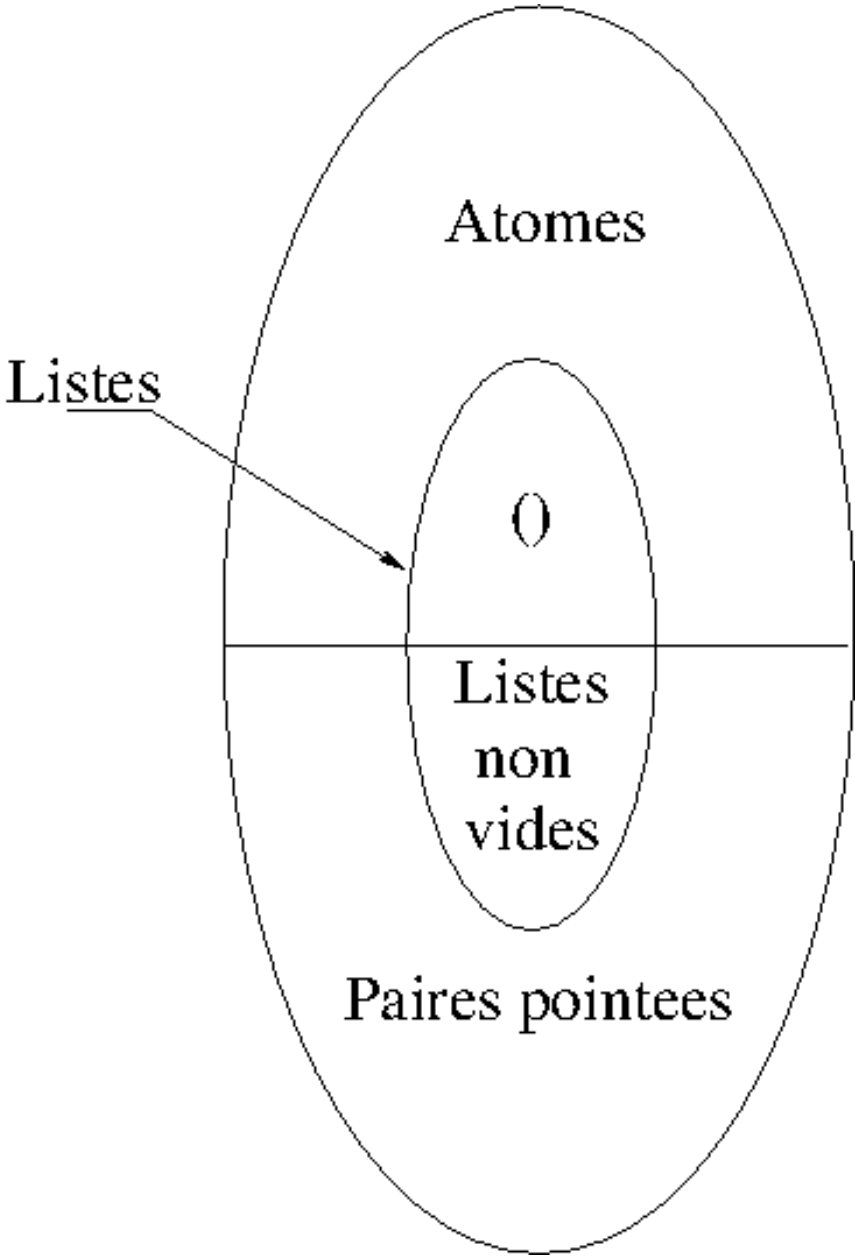
Programmation fonctionnelle - PG104 -

COURS 5

Myriam Desainte-Catherine et David Renault

February 3, 2020

Les objets Scheme



Les objets

Les atomes

- Numériques,
- Booléens,
- Symboles,
- Chaînes de caractères
- Liste vide : ()
- Prédicat : **null**?

