Exercise 5

Consider a language of concatenated instructions. Instructions can be blocks or other kind of instructions. A block consists of a declaration of variables, which can be of type integer or real, followed by a sequence of instructions.

Define a Syntax Directed Translation Scheme suitable for being implemented during top-down parsing that computes, for each instruction, an attribute *depth* and an attribute *occupation*. The depth is the number of blocks in which the instruction is enclosed and the occupation, for a block, is the maximum number of bytes needed to store all the variables declared in the block and in its sub-blocks. The space used for a sub-block S can be reused by another sub-block S' when the block S is closed. Integer variables occupy two bytes, real variables occupy four bytes and other kind of instructions occupies zero bytes.

For example, in the following case:

```
begin
   var a,b,c: integer;
   S;
   begin
      var d: real;
      S'
   end;
   begin
      var e: integer;
      S";
      S'"
   end;
end;
```

supposing that instructions S, S', S" and S'" are not blocks, the expected value of the attributes are: external block: depth = 0, occupation = 10; instruction S: depth = 1, occupation = 0; first internal block: 1, 4; instruction S': 2, 0; second internal block: 1, 2; S" and S'": 2, 0.

The use of global data structures is forbidden.

Hints

Proceeds in the following order:

1. Define an LL(1) grammar and check its correctness
2. The requested analyses can be conducted independently from each others:

a. Define two synthesised attributes for the computation of the memory occupation
b. Define one synthesised attribute and one inherited attribute to compute the number of sub-block nesting

Solution

Let's use the following grammar (verify that it is LL(1)):

P*rogram*::= B*lock*
B::= **begin** D*eclaration* ; C*ommand* R*est_of_program* **end**
R::= ; C R
R::= ε
C::= B*lock*
C::= S*tatement*
D::= **var id** L*ist_of_identifiers* : T*ype*
L ::= , **id** L
L::= ε
T ::= **real**
T ::= **int**

The following attributes are used:

- **M** for B, D, R, C, T is a synthesized attribute containing a number of bytes
  - in case of B: the bytes required for the declaration and the maximum number of bytes for the blocks occurring in the body of B
  - in case of D: the bytes required for the variables in D
  - in case of R, C: bytes required by the associated blocks or statements
  - in case of T: bytes for one integer or one real
- **n** for L is a synthesised attribute containing the number of variables for computing the level of scoping (depth)
- **in** for B, C, R is an inherited attribute containing the level at which the associated structure is nested
- **H** for B, C is a synthesised attribute containing the depth of the associated structure

P::= {B.in:=0} **B**
B::= begin **D**; {C.in:= B.in +1} **C** {R.in:= B.in +1} **R** end {B.M:= D.M + max(C.M, R.M); B.H:= B.in +1}

R_1::= ; {C.in:= R_1.in} **C** {R_2.in:= R_1.in} **R_2** {R_1.M:= max(C.M, R_2.M)}
R::= ε {R.M:= 0}
C::= {B.in:= C.in} **B** {C.M:= B.M; C.H:= C.in}
C::= **S** {C.M:= 0; C.H:= C.in}
D::= var id **L** : **T** {D.M:= (1+L.n) * T.M}
L_1 ::= , id **L_2**{L_1.n:= 1 + L_2.n}
L::= ε {L.n:= 0}
T ::= real {T.M:= 4}
T ::= int {T.M:= 2}