Université Paris 13 - Licence d'informatique Année 2009-2010

TP de *Programmation Fonctionnelle* n° 3

1 Arbres binaires

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

```
type btree = Nil | BNode of int * btree * btree;;
```

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

```
Nil;;
BNode (42, Nil, Nil);;
BNode (42, BNode(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          -> ...
| BNode(n,g,d) -> ...
;;
```

Exercice 1 Ecrire les fonctions suivantes :

- `taille : btree -> int` renvoyant la taille d'un arbre, c'est-à-dire son nombre de noeuds.
- `hauteur : btree -> int` renvoyant la hauteur d'un arbre, c'est-à-dire la longueur de sa plus longue branche.
- `detect : btree -> int -> bool`, telle que `detect t n` renvoie `true` si et seulement si l'un des noeuds de l'arbre `t` est étiqueté par `n`.
- `entier : btree -> bool` déterminant si un arbre est entier, c'est-à-dire si tous les nœuds possèdent zéro ou deux fils.

Exercice 2 On rappelle qu'un *arbre binaire de recherche* (ABR) est un arbre dans lequel pour *chaque* noeud, on a la propriété suivante :

- soit `n` l'étiquette de ce noeud :
  - toutes les étiquettes du sous-arbre gauche de ce noeud sont $<$ `n`.
  - toutes les étiquettes du sous-arbre droit de ce noeud sont $>$ `n`.

Ecrire les fonctions suivantes :

- `detect_abr : int -> btree -> btree` 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.
- `ajout_abr : int -> btree -> btree` 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`.
- `est_abr : btree -> bool` prenant en argument un arbre quelconque, et déterminant s'il s'agit d'un ABR. Noter que cette fonction nécessite de déclarer plusieurs fonctions auxiliaires mutuellement récursives, qui pourront être déclarées en fonctions locales.

Exercice 3 Un arbre binaire sera dit *équilibré* si et seulement si la différence entre les nombres des noeuds du sous-arbre gauche et du sous-arbre droit de tout noeud ne peut excéder 1. Ecrire la fonction suivante :

- `est_equil : btree -> bool`

prenant en argument un arbre quelconque, et déterminant s'il s'agit d'un arbre équilibré.

2 Arbres n-aires

Le type des arbres étiquetés par des entiers et ayant noeuds avec degré arbitraire (c'est-à-dire avec un nombre arbitraire des fils) peut s'obtenir comme une composition de types inductifs :

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

Par exemple, voici une instance de ce type :

```
let tree_ex = Node (2,
        [ Node (3, [ ]) ;
          Node (3, [ ]) ;
          Node (0,
                [ Node (1, [ ])]
                )
        ]
);;
```

Exercice 4 Ecrire les fonctions suivantes :

- `taille : tree -> int` renvoyant la taille d'un arbre, c'est-à-dire son nombre de noeuds.
- `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 t n` renvoie `true` si et seulement si l'un des noeuds de l'arbre `t` est étiqueté par `n`.
- `size_etiq : tree -> int`, renvoyant la somme des étiquettes des noeuds d'un arbre.