CSE 307 - Relational Programming

TP2 Symbolic Differentiation and List Processing in Prolog

François Fages (Francois.Fages@inria.fr)

Bachelor of Science - Ecole polytechnique

1. Symbolic differentiation

Let us consider the symbolic representation of mathematical expressions by Prolog closed terms (i.e. terms containing no Prolog variable).
The mathematical variables will be thus represented by Prolog constants. The Prolog variables will then be used to do pattern matching on such mathematical expressions, e.g.

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

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

Although not necessary, you might find useful to use

- built-in predicates for testing constants `atomic/1` and equality `==/2 \=/2 =../2`
- the conditional expression of Prolog in addition to its pattern matching mechanism:
  ```
  (Condition -> ThenGoal ; ElseGoal).
  (Condition1 -> ThenGoal1 ; Condition2 -> ThenGoal2 ; ElseGoal).
  ```

Insert your answers in Prolog file `tp2.pl` and upload it on the Moodle at the end of the session.

Question 1. Define in Prolog the following predicate to differentiate a polynomial with positive coefficients 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)).

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

Question 2. Define a `simplify/2` predicate for performing simple 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 with reversible predicates

Lists in Prolog are formed with

- the constant [ ] for the empty list
- and the list contructor [I] 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,b,c]=[X,Y,Z,U].
false.
```

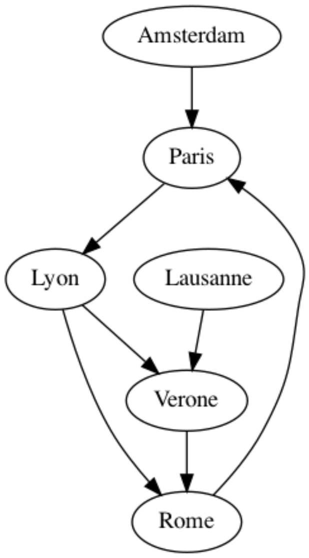
The predicate `member(X, L)` is true if X is a member of list L, it can be defined by

```
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]). % \+ is negation by failure
                          % \+ Goal is true if Goal finitely fails
                          % \+ Goal is false if Goal has a success
true.
```

Question 3. Define the relation remove(L1, X, L2) true if and only if L2 is the list L1 with **at most** one occurrence of X removed, e.g.

```
?- remove([a,b,a,c,d],a,L).
L = [b, a, c, d] ;
L = [a, b, c, d] ;
L = [a, b, a, c, d].

?- remove(L,a,[b,a,c,d]).
L = [a, b, a, c, d] ;
L = [b, a, a, c, d] ;
L = [b, a, a, c, d] ;
L = [b, a, c, a, d] ;
L = [b, a, c, d] ;
L = [b, a, c, d, a] ;
false.
```

In the first practical session (TP1) you were asked to represent a graph (a bus 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 4. Define in file `tp2.pl` that cyclic graph in extension with predicate `arc/2` and give a recursive definition of a non-looping predicate `path(X, Y, L)` true if and only if there is a path from X to Y and L is the list of cities explored during search. That list `L` will be empty in the initial query, e.g. `path(lyon, paris, []).`

Question 5. Prove that your program `path(X, Y, L)` always terminates even on cyclic graphs. *Hint: find a complexity measure on the arguments of `path(X, Y, L)` and the graph `arc/2` and show that this complexity measures strictly decreases at each recursive call.*

Finally, don't forget to upload your file `tp2.pl` on the Moodle !
https://moodle.polytechnique.fr/course/view.php?id=12795