Département d'Informatique
LICENCE 2ème année
Vendredi 9 janvier 2009

Corrigé du contrôle terminal de Programmation Fonctionnelle

I Récursivité et filtrage

A

1. `int * (int * float) list`
2.
```
let rec cherche k l = match(l) with
    [] -> 0.
|  (i,v)::r -> if i=k then v else cherche k r;;

let composante (n,l) i = if i>n or i<1
    then failwith "erreur"
    else cherche i l;;
```
3.
```
let rec add_aux l1 l2 = match(l1,l2) with
       ([],_) -> l2
     | (_,[]) -> l1
     | ((i1,v1)::r1,(i2,v2)::r2) ->
          if i1=i2 then let v = v1+.v2 in
                              if v = 0. then add_aux r1 r2
                                        else (i1,(v1+.v2))::(add_aux r1 r2)
                       else if i1<i2 then (i1,v1)::(add_aux r1 l2)
                                     else (i2,v2)::(add_aux l1 r2);;

let add (n1,l1) (n2,l2) = if n1<>n2
    then failwith "erreur"
    else (n1,add_aux l1 l2);;
```

B

1. `(int * int) * (int * (int * float) list) list`
2. `let transpose_ligne i = List.map (function (j,v) -> (j,[(i,v)]));;`

```
let rec ajoute_colonne l1 l2 = match(l1,l2) with
  ([],_)           -> l2
| (_,[])           -> l1
| ((i1,e1)::r1,(i2,e2)::r2) -> if i1<i2 then (i1,e1)::(ajoute_colonne r1 l2)
                               else
                            if i1>i2 then (i2,e2)::(ajoute_colonne r1 l2)
                               else
                            ajoute_colonne ((i1,e1@e2)::r1) r2;;

let rec compose l = match(l) with
  [] -> []
| (i,li)::ri -> ajoute_colonne (transpose_ligne i li) (compose ri);;

let transpose ((n,m),l) = ((m,n),(compose l));;
```

II Lambda-calcul

1. $\lambda n.\lambda m.(m\ n)_\curlywedge \lambda f.\lambda x.(f\ x)\ \lambda f.\lambda x.x$
   $\to_\beta \lambda m.(m\ \lambda f.\lambda x.(f\ x))_\curlywedge \lambda f.\lambda x.x$
   $\to_\beta \lambda f.\lambda x.x_\curlywedge \lambda f.\lambda x.(f\ x)$
   $\to_\beta \lambda x.x$

2. $\lambda n.\lambda m.(m\ n)_\curlywedge \lambda f.\lambda x.(f\ (f\ x))\ \lambda f.\lambda x.(f\ x)$
   $\to_\beta \lambda m.(m\ \lambda f.\lambda x.(f\ (f\ x)))_\curlywedge \lambda f.\lambda x.(f\ x)$
   $\to_\beta \lambda f.\lambda x.(f\ x)_\curlywedge \lambda f.\lambda x.(f\ (f\ x))$
   $\to_\beta \lambda x.(\lambda f.\lambda x.(f\ (f\ x))_\curlywedge x)$
   $\to_\alpha \lambda x.(\lambda f.\lambda a.(f\ (f\ a))_\curlywedge x)$
   $\to_\beta \lambda x.\lambda a.(x\ (x\ a))$

3. $\lambda n.\lambda f.\lambda x.(f\ (n\ f\ x))_\curlywedge \lambda f.\lambda x.(f\ x)\ \lambda f.(f\ x)$
   $\to_\beta \lambda f.\lambda x.(f\ (\lambda f.\lambda x.(f\ x)\ f\ x))_\curlywedge \lambda f.(f\ x)$
   $\to_\alpha \lambda f.\lambda a.(f\ (\lambda f.\lambda x.(f\ x)\ f\ a))_\curlywedge \lambda f.(f\ x)$
   $\to_\beta \lambda a.(\lambda f.(f\ x)_\curlywedge (\lambda f.\lambda x.(f\ x)\ \lambda f.(f\ x)\ a))$
   $\to_\beta \lambda a.(\lambda f.\lambda x.(f\ x)_\curlywedge \lambda f.(f\ x)\ a\ x)$
   $\to_\alpha \lambda a.(\lambda f.\lambda b.(f\ b)_\curlywedge \lambda f.(f\ x)\ a\ x)$
   $\to_\beta \lambda a.(\lambda b.(\lambda f.(f\ x)\ b)_\curlywedge a\ x)$
   $\to_\beta \lambda a.(\lambda f.(f\ x)_\curlywedge a\ x)$
   $\to_\beta \lambda a.(a\ x\ x)$