Christian Ikenmeyer, Peter Pfahler, Florian Schoppmann, Adrian Slowik, Patrick van Lück, Björn Zeiger

Vorlesung Grundlagen der Programmiersprachen Sommer 2004 - Übungsblatt 6

Ausgabe: 13.07.2004 -- Abgabe: 21.07.2004, 13:00 Uhr, im Abgabekasten auf dem D3-Flur.

Aufgabe 16 (Rekursive Definition von Typen)

****** Aufgabe 16 und Aufgabe 18 werden bewertet! ******

Die Spedition BlitzCargo verwaltet für jedes ihrer Fahrzeuge Transportlisten, die gemäß der folgenden Grammatik aufgebaut sind:

```
/* kontextfreie Grammatik fuer BlitzCargo Transportlisten, Version 0.0 */
start   ::= list                                  /* Startsymbol start */
list    ::= element list | element                /* rechtsrekursive Liste */
element ::= 'ELEMENT' identifier integer einheit  /* einzelnes Element */
einheit ::= 'kg' | 't'                            /* moegliche Einheiten */
```

In dieser Aufgabe sollen kontextfreie Grammatiken mit rekursiven Typen modelliert werden. Eine Produktion soll durch eine Liste von Elementen des Vokabulars, eine Grammatik durch eine Liste von Produktionen dargestellt werden.

a) Wenden Sie den parametrisierten Listen-Typ der Vorlesung an, um einzelne Produktionen (Liste von Elementen des Vokabulars) und Produktionen einer Grammatik (Liste von Produktionen) darzustellen. Jeweils ein Kommentar soll wie im Beispiel einzelnen Produktionen bzw. der Grammatik zugeordnet werden können.

b) Entwickeln Sie aus der abstrakten Typdefinition Klassendefinitionen in der Programmiersprache Java. Verwenden Sie den Java-Typ `Object` anstelle von Typ-Parametern. Die Klassendefinitionen sollen neben den Definitionen der Objektvariablen zunächst keine weiteren Definitionen enthalten.

c) Erweitern Sie Ihre Klassendefinitionen um Konstruktoren und Methoden `print` zur Ausgabe. Prüfen Sie Ihre Implementierung, indem Sie Listen für die oben angegebene Grammatik konstruieren und mit `print` ausgeben.

Hinweis:
"Wenden Sie den Listentyp aus der Vorlesung an" meint hier, dass man die abstrakten Typen zur Darstellung von kontextfreien Grammatiken hinschreiben soll. So kann man z.B. eine Produktion als Liste von Symbolen des Vokabulars (Terminale/Nichtterminale) auffassen: Also würde sich für die Darstellung einer Produktion (ohne den angehängten Kommentar) dann ergeben:

```
Produktion_ohne = Vokabular list
```

wobei der Typparameter "'a" durch den Typ "Vokabular" ersetzt worden ist und die Liste Elemente genau dieses Typs besitzt. Mit angehängtem Kommentar dann:

```
Produktion = Produktion_ohne x Kommentar
```

Eine Grammatik lässt sich entsprechend als Liste von Produktionen darstellen.

Aufgabe 17 (Generische Typen)

Die Sprache Pizza erweitert Java unter anderem um generische Klassen. So kann ein Datentyp zur Speicherung für Paare von Werten so definiert werden:

```
class Paar<A,B>
{

  private A erster;
  private B zweiter;

  public Paar(A erster, B zweiter)
  { this.erster = erster;
    this.zweiter = zweiter;
  }
  public A getErster()
  { return erster;
  }
  public B getZweiter()
  { return zweiter;
  }
  public String toString()
  { return "(" + erster + ", " + zweiter + ")";
  }
}

class PizzaTest
{
  public static void main(String argv[])
  {
    Paar<int, int> nummern = new Paar(17, 01754321);
    Paar<String, int> telefon = new Paar("Schmidt", nummern.getZweiter());
    System.out.println(nummern + "/" + telefon);
  }
}
```

(Dieses Programm finden Sie auch unter
`/homes/info-f/gdp/aufgaben/blatt6/blatt6_src/PizzaTest.pizza`)

a) Geben Sie zur Klasse "Paar<A,B>" eine entsprechende abstrakte Typdefinition an. Verwenden Sie parametrisierte Typen (Polytypen).

b) Erläutern Sie, wie Sie aus dem Pizza-Programm ein Java-Programm erstellen würden.

c) Das Pizza-System übersetzt Pizza-Programme nach Java. Vergleichen Sie den Java-Code, den der Pizza-Compiler für obiges Beispiel erzeugt mit Ihren Überlegungen aus Teil b).

