NFP 119 : Programmation Fonctionnelle

Session du 5 Juillet 2010 – Corrigé

Il vous est recommandé de rédiger de façon concise et précise, tout en écrivant lisiblement. Il sera tenu compte de la présentation dans la note de la copie. Tous les documents sont autorisés. Le barème n'est donné qu'à titre indicatif

Exercice 1 (6.5 points):

Un élève du Cnam est représenté par son nom, son numéro de carte et la liste des ses unités d'enseignement (UEs) inscrites. Une unité d'enseignement est caractérisée par son code et la note obtenue par l'élève. Cette note est négative si l'élève n'a pas encore été évalué.

```
type eleve =
  {nomEleve: string; num: string; inscriptions: uniteV list}

and uniteV = {code: string; note: float};;
```

1. Donnez 3 variables définissant les unités d'enseignement nfp107, avec la note de 13.5 ; nfp108, avec la note de 15 et nfp119 n on encore évaluée. Donnez une variable qui définit l'élève de nom "Martin", numéro de carte "1-2785" et ayant inscrit ces trois UEs.
   Réponse (1 pt) : 0.25 par variable

```
let nfp107 = {code="nfp107"; note=13.5};;
let nfp108 = {code="nfp107"; note= 15.0};;
let nfp119 = {code="nfp119"; note= -1.0};;
let martin = {nomEleve="Martin"; num = "1-2785" ;
  inscriptions= [nfp107; nfp108; nfp119]};;
```

2. Ecrivez une fonction `noteUE` qui prend **une liste d'UES** et le code d'une UE et retourne la note pour cette UE. Elle échoue si l'UE n'est pas trouvée. Donnez le type de cette fonction.
   Réponse (1.5 pts) : 1 pt fonction, 0.5 type

```
let rec noteUE codeUE l =
 match l
 with [] -> failwith "noteUE"
  | {code=c; note=n}::r ->
        if c=codeUE then n else noteUE codeUE r
;;
val noteUE : string -> uniteV list -> float = <fun>

# noteUE "nfp119" martin.inscriptions;;
- : float = -1.
```

3. Ecrivez une fonction `afficheEleve` qui prend un élève en paramètre et affiche ses données ainsi que ses unités d'enseigment avec les notes obetnues. On donnera un message approprié lorsque la note n'est pas encore disponible. *Indice : utilisez une fonction auxiliaire récursive `afficheUES` qui parcourt une liste de UEs en les affichant.*
Réponse (1 pts)

```
let afficheEleve e =
  let rec afficheUEs lU=
  match lU
  with [] -> ()
   | {code=c; note=n}::r ->
         print_string "␣␣␣"; print_string c; print_string "␣=>␣";
         print_float n; print_newline(); afficheUEs r
  in print_string"Nom:␣"; print_string(e.nomEleve);
    print_string"\n␣No.␣carte:␣"; print_string e.num;
    print_string"\n␣UEs␣inscrites:␣\n";
    afficheUEs e.inscriptions;;
val afficheEleve : eleve -> unit = <fun>

# afficheEleve martin;;
Nom: Martin
 No. carte: 1-2785
 UEs inscrites:
   nfp107 => 13.5
   nfp107 => -1.
   nfp119 => -1.
- : unit = ()
```

4. La fonctionnelle `List.do_list: ('a → unit) → 'a list → unit)` applique une fonction dont le résultat est de type `unit` sur tous les éléments d'une liste. Cela permet par exemple, d'appliquer une fonction d'affichage sur chacun des éléments d'une liste. Réecrivez la fonction `afficheEleve` de la question 3 en utilisant `List.do_list`.
Réponse (1 pts)

```
let afficheEleve2 e =
  let afficheUEs lU=
    List.do_list (fun ue ->
         print_string "␣␣␣"; print_string ue.code;
         print_string "␣=>␣"; print_float ue.note; print_newline())
  in print_string"Nom:␣"; print_string(e.nomEleve);
    print_string"\n␣No.␣carte:␣"; print_string e.num;
    print_string"\n␣UEs␣inscrites:␣\n";
    afficheUEs e.inscriptions;;
```

