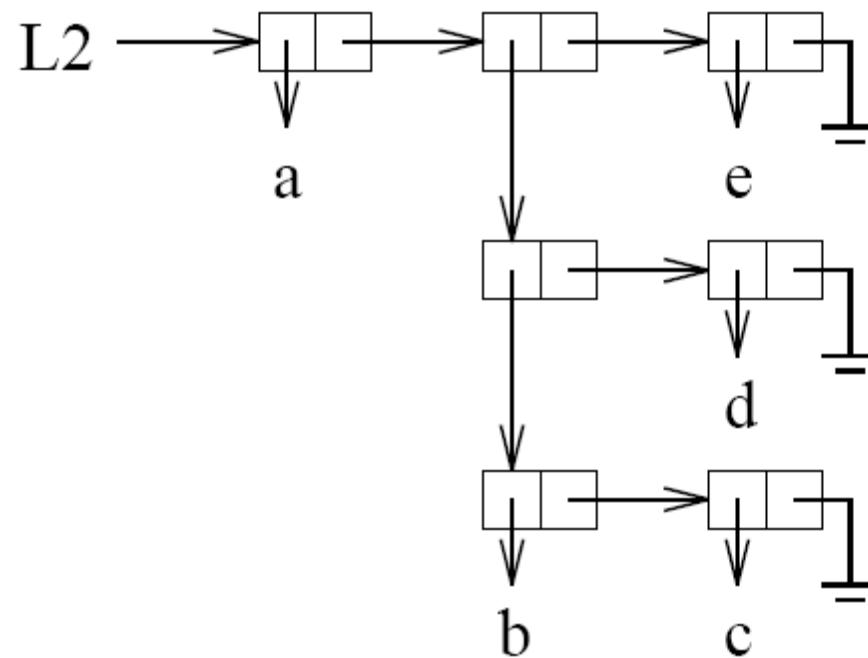


Représentation des listes

- ▶ A chacune des expressions formant une liste est associée une cellule mémoire constituée de deux pointeurs. Le premier de ces pointeurs donne l'adresse de l' atome ou de la liste correspondant, alors que le second pointeur donne l' adresse de la prochaine cellule.

Exemple

Si L2 est lié à (a ((b c) d) e)

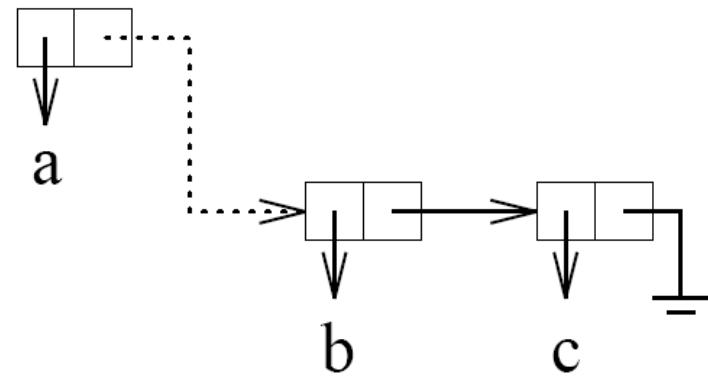


La fonction de construction

- ▶ Le premier paramètre de la liste est un atome à être placé en tête de la liste spécifiée comme second paramètre.
- ▶ Pour ce faire, une nouvelle cellule mémoire est créée
 - le premier de ses pointeurs pointe sur la première expression passée en paramètre
 - le second pointeur pointe sur la seconde expression

CONS

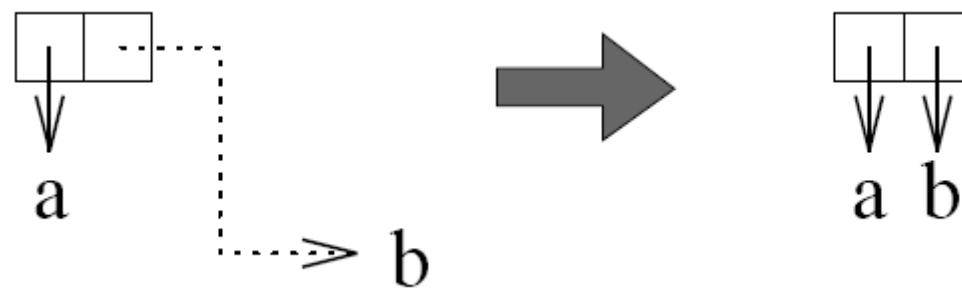
```
(cons `a `(b c))  
(a b c)
```



```
(cons ` (a b) ` (b c))  
((a b) b c)
```

Une paire pointée

(**cons** `a `b)



L'usage des paires pointée en Scheme est toutefois déconseillée
(les paires pointées ne sont pas des listes!)

