Université Paris Diderot
L2 Math/Info, L3 Info, M1 Lingustique

Programmation Fonctionnelle
Année 2011-2012

Examen

Jeudi 5 janvier 2012

Motivez bien vos réponses. On recommande de *bien lire* l'énoncé d'un exercice avant de commencer à le résoudre.

Tout document papier est autorisé. Les ordinateurs, les téléphones portables, comme tout autre moyen de communication vers l'extérieur, doivent être éteints. Le temps à disposition est de 3 heures.

Les exercices doivent être rédigés en fonctionnel pur : ni références, ni tableaux, ni boucles `for` ou `while`, pas d'enregistrements à champs mutables. Chaque fonction ci-dessous peut utiliser les fonctions prédéfinies (sauf indication contraire), et/ou les fonctions des questions précédentes.

Exercice 1 Donnez le type, et s'il y a lieu, la valeur des expressions suivantes. Si la valeur est une fonction, donnez seulement son type. Si OCaml renvoie une erreur, décrivez succintement l'erreur sans donner le message exact. Par exemple "`x` est de type `string`, mais `int` était attendu".

1.
```
let pproj (a,b) = a;;
```
2.
```
List.map (fun x -> -1) ["a";"b";"c"];;
```
3.
```
(+) 2;;
```
4.
```
let double f =
    let two = 2
    in fun x -> two*(f x) ;;
```
5.
```
let retourner l =
    let rec aux acc =
      function
        [] -> acc
      | a::r -> aux (a::acc) r
    in aux [] l ;;
```
6.
```
List.fold_left (fun x y -> y^x) "" ["abc";"def";"ghi"];;
```
7.
```
List.fold_right (fun a b -> a - b) [1;2;3] (-1);;
```
8.
```
if (1-1 = 0.0) then true else false;;
```
9.
```
type abc = A | B | C;;
(fun x -> match x with _ -> C) A;;
```
10.
```
let f x y z = ((x = y) = z);;
```

Exercice 2 Les quatre fonctions ci-dessous doivent être écrites sans l'aide des fonctions prédéfinies.

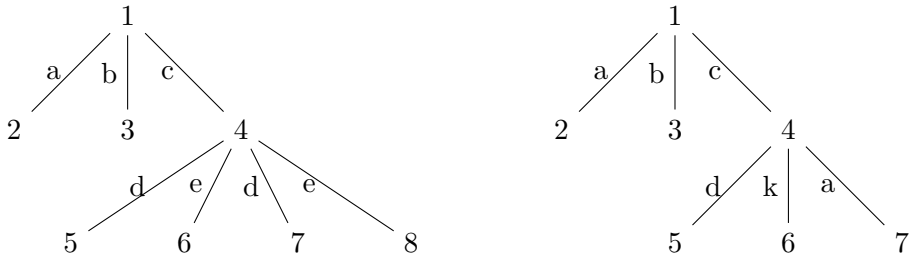
1. Écrire l'implémentation d'une fonction `for_all` telle que :
   `for_all : ('a -> bool) -> 'a list -> bool`
   `for_all p [x1;...;xn] = (p x1) && ... && (p xn)`
   Autrement dit, la fonction `for_all` considère une fonction `p` et une liste `[x1;...;xn]`, et retourne `true` si et seulement si *chaque* `(p xi)` prend la valeur `true`.
2. Écrire l'implémentation d'une fonction `for_all2` telle que :
   `for_all2 : ('a -> 'b -> bool) -> 'a list -> 'b list -> bool`
   `for_all2 p [x1;...;xn] [y1;...;yn] = (p x1 y1) && ... && (p xn yn)`
3. Écrire l'implémentation d'une fonction `exists` telle que :
   `val exists : ('a -> bool) -> 'a list -> bool`
   `exists p [x1;...;xn] =(p x1) || ... || (p xn)`
   Autrement dit, la fonction `exists` considère une fonction `p` et une liste `[x1;...;xn]`, et retourne `true` si et seulement si *l'un au moins* des `(p xi)` prend la valeur `true`.
4. Écrire l'implémentation d'une fonction `exists_croise` telle que :
   `exists_croise : ('a -> 'b -> bool) -> 'a list -> 'b list -> bool`
   `exists_croise p [x1;...;xn][y1;...;yn]` renvoie `true` si et seulement s'il existe $i, j \in \{1, \ldots, n\}$ tel que `(p xi yj)` prenne la valeur `true`.
   Indication : Utilisez la solution de la question précédente.

