Université Paris Diderot — Programmation Fonctionnelle
L2 Math-Info/L3 Informatique/M1 Linguistique — Année 2011-2012

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
| []    -> ...   (* expression associée si l est vide *)
| a::l' -> ...   (* expression associée si l est non vide.  *)
;;               (* a désigne le 1er élément de l, et l'    *)
                 (* la liste l privée de son 1er élément.    *)
```

La seconde ligne peut être interprétée par : *si* `l` *est de la forme : un certain élément* `a` *suivi d'une certaine liste* `l'`*, alors, en appelant effectivement* `a` *le premier élément de* `l` *et* `l'` *la liste qui suit ce premier élément, passer à l'évaluation de l'expression associé.* Autre exemple :

```
let rec f l = match l with
| []      -> ...   (* cas où l est vide                        *)
| [a]     -> ...   (* cas où l contient un seul élément        *)
| a::b::l'-> ...   (* cas où l contient au moins deux éléments *)
;;
```

Remarque. À l'évaluation, les différents cas d'un `match` sont examinés successivement. La suite des cas peut ne pas couvrir toutes les formes possibles : dans ce cas, la fonction définie est dite *partielle* : elle déclenche une exception si son argument n'est d'aucune des formes mentionnées.

Exercice 1 Écrire les fonctions suivantes sur les listes. Les fonctions précédées de ∘ sont prédéfinies en Caml (elles s'écrivent `List.map`, `List.mem`, etc) et le but est de retrouver leur implémentation.
- `list_sigma` : calcule la somme des éléments d'une liste d'entiers.
  Exemple : `list_sigma [1;2;3] = 6`.
- ∘ `mem` : déterminer si une liste contient une valeur donnée.
  Exemple : `mem 27 [12;27;1] = true`, `mem 3 [12;27;1] = false`.
- ∘ `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]`.

- `liste_min` : calcule le minimum d'une liste d'entiers non vide.
  Exemple : `liste_min [-30;2;549] = -30`.
- ◦ `append` : concatène deux listes – cette fonction ne doit pas utiliser l'opérateur `@`.
  Exemple : `append [1;2;3] [4;5] = [1;2;3;4;5]`.
- ◦ `rev` : inverse l'ordre des éléments d'une liste.
  Exemple : `rev [1;2;3] = [3;2;1]`.
- ◦ `flatten` : aplanir d'un niveau une liste de listes.
  Exemple : `flatten [[2];[];[3;4;5]] = [2;3;4;5]`.
- `is_sorted` : détermine si une liste est triée.
  Exemple : `is_sorted [1;3;5;6;4] = false`.
- 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 $\{0, \ldots, \texttt{n}\} \times \{0, \ldots, \texttt{p}\}$ 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.

- 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.
- Ecrire une fonction `sort` qui trie une `'a list` quelconque par ordre croissant, en fusionnant les doublons.
- Ecrire une fonction `mem_sorted` se comportant comme `mem` sur une liste triée, mais s'évaluant en un nombre minimum d'étapes.
- Ecrire les opérations d'union et d'intersection (`union_sorted, inter_sorted`) de deux listes triées.