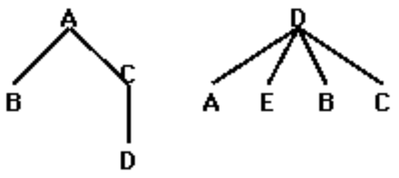
Exercise 1

Consider the language of forests of labelled trees represented as parenthesized lists of trees separated by commas. Each tree that is not just a leaf is represented by a pair of elements, separated by comma and enclosed in square brackets. The first of these elements is the label of the root; the second element is the forest of trees that are the children of the root. Trees that are composed of only one node (leaf) are represented by just the label of the node. The alphabet of the labels is {A, B, C, D, E}.
An example of a sentence belonging to the language is

([A,(B,[C,(D)]], [D,(A,E,B,C)])

which represents the following forest



Your tasks are:

1. Give a grammar for the language
2. Give an Syntax Directed Definition suitable to be implemented during top-down parsing that computes, for each non-leaf node, an attribute that is the parse tree derived from the node. For the construction of the tree the following operations can be used:
    1. MkT:: Label x TreeList -> Tree /* MkT(a,f) creates a tree with a root labelled "a" and whose children are the trees in "f", in the order in which they occur */
    2. MkE:: -> TreeList /* creates an empty list of trees */
    3. Add:: Tree x TreeList -> TreeList /* Add(t,f) adds the tree "t" to "f" as first element */
3. Give an SDD suitable to be implemented during top-down parsing that computes, as an attribute of the start symbol, the derived forest of trees. Use the operations above; in particular use list of trees for forests.
4. Give an SDD suitable to be implemented during top-down parsing that computes, as an attribute of the start symbol, the depth of the forest represented by the parsed string (i.e., the maximum height of the trees in the forest).
5. Give an SDD suitable to be implemented during top-down parsing that computes, as an attribute of the start symbol, the width of the forest represented by the parsed string (i.e., the maximum number of children of a node in the trees of the forest).
6. Give an SDD suitable to be implemented during top-down parsing that computes, as an attribute of the start symbol, the number of nodes in the forest that are labelled with "A".

7. Give an SDD suitable to be implemented during top-down parsing that computes, as an attribute of the start symbol, the maximum depth of the trees (and subtrees) in the forest whose root is labelled with “A”.
8. Give an SDD suitable to be implemented during top-down parsing that computes, as an attribute of the start symbol, the maximum width of the trees (and subtrees) in the forest whose root is labelled with “A”.

Hints

Exercise 1.1

Among the different options, it is convenient to give an LL(1) grammar because the analyses requested in the next steps all require that kind of grammar.

Exercise 1.2

It is sufficient to use a synthesised attribute *tree*. During parsing, for each non-leaf node P, the attribute P.*tree* will be assigned with the corresponding parse-tree constructed with the given operations.

Exercise 1.3

Introduce three synthesised attributes:

- **forest** for S and F: contains the list of trees coming from the traversal of the derivation tree for S and F, respectively
- **tree** for T: contains the tree derived during the traversal of the derivation tree of T
- **label** for L: contains the label of L

Exercise 1.4

Introduce a synthesised attribute **depth** for S, F, T. It contains the number of edges of the longer path contained in the forest or tree associated to S, F or T.

Exercise 1.5

Introduce two synthesised attributes:

- **outd** for S, F, T: contains the maximum number of children of the nodes in the forest or tree associated to S, F or T
- **length** for F: contains the number of trees (that is the length of the list of children) in the forest associated to F

Exercise 1.6

Introduce a synthesised attribute **A** for S, F, T, L that contains the number of nodes labelled with “A” in the forest or tree associated to the symbol.

Exercise 1.7

Introduce three synthesised attributes:

- **DepthA** for S, F, T: contains the maximum depth of the trees labelled with “A” in the forest or tree associated to the symbol
- **depth** for S, F, T as in 1.4
- **label** for L as in 1.3

Exercise 1.8

Introduce four synthesised attributes:

- **OutdA** for S, F, T: contains the maximum number of children for trees with root labelled with “A” in the forest or tree associated to the
- **outd** for F, T as in 1.5
- **length** for F as in 1.5
- **label** for L as in 1.3

Solutions

Exercise 1.1

```
S::= ( T F
T::= L
T::= [ L , S ]
F::= , T F
F::= )
L::= A
L::= B
L::= C
L::= D
L::= E
```

Esercizio1.2

S::= (TF	S.tree:= MkT("S", Add(MkT("(",MkE()), Add(T.tree, Add(F.tree, MkE()))))

