Université Paris Diderot — Programmation Fonctionnelle
Math-Info/Informatique/Linguistique — Année 2012-2013

TP n°3

Listes et autres types structurés

1 Listes d'entiers

Les listes considérées dans ce premier exercice seront des listes d'entiers.

Exercice 1 Écrire les fonctions suivantes. Chaque fonction devra être indépendante des autres, et ne pas se servir de fonctions prédéfines.

- `fois : int -> int list -> int list`
  telle que `fois n l` renvoie la liste obtenue en multipliant par `n` chaque élément de `l`.
- `plus : int list -> int list -> int list`
  telle que `plus` $[x_1;...;x_n]$ $[y_1;...;y_n]$ renvoie la liste $[x_1 + y_1;...;x_n + y_n]$.
  Comment gérér le cas où les deux listes ne sont pas de même longueur ?
- `diff : int list -> int list -> int list`
  telle que `diff` $[x_1;...;x_n]$ $[y_1;...;y_n]$ renvoie la liste $[x_1 - y_1;...;x_n - y_n]$.
  Même question.

Exercice 2 Rappelons que la fonction prédéfinie `List.map` permet d'appliquer une fonction à chacun des éléments d'une liste :

`List.map f` $[x_1;...;x_n] = [(f\ x_1);...;(f\ x_n)]$

Écrire une nouvelle implémentation de la fonction `fois` utilisant la fonction `List.map`, et déléguant entièrement la récurrence à cette fonction.

Exercice 3 La fonction prédéfinie `List.map2` considère *deux* listes, impérativement de même longueur, et applique une fonction `f` aux éléments des deux listes de même rangs :

`List.map2 f` $[x_1;...;x_n]$ $[y_1;...;y_n] = [(f\ x_1\ y_1);...;(f\ x_n\ y_n)]$

D'autre part, les opérateurs arithmétiques de Caml peuvent être utilisés comme des fonctions ordinaires, avec la notation préfixe habituelle des fonctions : il suffit de les entourer de parenthèses (en ajoutant des espaces autour de `*` – pourquoi ?). En particulier, `(+)`, `(-)`, `( * )` ... peuvent être passés à `List.map2` comme des fonctions ordinaires.

```
(+) x x' = x + x'
(-) x x' = x - x'
( * ) x x' = x * x'
...
```

Écrire deux nouvelles implémentations des fonctions `plus` et `diff` utilisant la fonction prédéfinie `List.map2`, et les versions préfixes des opérateurs `+` et `-`.

2 Arbres binaires

On peut effectuer en Caml des *déclarations de types*, c'est-à-dire créer de nouveaux types de valeurs. Les arbres binaires étiqueés par des entiers peuvent par exemple être représentés en Caml à l'aide de la déclaration de type suivante :

```
type tree = Nil | Node of int * tree * tree;;
```

`Nil` et `Node` seront appelés les *constructeurs* du type `tree`. Une fois ce type déclaré, on peut construire des valeurs de type `tree` :

```
Nil;;
Node (42, Nil, Nil);;
Node (42, Node(10, Nil, Nil), Nil);;
```

Comme pour les listes, une fonction peut être définie par cas sur la forme d'un arbre :

```
let rec f a = match a with
  Nil       -> ...
| Node(n,g,d) -> ...
;;
```

Exercice 4 Écrire les fonction suivantes :

- `taille : tree -> int` renvoyant la taille d'un arbre, c'est-à-dire son nombre de noeuds internes.
- `hauteur : tree -> int` renvoyant la hauteur d'un arbre, c'est-à-dire la longueur de sa plus longue branche.
- `detect : tree -> int -> bool`, telle que `detect a n` renvoie `true` si et seulement si l'un des nœuds de l'arbre est étiqueté par `n`.
- `complet : tree -> bool` déterminant si un arbre est complet, c'est-à-dire si toutes ses feuilles sont à la même profondeur,

Exercice 5 On rappelle qu'un *arbre binaire de recherche* est un arbre dans lequel pour *chaque* noeud, on a la propriété suivante :

- soit `n` l'étiquette de ce nœud :
  - toutes les étiquettes du fils gauche de ce nœud sont `< n`.
  - toutes les étiquettes du fils droit de ce nœud sont `> n`.

Écrire les fonctions suivantes :

1. `detect_abr : int -> tree -> bool` tel que si `a` est un ABR et `n` un entier, `detect_abr n a` renvoie `true` si `n` apparaît dans l'arbre, et `false` sinon.

   L'exploration de `a` devra être minimale, en tenant compte du fait qu'il s'agit d'un ABR.

2. `ajout_abr : int -> tree -> tree` tel que si `a` est un ABR et `n` est un entier, `ajout_abr` renvoie l'arbre `a` dans lequel on a ajouté `n` s'il n'y est pas déjà. Le résultat doit être encore un ABR.

   Si `n` est déjà dans l'arbre, cette fonction renvoie simplement `a`.

3. `est_abr : tree -> bool` prenant en argument un arbre quelconque, et déterminant s'il s'agit d'un ABR. A noter que cette fonction nécessite de déclarer plusieurs fonctions auxiliaires, qui pourront être déclarées en fonctions locales.