Université Paris Diderot
M1 Informatique

Programmation Fonctionnelle Avancée
Année 2013-2014

TP n°2 bis

Usages avancés des types

Exercice 1 [Variants polymorphes et variance]
Soit le code suivant :

```
module M : sig
  type ('a, 'b) t
end = struct
  type ('a, 'b) t = 'a * 'b
end
```

1. Est-ce que l'expression suivante est bien typée ? Pourquoi ?

   ```
   let f x = (x : ([ `A ], [ `B ]) M.t :> ([ `A | `C ], [ `B | `D ]) M.t)
   ```

2. Sans utiliser des types avancés, comment peut-on faire pour que cela type ?
3. Modifier le code afin de ne rien dévoiler de l'implémentation et que l'expression de la question 1 soit bien typée.
4. Corriger la signature du code suivant pour qu'il compile :

   ```
   module M : sig
     type (+'a, +'b) t
   end = struct
     type ('a, 'b) t = 'a -> 'b
   end
   ```

Exercice 2 [Types fantômes]
On veut écrire un module connection qui permet de se connecter en lecture seule ou lecture/écriture. Pour cela on va utiliser les types fantômes et les variants polymorphes.

Complétez le code suivant là où il ya des ... :

```
module Connection : sig
  ...
  val connect_readonly : ...
  val connect : ...
  val status : ...
  val destroy : ...
end = struct
  type 'a t = int
  let count = ref 0
  let connect_readonly () = incr count; !count
  let connect () = incr count; !count
  let status c = c
  let destroy c = ()
end
```

Pour tester votre code vous pourrez essayer :

```
open Connection
open Printf

let () =
  let conn = connect_readonly () in
    printf "status = %d\n" (status conn);
    (*destroy conn; (* error *) *)
    let conn = connect () in
      printf "status = %d\n" (status conn); destroy conn
```

sachant que si vous décommentez `destroy conn` vous *devez* avoir une erreur.

Exercice 3 [GADT]
On veut écrire une structure de données qui soit une liste pouvant contenir des entiers et des flottants avec une unique fonction `get` permettant de récupérer le premier entier (ou flottant).

1. Ecrire en utilisant un GADT le type kind dont on se servira pour indiquer à la fonction `get` le type que l'on veut récupérer.
2. Ecrire le type somme value permettant de contenir des entiers et des flottants
3. Ecrire le type list en utilisant un GADT.
4. Ecrire la fonction `get`
5. Ecrire une fonction `print` qui affiche le contenu d'une liste.
6. Testez vos fonctions avec les expressions suivantes :

   ```
   let empty = Nil
   let l1 = Cons ((Int 1), empty)
   let l2 = Cons ((Float 2.), l1)
   let () = print l2
   let i = get l2 Int_kind
   let f = get l2 Float_kind;;
   ```