Programmation Fonctionnelle

Séance 2 : Map, Fold

Alexandros Singh

Université Paris 8

13 octobre 2023

Problème : Successeurs

Etant donné une liste `l` des entiers naturels, générer une nouvelle liste dont les éléments sont les successeurs de ceux de `l`.

Problème : Successeurs

Etant donné une liste `l` des entiers naturels, générer une nouvelle liste dont les éléments sont les successeurs de ceux de `l`.

```
(define add-1
  (lambda (l)
    (cond
      [(null? l) l]
      [else (cons (+ 1 (car l)) (add-1 (cdr l)))])))
```

Problème : Multiplier par 10

Etant donné une liste `l` des entiers naturels, générer une nouvelle liste dont les éléments sontceux de `l` multipliés par 10.

Problème : Multiplier par 10

Etant donné une liste `l` des entiers naturels, générer une nouvelle liste dont les éléments sontceux de `l` multipliés par 10.

```
(define mul-10
  (lambda (l)
    (cond
      [(null? l) l]
      [else (cons (* 10 (car l)) (mul-10 (cdr l)))])))
```

Problème : Multiplier par 10

Etant donné une liste `l`, générer une nouvelle liste dont l'élément à l'indice n est `#t` si le n-ième élément de `l` est égal à "5", sinon c'est `#f`.

Problème : Multiplier par 10

Etant donné une liste `l`, générer une nouvelle liste dont l'élément à l'indice n est `#t` si le n-ième élément de `l` est égal à "5", sinon c'est `#f`.

```
(define contains-5?
  (lambda (l)
    (cond
      [(null? l) #f]
      [else (or (eq? (car l) 5) (contains-5? (cdr l)))])))
```

Problème : Map

Étant donné une liste `l` et une fonction `f` à un argument, retourner une liste dont les éléments sont les résultats de l'application de `f` à chacun des éléments de `l`.

`'(x y z w)` $\rightarrow$ `'((f x) (f y) (f z) (f w))`

Problème : Map

Étant donné une liste `l` et une fonction `f` à un argument, retourner une liste dont les éléments sont les résultats de l'application de `f` à chacun des éléments de `l`.

```
(define my-map
 (lambda (f l)
   (cond
     [(null? l) l]
     [else (cons (f (car l)) (my-map f (cdr l)))])))
```

Problème : Somme d'une liste

Étant donné une liste `l` d'entiers naturels, retourner la somme de ses éléments.

Problème : Somme d'une liste

Étant donné une liste `l` d'entiers naturels, retourner la somme de ses éléments.

```
(define sum-list
  (lambda (l)
    (cond
      [(null? l) 0]
      [else (+ (car l) (sum-list (cdr l)))])))
```

Problème : Produit d'une liste

Étant donné une liste `l` d'entiers naturels, retourner le produit de ses éléments.

Problème : Produit d'une liste

Étant donné une liste `l` d'entiers naturels, retourner le produit de ses éléments.

```
(define mul-list
  (lambda (l)
    (cond
      [(null? l) 1]
      [else (* (car l) (mul-list (cdr l)))])))
```

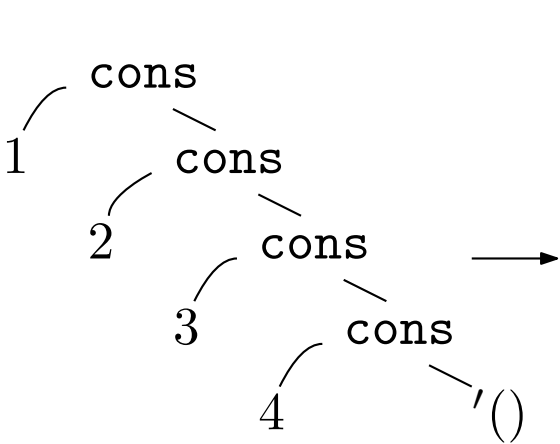
Problème : Recherche dans une liste

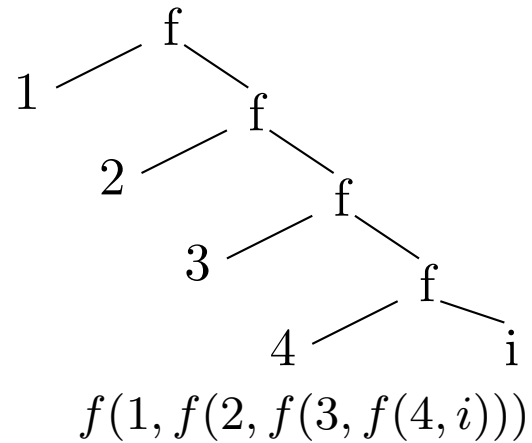
Étant donné une liste `l` d'entiers naturels, retourner vrai si la `l` contient 5, sinon retourner faux.

Problème : Recherce dans une liste

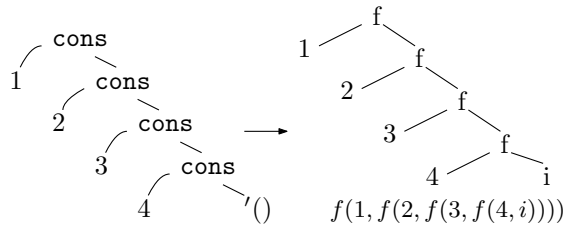
Étant donné une liste `l` d'entiers naturels, retourner vrai si la `l` contient 5, sinon retourner faux.

```
(define contains-5?
  (lambda (l)
    (cond
      [(null? l) #f]
      [else (or (eq? (car l) 5) (contains-5? (cdr l)))])))
```


cons
1
cons
2
cons
3
cons
4
'()


f
1
f
2
f
3
f
4
i
$f(1, f(2, f(3, f(4, i))))$

Fold




```
(define my-fold-right
  (lambda (f i l)
    (cond
      [(null? l) i]
      [else (f (car l) (my-fold-right f i (cdr l)))])))
```