INFO0054 - Programmation fonctionnelle

Répétition 5: Les spécificités de la programmation fonctionnelle

Jean-Michel Begon

20 mars 2018

Les listes variables d'arguments

Exercice 1.
Que renvoie l'évaluation des expressions suivantes :

```
(+)        (*)        (list)
```

Exercice 2.
Ecrire une fonction `my+` à un nombre variable d'arguments et qui renvoie la somme de ces éléments.

Closure et factory

Exercice 3.
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 4.
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

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

Les listes variables d'arguments et closure

Exercice 5.
Ecrire une fonction `linear-map-factory` à un nombre variable d'arguments réels et qui renvoie une fonction prenant une liste de réel en argument et renvoyant le produit scalaire entre cette liste et les arguments encapsulés.

Exercice 6.
Ecrire une fonction `curry` qui prend deux arguments, une fonction $f : D_1 \times \ldots \times D_n \to Y$ et un élément $d_1 \in D_1$ et qui renvoie une fonction $h : D_2 \times \ldots \times D_n \to Y$ telle que $(h\ d_2 \ldots d_n) = (f\ d_1\ d_2 \ldots d_n)$.

Exercice 7.
Ecrire une fonction `my-map` qui prend une fonction $f : D_1 \times \ldots \times D_n \to Y$ $(n \geq 1)$ ainsi que n listes $l_1, \ldots, l_n$ telles que $|l_i| = m$ et les éléments de l_i appartiennent à D_i $(1 \leq i \leq n)$ et qui renvoie une liste ys de m éléments telle que $ys_j = f(l_1^{(j)}, \ldots, l_n^{(j)})$ $(1 \leq j \leq m)$.

```
(my-map (lambda (x y) (+ (* x 2) y)) '(1 2 3) '(1 1 1)) ==> '(3 5 7)
```

Store passing style

Exercice 8.
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 $i + 1$.

Continuation passing style

Exercice 9.
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 10.
Ecrire une fonction `sqrt*-inv` qui à tout entier strictement positif n associe le nombre

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

en CPS.