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

D-35032 Marburg
Hans Meerwein Straße
Lahnberge

24.November 2003

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

Nr. 5 , Abgabe und Besprechung: 1. Dezember in der Übung

Mündliche Aufgaben

5 .1 **Dangling `else`**
Die Syntax des wohlbekannten `if`-Konstrukts für bedingte Ausführung in Programmiersprachen sei über die folgende Grammatik definiert:

$$\begin{aligned} G: \quad S \to\ & a \\ & \mid \texttt{if}\ b\ \texttt{then}\ S\ E \\ E \to\ & \texttt{else}\ S \\ & \mid \varepsilon \end{aligned}$$

(a) Konstruieren Sie nach Definition der Vorlesung den nichtdeterministischen TDA zu dieser Grammatik und geben Sie eine akzeptierende Konfigurationsfolge für die folgende Tokenfolge an: `if` b `then` `if` b `then` a `else` a

(b) Zeigen Sie, dass G mehrdeutig ist und geben Sie eine eindeutige Grammatik G' mit $L(G') = L(G)$ an.

5 .2 Seien $\alpha, \beta \in (\Sigma \cup N)^*$ Satzformen und $v \in \Sigma^*$ sowie $k \in N_0$.
Begründen Sie die folgenden **Eigenschaften von first**:

$$\varepsilon \in \text{first}_k(\alpha) \iff k = 0 \vee \alpha \Rightarrow^* \varepsilon \quad (1)$$

$$\alpha \Rightarrow^* \beta \implies \text{first}_k(\beta) \subseteq \text{first}_k(\alpha) \quad (2)$$

$$v \in \text{first}_k(\alpha) \iff \exists_{x \in \Sigma^*} : \alpha \Rightarrow_l^* x \wedge \text{first}_k(x) = \{v\} \quad (3)$$

Schriftliche Aufgaben

5 .3 **LL(0)-Sprachen** 3 Punkte
LL(k)-Grammatiken sind, informell beschrieben, diejenigen Grammatiken, für die der TDA mit einem Lookahead von k Zeichen deterministisch entscheiden kann, welche Produktion für die Linksableitung eines Worts auszuwählen ist.

(a) Charakterisieren Sie die Menge der LL(0)-Grammatiken sowie die Menge der LL(0)-Sprachen auf möglichst einfache Weise.

(b) Zeigen Sie: Jede reguläre Sprache lässt sich durch eine LL(1)-Grammatik erzeugen.

5 .4 **TDA-Durchlauf** 4 Punkte

Die Grammatik G sei durch die folgenden Produktionen gegeben:

$$\begin{aligned} G: \quad S &\to aBSS \mid bBS \mid cSB \mid dLe \mid f \\ L &\to SaL \mid \varepsilon \\ B &\to aC \mid bC \\ C &\to eB \mid \varepsilon \end{aligned}$$

Konstruieren Sie den Top-Down-Analyseautomaten zu G gemäß den Definitionen der Vorlesung und geben Sie eine akzeptierende Konfigurationsfolge für die Eingabe `dbaabffae` an. Gibt es mehrere Möglichkeiten dafür (mit Begründung)?

5 .5 **LL(1)-Test in Haskell** 5 Punkte

Üblicherweise benutzen Grammatiken für Programmiersprachen die Ausgabe des Scanners, also `[Token]`, als Terminalsymbole. Zur Vereinfachung sollen für diese Aufgabe die Grammatiken in Haskell statt dessen Buchstaben als Symbole benutzen:

Haskell Code

```
-- CFG = (Nonterminals,Terminals,StartSymbol,Produktionen)
type CFG = ([Nonterminal], [Terminal], Nonterminal, Rules)
type Rules = [(Nonterminal,Satzform)]
type Nonterminal = Char -- Konvention: Großbuchstaben
type Terminal = Char    -- Konvention: Kleinbuchstaben
type Satzform = String
```

Dabei seien (als Konvention, *nicht* typsicher) Nonterminale stets Großbuchstaben, Terminale hingegen stets Kleinbuchstaben.

(a) Definieren Sie geeignete Terminalsymbole und stellen Sie die Grammatik aus 5.1 als `CFG` dar. / 1

In der Vorlesung wurden Verfahren zur Bestimmung der Mengen $\text{first}_1(\alpha)$ und $\text{follow}_1(N)$ eingeführt. Mit diesen Mengen lässt sich ein einfaches LL(1)-Kriterium angeben.

(b) Implementieren Sie eine Funktion `first1 :: CFG -> String -> [String]`, welche die First-Menge einer Satzform bestimmt. / 2

Wir benutzen für das Ergebnis `[String]` statt `[Char]`, weil ε enthalten sein kann. Zeichen aus `first1(x)` werden als einelementige Liste dargestellt, ε als leere Liste.

Um Endlosschleifen durch Linksrekursion zu vermeiden, sollten Sie eine Hilfsfunktion verwenden, die in einem zusätzlichen `String`-Parameter protokolliert, welche Nonterminale bereits bearbeitet wurden.

(c) Die Follow-Menge eines Nonterminals wird auf ähnliche Weise mit einer Funktion `follow1 :: CFG -> Char -> [String]` bestimmt. Implementieren Sie auch diese Funktion; beachten Sie dabei die Möglichkeit von ε-Regeln. / 2

Zusatzaufgabe:

Implementieren Sie einen LL(1)-Test `ll1 :: CFG -> Bool`, indem Sie (mit `map`) die Lookahead-Mengen zu den Produktionen bestimmen und vergleichen. 2 Punkte