CSE 307 - Constraint Logic Programming

https://moodle.polytechnique.fr/course/view.php?id=12795

TP2 Symbolic Differentiation and Lists in Prolog

François Fages (Francois.Fages@inria.fr)

Bachelor of Science - Ecole polytechnique

1. Symbolic differentiation

Let us consider the symbolic representation of mathematical expression by Prolog closed terms (i.e. containing no Prolog variable) in which mathematical variables are thus represented by Prolog constants.

The Prolog variables can be used to do pattern matching on such mathematical expressions:

```
?- 2*x+3*x*y=A*B.
false.

?- 2*x+3*x*y=A+B.
A = 2*x,
B = 3*x*y.
```

The purpose of the following questions is to let you write a symbolic differentiation predicate, allowing you to compute symbolic partial derivates with respect to a mathematical variable, first without performing any simplification of the result:

For these questions, you might find useful to use the conditional expression of Prolog in addition to its pattern matching mechanism:

```
(Condition -> ThenGoal ; ElseGoal).
(Condition1 -> ThenGoal1 ; Condition2 -> ThenGoal2 ; ElseGoal).
```

Write your answers in Prolog file `tp2.pl`

Question 1. Defined in Prolog the following predicate to differentiate an expression with respect to a variable, without performing simplifications, as follows

```
?- differentiate(2*x+3*x*y, x, D).
D = 0*x+1*2+((0*x+1*3)*y+0*(3*x)) ;
false.

?- differentiate_aux(2*x+3*x*y, y, D).
D = 0*x+0*2+((0*x+0*3)*y+1*(3*x)) ;
false.
```

Question 2. Then write a simplify/2 predicate for usual algebraic simplifications:

```
?- differentiate(2*x+3*x*y, x, D), simplify(D, E).
D = 0*x+1*2+((0*x+1*3)*y+0*(3*x)),
E = 2+3*y

?- differentiate(2*x+3*x*y, y, D), simplify(D, E).
D = 0*x+1*2+((0*x+1*3)*y+0*(3*x)),
E = 3*x
```

2. List processing

Lists in Prolog are formed with

- the constant [ ] for the empty list
- and the list contructor [l] for (des)assembling the head and the tail of a list

```
?- [a,b,c]=[X|Y].
X = a,
Y = [b, c].

?- [a,b,c]=[X,Y|Z].
X = a,
Y = b,
Z = [c].

?- [a,b,c]=[X,Y,Z].
X = a,
Y = b,
Z = c.

?- [a,b,c]=[X,Y,Z|L].
X = a,
Y = b,
Z = c,
L = [].

?- [a]=[X|Y].
X = a,
Y = [].
```

The predicate `member(X, L)` true if X is a member of list L can be defined by

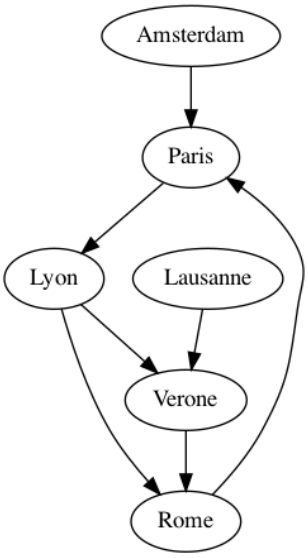
```
member(X, [X | _]).
member(X, [_ | L]) :- member(X, L).

?- member(X,L).
L = [X|_24160] ;
L = [_24158, X|_24890] ;
L = [_24158, _24888, X|_25620] .

?- member(a, [b,c,d]).
false.

?- \+ member(a, [b,c,d]).
true.
```

In the first practical session (TP1) you were asked to represent a graph (a route map) with the predicate `arc/2` and to define its transitive closure `path/2` when the graph contains no circuit. Now, using a list as third argument for predicate `path/3`, you can memorize the list of visited cities during search, and write a program to find pathways in such cyclic graphs without looping.