Exercice 3

1. Appelons *suffixe* de `[x1;...;xn]` chacune des listes
   `[x1;...;xn]`, `[x2;...;xn]` ..., `[xn-1;xn]`, `[xn]`, `[]`
   Ecrire une fonction `suffixes : 'a list -> 'a list list` telle que `suffixes l` renvoie la liste complète des suffixes de `l`, triés (comme ci-dessus) par taille décroissante.
2. Appelons *préfixe* de `[x1;...;xn]` chacune des listes
   `[]`, `[x1]`, `[x1;x2]` ...`[x1;...;xn-1]`, `[x1;...;xn]`
   Ecrire une fonction `prefixes : 'a list -> 'a list list` telle que `prefixes l` renvoie la liste complètes des préfixes, triés (comme ci-dessus) par taille croissante.
3. Appelons *séparation* de `[x1;...;xn]` chacun des couples de listes
   `([],[x1;...;xn])`, `([x1],[x2;...;xn])` ...`([x1;...;xn-1],[xn])`, `([x1;...;xn], [])`
   Ecrire une fonction `separations : 'a list -> 'a list list` telle que `separations l` renvoie la liste complètes des séparations de `l`, triés (comme ci-dessus) par ordre de membre gauche croissant.
4. Appelons *rotation* de `[x1;...;xn]` chacune des listes
   `[x1;...;xn]`, `[x2;...;xn;x1]`, `[x3;...;xn;x1;x2]` ...`[xn;x1;...;xn-1]`
   La liste `[xi+1;...;xn;x1;...;xi]` est appelée *rotation gauche de rang* `i` de `[x1;...;xn]`. Par convention, la rotation gauche de `[x1;...;xn]` de rang `n + i` est égale à sa rotation gauche de rang `i`. La liste vide est l'unique rotation de la liste vide.
   Ecrire une fonction `rotation_gauche : 'a list -> int -> 'a list` telle que `rotation_gauche l i` renvoie la rotation gauche de `l` de rang `i`.

Exercice 4 Un *arbre de traits* est une structure récursive définie comme suit : un arbre de trait consiste en un nœud contenant une certaine information – appelée *étiquette* de ce nœud – et qui a un nombre quelconque de fils (ce nombre peut être zéro). Chacun des fils d'un nœud est adressé par un *nom*. Un nœud peut avoir plusieurs fils avec le même nom. Un arbre de traits sera dit *bien formé* si aucun de ses nœuds n'a deux fils avec le même nom.

Voici deux exemples d'arbres de traits. L'arbre à gauche n'est pas bien formé, tandis que celui à droite l'est. Dans ces dessins, le nom du fils d'un nœud est noté à gauche du trait reliant ce nœud à ce fils.



Dans un arbre de trait, tous les noms doivent être de même type, ainsi que toutes les étiquettes de nœuds ; mais ces deux types peuvent être différents.

1. Donner en OCaml la définition d'un type polymorphe permettant de représenter des arbres de traits (bien formés ou pas) pour un type de noms quelconque et un type d'étiquettes quelconque.
2. Donner une expression OCaml représentant l'arbre dessiné à droite dans l'exemple précédent, selon votre définition de type trouvée dans la question 1.
3. Écrire une fonction prenant en entrée un arbre de trait, et renvoyant un booléen indiquant si l'arbre est bien formé ou pas.
4. Etant donnés un arbre de trait t bien formé et une liste de noms l, appelons (si elle existe) *étiquette de t à l'adresse l* l'étiquette du nœud atteint en suivant successivement les noms de l à partir de la racine de t.

   Par exemple, si t est l'arbre droit de l'exemple, son étiquette à l'adresse `['a']` est 2, son étiquette à l'adresse `['c';'a']` est 7, et son étiquette à l'adresse `['b';'k';'d']` n'est pas définie.

   Écrire une fonction prenant en entrée un arbre de trait t (qu'on supposera bien formé), une liste de noms l, et qui renvoie l'étiquette de t à l'adresse l. Que proposez-vous pour gérer le cas où l'étiquette de t à l'adresse l n'est pas définie ?