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

D-35032 Marburg
Hans Meerwein Straße

24.November 2005

5. Übung zu „Grundlagen des Compilerbaus“, WS 2005/06

Abgabe schriftlicher Aufgaben: Do, 1.Dezember 2005 (vor der Vorlesung)
Besprechung mündlicher Aufg.: ab 28.November 2005 in der Übung

---

Mündliche Aufgaben

5.1 **Keine LL(k)-Grammatiken**

(a) Gegeben sei die Grammatik $G = (\{S\}, \{a, b\}, P, S)$ mit

$$P = \{S \to aSab \mid aSabb \mid b\}$$

Zeigen Sie: Es existiert *kein* k mit $G \in LL(k)$.

(b) Zeigen Sie formal (mit Hilfe der LL(k)-Definition):

$$G_l \in CFG \text{ linksrekursiv } \Rightarrow \forall k \in \mathbb{N} : G_l \notin LL(k)$$

5.2 **Parserkombinatoren**

(a) Schreiben Sie einen Parser

```
nat :: Parser Char Int
```

für natürliche Zahlen.

(b) Erweitern Sie `nat` zu einem Parser

```
int :: Parser Char Int,
```

der ganze Zahlen erkennt.

(c) Oft soll ein Parser eine Reihe von Objekten erkennen, die durch Separatoren voneinander getrennt sind, zum Beispiel:

`Stmt1; Stmt2; Stmt3; Stmt4` oder `55, 44, 1, -99`

Schreiben Sie einen Parserkombinator

```
sepBy :: Parser tok a -> Parser tok b -> Parser tok [a],
```

der eine *nichtleere* Reihe von durch Separatoren getrennten Objekten erkennt und nur die Objekte zurückliefert. Als Argumente werden dabei ein Parser zum Erkennen der Objekte und ein Parser zum Erkennen der Separatoren mitgegeben.

(d) Definieren Sie mit Hilfe der Parser aus 2b und 2c einen Parser, der eine Liste von ganzen Zahlen erkennt.

Bitte wenden!

Schriftliche Aufgaben

5.3 Starke LL(k)-Grammatiken

3 Punkte

Für reduzierte kontextfreie Grammatiken sei die *starke LL(k)-Eigenschaft* (SLL(k)) folgendermaßen definiert:

Sei $G = (N, \Sigma, S, P) \in CFG$ reduziert.

$$\begin{aligned} G \in \mathrm{SLL}(k) \quad :\Longleftrightarrow \quad & \forall A \in N; \beta, \gamma \in \Sigma^* \text{ gilt:} \\ & A \to \beta,\ A \to \gamma \in P \wedge \beta \neq \gamma \\ & \quad \Longrightarrow \mathit{first}_k(\beta\ \mathit{follow}_k(A)) \cap \mathit{first}_k(\gamma\ \mathit{follow}_k(A)) = \emptyset \end{aligned}$$

Ein Satz der Vorlesung zeigt, daß SLL(1) = LL(1) ist. Dies ist nicht auf beliebige k verallgemeinerbar!

Zeigen Sie, dass für die folgende Grammatik gilt: $G \in LL(2) \setminus SLL(2)$

$$\begin{aligned} \text{G:}\quad S &\to aAab \mid bAbb \\ A &\to a \mid \epsilon \end{aligned}$$

5.4 Transformation von Grammatiken

3 Punkte

Eine Grammatik sei durch die folgenden Produktionen gegeben:

$$\begin{aligned} G:\quad S &\to (a) \mid () \mid (T) \\ T &\to T, S \mid S \end{aligned}$$

Beseitigen Sie die Linksrekursion und führen Sie eine Linksfaktorisierung durch.

Bestimmen Sie die look-ahead-Mengen der Regeln für die resultierende Grammatik. Ist sie aus $LL(1)$?

5.5 Parser-Kombinator für verschachtelt geklammerte Ausdrücke

6 Punkte

Die nebenstehende Grammatik definiert geschachtelte Klammerstrukturen. Sowohl die Korrektheit der Schachtelung, als auch die maximale Verschachtelungstiefe können mit einem geeigneten Parser festgestellt werden.

$$\begin{aligned} G: S \to\ & () \\ \mid\ & (S) \\ \mid\ & SS \end{aligned}$$

(a) Definieren Sie zunächst einen Kombinator, der Inhalte zwischen zwei definierbaren Begrenzungen erkennt:

Haskell Code
```haskell
between :: Parser tok open -> Parser tok close
           -> Parser tok a -> Parser tok a
```

Dabei seien die Parameter von `between` ein Parser für die öffnende Begrenzung, einer für die schließende Begrenzung und einer für den Inhalt. Zurückgegeben wird nur das Parse-Ergebnis für den Teil zwischen den Begrenzungen.

(b) Mit `between` können Sie einen rekursiven Parser `bracket :: Parser Char Int` definieren, der eine Folge von öffnenden und schließenden Klammern erkennt und als Ergebnis die maximale Verschachtelungstiefe zurückgibt.
Sie sollten die Linksrekursion in Regel 3 vorher durch eine geeignete Hilfskonstruktion ersetzen. Die Erzeugung einer LL(1)-Grammatik ist aber nicht erforderlich, da die Kombinatoren stets alle Parsing-Alternativen betrachten.