Seminar 4: Backtracking in Prolog

1. We are given a sequence a1...an of distinct integer numbers. We have to generate all the subsequences which have a valley aspect. For example, for the list [5,3,4, 2, 7, 11, 1,8 , 6] some solutions would be: [5,2, 7], [11,5, 3, 1, 2,4,6], etc. (there are 8828 solutions 986328).

* We will have a candidate list (collector variable) in which we will build the solution element by element.
* We will have a parameter, called Direction, which tells us which part of the valley we are building
  + 0 – for the decreasing part
  + 1 – for the increasing part
* We will build the solution in the reverse
  + [7]
  + [6, 7]
  + [4,6,7]
  + [5, 4, 6, 7]
  + [8, 5, 4, 6, 7]
* We have 4 possible cases:
  + Direction = 1, and we have an element to be added which is less than the first element of the collector variable (for ex: add 5 to the list [ 7, 9]) => [5, 7, 9], Direction is 1.
  + Direction = 1, and we have an element to be added which is greater than the first element of the collector variable (for ex: add 8 to the list [7,9]) => [8, 7, 9], Direction becomes 0
  + Direction = 0, and we have element to be added which is less than the first element of the collector variable (for ex: 5 to the list [8, 7, 9]) => not possible
  + Direction = 0 and we have element to be added which is greater than the first element of the collector variable ( for ex: 10 to the list [8, 7, 9]) => [10, 8, 7, 9], Direction is 0.
* When do we have a solution?
  + When direction is 0.
* We need a predicate which can give us one element of the list.

GetElement(l1l2...ln) = { l1,

GetElement(l2...ln) }

% flow model (I, o), (I, I)

% getElement(list, number)

GetElement([H|\_], H).

GetElement([\_|T], E):-

GetElement(T, E).

GetElement([1,2,3], E).

E = 1;

E = 2;

E = 3;

GetElement2([1,2,3], E, R).

E = 1, R = [2,3];

E = 2, R = [1,3];

E = 3, R = [1,2];

GetElement2(l1...ln) = { (l1, l2...ln)

(e, l1 U list), (e, list) = getElement2(l2...ln)

%getElement2(list,element,list)

%flow model:(I,o,o)

GetElement2([H|T],H,T).

GetElement2([H|T],E,[H|R]):-

GetElement2(T,E,R).

We will use getElement2.

The predicate that will generate the solutions:

* Input list
* Direction
* Collector list
* Result (only in Prolog)

Solution(l1l2..ln,d,c1c2..cn)={ Solution(list,d,ec1c2..cn), (e,list)=getElement2(l1l2...ln), e<c1, d=1

Solution(list,0,ec1...cn),(e,list)=getElement2(l1...ln),e>c1,d=1

Solution(list,d,ec1...cn),(e,list)=getElement2(l1...ln),e>c1,d=0

c,d=0}

%Solution(list, number, list, list)

%flow model: (I, I, I, o)

Solution(\_,0,C,C).

Solution(L, D, [H|T], Res) :-

D =:= 1,

GetElement2(L, E, L1),

E < H,

Solution(L1, D, [E,H|T], Res).

Solution(L, D, [H|T], Res) :-

D =:= 1,

GetElement2(L, E, L1),

E > H,

Solution(L1, 0, [E,H|T], Res).

Solution(L,D,[H|T], Res):-

D =:= 0,

GetElement2(L, E, L1),

E > H,

Solution(L1, D, [E,H|T], Res).

?-solution([1,2,3,4,5,6], 1, [], R).

False.

Wrapper(l1...ln) = solution(list2, 1, [e1, e2]), (e1, list) = getElement2(l1..ln), (e2, list2) = getElement2(list), e1 < e2

[3, 9] 1

[4,3, 9] 0

[5,4,3, 9] 0

%Wrapper(List,List)

%flow mode: (I,o)

Wrapper(List,Output):-

GetElement2(List,Element1,CollectorList),

GetElement2(CollectorList,Elemet2,CollectorList2),

Element1 < Element2,

Solution(CollectorList2,1,[Element1,Element2]).

WrapperMain(l1...ln) = U wrapper(l1...ln)

%WrapperMain(List, List)

%flow model(I,o)

WrapperMain(L, Rez):- findall(Output , wrapper(L, Output) , Rez).

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

What if we want to generate only those solutions in which the elements are in the same relative order as in the initial list?

[1,4,2,5,3, 6]

* [4,2,6] is a good solution
* [6, 2, 4] is not a good solution

[1,4,2,5,3,6] => if returns element 2, [1,4]

GetElement3 – to return an element and the list until that element

GetElement3(l1...ln) = { (l1, [])

(e, l1 U list), (e, list) = getElement3(l2...ln)