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

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

7. Übung „Übersetzerbau“

Bearbeitung bis zum 3. Juni 2008

Bitte senden Sie die Programmieraufgabe 24 zusätzlich zur Abgabe in ausgedruckter Form auch per Email an Axel Stronzik (`axs@informatik.uni-kiel.de`)!

Aufgabe 22
In der Vorlesung wurden für die folgende Grammatik die LR(1)- und LALR(1)-Parsingtabellen konstruiert:

$$\begin{array}{lcllcl} S' & \to & S\$ & E & \to & V \\ S & \to & V = E & V & \to & x \\ S & \to & E & V & \to & *E \end{array}$$

Zeigen Sie, dass diese Grammatik die SLR(1)-Eigenschaft nicht besitzt.

Aufgabe 23

a) Überlegen Sie sich ein Verfahren, wie Sie direkt die $LALR(1)$–Mengen berechnen können, ohne zuvor die $LR(1)$-Mengen zu berechnen (nicht formal, Idee reicht aus).

b) Wenden Sie Ihr Verfahren auf folgende Grammatik an:

$$\begin{array}{rlll} G: & S & \longrightarrow aaSb \mid A \mid \varepsilon \\ & A & \longrightarrow Aa \mid c \end{array}$$

Liegt G in $LR(0)$, $SLR(1)$ bzw. $LALR(1)$?
Konstruieren Sie ggf. die $LALR(1)$–Analysetabelle.

Aufgabe 24
In dieser Aufgabe sollen Sie einen Interpreter für die Programmiersprache MPS implementieren. Hierbei sei MPS wie in Aufgabe 16 (also mit Deklarationen getrennt durch Kommas) definiert. Außerdem seien die Statements um eine Print-Anweisung erweitert:

Stm $\longrightarrow$ print (Expr)

wobei print,(und) Terminale seien und Expr das Nichtterminal zur Erzeugung von Ausdrücken sei.
Gehen Sie in folgenden Arbeitsschritten vor:

a) Überarbeiten Sie den Scanner für MPS, so dass er auch die neuen Sprachkonstrukte von MPS abdeckt.

b) Implementieren Sie aufbauend auf dem Scanner einen Happy-Parser für MPS. Das Happy-Tool ist auf den Suns unter dem Nutzer `haskell` eingerichtet und kann mit `/home/haskell/bin/happy` *datei*`.y` gestartet werden. Die Happy-Datei aus der Vorlesung finden Sie auf der Web-Seite zur Übung.

c) Erweitern Sie Ihren Parser um semantische Aktionen zum Interpretieren von MPS. Gehen Sie hierbei analog zu dem in der Vorlesung vorgestellten Interpreter für SIMPLE mit print-Anweisungen vor (s.u.). Alle Ausgaben des Programms sollen in einer Zahlenliste (`[Int]`) gespeichert werden und schließlich der Rückganbewert des Programms sein.

Ein einfacher Interpreter für SIMPLE:

```
{
... -- definition of update and empty_env
}

%name interpreter       -- name of created function
%tokentype { Token }    -- type of accepted tokens

%token ';'   { SEMICOLON }
       '='   { ASSIGN }
       print { PRINT }
       '+'   { PLUS }
       '*'   { MULT }
       '('   { LPAREN }
       ')'   { RPAREN }
       id    { ID $$ }
       num   { NUM $$ }
%%                      -- as in yacc

program : stmts             { snd ($1 (empty_env,"")) }

stmts : stm ';' stmts       { $3 . $1 }
      | stm                 { $1 }

stm : id '=' exp            { \(e,o)->(update e $1 ($3 e),o) }
    | print '(' exp ')'     { \(e,o)->(e,o++show($3 e ::Int)++"\n") }

exp : exp '+' term          { \e->($1 e + $3 e)::Int }  -- typing necessary (classes)
    | term                  { $1 }

term : term '*' factor      { \e->($1 e * $3 e)::Int }
     | factor               { $1 }

factor : id                 { \e->e $1 }
       | num                { \_->$1 }
       | '(' exp ')'        { $2 }

{
... -- definition of happyError, type Token
}
```