5. La fonctionnelle `List.find: ('a → bool) → 'a list → 'a)` permet de retourner le premier élément d'une liste qui satisfait la condition passée en premier argument sous forme de fonction. Par exemple, l'appel suivant retourne le premier élément pair de la liste :

```
# List.find (fun x -> (x mod 2 = 0)) [7; 9; 18; 4];;
- : int = 18
```

(a) Que calculent les fonctions `mystere1` et `mystere2` ? (b) Donnez le type de chacune. (c) L'une d'entre elles correspond à une solution non récursive pour l'une des questions de cet exercice. Laquelle ?

```
let mystere1 codeUE l =
  let uv = List.find (fun ue -> ue.code = codeUE) l
  in uv.note;;

let mystere2 codeUE e =
   mystere1 codeUE e.inscriptions
;;
```

Réponse (1 pts) : 0.5 chaque réponse bien expliquée avec le bon type

```
(* Donne la note d'un code d'UE dans une liste d'UES *)
(* correspond a noteUE de la question 2  *)

val mystere1 : string -> uniteV list -> float = <fun>

(* Donne la note d'un code d'UE pour un eleve *)
val mystere2 : string -> eleve -> float = <fun>
```

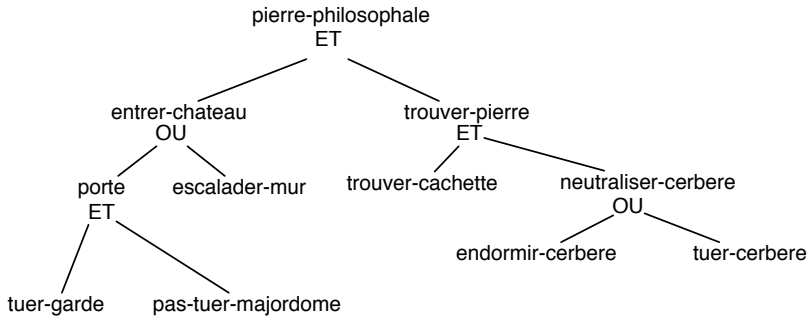
6. Ecrivez la fonction `listeNotesUE` qui prend un code d'unité d'enseignement et une liste d'élèves et qui calcule la liste de notes des élèves pour cette unité. Vous devez impérativement utiliser la fonction `noteUE` de la question 2. Vous pouvez donner une solution récursive ou utiliser la fonctionnelle `List.map` vue en cours.
   Réponse (1 pts)

```
let rec listeNotesUE codeUE le =
match le
with [] -> []
  | e::r -> let n= noteUE codeUE e.inscriptions
          in n::(listeNotesUE codeUE r)
;;
val listeNotesUE : string -> eleve list -> float list = <fun>

listeNotesUE "nfp119" [martin; martin];;

(* Version avec List.map *)
let listeNotesUE2 codeUE le =
  List.map (fun e -> noteUE codeUE e.inscriptions) le;;
```

□

Exercice 2 (5 points)**:**

Les arbres ET-OU permettent de modéliser les règles de déduction employées (p.e, en programmation logique) afin de tester si un but recherché est atteignable. Voici par exemple un arbre ET-OU qui modélise les règles d'un jeu vidéo où le joueur doit tenter de s'emparer de la pierre philosophale :



Le but final à atteindre est représenté par la racine de l'arbre, nommé ici : ≪ pierre-philosophale ≫. Chaque noeud ou feuille de l'arbre est étiquetté par le nom d'un sous-but à atteindre. Chaque sous-arbre correspond à un sous-but et chaque feuille à une action que le joueur peut ou non réussir. Il y a deux sortes de noeuds : noeuds ET et noeuds OU. Un noeud ET réussit si tous ses sous-arbres réussisent. Un noeud OU réussit si au moins l'un de ses sous-arbfres réussit.

