Licence STS mention Informatique, semestre 5 — 2017–2018
Programmation fonctionnelle avancée — Durée : 2h
2 pages

Examen du 25 octobre 2017

Les notes de TD/TP manuscrites ainsi que les transparents de cours sont les seuls documents autorisés. Veuillez lire attentivement les questions. Veuillez rédiger proprement, clairement et de manière concise et rigoureuse.

1 Typage

Question 1 Les fonctions suivantes (`f1`, `f2`, `f3` et `f4`) sont-elles bien typées ? Si oui, donner leur type, sinon préciser pourquoi.

```
let f1 (x, y) z = if z then x :: [y] else []

let f2 x y = 4 + ((y x) 1)

let rec f3 x = not (f3 x)

let rec f4 x =
  match x with
  | [] -> []
  | z :: _ -> f4 z
```

Question 2 Étant donnée une fonction `g` de type `('a * 'b) -> 'a`, les applications suivantes de cette fonction sont-elles bien typées ? Si non, préciser pourquoi.

(**2.1**) `g ('e', 'b')`
(**2.2**) `g (g ((1,2),3))`
(**2.3**) `(g (2.5, 4)) + 4`

2 Fonction récursive terminale

Question 3 Soit la fonction `somme_carre`, de type `int -> int`, telle que `somme_carre n` renvoie la somme des carrés de 1 à `n`, c'est-à-dire $1+2^2+3^2+\cdots+n^2$. Écrire une version récursive terminale de cette fonction.

Question 4 Soit la fonction `foo` ci-dessous.

```
let rec foo f x =
  if x = 0 then
    f x
  else
    foo (fun y -> f (x + y)) (x-1)
```

(1) Quel est le type de cette fonction ?

(2) Quel est le résultat de `foo (fun x->x) 5` ?

(3) Donner le code d'une fonction équivalente *non récursive terminale*.

3 Programmation sur les listes

Question 5 Écrire une fonction `intersection`, de type `'a list -> 'a list -> 'a list`, qui renvoie l'intersection de deux listes supposées *triées* par ordre croissant. La liste renvoyée sera elle-même triée par ordre croissant. Par exemple, `intersection [1;3;5] [3;4;5;8]` renverra `[3;5]`.

Question 6 En utilisant *obligatoirement* un itérateur sur les listes, écrire une fontion `duplique`, de type `'a list -> 'a list`, qui permet de dupliquer tous les éléments d'une liste. Par exemple, `duplique [1;2;3]` renverra la liste `[1;1;2;2;3;3]`.

Question 7 Écrire une fonction `nombre_pair_d_elements`, de type `'a list -> bool` qui vérifie qu'une liste a un nombre pair d'éléments.

Question 8 En utilisant *obligatoirement* un itérateur sur les listes, écrire une fonction `second_max`, de type `'a list -> 'a`, qui renvoie le second plus grand élément d'une liste. Par exemple, `second_max [3;1;5;2;9;0]` renverra `5`. La fonction lèvera une exception sur une liste vide. Par définition, le second plus grand maximum d'une liste `[v]` réduite à un élément sera égal à `v`.

Les deux questions suivantes ont pour objectif de programmer des fonctions de compression/décompression de listes en utilisant la technique de *codage par plage* (algorithme de *run-length encoding*, ou RLE).

Cette technique de compression consiste à remplacer les suites consécutives de valeur `v` par des paires $(\texttt{v}, n)$, où n est la longueur de la suite. Par exemple, la compression de la liste de caractères `['a';'a';'b';'c';'c';'c';'a';'a';'a';'b';'b']` doit produire la liste `[('a', 2); ('b', 1); ('c', 3); ('a', 3); ('b', 2)]`.

Question 9 Écrire une fonction `compression`, de type `'a list -> ('a * int) list`, qui applique la méthode RLE pour compresser une liste.

Question 10 Écrire une fonction `ajoute_a_liste`, de type : `'a * int -> 'a list -> 'a list`, telle que `ajoute_a_liste (c,n) l` ajoute `n` valeur `v` en tête de la liste `l`. Par exemple, `ajoute_a_liste ('a', 2) ['b';'b';'c']` doit renvoyer la liste `['a'; 'a'; 'b'; 'b'; 'c']`.

Question 11 En utilisant la fonction précédente ainsi qu'un *itérateur* sur les listes, écrire une fonction `decompression`, de type `('a * int) list -> 'a list`, pour décompresser une liste selon la méthode RLE.