ISIMA 1ère année

8 septembre 2003
Durée : 2 heures
Documents autorisés

PROGRAMMATION FONCTIONNELLE

Exercice 1 :

1°) Le but de cette question est d'écrire une fonction `permut` qui a comme argument une liste `l` et qui retourne la liste de toutes les permutations possibles des éléments de `l`.
Par exemple, l'évaluation de `(permut ())` doit retourner `(())`,
celle de `(permut '(3))` doit retourner `((3))`,
celle de `(permut '(2 3))` doit retourner `((2 3) (3 2))`,
celle de `(permut '(1 2 3))` doit retourner `((1 2 3) (2 1 3) (2 3 1) (1 3 2) (3 1 2) (3 2 1))`.
a) Ecrire une fonction `distribuer` qui a comme argument une variable `e` et une liste `L` ayant *n* éléments, et qui retourne une liste de *n*+1 sous-listes, la $k^{ème}$ sous-liste étant obtenue en insérant `e` en $k^{ème}$ position dans `L`.
Par exemple, l'évaluation de `(distribuer 1 '(2 3))` doit retourner `((1 2 3) (2 1 3) (2 3 1))`
b) En utilisant la fonction `distribuer`, écrire la fonction `permut`.

2°) Le but de cette question est d'écrire une fonction `permut2` qui a comme argument une liste `l` de taille *n* contenant les *n* premiers entiers non nuls, et qui retourne la liste des permutations σ des éléments de `l` telles que $\forall i \in \{ 1, \ldots, n \}, \sigma(i) \neq i$.
Par exemple, l'évaluation de `(permut '(1))` doit retourner `()`,
celle de `(permut2 '(1 2))` doit retourner `((2 1))`,
celle de `(permut2 '(1 2 3))` doit retourner `((2 3 1) (1 3 2) (3 1 2))`.
a) Que fait la fonction F suivante :

```
(define F (lambda (L P)
   (append-map (lambda (e) (if (P e) (list e) ())) L) )) ?
```

b) Ecrire la fonction `permut2` en utilisant les fonctions `permut` et `F`.

Solution :
1°)
a)

```
(define distribuer (lambda (e L)
      (if L
         (let ((dcdr (distribuer e (cdr L))))
            (cons (cons e L)
                  (map (lambda (d) (cons (car l) d)) dcdr) ) )
         (list (list e)) ) ))
```

b)

```
(define permut (lambda (L)
      (if L
         (let ((pcdr (permut (cdr L))))
            (append-map (lambda (p) (distribuer (car L) p))
                        pcdr ) )
         '(()) ) ))
```

2°) a) La fonction `F` "filtre" les éléments de `L` qui satisfont le prédicat `P`.
b)

```
(define permut2 (lambda (L) (F (permut L) P)))
(define P (lambda (L)
```

```
   (qqs L (lambda (x) (not (equal? x (rang x L))))) ))
(define qqs (lambda (L P)
   (if L (and (P (car L)) (qqs (cdr L) P)) #t) ))
(define rang (lambda (x L)
   (if (equal? x (car L)) 1
       (+ 1 (rang x (cdr L))) ) ))
```

Exercice 2 :

1°) Ecrire une fonction `dilatation` ayant comme argument une liste `L` ne contenant que des `0` et des `1`, et telle que l'évaluation de `(dilatation L)` retourne une liste obtenue à partir de `L` de la manière suivante : tout `0` dont l'élément précédent ou l'élément suivant dans la liste vaut `1` est transformé en un `1` (on considérera, par défaut, que l'élément précédent le premier élément ainsi que l'élément suivant le dernier élément sont tous deux des `0`).
Par exemple l'évaluation de `(dilatation '(0 0 1 1 0 0 1 1 0 0 0 1 1 0 0))`
doit retourner `(0 1 1 1 1 1 1 1 1 0 1 1 1 1 0)`.

Solution :

```
(define dilatation (lambda (L)
   (if (and L (cdr L))
      (if (= (car L) 0)
         (if (= (cadr L) 0)
            (cons 0 (dilatation (cdr L)))
            (cons 1 (dilatation (cdr L))) )
         (cons 1 (cons 1 (cdr (dilatation (cdr L))))) )
      L ) ))
```

2°) Ecrire une fonction `nbiter` ayant comme argument une liste `L` ne contenant que des `0` et des `1`, et telle que l'évaluation de `(nbiter L)` retourne, si ce nombre est fini, le nombre minimum d'appels successifs de la fonction `dilatation` sur la liste `L` permettant d'obtenir une liste ne comportant que des `1`, et sinon le symbole `infini`. Si par exemple l'évaluation de `(dilatation L)` retourne une liste comportant encore des `0` mais que l'évaluation de `(dilatation (dilatation L))` retourne une liste ne comportant que des `1`, alors l'appel de `(nbiter L)` doit retourner `2`.