**Exercise: 6.6**

Consider the following grammar for integer binary trees (in linearized form):

***btree ( number btree btree ) | nil***

Write an attribute grammar to check that a binary tree is ordered, that is, that the values

of the numbers of the first subtree are the value of the current number and the values

of all the numbers of the second subtree are the value of the current number. For

example, (2 (1 nil nil) (3 nil nil)) is ordered, but (1 (2 nil nil)(3 nil nil)) is not.

**Solution**:

Attribute  good  is true if the tree is ordered, and false if not. Attribute smallest is the smallest value in the tree, and  largest  is the largest value in the tree. In the case of an empty tree, largest is -infinity and smallest is +infinity. All of these attributes are synthesized.

Btree 2

1 3

nil nil

|  |  |
| --- | --- |
| **Grammar Rule** | **Semantic Rule** |
| btree ( number btree1 btree2 ) | btree.largest = max(btree2.largest, number.val)  btree.smallest = min(btree1.smallest, number.val)  btree.good = btree1.good & btree2.good &  number.val > btree1.largest &  number.val < btree2.smallest |
| btree nil | |  | | --- | | btree.largest = - infinity | | btree.smallest = + infinity | | btree.good = true | |

**Exercise 6.10**

A. The grammar rules of Example 6.14

S → Exp

Exp → Exp / exp | **Num | num.num**

 ---------------------------------------------------------------------

The dependency graphs for each grammar rule:

 S → Exp

Val  S

 IsFloat Etype val Exp

-------------------------------------

Exp → Exp / exp

  IsFloat   Etype  Val Exp

IsFloat   Etype  Val  Exp / IsFloat   Etype  Val Exp

-------------------------------------------------------------------------

 Exp

IsFloat  Etype  Val  Exp

  Val ***Num***

--------------------------------------------------------------

Exp → **Num.num**

IsFloat  Etype  Val  Exp

  Val **Num. num**

----------------------------------------------------------------------------------------------------------------

B.

The dependency graphs for the expression: 5/2 / 2.0

Val  S

 IsFloat Etype Val Exp

  IsFloat   Etype  Val  Exp / IsFloat Etype Val Exp

  Val **Num.num**

IsFloat   Etype  Val Exp / IsFloat Etype Val Exp

(2.0)

Val Num Val Num

(5) ( 2 )

-------------------------------------------------------------------------------------------------------------------

C.

The pseudcode procedure for the computation of the isFloat.

Function EvalisFloat (T: treenode): Boolean

Var temp1, temp2: Boolean

Begin

Case nodekind of T of  Exp:

Temp1 = EvalisFloat (left child of T);

If right child of T is not nil then

Temp2 = EvalisFloat (right child of T)

Return temp1 or temp2

Else

Return temp1;

Num:

Return false

Num.num:

Return true;

End

Function Evalval (T: treenode, etype: integer): VALUE

Var temp1, temp2: VALUE

Begin

Case nodekind of T of  S:

Return (Evalval (left child of T, etype));

Exp:

If etype = EMPTY then

If EvalisFloat (T) then etype = FLOAT

Else etype = INT

----------------------------------------------------------------------------------------------------------------------------------

Exercise 6.13

Consider the following attribute grammar:

|  |  |
| --- | --- |
| **Grammar Rule** | **Semantic Rule** |
| S->ABC | B.u=S.u  A.u=B.v + C.v  S.v=A.v |
| A->a | A.v = 2 \* A.u |
| B -> b | B.v = B.u |
| C -> c | C.v = 1 |

1. Draw the parse tree for the string abc (the only string in the language) and draw the dependency graph for the associated attributes. Describe a correct order for the evaluation of the attributes

Ans:



1. Suppose that S.u is assigned the value 3 before attribute evaluation begins. What is the value S.v when evaluation has finished?

Ans: S.v = 8

Suppose the attribute equations are modified as follows:

|  |  |
| --- | --- |
| **Grammar Rule** | **Semantic Rule** |
| SABC | B.u=S.u  C.u=A.v.  A.u=B.v. + C.v.  S.v=A.v |
| Aa | A.v = 2 \* A.u |
| B b | B.v = B.u |
| C c | C.v = C.u - 2 |

1. What value does S.v have after attribute evaluation, if S.u = 3 before evaluation begins?



S.v. can not be calculated.

----------------------------------------- THE END ------------------------------------------------