PR ÜBERSETZERBAU 621.401 – 621.402 **SS 2007**

Institut für Informationstechnologie (ITEC) **Taschwer**

Übungsblatt 04

Termin: 2007/04/17

Ü 4.1 LL(1)-Parsing

Gegeben sei die LL(1)-Grammatik G = (V, T, P, S) mit V = { A, B, S }, T = { a, b, c, d, e } und folgenden Produktionen P:

S → a A c | c S
A → d B a | b A | ε
B → e B b | ε

a) Erstellen Sie die Parsing-Tabelle (ohne Fehlerbehandlung) für einen nicht-rekursiven LL(1)-Parser der Grammatik G.

b) Als Synchronisationssymbole für die Fehlerbehandlung sollen für jedes Nichtterminal A die Symbole in FOLLOW(A) dienen. Ergänzen Sie die Parsing-Tabelle entsprechend, und erläutern Sie die Aktionen des Parsers, wenn er auf einen leeren Eintrag bzw. auf ein Synchronisationssymbol in der Parsing-Tabelle trifft.

c) Stellen Sie die Aktionen des Parsers für die Eingabeworte w = cadebac ∈ L(G) und v = abdecac ∉ L(G) tabellarisch dar. Welchem gültigen Wort v' ∈ L(G) entsprechen die Parseraktionen infolge der Fehlerbehandlung für v?

Ü 4.2 MiniCompiler – Parser

Laden Sie von http://minicompiler.itec.uni-klu.ac.at/ die Version *Stage 2 Parser* herunter und bearbeiten Sie folgende Aufgaben mit Hilfe des Source Codes:

a) Begründen Sie die Implementierung der FIRST- und FOLLOW-Mengen des Nichtterminals *declarations* durch die PL0-Grammatik. Ist das Symbol EPSILON in der Parserimplementierung überflüssig?

b) Wofür wird der *Parser.expect()* Methode ein Nichtterminal übergeben? Wie viele Fehlermeldungen würde der Parser für folgenden PL0-Code ausgeben?

```
MODULE syntax;
CONST len := a++2!x; n := 7e+05;
VAR len, x, e :: integer;
END syntax.
```

c) Worin unterscheidet sich die Fehlerbehandlung in *Parser.expectWeak()* von *Parser.expect()*? Wofür wird der *Parser.weakSeparator()* Methode ein Nichtterminal übergeben? Welche Fehlermeldungen würde der Parser für folgende Record-Definition ausgeben?

```
RECORD f1: INTEGER f2: BOOLEAN;; f3. INTEGER END
```

Ü 4.3 MiniCompiler – Symboltabelle

Laden Sie von http://minicompiler.itec.uni-klu.ac.at/ die Version *Stage 3 Symbol Table* herunter.

a) Führen Sie den MiniCompiler mit dem Befehl *ant run* für die Quelldatei *quick.pl0* aus. Es werden alle Symbole ausgegeben, die der Parser in die Symboltabelle einfügt. Suchen Sie in der Ausgabe nach dem *StructuralSymbol* der *MODULE Quick* Deklaration sowie nach den darauffolgenden Symbolen *NL*, *Array* und *a*. Stellen Sie diese Symbole in einem Objektdiagramm (analog zur VO) dar, aus dem insbesondere die Verknüpfungen durch die *upscope*, *local* und *next* Felder ersichtlich sind.

b) Wie a) für das *StructuralSymbol* der *PROCEDURE QuickSort* Deklaration sowie die darauffolgenden Symbole *left* und *middle*. Auf welche Symbole muss der MiniCompiler beim Übersetzen eines Aufrufs der Prozedur *Quicksort* zugreifen können?

c) Studieren Sie den Source Code der Methoden *Parser.putSymbol()* und *SymbolTable.getSymbol()*, und begründen Sie die Fehlermeldungen des MiniCompilers für folgende PL0-Quelldatei:

```
 1  MODULE Scoping;
 2    PROCEDURE Write(n: INTEGER);
 3    END Write;
 4    PROCEDURE proc();
 5      VAR n: INTEGER;
 6      PROCEDURE nestedproc();
 7      BEGIN
 8        Write(n);
 9      END nestedproc;
10    END proc;
11  END Scoping.
```

Fehlermeldung in Zeile 2: *Write* wurde bereits deklariert.
Fehlermeldung in Zeile 8: *n* wurde nicht deklariert.