Université Paris Diderot
L2 Math/Info, L3 Info, M1 Linguistique

Programmation Fonctionnelle
Année 2013-2014

Examen 2ème session

Lundi 16 juin 2014

Tout document papier est autorisé. Les ordinateurs, les téléphones portables, comme tout autre moyen de communication vers l'extérieur, doivent être éteints et rangés.

Les exercices doivent être rédigés en fonctionnel pur : ni références, ni tableaux, ni boucles `for` ou `while`, pas d'enregistrements à champs mutables. Chaque fonction ci-dessous peut utiliser les fonctions prédéfinies (sauf indication contraire), et/ou les fonctions des questions précédentes.

Le temps à disposition est de 3 heures. Cet énoncé a 4 pages.

Exercice 1. Qu'affiche-t-il quand on entre l'une après l'autre les expressions suivantes dans le toplevel Caml ? Donner le type de chaque expression ou définition (ou le message d'erreur si l'expression est mal typée), ainsi que sa valeur et son effet de bord (ou l'exception levée) le cas échéant. Justifier si nécessaire.

1. `41 + 3 - 2 ;;`
2. `let a = 1.0 in let b = 2.0 in a <> b ;;`
3. `let a = 1.0 and b = 2.0 in let c = false in (a = b) = c ;;`
4. `let rec f a b = if b = 0 then a else f b (a mod b) ;;`
5. `f 12 9 ;;`
6. `let add1 = (+.) 1.0 ;;`
7. `let mult2 = ( *.) 2.0 ;;`
8. `let comp f g x = f (g x) ;;`
9. `comp add1 mult2 3.3 ;;`
10. `comp mult2 add1 3.3 ;;`
11. `comp (fun x -> sqrt x) add1 (4. *. 2.) ;;`
12. `let id x = x in (id 2) ^ "bar" ;;`
13. `let id x = x in id (fun x y -> x y) ;;` *(Bonus)*

Exercice 2. Dans les questions qui suivent, toutes les fonctions prédéfinies sur les listes (`List.mem`, `List.filter`, ...) sont librement utilisables, en particulier si elles améliorent l'efficacité et/ou la lisibilité du code.

1. En guise d'échauffement, écrire une fonction `supprime : 'a -> 'a list -> 'a list` telle que `supprime x l` supprime toutes les occurrences de l'élément `x` dans la liste `l`.
2. Rappelons que `List.assoc x [(x1, v1); ...; (xn, vn)]` renvoie le premier v_k tel que x_k = `x` si ce x_k existe, et déclenche une exception sinon. La liste de couples est souvent appelée *liste d'association*, et la valeur de retour, la *valeur associée à* `x` dans cette liste.

   Noter que la valeur associée à un `x` dans une liste d'association `l` ne dépend que du *premier* couple de membre gauche `x`. La liste `l` sera dite *optimale* si, pour chaque membre gauche `x`, elle ne contient qu'un seul couple de membre gauche `x`.

   Écrire une fonction `optimisation : ('a * 'b) list -> ('a * 'b) list`. Appliquée à une liste d'association `l`, cette fonction doit renvoyer une liste d'association optimale de mêmes membres gauches, et associant à ses membres gauches les même valeurs. Exemple :

   `optimisation [(2,'a');(3,'b');(5,'a');(3,'c')] = [(2,'a');(3,'b');(5,'a')].`

3. Appelons *domaine* d'une liste d'association la liste sans répétitions des membres gauches de ses couples, en ne gardant que le premier exemplaire de chaque membre gauche.

   Écrire une fonction `domaine : ('a * 'b) list -> 'a list` renvoyant, à l'ordre de ses éléments près, le domaine d'une liste d'association. Exemple :

   ```
   domaine [(2,'a');(3,'b');(5,'a');(3,'c')] = [2; 3; 5]
   ```

   ou éventuellement, si la méthode employée est différente :

   ```
   domaine [(2,'a');(3,'b');(5,'a');(3,'c')] = [2; 5; 3]
   ```

4. Appelons *image* d'une liste d'association la liste sans répétitions de valeurs associées aux éléments de son domaine, en ne gardant que le premier exemplaire de chaque valeur.

