

# Semantic Analysis

## 4.8 Exercises

- 4.25 Repeat Exercise 4.7 using ad hoc attribute space management. Instead of accumulating the translation into a data structure, however, write it to a file on the fly.
- 4.26 Rewrite the grammar for declarations of Example ©4.28 without the requirement that your attribute flow be L-attributed. Try to make the grammar as simple and elegant as possible (you shouldn't need to accumulate lists of identifiers).
- 4.27 Fill in the missing lines in Figure ©4.19.
- 4.28 Consider the following grammar with action routines.

```
params  $\rightarrow$  mode ID par_tail
           { params.list := insert( $\langle$ mode.val, ID.name $\rangle$ , par_tail.list) }
par_tail  $\rightarrow$  , params { par_tail.list := params.list }
            $\rightarrow$  { par_tail.list := null }
mode  $\rightarrow$  IN { mode.val := IN }
            $\rightarrow$  OUT { mode.val := OUT }
            $\rightarrow$  IN OUT { mode.val := IN OUT }
```

Suppose we are parsing the input IN a, OUT b, and that our compiler uses an automatically maintained attribute stack to hold the active slice of the parse tree. Show the contents of this attribute stack immediately before the parser predicts the production *par\_tail*  $\rightarrow$   $\epsilon$ . Be sure to indicate where lhs and rhs point in the attribute stack. Also show the stack of saved lhs and rhs values, showing where each points in the attribute stack. You may ignore the ssf (seen so far) pointer.

- 4.29 One problem with automatic space management for attributes in a top-down parser occurs in lists and sequences. Consider for example the following grammar:

$$\begin{aligned}
 \text{block} &\longrightarrow \text{begin stmt\_list end} \\
 \text{stmt\_list} &\longrightarrow \text{stmt stmt\_list\_tail} \\
 \text{stmt\_list\_tail} &\longrightarrow ; \text{ stmt\_list} \mid \epsilon \\
 \text{stmt} &\longrightarrow \dots
 \end{aligned}$$

After predicting the final statement of an  $n$ -statement block, the attribute stack will contain the following (line breaks and indentation are for clarity only):

```

block begin stmt_list end
  stmt stmt_list_tail ; stmt_list
  stmt stmt_list_tail ; stmt_list
  stmt stmt_list_tail ; stmt_list
  {  $n$  times }

```

If the attribute stack is of finite size, it is guaranteed to overflow for some long but valid block of straight-line code. The problem is especially unfortunate since, with the exception of the accumulated output code, none of the repeated symbols in the attribute stack contains any useful attributes once its substructure has been parsed.

Suggest a technique to “squeeze out” useless symbols in the attribute stack, dynamically. Ideally, your technique should be amenable to automatic implementation, so it does not constitute a burden on the compiler writer.

Also, suppose you are using a compiler with a top-down parser that employs an automatically managed attribute stack, but does not squeeze out useless symbols. What can you do if your program causes the compiler to run out of stack space? How can you modify your program to “get around” the problem?