Programmation Avancée et Compilation 2001
(Exercises)

7 Abstract Syntax

Keywords 7

Give short explanations of the following keywords:

Abstract syntax (Abstract Syntax 3), *concrete syntax* (Abstract Syntax 3).

Exercise 13

Consider the following two grammars for expressions with arrays. The first one specifies a *concrete syntax*, whereas the second one specifies an *abstract syntax*.

```
Expression  ::=  Expression "+" Term | Term
Term        ::=  Term "*" Factor | Factor
Factor      ::=  NUMLIT | IDENT | "(" Expression ")" | Factor "[" Expression "]"
```

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

As you know, the parser of a compiler generates a *concrete syntax-tree (parse-tree)* according to the first grammar. Then this representation is then transformed into an *abstract syntax-tree* according to the second grammar, upon which all further reasoning is conducted.

(a) Construct both the concrete and the abstract syntax-tree for the expression a[2*(i+3)].

(b) Why is further work based on the abstract syntax-tree, and not on the concrete one?

(c) Why is the specification of a parser based on a concrete syntax instead on an abstract one?

Remark: The terms *concrete syntax-tree* and *parse-tree* are interchangeable.

Exercise 14

The following grammar is not suitable for top-down parsing.

```
List     ::=  "[" "]" | "[" Content "]"
Content  ::=  Content "," IDENT | IDENT
```

(a) What are the problems of a top-down parser based on this grammar?

(b) Rewrite the grammar so to make it viable to top-down parsing.