \text{“b”}\ B\ \text{“a”} \\ B & \rightarrow & \text{“b”}\ \text{“a”} \end{array}$$

Aufgabe: Es sei die folgende Menge von Terminalen gegeben:

$$T = \{\mathbf{variable}, \text{“}\wedge\text{”}, \text{“}\vee\text{”}, \text{“}\neg\text{”}, \text{“(”}, \text{“)”}\}$$

Geben Sie eine Grammatik $G_{\texttt{Prop}}$ an, welche die Sprache der aussagenlogischen Formeln beschreibt.

Lösung: Die Grammatik R kann durch folgende Regeln definiert werden

$$\begin{array}{lcl} \mathit{Prop} & \rightarrow & \text{“}\neg\text{”}\ \mathit{Prop} \\ & | & \mathit{Prop}\ \text{“}\wedge\text{”}\ \mathit{Prop} \\ & | & \mathit{Prop}\ \text{“}\vee\text{”}\ \mathit{Prop} \\ & | & \text{“(”}\ \mathit{Prop}\ \text{“)”} \\ & | & \mathbf{variable} \end{array}$$