PFA - Contrôle des connaissances 2009/2010
Master Informatique - Institut Galilée - Paris 13
J.-V. Loddo

Barème et solutions du contrôle du 11/01/2010
Copies notées sur 21 points

Première partie
Compréhension de code OCaml et typage (9 points)

Q1. (3 points)

(*a*) `['d';'a';'d';'a'] : char list`
(*b*) `('d','a','d','a') : char * char * char * char`
(*c*) `('d',12,"hello",[3.14;6.28]) : char * int * string * float list`
(*d*) `('d','a','d','a') = ('d','o','d','o') : bool`
(*e*) `['d';'a';'d'] = ['d';'o';'d';'o'] : bool`
(*f*) `('d','a','d') = ('d','o','d','o')` incorrecte
`d'une part : char * char * char`
`d'autre part : char * char * char * char`
(*g*) `let x="hello" in ['d';'a';x] < ['d';'o';x;'o']` incorrecte
`les listes doivent être homogènes et on mélange ici des char et des string`
(*h*) `let x="hello" in ('d','a',x) < ('d','o',x) : bool`
(*i*) `(=) 3 : int -> bool (application partielle de (=))`
(*j*) `(<) [3;5] : int list -> bool (application partielle de (<))`

Q3. (2 points)

(*a*) `fun x -> ([x],x) : 'a -> ('a list * 'a)`
(*b*) `function ([x]:_,"salut") -> x : ('a list list * string) -> 'a`

Remarque : le fait que le motif ne soit pas irréfutable n'est pas un problème pour le typage

---

Q4. (2 points)

(*a*) Le résultat du programme est `No`, puisque la liaison des noms est statique (`f` est liée à `eq` au moment de sa définition, pas au moment de l'appel).

(*b*) Oui, il y a une différence, cet autre programme répond `Yes`, il n'y a pas d'ambiguïté possible sur la valeur de `eq`.

(*c*) Oui, il y a une différence, cet autre programme répond `Yes`, puisque la référence sera lue (`!eq`) au moment où la fonction `f` sera appelée.

Q5. (2 points)

(*a*) `foo : ('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list`

(*b*) `foo` est exactement la fonction `List.map2`, qui prend une fonction `f` et deux listes `[x1;..;xn]` et `[y1;..;yn]` et construit une liste par application de `f` aux éléments des deux listes : `[(f x1 y1); ... ; (f xn yn)]`.

(*c*) `foo (+) [1;2;3] [4;5;6] = [5; 7; 9]`

Deuxième partie
Typage formel (4 points)

Q6 (4 points). Le *while* traditionnel est un effet de bord (`unit`) répété en boucle. On reste ou sort de la boucle selon une condition vrai ou fausse (`bool`). Le résultat est une composition séquentielle d'effets de bords, donc un (grand) effet de bord. Autrement dit :

(*a*)
$$\frac{\Gamma \vdash e_0 : \mathit{bool} \qquad \Gamma \vdash e_1 : \mathit{unit}}{\Gamma \vdash \texttt{while}\ e_0\ \texttt{do}\ e_1\ :\ \mathit{unit}}$$

(*b*)

```
let rec type_of_expr env = function
 ...
 | While (e0,e1) ->
    let t0 = type_of_expr env e0 in
     (match t0 with
      | TBool ->
          let t1 = type_of_expr env e1 in
          (match t1 with
            | TUnit -> TUnit
            | _ -> failwith ‘‘unit expression expected’’
            )
      | _ -> failwith ‘‘bool expression expected’’
      )
 ...
```

Troisième partie

Programmation (8 points)

Q7. (4 points)

```
module PArray_completion (K:PArray_kernel_signature) : PArray_signature =
 struct
  include K

  let map f a = K.create (K.length a) (fun i -> f i (K.get a i))
  let copy a  = map (fun i x -> x) a

  let set a xs =
     let b = copy a in
     let () = List.iter (fun (i,v) -> Array.set b i v) xs in
     b

  let rev a =
     let n = length a in
     map (fun i x -> get a (n-(i+1))) a

 end
```

Remarque (pour `set`) : si on ne connaît pas `List.iter`, on peut écrire une fonction qui fera le travail de `List.iter` par récursion (deux lignes de plus).

Q8. (4 points)

(*a*)

```
let rec map2 f l1 l2 =
  match (l1, l2) with
  | ([], []) -> []
  | (xl1, yl2) -> (f x y) : map2 f l1 l2
  | (_, _) -> raise (Invalid_argument "map2")
```

(*b*)

```
let rev_map2 f l1 l2 =
  let rec rmap2_f acc l1 l2 =
   match (l1, l2) with
   | ([], []) -> acc
   | (xl1, yl2) -> rmap2_f ((f x y) : acc) l1 l2
   | (_, _) -> raise (Invalid_argument "rev_map2")
  in rmap2_f [] l1 l2
```