Universität Karlsruhe (TH)

Lehrstuhl für Programmierparadigmen

Sprachtechnologie und Compiler WS 2008/2009 — http://pp.info.uni-karlsruhe.de/
Dozent: Prof. Dr.-Ing. G. Snelting — snelting@ipd.info.uni-karlsruhe.de
Übungsleiter: Matthias Braun — braun@ipd.info.uni-karlsruhe.de

Übungsblatt 3 | Ausgabe: 7.11.2008 | Besprechung: 12.11.2008

Aufgabe 1: *Praxis: Symboltabelle, Rekrusiver Abstieg*

Unter `http://pp.info.uni-karlsruhe.de/lehre/WS200809/compiler/uebung/minicalc2.zip` finden Sie eine Lösung zu Aufgabe 1 aus dem letzten Übungsblatt. Diese wurde um eine Symboltabelle und neue Tokentypen (`T_PI`, `T_SIN`, `T_COS`, `T_TAN`) erweitert.

*H*inweis: Zum Übersetzen der Dateien wird gcc, flex und make benötigt. 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.

1.1 *Ausdrucksgrammatiken*

Der Parser benutzt folgende Grammatik:

$$
\begin{aligned}
\text{Z} &\rightarrow \text{Z}'\ \$ \\
\text{Z}' &\rightarrow \text{expression Z}' \mid \epsilon \\
\text{expression} &\rightarrow \texttt{newline} \mid \text{add\_sub\_expression } \texttt{newline} \\
\text{add\_sub\_expression} &\rightarrow \text{mul\_div\_expression add\_sub\_expression}' \\
\text{add\_sub\_expression}' &\rightarrow + \text{ add\_sub\_expression} \mid - \text{ add\_sub\_expression} \\
\text{mul\_div\_expression} &\rightarrow \text{atomic\_expression mul\_div\_expression}' \\
\text{mul\_div\_expression}' &\rightarrow * \text{ atomic\_expression} \mid / \text{ atomic\_expression} \\
\text{atomic\_expression} &\rightarrow \texttt{number} \mid \texttt{lbrace add\_sub\_expression rbrace}
\end{aligned}
$$

Für einige mathematischen Ausdrücke werden mit dieser Grammatik „falsche" also nicht dem üblichen mathematischen Verständnis entsprechenden Syntaxbäume aufgebaut.

- Welche sind das?
- 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.

1.2 *Nutzung der Symboltabelle*

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 Symboltabelle nachgeschlagen (find_or_insert_symbol ("symbolstring")) werden welchem Token-Typ er entspricht (symbol−>id). Dazu muss die Symboltabelle 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. *H*inweis: In C kann man die entsprechenden Berechnungen mit den gleichnamigen Funktionen sin(x), cos(x) und tan(x) durchführen.

1.3 *Skalierbarkeit der Symboltabelle*

Sehen Sie sich die Implementierung der Symboltabelle an.

- Was passiert wenn sehr viele Einträge in der Symboltabelle erstellt werden?
- Verbessern Sie die Skalierbarkeit durch geeignete Maßnahmen (*Zusatzaufgabe*).

Aufgabe 2: 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:

2.1 *Ein* SLL(1) *Kriterium*

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

$$\mathrm{FIRST}_1(l_1\, \mathrm{FOLLOW}_1(A)) \cap \mathrm{FIRST}_1(l_2\, \mathrm{FOLLOW}_1(A)) = \emptyset.$$

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

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

2.3 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.

2.4 $\neg$SLL(2)

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

Aufgabe 3: *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 4: LL(0)*-Eigenschaft*

4.1 $|L(G)|$

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

4.2 $|P|$

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