

Practical Session E3

More Exercises on Prolog *Prolog (2)*

a.a. 2011/2012

Stefano Benedettini

Part 1

- The CUT predicate
 - examples of controlling a search predicate

Case 1.1: lookAny

```
% lookAny(?Elem,?List,?Position)

lookAny(X,[X|_],0).
lookAny(X,[_|Xs],s(N)):-lookAny(X,Xs,N).
```

- **Analise this predicate and the followings**
- It looks for the position of any element
 - lookAny(10,[10,20,10,30,10],X)
 - X/0
 - X/s(s(0))
 - X/s(s(s(s(0))))

Case 1.2: lookFirst

```
% lookFirst(?Elem,?List,?Position)
```

```
lookFirst(X,[X|_],0).
```

```
lookFirst(X,[X1|Xs],s(N)):-X=\=X1, lookFirst(X,Xs,N).
```

- It looks for the position of the first element
 - lookFirst(10,[10,20,10,30,10],X)
 - X/0
 - no

Case 1.3: lookAnyLast

```
% lookAnyLast(?Elem,?List,?Position)
```

```
lookAnyLast(X,[_|Xs],s(N)):-lookAnyLast(X,Xs,N).  
lookAnyLast(X,[X|_],0).
```

- It looks for the position of any element, starting from the end
 - lookAnyLast(10,[10,20,10,30,10],X)
 - X/s(s(s(s(0))))
 - X/s(s(0))
 - X/0
- Why? What is the difference with lookAny?
 - **try to construct the resolution tree!**

Case 1.4: lookFirstC (w/ Cut)

```
% lookFirstC(+Elem,+List,-Position)
```

```
lookFirstC(X,[X|_],0):-!.
```

```
lookFirstC(X,[_|Xs],s(N)):- lookFirstC(X,Xs,N).
```

- It looks for the position of the first element only
 - lookFirstC(10,[10,20,10,30,10],X)
 - X/0

Case 1.5: lookLastC (w/ Cut)

```
% lookLastC(+Elem,+List,-Position)
```

```
lookLastC(X,[_|Xs],s(N)):- lookLastC(X,Xs,N),!.  
lookLastC(X,[X|_],0).
```

- It looks for the position of the last element
 - lookLastC(10,[10,20,10,30,10],X)
 - X/s(s(s(s(0))))

Case 1.5: lookAll

```
% lookAll(+Elem,+List,-ListOfPositions)
```

```
lookAll(X,L,Ps):- findall(P,lookAny(X,L,P),Ps).
```

- It returns the list of all positions of the element
 - lookAll(10,[10,20,10,30,10],L)
 - L/[0,s(s(0)),s(s(s(s(0))))]

Part 2

- The CUT predicate
 - application to a predicate for dropping elements

Case 2.1: dropAny

```
% dropAny(?Elem,?List,?OutList)
```

```
dropAny(X,[X|T],T).
```

```
dropAny(X,[H|Xs],[H|L]):-dropAny(X,Xs,L).
```

- Drops any occurrence of an element
 - dropAny(10,[10,20,10,30,10],L)
 - L/[20,10,30,10]
 - L/[10,20,30,10]
 - L/[10,20,10,30]

Ex2.2: other drops

- Try to realise some of the following
 - dropFirst: drops only the first occurrence
 - dropFirstC: dropFirst with cut
 - dropAnyLast: drops any occurrence, from the last
 - dropLastC: dropLast with cut
 - dropAll: drop all, returning a list

Part 3

- Some functions on lists

Ex3.1: replace

- Realise a function

- `replace(+List,+Elem,-NewElem,-OutList)`
- it looks for `Elem` and creates a new list with `NewElem` instead
- `replace([1,2,3,1],1,5,L). -> L/[5,2,3,1]`

- Realise another version

- working on all occurrences of `Elem` (`replaceAll`)
- `replaceAll([1,2,3,1],1,5,L). -> L/[5,2,3,5]`

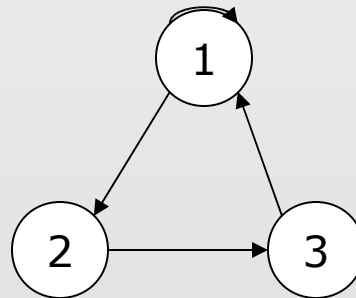
Ex3.2: pushNew

```
% pushNew(+List,+Elem,-OutList)
% Adds the element to the end of the list only if
% not already there
```

