Licence STS mention Informatique, semestre 4 — 2017–2018
Introduction à la programmation fonctionnelle — Durée : 2h
2 pages

Examen du 26 juin 2018

Les notes et les transparents du cours sont les seuls documents autorisés. Veuillez lire attentivement les questions. Veuillez rédiger proprement, clairement et de manière concise et rigoureuse. Le langage à utiliser dans cet examen pour répondre aux questions est OCaml, sans utiliser de traits impératifs du langage (autre que les fonctions d'affichage).

Question 1 Les fonctions suivantes sont-elles bien typées ? Si oui, donner leur type, sinon, justifier pourquoi.

```
1) let rec f1 x = if x < 1 then f1 (x - 1) else [0]
2) let rec f2 x y z = if z = 0 then f2 y y 0 else f2 x x 1
3) let f3 x = match x with [] -> 0 | _ :: s -> s + 1
4) let rec f4 f l = match l with [] -> [] | _ :: s -> f4 f (f s)
```

Question 2 Donner le type de la fonction `f5` ci-dessous et les valeurs des variables `v1`, `v2`, `v3`, `v4` et `v5` définies également ci-dessous.

```
let f5 p =
  match p with
  | (0, _) -> 0
  | (_, 0) -> 1
  | (1, x) -> 10 + x
  | (x, 1) -> x + 20
  | _ -> 2

let v1 = f5 (0,1)
let v2 = f5 (0,0)
let v3 = f5 (1,0)
let v4 = f5 (1,4)
let v5 = f5 (4,1)
let v5 = f5 (25,5)
```

Programmation sur les listes

Question 3 Écrire une fonction `compression`, de type `'a list -> 'a list`, telle que `compression l` remplace les éléments consécutifs égaux de `l` par un seul élément. Par exemple, `compression ['a'; 'a'; 'a'; 'b'; 'a'; 'b'; 'c'; 'c'; 'b';'b']` renvoie `['a'; 'b'; 'a'; 'b'; 'c'; 'b']`.

Question 4 Écrire une fonction `elim`, de type `'a list -> int -> 'a list`, telle que `elim l k` renvoie une liste où tous les `k`ème éléments sont éliminés. Par exemple, `elim [1;2;3;4;5;6] 2` renvoie la liste `[1;3;5]`. De même, `elim [1;2;3;4;5;6] 3` renvoie la liste `[1;2;4;5]`. Enfin, `elim [1;2;3;4;5;6] 1` renvoie la liste `[]`.

Question 5 Écrire une fonction `paires`, de type `'a list -> ('a * 'a) list` telle que `paires l` renvoie *toutes* les paires constituées des éléments de `l`. Par exemple, `paires [1;2;3;4]` renvoie la liste `[(1, 3); (3, 1); (1, 2); (2, 1); (2, 3); (3, 2)]`.

Polynômes

Les questions suivantes vont permettre d'écrire des fonctions pour manipuler des polynômes à une variable X avec coefficients entiers.

Un polynôme est représenté par une liste de monômes. Chaque monôme est défini comme un enregistrement avec deux champs : `coeff`, de type `int`, contient le coefficient et `degre`, de type `int`, contient le degré.

```
type monome = { coeff : int; degre : int }
type polynome = monome list
```

On suppose également que la liste des monômes est toujours triée par ordre *décroissant* et qu'elle ne contient aucun monôme avec un coefficient nul.

Par exemple, le polynôme $X^5 - 2X^4 + 3X + 1$ est représenté par la valeur OCaml suivante :

```
let p = [ {coeff=1;degre=5}; {coeff=(-2);degre=4};
          {coeff=3;degre=1} {coeff=1;degre=0}     ]
```

Question 6 Écrire une fonction `afficher_m`, de type `monome -> unit` qui permet d'afficher un monôme. On fera en sorte que l'affichage respecte les spécifications suivantes :

- L'affichage d'un monôme de degré 0 est réduit à l'affichage de son coefficient. Si ce dernier est égal à 0 alors rien n'est affiché.
- Le degré du monôme n'est pas affiché s'il est égal à 1

e Par exemple, `afficher_m` {`coeff=3; degre=1`} soit afficher `3X`, `afficher_m` {`coeff=4; degre=0`} soit afficher `4` et `afficher_m` {`coeff=-5; degre=2`} affiche `-5X^2`.

Question 7 En utilisant la fonction précédente, écrire une fonction `afficher_`, de type `polynome -> unit`, qui affiche un polynôme. On fera attention à ce que le signe du premier monôme de soit affiché que s'il est négatif. Par exemple, l'affichage du polynôme `p` défini ci-dessus doit donner :

```
# afficher_p p;;
1X^5-2X^4+3X+1
```

Question 8 Écrire une fonction `deriver`, de type `polynome -> polynome`, telle que `deriver p` renvoie le polynôme dérivé de `p`. On fera attention à ne pas générer de monômes avec des coefficients nuls.

Question 9 Écrire une fonction `somme`, de type `polynome -> polynome -> polynome`, telle que `somme p1 p2` renvoie la somme des deux polynômes `p1` et `p2`, en faisant attention à bien respecter les invariants de la structure de données `polynome` (comme définis au dessus).

Question 10 Écrire une fonction `evaluer`, de type `polynome -> int -> int`, telle que `evaluer p v` renvoie la valeur du polynôme `p` pour $X =$ `v`.