utbm

IA41
Initiation à l'Intelligence Artificielle et représentation des connaissances

CM #2

Introduction au LISP

Fabrice LAURI

Plan

- Historique du LISP

- Concepts de base et programmation fonctionnelle

- Autre point de vue sur les listes : représentation sous forme d'arbres

- Les listes spéciales : les formes (programmes)

- Mécanisme d'évaluation de l'interpréteur LISP

- Fonctions primitives manipulant les listes

- Définition de fonctions utilisateurs

- Notions fondamentales de la récursivité

Prolégomènes

LISP est l'acronyme de LISt Processing. LISP manipule donc des listes, et plus particulièrement des symboles.

Qu'est ce qu'une liste ?

$$\text{liste} := (\{e_1\ e_2\ \ldots\ e_n\})$$

e_n est un élément, défini par :

$$e_n := \text{atome} \mid \text{liste}$$

avec :

$$\text{atome} := \text{nombre} \mid \text{nom}$$

Quelques exemples d'atomes et de listes

- 123 : nombre
- « bonjour » : chaîne de caractères
- () : liste vide
- (ceci est une liste) : liste de longueur 4, profondeur 1
- (ceci (est une) autre (liste)) : liste de longueur 4, profondeur 2

Un autre point de vue sur les listes : représentation sous forme d'arbres

Pour construire un arbre à partir d'une liste, on suppose que :

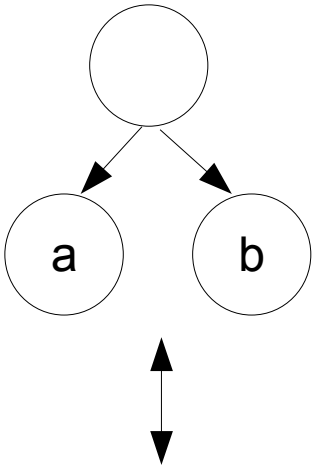
- une liste est représentée par un cercle vide

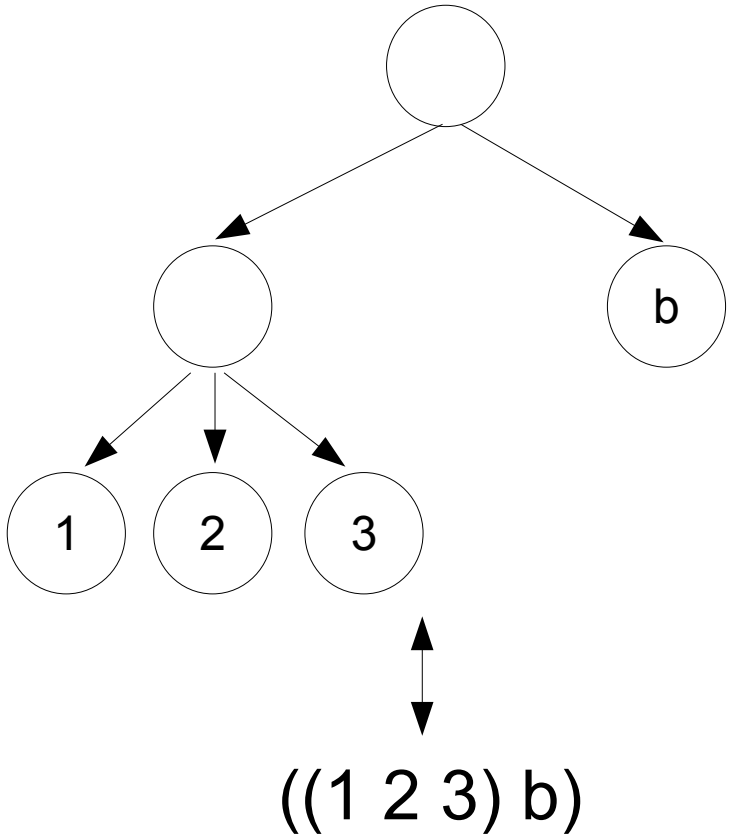
- les éléments d'une liste sont représentés par des cercles connectés directement au cercle de la liste

- la profondeur d'un élément est déterminée par le nombre de flèches que l'on rencontre pour l'atteindre dans l'arbre

La représentation d'une liste sous forme d'arbre permet à la fois de reconnaître graphiquement les éléments et leur profondeur.

