

Grundlagen der Programmiersprachen und Compilerbau

Wintersemester 2007/2008

3. Übungsblatt

14. November 2007

Aufgabe 3.1

Betrachten Sie die Grammatik

$$R \rightarrow R \mid R \mid RR \mid R^* \mid (R) \mid a \mid b$$

Beachten Sie, dass der erste senkrechte Strich das „oder“-Symbol ist und kein Trenner zwischen Alternativen.

- Zeigen Sie, dass diese Grammatik alle regulären Ausdrücke über den Symbolen a und b generiert.
- Zeigen Sie, dass diese Grammatik mehrdeutig ist.
- Konstruieren Sie eine äquivalente eindeutige Grammatik, die den Operatoren $*$, Konkatenation und $|$ die auf Seite S.32 des Skriptes definierten Prioritäten und Assoziativitäten zuordnet.
- Konstruieren Sie in beiden Grammatiken einen Syntaxanalysebaum für den Satz $a|b^*c$

Aufgabe 3.2

Gegeben sei die Grammatik mit folgenden Produktionen:

$$S \rightarrow ABAB$$

$$A \rightarrow aA \mid \varepsilon$$

$$B \rightarrow bB \mid c$$

1. Bestimmen Sie die FIRST- und FOLLOW-Mengen aller Nichtterminalsymbole.
2. Stellen Sie die LL(1)-Analysematrix für diese Grammatik auf.
3. Bestimmen Sie, sofern möglich, mit Hilfe des tabellengesteuerten LL(1)-Analyseverfahrens Linksableitungen für die Wörter
 - a) $abcac$,
 - b) $aabc$ und
 - c) $bacbc$.

Aufgabe 3.3

Zeigen Sie, dass die Grammatik LL(2) ist.

$$\begin{aligned} S &\rightarrow G\$ \\ G &\rightarrow P \mid PG \\ P &\rightarrow \text{id} : R \\ R &\rightarrow \text{id} R \mid \varepsilon \end{aligned}$$

Aufgabe 3.4

Definieren Sie EBNF in EBNF. Geben Sie für die Grammatik auch die Übergangsdigramme an.

Aufgabe 3.5

Mit der Methode des rekursiven Abstiegs kann man Parser für die Klasse der LL(1)-Grammatiken implementieren.

- a) Geben Sie ein allgemeines Schema an, mit dessen Hilfe man aus einer in EBNF-Notation vorliegenden LL(1)-Grammatik einen in Ada95 formulierten Parser nach der Methode des rekursiven Abstiegs konstruieren kann. Dazu muss jeder Struktur der EBNF eine passende Anweisung zugeordnet werden. Dabei seien folgende Vereinbarungen vorgegeben:

```
package Rekursiver_Abstieg_Parser
is
  type Token_Type is (was auch immer);
  Token : Token_Type; -- enthaelt das aktuelle Token

  procedure Get_Next-Token;
  -- enthaelt den Scanner und liefert jeweils
  -- das naechste Token des Eingabetextes
  -- in der Variablen Token zurueck.

  procedure Error;
  -- diese Prozedur wird zur Fehlerbehandlung aufgerufen
end Rekursiver_Abstieg_Parser;
```

Zu Beginn der Syntaxanalyse enthalte `Token` bereits das erste Symbol. Zur Abkürzung kann folgende Notation verwendet werden: $E_1 \dots E_n$ sind EBNF-Ausdrücke, $M(E_i)$ ist der einem EBNF-Ausdruck zugeordnete Programmtext, A bezeichnet ein Nichtterminalsymbol und $F(E)$ ist die FIRST-Menge des EBNF-Ausdrucks E . `t` steht für ein Terminalsymbol.

- b) Wenden Sie das Schema auf die folgende Deklaration an:

```
CaseStatement := "CASE" Expression "OF" CaseEntry {"|" CaseEntry}
               ["ELSE" StatementSequence] "END"
```

Für das Parsen der Nichtterminale `Expression`, `StatementSequence` und `CaseEntry` existieren bereits gleichnamige Unterprogramme.