Corrigé de Programmation fonctionnelle

1 Bases

1.

$$\frac{}{\Gamma \vdash \mathtt{A}{:}\alpha\ \mathtt{t}}\mathtt{A} \qquad \frac{\Gamma \vdash s_1{:}\mathtt{int} \qquad \Gamma \vdash e_2{:}\alpha\ \mathtt{t}}{\Gamma \vdash \mathtt{B}(e_1,e_2){:}\alpha\ \mathtt{t}}\mathtt{B}$$

2.
```
#let m = fun n m f x -> n (m f) x;;
val m : ('a -> 'b -> 'c) -> ('d -> 'a) -> 'd -> 'b -> 'c = <fun>
```

Le calcul de s ne fait intervenir que des λ-termes (pas de récursivité) typés. Or tout terme typé possède une forme normale donc le calcul termine.

2 Programmation

1.
```
#let rec enleve = fun x l ->
#  match l with
#    []    -> raise Not_found
#  | y::ys -> if x = y then ys else y::enleve x ys;;
val enleve : 'a -> 'a list -> 'a list = <fun>
```

2.
```
#let rec couples = fun l ->
#  match l with
#    [] -> []
#  | x::xs ->
#     List.fold_right (fun y r -> (x,y)::r) xs (couples xs);;
val couples : 'a list -> ('a * 'a) list = <fun>
```

3.
```
#(* Liste des opérations avec leur nom *)
#let ops = [ (+), "+"; ( * ), "*"; (fun a b -> abs (a - b)), "-";
#  (fun a b -> let a = max a b and b = min a b in
#    if b = 0 || a mod b <> 0 then a else a / b), "/"];;
val ops : ((int -> int -> int) * string) list =
  [<fun>, "+"; <fun>, "*"; <fun>, "-"; <fun>, "/"]

#
#let le_compte_est_bon l n =
#  (* Recherche récursive: 'trace' permet d'accumuler les opérations
#     effectuées pour l'affichage de la solution. 'trace' contient des
#     quadruplets avec les deux opérandes, l'opérateur et le résultat.
#     'l' est la liste des nombres disponibles *)
#  let rec lceb trace l =
#    if List.mem n l then begin
#      (* Le compte est bon: affichage de la solution *)
#      List.iter (fun (x,f,y,z) ->
#                   Printf.printf "%d %s %d = %d\n" (max x y) f (min x y) z)
#                (List.rev trace);
#      raise Exit
#    end;
#    let cs = couples l in
#    (* On tente en séquence tous les couples et opérations possibles *)
#    List.iter     (* Pour tous les couples *)
#      (fun (a, b) ->
#        List.iter      (* Pour toutes les opérations *)
```

```
#          (fun (f, nf) ->
#            let x = f a b in      (* calcul *)
#            let l = enleve b (enleve a l) in  (* Suppression des opérandes *)
#            lceb ((a, nf, b, x)::trace) (x::l))  (* On continue *)
#   ops)
#      cs in
#  try lceb [] l with Exit -> ();;
val le_compte_est_bon : int list -> int -> unit = <fun>
```