Récursivité sur les listes

Objectif : définition **récursive** de la somme

Question : sur quoi porte la récursion ?

Réponse : sur le paramètre **l** donc sur la liste

- **Cas récursif :**

Si la liste est non vide, alors la somme est l'addition + du premier élément **(car l)** avec la somme du reste **(cdr l)** de la liste

- **Cas de base :**

Si la liste **l** est vide, la somme est zéro

Récursivité sur les listes

On veut définir une fonction **somme** effectuant la somme des éléments (nombres) d'une liste

Spécification :

;;; (somme l) retourne la somme des éléments

;;; de l

Récursivité sur les listes

```
;;; (somme l) retourne la somme des éléments
;;; de l
(define (somme l)
(if (pair? l) ; on teste si la liste est different de vide
(+ (car l) (somme (cdr l)))
0))
```

Récursivité sur les listes

Deux question à se poser pour une récursion bien fondée :

• Appels récursifs sur des ensembles "plus petits" ?

Oui, **(cdr l)** est "plus petit" que **l**

• Est-ce que le calcul se termine ?

Oui, quand on "arrive" à la liste vide **()**

Récursivité sur les listes

Définir la fonction **longueur** qui retourne la longueur d'une liste

```
;;;; (longueur l) retourne la longueur
;;; de la liste l
(define (longueur l)
(if (pair? l)
(+ 1 (longueur (cdr l)))
0))
```

Récursivité sur les listes

```
;;; (ajout-en-fin e l) retourne la liste l
;;; a laquelle on ajoute l'élément e en fin
(define (ajout-en-fin e l)
(if (pair? l)
(cons (car l) (ajout-en-fin e (cdr l)))
(cons e '())))
```

Récursivité sur les listes

• Une fonction va parcourir récursivement en profondeur une liste L si elle s'applique pour chaque liste l de cette liste L, de la même manière qu'elle s'applique sur L, et ceci de manière récursive : elle s'applique aussi sur les listes de l …

• On a ainsi deux niveaux de récursivité : le premier, traditionnel, sur la structure de L, et le second sur les éléments de L qui sont des listes

Récursivité sur les listes

Premier exemple : la fonction somme

• Définissons la fonction somme qui additionne tous les nombres d'une liste quelconque

```
(define somme ; → nombre
(lambda (L) ; L liste
(cond  ((null? L) 0)
       ((number? (car L))
         (+ (car L) (somme (cdr L))))
       (else (+ (somme (car L)) (somme (cdr L)))))))
```

Récursivité sur les listes

Pourquoi une version en profondeur ?

- (somme '(1 2 3 z 4)) → 10
- (somme '(1 (2 a 3) z 4)) → 5
- (somme '(1 (2 (3 b 6) 7) z 4) → 5

La fonction somme effectue seulement la somme des nombres non imbriqués dans des listes. Nous aimerions une fonction somme-prof qui permette les appels suivants :

- (somme-prof '(1 2 3 z 4)) → 10
- (somme-prof '(1 (2 a 3) z 4)) → 10
- (somme-prof '(1 (2 (3 b 6) 7) z 4) → 23

Récursivité sur les listes

Fonction somme : version en profondeur

```
(define somme-prof ; → nombre
  (lambda (L) ; L Liste
  (cond ((null? L) 0)
        ((number? (car L))
         (+ (car L) (somme-prof (cdr L))))
        ((list? (car L)) (+ (somme-prof (car L))
           (somme-prof (cdr L))))
        (else (somme-prof (cdr L))))))
```