Programmation fonctionnelle
LISP

D.E ZEGOUR
École Supérieure d'Informatique
ESI

Programmation fonctionnelle / LISP (LISt Processor (1960)

Sommaire

Programmation fonctionnelle / LISP

Structure d'un programme

Grammaire

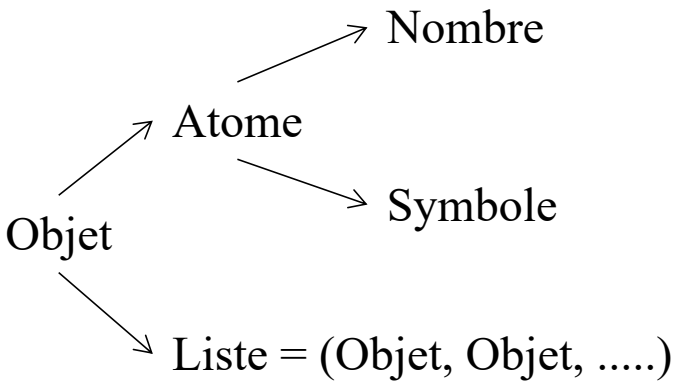
<Liste> --> <atome> | (<Liste> , <Liste>)

<atome> --> Nombre | Symbole

- Programme LISP = Liste (premier terme : fonction, les termes suivants : paramètres d'appel)

Exemple: (ceci est une fonction) ceci : fonction avec les paramètres est, une et fonction

- Tout est liste.



Programmation fonctionnelle / LISP

Génénéralités

- Lisp utilise la notation préfixée

(Sin x) c'est sin(x)

(+ 3 2) c'est 3 + 2.

- Insensible à la casse

- Commentaires commencent par ';'

- Format : libre

- Offre l'affectation et séquentiabilité pour les raisons commerciales (langage fonctionnel non pur)

Programmation fonctionnelle / LISP

Quelques fonctions utiles

Conditionnelle : ***(IF cond liste1 liste2 ..listen)***

Évalue suivant la valeur de Cond soit liste1 soit en "séquentielle" liste2, liste3, ..., listen.

Valeur retournée : dernière forme évaluée.

-->On se limite à l'utilisation de ***(If Cond liste1 liste2).***

Valeurs de vérité :

T pour vrai

Nil et 0 pour faux

T et Nil sont des atomes spéciaux.

Programmation fonctionnelle / LISP

Une fonction utile

(Quote objet) c'est retourner l'objet sans évaluation

Exemple :

(Quote (+ 1 2)) c'est équivalent à (+ 1 2)

Autre écriture:
(' (+ 1 2))

Programmation fonctionnelle / LISP

Quelques fonctions utiles (Primitives de manipulation des listes)

(Car Listenonvide) : fournit la première composante de la liste.

Exemple :

(Car '(1 2 3)) retourne 1

(Car '((a b) 2 (3 c))) retourne (a b)

(Cdr Listenonvide) : fournit listenonvide privée de son premier élément.

Exemples:

(Cdr '(1 2 3)) retourne (2 3)

(Cdr '((a b) 2 (3 c))) retourne (2 (3 c))

(Cdr '(1)) retourne 0

Programmation fonctionnelle / LISP

Quelques fonctions utiles

(Cons objet1 Liste) : retourne une liste dont le premier terme est son premier argument (objet1) et les termes suivants sont ceux du second argument.

Exemples :

(Cons'1 '(2 3 4)) retourne (1 2 3 4)

(Cons '(1 2) '(3 4 5)) retourne ((1 2) 3 4 5)

Propriétés :

(Car (Cons x y)) = x

(Cdr (Cons x y)) = y

Programmation fonctionnelle / LISP

Primitives booléennes

(Atom objet) : retourne T si objet est un atome, Nil autrement.

(NumberP obj) *:* retourne T ssi obj est un nombre, Nil autrement.

(SymbolP Obj) *:* retourne T ssi obj est un symbole, Nil autrement.

(Consp Obj) *:* retourne T ssi Obj est une liste non vide, Nil autrement.

(Null Obj) *:* retourne T ssi Obj est une liste vide, Nil autrement.

(Eq Symb1 Symb2) : teste l'égalité de 2 atomes.

(Equal Obj1 Obj2) : teste l'égalité de 2 objets.

Programmation fonctionnelle / LISP

Définition de nouvelles fonctions :

(DE Nomdefonction (Var1 Var2 ..Varn) Liste1 Liste2 ... Listen)

Retourne Nomdefonction comme valeur.

On se limitera à

(DE Nomdefonction (Var1 Var2 ..Varn) Liste)

```
;Longueur d'une liste
(DE Long(L)
        ( IF(Null L) 0
          (+ (Long ( Cdr L)) 1 )
        )
)
```

Programmation fonctionnelle / LISP

Fonctionnelle (Fonction de fonctions)

(DE f(x) (...)))) : valeur fonctionnelle de f

Cas d'une fonction de fonction : (DE g(f) (...)))

f a deux valeurs :

- une valeur fonctionnelle (celle de la fonction f(x)
- une valeur nominale (provenant de l'appel)

Utilisation de **Funcall** pour désigner la valeur nominale

(Funcall f Arg1 Arg2 ...Argn)

Programmation fonctionnelle / LISP

Fonctionnelle (Fonction de fonctions)

Construire une fonction H qui étant donnée une fonction f et une liste X=(x1 x2 xn), retourne une liste Y=(y1 y2 yn) tel que yi=f(xi)

```
(De H (f x)
(if (consp x)
  (cons (funcall f (car x))
        (H f (cdr x))
)))
```

Exemple:

- Soit f et g : (de f(x) (+ 3 x)) (de g(x) (* 3 x))
- Avec Funcall (H 'g '(1 2 3)) renvoie (3 6 9)
- Sans Funcall (H 'g '(1 2 3)) renvoie (4 5 6).