Compiler-Aufruf: `~gdp/java/pizza/pizza -pizza -s PizzaTest.pizza`

Aufgabe 18 (Arten der Parameterübergabe)

****** Aufgabe 16 und Aufgabe 18 werden bewertet! ******

Der Entwickler S. Schrauber hat die Aufgabe erhalten, sein erfolgreiches Java-Programm MoMoney in die ihm noch unbekannte Programmiersprache MYSTERY zu übersetzen. Da die mitgelieferte Doku-CD plötzlich Lesefehler produziert, kann er die Dokumentation zu MYSTERY unglücklicherweise nicht einsehen! Auf der CD-Hülle ist allerdings die kontextfreie Grammatik von MYSTERY abgedruckt:

```
Program    ::=   DeclList
DeclList   ::=   Decl ';' DeclList |
Decl       ::=   'VAR' id ':' Type | TYPE id '=' Type | ProcDecl
ProcDecl   ::=   'PROCEDURE' id '(' Formals ')' ':' Type '=' Block
Formals    ::=   FormalList |
FormalList ::=   Formal | FormalList ';' Formal
Formal     ::=   id ':' Type
Type       ::=   'INTEGER' | SubrTy | ArrayTy | id | ProcTy
SubrTy     ::=   '[' Number 'TO' Number ']'
ArrayTy    ::=   'ARRAY' SubrTy 'OF' Type
ProcTy     ::=   'PROCEDURE' '(' Formals ')' ':' Type
```

```
Block      ::=   DeclList ’BEGIN’ StmtList ’END’
StmtList   ::=   Stmt | Stmt ’;’ StmtList |
Stmt       ::=   Assignment | Return | Block | Cond | Iteration | Output
Assignment ::=   Expr ’:=’ Expr
Return     ::=   ’RETURN’ Expr
Cond       ::=   ’IF’ Expr ’THEN’ StmtList ’ELSE’ StmtList ’END’
Iteration  ::=   ’WHILE’ Expr ’DO’ StmtList ’END’
Output     ::=   ’PRINT’ Expr
Expr       ::=   Operand | Expr Operator Operand
Operand    ::=   Number | id |  Operand ’[’ Expr ’]’ | Operand ’(’ Actuals ’)’ | ’(’ Expr ’)’
Operator   ::=   ’+’ | ’>’ | ’AND’
Actuals    ::=   ActualList |
ActualList ::=   Expr | Actuals ’,’ Expr
```

Hinweis:
`Program` ist das Startsymbol der Grammatik, die in EBNF-Form angegeben ist. Die Symbole ’[’ und ’]’ gehören zu den Terminalen der Sprache und stehen nicht für optionale Konstruktionen in EBNF. Die Ausführung eines MYSTERY-Programms beginnt in einer zu definierenden Prozedur `main`.

Herr S. Schrauber möchte zunächst herausfinden, welche Variante der Parameterübergabe in MYSTERY verwendet wird. Geben Sie ein möglichst einfaches Testprogramm in der Programmiersprache MYSTERY an, mit dem S. Schrauber herausfinden kann, ob MYSTERY

- call-by-value
- call-by-reference
- call-by-value-and-result

verwendet. Testen Sie Ihr Programm mit dem *Programming Language Detective* (PL Detective). Begründen Sie Ihre Vorgehensweise und geben Sie Testprogramm und Programmausgabe ab.

Hinweis:
Der PL Detective kann über ein HTML-Formular bedient werden. Dazu wird das zu übersetzende und auszuführende Programm in das Eingabefeld übertragen und über den Submit-Button an den PL Detective übermittelt. Der PL Detective sendet dann unmittelbar Antworten des Übersetzers (Syntax-Fehler, Fehler in der stat. Semantik, Laufzeitfehler und natürlich Programmausgabe) an den Browser zurück. Bitte das vorgegebene Passwort verwenden!

Bei technischen Problemen bitte Email an adrian@uni-paderborn.de!