Les paires pointées

- Constructeur : **cons**
- Accesseurs : car, cdr
  (argument paire pointée uniquement)
- Prédicats : pair?, **cons**?
- Abbréviation : cddr, caadr, ..., cadddr, ..., list−ref

Forme quote et symboles

```
> (quote Pierre)
'Pierre
>'Pierre
'Pierre
> (define a 'Pierre)
> a
'Pierre
> Pierre
Error : Pierre undefined;
> '(define a 'Pierre)
'(define a 'Pierre)
```

Exemples

```
> (cons 1 2)
'(1 . 2)
> (cons (cons (cons 1 2) 3) 4)
'(((1 . 2) . 3) . 4)
> (cons 1 (cons 2 (cons 3 4)))
'(1 2 3 . 4)
> (car (cons 1 2))
1
> (cdr (cons 3 (cons 1 4))
'(1 . 4)
```

Affichage des paires pointées

- (a . (pp)) $\longrightarrow$ (a pp) si pp est une paire pointée.
- (a . ()) $\longrightarrow$ (a)

Exemple

```
> '(1 . (2 3))
'(1 2 3)
> (define f '(define (f x) x))
> f
'(define (f x) x)
> (cons '(1 2) 3)
'((1 2) . 3)
```

Utilisation de cons, car et cdr

- Quel est le résultat de l'expression (cons 1 (cons 2 (cons 3 '()))) ?
  - '(1 2 3)
  - '(1 . (2 . (3 . ())))
  - '(1 . (2 . (3 . '())))
  - '(1 2 . 3)
  - '(((1 . 2) . 3) . ())
- Soit l'expression (cons 1 (cons 2 (cons 3 '()))). Quelle expression permet d'accéder à l'élément 2?
  - (car (cons 1 (cons 2 (cons 3 '()))))
  - (cdr (car (cons 1 (cons 2 (cons 3 '())))))
  - (car (cdr (cons 1 (cons 2 (cons 3 '())))))

Utilisation de cons, car et cdr

- Quel est le résultat de l'expression (cons 1 (cons 2 (cons 3 '()))) ?
  - '(1 2 3)
- Soit l'expression (cons 1 (cons 2 (cons 3 '()))). Quelle expression permet d'accéder à l'élément 2?
  - (car (cons 1 (cons 2 (cons 3 '()))))
  - (cdr (car (cons 1 (cons 2 (cons 3 '())))))
  - (car (cdr (cons 1 (cons 2 (cons 3 '()))))

Réponses

- Quel est le résultat de l'expression (cons 1 (cons 2 (cons 3 '()))))?
  - '(1 2 3)
- Soit l'expression (cons 1 (cons (2 (cons 3 '())))). Quelle expression permet d'accéder à l'élément 2?
  - (car (cdr (cons 1 (cons 2 (cons 3 '())))))

Les listes

Définition récursive des listes Scheme

- Liste vide : '() ou **null**
- Une paire pointée dont le car est un élément de la liste, et le cdr est une liste;

Autres types de structures

- Liste impropre : une liste qui ne se termine pas par la liste vide.
- Liste circulaire : une chaîne de cons sans fin;

Ces autres types de structures ne sont pas des listes

Définitions formelles

Atome

atome ::= number | symbol | string | ()

Paire pointée

paire-pointée ::= (objet . objet)

Objet

objet ::= atome | paire-pointée

Liste

liste ::= () | (objet . liste)

Liste impropre

liste-impropre ::= () | paire-pointée

Fonctions de base sur les listes

- Prédicats **list**?, empty? et **null**?
- Prédicats d'égalité : **eq**?, **equal**?
- Fonction de construction : **list**, (voir aussi **list***), make-list
- Fonctions prédéfinies : **length**, list-ref, list-tail, **append**, **reverse**, **member**, **remove**, **first**, ... **tenth**, **nth**, **rest**, **last**, last-pair, take, drop, split-at, take-right, drop-right, split-at-right, flatten, **remove***, remove-duplicates, range, shuffle, permutations, remv, remq, memv, memq.
- Fonctions de a-listes : assq, **assoc**

Exemple

```
> (define s1 '(1 2 3))
> (define s2 '(1 2 3))
> (define s3 s1)
> (eq? s1 s2)
#f
```

