MIAS 1 et MASS 1 – Programmation fonctionnelle

TP n° 8-9 : corrigé

```
; ============================================
; 1)  Implantation du TDA arbres binaires
;     fondee sur les listes Scheme
; ============================================

; *** Fonctions génératrices ***

; arbre-vide : --> Arbre
;              |-> ø
(define arbre-vide
  (lambda () '()))


; cons-arbre : Entier, Arbre, Arbre --> Arbre
;                n   ,   g  ,   d   |-> <n, g, d>
(define cons-arbre
  (lambda (n g d)
    (list n g d)))


; *** Fonctions caractéristiques ***

; est-vide? : Arbre --> Booleen
;              a   |-> vrai si a=ø, faux sinon
(define est-vide?
  (lambda (a)
    (null? a)))

; val-racine : Arbre --> Entier
;               a   |->   n   si a=<n, g, d>
(define val-racine
  (lambda (a)
    (car a)))

; fils-gauche : Arbre --> Arbre
;               a    |->   g   si a=<n, g, d>
(define fils-gauche
  (lambda (a)
    (cadr a)))

; fils-droit : Arbre --> Arbre
;               a    |->   d   si a=<n, g, d>
(define fils-droit
  (lambda (a)
    (caddr a)))


; feuilles? : Arbre --> Booleen
;              a   |-> vrai si les fils de a sont des arbres vides
(define feuille?
  (lambda (a)
    (and (not (est-vide? a))
         (est_vide? (fils-gauche a))
         (est_vide? (fils-droit a)))))
```

```
; ============================================================================
;                    UTILISATION DU TYPE ABSTRAIT Arbre
;
;     Dans la suite, on ignore que les arbre sont representes par des listes.
;     On ne peut creer et manipuler des arbre qu'a l'aide des fonctions
;     generatrices et caracteristique definies dans la partie implantation.
; ============================================================================


; ================================================
; 2) Fonctions elementaires sur les arbres binaires


; arbres=? : Arbre, Arbre --> Booleen
;             a1 ,   a2  |-> vrai si  a1 et a2 sont égaux
; Deux arbres sont egaux si les valeurs de leurs racines
; sont egales et leurs fils gauche sont deux arbres egaux
; et leurs fils droits sont deux arbres egaux

(define arbre=?
  (lambda (a1 a2)
    (cond ((est-vide? a1) (est-vide? a2))
          ((est-vide? a2) #f)
          (else (and (equal? (val-racine a1) (val-racine a2))
                     (arbre=? (fils-gauche a1)
                              (fils-gauche a2))
                     (arbre=? (fils-droit a1)
                              (fils-droit a2)))))))



; nb-noeuds : Arbre --> Entier
;              a   |-> le nombre de noeuds de l'arbre a
; Un arbre vide n'a aucun noeud.
; Si a n'est pas vide il contient : 1 noeud qui est la racine
;                                   + le nombre de noeuds de son fils gauche
;                                   + le nombre de noeuds de son fils droit

(define nb-noeuds
  (lambda (a)
    (if (est-vide? a)
        0
        (+ 1 (nb-noeuds (fils-gauche a))
           (nb-noeuds (fils-droit a))))))


; hauteur : Arbre --> Entier
;            a   |-> la hauteur de l'arbre a
; La hauteur de l'arbre vide est 0
; Si a n'est pas vide sa hauteur est le maximum des hauteurs
; de ses fils + 1 (pour la racine)

(define hauteur
  (lambda (a)
    (if (est-vide? a)
        0
        (max (+ 1 (hauteur (fils-gauche a)))
             (+ 1 (hauteur (fils-droit a)))))))
```

```
; valmax : Arbre --> Entier
;            a   |-> la valeur maximale presente dans a
; Si l'arbre est vide la valeur maximale est -1 par convention
; Si a n'est pas vide, la valeur maximum est le plus grand
; des trois entiers : valeur de la racine, valeur maximale des noeuds
; du fils gauche, et valeur maximale des noeuds du fils droit.

