7.Deep lists

The deep list type in prolog represents a recursive structure, where several lists of variable depth are nested one in another. Formally a deep list can be defined by:

$$DL = [H|T], where\ H \in \{atom, list, deep\ list\}\ and\ T\ is\ a\ deep\ list.$$

A trivial case of a deep list is a simple (or shallow) list.

Examples of deep lists include:

a. L1 = [1,2,3,[4]].
b. L2 = [[1],[2],[3],[4,5]]
c. L3 = [[],2,3,4,[5,[6]],[7]].
d. L4 = [[[[1]]],1, [1]].
e. L5 = [1,[2],[[3]],[[[4]]],[5,[6,[7,[8,[9],10],11],12],13]].
f. L6= [alpha, 2,[beta],[gamma,[8]]].

7.1 Simple operations with deep lists

All operations defined for shallow lists can also be used with deep lists including (but not only) the studied member, append and delete predicates. To understand how operations on deep lists work, one can consider a deep list equivalent to a shallow list with different types of elements, but only those on the first level.

Exercise 7.1

Using the list L5 defined above try to give the result of the following queries:

a. ? - member(2 ,L5).
b. ? – member([2] , L5).
c. ? – member(X, L5).
d. ? – append(L1,R,L2).
e. ? – append(L4,L5,R).
f. ? – delete(1, L4,R).
g. ? – delete(13,L5,R).

Execute the queries in Prolog to check your results.

7.2 Advanced operations with deep lists

7.2.1 The atomic predicate

To execute different operations on all elements of a deep list we have to know how to treat these elements according to their type(if they are atoms we process them, if they are lists further decomposition might be needed before processing). To find out if a given element is an atom or a complex structure the built in ***atomic*** predicate can be used.

Exercise 7.2.

Answer then execute the following queries:

a. ? – atomic(apple).
b. ? – atomic(4).
c. ? – atomic(X).
d. ? – atomic(apple(2)).
e. ? – atomic([1,2,3]).
f. ? – atomic([]).

7.2.2. The depth of a deep list

The depth represents the maximum nesting level in the case of a deep list. The depth of an atom is defined as 0, and the depth of a shallow list(including the empty list) as 1. When computing the maximum depth we will have three branches:

a. We arrived at the empty list. The depth is 1.
b. We have an atomic head, we ignore it, since it doesn't influence the depth. The depth of the list will be equal to the depth of the tail.
c. We have a list in the head of the deep list. In this case the depth of the list will be either the depth of the tail, or the depth of the head increased by one(think about why).

The predicate corresponding to the description is:

```
depth([],1).
depth([H|T],R):-atomic(H),!,depth(T,R).
depth([H|T],R):- depth(H,R1), depth(T,R2), R3 is R1+1, max(R3,R2,R).
```

Exercise 7.3.

Trace the execution of the following queries for the predicate depth, and the lists L1-6 defined as above:

a. ? – depth(L1,R).
b. ? – depth(L2,R).
c. ? – depth(L3,R).
d. ? – depth(L4,4).
e. ? – depth(L5,R).
f. ? – depth(L6,4).

7.2.3. Flattening a deep list

This operation means obtaining an equivalent shallow list from a deep list, containing all the elements, but with nesting level 1. In order to do this, we take only the atomic elements from the source list and place them in the result.

We have again 3 main cases:

a. We have to deal with the empty list. Flattening the empty list results in an empty list.
b. If the first element of the list is atomic we put it into the result, and process the rest of the list.
c. If the first element is not atomic, the result will be composed of all the atomic elements(or flattening) of the head, and all the atomic elements(flattening) of the tail.(How do we collect the two results in a single list?)

The solution is:

```
flatten([],[]).
flatten([H|T], [H|R]) :- atomic(H),!, flatten(T,R).
flatten([H|T], R) :- flatten(H,R1), flatten(T,R2), append(R1,R2,R).
```

Exercise 7.4.

Trace the execution of the following queries:

a. ? – flatten(L1,R).
b. ? – flatten(L2,R).
c. ? – flatten(L3,R).
d. ? – flatten(L,[1,2,3,4]).

e. ? – flatten(L5,R).
f. ? – flatten(L6, [alpha,2,beta,gamma,8]).

7.2.4. List heads

Returns all atomic elements which are at the head of a shallow list.

Several solutions exist, but we will present an efficient solution, which uses a flag to determine if we are at the first element of a list.

```
heads3([],[],_).
heads3([H|T],[H|R],1):-atomic(H),!,heads3(T,R,0).
heads3([H|T],R,0):-atomic(H),!,heads3(T,R,0).
heads3([H|T],R,_):-heads3(H,R1,1),heads3(T,R2,0), append(R1,R2,R).
heads_pretty(L,R) :- heads(L, R,1).
```

Exercise 7.4.

Trace the execution of the following queries:

a. ? – heads(L1,R).
b. ? – heads(L2,R).
c. ? – heads(L3,R).
d. ? – heads(L6,R).
e. ? – heads(L5,R).
f. ? – heads(L,[1,2,3,4,5]).

7.2.5 The nested member function

Works similarly to the member function in the case of the shallow lists, considers as member all elements appearing in the list, atomic or not, at any level.

```
member1(H,[H|_]).
member1(X,[H|_]):-member1(X,H).
member1(X,[_|T]):-member1(X,T).
```

Exercise 7.6.

Trace the execution of the following queries:

a. ? – member1(1,L1).

b. ? – member1(4,L2).
c. ? – member1([5,[6]], L3).
d. ? – member1(X,L4).
e. ? – member1(X,L6).
f. ? – member1(14,L5).

Observation: If we want our nested function to find only atomic elements we can use the flattening of the list to obtain a short solution:

```
member2(X,L):- flatten(L,L1), member(X,L1).
```

7.3 Quiz exercises

7.3.1. Define a predicate which computes the number of atomic elements in a deep list.

7.3.2. Define a predicate computing the sum of atomic elements from a deep list.

7.3.3. Define the deterministic version of the member predicate.

7.4 Problems

7.4.1. Define a predicate returning the elements from a deep lists, which are at the end of a shallow list (immediately before a ']').

7.4.2. Write a predicate which replaces an element/list/deep list in a deep list with another expression.

7.4.3.** Define a predicate ordering the elements of a deep list by depth (when 2 sublists have the same depth, order them in lexicographic order – after the order of elements).

Hints:

- L1< L2, if L1 and L2 are lists, or deep lists, and the depth of L1 is smaller than the depth of L2.
- L1<L2, if L1 and L2 are lists, or deep lists with equal depth, all the elements up to the k-th are equal, and the k+1-th element of L1 is smaller than the k+1th element of L2.