Exemple

```
> (equal? s1 s2)
#t
> (eq? s1 s3)
#t
> (equal? s1 s3)
#t
```

Construction de listes

- quote : aucun élément de la liste n'est évalué

```
> (quote a b c)
'(a b c)
```

```
> '(1 (* 1 2) 3)
'(1 (* 1 2) 3)
```

- **list** : tous les éléments sont évalués

```
> (list 1 2 3)
'(1 2 3)
```

```
> (list 1 (* 1 2) 3)
'(1 2 3)
```

- **cons** : tous les arguments sont évalués (pour rappel)

```
> (cons (* 1 2) '(1 2 3))
'(2 1 2 3)
```

- **list** $*$: (**list** $*$ 1 2 3) $\longrightarrow$ '(1 2 . 3)
- make−list: (make−list 3 1) $\longrightarrow$ '(1 1 1)
- range : intervalle

```
> (range 10)
'(0 1 2 3 4 5 6 7 8 9)
> (range 10 20 2)
'(10 12 14 16 18)
```

(range end)
(range start end [step])

Construction de listes

Soit les définitions suivantes

```
(define a 0)
(define b 10)
(define c 3)
```

Quels sont les résultats des expressions suivantes?

- (cons a (cons b (cons c '()))) : (a b c) ou (0 10 3)
- (list a b c) : (a b c) ou (0 10 3)
- (list 'a b c) : (a b c) ou (0 10 3) ou (a 10 3) ou (0 b c)
- '(a b c) : (a b c) ou (0 10 3)
- (make-list 3 a) : (a a a) ou (0 0 0)

Construction de listes

Soit les définitions suivantes

```
(define a 0)
(define b 10)
(define c 3)
```

Quels sont les résultats des expressions suivantes?

- (cons a (cons b (cons c '()))) : (0 10 3)
- (list a b c) : (a b c) ou (0 10 3)
- (list 'a b c) : (a b c) ou (0 10 3) ou (a 10 3) ou (0 b c)
- '(a b c) : (a b c) ou (0 10 3)
- (make-list 3 a) : (a a a) ou (0 0 0)

Construction de listes

Soit les définitions suivantes

```
(define a 0)
(define b 10)
(define c 3)
```

Quels sont les résultats des expressions suivantes?

- (cons a (cons b (cons c '()))) : (0 10 3)
- (list a b c) : (0 10 3)
- (list 'a b c) : (a b c) ou (0 10 3) ou (a 10 3) ou (0 b c)
- '(a b c) : (a b c) ou (0 10 3)
- (make-list 3 a) : (a a a) ou (0 0 0)

Construction de listes

Soit les définitions suivantes

```
(define a 0)
(define b 10)
(define c 3)
```

Quels sont les résultats des expressions suivantes?

- (cons a (cons b (cons c '()))) : (0 10 3)
- (list a b c) : (0 10 3)
- (list 'a b c) : (a 10 3)
- '(a b c) : (a b c) ou (0 10 3)
- (make-list 3 a) : (a a a) ou (0 0 0)

Construction de listes

Soit les définitions suivantes

```
(define a 0)
(define b 10)
(define c 3)
```

Quels sont les résultats des expressions suivantes?

- (cons a (cons b (cons c '()))) : (0 10 3)
- (list a b c) : (0 10 3)
- (list 'a b c) : (a 10 3)
- '(a b c) : (a b c)
- (make-list 3 a) : (a a a) ou (0 0 0)

Réponses

Soit les définitions suivantes

```
(define a 0)
(define b 10)
(define c 3)
```

Quels sont les résultats des expressions suivantes?

- (cons a (cons b (cons c '()))) : (0 10 3)
- (list a b c) : (0 10 3)
- (list 'a b c) : (a 10 3)
- '(a b c) : (a b c)
- (make-list 3 a) : (0 0 0)

Fonction **append** : Concaténation de listes

- Fonction n-aire
- Les arguments sont des listes, sauf le dernier qui est un objet quelconque
- La dernière paire pointée de l'argument n est remplacée par la première de l'argument $n+1$
- Sans effets de bord : recopie des paires pointées de toutes les listes en argument (avec partage de tous leurs éléments), sauf le dernier argument qui est partagé.

