

# TD - Programmation Fonctionnelle 2

Louis Thevenet

## 1. TD5

### 1.1. Exercice 1

```
1 module type FL2C = sig
2   type zero
3   type _ succ
4   type 'a fichier
5
6   val open_ : string -> zero fichier
7   val read : 'n fichier -> char * 'n succ fichier
8   val close : zero succ succ fichier -> unit
9 end
```

```
1 module type FLPair = sig
2   type even
3   type odd
4   type fichier
5
6   val open_ : string -> (even, odd) fichier
7   val read : ('a*'b) fichier -> char * ('b*'a) succ fichier
8   val close : (even*odd) fichier -> unit
9 end
```

### 1.2. Exercice 2

```
1 type 'a perfect_tree = Empty | Node of 'a * ('a * 'a) perfect_tree
2
3 let rec split : 'a. ('a * 'a) perfect_tree -> 'a perfect_tree * 'a perfect_tree
4   =
5   fun tree ->
6     match tree with
7     | Empty -> (Empty, Empty)
8     | Node ((l1, l2), subtree) ->
9       let t1, t2 = split subtree in
10      (Node (l1, t1), Node (l2, t2))
```

## 2. TD6

$\text{fold\_right} : (\alpha \rightarrow \beta \rightarrow \beta) \rightarrow \alpha \text{ list} \rightarrow \beta \rightarrow \beta \equiv ((\alpha \times \beta) \rightarrow \beta) \rightarrow (\text{unit} \rightarrow \beta) \rightarrow \alpha \text{ list} \rightarrow \beta$   
 $\equiv ((\alpha \times \beta) \text{ option} \rightarrow \beta) \rightarrow \alpha \text{ list} \rightarrow \beta$

$$\text{unfold} : \left( \underbrace{\beta}_{\text{type g n rateur}} \rightarrow \left( \alpha \times \underbrace{\beta}_{\text{pour la prochaine g n ration}} \right) \text{ option} \right) \rightarrow (\beta \rightarrow \alpha \rightarrow \alpha \text{ flux})$$

```
1 module type Iter =
2   sig
3     type 'a t
4     val vide : 'a t
5     val cons : 'a -> 'a t -> 'a t
6     val uncons : 'a t -> ('a * 'a t) option
7     val apply : ('a -> 'b) t -> ('a t -> 'b t)
8     val unfold : ('b -> ('a * 'b) option) -> ('b -> 'a t)
9     val filter : ('a -> bool) -> 'a t -> 'a t
10    val append : 'a t -> 'a t -> 'a t
11  end
```

```
1 let flux_nul = Flux.unfold (fun () -> Some(0, ())) ()
2 (* le flux qui contient tous les entiers relatifs pairs, par ordre croissant en
3    valeur absolue *)
3 let flux_pair = Iter.unfold (fun i -> Some(2*i, if i <= 0 then 1-i else -i))
```

### 2.1. Exercice 1

```
1 let constant e = Iter.unfold (fun () -> Some(e, ())) ()
2 let map f fl = Flux.(apply (constant f) fl)
3 let map2 f fl fl' = Flux.(apply (map f fl) fl')
```

## 3. TD7