Seminar 3: Heterogeneous lists in Prolog

* A list in which the elements have different types, ex: [1,2, 3, t, s, v, [1,2,3, y,t], 5, 3, [1,2,3, [7, 3, 6], 6 ], y].
* How to process such a list?
  + [H|T]
    - [1,2,3,4] => [H|T] => H = 1, T = [2,3,4]
    - [[1,2,3], 4,5,6] =>[H|T] => H= [1,2,3], T = [4,5,6]
  + is\_list(H) - checks if H is a list
  + number(H) - checks if H is a number
  + atom(H) - checks if H is a symbol
* How to build such a list?
  + [H|T]

|  |  |  |
| --- | --- | --- |
| [H|T] | T = 3 | T = [2,3,4] |
| H = 2 | [2 | 3] | [2, 2, 3, 4] |
| H = [1,2,3] | [[1,2,3] | 3] | [[1,2,3], 2, 3, 4] |

Obs: [1,2,3,4,5,6,7,8 | … ]

?-call([1,2,3,4,5,6,7,8,9], R).

R = [1,2,3,4,5,6,7,8| … ] w

R = [1,2,3,4,5,6,7,8,9,10]

?-call([1,2,3,4,5,6,7,8,9], R).

R = [1,2,3,4,5,6,7,8| … ]

V1

?-call([1,2,3,4,5,6,7,8,9], R); true.

R = [1,2,3,4,5,6,7,8| … ] w

R = [1,2,3,4,5,6,7,8,9, 10]

V2:

?-call([1,2,3,4,5,6,7,8,9], R), write(R).

[1,2,3,4,5,6,7,8,9,10]

R = [1,2,3,4,5,6,7,8| … ]

V3: setting a flag

set\_prolog\_flag(answer\_write\_options,[max\_depth(0)]).

1. You are given a heterogeneous list, made of numbers and lists of numbers. You will have to remove the odd numbers from the sublists that have a mountain aspect. A list has a mountain aspect if it is made of a sequence of increasing numbers, followed by a sequence of decreasing numbers. Ex:

[1,2, [1,2,3,2], 6, [1,2], [1,4,5,6,7,1], 8, 2, [4,3,1], 11, 5, [6,7,6], 8 ] = >

[1, 2, [2, 2], 6, [1, 2], [4, 6], 8, 2, [4, 3, 1], 11, 5, [6, 6], 8].

How can we solve the problem?

* Check if a list is a mountain
* Remove the odd numbers from a linear list
* Process the heterogeneous list and remove odd numbers from every list with mountain aspect.

How do we check if a list is a mountain?

**V1:**

* + Have a flag that increases every time the sense changes. ( start with 0, change to 1 if l1 < l2, and when l1 > l2 change it to 2).

V2:

* + Two predicates
    - One for ascending part (removes the ascending part and when it is over it calls the second)
    - Checks if the entire (Remaining) list is decreasing

V3:

* + Look for the peak of the mountain (l1 < l2 > l3)
  + [1,2,3,4,3,2,1,2,3,4,3,2,1]
  + [1,2,3,4,3,2,1,2,3,4] - make sure you do not have l1 > l2 < l3

Flag will be initially 0, 1 for increasing, and 2 for decreasing

Mountain(l1l2...ln, flag) = mountain(l2...ln, 1), l1 < l2 and flag = 0

False, l1 > l2 and flag = 0

Mountain(l2...ln, 1), l1 < l2 and flag=1

Mountain(l2..ln, 2) , l1 > l2 and flag=1

False, l1 < l2 and flag = 2

True, n=1 and flag =2

Mountain(l2..ln, 2), l1 > l2 and flag = 2

False, n=1 flag!=2

Simplified version

Mountain(l1l2...ln, flag) = mountain(l2...ln, 1), l1 < l2 and flag < 2

Mountain(l2..ln, 2) , l1 > l2 and flag>0

True, n=1 and flag =2

False, otherwise

Mountain([4,3,2,1], 0)

Mountain([3,2,1], 2)

Mountain([2,1], 2)

Mountain([1], 2)

True

%m(L: List, F: Integer)

%flowModel(i, i)

m([\_], 2).

m([H1, H2 | T], F):-

F < 2,

H1 < H2,

!,

m([H2|T], 1).

m([H1, H2 | T], F):-

F > 0,

H1 > H2,

!,

m([H2|T], 2).

mountain(L):-m(L, 0).

* Remove the odd elements from a linear list
* RemoveOdd(l1...ln) =
  + [], n=0
  + RemoveOdd(l2...ln), l1 is odd
  + L1 U removeOdd(l2...ln), l1 is even

%removeOdd(L:List, R:List)

%FlowModel(I, o)

removeOdd([], []).

removeOdd([H|T], [H|Rez]) :- (mod(H, 2)=:=0), removeOdd(T, Rez).

removeOdd([H|T], Rez) :- (mod(H, 2)=\=0), removeOdd(T, Rez).

%removeOdd([], []).

%removeOdd([H|T], [H|Rez]) :- (mod(H, 2)=:=0), !, removeOdd(T, Rez).

%removeOdd([H|T], Rez) :- removeOdd(T, Rez).

* Removes the odd numbers from the sublists with a mountain aspect.

RemoveOddFromSublist(l1, l2, …, ln) = {

[], if n = 0

removeOdd(l1) U RemoveOddFromSublist(l2, …, ln), if number(l1) = false and mountain(l1) = true

L1 U RemoveOddFromSublist(l2, .., ln), otherwise

%remove(L: list, Rez : list)

%flow model: (i, o)

Remove([],[]).

%Remove([H|T],Rez):- is\_list(H), mountain(H), removeOdd(H, H1), remove(T, [H1|Rez]).

% in the recursive call remove(T, [H1|Rez])

% result from recursive call = [H1| Rez]

% has a value = has a value | has no value

Remove([H|T],[H1|Rez]):- is\_list(H), mountain(H), removeOdd(H, H1), remove(T, Rez).

% output param = [H1|Rez]

% has no value = has no value | has no value

Remove([H|T], Rez):-number(H), remove(T, [H|Rez]).

Remove([],[]).

Remove([H|T],Rez):- is\_list(H), mountain(H), removeOdd(H, H1), remove(T, Result), Result = [H1 | Rez].

%Remove([H|T],Result):- is\_list(H), mountain(H), removeOdd(H, H1), remove(T, Rez), Result = [H1 | Rez]

Remove([H|T], Rez):-number(H), remove(T, [H|Rez]).

Remove([],[]).

Remove([H|T],Result):- is\_list(H), mountain(H), !, removeOdd(H, H1), remove(T, Rez), Result = [H1 | Rez].

Remove([H|T], Rez):- remove(T, [H|Rez]).