TD et TP de Scheme: corrigé

Pierre Boulet, Jean-Marc Nicod, Frédéric Vivien

9 – 21 février 1997

1 Évaluation d'expressions

Donner la valeur retournée par les expressions ci-dessous:

1. (+ 2 3) ⇒ 5
2. (+ (* 2 3 5) (- 6 8)) ⇒ 28
3. (and (> 12 45) (=3 (/ 12 4))) ⇒ ()
4. (and (> 12 45) (=3 (/ 12 0))) ⇒ ()
5. (or #t (< 12 3)) ⇒ $\#t$
6. (not #f) ⇒ $\#t$
7. (symbol? 'tigre) ⇒ $\#t$
8. (symbol? "tigre") ⇒ ()
9. Supposez que les définitions suivantes aient été faites dans l'ordre donné:
   - (define puce 0.01)
   - (define baleine 99999)
   - (define bestiole 'baleine)
   - (define dauphin 500)
   - (define tigre 'félin)
   - (define animal baleine)

   Trouvez quels seront les résultats des évaluations suivantes:
   - baleine ⇒ 99999
   - 'baleine ⇒ baleine
   - (+ animal puce) ⇒ 99999.01

- dauphin ⇒ 500
- puce ⇒ 0.01
- bestiole ⇒ baleine
- tigre ⇒ félin
- 'tigre ⇒ tigre
- "animal ⇒ (QUOTE animal)
- (+ bestiole baleine) ⇒ error

10. ((lambda (x y z) (+ x y z)) 22 3 9) ⇒ 34
11. ((lambda (f x y) (+ (f x) (f y))) (lambda (x) (* x x)) 3 4) ⇒ 25
12. (car (cdr '(a b c d))) ⇒ *b*
13. (cdr '(a (b c d))) ⇒ ((*bcd*))
14. (cdr '((a b c d))) ⇒ ()
15. (cdr (car (cdr '(a (b c) (d e))))) ⇒ (*c*)

2 Premières fonctions

Écrire:

- une fonction qui rend le nombre entier suivant du nombre entier passé en argument;

```
(define suivant (lambda (n) (+ 1 n)))
```

- une fonction constante égale à 7;

```
(define sept (lambda a 7))
```

- une fonction qui rend le signe d'un nombre (versions avec et sans *cond*;

```
(define signe1
    (lambda (n)
        (if (> n 0)
            '+
            (if (< n 0)
                '-
                0))))

(define signe2
    (lambda (n)
        (cond ((> n 0) '+)
              ((< n 0) '-)
              (else 0))))
```

– une fonction qui renvoie le second élément d'une liste d'au moins deux éléments;

```
(define second
   (lambda (l)
      (car (cdr l))))
```

– une fonction qui rend vrai si et seulement si une liste n'a qu'un seul élément.

```
(define un_seul_1?
   (lambda (l)
      (and (not (null? l)) (null? (cdr l)))))

(define un_seul_2?
   (lambda (l)
      (if (null? l) #f (if (null? (cdr l)) #t #f))))

(define un_seul_3?
   (lambda (l)
      (cond ((null? l) #f)
            ((null? (cdr l)) #t)
            (else #f))))
```

3 Fonctions récursives sur les listes

Écrire une fonction:

– qui calcule la longueur d'une liste;

```
(define longueur
  (lambda (l)
    (if (null? l)
       0
       (+ 1 (longueur (cdr l))))))
```

– qui calcule la somme des éléments d'une liste de nombres;

```
(define somme
  (lambda (l)
    (if (null? l)
       0
       (+ (car l) (somme (cdr l))))))
```

– qui concatène deux listes;

```
(define concat
  (lambda (l1 l2)
    (if (null? l1)
       l2
       (cons (car l1) (concat (cdr l1) l2)))))
```

– qui renverse une liste;

```
(define renverse
  (lambda (l)
    (if (null? l)
        l
        (concat (renverse (cdr l)) (list (car l))))))
```

– qui rend le dernier élément d'une liste non vide;

```
(define dernier
  (lambda (l)
    (if (null? (cdr l))
        (car l)
        (dernier (cdr l)))))
```

– qui prend une liste et un élément en arguments et renvoie *vrai* si et seulement si l'élément appartient à la liste.

```
(define appartient?
  (lambda (l e)
    (cond ((null? l) #f)
          ((= e (car l)) #t)
          (else (appartient? (cdr l) e)))))
```

– qui rend la liste des symboles de la liste argument: (flatten (a (b c) d)) ⇒ (a b c d).

```
(define flatten
  (lambda (l)
    (cond ((null? l) l)
          ((list? (car l)) (concat (flatten (car l)) (flatten (cdr l))))
          (else (cons (car l) (flatten (cdr l)))))))
```

4 Questions subsidiaires

– Fonction qui insère un entier dans une liste d'entiers triés par ordre croissant :

```
(define insertion (lambda (x l)
   (cond ((NULL? l)     (cons x '()))
         ((< (car l) x) (cons (car l) (insertion x (cdr l))))
         (else  (cons x l)))))
```

– Fonction qui trie par insertion une liste d'entiers dans l'ordre croissant :

```
(define tri_insertion
   (lambda (l)
      (if (NULL? l)
          l
          (insertion (car l) (tri_insertion (cdr l))))))
```