ISIMA 1ère année

vendredi 8 février 2003
Durée : 2 heures
Documents autorisés

PROGRAMMATION FONCTIONNELLE

• ***Expliquez brièvement, mais le plus clairement possible, le principe des fonctions que vous concevez, car une explication juste, même brève, vaudra toujours plus qu'une fonction fausse et sans explication !***

Exercice 1 :

Ecrire une fonction `takewhile` ayant comme arguments une liste `L` et un prédicat `P` et telle que l'évaluation de l'expression `(takewhile L P)` retourne le segment initial de `L` "vérifiant" `P`, c'est-à-dire la plus grande sous liste des éléments `x` de `E` tels qu'aucun autre élément de `E` ne satisfaisant pas `P` soit placé avant `x` dans la liste `L`.
Par exemple, l'évaluation de `(takewhile '(1 2 3 a 4 5) integer?)` doit retourner `(1 2 3)`.
Solution :

```
(define takewhile (lambda (L P)
   (if (and L (P (car L)))
      (cons (car L) (takewhile (cdr L) P))
      () ) ))
```

Exercice 2 :

1°) a) Ecrire une fonction `fct` ayant comme arguments deux listes `d` et `a` de même longueur, l, et telle que l'évaluation de l'expression `(f d a)` retourne une fonction qui appliquée à tout élément placé en $n^{ième}$ position dans `d` (avec $1 \leq n \leq l$) retourne l'élément placé en $n^{ième}$ position dans `a`.
Par exemple l'évaluation de `((fct '(a b c) '(x y z)) 'b)` doit retourner `y`.
Solution :

```
(define fct (lambda (d a)
   (if (cdr d)
        (let ((fcdr (fct (cdr d) (cdr a))))
             (lambda (x)
                (if (equal? x (car d))
                   (car a)
                   (fcdr x) ) ) )
        (lambda (x) (car a)) ) ))
```

b) On suppose que l'on dispose d'une fonction `PC` ayant comme arguments une liste `E` représentant un ensemble, et un entier `n`, et telle que l'évaluation de l'expression `(PC E n)` retourne la liste de tous les `n`-uplets de l'ensemble `E`.
Ecrire, en utilisant les fonctions `fct` et `PC`, une fonction `LF` ayant comme argument deux listes non vides représentant deux ensembles non vides `E` et `F`, et telle que l'évaluation de l'expression `(LF E F)` retourne la liste de toutes les applications de `E` dans `F` (une application de `E` dans `F` est une fonction de `E` dans `F` pour laquelle chaque élément de `E` a une image).
Indication : si `E` est un ensemble de cardinalité n, toute fonction de `E` dans `F` est caractérisée par un n-uplet d'éléments de `F`, chaque terme en $k^{ième}$ position d'un tel n-uplet étant l'image du $k^{ième}$ élément de `E`.
Solution :

```
(define LF (lambda (E F)
   (let* ((n (length E)) (LP (PC F n)))
   (map (lambda (P) (fct E P)) LP) ) ))
```

2°) a) Soit `SR` le schéma suivant :

```
(define SR (lambda (L B C)
   (if L
      (C (car L) (SR (cdr L) B C))
       B ) ))
```

En utilisant le schéma `SR`, écrire une fonction `filtrer` ayant comme arguments une liste `E` et un prédicat `P`, et telle que l'évaluation de l'expression `(filtrer E P)` retourne la liste des éléments de `E` satisfaisant `P`.
Par exemple, l'évaluation de `(filtrer '(1 a 2 b) integer?)` doit retourner la liste `(1 2)`.
Solution :

```
(define filtrer (lambda (E P)
```

```
    (SR E ()
       (lambda (t q)
          (if (P t)
              (cons t q)
               q ) ) ) ))
```

b) Soit `qqs?` le schéma suivant :

```
(define qqs? (lambda (L P)
   (if L
      (and (P (car L)) (qqs? (cdr L) P))
       #t ) ))
```

En utilisant le schéma `qqs?`, écrire un prédicat `stabilise?` ayant comme arguments une fonction `f` et une liste `P` représentant un ensemble, et telle que l'évaluation de l'expression `(stabilise? f P)` retourne une valeur logique vraie si `f` stabilise `P` (c'est-à-dire si l'image de tout élément de `P` par `f` appartient à `P`), et une valeur logique fausse sinon.
On pourra utiliser la fonction prédéfinie `member?` Ayant comme argument un élément `x` et une liste `L` et telle que l'évaluation de l'expression `(member? X L)` retourne une valeur logique vraie si `x` est élément de `L`, et une valeur logique fausse sinon.
<u>Solution</u> :

```
(define stabilise? (lambda (f P)
   (qqs P (lambda (x) (member? (f x) P))) ))
```

c) Ecrire une fonction `LFS` ayant comme arguments une liste `E` représentant un ensemble, et une liste `P` représentant une partie de `E`, et telle que l'évaluation de l'expression `(LFS E P)` retourne la liste des fonctions de `E` dans `E` qui stabilisent `P`.
<u>Solution</u> :

```
(define LFS (lambda (E P)
   (filtrer (LF E E) (lambda (f) (stabilise? f P))) ))
```