(define valmax
  (lambda (a)
    (if (est-vide? a)
        -1
        (let ((maxg (valmax (fils-gauche a)))
              (maxd (valmax (fils-droit a))))
          (max (val-racine a) maxg maxd)))))


; meme-struct : Arbre, Arbre --> Booleen
;                 a  ,   b   |-> vrai si a et b ont la meme structure
; Deux arbre ont la meme structure s'ils sont egaux aux valeurs pres des noeuds.

(define meme-struct
  (lambda (a b)
    (cond ((est-vide? a) (est-vide? b))
          ((est-vide? b) #f)
          (else (and (meme-struct (fils-gauche a)
                                  (fils-gauche b))
                     (meme-struct (fils-droit a)
                                  (fils-droit b)))))))



; ============================================
; 3) Parcours d'un arbre en profondeur

; somme-noeud : Arbre --> Entier
;                 a   |-> la somme des valeurs des noeuds de a
; Si l'arbre est vide cette somme vaut 0.
; Si a n'est pas vide, cette somme est egale a la valeur de
; la racine de a + la somme des noeuds de ses fils gauche et droit

(define somme-noeuds
  (lambda (a)
    (if (est-vide? a)
        0
        (+ (val-racine a)
           (somme-noeuds (fils-gauche a))
           (somme-noeuds (fils-droit a))))))


; fois-n : Arbre --> Arbre
;            a   |-> un arbre de meme structure que a
;                    et dont les valeurs des noeuds sont
;                    multipliees par n

(define fois-n
  (lambda (a n)
    (if (est-vide? a)
        (arbre-vide)
        (cons-arbre (* (val-racine a) n)
                    (fois-n (fils-gauche a) n)
                    (fois-n (fils-droit a) n)) )))
```

```
; liste-valeurs : Arbre --> Liste
;                   a   |-> la liste des valeurs des noeuds de a
; Si l'arbre est vide cette liste est vide.
; Sinon, cette liste est construite a partir de la valeur de
; la racine et des listes des valeurs des fils gauche et droit.

(define liste-valeurs
  (lambda (a)
    (if (est-vide? a)
        '()
        (cons (val-racine a)
              (append (liste-valeurs (fils-gauche a))
                      (liste-valeurs (fils-droit a))))
        )))



; ============================================
; 4) Arbres binaires ordonnees

; est-dans? : Entier, ArbreBO --> Booleen
;               n  ,    a    |-> vrai si n est la valeur d'un des noeuds de a
; Si l'arbre est vide la recherche est infructueuse.
; Si l'arbre n'est pas vide et que la valeur de la racine n'est pas
; la valeur n recherchee, on utilise la propriete d'arbre bianire ordonne
; pour poursuivre la recherche soit dans le fils gauche, soit dans le fils droit.

(define est-dans?
  (lambda (n a)
    (cond ((est-vide? a) #f)
          ((> n (val-racine a)) (est-dans n (fils-droit a)))
          ((< n (val-racine a)) (est-dans n (fils-gauche a)))
          (else #t))))


; insere : Entier, ArbreBO --> ArbreBO
;            n   ,    a    |-> l'arbre binaire ordonnee a dans
;                             lequel on a insere un nouveau noeud de valeur n
; Si l'arbre est vide on construit la feuille dont la valeur est n.
; Sinon, on utilise la propriete d'arbre ordonne pour inserer le nouveau noeud
; soit dans le fils gauche (si la valeur a inserer est plus grande que la valeur
; de la racine), soit dans le fils droit (dans le cas contraire).
; Si la valeur est deja celle d'un noeud de l'arbre, on ne fait rien.

(define insere
  (lambda (n a)
    (cond ((est-vide? a)
           (cons-arbre n (arbre-vide) (arbre-vide)))
          ((< n (val-racine a))
           (cons-arbre (val-racine a)
                       (insere n (fils-gauche a))
                       (fils-droit a)))
          ((> n (val-racine a))
           (cons-arbre (val-racine a)
                       (fils-gauche a)
                       (insere n (fils-droit a))))
          (else a))))
```