Prof. Dr. R. Loogen, M. Dieterle
Fachbereich Mathematik und Informatik
Hans-Meerwein-Straße
D-35032 Marburg

8. November 2011

Philipps Universität Marburg

Übungen zu „Grundlagen des Compilerbau“, Winter 2011/12

Nr. 4, Abgabe der Aufgaben: 15. November 2011 vor der Vorlesung

---

Aufgaben

4.1 LL(1)-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.

Zeigen oder widerlegen Sie:

(a) Jede reguläre Sprache lässt sich durch eine LL(1)-Grammatik erzeugen.

(b) Jede durch eine LL(1)-Grammatik erzeugbare Sprache ist regulär.

4.2 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.

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

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: $G \in LL(2) \setminus SLL(2)$

G: $S \to aAab \mid bAbb$
$A \to a \mid \epsilon$

4.3 Deterministischer LL(1)-TDA in Haskell

6 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:

```
type Nonterminal = Char     -- Konvention: Großbuchstaben
type Terminal    = Char     -- Konvention: Kleinbuchstaben und Sonderz.
type Ruleindex   = Int      -- Index einer Regel in einer Grammatik

data CFG = CFG {nterm :: [Nonterminal],
                term  :: [Terminal],
                start :: Nonterminal,
                rules :: [Rule]}

data Rule = Rule {leftp  :: Nonterminal,
                  rightp :: [Char]} deriving Show

type LASet = [(Ruleindex,[Maybe Terminal])] --Nothing: Epsilon
data LAnalysis = LAnalysis [Ruleindex] --Indexe der abgeleiteten Regeln
               | SynErr [Terminal] [Ruleindex] --Resteingabe Indexe
```

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

(a) Drücken Sie die Grammatik für arithmetische Ausdrücke G'_{AE} (Skript S. 37) und die dazugehörenden *la*-Mengen als `gAE :: CFG` und `la_gAE :: LASet` in Haskell aus. Ersetzen Sie die gestrichenen Buchstaben dabei mit den nächst kleineren Buchstaben im Alphabet. / 1

(b) Schreiben Sie die Funktion: / 2

```
mkAnaTable :: CFG → LASet → Maybe Terminal → Nonterminal
                  → Maybe ([Char],Int),
```

welche zu einer Grammatik und den *la*-Mengen eine Funktion definiert, mit deren Hilfe der TDA gesteuert wird. Letztere soll (wie die linke Hälfte der Analysetabelle) für ein Eingabezeichen und ein Nonterminal eine abgeleitete rechte Regelseite und die Regelnummer liefern. Dabei steht ein **`Nothing`** auf der Eingabe (3. Parameter) für das Eingabeende und ein **`Nothing`** im Ergebnis für einen undefinierten Tabelleneintrag (Fehler).

(c) Implementieren Sie einen deterministischen LL(1)-TDA: / 3

```
ll1TDA :: CFG → LASet → [Terminal] → LAnalysis.
```