<u>Exercice 3</u> :

Le principe de représentation d'un nombre entier dans la théorie du λ-calcul (représentation dite des *nombres de Church*) est le suivant : un entier $n > 1$ est représenté par une fonction qui, à toute fonction $f : x \quad f(x)$, associe la composition de f, $n - 1$ fois par elle même, c'est-à-dire,

$$\underset{1}{f} \circ \underset{2}{f} \circ \ \ldots \circ \underset{n}{f} \quad .$$

Par exemple l'entier 2 est représenté par la fonction $deux : f \quad deux\,(f) = f \circ f$.
On a donc : $deux\,(f) : x \quad [\, deux\,(f)\,]\,(x) = [\, f \circ f\,]\,(x) = f\,[\, f\,(x)\,]$.

Par généralisation :

- l'entier 1 est représenté par la fonction : $f \quad f$;
- l'entier 0 est représenté par la fonction : $f \quad I$, où I représente la fonction identité.

1°) Ecrire les fonctions `zero`, `un`, et `deux`, représentant respectivement les entiers 0, 1 et 2.
<u>Solution</u> :

```
(define zero (lambda (f)
   (lambda (x) x) ))
(define un (lambda (f) f))
(define deux (lambda (f)
   (lambda (x) (f (f x))) ))
```

2°) Ecrire la fonction `trois`, représentant l'entier 3, en utilisant la fonction `deux`.
<u>Solution</u> :

```
(define trois (lambda (f)
   (lambda (x) (f ((deux f) x))) ))
```

ou bien :

```
(define trois (lambda (f)
   (lambda (x) ((deux f) (f x))) ))
```

3°) En déduire une fonction `succ` ayant comme argument la représentation de Church d'un entier n, `rn`, et telle que l'évaluation de `(succ rn)` retourne la représentation de Church de l'entier $n + 1$.

<u>Solution</u> :

```
(define succ (lambda (rn)
   (lambda (f)
      (lambda (x) (f ((rn f) x))) ) ))
```

ou bien :

```
(define succ (lambda (rn)
   (lambda (f)
      (lambda (x) ((rn f) (f x))) ) ))
```

4°) Ecrire une fonction `nom` ayant comme argument la représentation de Church d'un entier n, `rn`, et une liste de symboles, `listenoms` (supposée de longueur au moins égale à $n + 1$), telle que l'évaluation de l'expression `(nom rn listenoms)` retourne le $(n + 1)^{\text{ème}}$ élément de `listenoms`.

Par exemple l'évaluation de `(nom trois '(Z I II III IV V VI))` retourne `III`.

<u>Solution</u> :

```
(define nom (lambda (rn listenoms)
   (car ((rn cdr) listenoms)) ))
```

5°) Ecrire une fonction `entier` ayant comme argument un symbole, `nom`, et une liste de symboles, `listenoms`, et retournant la représentation de Church de l'entier $n - 1$ si `nom` figure en $n^{\text{ième}}$ position dans `listenoms`, et `()` sinon.

Par exemple l'évaluation de `(entier 'III '(Z I II III IV V VI))` retourne la fonction qui à toute fonction f associe $f \circ f \circ f$.

On pourra utiliser les fonctions `zero` et `succ` définies ci-dessus.

<u>Solution</u> :

```
(define entier (lambda (nom listenoms)
   (if listenoms
      (if (equal ? nom (car listenoms))
          zero
         (succ (entier nom (cdr listenoms))) )
      () ) ))
```

6°) a) Ecrire une fonction `plus` ayant comme arguments les représentations de Church de deux entiers n et m, respectivement `rn` et `rm`, et telle que l'évaluation de l'expression `(plus rn rm)` retourne la représentation de Church de l'entier $n + m$.

<u>Solution</u> :

```
(define plus (lambda (rn rm)
      (lambda (f)
         (lambda (x) ((rn f) ((rm f) x))) ) ))
```

ou mieux :

```
(define plus (lambda (rn rm)
   ((rn succ) rm) ))
```

b) Que fait la fonction suivante ? Justifiez votre réponse.

```
 (define A (lambda (rn rm)
         (lambda (f)
            (lambda (x) (((rn rm) f) x) ) ))
```

<u>Solution</u> :

`rm` est la fonction : $f \quad f \circ \ldots \circ f$, m fois, c'est-à-dire f^m, donc `rm` $\circ$ `rm` est la fonction :

$$f \quad f^m \circ \ldots \circ f^m, \ m \text{ fois, c'est-à-dire } f^{m^2},$$

et donc par récurrence, `(rn rm)` est la fonction : $f \quad f^{m^n}$, donc `(A rn rm)` retourne m^n.