2e année de licence de math-info,
3e année de licence d'informatique,
1re année master de linguistique informatique 2009–2010

Examen de Programmation Fonctionnelle

1re session — Janvier 2010

Durée : 3h

Documents autorisés : cours, TPs, projets.
Livres interdits — Manuel d'Ocaml et documentation de la bibliothèque standard interdits —
Échange de documents interdit — Ordinateurs et téléphones interdits

Vous devez répondre aux questions en OCaml, en utilisant uniquement un style fonctionnel (pas de références, mais vous pouvez utilisez des exceptions). Une fonction écrite en style impératif ne rapportera aucun point.

Le barème ci-dessous est indicatif et est susceptible d'être modifié.

Vous pouvez utiliser les fonctions de la bibliothèque standard.

1 Exercices (9 points)

1. Donner le type de l'expression suivante. Justifier rapidement.

```
fun x -> 1-x
```

2. Donner le type et la valeur de l'expression suivante. Justifier.

```
(fun x -> print_string (x^"b")) "a"
```

3. Donner le type de l'expression suivante. Justifier.

```
Some (fun x -> None)
```

4. Donner le type et la valeur de l'expression suivante. Justifier.

```
(fun x y -> y x) (fun x -> x) (fun x -> x) 2
```

5. Donner le type et la valeur de l'expression suivante. Justifier rapidement.

```
let a = 1 in
let f () = let a = 2 in a + a in
f () + a
```

6. Donner le type de l'expression suivante. Justifier.

```
List.map (fun p -> p 7) [(fun n m -> n+m)]
```

7. On définit la fonction `f` de la façon suivante.

```
let f s x = try int_of_string s with Failure _ -> x;;
```

Quel est son type ? Justifier.

8. Sachant que l'interface du module `List` contient :

```
val fold_right2 :
    ('a -> 'b -> 'c -> 'c) -> 'a list -> 'b list -> 'c -> 'c
```

Les expressions suivantes sont elles bien typées ? Si oui, donner leur type.
Dans les deux cas, justifier.

```
(* 1 *)
List.fold_right2 (fun a b c -> (a, b)::c) ['a'] [1] []

(* 2 *)
List.fold_right2 (fun a b c -> a) [100] [1] (Some 27);;

(* 3 *)
List.fold_right2
     (fun a c -> function None -> Some (a+1) | _ -> c)
```

9. On définit :

```
type t1 = Aa of string t2 | Bb of int
and 'a t2 = { ch1 : 'a; ch2 : t1; }
```

Quel est le type de la valeur suivante ? Justifier.

```
fun x z -> match x.ch2 with
  | Aa y -> y.ch1
  | _ -> z;;
```

10. Curryfier la fonction suivante :

```
let f x = (fst x) - (snd x)
```

11. Écrire une version du programme suivant qui n'utilise ni boucles, ni références, ni données mutables. La fonction doit avoir exactement le même type et le même comportement.

```
let somme n =
   let resultat = ref 0 in
   for i = 1 to n do
     let res = ref (-1 ) in
     while !res < 0 do
       print_endline "Entrez un entier positif :";
       res := read_int ()
     done;
     resultat := !resultat + !res;
   done;
   !resultat
```

2 Problème (11 points)

On souhaite écrire un module pour manipuler des ensembles de valeurs. Pour l'exercice, nous représenterons d'abord ces ensembles par des listes. Vous pourrez utiliser des fonctions prédéfinies de la bibliothèque standard mais ce n'est pas obligatoire.

2.1 Représentation naïve des ensembles

1. Définir l'interface d'un module `Ensemble` qui contiendra notamment les valeurs suivantes :
   - l'ensemble vide ;

- une fonction d'ajout dans un ensemble ;
- une fonction de recherche d'un élément dans un ensemble.

2. Donner l'implémentation du module. Pour l'instant la fonction d'ajout se contentera de mettre l'élément dans la liste sans tester sa présence éventuelle. Les listes pourront donc comporter plusieurs fois le même élément.
3. Ajouter une fonction qui calcule le cardinal (nombre d'éléments) d'un ensemble. Donner son interface et son implémentation. Par exemple, le cardinal de l'ensemble représenté par la liste `[1; 4; 5; 1; 4; 2; 4; 3]` est 5.
4. Quelle est la complexité de la fonction précédente ?

2.2 Listes sans répétition

Nous allons modifier la représentation des ensembles pour que chaque élément n'apparaisse qu'une seule fois dans une liste.

