Corrigé de l'examen de programmation fonctionnelle et logique

Jeudi 27 janvier 2000, 14h00-17h00

Remarques et commentaires :
- Commencez par lire le sujet dans son intégralité.
- Les exercices sont indépendants les uns des autres. Les questions au sein d'un même exercice sont elles aussi indépendantes. Vous pouvez très bien ne pas savoir répondre à une question et réussir les suivantes.
- Écrivez lisiblement et en français, car les copies seront lues (anonymat oblige) !
- Facilitez la lecture et la compréhension des codes proposés.

Rappels

Les fonctionnelles sur liste prédéfinies (en `Ocaml`)

Les quatre fonctionnelles sur listes prédéfinies sont :

- ```
  #List.iter;;
   - : ('a -> unit) -> 'a list -> unit = <fun>
  ```
  `List.iter f [a1; ...; an]` est équivalent à `begin f a1; ...; f an; () end`.
- ```
  #List.map;;
   - : ('a -> 'b) -> 'a list -> 'b list = <fun>
  ```
  `List.map f [a1; ...; an]` est équivalent à `[f a1; ...; f an]`.
- ```
  #List.fold_left;;
   - : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a = <fun>
  ```
  `List.fold_left f a [b1; ...; bn]` est équivalent à `f ( ... (f (f a b1) b2) ... ) bn`.
- ```
  #List.fold_right;;
   - : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b = <fun>
  ```
  `List.fold_right f [a1; ...; an] b` est équivalent à `f a1 (f a2 (... (f an b) ...))`.

Concaténation de deux listes

```
#[1;2]@[3;4];;
 - : int list = [1; 2; 3; 4]
```

Exercice 1 : typage d'expressions `Ocaml`

Donnez le type des expressions suivantes.
Important : la justification du résultat trouvé comptera autant que le résultat lui-même !

1. ```
   #let rec f g h x = if x > 0 then h x else g (h x);;
   ```
   ```
   #f;;
    - : ('a -> 'a) -> (int -> 'a) -> int -> 'a = <fun>
   ```

2. ```
   #let f x y z =
        List.fold_right
           (function u -> function v -> if u=v then 2*(x u) else 3*(x u))
           y
           z;;
   ```

```
# f;;
 - : (int -> int) -> int list -> int -> int = <fun>
```

3.
```
# exception Truc of float;;

# let f x y =
    let rec f_aux z =
       match z with
          t::r -> if t < y then raise (Truc t) else (f_aux r)
        | _    -> failwith "Erreur"
    in try (f_aux x) with Truc n -> n;;

# f;;
 - : float list -> float -> float = <fun>
```

Exercice 2 : établissement de menus en **`Ocaml`**

Le but de ce problème est de construire un menu pour tous les repas de midi d'une semaine. Nous supposons disposer au départ d'une liste d'entrées, d'une liste de plats de résistance et d'une liste de desserts. Bien sûr, nous ne voulons pas que les menus prévus pour deux jours différents aient la même entrée, ou le même plat de résistance ou le même dessert.

Les fonctions à la résolution de cet exercice seront naturellement polymorphes. C'est pourquoi l'énoncé ne précise pas le type utilisé pour représenter les entrées, les plats de résistance te les desserts (les trois sortes d'objets sont supposés de même type). Pour les exemples figurant dans l'énoncé, des chaînes de caractères ont été utilisée pour faciliter la compréhension.

1. **Construction des *menus*.**

   (a) Nous avons tout d'abord besoin d'une fonction qui prend en entrée une liste `l` d'éléments et qui renvoie la liste de toutes les listes qui contiennent un seul élément, cet élément appartenant à `l` (pour plus de clarté, voir l'exemple !).

   i. Écrivez une version récursive de cette fonction.

   ```
   # let rec explose = function
         [] -> []
       | t::r -> [t]@(explose r);;
         val explose : 'a list -> 'a list = <fun>
   ```

   ii. Écrivez une version de cette fonction utilisant une des fonctionnelles prédéfinies.

   ```
   # let explose l =
        List.map (function x -> [x]) l;;
      val explose : 'a list -> 'a list list = <fun>
   ```

   Exemple d'utilisation :

   ```
   # explose [1;2;4;5;6];;
    - : int list list = [[1]; [2]; [4]; [5]; [6]]
   ```

   (b) Nous avons besoin d'une fonction qui prend en entrée un élément, `x`, et une liste de listes, `ll`, et qui rajoute `x` en tête de chacune des listes de `ll`.

   i. Écrivez une version récursive de cette fonction.