Exemple d'arbres et listes correspondantes





Bien sûr, le processus de transformation d'une liste en arbre est réversible. En particulier, les connaissances représentées dans un arbre peuvent également se traduire aisément sous la forme d'une liste.

Les listes spéciales : les formes

Les listes et les atomes sont les objets que vous pouvez manipuler en LISP.
Il existe des listes spéciales, les *formes*, qui indique à l'interpréteur LISP qu'il doit appliquer une fonction sur un ensemble d'arguments.

Une forme est définie par :

forme := (nom-de-fonction {arg1 arg2 ... argn})

Le mécanisme d'évaluation de LISP (1/2)

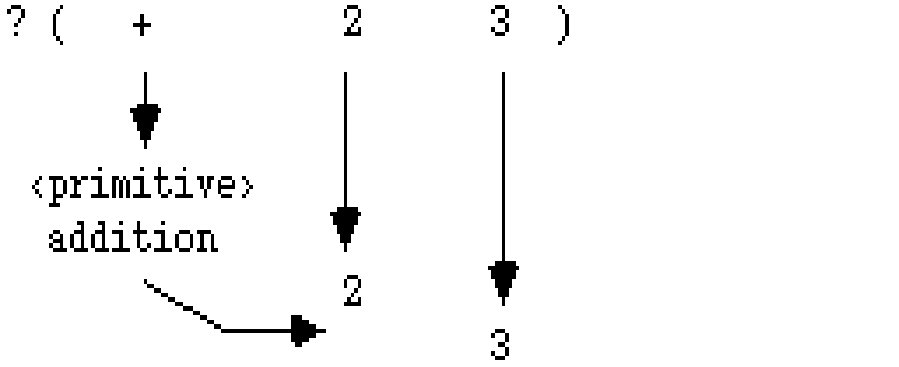
Le mécanisme d'évaluation est systématique :

- si l'expression est un **nombre** ou une **chaîne de caractères**, cette expression est retournée comme valeur.

- si l'expression est une **liste** (commence par une p.o.), l'expression est considérée comme une fonction suivie d'arguments. Le premier élément de la liste (nom de la fonction) est évalué : il doit être une primitive ou une fonction utilisateur. Les éléments suivants de la liste sont évalués de gauche à droite et passés en argument à la fonction.

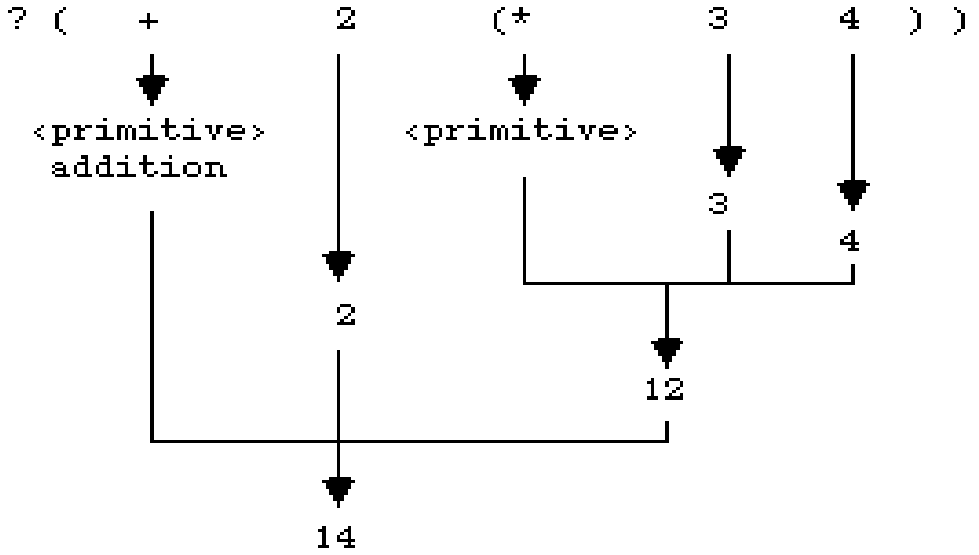
Ce mécanisme s'applique de façon récursive.

