Université Paris Diderot
Math-Info/Informatique/Linguistique

Programmation Fonctionnelle
Année 2012-2013

TP n°2

Listes

Les expressions suivantes sont des listes :

- liste vide : `[]`
- liste d'entiers à 3 éléments : `[42; 37; 15]`
- liste de listes d'entiers : `[[0; 1]; [2; 2; 1]; []]`

Les éléments d'une liste doivent être de même type. L'opérateur `::` permet d'ajouter un élément en tête d'une liste. L'opérateur `@` permet de concaténer deux listes. Les listes suivantes sont par exemple égales :

```
[42;37;15]                  42::[37;15]              42::(37::[15])
42::(37::(15::[]))          42::37::15::[]           42::37::[15]
                            [42;37]@[15]
```

La construction suivante permet de définir une fonction en raisonnant par cas sur la forme d'une liste :

```
let rec f l = match l with
```

3. ∘ `map` : la fonction telle que `map f [a1;...;an] = [(f a1);...;(f an)]`.
   Exemple : `map (fun x -> x + 1) [1;2;3] = [2;3;4]`.
4. `liste_min` : calcule le minimum d'une liste d'entiers non vide.
   Exemple : `liste_min [-30;2;549] = -30`.
5. ∘ `append` : concatène deux listes – cette fonction ne doit pas utiliser l'opérateur `@`.
   Exemple : `append [1;2;3] [4;5] = [1;2;3;4;5]`.
6. ∘ `rev` : inverse l'ordre des éléments d'une liste.
   Exemple : `rev [1;2;3] = [3;2;1]`.
7. ∘ `flatten` : aplanir d'un niveau une liste de listes.
   Exemple : `flatten [[2];[];[3;4;5]] = [2;3;4;5]`.
8. `is_sorted` : détermine si une liste est triée.
   Exemple : `is_sorted [1;3;5;6;4] = false`.
9. Question subsidiaire : `moyenne`, qui calcule la moyenne des éléments d'une liste (potentiellement vide) d'entiers non vide. Exemple : `moyenne [1;2;3] = 2`.

Exercice 2 Comme indiqué ci-dessus, la fonction `map` de l'exercice précédent est prédéfinie : elle s'écrit `List.map`

1. Ecrire une fonctions `enum` telle que `enum n` renvoie la liste `[0;1;...;n]`.
2. A partir de cette fonction et de `List.map`, écrire une fonction `enum_droite` telle que `enum_droite i n` renvoie la liste `[(i,0);(i,1);...;(i,n)]`.
3. En déduire le code de `enum_paires : int -> int -> int list` telle que `enum_paires n p` renvoie la liste des éléments de $\{0,\ldots,\mathtt{n}\} \times \{0,\ldots,\mathtt{p}\}$ dans l'ordre lexicographique. Exemple :

   ```
   enum_paires 1 2 = [(0,0);(0,1);(0,2);(1,0);(1,1);(1,2)]
   ```

   Pour écrire cette fonction par récurrence sur son premier argument, vous pouvez vous poser les deux questions suivantes : que vaut `enum_paires n p` si `n = 0` ? en supposant `enum_paires (n - 1) p` déjà calculé, que manque-t-il pour calculer `enum_paires n p` ?
4. En utilisant cette fonction, écrire une fonction `table_mult n` calculant la table de multiplication de 0 à `n`, sous la forme d'une liste de triplets $(i, j, i \times j)$. Exemple :

   ```
   table_mult 2 = [(0,0,0);(0,1,0);(0,2,0);
                   (1,0,0);(1,1,1);(1,2,2);
                   (2,0,0);(2,1,2);(2,2,4)]
   ```

Exercice 3 On supposera que les fonctions suivantes attendent une liste dont les éléments sont d'un type muni d'un ordre total.

1. Ecrire une fonction `insert` qui insère un élément `x` à la bonne place dans une `'a list` supposée triée par ordre strictement croissant. Si `x` est déjà dans la liste, laissera la liste telle quelle.
2. En utilisant la fonction `insert`, écrire une fonction `sort` qui trie une `'a list` quelconque par ordre croissant, en fusionnant les doublons.
3. Ecrire une fonction `mem_sorted` se comportant comme `mem` sur une liste triée, mais s'évaluant en un nombre minimum d'étapes.
4. Ecrire les opérations d'union et d'intersection (`union_sorted, inter_sorted`) de deux listes triées.