- Examples:
 - pushNew([10,20,30],15,[10,20,30,15])
 - pushNew([10,20,30],20,[10,20,30])
- **Implement it**; suggestion:
 - if the list is void, add the element
 - if the head of the list is the element, return the list (and cut)
 - otherwise: propagate pushNew to the tail

Part 4

- A directed graph data structure
- Our representation
 - Graph as an edge list
 - List of pairs of vertices
 - $[e(1,1), e(1,2), e(2,3), e(3,1)]$
 - The order of elements is not relevant



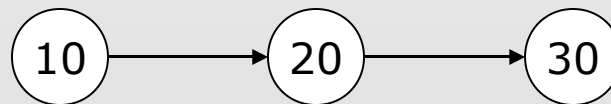
Ex4.1: fromList

```
% fromList(+List,-Graph)
```

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

```
fromList([H1,H2|T],[e(H1,H2)|L]):- fromList([H2|T],L).
```

- It obtain a graph from a list
 - fromList([10,20,30],[e(10,20),e(20,30)]).
 - fromList([10,20],[e(10,20)]).
 - fromList([10],[]).

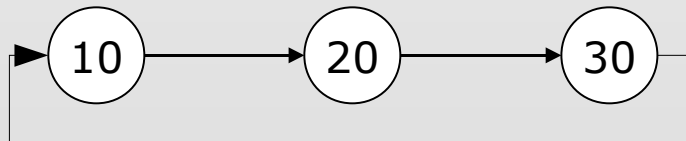


Ex4.2: asCycleGraph

```
% asCycleGraph(+List,-Graph)
```

```
% which implementation?
```

- **Implement it!**
- Obtain a cycle graph from a list
 - `asCycleGraph([10,20,30],[e(10,20),e(20,30),e(30,10)])`.
 - `asCycleGraph([10,20],[e(10,20),e(20,10)])`.
 - `asCycleGraph([10],[e(10,10)])`.



Ex4.3: dropVertex

```
% dropVertex(+Graph, +Vertex, -OutGraph)

% drop all edges starting and leaving from a Vertex
% use dropAll defined in 2.2

dropVertex(G,N,O):- dropAll(G,e(N,_),G2),
                    dropAll(G2,e(_,N),O).
```

- Analyse this predicate
- `dropVertex([e(1,2),e(1,3),e(2,3)],1,[e(2,3)])`.

Ex4.4: adjacent

```
% adjacent(+Graph, +Vertex, -List)
```

```
% all the vertices that can be reached in 1 step from Vertex  
% just use findall, looking for e(Vertex,_) combined with  
% member(?Elem,?List)
```

- **Implement it!**
- `adjacent([e(1,2),e(1,3),e(2,3)],1,L). -> L/[2,3]`
- `adjacent([e(1,2),e(1,2),e(2,3)],1,L). -> L/[2,2]).`

Ex4.5: anypath

```
% anypath(+Graph, +Vertex1, +Vertex2, -ListPath)
```

```
% a path from Vertex1 to Vertex2
```

- `anypath([e(1,2),e(1,3),e(2,3)],1,3,L).`
 - `L/[e(1,2),e(2,3)]`
 - `L/[e(1,3)]`
- **Implement it**; suggestion:
 - a path from N1 to N2 exists if there is an edge `e(N1,N2)`
 - a path from N1 to N2 is OK if N3 can be reached from N1, and then there is a path from N2 to N3, recursively

Ex4.6: allreachable

```
% allreachable(+Graph, +Vertex, -List)

% all the vertices that can be reached from Vertex
% Suppose the graph HAS NO CYCLES!
% Use findall and anypath!
```

- Implement it using the above suggestions
- `allreachable([e(1,2),e(2,3),e(3,5)],1,[2,3,5]).`

Find goals for the following

- Find one query for each problem:
 - all vertices that can be reached from N in 2 steps
 - all vertices that can be reached from N in 3 steps
 - all paths from N which are 2 steps long
 - all circular paths from N to N
 - ...

Ex5: remove couples of "0"

(Suggested Exercise)

```
% remove2zero(+List,-OutList)
% Removes each double occurrence of zero
```

- Implement it
- Tests:
 - `remove2zero([10,0,0,30,0,0,40],L).` -> `L/[10,30,40]`
 - `remove2zero([10,0,30,0,0,40],L).` -> `L/[10,0,30,40]`
 - `remove2zero([],L).` -> `L/[]`
 - `remove2zero([10,0,0,0,30],L).` -> `L/[10,0,0,0,30]`
 - `remove2zero([0,0],L).` -> `L/[]`