Programmation applicative – L2
TD 8 : Récursivité sur les arbres

G.Artignan {artignan@lirmm.fr} M.Lafourcade {lafourcade@lirmm.fr}
S.Daudé {sylvain.daude@univ-montp2.fr} A.Chateau {chateau@lirmm.fr}
B.Paiva Lima da Silva {bplsilva@gmail.com} C.Dony {dony@lirmm.fr}

1 Récursivité arborescente : rendre la monnaie

On rappelle l'exemple vu en cours : le nombre de façons de rendre une somme S (par exemple donnée en centimes d'euros) en utilisant n types de pièces (8 types de pièces, de 1, 2, 5, 10, 20, 50, 100 et 200 cts) est :

- Si $S = 0$, il y a 1 solution qui consiste à ne rien faire.
- Sinon si $S < 0$, aucune façon de rendre une somme négative ou si $n = 0$, aucune solution pour rendre une somme non nulle avec aucune pièce
- Sinon : le nombre de façons de changer la même somme S avec $n - 1$ types de pièces, plus le nombre de façons de changer la somme $(S - d)$ avec n pièces, d étant la valeur du premier des types de pièces

et dont voici l'implémentation :

```
(define (rendreMonnaie somme nbSortesPieces valeurPiece)
   (define (rendre somme n)
     (cond ((= somme 0) 1)
           ((or (< somme 0) (= n 0)) 0)
           (#t (+ (rendre somme (- n 1)) (rendre (- somme (valeurPiece n)) n)))))
(rendre somme nbSortesPieces))

(define (valeurPiecesEuro piece)
   (cond ((= piece 1) 1)
         ((= piece 2) 2)
         ((= piece 3) 5)
         ((= piece 4) 10)
         ((= piece 5) 20)
         ((= piece 6 ) 50)
         ((= piece 7 ) 100)
         ((= piece 8 ) 200)
         ))

(define (rendreEuro somme)
   (rendreMonnaie somme 8 valeurPiecesEuro))
```

Exemple : il y a 4 façons de rendre 5 centimes, 11 façons de rendre 10 centimes et 4112 façons de rendre 100 centimes, 73682 façons de rendre 200 centimes.

Exercice 1 *Modifier le programme précédent pour qu'il rende la liste des solutions plutôt que leur nombre.*

Exemple : Pour 10 centimes, voici les 11 solutions

```
((1 1 1 1 1 1 1 1 1 1) (1 1 1 1 1 1 1 1 2) (1 1 1 1 1 1 2 2) (1 1 1 1 2 2 2)
 (1 1 2 2 2 2) (2 2 2 2 2) (1 1 1 1 1 5) (1 1 1 2 5) (1 2 2 5) (5 5) (10))
```

2 Récursivité sur les arbres

Il sera question d'arbres colorés dans cette partie. Un arbre rouge-noir est un arbre binaire dont les nœuds sont étiquetés par un couple d'informations :

- un entier
- une couleur, soit rouge, soit noir

La figure 2 représente un arbre rouge-noir (les nœuds grisés représentent les nœuds rouges).

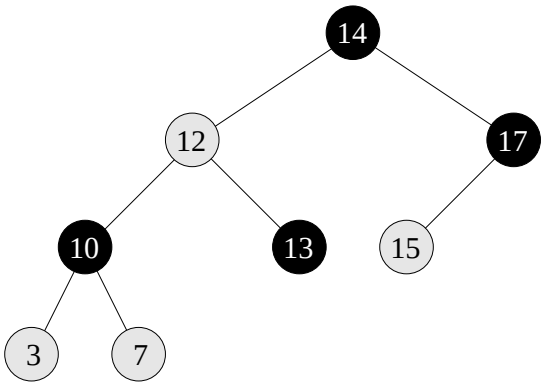

FIGURE 1 – Un arbre rouge-noir

Dans les arbres classiques, les nœuds sont étiquetés par des éléments atomiques comme des entiers par exemple. Il n'y a donc pas besoin de manipuler le contenu de ces nœuds d'une façon spéciale. Ici les nœuds comportent deux informations : l'entier et la couleur. On va donc commencer par s'intéresser à cette "sous-structure de données" de nœuds, avant de traiter la structure de données d'arbre rouge-noir.

2.1 Structure de données Nœud

Voici l'interface souhaitée pour la structure de données Nœud :

Constructeur

`make-noeud int col`	Construit le nœud ayant pour entier `int` et pour couleur `col`

Accesseurs

`get-noeud-int node`	Renvoie la valeur de l'entier du nœud `node`
`get-noeud-col node`	Renvoie la valeur de la couleur du nœud `node`

Prédicats

`noir? node`	Renvoie vrai si la couleur du nœud est noir, faux sinon
`pair? node`	Renvoie vrai si l'entier du nœud est pair, faux sinon

Exercice 2 *Par quelle structure de données choisissez-vous de représenter un nœud ? Expliciter notamment le choix de la représentation de la couleur.*

Exercice 3 *Implémenter l'interface pour la structure de données Nœud.*

2.2 Structure de données ArbreRN

Voici l'interface pour manipuler les arbres rouge-noir :

Constante

`empty-arn`	L'arbre vide

Prédicat

`empty-arn?`	Renvoie Vrai si l'arbre est vide, Faux sinon
`leaf-arn?`	Renvoie Vrai si l'arbre est une feuille, Faux sinon.

Constructeur

`make-arn root left right` Construit un ArbreRN ayant pour racine le nœud `root`, et comme sous-arbres gauche et droite respectivement `left` et `right`

Accesseurs

`get-root-arn arn`	Donne la racine d'un ArbreRN
`get-left-arn arn`	Donne le sous-arbre gauche d'un ArbreRN
`get-right-arn arn`	Donne le sous-arbre droit d'un ArbreRN

NB : Dans la suite on ne demande PAS d'implémenter cette interface, mais seulement de l'utiliser...

Exercice 4 *Donnez une procédure* `pairARN` *qui à partir d'un arbre RN, donne un arbre RN de même structure et avec les mêmes valeurs de nœuds mais pour lequel :*

- *un nœud de valeur paire est rouge*
- *un nœud de valeur impaire est noir*

Exercice 5 *Une branche monochrome est une branche d'un arbre RN dont tous les nœuds sont de la même couleur, de la racine jusqu'à la feuille. Donnez une procédure* `monochrome?` *qui renvoie vrai s'il existe une branche monochrome dans l'arbre, et faux sinon.*