   ```
   # let rec étend_listes x = function
         [] -> []
       | t::r -> (x::t)::(étend_listes x r);;
         val étend_listes : 'a -> 'a list list -> 'a list list = <fun>
   ```

   ii. Écrivez une version de cette fonction utilisant une des fonctionnelles prédéfinies.

```
#let étend_listes x l =
    List.map (function y -> x::y) l;;
   val étend_listes : 'a -> 'a list list -> 'a list list = <fun>
```

Exemple d'utilisation :

```
#étend_listes 3 [[1;2];[4];[5;6]];;
 - : int list list = [[3; 1; 2]; [3; 4]; [3; 5; 6]]
```

(c) Nous avons besoin d'une fonction qui prend en entrée une liste, `a`, et une liste de listes, `b`, et qui construit la liste de toutes les listes obtenues en ajoutant un élément de `a` en tête d'une liste élément de `b`.

i. Écrivez une version récursive de cette fonction.

```
#let rec rallonge_listes a b = match a with
     [] -> []
   | t::r -> (étend_listes t b)@(rallonge_listes r b);;
     val rallonge_listes : 'a list -> 'a list list -> 'a list list = <fu
n>
```

ii. Écrivez une version de cette fonction utilisant une des fonctionnelles prédéfinies.

```
#let rec rallonge_listes a b =
    List.fold_right (function x -> function y -> (étend_listes x b)@y)
                  a
                  [];;
       val rallonge_listes : 'a list -> 'a list list -> 'a list list = <
fun>
```

Exemple :

```
#rallonge_listes [1;2] [[3];[4;5]];;
 - : int list list = [[1; 3]; [1; 4; 5]; [2; 3]; [2; 4; 5]]
```

(d) Écrivez une fonction qui prend en entrée trois listes, `a`, `b` et `c`, et qui construit la liste de toutes les listes `[x;y;z]` possibles à trois éléments telles que `x` est élément de `a`, `y` élément de `b` et `z` élément de `c`. Par la suite nous appellerons *menu* une liste à trois éléments.

```
#let combine_trois_listes a b c =
    rallonge_listes a (rallonge_listes b (explose c));;
   val combine_trois_listes : 'a list -> 'a list -> 'a list -> 'a list l
ist =
  <fun>
```

2. **Construction des *menus sur la semaine***. Nous voulons considérer l'ensemble des menus de midi sur une semaine (cinq jours ouvrables). Par la suite nous appellerons *menu sur la semaine* une liste quelconque contenant exactement cinq *menus*. Écrivez donc une fonction qui prend en entrée la liste de tous les *menus* possibles et qui construit la liste de tous les *menus sur la semaine* possibles.

```
#let semaine l =
    rallonge_listes l (rallonge_listes l (rallonge_listes l (rallonge_listes
 l (explose l))));;
     val semaine : 'a list -> 'a list list = <fun>
```

3. **Non redondance de deux *menus***. Écrivez une fonction qui prend en entrée les *menus* de deux repas et qui indique si ces deux *menus* sont ou non redondants : deux *menus* sont dits redondants s'ils contiennent la même entrée, ou le même plat de résistance ou le même dessert. La fonction demandée doit renvoyer `true` si les deux *menus* arguments ne sont pas redondants, et elle doit renvoyer une exception `Redondants` dans le cas contraire.

```
#exception Redondants;;
 exception Redondants
```

```
#let nonredondants m n =
    match (m,n) with
        ([a;b;c],[d;e;f]) -> if a=d or b=e or d=f
                              then raise Redondants
                              else true
       |        _          -> failwith "Mauvais arguments pour
 nonredondants";;
           val nonredondants : 'a list -> 'a list -> bool = <fun>
```

Exemple d'utilisation :

```
#nonredondants
  ["Gougères aux deux céleris";"Filets de bondelle à la neuchâtelloise";
   "Rozenn à l'orange sauce mentholée"]
  ["Potage à la citrouille de Saint-Jacques-de-Montcalm";"Côte de boeuf rôtie à la
   bouquetière";"Demi-fraise à la maltaise"];;
         - : bool = true
```

```
#nonredondants
  ["Soufflé de cervelle à la chanoinesse";"Aileron de dindonneau Sainte-Menehould";
   "Bordure de riz à la Montmorency"]
  ["Soufflé de cervelle à la chanoinesse";"Fricandeau de veau à l'oseille";
   "Croquets de Bar-sur-Aube"];;
         Uncaught exception: Redondants
```

4. **Non redondance d'un *menu* avec une liste de *menus***. Nous avons besoin d'une fonction qui prend en entrée un *menu* `m` et une liste `l` de *menus*, et qui vérifie que le *menu* `m` n'est redondant avec aucun des *menus* de la liste `l`. Cette fonction doit impérativement utiliser la fonction `nonredondants` définie à la question précédente (question 3).