Exemple

```
> (append '(1 (2 . 3)) '(1))
'(1 (2 . 3) 1)
> (append '() '())
'()
> (append '(1) '(() 2) 3)
'(1 () 2 . 3)
```

Différencier cons, list et append

Quels sont les résultats des expressions suivantes?

- (cons 1 '(a b)) : (1 (a b)) ou (1 a b) ou ((1) a b) ou Erreur
- (cons '(a b) 1) : ((a b) 1) ou (a b 1) ou ((a b) . 1) ou Erreur
- (list 1 '(a b)) : (1 (a b)) ou (1 a b) ou ((1) a b) ou Erreur
- (list '(a b) 1) : ((a b) 1) ou (a b 1) ou ((a b) . 1) ou Erreur
- (append '(1) '(a b)) : (1 (a b)) ou (1 a b) ou ((1) a b) ou Erreur
- (append '(a b) 1) : ((a b) 1) ou (a b 1) ou (a b . 1) ou Erreur
- (append 1 '(a b)) : (1 (a b)) ou (1 a b) ou ((1) a b) ou Erreur

Différencier cons, list et append

Quels sont les résultats des expressions suivantes?

- (cons 1 '(a b)) : (1 a b)
- (cons '(a b) 1) : ((a b) 1) ou (a b 1) ou ((a b) . 1) ou Erreur
- (list 1 '(a b)) : (1 (a b)) ou (1 a b) ou ((1) a b) ou Erreur
- (list '(a b) 1) : ((a b) 1) ou (a b 1) ou ((a b) . 1) ou Erreur
- (append 1 '(a b)) : (1 (a b)) ou (1 a b) ou ((1) a b) ou Erreur
- (append '(a b) 1) : ((a b) 1) ou (a b 1) ou (a b . 1) ou Erreur
- (append '(1) '(a b)) : (1 (a b)) ou (1 a b) ou ((1) a b) ou Erreur

Différencier cons, list et append

Quels sont les résultats des expressions suivantes?

- (cons 1 '(a b)) : (1 a b)
- (cons '(a b) 1) : ((a b) . 1)
- (list 1 '(a b)) : (1 (a b)) ou (1 a b) ou ((1) a b) ou Erreur
- (list '(a b) 1) : ((a b) 1) ou (a b 1) ou ((a b) . 1) ou Erreur
- (append 1 '(a b)) : (1 (a b)) ou (1 a b) ou ((1) a b) ou Erreur
- (append '(a b) 1) : ((a b) 1) ou (a b 1) ou (a b . 1) ou Erreur
- (append '(1) '(a b)) : (1 (a b)) ou (1 a b) ou ((1) a b) ou Erreur

Différencier cons, list et append

Quels sont les résultats des expressions suivantes?

- (cons 1 '(a b)) : (1 a b)
- (cons '(a b) 1) : ((a b) . 1)
- (list 1 '(a b)) : (1 (a b))
- (list '(a b) 1) : ((a b) 1) ou (a b 1) ou ((a b) . 1) ou Erreur
- (append 1 '(a b)) : (1 (a b)) ou (1 a b) ou ((1) a b) ou Erreur
- (append '(a b) 1) : ((a b) 1) ou (a b 1) ou (a b . 1) ou Erreur
- (append '(1) '(a b)) : (1 (a b)) ou (1 a b) ou ((1) a b) ou Erreur

Différencier cons, list et append

Quels sont les résultats des expressions suivantes?

- (cons 1 '(a b)) : (1 a b)
- (cons '(a b) 1) : ((a b) . 1)
- (list 1 '(a b)) : (1 (a b))
- (list '(a b) 1) : ((a b) 1)
- (append 1 '(a b)) : (1 (a b)) ou (1 a b) ou ((1) a b) ou Erreur
- (append '(a b) 1) : ((a b) 1) ou (a b 1) ou (a b . 1) ou Erreur
- (append '(1) '(a b)) : (1 (a b)) ou (1 a b) ou ((1) a b) ou Erreur

Différencier cons, list et append

Quels sont les résultats des expressions suivantes?

