Récursivité sur les listes

Deuxième exemple : la fonction « aplatit »

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

- (aplatit '(a (z e (a h) a b) i)) → (a z e a h a b i)

Récursivité sur les listes

```
(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)))))))
```

Récursivité sur les listes

Les listes d'associations [A-listes]

- Une **A-liste** est une liste de doublets $((x_1 . e_1) \ldots (x_n . e_n))$:

```
(let ((x 2) (y 3) (z (sqrt 6)))
  (+ x y z))
```

une A-liste de 3 associations

- Chaque $(x_i . e_i)$ est une association, de clé x_i et de valeur associée e_i.

- La primitive (assoc x AL) permet de retrouver l'association de clé x dans la A-liste AL, ou bien #f :

```
(define ($assoc x AL)
  (cond ((null? AL) #f)
        ((equal? x (caar AL)) (car AL))
        (else ($assoc x (cdr AL)))))
```

```
> (assoc 'deux '((un 1) (deux 2) (trois 3)))
(deux 2)
> (assoc 'quatre '((un 1) (deux 2) (trois 3)))
#f
```

Fonctionnelles

Les fonctions qui prennent d'autres fonctions en argument, ou retournent des fonctions en résultat sont dites **fonctionnelles** ou fonctions d'ordre supérieur

• On utilise beaucoup les fonctionnelles dans les langages fonctionnels comme Scheme : on les utilisera surtout pour écrire des fonctionnelles mettant en oeuvre un **schéma de récursion**

Schémas de récursion sur les listes et itérateurs

- **Schéma d'application** : itérateur **map**

On transforme une liste en une autre liste en appliquant une fonction à chaque élément

Fonctionnelles : Schémas d'application avec map

• La fonctionnelle **map** généralise le schéma d'application

Exemples :

```
(define (liste-positive? n)
     (if (> n 0) #t #f) )
(map positive? (list 5 0 -9 4 8 -7)) →
(#t #f #f #t #t #f )
;;; carre : Nombre -> Nombre
(define (carre x)
   (* x x))
(map carre (list 2 3 4 5)) → (4 9 16 25 )
;;; divise-5 : Nombre -> Nombre
(define (divise-5 x)
  (/ x 5))
(map divise-5 (list 10 20 30 40)) → (2 4 6 8 )
```

MAP

La fonction Map

• La fonction applique-à-tous est tellement utile qu'elle est prédéfinie en Scheme

• Elle porte le nom map et est en fait une version plus générale de applique-à-tous

MAP

Map-Apply : un exemple

• Une fonction pour calculer le produit scalaire de deux vecteurs :

L1 = '(x1 x2 … xn)

L2 = '(y1 y2 … yn)

produit scalaire = x1y1 + x2y2 + … + xnyn

• (define scalaire ; → nombre

(lambda (L1 L2) ; listes de nb

(apply + (map * L1 L2))))

Map-Apply : les différences (1)

MAP

• Prend autant de listes que l'arité de la fonction, toutes les listes étant de même longueur

APPLY

• Prend une liste comme argument, la longueur de cette liste étant égale à l'arité de la fonction

Map-Apply : les différences (2)

MAP

- Applique la fonction à chaque élément de la liste (ou des listes)
- Retourne toujours une liste de résultats du type de celui de la fonction

APPLY

- Applique la fonction à l'ensemble des éléments de la liste
- Retourne un résultat du type de celui de la fonction

La fonction map

```
(map abs '(1 -2 3 -4 5 -6))
(1 2 3 4 5 6 )

(map (lambda (x y) (* x y))
        '(1 2 3 4) '(8 7 6 5))
(8 14 18 20)
```

TP N°2