PPS2017 - lab11

Advanced Exercises in Prolog

a.a. 2017/2018

Prof. Mirko Viroli

Outline

- Other exercises in Prolog, mostly using cut
- Supporting a new data structure (graphs)
- An advanced exercise
  - generation of TicTacToe tables

Case 1: dropAny

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

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

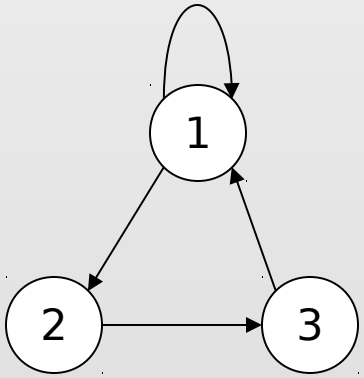
- Check the above code
- Drops any occurrence of element
  - dropAny(10,[10,20,10,30,10],L)
  - L/[20,10,30,10]
  - L/[10,20,30,10]
  - L/[10,20,10,30]

Ex1.1: other drops

- Try to realise some of the following variations, by using cut and/or reworking the implementation
  - dropFirst: drops only the first occurrence (showing no alternative results)
  - dropLast: drops only the last occurrence (showing no alternative results)
  - dropAll: drop all occurrences, returning a single list as result

Case 2

- A graph data structure
- Our model
  - as a list of couples [e(1,1),e(1,2),e(2,3),e(3,1)]
  - the order of elements in the list is not relevant

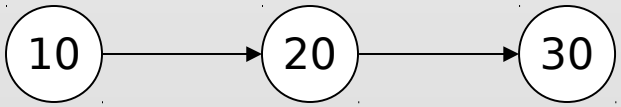


Ex2.1: fromList

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

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

- Just analyse the code
- It obtains a graph from a list
  - fromList([10,20,30],[e(10,20),e(20,30)]).
  - fromList([10,20],[e(10,20)]).
  - fromList([10],[]).

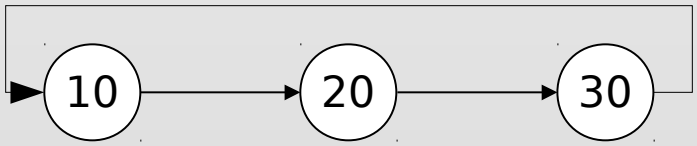


Ex2.2: fromCircList

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

% which implementation?
```

- Implement it!
- Obtain a graph from a circular list
  - fromCircList([10,20,30],[e(10,20),e(20,30),e(30,10)]).
  - fromCircList([10,20],[e(10,20),e(20,10)]).
  - fromCircList([10],[e(10,10)]).



Ex2.3: dropNode

```
% dropNode(+Graph, +Node, -OutGraph)

% drop all edges starting and leaving from a Node
% use dropAll defined in 1.1

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

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

Ex2.4: reaching

```
% reaching(+Graph, +Node, -List)

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

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

Ex2.5: anypath (advanced!!)

```
% anypath(+Graph, +Node1, +Node2, -ListPath)

% a path from Node1 to Node2
% if there are many path, they are showed 1-by-1
```

- 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 a 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

Ex2.6: allreaching

```
% allreaching(+Graph, +Node, -List)

% all the nodes that can be reached from Node
% Suppose the graph is NOT circular!
% Use findall and anyPath!
```

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

Case 3: Generating TicTacToes

- Implement predicate next/4 as follows
  - next(@Table,@Player,-Result,-NewTable)
  - Table is a representation of a TTT table where players x or o are playing
  - Player (either x or o) is the player to move
  - Result is either (either win(x), win(o), nothing, or even)
  - NewTable is the table after a valid move
  - Should find a representation for the Table
  - Calling the predicate should give all results
- Secondarily, implement predicate:
  - game(@Table,@Player,-Result,-TableList)
  - TableList is the sequence of tables until Result win(x), win(o) or even

Some hint

- Choosing the right representation for a table is key
  - with a good representation it is easier to select the next move, and to check if somebody won
  - if needed, prepare to separate representation from visualisation
- Possibilities
  - [[_,_,_],[x,o,x],[o,x,_]]: nice but advanced
  - [[n,n,n],[x,o,x],[o,x,n]]: compact, but need work
  - [cell(0,1,x),cell(1,1,o),cell(2,1,x),…]: easier
  - … do you have a different proposal?