

## RÉCURSIVITÉ PROFONDE

# DÉFINITION

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

# 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 (cdr L))))))
```

# 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 aimerais 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

# 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)))))))
```

# ILLUSTRATION

(somme-prof '(1 (2 a 3 ) z 4) )

$\downarrow$

(+ 1 (somme-prof '( (2 a 3) z 4) ))

$\downarrow$

(+ (somme-prof '(2 a 3) )

$\downarrow$

(+ 2 (somme-prof '(a 3)))

$\downarrow$

(somme-prof '(3))

$\downarrow$

(+ 3 (somme-prof '()))

0

$\downarrow$

(somme-prof '(z 4) ))

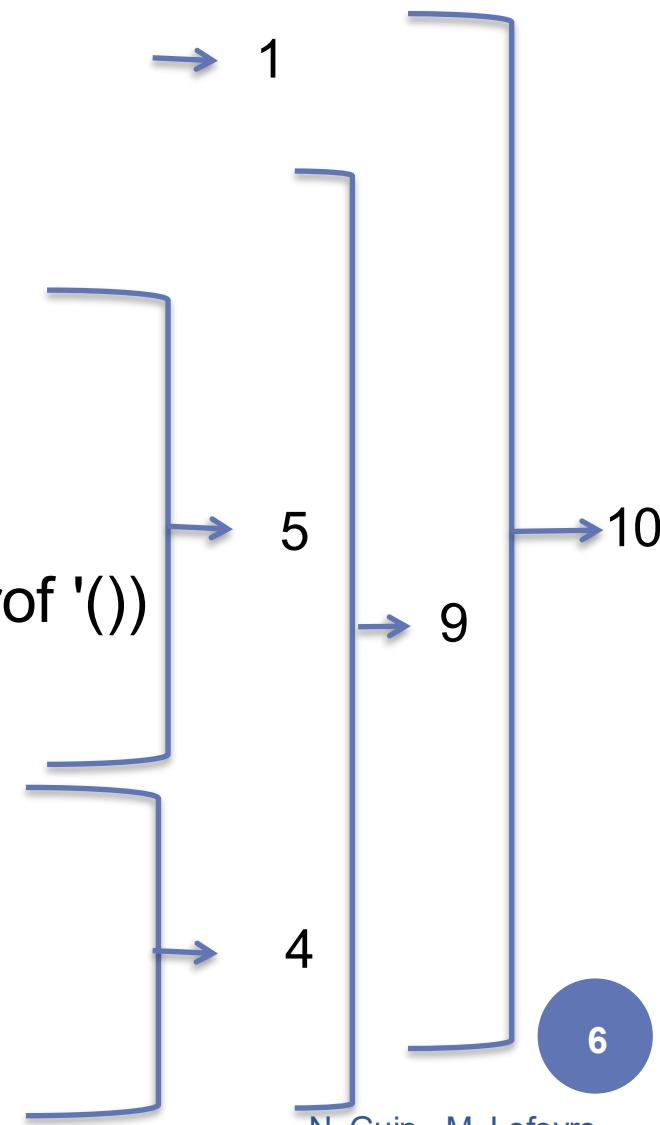
$\downarrow$

(somme-prof '(4))

$\downarrow$

(+ 4 (somme-prof '()))

0



## DEUXIÈME EXEMPLE :

### LA FONCTION « APLATIT »

- Écrivons une fonction qui enlève toutes les parenthèses d'une liste quelconque

```
(define aplatit ; → liste d'atomes
  (lambda (L) ; L Liste
    (cond ((null? L) '())
          ((list? (car L))
           (append (aplatit (car L))
                   (aplatit (cdr L)))))
          (else (cons (car L) (aplatit (cdr L)))))))
```

- $(\text{aplatit } '(\text{a} \ (\text{z e} \ (\text{a h}) \ \text{a b}) \ \text{i})) \rightarrow (\text{a z e a h a b i})$

# ILLUSTRATION

(applatit '(a (z e) b))

(cons a (aplatit '((z e) b)))

(append (aplatit '(z e)) )

(cons z (aplatit '(e)))

(cons e (aplatit '()))

'()

(aplatit '(b)))

(cons b (aplatit '()))

'()