On enregistre les actions du joueur sous forme de liste de *faits*, qui peuvent avoir chacun une valeur *true* ou *false*. Par exemple :

[*(tuer-majordome, false) ; (tuer-garde, true) ; (trouver-cachette, true) ; (endormir-cerbere,true)*]

est une liste de faits, où tous les faits de la liste on la valeur *true* sauf pour *tuer-majordome* qui vaut *false*. Le but de l'exercice est de tester si une liste de faits donnée en paramètre permet de satisfaire les conditions representées par un arbre ET-OU. Par exemple, la liste de faits donnée plus haut permet d'atteindre le but de la racine, alors que [*(tuer-majordome, false), (tuer-cerbere,true)*] ne le permet pas car il manquent les faits *(trouver-cachette, true)* et *(tuer-garde, true)*.

On utilisera les types Ocaml `arbreEtOu` pour représenter les arbres ET-OU, et `listeFaits` pour représenter les liste de faits. La variable `lf` est un exemple de liste de faits, et `pasTuerMaj` est un exemple de feuille pour l'arbre de notre exemple.

```
(* Arbres ET-OU *)
type arbreEtOu = Vide
           | Cond of string * bool
           | Et of string * arbreEtOu * arbreEtOu
           | Ou of string * arbreEtOu * arbreEtOu

(* Liste de faits *)
type listeFaits = (string * bool) list;;

(* Exemples *)

let lf = [("tuer-garde",true); ("tuer-majordome",false);
      ("trouver-cachette",true); ("endormir-cerbere", true)];;

let pasTuerMaj = Cond("tuer-majordome", false);;
```

1. Donnez la défintion complète du sous-arbre ”trouver-pierre” de l'exemple. Utilisez des variables intermédiaires pour représenter les sous-arbres (`trouverCache`, `neutraliserCerb`, etc.).

```
let toruverCache = ... (* a completer *)

let neutraliserCerb = ...
```

Réponse (1.25 pt) : *0.25 par constructeur*

```
let endormir = Cond("endormir-cerbere", true);;

let tuer = Cond("tuer-cerbere", true);;

let neutraliserCerb =
  Ou("neutraliser-cerbere", endormir, tuer);;

let cachette = Cond("trouver-cachette", true);;

let trouverPierre =
 Et("trouver-pierre", cachette, neutraliserCerb);;
```

2. Ecrire une fonction `testeArbreEtOu` qui teste si une liste de faits lf permet de valider un arbre ET-OU passés en paramètre.
Réponse : *1.75 pt → 0.5 par cas correct (sauf pour Vide → 0.25)*

```
(* question 2 *)

let rec testeArbreEtOu aEO lf =
 match aEO
 with Vide    -> true
  |   Cond(f, v) -> if (List.mem_assoc f lf) then
                      (List.assoc f lf) = v
                 else not(v)
  |   Et (e, g, d) -> testeArbreEtOu g lf && testeArbreEtOu d lf
  |   Ou (e, g, d) -> testeArbreEtOu g lf || testeArbreEtOu d lf;;

testeArbreEtOu pierrePhilo lf;;
```

3. Ecrire une fonction `testeSousArbreEtOu` qui prend une liste de faits l_f, un arbre a et une étiquette de sous-but s et qui teste si ce sous-but est validé dans l'arbre a par la liste de faits l_f. Si on suppose que la variable `pierrePhilo` contient l'arbre de notre exemple, l'appel `testeSousArbreEtOu pierrePhilo lf "porte"` doit répondre `true` alors que `testeSousArbreEtOu pierrePhilo lf "fenetre"` doit répondre `false`. *Indice* : on peut chercher récursivement le bon sous-arbre, et une fois trouvé, utiliser la fonction de la question précédente.
Réponse : *2 pt → 0.5 par cas correct*

```
(* question 3 *)

let rec testeSousArbreEtOu aEO lf a =
```

```
match aEO
with Vide    -> false
 |   Cond(f, v) -> if f=a then
                    (List.mem_assoc f lf) && (List.assoc f lf) = v
                  else false
 |   Et (e, g, d) -> if e=a then
                      testeArbreEtOu g lf && testeArbreEtOu d lf
                    else testeSousArbreEtOu g lf a || testeSousArbreEtOu d lf a
 |   Ou (e, g, d) -> if e=a then
                      testeArbreEtOu g lf || testeArbreEtOu d lf
                    else testeSousArbreEtOu g lf a || testeSousArbreEtOu d lf a;;
```