   (a) Nous voulons ici que notre fonction, appelée `sansredondances_exc` déclenche une exception en cas de redondance, suivant le comportement défini par l'exemple d'utilisation.

      i. Écrivez une version récursive de cette fonction.

```
#let rec sansredondances_exc m = function
      []  -> true
    | t::r -> (nonredondants m t) & (sansredondances_exc m r);;
       val sansredondances_exc : 'a list -> 'a list list -> bool = <fun>
```

      ii. Écrivez une version de cette fonction utilisant une des fonctionnelles prédéfinies.

```
#let sansredondances_exc m l =
     List.fold_right
        (function x -> function y -> y & (nonredondants m x))
        l
        true;;
          val sansredondances_exc : 'a list -> 'a list list -> bool = <fu
n>
```

   Exemple d'utilisation :

```
#sansredondances_exc
  ["Gougères";"Filets de bondelle";"Rozenn"]
  [["Potage";"Côte de boeuf";"Demi-fraise"];
   ["Soufflé";"Dindonneau";"Bordure de riz"]];;
       - : bool = true
```

```
#sansredondances_exc
  ["Gougères";"Filets de bondelle";"Rozenn"]
  [["Gougères";"Côte de boeuf";"Demi-fraise"];
   ["Soufflé";"Dindonneau";"Bordure de riz"]];;
       Uncaught exception: Redondants
```

(b) Nous voulons ici que notre fonction, appelée `sansredondances_bool` renvoie une valeur booléenne en cas de redondance, suivant le comportement défini par l'exemple d'utilisation. Comme précédemment, cette fonction doit impérativement utiliser la fonction `nonredondants` définie à la question 3.

i. Écrivez une version récursive de cette fonction.

```
# let rec sansredondances_bool m = function
      [] -> true
    | t::r ->    (try (nonredondants m t) with Redondants -> false)
               & (sansredondances_bool m r);;
        val sansredondances_bool : 'a list -> 'a list list -> bool = <fun
>
```

ii. Écrivez une version de cette fonction utilisant une des fonctionnelles prédéfinies.

```
# let sansredondances_bool m l =
     List.fold_right
        (function x -> function y ->
           (try (nonredondants m x) with Redondants -> false) & y)
        l
        true;;
            val sansredondances_bool : 'a list -> 'a list list -> bool =
<fun>
```

Exemple d'utilisation :

```
# sansredondances_bool
  ["Gougères";"Filets de bondelle";"Rozenn"]
  [["Potage";"Côte de boeuf";"Demi-fraise"];
   ["Soufflé";"Dindonneau";"Bordure de riz"]];;
      - : bool = true

# sansredondances_bool
  ["Gougères";"Filets de bondelle";"Rozenn"]
  [["Gougères";"Côte de boeuf";"Demi-fraise"];
   ["Soufflé";"Dindonneau";"Bordure de riz"]];;
      - : bool = false
```

5. **Non redondance d'un *menu sur la semaine***. Écrivez, à partir de `sansredondances_bool`, une fonction à valeurs booléennes qui prend en entrée un *menu sur une semaine* et qui est vrai si et seulement si ce *menu sur la semaine* n'est pas redondant.

```
# let menusansredondances = function
      []  -> true
   | t::r -> sansredondances_bool t r;;
     val menusansredondances : 'a list list -> bool = <fun>
```

6. **Construction des *menus sur la semaine* non redondants**. Nous voulons une fonction qui prend en entrée la liste de tous les *menus sur la semaine* possibles et qui renvoie la liste de tous les *menus sur la semaine* non redondants.

(a) Écrivez une version récursive de cette fonction.

```
# let rec semainesnonredondantes = function
      []  -> []
    | t::r -> if (menusansredondances t)
              then t::(semainesnonredondantes r)
              else (semainesnonredondantes r);;
         val semainesnonredondantes : 'a list list list -> 'a list list
list = <fun>
```

(b) Écrivez une version de cette fonction utilisant une des fonctionnelles prédéfinies.

```
#let semainesnonredondantes l =
    List.fold_right (function x -> function y ->
                         if (menusansredondances x) then x::y else y)
                    l
                    [];;
        val semainesnonredondantes : 'a list list list -> 'a list list
list = <fun>
```

7. **La solution**. Écrivez une fonction qui prend en entrée la liste de toutes les entrées, celle de tous les plats de résistances, et celle de tous les desserts, et qui renvoie la liste de tous les *menus sur la semaine* possibles et non redondants.

```
#let touslesmenusnonredondants a b c =
    semainesnonredondantes (semaine (combine_trois_listes a b c));;
   val touslesmenusnonredondants :
  'a list -> 'a list -> 'a list -> 'a list list list = <fun>
```

