Examen de Programmation Fonctionnelle

8 novembre 2023

1 Typage et Syntaxe

Question 1.1 *Les fonctions suivantes sont typables ; donner leur type.*

1. `let f x = x`
2. `let f x = if (x 0) then (x 1) else (x 2)`
3. `let f g h x = g (h x)`
4. `let f x y z = (if x then (y x) else 0) + z`
5. `let f x y = x y + 3`
6. `let f x y = x (y + 3)`
7. `let f x = match x with None -> [] | Some z -> [x]`
8. `let f x = let (gauche,droite) = x in (gauche || droite)`
9. `let f g = g ()`
10. `let f l z = List.map (fun x -> x + z) l`
11. `let f l z = List.map (fun x -> z :: x) l`
12. `let f g h = if (h ()) then  g() else (); g ()`
13.
```
let rec f l x = match l with
    | [] -> x
    | y :: ys when x = y -> f ys x
    | y :: ys -> if y then (f ys x) else y
```

Indiquer si les fonctions suivantes sont typables ; si oui, donner leur type.

1. `let f g = fun x -> g (g x)`
2. `let f g = g g`
3.
```
let rec f l x = match l with
    | [] -> if x then [x] else []
    | y :: ys -> (y + y) :: (f ys x)
```
4.
```
let rec f l = match l with
    | [] -> [[]]
    | y :: ys -> ys :: (f ys)
```
5. `let f g = let x = g 3`

Question 1.2 *Considérons le code suivant.*

```
let x x y z =
let z y = y + z in
let y x = x + y in
z (y x)
```

1. *Pour chaque occurrence des noms x, z et y, indiquer où se trouve son lieur — c'est-à-dire la déclaration qui correspond à cette occurrence — soit en utilisant des flèches, soit un code couleur, soit en commentant le numéro de la ligne du lieur à la ligne où apparaît la variable. Faire en sorte que cela reste lisible, quitte à recopier plusieurs fois le code.*
2. *Dire ce que la fonction calcule.*

3. *Considérons maintenant le code suivant :*

```
let x x y z =
let z y f = y + f z in
let y x = x + y in
z x (fun x -> y (x*x))
```

Indiquer ce que la fonction calcule (pas besoin d'indiquer les lieurs).

2 Arbres

Question 2.1 *a. Définir un type qui permet de représenter les arbres binaires, avec des entiers aux nœuds.*

b. Écrire une fonction récursive qui retourne la taille d'un tel arbre.

c. Écrire une fonction récursive qui retourne la liste de tous les sous-arbres de l'arbre donné en argument. (Un sous-arbre est un arbre qui apparaît tel quel, à partir d'un noeud de l'arbre de départ et qui va jusqu'aux feuilles)

3 Listes

Question 3.1 *a. Écrire une fonction récursive qui retourne la taille d'une liste (sans itérateur).*

b. Écrire la même fonction en utilisant un itérateur (`List.fold_left` *ou* `List.fold_right`*). (**Rappel :** l'ordre des arguments est* `List.fold_left f x l` *et* `List.fold_right f l x` *— et f prend 2 arguments)*

c. Écrire une fonction récursive `iter : ('a -> unit) -> 'a list -> unit`*, telle que* `iter f l` *applique f une fois à chaque élément de la liste l.*

d. Écrire la même fonction `iter`*, mais cette fois en utilisant* `List.map`*.*

e. Écrire la même fonction `iter`*, mais cette fois sans récursivité (en utilisant une référence de liste et une boucle* `for`*).*

Question 3.2 *Fonction mystère : expliquer ce que fait* `mystere1`*. Donner son type et donner la valeur de l.*

```
let rec mystere1 f l = match l with
  | [] -> []
  | [None] -> []
  | [Some a] -> [(f a)]
  | x :: xs -> (mystere1 f [x]) @ (mystere1 f xs)

let l = mystere1 (fun x -> x + x) [Some 0; None; Some 1; None; Some 4]
```

4 Effets de bord

Question 4.1 *a. Quel est le problème avec le code* `let mat = Array.make 3 (Array.make 3 0)` *?*

b. Donner la valeur de x après l'exécution du code ci-dessous :

```
let t = let r = ref 0 in [| r; r; ref (!r) |]
let x = t.(2) := 7;
        t.(1) := 1;
        t.(0) <- t.(2);
        t.(0) := 4;
        t.(2) := !(t.(0)) + !(t.(1));
        !(t.(0))
```

Question 4.2 *Fonction mystère :*

a. *Expliquer ce que fait* `mystere2`. *Donner aussi son type.*

b. *Donner le résultat,* ***à chaque étape****, si on appelle successivement* `mystere2 (In 4)`, `mystere2 (In 0)`, `mystere2 (In 3)`, `mystere2 (Out 0)`, `mystere2 (Out 2)`, `mystere2 (Out 0)`, `mystere2 (Out 2)`, `mystere2 (Out 0)`.

```
type 'a input = In of 'a | Out of int

let rec mystere2 =
  let li = ref [] in
  fun x -> match x with
    | In a -> li := a :: (!li); None
    | Out n -> (match (!li,n) with
        | [],_ -> None
        | y::ys, 0 -> Some y
        | y::ys, n when n < 0 -> None
        | y::ys, _ -> li := ys; mystere2 (Out (n - 1))
      )
```

Question 4.3 *Expliquer ce que font les fonctions* `cover` *et* `uncover`. *Donner aussi leur type.*

```
exception Wrong
let (cover,uncover) =
  let s = ref 0 and k = ref 0 in
  ((fun x y -> s := x; k := y),
  (fun x -> if (x = !k) then !s else raise Wrong))
```