(Séquence 4.7

somme-cumulee

Un exemple plus complexe

```
;;; somme-cumulee: LISTE[Nombre] -> LISTE[Nombre]
;;; (somme-cumulee L) rend la liste dont le premier
;;; élément est égal à la somme des éléments de L,
;;; dont le deuxième élément est égal à la somme
;;; des éléments de (cdr L) ... dont le dernier
;;; élément est égal au dernier élément de L.

;;; chaque élément de la liste résultat est égal à
;;; la somme des éléments qui le suivent (au sens
;;; large) dans la liste initiale

(somme-cumulee (list 4))
   → (4)
(somme-cumulee (list 3 4))
   → (7 4)
(somme-cumulee (list 2 3 4))
   → (9 7 4)
```

Une définition à proscrire

```
(define (somme-cumulee L)
  (if (pair? L)
      (cons (somme L)
            (somme-cumulee (cdr L)))
      (list)))
```

A NE PAS IMITER

Double balayage de la liste `L` !
Si `L` comporte n termes, `somme` a une complexité linéaire (en nombre d'appels à `cdr`). Donc `somme-cumulee` consomme $n + (n-1) + (n-2) + \ldots + 1$ appels à `cdr` de l'ordre de n^2 appels donc une complexité quadratique.

Une autre définition à proscrire

```
(define (somme-cumulee L)                    A NE PAS IMITER
  (if (pair? L)
      (if (pair? (cdr L))
          (cons (+ (car L)
                   (car (somme-cumulee (cdr L))))
                (somme-cumulee (cdr L)))
          L)
      (list)))
```

Nommer pour éviter des recalculs !

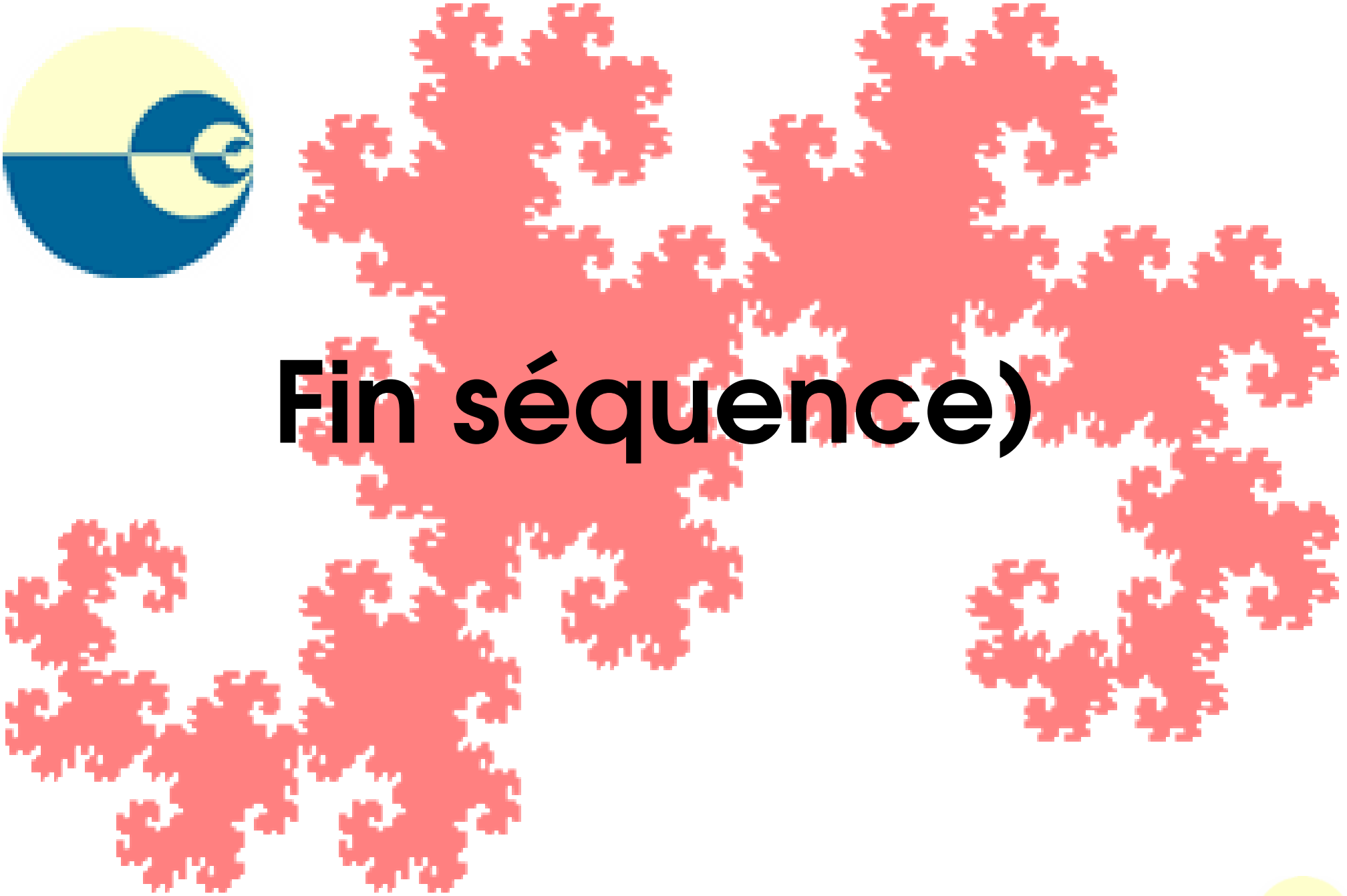
Une première définition

```
(define (somme-cumulee L)
  (if (pair? L)
      (if (pair? (cdr L))
          (let ((reste-fait (somme-cumulee (cdr L))))
            (cons (+ (car L) (car reste-fait))
                  reste-fait))
          L)
      (list)))
```

Définition interne pour éviter des tests

```
(define (somme-cumulee L)
  ;; sc-non-vide: LISTE[Nombre] -> LISTE[Nombre]
  ;; (sc-non-vide L) == (somme-cumulee L)
  ;; HYPOTHÈSE: L non vide
  (define (sc-non-vide L)
    (if (pair? (cdr L))
        (let ((reste-fait (sc-non-vide (cdr L))))
          (cons (+ (car L) (car reste-fait))
                reste-fait))
        L))
  (if (pair? L)
      (sc-non-vide L)
      L))
```

Deux fois moins d'appels à `pair?`.



Fin séquence)