Le mécanisme d'évaluation de LISP (2/2)




```
; l'addition appliquée à 2 et 3 retourne 5
5
```

Exemple 1



Exemple 2

Les primitives sur les listes (1/2)

Fonctions sur les listes :

(car l) : tête (premier élément) de la liste l

(cdr l) : reste de la liste l

c...r : combinaison *car* et *cdr* : *(caddr l)* = *(car (cdr (cdr l)))*

(cons e l) : construction d'une liste à partir de l'élément e et de l

(append l1 l2) : concaténation de deux listes l1 et l2

(list e1 e2 ...) : construction d'une liste à partir d'éléments

(length l) : nombre d'éléments de la liste l

Les primitives sur les listes (2/2)

*(**quote** l)* : bloque l'évaluation d'une expression

Exemples : (quote (a b c)) → (a b c)
(quote (+ 1 3 5)) → (+ 1 3 5)
'(+ 1 3 5) → (+ 1 3 5)

*(**eval** liste)* : force l'évaluation d'une expression. Retourne la valeur de l'évaluation de la valeur d'un argument.

Exemples : (cons '+ '(1 3 5)) → (+ 1 3 5) ; programme représenté comme une donnée
(eval (cons '+ '(1 3 5))) → 9 ; évaluation du prog.

Les primitives sur les listes (3/2)

Prédicats sur les listes :

null : argument est-il une liste vide ?

atom : argument est-il un atome ?

consp : argument est-il une liste **non vide** ?

endp : la liste est-elle vide ?

member : argument est-il membre d'une liste ?

Définition de fonctions utilisateurs

Il est possible d'automatiser des traitements symboliques spécifiques en définissant des fonctions utilisateurs.
La fonction *defun* permet de définir une nouvelle fonction.

La définition syntaxique de *defun* est :

(*defun* nom-de-fonction ({var_1 var_2 ... var_n}) corps-de-fonction)

Par exemple, la fonction *quatrieme,* qui retourne le 4ème élément d'une liste, pourra être définie ainsi :

(*defun* quatrieme (l) (car (cdr (cdr (cdr l)))))

Définition de la récursivité

La récursivité est la possibilité de faire apparaître dans la définition d'une entité une référence à elle-même. Dans ce cas, on dit que l'entité possède une définition récursive ou qu'elle est intrinsèquement récursive.

En programmation, on distingue deux types d'entités récursives :

- les fonctions récursives
    - fonction factorielle, fonction puissance
    - fonction de Fibonacci
    - fonctions reverse, append ...

- les structures de données récursives
    - les nombres
    - les listes
    - les arbres
    - les objets fractals

Approche pour la construction d'algorithmes récursifs (1/2)

Un problème se prête bien à l'analyse récursive lorsqu'il peut être décomposé en sous-problèmes de taille plus petite et de même nature que le problème initial.

L'analyse récursive est constituée de trois étapes :

1) paramétrage du problème

2) recherche d'un cas trivial et de sa solution

3) décomposition du cas général

Approche pour la construction d'algorithmes récursifs (2/2)

1) Paramétrage du problème

Il s'agit d'identifier tous les paramètres du problème, en particulier ceux dont la taille décroît à chaque appel récursif.

2) Recherche d'un cas trivial et de sa solution

Un cas trivial est un sous-problème qui peut être résolu **sans** appel récursif. Il correspond souvent au cas où la taille est nulle.

3) Décomposition du cas général

Cette étape a pour but de ramener le problème donné à l'instant t vers un ou plusieurs sous-problèmes que l'on suppose déjà traités à des instants précédents et de taille plus petite.

Application de l'approche pour la définition de la fonction factorielle

Par exemple, définir la fonction factorielle revient à spécifier les trois étapes suivantes :

1) Paramétrage de la fonction

$$factorielle : N \rightarrow N$$

2) Cas trivial

$$Pour\ n < 2,\ factorielle\ n = 1$$

3) Cas général (appel récursif)

$$Pour\ n >= 2,\ factorielle\ n = n * factorielle\ n-1$$

Exécution d'une fonction récursive exemple avec la fonction factorielle