Programmation fonctionnelle / LISP

Exemples

Somme des éléments d'une liste

```
(DE Som(L)
     (IF (Consp L)
        (+ (Car L)  (Som  (Cdr L))
       0
      )
)
```

Appartenance d'un élément à une liste.

```
(DE App (a L)
   (IF (Null L )  0
      (IF (eq a (Car L) ) T
          (App a (Cdr L))
      )
   )
)
```

Programmation fonctionnelle / LISP

Autres exemple

Inverser une liste

```
(De Inverse(L)
   (Cons
      (Dernier L)
      (Inverse(Supprimer(Dernier L) L)
   )
)
```

C'est le dernier concaténé à l'inverse de la liste sans le dernier.

Dernier d'une liste

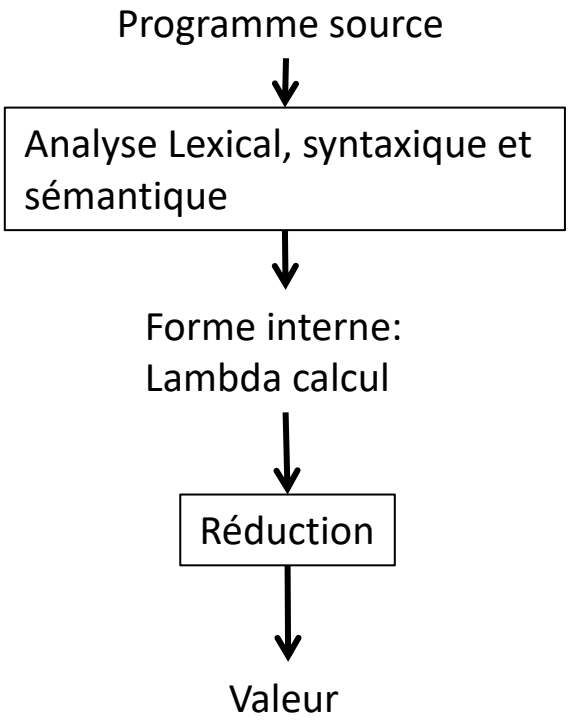
```
(De Dernier(L)
   (IF (Consp L)
      (IF (Null (Cdr L) )
         (Car L)
       (Dernier (Cdr L))  )) )
```

Supprimer un élément d'une liste

```
(DE Supprimer(a L)
  (IF (Consp L)
    (IF (Eq a (Car L))
      (Cdr L)
      (Cons( (Car L) (Supprimer(a (Cdr L)))
     )
    )
)
```

Programmation fonctionnelle / LISP

Interprétation des langages fonctionnels



Forme interne: imbrication de
Listes chaînées

Les listes sont créées a chaque rencontre de parenthèses ouvrantes.

Programmation fonctionnelle / LISP

Traduction vers Lambda-calcul : CAR, CDR et CONS

Premier élément d'une liste

CAR = (λc.c(λa. λb.a))

Liste sans le premier élément

CDR = (λc.c(λa. λb.b))

Concaténation de deux listes

CONS = (λa. λb. λf. fab)

Traduction vers Lambda-calcul :

Nombres

Codage des entiers naturels :

$\underline{0} = \lambda f.\lambda x.x$

$\underline{1} = \lambda f.\lambda x.f\ x$

$\underline{2} = \lambda f.\lambda x.f(f\ x)$

$\underline{3} = \lambda f.\lambda x.f(f(f\ x))$

...

$\underline{n} = \lambda f.\lambda x.f(f(...(f\ x))$

Successeur

$S = \lambda n.\lambda f.\lambda x.f((n\ f)\ x)$

$S(n) = (\lambda nfx.f\ (n\ f\ x))\ \underline{n}$

$\rightarrow \lambda fx.f\ (\underline{n}\ f\ x)$

$= \lambda fx.f\ \ \underline{\lambda f.\lambda x.f(f(...(f\ x))}\ (\ f\ x)$

$= \lambda fx.f\ \ \underline{f(f(...(f\ x))}$

$= n+1$

Addition

$ADD = \lambda mn.\lambda fx.(m\ f)\ ((n\ f)\ x))$

$ADD = \lambda mn.(m\ S)\ n$

Multiplication

$MUL = \lambda mn.m\ (ADD\ n)\ 0$

Traduction vers Lambda-calcul :Booléens

TRUE = λx.λy.x

FALSE = λx.λy.y

IF = λb.λt.λf.b t f

(b : boolean, t et f termes)

AND = λbb'. IF b b' FALSE

OR = λbb'. IF b TRUE b'

NOT = λb.IF b FALSE TRUE

Vérification pour le AND

AND = λbb'. IF b b' FALSE

AND = λbb'. (λb.λt.λf.b t f) b b' FALSE

AND = λbb'. (λt.λf.b t f) b' FALSE

AND = λbb'. (λf.b b' f) FALSE

AND = λbb'. b b' FALSE

Programmation fonctionnelle / LISP

Evaluation Lambda-calcul : Récursivité

Soit f une fonction récursive

f = (λn. (….f …))

f = H f

avec H= (λg .(λn. (…g …))) f

avec

Y =(λh . (λx. h (x x)) (λx.h (x x)))

Evaluation des fonctions récursives:

f Arg = (YH) Arg = H (YH) Arg