Karlsruher Institut für Technologie
Lehrstuhl für Programmierparadigmen
Sprachtechnologie und Compiler WS 2010/2011
Dozent: Prof. Dr.-Ing. G. Snelting
Übungsleiter: Matthias Braun `matthias.braun@kit.edu`

Übungsblatt 3 | Ausgabe: 30.4.2012 | Besprechung: 7.5.2012

Aufgabe 1: *EBNF*

Entwerfen Sie eine Grammatik für erweiterte Backus-Naur-Form (EBNF). Verwenden Sie dabei EBNF um die Produktionen ihrer Grammatik zu spezifizieren.

Aufgabe 2: *Praxis: Stringtabelle, Rekursiver Abstieg*

Unter `http://pp.info.uni-karlsruhe.de/lehre/SS2012/compiler/uebung/intern/minicalc2.zip` finden Sie eine Lösung zu Aufgabe 1 aus dem letzten Übungsblatt. Diese wurde um eine Stringtabelle und neue Tokentypen (**T_PI**, **T_SIN**, **T_COS**, **T_TAN**) erweitert.

Hinweis: Zum Übersetzen der Dateien wird gcc, flex und make benötigt. Wir empfehlen die Aufgabe unter Linux zu bearbeiten, da die entsprechenden Tools in allen Distributionen vorhanden sind. Windows-Portierungen dieser Tools sind unter `http://sourceware.org/cygwin/` erhältlich. Mac-Versionen zum Beispiel unter `http://www.macports.org/`. Die Dateien werden dann durch Eingabe von **make** auf der Kommandozeile übersetzt.

2.1 *Ausdrucksgrammatiken*

Der Parser benutzt folgende Grammatik:

$$
\begin{aligned}
Z &\rightarrow Z' \ \# \\
Z' &\rightarrow \mathit{expression}\ Z' \mid \varepsilon \\
\mathit{expression} &\rightarrow \textbf{newline} \mid \mathit{add\_sub\_expression}\ \textbf{newline} \\
\mathit{add\_sub\_expression} &\rightarrow \mathit{mul\_div\_expression}\ \mathit{add\_sub\_expression}' \\
\mathit{add\_sub\_expression}' &\rightarrow \textbf{plus}\ \mathit{add\_sub\_expression} \mid \textbf{minus}\ \mathit{add\_sub\_expression} \mid \varepsilon \\
\mathit{mul\_div\_expression} &\rightarrow \mathit{atomic\_expression}\ \mathit{mul\_div\_expression}' \\
\mathit{mul\_div\_expression}' &\rightarrow \textbf{star}\ \mathit{mul\_div\_expression} \mid \textbf{slash}\ \mathit{mul\_div\_expression} \mid \varepsilon \\
\mathit{atomic\_expression} &\rightarrow \textbf{number} \mid \textbf{lbrace}\ \mathit{add\_sub\_expression}\ \textbf{rbrace}
\end{aligned}
$$

Der Autor der Grammatik ist vermutlich von der aus der Vorlesung bekannten Grammatik für arithmetische Ausdrücke ausgegangen. Er hat sich dann allerdings nicht exakt an die Vorschriften zur Elimination der Linksrekursion gehalten.

- Was hat er falsch gemacht?
- Ist die Grammatik für Top-Down-Parsing geeignet?
- Bei welchen Eingabe treten Fehler auf? Geben Sie ein Beispiel an!
- Geben Sie eine alternative Grammatik an, die für SLL(1)-Parser geeignet ist und das Problem behebt.
- Ändern sie den rekursiven Parser entsprechend ihrer neuen Grammatik ab.

2.2 *Nutzung der Stringtabelle*

Erweitern Sie den Scanner um Regeln die Bezeichner erkennen. Bezeichner beginnen mit einem Buchstaben (a-z, A-Z) optional gefolgt von beliebig vielen Buchstaben oder Ziffern.

Wurde ein Bezeichner erkannt, so soll in der Stringtabelle nachgeschlagen (find_or_insert_symbol ("symbolstring")) werden welchem Token-Typ er entspricht (symbol−>id). Dazu muss die Stringtabelle vorher mit Einträgen befüllt werden (am Anfang der Funktion main).

Erweitern Sie das Programm, so dass die mathematischen Ausdrücke $\sin x$, $\cos x$ und $\tan x$ erkannt werden. *Hinweis*: In C kann man die entsprechenden Berechnungen mit den gleichnamigen Funktionen sin(x), cos(x) und tan(x) durchführen.

Aufgabe 3: LL(k) *und* SLL(k)

Eine kontextfreie Grammatik $G = (T, N, P, Z)$ heißt SLL(k), $k > 0$, falls für beliebige Ableitungen

$$\begin{array}{ll} Z \Rightarrow_L^* \mu A\chi \Rightarrow \mu\nu\chi \Rightarrow^* \mu\gamma & \mu, \gamma \in T^*, \nu, \chi \in V^*, A \in N \\ Z \Rightarrow_L^* \mu' A\chi' \Rightarrow \mu'\omega\chi' \Rightarrow^* \mu'\gamma' & \mu', \gamma' \in T^*, \omega, \chi' \in V^* \end{array}$$

aus $(k : \gamma = k : \gamma')$ $\quad \nu = \omega$ folgt.

Zeigen Sie:

3.1 *Ein* SLL(1) *Kriterium*

Eine kontextfreie Grammatik $G = (T, N, P, Z)$ ist SLL(1) gdw. für alle erreichbaren Produktionen $A \to l_1$, $A \to l_2$ mit $l_1 \neq l_2$ gilt:

$$\mathrm{Anf}_1(l_1\mathrm{Folge}_1(A)) \cap \mathrm{Anf}_1(l_2\mathrm{Folge}_1(A)) = \emptyset.$$

Eine Produktion A ist erreichbar, wenn eine Ableitung $Z \to \mu A\chi$ mit $\mu, \chi \in V^*$ existiert.

3.2 SLL(k) $\to$ LL(k)

Jede SLL(k)-Grammatik ist auch LL(k).

3.3 LL(1) $\leftrightarrow$ *S*LL(1), $\neg$(LL(k) $\leftrightarrow$ SLL(k))

Eine kontextfreie Grammatik $G = (T, N, P, Z)$ ist LL(1) genau dann, wenn sie SLL(1) ist. Die Aussage (LL(k) $\iff$ SLL(k)) gilt nicht für beliebige k.

3.4 $\neg$SLL(2)

Grammatiken sind nicht alle SLL(2) (Gegenbeispiel).

Aufgabe 4: *Was bin ich?*

Für welches k sind die folgenden Grammatiken LL(k)? Sind sie auch SLL(k)?

(a)
$$\begin{array}{l} Z \to S \\ S \to aS \\ S \to a \end{array}$$

(b)
$$\begin{array}{l} Z \to S \\ S \to aA \\ A \to S \\ A \to \varepsilon \end{array}$$

(c)
$$\begin{array}{l} Z \to C \mid D \\ C \to aC \mid b \\ D \to aD \mid c \end{array}$$

(d)
$$\begin{array}{l} Z \to S \\ S \to Ax \mid By \mid dAy \\ A \to C \mid z \\ B \to C \\ C \to c \end{array}$$

Aufgabe 5: LL(0)*-Eigenschaft*

5.1 $|L(G)|$

Wieviele Wörter enthält eine Sprache, die durch eine LL(0)-Grammatik beschrieben werden kann?

5.2 $|P|$

Wieviele Produktionen gibt es in einer LL(0)-Grammatik für jedes Nichtterminal?