Feuille de TD n°8

Exercice 1 – Signature par défaut d'un module

La signature par défaut d'un module, est la signature dans laquelle tous les éléments définis dans ce module sont visibles.

Quelle est la signature par défaut des modules suivants :

1. ```
   module Choice =
     struct
       type choice = | Yes | No

       let of_bool =
         function
         | true -> Yes
         | false -> No

       let to_bool =
         function
         | Yes -> true
         | No -> false

       let to_string =
         function
         | Yes -> "yes"
         | No -> "no"

       let select =
         fun choice f1 f2 value ->
         match choice with
         | Yes -> f1 value
         | No -> f2 value
     end
   ```

2. ```
   module Color =
     struct
       type color = | Red | Blue | Green

       let to_int =
         function
         | Red -> 0
         | Blue -> 1
         | Green -> 2

       exception Not_a_color_index

       let of_int =
         function
   ```

```
        | 0 -> Red
        | 1 -> Blue
        | 2 -> Green
        | _ -> raise Not_a_color_index

    end
```

Exercice 2 – Modules et signatures

On considère la signature et les modules suivants :

```
module type S =
  sig
    type t
    val create: unit -> t
    val next: t -> t
    val compare: t -> t -> int
  end

module A =
  struct
    type t = | One | Two
    let create = fun () -> One
    let next = function
      | One -> Two
      | Two -> One
    let compare =
      fun number1 number2 ->
      match (number1, number2) with
      | (One, One) -> 0
      | (One, Two) -> 1
      | (Two, One) -> -1
      | (Two, Two) -> 1
  end

module B =
  struct
    type t = bool
    let create : 'a -> bool =
      fun x -> true
    let succ = fun b -> not b
    let compare =
      fun b1 b2 ->
      if b1=b2 then 0
      else if b1 < b2 then 1
      else -1
  end

module C =
  struct
    type t = int
```

```
  let create = fun x -> 0
  let next = (+) 1
  let compare = (<)
end
```

1. Est-ce que le module `A` implante la signature `S` ? Pourquoi ?
2. Est-ce que le module `B` implante la signature `S` ? Pourquoi ?
3. Est-ce que le module `C` implante la signature `S` ? Pourquoi ?

Exercice 3 – Implantations de modules

Écrire des modules qui implantent de trois façons différentes – avec trois définitions du type `t` différentes – la signature suivante :

```
module type PAIR =
  sig
    type ('a, 'b) pair
    val create: 'a -> 'b -> ('a, 'b) pair
    val fst: ('a, 'b) pair -> 'a
    val snd: ('a, 'b) pair -> 'b
  end
```

Toutes ces implantations doivent vérifier les propriétés suivantes :

— pour toute valeurs `a` et `b`, `fst(create a b) = a`,

— pour toute valeurs `a` et `b`, `snd(create a b) = b`.

Question bonus. Essayez d'écrire un module qui implante la signature mais ne respecte pas les propriétés.

Exercice 4 – Implantations de modules

Soit la signature `PERSONNE` :

```
module type PERSONNE =
  sig
    (** Le type t est un type pour représenter une personne :
        son nom, son prénom, et son âge (en années). *)
    type t
    (** (creer nom prenom age) créer une personne dont
        le nom est nom, le prénom prenom, et l'âge age.
        L'âge doit être positif, sinon une exception
        Invalid_argument "creer: age invalide" est levée. *)
    val creer: string -> string -> int -> t
    (** (nom p) retourne le nom de la personne p. *)
    val nom: t -> string
    (** (prenom p) retourne le prénom de la personne p. *)
    val prenom: t -> string
    (** (age p) retourne l'âge de la personne p. *)
    val age: t -> int
    (** (meme_identite p1 p2) renvoie true si les deux
        personnes p1 et p2 ont le même nom et prénom,
```

```
    faux sinon. *)
  val meme_identite: t -> t -> bool
  (** (plus_jeune p1 p2) retourne true si la personne p1
      est strictement moins âgée que la personne p2. *)
  val plus_jeune: t -> t -> bool
end
```

1. En utilisant un type enregistrement pour définir le type `t`, écrire un module `Personne_E` qui implante la signature `PERSONNE`.
2. En utilisant le type `list` pour définir le type `t`, écrire un module `Personne_L` qui implante la signature `PERSONNE`.

Exercice 5 – Bonus – Extension de module

On souhaite ajouter à notre représentation des personnes, l'adresse de la personne (sous forme d'une chaîne de caractères), mais l'on souhaite pouvoir étendre n'importe quelle implantation de la signature `PERSONNE` de cette façon.

Le module étendu devra contenir les mêmes fonctions que données par la signature `PERSONNE` sauf que le type de `creer` sera désormais `string->string->int->string->t` et qu'une fonction `adresse: t -> string` sera fournie.

Pour ceci, écrire un foncteur prenant en argument un module de signature `PERSONNE` et renvoyant le module décrit au paragraphe précédent.

Définir des modules étendus à partir des modules `Personne_E` et `Personne_L`.