Programmation récursive DEUG MIAS — UPMC

queinnec

Octobre 2003 — janvier 2004

Exercice 1 – Arbres binaires : Interrogation électronique

Mots-clés : récursion sur les arbres, barrière d'abstraction

Auteur	Mainteneur	Révision
Michele.Soria@lip6.fr	Michele.Soria@lip6.fr	Id : e-ms-interro-arbres.hbk,v 1.4 2002/07/12 07 :36 :35 queinnec Exp

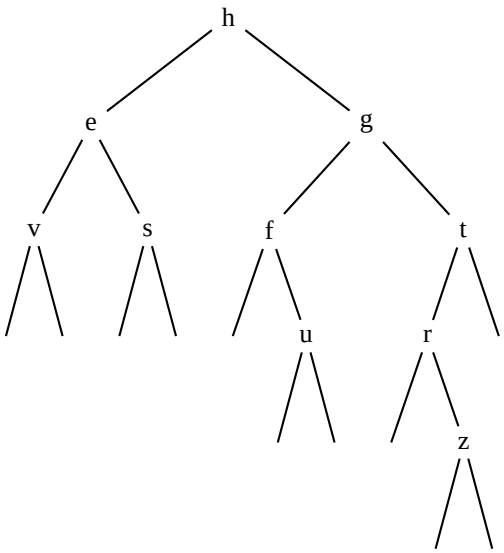
Cette interrogation porte sur les arbres binaires.
Avec le menu «add teachpack», chargez le teachpack

`collects/drscheme/tools /mias/exos/arbre-binaire.ss`

Puis écrivez toutes les fonctions demandées en utilisant la barrière d'abstraction des arbres binaires :

```
;;; ab-arbre : alpha * ArbreBin[alpha] * ArbreBin[alpha] -> ArbreBin[alpha]
;;; (ab-arbre e B1 B2) rend l'arbre binaire forme de la racine d'etiquette e,
;;; du sous-arbre gauche B1 et du sous-arbre droit B2

;;; ab-vide :  -> ArbreBin[alpha]
;;; (ab-vide) rend l'arbre binaire vide

;;; ab-vide? : ArbreBin[alpha] -> bool
;;; (ab-vide? B) rend vrai ssi B est l'arbre vide

;;; ab-etiquette : ArbreBin[alpha]/non vide/-> alpha
;;; (ab-etiquette B) rend l'etiquette de la racine de l'arbre B

;;; ab-ss-arbre-gauche : ArbreBin[alpha]/non vide/ -> ArbreBin[alpha]
;;; (ab-ss-arbre-gauche B) rend le sous-arbre gauche de B

;;; ab-ss-arbre-droit : ArbreBin[alpha]/non vide/ -> ArbreBin[alpha]
;;; (ab-ss-arbre-droit B) rend le sous-arbre droit de B
```

Question 1 : Écrire la fonction `ab-nombre-noeuds` qui rend le nombre de nœuds d'un arbre binaire. On compte uniquement les nœuds portant une étiquette, par exemple l'arbre de la figure suivante a 10 nœuds.
Écrire une fonction de test `test-ab-nombre-noeuds` .



Solution de la question 1 de l'exercice 1 :

```
;;; ab-nombre-noeuds : ArbreBin[alpha] -> nat
;;; (ab-nombre-noeuds B) rend le nombre de noeuds de B
(define (ab-nombre-noeuds B)
  (if (ab-vide? B)
      0
      (+ 1 (ab-nombre-noeuds (ab-ss-arbre-gauche B))
         (ab-nombre-noeuds (ab-ss-arbre-droit B)))))
```