Question 3. Define in file `tp2.pl` this cyclic graph with predicate `arc/2` and define a non-looping predicate `path(X, Y, L)` to go from X to Y where L is a list (e.g. empty) of forbidden already visited cities.

3. List and Arithmetic

Now, using arithmetic constraints (next course) rather than evaluable predicates, one can define the length of a list as follows

```
:- use_module(library(clpfd)).

listlength([], 0).

listlength([X|L], N) :-
  N#>1,
  N1#=N-1,
  listlength(L, N1).

?- listlength([a,b,c], N).
N = 3.

?- listlength(L, 3).
L = [_3446, _3458, _3470] ;
false.

?- listlength(L, N).
L = [],
N = 0 ;
L = [_9994],
N = 1 ;
L = [_9994, _11696],
N = 2 ;
L = [_9994, _11696, _13878],
N = 3 .
```

Question 4. Write in file `tp2.pl` the predicate `enumerate(Min, Max, L)` that is true if L is the list of integers from Min to Max

```
?- enumerate(-2,3,L).
L = [-2, -1, 0, 1, 2, 3] ;
false.

?- enumerate(-2,M,L).
L = [],
M in inf.. -3 ;
M = -2,
L = [-2] ;
M = -1,
L = [-2, -1] ;
M = 0,
L = [-2, -1, 0] ;
M = 1,
L = [-2, -1, 0, 1] .

?- enumerate(M,3,L).
L = [],
M in 4..sup ;
M = 3,
L = [3] ;
M = 2,
L = [2, 3] ;
M = 1,
L = [1, 2, 3] ;
M = 0,
L = [0, 1, 2, 3]
```

4. GPS Map Navigation

Let us suppose that the users of your GPS map navigation system sends their position regularly, and that by averaging those data, you can equip the arcs of your map with the average travel time (in hours) it is currently taking.

Question 5. Label the arcs of the graph with such travel times using a ternary `arc/3` predicate, and add the total expected time it will take, as fourth argument to `path(X, Y, L, Time)`.

```
?- path(paris, rome, [], T).
T = 15 ;
T = 17 ;
false.
```

5. Hanoi Towers puzzle

The file tp2.pl contains the program presented in the course to solve the Hanoi Tower puzzle with the predicate `hanoi(N,X,Y,Z)` that moves N disks from X to Z using Y, assuming that the N top disks of X are of smaller than the disks on Y and Z.

```
hanoi(N,X,_,Z) :-
  N #= 1,
  format('Move top disk from ~w to ~w~n', [X,Z]).

hanoi(N,X,Y,Z) :-
  N #> 1,
  N1 #= N-1,
  hanoi(N1,X,Z,Y),
  format('Move top disk from ~w to ~w~n', [X,Z]),
  hanoi(N1,Y,X,Z).

?- hanoi(3,left,middle,right).
Move top disk from left to right
Move top disk from left to middle
Move top disk from right to middle
Move top disk from left to right
Move top disk from middle to left
Move top disk from middle to right
Move top disk from left to right
true ;
false.
```

Question 6. Modify that program with lists to visualize the three pegs over time as follows (the predicate `sort/2` for sorting a listing will be useful to write the left, middle, right pegs in order):

```
?- hanoiviz(3,[left,1,2,3],[middle],[right],L,M,R).

 [left,2,3]
 [middle]
 [right,1]

 [left,3]
 [middle,2]
 [right,1]

 [left,3]
 [middle,1,2]
 [right]

 [left]
 [middle,1,2]
 [right,3]

 [left,1]
 [middle,2]
 [right,3]

 [left,1]
 [middle]
 [right,2,3]

 [left]
 [middle]
 [right,1,2,3]
L = [left],
M = [middle],
R = [right, 1, 2, 3] ;
false.
```

Finally, don't forget to upload your file on the Moodle !
https://moodle.polytechnique.fr/course/view.php?id=12795