- (cons 1 '(a b)) : (1 a b)
- (cons '(a b) 1) : ((a b) . 1)
- (list 1 '(a b)) : (1 (a b))
- (list '(a b) 1) : ((a b) 1)
- (append 1 '(a b)) : Erreur Les arguments, autres que le dernier, doivent être des listes
- (append '(a b) 1) : ((a b) 1) ou (a b 1) ou (a b . 1) ou Erreur
- (append '(1) '(a b)) : (1 (a b)) ou (1 a b) ou ((1) a b) ou Erreur

Différencier cons, list et append

Quels sont les résultats des expressions suivantes?

- (cons 1 '(a b)) : (1 a b)
- (cons '(a b) 1) : ((a b) . 1)
- (list 1 '(a b)) : (1 (a b))
- (list '(a b) 1) : ((a b) 1)
- (append 1 '(a b)) : Erreur Les arguments, autres que le dernier, doivent être des listes
- (append '(a b) 1) : (a b . 1)
- (append '(1) '(a b)) : (1 (a b)) ou (1 a b) ou ((1) a b) ou Erreur

Réponses

Quels sont les résultats des expressions suivantes?

- (cons 1 '(a b)) : (1 a b)
- (cons '(a b) 1) : ((a b) . 1)
- (list 1 '(a b)) : (1 (a b))
- (list '(a b) 1) : ((a b) 1)
- (append 1 '(a b)) : Erreur Les arguments, autres que le dernier, doivent être des listes
- (append '(a b) 1) : (a b . 1)
- (append '(1) '(a b)) : (1 a b)

Filtrage de listes : fonction **remove**

La fonction **remove** prend en arguments un élément et une liste et elle renvoie la liste privée de la 1ère occurrence de l'élément. Elle admet un 3ième argument optionnel, qui est le prédicat de test de l'égalité. Par défaut c'est **equal?** qui est utilisé.

Exemple avec le prédicat par défaut

```
> (remove 2 '(1 2 3 2 4))
'(1 3 2 4)
> (remove* '(1 2) '(1 2 3 2 4));Enleve toutes les occurrenc
'(3 4)
```

Exemple avec des prédicats passés en argument

```
> (remove 2 '(1 2 3 4) >)
'(2 3 4)
> (remove 2 '(1 2 3 4) (lambda (x y) (not (= x y))))
```

Devinez le résultat : '(2 3 4) ou '(1) ?

Voir aussi remove-dupliquates, remv, remq

Appartenance à une liste : fonction **member**

La fonction **member** prend en arguments un élément e et une liste l et elle renvoie **#f** si e n'appartient pas à l, et la liste l privée de ses éléments jusqu'à l'occurrence de e si celui-ci apparaît dans la liste. Le prédicat d'égalité utilisé est **equal?**.

Exemples

```
> (member 2 '(1 2 3))
'(2 3)
> (member 3 '(1 2 . 3))
'(1 2 . 3) : not a proper list
> (member 2 '(1 2 . 3))
'(2 . 3)
> (member 5 '(1 2 3 4))
#f
```

- Quel est le résultat de cette expression : `(member 2 '(3 2 1 2 4 5 2))` ?

#t ou #f ou '(2 4 5 2) ou '(2 1 2 4 5 2) ou '(2)

Voir aussi memv, memq, memf

Appartenance à une liste : fonction **member**

Exemple

```
> (member 3 '(3 2 1 2 4 5 2) >)
'(2 1 2 4 5 2)
> (member 3 '(3 2 1 2 4 5 2) <)
'(4 5 2)
```

Devinez le résultat de l'expression suivante :

```
> (member 2 '(3 2 1 2 4 5 2) >)
```

#t ou #f ou '(1 2 4 5 2) ou '(2 1 2 4 5 2)