   Écrire une fonction `image : ('a * 'b) list -> 'b list` renvoyant, à l'ordre de ses éléments près, l'image d'une liste d'association. Exemple :

   ```
   image [(2,'a');(3,'b');(5,'a'); (3;'c')] = ['a'; 'b'].
   ```

5. Etant donné un `v` quelconque et une liste d'association `l`, appelons *image inverse de* `v` *dans* `l` la liste sans répétitions (éventuellement vide, et considérée à l'ordre de ses éléments près) des `x` associés à `v` dans `l`.

   Écrire `image_inverse : 'b -> ('a * 'b) list -> 'a list` telle que `image_inverse v l` renvoie l'image inverse de `v` dans `l`. Exemple :

   ```
   image_inverse 'a' [(2,'a');(3,'b');(5,'a'); (3;'c')] = [2; 5]
   image_inverse 'c' [(2,'a');(3,'b');(5,'a'); (3;'c')] = []
   image_inverse 'd' [(2,'a');(3,'b');(5,'a'); (3;'c')] = []
   ```

Exercice 3. Le but de cet exercice est d'écrire un formateur de texte qui permet de mettre en page un paragraphe de texte pour qu'il soit aligné avec les marges. Par exemple, à partir de l'entrée suivante :

```
Quo usque tandem abutere, Catilina, patientia nostra? Quam diu etiam
furor iste tuus nos eludet?  Quem ad finem sese effrenata jactabit
audacia?
```

votre programme sera capable de scinder le document en lignes et de varier la taille des espaces pour produire la sortie suivante :

> Quo usque tandem abutere, Catilina, patientia nostra ? Quam diu etiam furor iste tuus nos eludet ? Quem ad finem sese effrenata jactabit audacia ?

Les questions sont indépendantes — si vous ne savez pas traiter une question, n'hésitez pas à passer à la suite en vous servant des fonctions que vous n'avez pas su écrire.

On suppose donnée [1] une fonction

```
charwidth : char -> float
```

qui retourne la largeur d'un caractère lorsqu'il sera mis en page, en millimètres. On suppose aussi donnée la structure de données suivante :

```
type box = Word of string | Space of float | Aside of box * box
```

qui représente une suite de caractères mis en page. `Word w` représente un mot constitué de la chaîne `w`, `Space s` un espace d'une largeur de `s` millimètres, et enfin `Aside(b1,b2)` représente les boîtes `b1` et `b2` mises en page à la suite. Par exemple, la mise en page de ligne de texte *Quo usque tandem* pourra être

```
Aside (Aside (Word "Quo",
              Aside (Space 1., Word "usque")),
       Aside (Space 1., Word "tandem"))
```

1. Vous n'avez pas à l'écrire, elle est déjà écrite.

1. Écrivez une fonction `stringwidth : string -> float` qui retourne la somme des `charwidth` des caractères contenus dans son paramètre. Par exemple, `stringwidth "toto"` retournera la valeur de

   `(charwidth 't') + (charwidth 'o') + (charwidth 't') + (charwidth 'o')`.

   On rappelle que les fonctions

   ```
   String.length : string -> int
   String.get : string -> int -> char
   ```

   retournent, respectivement, la longueur d'une chaîne et le i-ème caractère d'une chaîne (en comptant à partir de 0).

2. Écrivez une fonction `string_of_box : box -> string` qui retournera la représentation sous forme de chaîne de son paramètre. Pour un `Word w`, elle retournera simplement `w`, pour un `Space`, un espace, et pour un `Aside`, la concaténation des représentations des deux paramètres (sans espace intervenant). On rappelle que l'opérateur « `^` » permet de concaténer deux chaînes (les mettre bout-à-bout).

3. Écrivez une fonction `boxwidth : box -> float` qui retourne la largeur totale d'une boîte, c'est-à-dire la somme des largeurs des mots et des espaces qui la composent.

4. Écrivez une fonction `box_of_words : string list -> float -> box list` qui prend comme paramètres :
   - une liste de mots `words`,
   - la largeur d'un espace `spacewidth`,

   et qui retourne une boîte consistant des mots contenus dans `words` séparés par des espaces de largeur `spacewidth`. Elle retournera un espace invisible `Space 0.` si `words` est une liste vide.

   Par exemple,

   ```
   box_of_words ["Quo"; "usque"; "tandem"] 1.0
   ```

   returnera la boîte donnée en exemple ci-dessus, ou une boîte équivalente.

5. Écrivez une fonction `count_spaces : box -> int` qui retourne le nombre de `Space` contenus dans une boîte, même à l'intérieur de plusieurs `Aside`.

