Programmation fonctionnelle

Examen du 15 janvier 2021
L2/L3 Sciences – Université Nice Sophia Antipolis

Durée: 2 heures.

- Aucun document ni aucune machine ne sont autorisés. Un mémo est donné en fin de sujet.
- Les téléphones doivent être rangés.
- Les réponses sont à reporter sur les feuilles de réponse en fin de sujet. Ces feuilles sont à dégrafer du sujet et à rendre en fin d'épreuve sans agrafe.
- Les réponses doivent être écrites lisiblement. Le correcteur blanc et l'effaceur sont autorisés, ils peuvent être utilisés pour décocher une case cochée par erreur, mais dans ce cas, n'essayez pas de redessiner la case.
- Les questions sont indépendantes: une question peut être sautée, et une fonction `f` demandée à une question peut être utilisée pour définir une fonction `g` dans une question ultérieure même si la solution pour `f` n'a pas été trouvée.

Réseau social (5 points)

On représente un réseau social par une liste `l:(string * string list) list`. Chaque couple `(s,[s1;s2;...;sn])` de la liste représente un membre du réseau (`s`) ainsi que la liste de ses followers. Par exemple, dans le réseau social ci-dessous

```
[("ann",["bob";"chris"]);("bob",["ann"]);("chris",[])]
```

il y a trois membres (ann, bob, chris), les followers de ann sont bob et chris, ann est la seule follower de bob, et chris n'a pas de follower.

Pour cet exercice il est recommandé d'utiliser au maximum des fonctions d'ordre supérieur sur les listes (filter, map, fold_left, etc).

Question 1 Définissez la fonction `following l s` qui renvoie la liste des membres du réseau suivi par s. Par exemple, pour le réseau social ci-dessus, `following l "ann"` renvoie `["bob"]`.

Question 2 Définissez la fonction `most_popular l` qui renvoie le nom d'un membre qui a le plus grand nombre de followers (la fonction renvoie une exception avec `invalid_arg` si l est vide). Par exemple, pour le réseau social ci-dessus, la fonction renvoie `"ann"`.

Question 3 Définissez la fonction `rate:('a->bool)->'a list->float` telle que `rate f l` renvoie la proportion d'éléments de l pour lesquels f renvoie `true`. Par exemple,

```
rate (fun x -> x mod 2 = 0) [1;2;3;4]
```

renvoie `0.5`.

Question 4 Définissez la fonction `recommandations l s` qui renvoie la liste des membres du réseau social `l` que l'on peut recommander au membre `s` de suivre. On recommande à `s` de suivre `s'` si `s`≠`s'`, `s` ne suit pas déjà `s'`, et plus de 10% des followers de `s` sont aussi des followers de `s'`.

Table associative (3 points)

On considère la signature ci-dessous pour un module de table associative (les "dictionnaires" de Python).

```
module type DICT = sig
  type ('a,'b) dict
  val empty :  ('a,'b) dict (* {} en Python *)
  val get :  ('a,'b) dict -> 'a -> 'b (* D[k] en Python *)
  val set :  ('a,'b) dict -> 'a -> 'b -> ('a,'b) dict (* D[k] = v en Python *)
  val len :  ('a,'b) dict -> int (* len(D) en Python *)
end
```

On se propose d'implémenter un `('a,'b) dict` par une liste associative OCaml. Par exemple, la table associative qui serait le dictionnaire `{1:'a', 2:'b'}` en Python est implémentée par la liste `[(1,'a');(2,'b')]`.

Question 5 Complétez le code ci-dessous. La fonction `get` lèvera l'exception `Not_found` lorsque la clé n'est pas dans la liste. La fonction `set` a deux comportements possible (comme en Python) : soit on ajoute une nouveau couple clé-valeur, soit on met à jour la valeur associée à une clé. La longueur est le nombre de clés.

```
module Dict :  DICT = struct
  type ('a,'b) dict = ('a * 'b) list
  ...
end
```

Générateurs et listes paresseuses (4 points)

On rappelle qu'un générateur est une fonction sans argument qui renvoie une valeur potentiellement différente à chaque appel. Par exemple, le générateur de la suite des entiers est

```
let entiers =
  let n = ref 0 in
  fun () -> n:=!n+1; !n-1
```

Question 6 Définissez le générateur de la suite de Fibonacci (1,1,2,3,5,8,...)

On rappelle la définition des listes paresseuses (flot) vue en cours de même que la liste paresseuse des entiers

```
type 'a item = Item of 'a * 'a flot
and 'a flot = 'a item Lazy.t

let rec entiers_depuis n = lazy(Item(n, entiers_depuis (n+1)))
let entiers = entiers_depuis 0
```

Question 7 Définissez la liste paresseuse de Fibonacci.

