ISIMA 1ère année

vendredi 8 février 2002
Durée : 2 heures
Documents autorisés

PROGRAMMATION FONCTIONNELLE

Exercice 1 :

Le but de cet exercice est d'écrire une fonction `suite` qui a comme argument une liste d'entiers `l` et qui retourne une liste construite à partir de `l` de la manière suivante :
si `l` est la liste vide `suite` retourne la liste vide ;
si `l` est la liste `(1)`, `suite` retourne la liste `(1 1)` car `l` contient un 1 ;
si `l` est la liste `(1 1)`, `suite` retourne la liste `(2 1)` car `l` contient deux 1 ;
si `l` est la liste `(2 1)`, `suite` retourne la liste `(1 2 1 1)` car `l` contient un 2 suivi de un 1 ;
si `l` est la liste `(1 2 1 1)`, `suite` retourne la liste `(1 1 1 2 2 1)` car `l` contient un 1, suivi de un 2, suivi de deux 1 ;
1°) Ecrire une fonction `f` qui a comme argument une liste non vide d'entiers `l` et qui retourne une liste de deux sous-listes construite à partir de `l` de la manière suivante : si les *n* premiers éléments de `l` sont tous égaux à un entier *k*, la première sous-liste sera (*n k*), et la seconde sera la liste `l` tronquée de ces *n* premiers éléments.
Par exemple `(f '(1 1 1 2 2 1))` doit retourner `((3 1) (2 2 1))`.
2°) En utilisant la fonction `f`, écrire la fonction `suite`.

Solution :
1°)
```
(define f (lambda (l) (g (list 1 (car l)) (cdr l))))
(define g (lambda (c l)
   (if (and l (equal? (cadr c) (car l)))
       (g (cons (+ (car c) 1) (cdr c)) (cdr l))
       (list c l) ) ))
```

2°)
```
(define suite (lambda (l)
      (if l
         (let ((c (f l)))
            (append (car c) (suite (cadr c))) )
         () ) ))
```

Exercice 2 :

1°) Ecrire une fonction `compose` qui a comme arguments deux fonctions d'une seule variable, `f` et `g`, et qui retourne la fonction `fog` : $x \mapsto f(g(x))$.
2°) Ecrire une fonction `trace` qui a comme arguments une fonction d'une seule variable, `f`, et un entier `n`, et qui retourne la liste de fonctions `(Id f` f^2 `...` f^n`)`, où `Id` désigne la fonction identité et f^n désigne la composition de `f` (`n`-1) fois par elle-même.
3°) Ecrire une fonction `applique` qui a comme arguments une liste de fonctions d'une seule variable, `Lf`, et une variable `x` et qui retourne la liste des applications des fonctions de `Lf` à la variable `x`.

Par exemple, l'évaluation de `(applique (trace (lambda (x) (* x x)) 3) 2)` doit retourner la liste `(2 4 16 256)`.

Solution :

1°)
```
(define compose (lambda (f g)
      (lambda (x) (f (g x))) ))
```

2°)
```
(define trace (lambda (f n)
      (let ((Id (lambda (x) x)))
         (if (= n 0)
            (list Id)
             (cons Id (map (lambda (fk) (compose f fk)) (trace f (- n 1))))
         ) ) ))
```

3°)
```
(define applique (lambda (Lf x)
      (map (lambda (f) (f x)) Lf) ))
```

Exercice 3 :

Etant donné un ensemble E, on appelle partition de E tout ensemble P_E de parties de E tel que:

$$\forall A \in P_E, A \neq \emptyset ,$$
$$\forall (A , B) \in (P_E)^2, A \cap B = \emptyset ,$$
$$\bigcup_{A \in P_E} A = E.$$

On note P_E^2 l'ensemble des partitions de E en deux parties.

Cet ensemble est forcément vide si E est de cardinalité inférieure à 2.

Pour $E = \{3,4\}$, on a $P_E^2 = \{\{\{3\},\{4\}\}\}$.

Pour $E = \{2,3,4\}$, on a $P_E^2 = \{ \{\{2,3\},\{4\}\},\{\{3\},\{2,4\}\},\{\{2\},\{3,4\}\}\}$.

Pour $E = \{1,2,3,4\}$, on a $P_E^2 = \{ \{\{1,2,3\},\{4\}\},\{\{1,3\},\{2,4\}\},\{\{1,2\},\{3,4\}\},$
$\{\{2,3\},\{1,4\}\},\{\{3\},\{1,2,4\}\},\{\{2\},\{1,3,4\}\},$
$\{\{1\},\{2,3,4\}\}\}$.

Ecrire une fonction `P2` qui a comme argument une liste `E`, représentant un ensemble E, et telle que l'évaluation de l'expression `(P2 E)` retourne la liste représentant l'ensemble des partitions en deux parties de l'ensemble E.

Solution :

```
(define P2 (lambda (E)
   (cond ((or (null? E) (null? (cdr E))) ())
         ((null? (cddr E)) (list (list (list (car E)) (cdr
E))))
         (#t (let ((Pcdr (P2 (cdr E))))
             (append (map (lambda (P)
                          (cons (cons (car E) (car P)) (cdr
P)) )
                          Pcdr )
                     (map (lambda (P)
                          (cons (car P) (list (cons (car E)
(cadr P)))) )
                          Pcdr )
                     (list (list (list (car E)) (cdr E))) ) ))
) ))
```

Exercice 4 :

Ecrire une macro `maplet` qui a comme arguments un symbole `x`, une liste `L`, et une expression fonctionnelle `E`, dans laquelle figure le symbole `x`, et qui retourne la liste des évaluations de l'expression `E` obtenues en remplaçant dans `E` le symbole `x` par chacune des valeurs de la liste `L`.
Par exemple, l'évaluation de `(maplet x (1 2 3 4) (* x x))` doit retourner `(1 4 9 16)`.

Solution :

```
(define-macro (maplet x L E)
   `(map (lambda (,x) ,E) ',L) )
```