Seminar 2 – Lists in Prolog

1. Write a predicate to remove from a list all the elements that appear only once. For example: for [1,2,1,4,1,3,4] the result will be [1,1,4,1,4].

- How do we determine if an element appears only once? We need a predicate to count the number of occurrences of an element in a list.
- In order to have the correct answer, nrOccurrences has to be called for the initial list, instead of the list from which we kept eliminating elements during the recursive call. Otherwise, when we get to the last occurrence of an element, the number of occurrences in the rest of the list will be 0.

$$nrOccurrences(l_1 l_2 \dots l_n, e) = \begin{cases} 0, if\ n = 0 \\ 1 + nrOccurences(l_2 l_3 \dots l_n, e), if\ l_1 = e \\ nrOccurences(l_2 l_3 \dots l_n, e), otherwise \end{cases}$$

```
% nrOccurrences(L:list, E:el, S:integer)
% flow model: (i, i, o) or (i, i, i)
% L – list in which we count the occurrences
% E – the element we are looking for
% S – the result, the number of occurrences

nrOccurrences([], _, 0).
nrOccurrences([H|T], E, S):-
      H = E,
      nrOccurrences(T, E, S1),
      S is S1 + 1.
nrOccurrences([H|T], E, S):-
      H \= E,
      nrOccurrences(T, E, S).
```

$$remove(l_1 l_2 \dots l_n, L_1, L_2 \dots L_m) = \begin{cases} \emptyset, if\ n = 0 \\ remove(l_2 \dots l_n, L_1 L_2 \dots L_m), if\ \text{nrOccurrences}(L_1 L_2 \dots L_n, l_1) = 1 \\ l_1 \cup remove(l_2 \dots l_n, L_1 L_2 \dots L_m), otherwise \end{cases}$$

```
% remove(L: List, LO:List, R:List)
% flow model: (i, i, o) or (i, i, i)
% L – list from which we remove elements
% LO – copy of the original list,this is where we count occurrences
% R – resulting list

remove([], _, []).
```

```
remove([H|T], LO, R):-
	nrOccurrences(LO, H, S),
	S = 1,
	remove(T, LO, R).
remove([H|T], LO, [H|R]):-
	nrOccurrences(LO, H, S),
	S > 1,
	remove(T, LO, R).
```

- We need another function to initialize the copy of the original list

$$removeMain(l_1 l_2 \dots l_n) = remove(l_1, l_2 \dots l_n, l_1 l_2 \dots l_n)$$

```
% removeMain(L: List, R:List)
% flow model: (i, o) or (i, i)
% L – input list
% R – resulting list

removeMain(L, R):-remove(L,L,R).
```

- There is another way of solving this problem (and of solving almost every problem), using an accumlator (collector variable), which is an extra parameter which represents the result of the function.
- Let's rewrite the nrOccurrences using a collector variable:

$$nrOccurrencesC(l_1 l_2 \dots l_n, e, res) = \begin{cases} res, if\ n = 0 \\ nrOccurencesC(l_2 l_3 \dots l_n, e, res + 1), if\ l_1 = e \\ nrOccurencesC(l_2 l_3 \dots l_n, e, res), otherwise \end{cases}$$

- In the implementation of nrOccurrencesC, will the collector variable be an input or an output parameter?
- We need it to start from the value 0. Since it is a parameter that we initialize with a specific value, it is an input parameter.

```
% nrOccurrencesC(L:list, E:el, Col: integer, S:integer)
% flow model: (i, i, i, o) or (i, i, i, i)
% L – list in which we count the occurrences
% E – the element we are looking for
%Col – collector variable
% S – the result, the number of occurrences

nrOccurrencesC([], _, Col, Col).
nrOccurrencesC([H|T], E, Col, S):-
	H = E,
	Col1 is Col + 1,
	nrOccurrencesC(T, E, Col1, S).
nrOccurrencesC([H|T], E, Col, S):-
	H \= E,
```

```
nrOccurrencesC(T, E, Col, S).
```