Exercice 3 : établissement de menus en `SWI-Prolog`

1. Écrivez un prédicat qui vérifie si un élément appartient à une liste.

```
appartient(X,[X|_]).
appartient(X,[_|R]):-appartient(X,R).
```

2. Optimisez la fonction de la question 1 par l'utilisation de coupure.

```
appartient(X,[X|_]):-!.
appartient(X,[_|R]):-appartient(X,R).
```

3. **Construction des *menus***. Nous voulons un prédicat qui vérifie si une liste représente un menu.
   Écrivez un prédicat qui prend en entrée quatre listes, notées ici `X`, `Y`, `Z` et `T`, et qui est vrai si et seulement si la liste `T` est une liste à trois éléments, le premier appartenant à la liste `X`, le deuxième à la liste `Y` et le troisième à la liste `Z`.

```
menu(X,Y,Z,[A,B,C]):-appartient(A,X), appartient(B,Y), appartient(C,Z).
```

   Par la suite on appellera *menu* une liste pour laquelle ce prédicat est vrai.

4. **Construction des *menus sur la semaine***. Nous voulons un prédicat qui vérifie si une liste est un *menu sur la semaine*, c'est-à-dire si une liste est une liste d'exactement cinq *menus*.
   Écrivez un prédicat qui prend en entrée quatre listes, les trois premières correspondants, dans l'ordre, aux listes des entrées, plats de résistance et desserts. Ce prédicat doit être vrai si et seulement si la quatrième liste contient exactement cinq éléments et que chacun de ces éléments est un *menu*.

```
menussurlasemaine(X,Y,Z,[A,B,C,D,E]):-
   menu(X,Y,Z,A),menu(X,Y,Z,B),menu(X,Y,Z,C),menu(X,Y,Z,D),menu(X,Y,Z,E).
```

5. **Non redondance de deux *menus***. Nous voulons un prédicat `nonredondants` prenant en entrée deux *menus* et qui est vrai si et seulement si ces deux *menus* ne sont pas redondants, c'est-à-dire si et seulement si ces deux *menus* ne contiennent pas la même entrée, ni le même plat de résistance, ni le même dessert.

   (a) Écrivez le prédicat `nonredondants`.

```
nonredondants([A,B,C],[D,E,F]) :- A\=D,B\=E,C\=F.
```

(b) Écrivez un prédicat `redondants` qui est vrai si deux *menus* sont redondants.

```
redondants([A,_,_],[D,_,_]).
redondants([_,B,_],[_,E,_]).
redondants([_,_,C],[_,_,F]).
```

(c) Optimisez le prédicat de la question 5b par l'utilisation de coupure.

```
redondants([A,_,_],[D,_,_]):- !.
redondants([_,B,_],[_,E,_]):- !.
redondants([_,_,C],[_,_,F]).
```

(d) Écrivez une nouvelle version du prédicat `nonredondants`, cette version devra calculer sa valeur de retour en fonction de la valeur du prédicat `redondants` (usage de la négation).

```
nonredondants(A,B) :- redondants(A,B), !, fail.
nonredondants(A,B).
```

6. **Non redondance d'un *menu sur la semaine***.

(a) Écrivez un prédicat qui vérifie qu'un *menu* n'est pas redondant avec une liste de *menus* (ce prédicat vérifie que le *menu* en question n'est redondant avec aucun des *menus* de la liste).

```
menuetliste(A,[]).
menuetliste(A,[T|R]):- nonredondants(A,T),menuetliste(A,R).
```

(b) Écrivez un prédicat qui est vrai si et seulement si une liste de *menus* ne contient pas de redondances.

```
listenonredondantes([]).
listenonredondantes([T|R]):- menuetliste(T,R),listenonredondantes(R).
```

7. **Obtention des *menus sur la semaine* non redondants**. Finalement, écrivez un prédicat qui prend en entrée quatre arguments, les trois premiers étant dans l'ordre une liste d'entrées, une de plats de résistance et une de desserts. Ce prédicat doit être vrai si et seulement si son quatrième argument est un *menu sur la semaine* non redondant.

```
menusurlasemainenonredondant(A,B,C,M):-
   menussurlasemaine(A,B,C,M),listenonredondantes(M).
```

8. **Obtention du premier *menu sur la semaine* non redondant**. Modifiez le prédicat précédent pour qu'il ne retourne que le premier *menu sur la semaine* non redondant trouvé.

```
menusurlasemainenonredondant(A,B,C,M):-
   menussurlasemaine(A,B,C,M),listenonredondantes(M),!.
```