1. Écrire une fonction qui transforme une liste quelconque en un ensemble (liste sans répétition).
2. Donner une nouvelle version de la fonction d'ajout qui ne permet pas de construire d'ensembles mal formés (c'est-à-dire d'ensembles contenant plusieurs fois le même élément).
3. Votre fonction de recherche doit-elle être modifiée ?
4. Donner une nouvelle version de la fonction de calcul du cardinal.
5. Quel est la complexité de cette dernière fonction ?
6. Comparez la complexité de votre nouvelle fonction d'ajout à celle de l'ancienne.
7. Proposez une solution pour que la fonction qui renvoie le cardinal soit en temps constant.
8. Comment faire pour assurer que les utilisateurs de votre module ne manipulent jamais d'ensembles mal formés ?

2.3 Parties d'un ensemble

1. Ajouter au module une fonction qui renvoie l'ensemble des parties d'un ensemble.
2. Ajouter une fonction qui renvoie l'ensemble des parties à n éléments d'un ensemble.

2.4 Représentation d'un ensemble par sa fonction caractéristique

On définit le type

```
type intinf = Infini | Int of int
```

On souhaite représenter un ensemble par le type suivant :

```
type 'a ens = {card : intinf; f : 'a -> bool}
```

Le champ `card` représente le cardinal de l'ensemble. `Infini` signifie que l'ensemble est infini.
Le champ `f` est la fonction caractéristique de l'ensemble (c'est-à-dire une fonction f telle que $f(x)$ est vrai si et seulement si x est dans l'ensemble).

1. Définir l'ensemble de caractères $\{'a'; 'c'; 'o'\}$ en utilisant cette représentation
2. Définir l'ensemble des entiers positifs ou nuls pairs en utilisant cette représentation.
3. Définir la fonction de recherche d'un élément dans un ensemble.
4. Définir la fonction d'ajout d'un élément à un ensemble.
5. Définir une fonction qui réalise l'union de deux ensembles.

$g(f(a))$

2.5 Fonctions d'énumération

On appellera *fonction d'énumération* d'un ensemble D (fini ou dénombrable) une bijection de $\{0, .., card(D) - 1\}$ dans D (ou de $\mathbb{N}$ dans D si D est infini). Autrement dit : appeler successivement f sur les entiers entre 0 et $card(D) - 1$ énumérera tous les éléments de D.

En OCaml, nos fonctions d'énumération lèveront l'exception `Non_defini` si elles sont appelées sur des valeurs négatives ou supérieures au cardinal de l'ensemble.

Les fonctions d'énumération peuvent servir à représenter des ensembles. Nous nous en servirons également pour connaître l'ensemble de définition des fonctions caractéristiques.

1. Définir un type `'a enum` pour les fonctions d'énumération d'éléments de type `'a`.
2. Définir la fonction d'énumération pour l'ensemble des entiers positifs ou nuls pairs.
3. Définir une fonction

   ```
   list_of_set : int -> 'a enum -> 'a ens -> 'a list
   ```

   telle que `list_of_set max enum e` renvoie la liste des éléments énumérés par `enum` qui sont dans `e`, en se limitant aux `max` premiers.
   (où `'a ens` est défini dans la section 2.4).

2.6 Itérateurs

Dans cette partie, on demande d'écrire des itérateurs (qui pourront boucler indéfiniment si les ensembles sont infinis).

1. Quel serait le type d'un itérateur similaire à `List.iter` qui fonctionne sur les ensembles représentés par leur fonction d'énumération ? Implémentez-le. Est-il récursif terminal ?
2. Quel serait le type d'un itérateur similaire à `List.map` qui fonctionne sur les ensembles représentés par leur fonction d'énumération ? Implémentez-le.
3. Quel serait le type d'un itérateur similaire à `List.fold_left` ou `List.fold_right` qui fonctionne sur les ensembles représentés par leur fonction d'énumération ? Implémentez-le. Est-il récursif terminal ?
4. Quel serait le type d'un itérateur similaire à `List.iter` qui fonctionne sur les ensembles représentés par leur fonction caractéristique et une fonction d'énumération de l'ensemble de définition de la fonction caractéristique ? (Autrement dit on itère sur tous les éléments énumérés qui sont dans l'ensemble défini par la fonction caractéristique). Implémentez-le. Est-il récursif terminal ?
5. Quel serait le type d'un itérateur similaire à `List.fold_left` ou `List.fold_right` qui fonctionne sur les ensembles représentés par leur fonction caractéristique et la fonction d'énumération associée ? Implémentez-le. Est-il récursif terminal ?
6. Quel serait le type d'un itérateur similaire à `List.map` qui fonctionne sur les ensembles représentés par leur fonction caractéristique et la fonction d'énumération associée ? Quel(s) problème(s) pose l'implémentation ? Avez-vous une idée de solution ?