SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

MPIL(LU3IN008)
Année 2019/2020
Séance 2 de TD/TME

Objectifs

1. Programmation fonctionnelle.
2. Filtrage par motifs.
3. Listes.
4. Définition de types somme.

Travaux Dirigés

Comme pour la séance précédente, et toutes les autres, n'oubliez pas de donner le type de toutes les fonctions que vous écrirez. Dans la mesure du possible, ces fonctions seront écrites dans leur version récursive terminale.

Exercice 1 : Listes

Q.1.1 Implémenter la fonction `zip` qui prend en paramètre deux listes `l1`, `l2` et retourne une liste de couples tel que pour des listes en entrée `[x1; x2; ...; xn]` et `[y1; y2; ...; yn]` retourne la liste `[(x1, y1); (x2, y2); ...; (xn, yn)]`. Dans le cas où une des deux listes est plus courte que l'autre, la liste résultante aura la taille de la plus petite et les données en trop seront ignorées.

Exercice 2 : Filtrage

Q.2.1 (Quiz) Dites si les codes suivant compilent et s'ils se comportent comme ce que leur auteur a voulu écrire. Si non, corrigez-les.

1. ```
   let egale_trois = function
       3 -> true
     | _ -> false
   ```

2. ```
   let egale_ixe x = function
       n -> true
     | _ -> false
   ```

3. ```
   let double_zero n m =
     match n m with
       | 0 0 -> true
       | _ -> false
   ```

4. ```
   let rec sorted = function
       [] -> true
     | x :: y :: q -> x < y && sorted q
   ```

5. ```
   let rec intercale l sep =
     match l with
       | s :: l -> s ^ sep ^ intercale l sep
       | s :: [] -> s
       | [] -> ""
   ```

Exercice 3 : Sommes et filtrage

Q.3.1 À la petite école, vous avez vu la représentation dite *unaire* des nombres entiers positifs : pour un nombre entier positif n donné, on le représente en posant exactement n bâtons sur la table. Les entiers de *Peano* formalisent cette représentation intuitive des nombres entiers : un nombre n est soit zero (aucun bâton sur la table) soit le successeur d'un autre nombre n' (on ajoute un bâton aux bâtons nécessaires pour représenter n').

Définissez en OCaml le type `peano` qui correspond à cette représentation des entiers positifs.

Q.3.2 Écrivez les fonctions de conversion vers les entiers machine prédéfinis dans OCaml : `peano_of_int` et `int_of_peano`.

Q.3.3 Donnez une méthode pour tester que la conversion est valide jusqu'à un certain rang et écrivez une fonction implémentant cette méthode.

Q.3.4 Écrivez le prédicat `even` décidant si un nombre en représentation de Peano est pair, sans passer par la conversion. **Rappel :** le filtrage peut parcourir en profondeur les valeurs.

Q.3.5 Écrivez une fonction de test vérifiant que le prédicat est valide, de la même façon que la conversion.

Q.3.6 Déduisez une manière de tester qu'une fonction à un argument agissant sur les entiers de Peano `(peano -> 'a)` se comporte de la même façon que sa version travaillant sur les entiers OCaml. Écrivez une telle fonction de test générique et donnez son type.

Q.3.7 Écrivez les fonction `half` (qui calcule le résultat de la division par deux) et `twice` (qui calcule le résultat de la multiplication par deux). L'utilisation de la conversion et de l'égalité structurelle est interdite.

Q.3.8 Écrivez les appels à la fonction de test générique pour `half`, `twice`, `int_of_peano` et `even`.

Exercice 4 : Tri par insertion

On veut réaliser une fonction de tri par insertion pour les listes.

Q.4.1 Écrire une fonction `ins` prenant un entier `x` et une liste d'entiers `l` triée en paramètre et insérant `x` dans `l` de manière telle que la liste obtenue soit triée.

Q.4.2 En déduire la fonction de tri par insertion `sort`.

Q.4.3 Quelle est la complexité de ce tri ?

Exercice 5 : Tri fusion (Bonus)

On veut maintenant réaliser une fonction de tri plus efficace mais tout aussi simple à implémenter que le tri par insertion : le tri fusion.

Q.5.1 Écrire une fonction `split` prenant une liste d'entiers `l` et qui retourne cette liste coupée en deux (i.e. un couple de liste d'entier).

Q.5.2 Écrire une fonction `merge` prenant deux listes d'entiers `l1` et `l2` supposées triées, et retourne la liste triée issue de la fusion de ces deux listes.

Q.5.3 Écrire la fonction `sort` de tri par fusion (veiller à bien traiter les cas de base en premier).

Q.5.4 Quelle est la complexité de ce tri ?
   ```

Travaux sur Machines Encadrés

Exercice 6 : Listes

On veut créer une fonction générant, pour un entier $k > 0$, la suite de k lignes suivantes :

```
liste 1: [1]
liste 2: [1;1]
liste 3: [2;1]
liste 4: [1;2;1;1]
liste 5: [1;1;1;2;2;1]
liste 6: [3;1;2;2;1;1]
liste 7: [1;3;1;1;2;2;2;1]
```

On obtient la liste `k` en lisant le contenu de la liste $k-1$.

