Lehrstuhl für Programmiersprachen und Übersetzerkonstruktion
CAU Kiel, Institut für Informatik
Prof. Dr. M. Hanus

24118 Kiel
Christan-Albrechts-Platz 4
Tel.: 0431 / 880-7271

9. Übung „Übersetzerbau“

Bearbeitung bis zum 17. Juni 2008

Bitte senden Sie die Programmieraufgaben zusätzlich zur Abgabe in ausgedruckter Form auch per Email an Axel Stronzik (`axs@informatik.uni-kiel.de`)!

Aufgabe 27
In dieser Aufgabe sollen Sie eine einfache Typprüfung implementieren. Wir erweitern MPS zunächst um getypte Deklarationen:

$$\begin{array}{lcl} \mathit{Decl} & \longrightarrow & \textbf{id} : \mathit{Type}\ , \mathit{Decl} \mid \textbf{id} : \mathit{Type} \\ \mathit{Type} & \longrightarrow & \textbf{bool} \mid \textbf{int} \end{array}$$

Außerdem unterscheiden wir nun nicht mehr zwischen Bedingungen (*Cond*) und Ausdrücken (*Exp*). Hierdurch vereinfacht sich die Grammatik für *Stm* und *Exp* wie folgt:

$$\begin{array}{l} \mathit{Stm} \rightarrow \textbf{id} := \mathit{Exp} \mid \textbf{begin}\ \mathit{StmList}\ \textbf{end} \mid \textbf{if}\ \mathit{Exp}\ \textbf{then}\ \mathit{Stm} \mid \textbf{while}\ \mathit{Exp}\ \textbf{do}\ \mathit{Stm} \\ \mathit{Exp} \rightarrow \textbf{id} \mid \textbf{numConst} \mid \textbf{boolConst} \mid (\mathit{Exp}\ \mathit{Op}\ \mathit{Exp}) \\ \mathit{Op} \rightarrow + \mid * \mid <= \mid >= \mid \&\& \mid \| \end{array}$$

a) Definieren Sie algebraische Datentypen (`ASTree`) für die Repräsentation des abstrakten Syntaxbaums eines solchen MPS-Programms.

b) Erweitern Sie einen Ihrer Parser für MPS (Top-down oder Bottom-up) um die Generierung des abstrakten Syntaxbaums.

c) Implementieren Sie eine Funktion `typeCheck :: ASTree -> Bool`, welche die korrekte Verwendung der Typen **bool** und **int** überprüft.

Aufgabe 28
In dieser Aufgabe wollen wir den Zussammenhang zwischen beliebigen zykelfreien attributierten Grammatiken und Top-Down-Parsern in lazy funktionalen Sprachen untersuchen. Wir betrachten hierzu folgende (zugegeben etwas unsinnige) Programmiersprache:

$$\begin{array}{lcl} \mathit{Prg} & \rightarrow & \mathit{StmList} \\ \mathit{StmList} & \rightarrow & \mathit{Stm}\ ;\ \mathit{StmList} \mid \epsilon \\ \mathit{Stm} & \rightarrow & \textbf{def id} \mid \textbf{use id} \end{array}$$

Die Sprache unterscheidet nur definierende und verwendende Vorkommen von Variablen. Alle verwendeten Variablen müssen in dieser Programmiersprache allerdings auch definiert werden. Dies muss aber nicht vor ihrer Verwendung geschehen.

a) Definieren Sie für diese Grammatik eine Attributierung, welche diese Eigenschaft überprüft.

   Als Einschränkung haben die Nichtterminalsymbole jedoch nur folgende Attribute:

$$Syn(StmList) = Syn(Stm) = \{b, d\}$$
$$Syn(Prg) = \{b\}$$
$$Inh(StmList) = Inh(Stm) = \{s\}$$

   wobei d und s Symboltabellen (hier: Listen von Variablenbezeichnern) sind und b vom Typ `Bool` ist. Hierbei sollen in der Liste d alle in einem Programm definierten Bezeichner aufgesammelt werden, die dann mittels dem vererbten Attribut s an alle Anweisungen heruntergereicht wird. Außerdem habe jedes Teminalsymbol **id** ein synthetisiertes Attribut *name*, welches seinen Bezeichner enthalte. Ob in einem Programm alle Bezeichner deklariert sind, soll mit dem Attribut b des Startsymbols *Prg* angezeigt werden.

b) Implementieren Sie eine rekursiven Abstiegsparser für diese Grammatik.

c) In der Vorlesung wurde erläutert, wie L-Attributierungen in rekursiven Abstiegsparsern implementiert werden können. Verwenden Sie genau diese Technik (inherite Attribute als Argumente, synthetisierte Attribute als Rückgabewerte) zur Implementierung der hier entwickelten Attributierung.
   Entsteht ein ausführbares Programm? Warum/warum nicht?