Lehrstuhl für Programmiersprachen und Übersetzerkonstruktion
CAU Kiel, Informatik und Prakt. Mathematik
Prof. Dr. M. Hanus, Frank Huch

24118 Kiel
Christan-Albrechts-Platz 4
Tel.: 0431 / 880-7277

8. Übung „Übersetzerkonstruktion“

Abgabe am 17. Dezember in der Vorlesung

Aufgabe 24 4 Punkte

Implementieren Sie den in der Vorlesung vorgestellten Interpreter für SIMPLE mit print-Anweisungen mit Happy. Erweitern Sie ihn um `if-then-else` und `while`.

Aufgabe 25 5 Punkte

Die Sprache $L = \{a^{2^n} \mid n \in \mathbb{N}\}$ ist nicht kontextfrei. Definieren Sie eine attributierte Grammatik für die Sprache $\{a\}^*$, welche mittels eines synthetischen, booleschen Attributs des Startsymbols anzeigt, ob ein abgeleitetes Wort in L liegt. Verwenden Sie nur einfache Funktionen, wie Addition, Multiplikation, Test auf Gleichheit. D.h. verwenden Sie keine Funktion, welche testet, ob eine Zahl eine Zweierpotenz ist.
Geben Sie die Attributierung des Syntaxbaums für das Wort a^4 an und lösen Sie sie.

Aufgabe 26 11 Punkte

a) Erweitern Sie die folgende Grammatik für MPS

$$
\begin{array}{lcll}
\mathit{Prg} & \rightarrow & \textbf{var}\ \mathit{Decl};\mathit{Stm}. & (1) \\
\mathit{Decl} & \rightarrow & \textbf{id},\ \mathit{Decl} \mid \textbf{id} & (2,3) \\
\mathit{Stm} & \rightarrow & \textbf{id} := \mathit{Exp} \mid \textbf{begin}\ \mathit{List}\ \textbf{end} \mid & (4,5) \\
 & & \textbf{if}\ \mathit{Cond}\ \textbf{then}\ \mathit{Stm} \mid \textbf{while}\ \mathit{Cond}\ \textbf{do}\ \mathit{Stm} & (6,7) \\
\mathit{List} & \rightarrow & \mathit{Stm};\ \mathit{List} \mid \mathit{Stm} & (8,9) \\
\mathit{Cond} & \rightarrow & \mathit{Exp}\ \textbf{relop}\ \mathit{Exp} & (10) \\
\mathit{Exp} & \rightarrow & \textbf{id} \mid \textbf{num} \mid (\mathit{Exp}\ \textbf{arop}\ \mathit{Exp}) & (11-13)
\end{array}
$$

um eine Attributierung, welche im Deklarationsteil eine einfache Symboltabelle aller deklarierten Variablen aufbaut und hierbei testet, ob Variablen mehrfach deklariert werden. Der Symboltabelleneintrag einer Variablen enthalte hierfür mindestens die Speicheradressen ($\in \mathbb{N}$), welche für diese Variable reserviert wird. Wir nehmen vereinfacht an, daß alle Variablen nur genau eine Speichereinheit benötigen. Somit erhalten n Variablen die Speicheradressen 1 bis n. Gehen Sie davon aus, daß jedes Terminalsymbol **id** seinen Variablenbezeichner als synthetisches Attribut hat.

Überprüfen Sie mit Hilfe dieser Symboltabelle, ob alle verwendeten Bezeichner deklariert sind. Erzeugen Sie weiterhin in einem synthetischen Attribut des Startsymbols *Prg* Code für das gegebene Programm. Dieser Code bestehe hier zunächst aus der Folge der Speicheradressen der vorkommenden Variablen des Anweisungsteils. Hierbei soll natürlich die Reihenfolge der Variablen berücksichtigt werden.

b) In vielen Programmiersprachen ist die Trennung zwischen Deklarations- und Anweisungsteil weniger streng. So können in Blöcken neue Variablen deklariert werden. Ihre Gültigkeit ist auf diesen Block beschränkt und bereits deklarierte Variablen mit dem gleichen Namen werden ausgeblendet.

   Wir fügen zur Grammatik die Regel

$$Stm \;\to\; \textbf{begin}\ Decl\ List\ \textbf{end} \tag{5'}$$

   hinzu. Erweitern Sie Ihre Attributierung für lokal definierte Variablen. Verwenden Sie Speicherzellen nach dem Ende eines Blocks weiter. Der erzeugte Code bestehe wiederum aus der Folge der verwendeten Bezeichner.

c) Erweitern Sie Ihren Top-Down-Parser für MPS um diese Attributierung.