Logic Programming – Laboratory 3

Recursion

Isabela Drămnesc

1 Questions

- What are anonymous variables? When do we use them?
- What do you understand by unification in Prolog? How does it work?
- What are data structures? How do we represent:

```
course(datap(fname(ardelean),lname(john)),
      datac(tuesday,coursen(lp),room(a11))).
library(university(ofWest),
              book(poems,author(mihai,eminescu))).
```

- What do we need to add into the database to find solutions for:
  1. Who has the lp (Logic Programming) course on tuesday in the room 102?
  2. What books do we find in the library from the university of cluj?
- How do we represent and how do we interogate Prolog:
  1. $3X^2 + (7N/Z^2)$.
  2. $X^{10} - 2X^4$

2 Concepts

- Trace
- Inductive domain
- Recursion
- Recursive procedures
- Boundary conditions
- Recursive call
- Recursion on lists
  - Recursive mapping
  - Recursive comparison
  - Joining structures

3 Trace

Introduce the following database:

```
visit(john, spain).
visit(mary, spain).
visit(alex, italy).
visit(alex, germany).

traveler(john).
traveler(alex).
traveler(victoria).

journey(X,Y):-traveler(X), visit(X,Y).
```

For each interogation follow all the steps and find all the possible solutions using the *trace* command:

```
?-journey(john,X).
?-journey(mary,X).
?-journey(N, spain).
?-journey(C, germany).
?-journey(X,Y).
```

4 Induction/Recursion

4.1 Inductive domain

Is a domain composed of objects which can be decomposed into a finite number of simpler objects. The process continues until one reaches the "simplest" objects.

Example of an inductive domain:

```
[]      /*-the empty list; the simplest object */
[H|T]   /* -the generic list, where H=the head,
            T=the tail of the list.
            The head of the list is one single element,
            the tail of the list has to be a list. */
```

4.2 Recursion

A recursive procedure has to describe the behavior for:

- The "simplest" objects (this is the situation for which the computation stops), i.e. the boundary condition.
- The general case, which describes the recursive call.

Example: The predicate that defines a list:

```
1)
is_list([]).  /* the boundary condition */
```

```
is_list([H|T]):- is_list(T).      /* recursive call */

/*How many solutions we obtain for the interogation: */
?- is_list(A).
```

There exists in SWI-Prolog a predicate which is already defined for defining the lists: *is_list*. See *help*(*is_list*).

Attention: The order of declaring the clauses is very important:

Be careful with cases like :

```
2)
is_list([H|T]):- is_list(T).
is_list([]).

?- is_list(X).

3)
parent(X,Y):- child(Y,X).
child(X,Y):- parent(Y,X).

4)
person(X):- person(Y), father(Y,X).
person(john).
father(gabriel,john).

?- person(X).
```

4.3 Exercises:

5) For the exercise "Write a predicate *swapfirst*2/2 (i.e. binary) that accepts a list and generates from it a similar list with the first two elements swapped" ask Prolog to:

```
?- swapfirst2([1,2,3,4],X).
?- swapfirst2(X,Y), swapfirst2(Y,X).
```

What do you notice? Use the *trace* command to see what is happening on the last interogation.

6) A predicate to find out if an element belongs to a list. Example:

```
member(X,[X|_]).
member(X,[_|Y]):- member(X,Y).

?- member(3,[1,2,3]).
true
```

See also *help*(*member*).

7) Example: a predicate to eliminate an element from a list.

```
eliminate(X,[X|T],T).
eliminate(X,[Y|T1],[Y|T2]):- eliminate(X,T1,T2).

?- eliminate(2,[1,2,3,4],X).
```

```
X=[1,3,4];
false
```

Test in Prolog all the posibilities!

8) Write a predicate which calculates $n!$

We know $0! = 1$ and $n! = 1 \cdot 2 \cdot ... \cdot (n-1) \cdot n$.

```
?- factorial(5,X).
X = 120 .

?- factorial(10,F).
F = 3628800 .
```

9) Write a predicate which calculates the greatest common divisor of two numbers. Use the Euclid's recursive definition.

Let a and b be two natural nonzero numbers. If b=0, then gcd(a,b)=a; otherwise gcd(a,b)=gcd(b,r), where r is the remainder of the division of a to b.

```
?- gcd(25,100,C).
C = 25 ;
false.

?- gcd(27,99,K).
K = 9 ;
false.
```

5 Recursive mapping

Mapping: given 2 similar data structures change the first one into the second one and follow some given rules.

Example: "you are a computer" maps to "i am not a computer", "do you speak french" maps to "i do not speak german".

```
10)
change(you,i).
change(are,[am,not]).
change(french,german).
change(do,no).
change(X,X).

alter([],[]).
alter([H|T],[X|Y]):-change(H,X),alter(T,Y).

?- alter([you,are,a,computer],W).
?- alter([i,do,like,you],W).
```

6 Joining the data structures (Append)

11) Given A and B, two lists, write a predicate appendLists/3 which returns a list containing the elements from the list A followed by the elements from the list B. Example: for A=[a,b,c], B=[7,8,9], the resulting list will be C=[a,b,c,7,8,9].

Test for the following interogations:

```
 ?- appendLists([1,2,3],[s,d,3,4],X).
X=[1,2,3,s,d,3,4].

 ?- appendLists(X,[1,2,3],[1,2,s,d,3,4]).
false.

 ?- appendLists(X,[1,2,3],[d,f,g,ssss,1,2,3]).
X=[d,f,g,ssss].

 ?- appendLists(X,Y,[d,f,g,ssss,1,2,3]).
X = [],
Y = [d, f, g, ssss, 1, 2, 3] ;
X = [d],
Y = [f, g, ssss, 1, 2, 3] ;
X = [d, f],
Y = [g, ssss, 1, 2, 3] ;
X = [d, f, g],
Y = [ssss, 1, 2, 3] ;
X = [d, f, g, ssss],
Y = [1, 2, 3] ;
X = [d, f, g, ssss, 1],
Y = [2, 3] ;
X = [d, f, g, ssss, 1, 2],
Y = [3] ;
X = [d, f, g, ssss, 1, 2, 3],
Y = [] ;
false.
```

7 Homework:

Homework 3. Deadline: next lab.