Programmation fonctionnelle et logique

TD10 : révisions !

Caml

1. Si l'expression suivante est bien formée, donnez son type :

```
exception Erreur of int;;
let f a b c =
  let rec g aa bb cc =
     let d = if aa > 0 then cc
        in if aa > bb then (raise (Erreur bb))
                      else g (aa+1) bb cc
           in try g a b c with Erreur t -> t;;

val f : int -> int -> unit -> int = <fun>
```

   Explications :

   - `aa > bb` donc `aa` et `bb` sont de même type.
   - `aa > 0` donc `aa` est de type entier, tout comme `bb`, `a` et `b`.
   - `if aa > 0 then cc` : pas de `else`, donc `cc` est de type `unit`, et donc `c` est de type `unit`.
   - `Erreur t` : `t` est de type entier, donc `try g a b c with Erreur t -> t` renvoie un entier.
   - On vérifie ensuite qu'il n'y a pas d'incohérences.

2. Écrire une fonctionnelle qui réalise la composition de deux fontions.

```
let compose f g = function x -> f(g(x));;
val compose : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b = <fun>
```

3. Écrire une fonction qui calcule la somme des valeurs d'une fonction `f` sur les N premiers entiers. (Ne pas utiliser de boucle `for`.)

```
let rec somme f add neutre = function
   0 -> neutre
 | n -> add (somme f add neutre (n-1)) (f n);;
val somme : (int -> 'a) -> ('b -> 'a -> 'b) -> 'b -> int -> 'b = <fun>
```

4. Écrire, au moyen d'un itérateur, une fonction qui calcule le produit des éléments d'une liste d'entiers et qui renvoie 0 **dès** qu'un « 0 » est trouvé dans la liste.

```
exception Zéro;;
let produit l =
let mul x y = match (x,y) with   (0,_) -> raise Zéro
                               | (_,0) -> raise Zéro
                               | (x,y) -> x * y
in try  (List.fold_left mul 1 l) with Zéro -> 0;;
val produit : int list -> int = <fun>
```

5. Écrire, au moyen d'un itérateur, une fonction qui prend en entrée une liste de caractères et qui construit la chaîne de caractères correspondante. On suppose que l'on a à notre disposition une fonction `cons` qui ajoute un caractère en début d'une chaîne.

```
let cons c s = (String.make 1 c)^s;;
val cons : char -> string -> string = <fun>
let concat l =
List.fold_right cons l "";;
val concat : char list -> string = <fun>
```

Prolog

1. **N-reines.** Le problème est de placer N reines sur un échiquier de telle sorte qu'il n'y ait pas deux reines sur une même ligne, colonne ou diagonale. Écrire un prédicat qui permette de construire une liste de N places sur un échiquier carré de taille N.

```
deuxindep(X,Y,U,V) :- X =\= U, Y =\= V, X+Y =\= U+V, X-Y =\= U-V.

tousindep(_,_,[]).
tousindep(X,Y,[[U,V]|L]) :- deuxindep(X,Y,U,V), tousindep(X,Y,L).

indep(_,0,S,S).
indep(N,M,L,S) :- between(1,N,X), between(1,N,Y), tousindep(X,Y,L),
                  P is M-1, indep(N,P,[[X,Y]|L],S).

nreines(N,L) :- indep(N,N,[],L).
```

2. Même question mais en utilisant des négations par l'échec pour proscrire les mauvaises configurations.

```
deuxindep(X,_,X,_) :- !, fail.
deuxindep(_,Y,_,Y) :- !, fail.
deuxindep(X,Y,U,V) :- X+Y =:= U+V, !, fail.
deuxindep(X,Y,U,V) :- X-Y =:= U-V, !, fail.
```