Q.6.1 Écrire une fonction `calcul_prefixe` qui prend une liste `l` ayant la forme des listes mentionnées ci-dessus, et qui renvoie 0 si `l` est vide ou le nombre de chiffres identiques au début de la liste `l`. Par exemple, l'application de `calc_prefixe` sur la liste 5 ci-dessus renvoie 3.

Q.6.2 Déduire des listes mentionnées et de la question précédente la fonction `genere_liste` qui prend en paramètre la liste $k-1$ et qui renvoie la liste k.

Q.6.3 Écrire une fonction `genere` qui prend en argument un entier $k > 0$ et qui renvoie une liste contenant les k premières listes de la suite.

Exercice 7 : Jeux de Belote

Q.7.1 Définir un type `couleur` pour représenter les couleurs d'un jeu de cartes.

Q.7.2 Définir un type `carte` pour représenter les cartes d'un jeu de belote : as, roi, dame, valet et petites cartes.

Q.7.3 Écrire une fonction `valeur` qui prend la couleur de l'atout et une carte et renvoie la valeur de la carte. On rappelle les règles de comptage suivantes : l'as vaut 11, le roi 4, la dame 3, le valet 20 s'il a la couleur de l'atout, 2 sinon, le 10 vaut 10, le 9 vaut 14 s'il a la couleur de l'atout, 0 sinon, les autres cartes valent 0.

Q.7.4 Écrire une fonction `valeur_jeu` qui prend en paramètre la couleur de l'atout et une liste de carte, et renvoie la valeur du jeu.

Q.7.5 Tester la fonction précédente sur le jeu suivant : valet de carreau, 10 de trèfle, 7 de cœur, 8 de carreau, 9 de pique avec atout à carreau (on doit trouver 30) .

Exercice 8 : Fil d'actualité d'un réseau social

Notre but dans cet exercice est de modéliser le filtrage des publications visibles en fil d'actualité d'un réseau social en fonction de l'utilisateur connecté.

Les éléments manipulés seront les suivants :

1. les utilisateurs définis par un nom et une liste de noms d'amis
2. les publications définies par le nom de l'auteur, un message et la cible
3. les cibles de publications qui peuvent être soit : *Tout le monde*, *Amis* ou *une liste d'amis spécifiques*.

Q.8.1 Définir les types et fonctions permettant de construire les éléments cités ci-dessus en utilisant les signatures suivantes :

```
type utilisateur = ...
type cible = ...
type publication = ...

val utilisateur :
  nom :string -> amis :string list -> utilisateur
val publication :
  ?cible :cible -> auteur :string -> string ->
  publication
```

Rappel : `nom`, `amis` et `auteur` sont des labels. `cible` est un paramètre optionel (la valeur par défaut sera *Amis*).

Q.8.2 Écrire la fonction `afficher_publication` qui prend en paramètre une `publication` et l'affiche sous la forme `auteur: message`.

Q.8.3 Écrire la fonction `acces_autorise` qui prend en paramètre un `utilisateur` et une `publication`, et retourne un booléen indiquant si l'utilisateur donnée

Q.8.4 Écrire la fonction `filtre_publications` qui prend en paramètre un `utilisateur` une liste de `publicatios` et retourne seulement les publications autorisées à être vues.

Q.8.5 Écrire la fonction `categoriser` qui prend en paramètre une liste de `publications` et retourne un couple qui contient à gauche toutes les publications privées (pour amis) et à droite les publications publiques.

Q.8.6 Définir 3 (ou plus) utilisateurs et une liste de 6 (ou plus) publications afin de tester les deux fonctions précédentes. Veiller à utiliser tous les types de cibles dans les publications et chercher à définir les utilisateurs tels que la liste publications visibles soit différente pour chacun d'eux.

Exercice 9 : Manipuler des chaînes de caractères

On peut concaténer deux chaînes de caractères avec l'opérateur ^ directement mais la plupart des fonctions disponibles permettant de manipuler des chaînes de caractères se trouvent dans le module `String` (documentation à l'adresse : http ://caml.inria.fr/pub/docs/manual-ocaml/libref/String.html).

Consulter la description des fonctions se trouvant dans le module `String` dans le manuel de référence.

Q.9.1 Écrire une fonction `cherche_car` qui cherche si un caractère `car` apparaît dans une chaîne de caractères `str` passée en paramètre : si tel est le cas, elle renvoie la position de la première occurence de `car` dans `str`, sinon elle renvoie -1. Vous n'utiliserez que `String.length` et `String.get`

Q.9.2 Étant donnée une chaîne `str`, on veut modifier `str` en ajoutant un espace derrière chaque point et chaque virgule apparaissant dans `str`. Écrire une fonction OCaml qui remplit cette tâche.