INFO0054 - Programmation fonctionnelle

Répétition 3

Jean-Michel Begon

10 Mars 2020

Exercice 1.
Définir la procédure `count-all` à deux arguments, un nombre `x` et une liste `ls`, et qui compte, en profondeur, le nombre de fois que `x` est contenu dans la liste `ls`.

```
(count-all 1 '(0 (1 2 (3 4 (1)) (3 (2 1) 1) 1) 0 (1 2 (1 2 3)))) ⇒ 7
```

Exercice 2.
Écrire une fonction `frequency` prenant en argument une liste d'atomes `ls` et renvoyant une table d'apparition de chacun des atomes dans la liste `ls`. Cette table sera représentée par une liste de paires pointées, dont le `car` est un atome et le `cdr` le nombre d'occurrences de cet atome. Tous les atomes de `ls` apparaissent une et une seule fois dans la table.

```
(frequency '(a b c b a b d a c b b))
   ⇒ ((a . 3) (b . 5) (c . 2) (d . 1))
```

Exercice 3.
Écrire une fonction `prod-cart` qui prend en entrée deux ensembles et qui retourne l'ensemble des paires pointées dont le `car` est un élément du premier ensemble et le `cdr` un élément du second :

```
(prod-cart '(1 2 3) '(a b)) ⇒ '((1 . a) (1 . b) (2 . a) (2 . b) (3 . a) (3 . b))
```

Exercice 4.
Écrire une fonction `lpref` prenant comme argument une liste `u` et retournant la liste des préfixes de `u`. (La liste vide et la liste `u` elle-même sont des préfixes de `u`.)

Exercice 5.
Écrire une fonction qui renvoie la liste des sous-ensembles d'un ensemble donné.

```
(lset '(a b c)) ⇒ '(() (c) (b) (b c) (a) (a c) (a b) (a b c))
```

Exercice 6.
Écrire une fonction qui renvoie la liste des permutations d'une liste donnée.

Exercice 7.
Ecrire la fonction `conway` qui calcule la n-ieme ligne de la suite de Conway :

$$1$$
$$1, 1$$
$$2, 1$$
$$1, 2, 1, 1$$
$$1, 1, 1, 2, 2, 1$$

Exercice 8.
Soit la fonction f définie sur $\mathbb{N}$ par :

$$f(n) = \left(\sum_{i=0}^{n-1} ([2 + f(i)] \times [3 + f(n - i - 1)]) \right) \mod (2n + 3)$$

implémentée par le programme suivant :

```
(define f (lambda (n) (fx n 0 '())))

(define fx
  (lambda (k i u)
    (let ((next
           (modulo (apply + (map (lambda (x y) (* (+ 2 x) (+ 3 y)))
                                 u
                                 (reverse u)))
                   (+ i i 3))))
      (if (zero? k) next
          (fx (- k 1) (+ i 1) (cons next u))))))
```

Spécifier la fonction `fx`.