### 5.3.2 The Structure of a Type

Inherited attributes are useful when the structure of the parse tree differs from the abstract syntax of the input; attributes can then be used to carry information from one part of the parse tree to another. The next example shows how a mismatch in structure can be due to the design of the language, and not due to constraints imposed by the parsing method.

**Example 5.13:** In C, the type **int** [2][3] can be read as, "array of 2 arrays of 3 integers." The corresponding type expression *array*(2, *array*(3, *integer*)) is represented by the tree in Fig. 5.15. The operator *array* takes two parameters, a number and a type. If types are represented by trees, then this operator returns a tree node labeled *array* with two children for a number and a type.

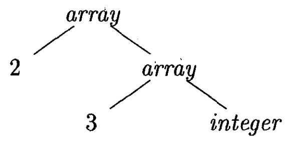

Figure 5.15: Type expression for **int**[2][3]

With the SDD in Fig. 5.16, nonterminal $T$ generates either a basic type or an array type. Nonterminal $B$ generates one of the basic types **int** and **float**. $T$ generates a basic type when $T$ derives $B\,C$ and $C$ derives $\epsilon$. Otherwise, $C$ generates array components consisting of a sequence of integers, each integer surrounded by brackets.

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $T \rightarrow B\ C$ | $T.t = C.t$<br>$C.b = B.t$ |
| $B \rightarrow$ **int** | $B.t = integer$ |
| $B \rightarrow$ **float** | $B.t = float$ |
| $C \rightarrow$ [ **num** ] $C_1$ | $C.t = array(\textbf{num}.val,\ C_1.t)$<br>$C_1.b = C.b$ |
| $C \rightarrow \epsilon$ | $C.t = C.b$ |

Figure 5.16: $T$ generates either a basic type or an array type

The nonterminals $B$ and $T$ have a synthesized attribute $t$ representing a type. The nonterminal $C$ has two attributes: an inherited attribute $b$ and a synthesized attribute $t$. The inherited $b$ attributes pass a basic type down the tree, and the synthesized $t$ attributes accumulate the result.

An annotated parse tree for the input string **int** [ 2 ] [ 3 ] is shown in Fig. 5.17. The corresponding type expression in Fig. 5.15 is constructed by passing the type *integer* from $B$, down the chain of $C$'s through the inherited attributes $b$. The array type is synthesized up the chain of $C$'s through the attributes $t$.

In more detail, at the root for $T \rightarrow B\ C$, nonterminal $C$ inherits the type from $B$, using the inherited attribute $C.b$. At the rightmost node for $C$, the production is $C \rightarrow \epsilon$, so $C.t$ equals $C.b$. The semantic rules for the production $C \rightarrow$ [ **num** ] $C_1$ form $C.t$ by applying the operator *array* to the operands **num**.*val* and $C_1.t$. □

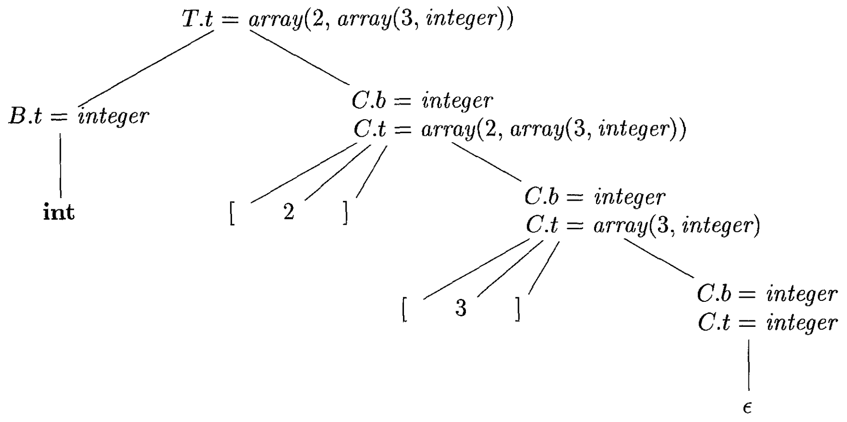
$T.t = array(2, array(3, integer))$
$B.t = integer$
int
$C.b = integer$
$C.t = array(2, array(3, integer))$
[
2
]
$C.b = integer$
$C.t = array(3, integer)$
[
3
]
$C.b = integer$
$C.t = integer$
$\epsilon$

Figure 5.17: Syntax-directed translation of array types