T::= L	T.tree:= MkT("T",Add(L.tree, MkE()))
T::= [L,S]	T.tree:= MkT("T",Add(MkT("[",MkE()), Add(L.tree, Add(MkT(",", MkE()),Add(S.tree, Add(MkT("]", MkE()))))))
F_1::=,TF_2	F_1.tree:=MkT("F",Add(MkT(",",MKE()),Add(T.tree,Add(F_2.tree,MKE()))))
F::=)	F.tree:=MkT("F",Add(MkT(")",MKE()))
L::= A	L.tree:=MkT("L",Add(MkT("A",MKE()))
L::= B	L.tree:=MkT("L",Add(MkT("B",MKE()))
L::= C	L.tree:=MkT("L",Add(MkT("C",MKE()))
L::= D	L.tree:=MkT("L",Add(MkT("D",MKE()))
L::= E	L.tree:=MkT("L",Add(MkT("E",MKE()))

Exercise 1.3

S::= (TF	S.forest:= Add(T.tree, Add(F.forest, MkE())))
T::= L	T.tree:= MkT(L.label,MkE())
T::= [L,S]	T.tree:= MkT(L.label, S.forest)
F_1::=,TF_2	F_1.forest:= Add(T.tree, F_2.forest)
F::=)	F.forest:= MKE()
L::= A	L.label:= "A"
L::= B	L.label:= "B"
L::= C	L.label:= "C"
L::= D	L.label:= "D"
L::= E	L.label:= "E"

Exercise 1.4

S::= (TF	S.depth:= max(T.depth, F.depth)
T::= L	T.depth:= 0
T::= [L,S]	T.depth:= 1+ S.depth
F_1::=,TF_2	F_1.depth:= max(T.depth, F_2.depth)
F::=)	F.depth:= 0
L::= A	

L::= B	
L::= C	
L::= D	
L::= E	

Exercise 1.5

S::= (TF	S.length:= 1+F.length; S.outd:= max(T.outd, max (1+F.length, F.outd))
T::= L	T.outd:= 0
T::= [L,S]	T.outd:= max(S.length, S.outd)
F_1::=,TF_2	F_1.length:= 1+ F_2.length F_1.outd:= max(T.outd,F_2.outd)

F::=)	F.length:= 0
	F.outd:= 0

L::= A	
L::= B	
L::= C	
L::= D	
L::= E	

Exercise 1.6

S::= (TF	S.A:= T.A+F.A
T::= L	T.A:= L.A
T::= [L,S]	T.A:= L.A+S.A
F_1::=,TF_2	F_1.A:= T.A+ F_2.A
F::=)	F.A= 0
L::= A	L.A:= 1
L::= B	L.A:= 0
L::= C	L.A:= 0

L::= D	L.A:= 0
L::= E	L.A:= 0

Exercise 1.7

S::= (TF	S.DepthA:= max(T.DepthA,F.DepthA); S.depth:= max(T.depth, F.depth)

T::= L	T.DepthA:= 0
	T.depth:= 0

T::= [L,S]	T.DepthA:= if (L.label = "A") then 1+S.depth else S.DepthA
	T.depth:= 1+ S.depth

F_1::=,TF_2	F_1.DepthA:= max(T.DepthA, F_2.DepthA) F_1.depth:= max(T.depth, F_2.depth)

F::=)	F.DepthA:= 0
	F.depth:= 0

L::= A	L.label:= "A"
L::= B	L.label:= "B"
L::= C	L.label:= "C"
L::= D	L.label:= "D"
L::= E	L.label:= "E"

Exercise 1.8

S::= (TF	S.OutdA:= max(T.OutdA,F.OutdA);
	S.length:= 1+F.length; S.outd:= max(T.outd, max (1+F.length, F.outd))

T::= L	T.OutdA:= 0
	T.outd:= 0

T::= [L,S]	T.OutdA:= if (L.label = "A") then max(max(S.length, S.outd), S.OutdA) else S.OutdA
	T.outd:= max(F.length, F.outd) **nota che S.OutdA<S.outd sempre: espressione max puo' essere semplificata

F_1::=,TF_2	F_1.OutdA:= max(T.OutdA, F_2.OutdA)

	F_1.outd:= max(T.outd,F_2.outd) F_1.length:= 1+ F_2.length

F::=)	F.OutdA:= 0
	F.outd:= 0
	F.length:= 0

L::= A	L.label:= "A"
L::= B	L.label:= "B"
L::= C	L.label:= "C"
L::= D	L.label:= "D"
L::= E	L.label:= "E"