PHILIPPS-UNIVERSITÄT MARBURG
Fachbereich Mathematik und Informatik
Prof. Dr. R. Loogen J.Berthold

D-35032 Marburg
Hans Meerwein Straße
Lahnberge

19. Januar 2004

Übungen zu „Grundlagen des Compilerbaus“, WS 2003/04

Nr. 10 , Abgabe und Besprechung: 26. Januar 2004 in der Übung

---

Mündliche Aufgaben

10 .1 Zeigen Sie, dass die folgende Grammatik LALR(1), aber nicht SLR(1) ist:

$$\begin{array}{lll} G: & S \rightarrow & Aa \mid bAc \mid dc \mid bda \\ & A \rightarrow & d \end{array}$$

10 .2 **Konfliktauflösung mit Operatorpräzedenzen**
Gegeben sei die folgende Grammatik für Boolesche Ausdrücke mit konstantem Wert.

$$\begin{array}{lll} \mathrm{S} & \rightarrow & \mathrm{B} \\ \mathrm{B} & \rightarrow & \mathrm{B} \wedge \mathrm{B} \mid (\mathrm{B}) \mid \neg \mathrm{B} \mid \mathrm{t} \mid \mathrm{f} \end{array}$$

Erstellen Sie eine Analysetabelle, in der die Shift/Reduce-Konflikte über Präzedenzregeln der Operatoren aufgelöst werden. Erläutern Sie Ihre Lösung.

Schriftliche Aufgaben

10 .3 **LR(1)-Mengen** 4 Punkte

(a) Berechnen Sie die LR(1)-Mengen für die folgende Grammatik.

$$\begin{array}{lll} G: S & \rightarrow & A \\ A & \rightarrow & bB \mid a \\ B & \rightarrow & C \mid Ce \\ C & \rightarrow & dA \end{array}$$

(b) Ist G in LALR(1) / in LR(1) ? Begründen Sie Ihre Antwort.

10 .4 **Parser für eine While-Sprache** 8 Punkte
Konstruieren Sie einen Parser für die in Aufgabe 6.3 definierte While-Sprache. Die Produktionen der Nonterminale *program* und A seien dabei durch die folgenden Produktionen ersetzt (B fällt ganz weg):

$$\begin{array}{lll} \textit{program} & \rightarrow & \textbf{program}\ \mathrm{Id}\ \textbf{\{ var}\ \ A\ \ \textit{stmt}\ \ \textbf{\}} \\ A & \rightarrow & A\ \textbf{;}\ \mathrm{Id} \\ & \mid & \mathrm{Id} \end{array}$$

(a) Erzeugen Sie einen Scanner mit ALEX und einen Parser mit HAPPY. Der Parser soll ein While-Programm einlesen und einen internen abstrakten Syntax-Baum erstellen. Die Datentypen für den Syntaxbaum seien dabei: / 5

Haskell Code

```
type Var = String
data Program = Prog String     [Var]       [Stmt]
--                  Name   Deklarationen Anweisungen

data Stmt = Print Var
          | Assign Var Expr
          | If Cond [Stmt]
          | While Cond [Stmt]

data Cond = Rel RelOp Expr Expr

data RelOp = EQ | NEQ | LE | LEQ | GR | GEQ
--           =    !=    <    <=     >   >=
data Expr = V Var | N Int | Op Char Expr Expr
```

(Benutzen Sie die Datei *ProgType.hs* von der Vorlesungsseite)

Konstruieren Sie den Parser in den folgenden Schritten:

- Definieren Sie für jedes Token aus dem Scanner ein entsprechendes Symbol in HAPPY-Syntax.
- Fügen Sie Ihre Haskell-Repräsentation eines While-Programms als neuen Datentyp in die HAPPY-Datei ein.
- Tragen Sie die Grammatik in die HAPPY-Datei ein. Versehen Sie dabei jedes Nonterminal mit einem **Haskell-Datentyp**. Erzeugen Sie in den Aktionen der Regelalternativen die Haskell-Repräsentation des zu erkennenden Sprachteils.
- Benutzen Sie die folgende `main`-Funktion:

  ```
  main :: IO ()
  main = getContents  >>=
         putStrLn . show . whileParser . alexScanTokens
  ```

  Damit wird vom Parser das zu erkennende Programm in der Standard-Eingabe erwartet und nach dem Parsen ausgegeben.

(b) Implementieren Sie eine Konsistenz-Überprüfung der Variablen-Deklarationen. Diese arbeitet auf der abstrakten Syntax und soll feststellen, ob / 3

i. alle benutzten Variablen auch vorher definiert wurden,

ii. alle Variablennamen nur genau einmal in einer Deklaration verwendet werden und

iii. ob Variablen deklariert wurden, die nie benutzt werden.