Licence STS mention Informatique, semestre 5 — 2016–2017
Programmation fonctionnelle avancée — Durée : 2h
3 pages

Examen du 26 octobre 2016

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 Mini QCM

Répondre aux questions en écrivant sur vos feuilles le numéro de la question et en indiquant *uniquement* **oui** ou **non**.

Question 1 Le code de la fonction `f` suivante est-il *syntaxiquement* correct ?

```
let rec f x y =
  if x < y then print_int x;  y + 2
;;
```

Question 2 Étant donnée la signature suivante d'une fonction `g` :

```
val g : ('a * 'b) list -> ('b -> unit) -> bool
```

Est-ce que les applications suivantes de la fonction `g` sont correctes ?

(1) `g [("bonjour", 10)] print_int`
(2) `g [('a', 5.5)] (fun x -> print_int x)`
(3) `g [] (fun x -> x)`

Question 3 Les fonctions suivantes sont-elles récursives *terminales* ?

(1)
```
let rec h1 l =
  match l with
   | [] -> 0
   | [x] -> h1 []
   | x :: y :: s -> x + h1 s
```

(2)
```
let rec h2 (x, y) = h2 (y, x - 1)
```

(3)
```
let rec h3 x = h3 (h3 (x - 1))
```

2 Typage

Question 4 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 = (x (y<z)) :: z

let rec f2 (x, y) =
  if x = 0 then y else f2 (y, x)

let f3 x = x x

let rec f4 x y = f4 y ([]::x)
```

3 Boucles à l'aide de fonctions

Question 5 Écrire à l'aide de fonctions récursives (sans traits impératifs) en OCaml la fonction `foo` ci-dessous (écrite en langage C) :

```
int foo(int x, int y) {
  int j = 0;
  int v = x;
  for (int i = 0; i<=y; i++)
    v = v + x;
  while (j < v ) {
    v  = v - x;
    j = j + v;
  };
  return (v);
}
```

4 Fonction récursive terminale

Question 6 Donner une version récursive terminale de la fonction `bar` ci-dessous.

```
let rec bar z l =
  match l with
  | [] -> 10
  | x :: s -> x * z + (bar (z - 1) s)
;;
```

5 Programmation

Le but de cet exercice est de manipuler des nombres entiers écrits dans une base quelconque. Dans la suite, on écrira par exemple 342_7 le nombre 342 en base 7.

Pour représenter ces nombres, nous allons utiliser le type enregistrement `t` suivant :

```
type t = {
    digits : int list;
    base : int;
}
```

où le champ `digit` contient la liste des chiffres du nombre, avec le chiffre des unités en tête de liste, et où le champ `base` contient la base dans laquelle est représenté le nombre. Par exemple, le nombre 342_7 sera représenté par la valeur suivante :

```
{
  digits = [2; 4; 3];
  base = 7;
}
```

Question 7 Écrire une fonction `decompose`, de type `int -> int -> t`, telle que `decompose b n` renvoie une valeur de type `t` correspondant à la décomposition du nombre `n` (donné comme un entier en base 10) vers la base `b`. Par exemple, le nombre 329_{10} correspond au nombre 2304_5. Ainsi, `decompose 5 329` doit renvoyer la valeur suivante :

```
{
  digits = [4; 0; 3; 2];
  base = 5;
}
```

Question 8 En utilisant *obligatoirement* un itérateur sur les listes, écrire une fonction `print`, de type `t -> unit`, qui affiche à l'écran une valeur de type `t` de la manière suivante :

```
Nombre : xxx
En base : xxx
```

Par exemple, la valeur de la question précédente doit être affichée comme ceci :

```
Nombre : 2304
En base : 5
```

Question 9 En utilisant *obligatoirement* un itérateur sur les listes, écrire la fontion `to_int`, de type `t -> int`, qui permet de convertir une valeur de type `t` vers un entier en base 10.

Question 10 *Sans réaliser de changement de base*, écrire une fonction `add`, de type `t -> t -> t`, telle que `add v1 v2` renvoie la somme de nombre `v1` et `v2` dans la base de ces deux nombres (les bases de `v1` et `v2` sont supposée être les mêmes).

Question 11 En utilisant *obligatoirement* un itérateur sur les listes, écrire une fonction `positions`, de type `t -> int list`, telle que `positions v` renvoie les positions (sous forme d'une liste d'entiers) des chiffres *non nuls* d'une valeur de type `t`. On prendra pour convention que le chiffre des unités est à la position 0.
Par exemple, si `v` est la valeur suivante (nombre en base 2) :

```
{ digits = [1; 0; 1; 1; 0; 0; 0; 1]; base = 2 }
```

alors `positions v` renvoie la liste de positions `[7; 3; 2; 0]`.