Programmation fonctionnelle

Répétition 6

26 mars 2015

Correction exercices proposés

Exercice 1.

On considère des arbres binaires complets (chaque nœud a exactement 0 ou 2 fils) dont les feuilles sont étiquetées par des symboles atomiques. Les nœuds internes ne sont pas étiquetés.

Soit `a` un tel arbre. Simplifier `a` consiste à supprimer dans cet arbre les feuilles redondantes. Ainsi tout nœud ayant deux fils étiquetés par le même symbole est remplacé par ce symbole et ainsi de suite.

Ecrire une fonction `simplify` qui prend pour argument un arbre binaire complet `a` et qui retourne l'arbre binaire complet simplifié correspondant.

Exercice 2.

Un entier naturel est *parfait* s'il est égal à la somme de ses diviseurs positifs propres. Écrire un prédicat `perfect?` déterminant si son argument est un nombre parfait.

Le nombre 28 est parfait parce que $28 = 1 + 2 + 4 + 7 + 14$;
le nombre 32 n'est pas parfait parce que $32 \neq 1 + 2 + 4 + 8 + 16$.

Fonctions

Exercice 3.

Définir les fonctions `symmetrize` et `anti-symmetrize`, qui prennent comme argument une fonction $f : R \to R$ et qui renvoient respectivement les fonctions

$$f' : R \to R \qquad x \mapsto \frac{f(x) + f(-x)}{2}$$

et

$$f' : R \to R \qquad x \mapsto \frac{f(x) - f(-x)}{2}$$

Définir ensuite une fonction `func-op` à trois arguments, un opérateur `op` de $R \times R \to R$, et deux fonctions unaires `f` et `g`. `func-op` renvoie la fonction unaire h telle que

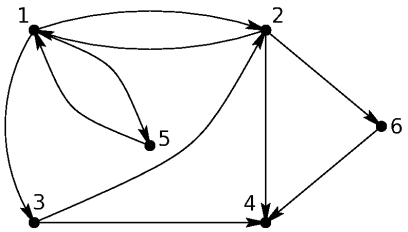
$$\forall x \in R : h(x) = \texttt{op}(\texttt{f}(x), \texttt{g}(x))$$

Redéfinir ensuite `symmetrize` et `anti-symmetrize` à partir de `func-op`.

Exercice sur les graphes

Exercice 4.
Écrire une fonction `inv` qui, à partir d'un graphe orienté, construit le graphe retourné correspondant. Nous convenons de représenter un graphe comportant n nœuds par une liste de n listes ; la liste correspondant au nœud x a pour premier élément x et pour éléments suivants les successeurs (immédiats) de x.

Par exemple le graphe



sera représenté par la liste

```
((1 2 3 5) (2 1 4 6) (3 2 4) (4) (5 1) (6 4))
```

Exercice 5.
Écrire une fonction `sublists` prenant comme argument une liste `l` et retournant la liste des sous-listes de `l`. (Une sous-liste de `l` est une liste de 0, 1 ou plusieurs éléments consécutifs de `l`.)

Exercices proposés

Exercice 6.
Un arbre n-aire est un arbre dont chaque nœud admet un nombre quelconque de fils ; seules les feuilles sont étiquetées, l'étiquette étant un nombre naturel. La fonction f prend comme argument un arbre n-aire a et renvoie l'arbre n-aire $f(a)$ obtenu en remplaçant chaque feuille de a étiquetée $n > 0$ par un nœud interne dont les n fils sont des feuilles étiquetées $n - 1$. On demande de programmer la fonction f.

Exercice 7.

Nous convenons de représenter un *graphe orienté* comportant n nœuds par une liste de n listes ; la liste correspondant au nœud x a pour premier élément x et pour éléments suivants les successeurs (immédiats) de x.

Écrire le prédicat `(path? n1 n2 g)` prenant deux nœuds $n1$ et $n2$ et la représentation g d'un *graphe orienté* en arguments et retournant `#t` s'il existe un chemin de $n1$ vers $n2$ dans g et `#f` sinon.

Exercice 8.

Ecrire une fonction `longest-inc` retournant la plus longue sous-liste croissante de la liste d'entiers donnée en argument.