CAR

- ▶ Content of the Address Register

```
(car ' (a b c))  
a  
(car ' ((a b) b c))  
(a b)
```

CDR

- ▶ Content of the Decrement Register

```
(cdr ' (a b c) )  
(b c)  
(cdr ' ((a b) b c) )  
(b c)  
(cdr ' (a (b c)) )  
( (b c) )
```

cons car et cdr sont complémentaires

```
(cons (car '(a b c)) (cdr '(a b c)))  
(a b c)
```

Utilisation cascadée

(cdr (car (cdr ' (a (b c d) e))))

peut s'écrire:

(cdadr ' (a (b c d) e))
(c d)

Concaténation de deux listes

```
(define (notre-append L1 L2)
  (if (null? L1)
      L2
      (cons (car L1)
            (notre-append (cdr L1) L2)))))

(notre-append '(a b) '(c d))
(a b c d)
```

Inversion d'une liste

```
(define (notre-reverse L)
  (if (null? L)
      ()
      (notre-append
        (notre-reverse (cdr L))
        (list (car L))))))

(notre-reverse '(a b c d))
(d c b a)
```

Appartenance à une liste

```
(define (notre-member a L)
  (cond ((null? L) ())
        ((equal? a (car L)) L)
        (#T (notre-member a (cdr L))))))

(notre-member 'a '(a b c))
(a b c)
(notre-member 'b '(a b c))
(b c)
(notre-member 'd '(a b c))
nil
```

La longueur d'une liste

```
(define (notre-length L)
  (if (null? L)
      0
      (+ 1 (notre-length
(cdr L))))))

(notre-length '(a b c))
```

D'autres exemples de fonctions

```
(define (voisins_egaux? L)
  (cond
    ((null? L) #f)
    ((null? (cdr L)) #f)
    ((equal? (car L) (cadr L)) #t)
    (else
      (voisins_egaux? (cdr L)))
  )
)
```

Qu'arrive-t-il si on change #t pour (cdr L) ?

```
(voisins_egaux? '(1 2 3 3 4 5))
(3 4 5)
```

Liste de nombre?

```
( define ( numberList? x )
  ( cond
    ( ( not ( list? x ) ) #f )
    ( ( null? x ) #t )
    ( ( not ( number? ( car x ) ) ) #f )
    ( else ( numberList? ( cdr x ) ) ) )
  ) )

( numberList? ' ( 1 2 3 4 ) )
#t
( numberList? ' ( 1 2 3 bad 4 ) )
#f
```

Equivalence?

```
(define ( eqExpr? x y )
  ( cond
    ( ( symbol? x ) ( eq? x y ) )
    ( ( number? x ) ( eq? x y ) )
    ; x doit etre une liste:
    ( ( null? x ) ( null? y ) )
    ; x doit etre une liste non vide:
    ( ( null? y ) #f )
    ( ( eqExpr? ( car x ) ( car y ) )
      ( eqExpr? ( cdr x ) ( cdr y ) ) )
    ( else #f )
  )
)
```

Retirer les duplicates

```
(define (unique L)
  (if (list? L)
      (doUnique L)
      'erreur-de-liste)
  )
(define (doUnique L)
  (cond
    ((null? L) '())
    ((member (car L) (cdr L))
     (doUnique (cdr L)))
    (else (cons (car L)
                 (doUnique (cdr L))))))
  ) )
```

Pile en Scheme version fonctionnelle

```
(define (empty? stack)
  (null? stack)
)
```

```
(define (push e stack)
  (cons e stack)
)
```

```
(define (pop stack)
  (if (empty? stack)
    ()
    (cdr stack)
) )
```

```
(define (top stack)
  (if (empty? stack)
    ()
    (car stack)
) )
```

Minimum d'une liste

```
define (minL x)
  (if (null? x)
      x
      (minL-aux (car x) (cdr x)))
  )
(define (minL-aux Elt x)
  (cond
    ((null? x) Elt)
    ((> Elt (car x))
     (minL-aux (car x) (cdr x)))
    (else (minL-aux Elt (cdr x))))
  )
)
```

Minimum d'une liste: variables locales

```
(define (minL-aux Elt Lst)
  (if (null? Lst)
      Elt
      (let
        ( (v1 (car Lst))
          (v2 (cdr Lst)) )
        (if
          (> Elt v1)
          (minl-aux v1 v2)
          (minl-aux Elt v2)
        )
      )
    )
)
```

Autre exemple de portée locale

```
(define (quadruple x)
  (let ((double (lambda (x) (+ x x))))
    (double (double x)))
  )
> (quadruple 8)
32
> (double 8)
double: undefined;
       cannot reference an identifier before
       its definition
```