Programmation Avancée et Compilation 2001
(Exercises)

8 Visitors

Keywords 8

Give short explanations of the following keywords:

Object-oriented decomposition (Abstract Syntax 11), *visitor* (Abstract Syntax 16).

Exercise 15

Consider the following concrete syntax for sequences of commands:

```
Sequence    ::=  Command { ";" Sequence }
                 .
Command     ::=  "if" Expression "then" Command [ "else" Command ]
              |  IDENT ":=" Expression
              |  "{" Sequence "}"
                 .
Expression  ::=  NUMLIT | BOOLLIT | IDENT
                 .
```

(a) To get accustomed to the syntax, give the concrete syntax-tree for the sequence

y := 0 ; **if** x then y:=1 **else** { y:=0 }

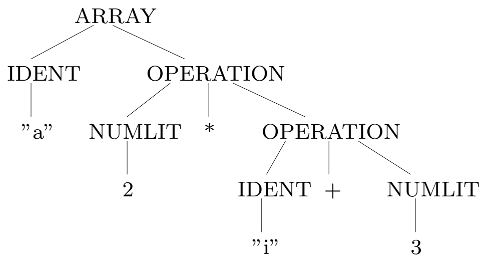
(b) Find a suitable abstract syntax for the above grammar for sequences.

(c) Give the abstract syntax-tree for the sequence from part (a) of this exercise.

Exercise 16

Consider again the abstract syntax for expressions with arrays from Exercise 13, as well as the abstract syntax-tree for a[2∗(i+3)]:

```
Expression  ::=  OPERATION Expression Expression Op
              |  NUMLIT int
              |  IDENT String
              |  ARRAY Expression Expression
                 .
```



Further consider the following visitor class:

```
public class PrintIdents implements Tree.Visitor {
      public static void prid(Tree tree) {
            tree.apply(new PrintIdents());
      }
      public static void caseOp(Tree.Operation tree) { prid(tree.left); prid(tree.right); }
      public static void caseNl(Tree.NumLit tree) { }
      public static void caseId(Tree.Ident tree) { System.out.print(tree.id + " "); }
      public static void caseAr(Tree.Array tree) { prid(tree.left); prid(tree.right); }
}
```

(a) Describe in your own words the effect of applying PrintIdents.prid to an abstract syntax-tree.

(b) Apply the visitor to the above example, and construct the call-tree for it. A call-tree specifies which function calls which one.