Question 8 Définissez la fonction `gen_to_flot` telle que `gen_to_flot g` est la liste paresseuse dont la suite de valeurs est celle renvoyée par le générateur g.

Question 9 Définissez la fonction `flot_to_gen` inverse: `flot_to_gen s` renvoie un générateur qui renvoie successivement les différentes valeurs du flot `s`.

Balise HTLM IMG (4 points)

On modélise une balise IMG avec ses attributs (simplifiés) par le type de données suivant:

```
type widthinfo = Pixels of int | Percents of int
type img = {
   src :  string;
   alt :  string option;
   width :  widthinfo option
}
```

Ce type de données permet de représenter des balises telles que

```
<img src="images/toto.jpg" alt="une photo de toto" width="200px">
<img src="images/toto.jpg" width="80%">
```

Question 10 Écrivez une fonction `img_to_string: img -> string` qui renvoie le code HTML correspondant aux informations de la balise IMG.

Question 11 Écrivez une fonction `img` avec les labels `src` (obligatoire), `alt` (optionnel), `widthpx` et `widthprct` (aussi optionnels) qui renvoie un enregistrement de type `img` initialisé en fonction des paramètres passés dans les labels. La fonction prend comme dernier argument `()` pour distinguer une application partielle d'une omission de paramètre optionnel. Exemples:

```
img ~src:"images/toto.jpg" ~alt:"une photo de toto" ~widthpx:200 ()
img ~src:"images/toto.jpg" ~widthprct:80 ()
```

La fonction génèrera une erreur `invalid_arg` si les deux labels `widthpx` et `widthprct` sont définis.

Try Finally (4 points)

Question 12 Définissez une fonction `try_finally : (unit -> 'a) -> (unit -> unit) -> 'a` qui prend en argument deux glaçons g1 et g2, et qui dégele le glaçon g1 puis le glaçon g2 et renvoie le résultat de g1. Le dégel de g2 a lieu même si une exception est levée par g1. Si g2 lève une exception, elle ne doit pas relancer g2.

Exemples:

- `try_finally (fun () -> 2+2) (fun () -> print_string "g2 dégelé")` affiche `g2 dégelé` et renvoie 4
- `try_finally (fun () -> 1/0) (fun () -> print_string "g2 dégelé")` affiche `g2 dégelé` et lève l'exception `Division_by_zero`.
- `try_finally (fun () -> print_string "g1 dégelé") (fun () -> print_string "g2 dégelé")` affiche `g1 dégelé` puis `g2 dégelé` et renvoie ()
- `try_finally (fun () -> print_string "g1 dégelé") (fun () -> print_string "g2 dégelé"; raise Not_found)` affiche `g1 dégelé` puis `g2 dégelé` (une seule fois) et lève l'exception `Not_found`.

Mémo sur les listes et les couples

- `fst c` : renvoie le premier élément du couple c
  exemple:`fst (1,2)` renvoie `1`
- `snd c` : renvoie le deuxième élément du couple c
  exemple: `fst (1,2)` renvoie `2`
  `List.mem x l` : renvoie `true` si x apparait dans l
  exemple: `List.mem 1 [1;2;3]` renvoie `true`
- `List.filter f l` : renvoie la liste des éléments `x` de `l` pour lesquels `f x = true`
  exemple: `List.filter (fun x -> x>0) [-1;2;0]` renvoie `[2]`
- `List.map f l` : renvoie la liste des images de f
  exemple: `List.map (fun x -> x+2) [1;2;3]` renvoie `[3;4;5]`
- `List.fold_left f a0 [a1;a2;...;an]` : renvoie `(f ... (f a0 a1)... an)`
  exemple: `List.fold_left (+) 0 [1;2;3;4]` renvoie `10`
- `List.assoc x l` : renvoie la valeur associée à x dans la liste associative l, lève l'exception `Not_found` si `x` n'est pas une clé valide
  exemple: `List.assoc 1 [(1,2);(3,4)]` renvoie `2`
- `List.remove_assoc x l` : renvoie `l` sans le couple clé-valeur associé à la clé `x`, ou `l` sans modification si `x` n'est pas une clé valide
  exemple : `List.remove_assoc 1 [(1,2);(3,4)]` renvoie `[(3,4)]`

Exemple avec des paramètres optionnels et des chaînes de format

```
let date ~day ~month ?year () = match year with
| None -> Format.sprintf "%d/%d" day month
| Some yy -> Format.sprintf "%d/%d/%d" day month yy

let _ = date ~day:14 ~month:7 () (*renvoie "14/7"*)
let _ = date ~day:14 ~month:7 ~year:1789 () (*renvoie "14/7/1789"*)
```