- We can use a collector variable to rewrite our remove predicate as well. There is one problem with this version, however: our collector variable is going to be a list, in which we will add elements one by one, after checking whether the element has to be added or not (depending on the number of occurrences). If we put the elements to the beginning of the list (where we can easily add elements), our result will be reversed. In order to have the elements in the right order, we need to put every element to the end of the collector variable, but for this we need another function: *addToEnd.*
- Let's assume that we already have this predicate, with the following header and flow model:
  - addToEnd(List, Elem, Result), (i, i, o)

- For counting the occurrences we can use nrOccurrences or nrOccurrencesC. We will use nrOccurrencesC
-

$$removeC(l_1 l_2 \dots L_n, L_1, L_2 \dots L_m, C_1 C_2 \dots C_k) = \begin{cases} C_1 C_2 \dots C_k, if\ n = 0 \\ removeC2(l_2 \dots l_n, L_1 L_2 \dots L_m, C_1 C_2 \dots C_k), if\ \text{nrOccurrencesC}(L_1 L_2 \dots L_m, l1, 0) = 1 \\ removeC2\big(l_2 \dots l_n, L_1 L_2 \dots L_m, addToEnd(C_1 C_2 \dots C_k, l1)\big), otherwise \end{cases}$$

```
% removeC(L:list, LO:list, Col:list, R:list)
% flow model: (i,i,i,o) or (i,i,i,i)
% L – list from which we remove elements that occur only once
% LO – copy of the initial list, to count occurrences
% Col – collector variable
% R – resulting list

removeC([], _, Col, Col).
removeC([H|T], LO, Col, R):-
      nrOccurrencesC(LO, H, 0, S),
      S = 1,
      removeC(T, LO, Col, R).
removeC([H|T], LO, Col, R):-
      nrOccurrencesC(LO, H, 0, S),
      S > 1,
      addToEnd(Col, H, Col1),
      removeC(T, LO, Col1, R).
```

- The LO list has to be initialized with the original list and the collector variable has to be initialized with the empty list. So we need another function.
-

$$removeCMain(l_1 l_2 \dots l_n) = removeC(l_1 l_2 \dots l_n, l_1 l_2 \dots l_n, \emptyset)$$

```
% removeCMain(L:list, R:list)
% flow model: (i,o), (i,i)
% L – list from which we remove elements that occur only once
% R – resulting list
```

```
eliminaCMain(L, R):-removeC(L, L, [], R).
```

2. Given a list of numbers, remove all the increasing sequences. Ex. remove([1,2,4,6,5,7,8,2,1]) => [2, 1]

- For this problem, it is not enough to just check if the first 2 elements are in increasing order and if they are so, remove them. If we do like this, in case of sequences with an odd number of elements, we will always be left with one element that is not removed because we do not know which element to compare it to.

$$removeInc(l_1 l_2 \dots l_n) = \begin{cases} \emptyset & ,n = 0 \\ l_1 & ,n = 1 \\ \emptyset & ,n = 2 \text{ şi } l_1 < l_2 \\ removeInc(l_2 \dots l_n) & , l_1 < l_2 < l_3 \\ removeInc(l_3 \dots l_n) & ,l_1 < l_2 \geq l_3 \\ l_1 \cup removeInc\ (l_2 \dots l_n) & ,otherwise \end{cases}$$

```
% removeInc(L:list, R:list)
% flow model: (i,o) or (i,i)
% L – list from which we remove the increasing sequences
% R – resulting list

removeInc([], []).
removeInc([H], [H]).
removeInc([H1,H2], []):- H1 < H2.
removeInc([H1,H2,H3|T], R):-
      H1 < H2,
      H2 < H3,
      removeInc([H2,H3|T], R).
removeInc([H1,H2,H3|T], R):-
      H1 < H2,
      H2 >= H3,
      removeInc([H3|T], R).
removeInc([H1,H2|T], [H1|R]):-
      H1 >= H2,
      removeInc([H2|T], R).
```