INFO0054 - Programmation fonctionnelle

Répétition 4: Les spécificités de la programmation fonctionnelle

Jean-Michel Begon

12 mars 2019

Closure et factory

Exercice 1.
Définir la fonction `filter_` à deux arguments, un prédicat unaire `p?` et une liste d'éléments appartenant à son domaine `ls`, et renvoyant la liste des éléments de `ls` pour lesquels l'application du prédicat est `#t`.
Remarque : la fonction `filter` est prédéfinie, nous la redéfinissons sous un autre nom à titre d'exercice.

Exercice 2.
Redéfinir les fonctions `big` et `remove-all` à l'aide de la fonction `filter`.

Exercice 3.
Définir la fonction `linear` qui prend en entrée deux réels, `m` et `p`, et qui renvoie la fonction $f : \mathbb{R} \to \mathbb{R}, f(x) = mx + p$.

Exercice 4.
Définir les fonctions `symmetrize` et `anti-symmetrize`, qui prennent comme argument une fonction $f : \mathbb{R} \to \mathbb{R}$ et qui renvoient respectivement les fonctions

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

et

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

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

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

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

Exercice 5.
Définir une fonction `compose-n` qui renvoie la fonction unaire donnée en argument `n` fois composée avec elle-même.

Variante :
Écrire une fonction `compose-fgf` qui prend comme argument une fonction unaire f et renvoie une fonction qui prend comme argument une fonction unaire g et qui renvoie la fonction $f \circ g \circ f$.

Variante 2 :
Écrire une fonction `compose-fgab` qui prend comme argument deux fonctions f et g, ainsi que deux entiers a et b et renvoie la fonction

$$\underbrace{f \circ \ldots f}_{a} \circ \underbrace{g \circ \ldots g}_{b}$$

Variante 3 :
Écrire une fonction `compose-fa` qui prend comme argument une fonction f et deux entier a, b et renvoie la fonction

$$x \mapsto \underbrace{f \circ \ldots f}_{a}(b^x)$$

Store passing style

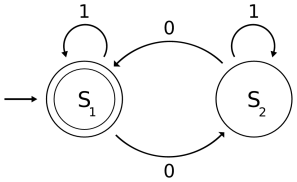
Exercice 6.
Le générateur de nombres aléatoires en C++ est une instance du schéma LCG (*linear congruent generator*) :

$$X_{n+1} = (aX_n + c) \mod m$$

où $m = 2^{31} - 1 = 2147483647$, $a = 48271$ et $c = 0$. Le choix de la graine X_0 ($0 \leq X_0 \leq m$) est laissé à l'utilisateur.

Implémenter la fonction `create-lcg` qui prend la graine en entrée et qui renvoie le générateur LCG d'ordre 0 pour cette graine. Le générateur d'ordre i est une fonction sans argument qui renvoie une paire dont le `car` est X_i et le `cdr` est le générateur d'ordre $i + 1$.

Exercice 7.
Informellement, un automate fini déterministe permet de détecter certaines structures dans des chaînes de caractères. Par exemple, l'automate suivant permet de déterminer s'il y a un nombre paire de 0 dans un nombre binaire :



Implémenter un tel automate.

Continuation passing style

Exercice 8.
Ecrire une fonction `sqrt*` qui à tout entier strictement positif n associe le nombre

$$\sqrt{n + \sqrt{n-1 + \cdots + \sqrt{1}}}$$

de manière classique (récursion directe) et en CPS.

Exercice 9.
Ecrire une fonction `sqrt*-inv` qui à tout entier strictement positif n associe le nombre

$$\sqrt{1 + \sqrt{2 + \cdots + \sqrt{n}}}$$

en CPS.