INFO0054 - Programmation fonctionnelle

Répétition 3: Récursion sur les nombres et sur les listes

Jean-Michel Begon

19 février 2019

Correction des exercices proposés

Exercice 1.
Ecrire une version *tail-recursive* des fonctions

- `sum-list`
- `prefix-n`

Exercice 2.
Soient les fonctions suivantes :

```
(define sublist-pos
  (lambda (l)
    (reverse (sublist-pos-acc l '()))))

(define sublist-pos-acc
    (lambda (l acc)
        (cond ((null? l) acc)
              ((> (car l) 0) (sublist-pos-acc (cdr l) (cons (car l) acc)))
              (else (sublist-pos-acc (cdr l) acc)))))
```

`sum-pos` renvoie la sous-liste des éléments positifs de la liste de nombre `l`. Spécifier `sum-pos-acc`.

Nouveaux exercices

Exercice 3.
Implémenter une fonction `remove-last` qui prend en argument un naturel n et une liste ℓ de taille $m \geq n$, et qui renvoie la liste contenant les $m - n$ premiers éléments de ℓ.

Exercice 4.
Implémenter une fonction `remove-end-0s` qui prend comme argument une liste de nombres et qui renvoie la liste le plus petit préfixe dont le suffixe complémentaire ne contient que des zéros.

```
(remove-end-0s '(0 1 2 0 3 4 0 0)) => '(0 1 2 0 3 4)
```

Exercice 5.
Soient les fonctions

```
(define ml
  (lambda (n x)
    (ml-aux n x '())))

(define ml-aux
    (lambda (n x acc)
        (if (zero? n) acc
            (ml-aux (- n 1) x (cons x acc)))))
```

Corriger la spécification suivante :

> Si `n` est un naturel, `x` est un naturel quelconque et `acc` est la liste vide, alors `(ml-aux n x acc)` renvoie une liste de taille `n` dont chaque élément est `x`.

se rapportant à la fonction `ml-aux`.

Exercice 6.
Les nombres de Gribomont sont définis comme suit :

si $n < 3$, $f(n) = n$
si $n \geq 3$, $f(n) = f(n-1) + f(n-2) + f(n-3)$.

Écrire une fonction *efficace* qui permet de calculer les nombres de Gribomont.

Exercice 7.
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 8.
Soient les fonctions suivantes :

```
(define xy
  (lambda (x y)
    (xyz x y 0)))

(define xyz
  (lambda (x y z)
    (cond ((null? y) z)
          ((= x (car y)) (xyz x (cdr y) (+ 1 z)))
          (else (xyz x (cdr y) z)))))
```

Ecrire une spécification pour la fonction `xyz`.

Exercice 9.
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`.

Exercice 10.
Calculer les coefficients binomiaux à l'aide de la relation $(0 \leq k \leq n)$

$$C_n^k = \begin{cases} 1, & \text{si } k = 0 \text{ ou } k = n \\ C_{n-1}^{k-1} + C_{n-1}^k, & \text{sinon} \end{cases}$$

Exercice 11.
Ecrire une fonction `div-ls` qui retourne la liste des diviseurs d'un entier strictement positif.