6. Écrivez une fonction `addspace : box -> float -> box` qui ajoute une valeur à tous les `Space` contenus à l'intérieur d'une boîte, même à l'intérieur de plusieurs `Aside`.

7. Écrivez une fonction `fill_line : box -> float -> box` qui ajuste une ligne en ajoutant de façon uniforme de l'espacement entre les mots pour que sa largeur soit égale à celle qui a été passée en paramètre. Que se passe-t-il s'il n'y a qu'un seul mot sur une ligne ?

   Indication : déterminez la largeur de la boîte à l'aide de la fonction `boxwidth`, calculez combien il manque d'espacement, déterminez le nombre d'espaces à l'aide de `count_spaces`, déterminez combien il faut ajouter d'espacement à chaque espace pour obtenir le bon total, puis passez le résultat à `addspace`.

8. Écrivez une fonction `fill_paragraph : box list -> float -> box list` qui formate un paragraphe (une liste de boîtes représentant chacune une ligne) en appliquant `fill_line` à toutes les lignes du paragraphe sauf la dernière.

9. Écrivez une fonction `split_lines : string list -> float -> float -> box list` qui prend comme paramètres :
   - une liste de mots `words`,
   - la largeur maximale d'une ligne `linewidth`,
   - la largeur d'un espace `spacewidth`

   et retourne un paragraphe qui consiste des mots de `words` séparés par des espaces de taille `spacewidth` et dont chaque ligne a une largeur d'au plus `linewidth`. Que se passe-t-il si un mot est plus long que `linewidth` à lui tout seul ?

10. Proposez une fonction `format_paragraph` qui combine les fonctions `split_lines` et `fill_paragraph`, et donnez son type.

Exercice 4. Dans cet exercice, vous allez implémenter des termes et des substitutions. Un terme est soit une variable (dont le nom est une chaîne quelconque), soit la constante A, soit le constructeur F appliqué à deux termes. Un terme comme $F(F(A,x),F(y,z))$ sera representé en OCaml par l'expression

$$t = \texttt{F(F(A, Var "x"), F(Var "y", Var "z"))}$$

1. Définir en OCaml un type `term` pour représenter les termes, en suivant l'exemple.
2. Définir en OCaml une fonction `contains: term -> string -> bool` qui prend un terme et le nom d'une variable en argument, et qui teste si la variable parait dans le terme. Par exemple, `(contains t "z")` doit envoyer `true`, et `(contains t "w")` doit envoyer `false`.
3. Une substitution est une fonction partielle qui associe des termes à des variables. On reprensente en OCaml une substitution par une liste d'association qui aux noms de variables (des chaînes) associe des termes. Par exemple,

   $$s = \texttt{[("x",A); ("y", F(A,A))]}$$

   est une substitution ; elle associe le term `A` à la variable `x`, et le terme `F(A,A)` à la variable `y`. Définir en OCaml un type `substitution` pour les substitutions, en suivant cet exemple.
4. Le domaine d'une substitution est l'ensemble des variables auxquelles la substitution affecte un terme. Écrire une fonction `domain: substitution -> string list` qui donne le domaine d'une substitution. Par exemple, `domain [("x",A); ("y", F(A,A))]` doit donner le résultat `["x"; "y"]`.
5. L'application d'une substitution s à un terme t donne un nouvau terme qui est une copie de t, sauf que chaque variable x du terme t est remplacée par le terme associé par s à x quand un tel terme existe.
   Écrire une fonction `apply: substitution -> term -> term` qui prend une substitution et un terme en argument, et qui renvoie le résultat de l'application de la substitution au terme. Par exemple, `apply s t` doit envoyer

   ```
   F(F(A,A), F(F(A,A), Var "z"))
   ```

6. La composition de deux substitutions s_1 et s_2 est la substitution dont l'application à un terme a le même effet que d'y appliquer d'abord s_1, puis s_2. Elle est obtenue en créant une copie de s_1 dans laquelle on a appliqué s_2 à tous les termes de l'image de s_1, et en concatenant le résultat avec la partie de s_2 qui n'est pas contenue dans le domaine de s_1.
   Écrire la fonction `composition: substitution -> substitution -> substitution`. Par exemple,
   `composition [("x1",F(Var "y1",Var "y1"));("x2",F(A,A))] [("x1",A);("y1",A)]`
   doit donner le résultat : `[ ("x1", F(A,A); ("x2", F(A,A)); ("y1", "A") ]`