La fonction de test :

```
(define (test-ab-nombre-noeuds)
  (test (ab-nombre-noeuds (ab-vide)) -> 0
        (ab-nombre-noeuds (ab-arbre 'r (ab-vide) (ab-vide))) -> 1
        (ab-nombre-noeuds (binExemple)) -> 10))
```

où l'arbre de la figure est engendré par la fonction

```
(define (binExemple)
  (let * ((B1 (ab-arbre 'r (ab-vide) (ab-arbre 'z (ab-vide)(ab-vide))))
          (B2 (ab-arbre 's (ab-vide) (ab-vide)))
          (B1-2 (ab-arbre 'u (ab-vide) (ab-vide)))
          (B2-2 (ab-arbre 'v (ab-vide) (ab-vide)))
          (B3 (ab-arbre 't B1 (ab-vide)))
          (B4 (ab-arbre 'e B2-2 B2))
          (B5 (ab-arbre 'f (ab-vide) B1-2))
          (B6 (ab-arbre 'g B5 B3)))
    (ab-arbre 'h B4 B6)))
```

Fin de la solution de la question 1 de l'exercice 1.

Question 2 : Écrire la fonction `ab-nombre-noeuds-niveau` qui, étant donnés un entier naturel *k* et un arbre binaire *B* rend le nombre de nœuds étiquetés à niveau *k* dans *B*. La racine d'un arbre non vide est à niveau 1, et le niveau d'un nœud est égal au niveau de son père augmenté de 1.

Par exemple l'arbre de la figure précédente a 1 nœud à niveau 1, 2 nœuds à niveau 2, 4 nœuds à niveau 3, 2 nœuds à niveau 4 et 1 nœud à niveau 5.

Écrire une fonction de test `test-ab-nombre-noeuds-niveau`.

Solution de la question 2 de l'exercice 1 :

```
;;; ab-nombre-noeuds-niveau : nat * ArbreBin[alpha] -> nat
;;; (ab-nombre-noeuds-niveau k B) rend le nombre de noeuds a niveau k dans B
(define (ab-nombre-noeuds-niveau k B)
```

```
(if (ab-vide? B)
    0
    (if (= k 1)
        1
        (+ (ab-nombre-noeuds-nive      au (- k 1) (ab-ss-arbre-gauche      B))
           (ab-nombre-noeuds-nive      au (- k 1) (ab-ss-arbre-droit      B))))))
```

La fonction de test

```
(define (test-ab-nombre-noeuds-      ni ve au )
  (test (ab-nombre-noeuds-nivea      u 0 (ab-vide))    -> 0
        (ab-nombre-noeuds-nivea      u 2 (ab-vide))    -> 0
        (ab-nombre-noeuds-nivea      u 1 (ab-arbre   'r (ab-vide)   (ab-vide)))    -> 1
        (ab-nombre-noeuds-nivea      u 1 (binExemple))    -> 1
        (ab-nombre-noeuds-nivea      u 2 (binExemple))    -> 2
        (ab-nombre-noeuds-nivea      u 3 (binExemple))    -> 4
        (ab-nombre-noeuds-nivea      u 4 (binExemple))    -> 2
        (ab-nombre-noeuds-nivea      u 5 (binExemple))    -> 1
        (ab-nombre-noeuds-nivea      u 6 (binExemple))    -> 0))
```

Fin de la solution de la question 2 de l'exercice 1.

Question 3 : Écrire une fonction `ab-suffixe` qui, étant donné un arbre binaire, retourne la liste de ses étiquettes en ordre suffixe. Par exemple pour l'arbre de la figure précédente, cela donne la liste

```
(v s e u f z r t g h).
```

Écrire une fonction de test `test-ab-suffixe` .

Solution de la question 3 de l'exercice 1 :

```
;;; ab-suffixe : ArbreBin[alpha] -> LISTE[alpha]
;;; (ab-suffixe B) rend la liste suffixe de toutes les etiquettes de B
(define (ab-suffixe B)
  (if (ab-vide? B)
      '()
      (append (ab-suffixe (ab-ss-arbre-gauche B))
              (ab-suffixe (ab-ss-arbre-droit B) )
              (list (ab-etiquette B)))))
```

La fonction de test

```
(define (test-ab-suffixe)
  (test (ab-suffixe (ab-vide))    -> '()
        (ab-suffixe (ab-arbre 'r (ab-vide) (ab-vide)))    -> '(r)
        (ab-suffixe (binExemple))    -> '(v s e u f z r t g h)))
```

Fin de la solution de la question 3 de l'exercice 1.