INFO0054 - Programmation fonctionnelle

Répétition 2 : Récursion sur les nombres et sur les listes

Jean-Michel Begon

20 Février 2018

Correction des exercices proposés

Exercice 1.
Écrire une fonction `sum-int` prenant en argument un naturel n et calculant la somme des naturels inférieurs ou égaux à n.

Exercice 2.
Définir la fonction `sum-list` qui calcule la somme des éléments d'une liste de nombres.

Les fonctions `length` et `expt` sont vues au cours théorique.

Les primitives `cons`, `list`, `append`

Exercice 3.
Donner le résultat de l'évaluation des expressions suivantes :

```
(car '(a b))
(cdr '(a b))
(car (cdr '(a b)))
(cdr (cdr '(a b)))
(cons 'a '())
(cons 'a '(b))
(cons '() '())
(cons '(a) '(b))
(list '(a) '(b))
(append '(a) '(b))
(cons 'a (cons 'b (cons 'c '())))
(car (cons 'a '()))
(cdr (cons 'a '()))
(cons x '(a b))
(cadr '(a b c d))
(cadar '((a b) (c d) (e f)))
(list? (cons 'a 'b))
(list? (cons 'a (cons 'b '())))
(list? (cons (cons 'b '()) 'a))
(list? (cons (cons 'b '()) (cons 'b '())))
(null? (car '(a)))
(null? (cdr '(a)))
(null? '())
(null? '(a b))
(null? (car '(())))
(null? (car '(((())))))
(symbol? (car '(a b c)))
(symbol? (cons '() '()))
(equal? 'a (car '(a b)))
(equal? '(a b c) '(a b c))
(equal? '(a (b c)) '(a b c))
(equal? (cdr '(a c d)) (cdr '(b c d)))
(equal? '(car '((b) c)) (cdr '(a b)))
(cons (car '(a b c))
      (cons
        (car (cdr '(a b c)))
        (cons (car (cdr (cdr '(a b c)))) '())))
```

Exercices sur les nombres

Exercice 4.
Définir la fonction `mod` qui renvoie le modulo de deux nombres.

Remarque : la fonction *modulo* est prédéfinie, nous la redéfinissons sous un autre nom à titre d'exercice.

Exercice 5.
Les nombres triangulaires sont définis par la relation :

$$T(n) = \begin{cases} 0, & \text{si } n = 0 \\ n + T(n-1), & \text{sinon} \end{cases}$$

Ecrire une fonction classique et une fonction *tail-recursive* permettant de calculer ces nombres.

Exercices sur les listes

Exercice 6.
Écrire une fonction `big` qui prend comme arguments un nombre entier `n` et une liste `l` de nombres entiers, telle que `(big n l)` est la liste des éléments de `l` plus grands que `n`.

`(big 5 '(7 -3 6 1 -2 5 6))` $\Rightarrow$ `(7 6 6)`

Exercice 7.
Ecrire une fonction renvoyant le minimum d'une liste de nombres.

Réflexions

Exercice 8.
Quel serait le cas de base d'une fonction qui calcule le produit des éléments d'une liste de nombres ?

Exercice 9.
Soient les fonctions suivantes :

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

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

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

Lambdas imbriqués

Exercice 10.
Donner le résultat de l'évaluation des expressions suivantes :

```
((lambda (x) (* x x)) ((lambda (x) (* 2 x)) 6))
((lambda (x) (* x x)) (lambda (x) (* 2 x)))
((lambda (x y) ((lambda (z) (+ 12 y)) (* x x x x))) 5 2)
((lambda (x y) ((lambda (z) (* 3 z y)) (* x x))) 5 2)
((lambda (f x y) (f y x)) (lambda (u w) (+ 2 (* (+ 8 u) w))) 6 7)
((lambda (x) (lambda (y) (+ y x))) 2)
(((lambda (f) (lambda (x) (f (+ x 3)))) (lambda (y) (* 4 y))) 10)
```