□

Exercice 3 (2.5 points):

Soit g la fonction suivante :

```
let rec somme x res =
  if x <= 0 then res
  else somme (x-1) (x+res);;
```

1. (2,5pt) Démontrez la propriété suivante : $\forall x, \forall res$ `somme` $x\ res = (\sum_{i=0}^{x} i) + res$
   Par récurrence sur x.
   - Base : $x = 0$, OK.
   - Réc : Supposons que pour un x quelconque $\forall res$, `somme` $x\ res = (\sum_{i=0}^{x} i) + res$. Montrons qu'alors $\forall res$, `somme` $(x+1)\ res = (\sum_{i=0}^{x+1} i) + res$. Soit res un entier positif quelconque, par définition `somme` $(x+1)\ res =$ `somme` $x\ (x+1+res)$. Or par hypothèse de récurrence `somme` $x\ (x+1+res) = (\sum_{i=0}^{x} i) + (x+1+res) = (\sum_{i=0}^{x+1} i) + res$. OK.
2. (0,5pt) En déduire que $\forall$ `x,somme x 0` = $(\sum_{i=0}^{\mathtt{x}} i)$

□

Exercice 4 (4 points):

Soit f la fonction suivante :

```
let rec f (x,y) =
  if x<=0 or y<=0 then 0
  else if x < y then f (x,x)
  else 1 + f (y-1,y)
```

1. Démontrez que cette fonction termine sur $\mathbb{N} \times \mathbb{N}$. On prend l'ordre $<_{\mathbb{N}}$ sur la somme des deux arguments.
2. Démontrez la propriété suivante : $\forall (x, y), f(x, y) \leq \sum_{i=1}^{x} i$. Vous procéderez par récurrence forte en traitant d'abord les cas $x = 0$ et $y = 0$. Par récurrence forte sur la somme de x et y.
   - Supposons que pour un $n \in \mathbb{N}$ quelconque, $\forall (x, y)$ t.q. $x + y < n, f(x, y) \leq \sum_{i=1}^{x} i$. Démontrons qu'alors $\forall (x, y)$ t.q. $x + y = n, f(x, y) \leq \sum_{i=1}^{x} i$.
     Soient x et y tels que $x + y = n$. On distingue 4 cas :
     (a) $x = 0$ alors $f(x, y) = 0$ OK.
     (b) $y = 0$ alors $f(x, y) = 0$ OK.

(c) $0 < x < y$, alors $f(x,y) = f(x,x)$, comme $x < y$ on a $x + x < n$ donc par hypothèse de récurrence, $f(x,x) \leq \sum_{i=1}^{x} i$. OK (puisque $f(x,y) = f(x,x)$).

(d) $x \geq y > 0$, alors $f(x,y) = 1 + f(y-1,y)$. Comme $x \geq y$ on a $y-1+y < x+y$ donc par hypothèse de récurrence, $f(y-1,y) \leq \sum_{i=1}^{y-1} i$. Donc $1 + f(y-1,y) \leq (\sum_{i=1}^{y-1} i) + 1$. Comme $y > 0$, $(\sum_{i=1}^{y-1} i) + 1 \leq (\sum_{i=1}^{y-1} i) + y = \sum_{i=1}^{y} i$, OK.

□

Exercice 5 (3 points)**:**

Soit f la fonction suivante

```
let rec f (n,l) =
 if n = 0 then true
 else
  match l with
   | [] -> false
   | 0::l' -> f (n, l')
   | x::l' -> 1 + f ((n-1), ((x-1)::l'))
```

On appelle $L_{\mathbb{N}}$ l'ensemble des listes d'entiers positifs ou nuls et $\sum(l)$ la somme des éléments de la liste d'entiers l.

1. Démontrez que la fonction f termine sur $\mathbb{N} \times L_{\mathbb{N}}$.
2. Démontrez que $\forall n \in \mathbb{N}, \forall l \in L_{\mathbb{N}}, f(n,l) = true \rightarrow n \leq \sum(l)$.

□