corrige

October 9, 2024

1 Correction 09/10/2024

1.1 Sujet 1

1.1.1 Exercice 2

[1]:
```
type 'a arbre_binaire = N of 'a * 'a arbre_binaire * 'a arbre_binaire | F
```

[1]:
```
type 'a arbre_binaire = N of 'a * 'a arbre_binaire * 'a arbre_binaire | F
```

[2]:
```
let rec est_abr arbre = match arbre with
|F -> true
|N(x,g,d) -> match g,d with
    |F,F -> true
    |N(xg,gg,dg), F -> xg<x && est_abr g
    |F, N(xd,gd,dd) -> xd>x && est_abr d
    |N(xg,gg,dg), N(xd,gd,dd) -> xg<x && xd>x && est_abr g && est_abr d
```

[2]:
```
val est_abr : 'a arbre_binaire -> bool = <fun>
```

[3]:
```
let rec est_dans_arbre x arbre = match arbre with
|F -> false
|N(y,g,d) -> x=y || est_dans_arbre x g || est_dans_arbre x d
```

[3]:
```
val est_dans_arbre : 'a -> 'a arbre_binaire -> bool = <fun>
```

[4]:
```
let rec est_dans_abr x arbre = match arbre with
|F -> false
|N(y,g,d) -> if x=y then true else if x<y then est_dans_arbre x g else␣
 ↪est_dans_arbre x d
```

[4]:
```
val est_dans_abr : 'a -> 'a arbre_binaire -> bool = <fun>
```

1.2 Sujet 2

[1]:
```
type intervalle = int * int

type tree =
| F of intervalle
| N of intervalle * tree * tree

let intervalle t = match t with
|F((x,y)) -> x,y
|N((x,y),_,_) -> x,y
```

[1]:
```
type intervalle = int * int
```

[1]:
```
type tree = F of intervalle | N of intervalle * tree * tree
```

[1]:
```
val intervalle : tree -> int * int = <fun>
```

[3]:
```
(* Pas très efficace mais fonctionne*)
let rec disponible (a,b) t = match t with
|F((x,y)) -> a>y || b<x
|N((x,y),g,d) -> if a>y || b<x then true else (* en dehors de l'intervalle *)
                 disponible (a,b) g && disponible (a,b) d
```

[3]:
```
val disponible : int * int -> tree -> bool = <fun>
```

[4]:
```
let fusionne a b = match a,b with
|None,None -> None
|None,Some(t) | Some(t), None -> Some(t)
|Some(t), Some(u) ->
    let (c1,c2) = intervalle t in
    let (c3,c4) = intervalle u in
    if c2<c3 then Some(N((c1,c4), t, u)) else Some(N((c3,c2), u, t))
```

[4]:
```
val fusionne : tree option -> tree option -> tree option = <fun>
```

[5]:
```
let rec scinde (a,b) t = match t with
|F((x,y)) |N((x,y),_,_) when a>y -> Some(t), None, None
|F((x,y)) |N((x,y),_,_) when b<x -> None, None, Some(t)
|F((x,y)) |N((x,y),_,_) when x>=a && y<=b ->  None, Some(t), None
|F((x,y)) -> if x<b then None, Some(F((x,b))), Some(F((b+1,y)))
             else Some(F((x,a-1))), Some(F((a,y))), None
|N((x,y),g,d) -> let sa,sb,sc = scinde (a,b) g in
```

```
                let sx,sy,sz = scinde (a,b) d in
                (fusionne sa sx), (fusionne sb sy), (fusionne sc sz)
```

[5]:
```
val scinde : int * int -> tree -> tree option * tree option * tree option =
  <fun>
```

[6]:
```
let rec inserer (a,b) t = match t with
|F((x,y)) -> if x<b then N((a,y),F((a,b)),t) else N((x,b),t, F((a,b)))
|N((x,y),_,_) when a>y ->
|N((x,y),_,_) when b<x ->
```

[6]:
```
val inserer : 'a * 'b -> 'c -> 'c = <fun>
```

1.3 Sujet 3

1.3.1 Exercice 2 : Arbre presque complet

[7]:
```
type tree = N of int* tree * tree | V;;
```

[7]:
```
type tree = N of int * tree * tree | V
```

[8]:
```
let rec generer n = match n with
| 1 -> N(1,V,V)
| n -> N(n,(generer (n-1)),V);;


generer 4;;
```

[8]:
```
val generer : int -> tree = <fun>
```

[8]:
```
- : tree = N (4, N (3, N (2, N (1, V, V), V), V), V)
```

[9]:
```
let log2 x = (log x)/.(log 2.);;

let pow2 x = int_of_float (2.**(float_of_int x));;

let rec generer2 n a b =
  if n=1 then N(a,V,V) else
  if n=2 then N(a,N(b,V,V),V) else
  let l = log2 (float_of_int (n+1)) in
  if Float.is_integer l then ( (*cas simple -> arbre complet*)
    let half = (n-1)/2 in
    N(a, (generer2 half (a+1) (a+half)), (generer2 half (a+half+1) b))
```

```
  )
  else ( (*Presque complet, on isole le nombre de noeud de l'arbre complet le␣
 ↪plus proche *)
    let pel = int_of_float l in
    let ncomplet = (pow2 pel)-1 in (* nombre de noeud d'un arbre complet *)
    let reste = n - ncomplet in
    let half = (ncomplet-1)/2 in
    N(a, (generer2 (half+reste) (a+1) (a+half+reste)), (generer2 half␣
 ↪(a+half+reste+1) b))
  )
;;

generer2 8 1 8;;
```

```
[9]: val log2 : float -> float = <fun>
```

```
[9]: val pow2 : int -> int = <fun>
```

```
[9]: val generer2 : int -> int -> int -> tree = <fun>
```

```
[9]: - : tree =
     N (1, N (2, N (3, N (4, V, V), V), N (5, V, V)),
      N (6, N (7, V, V), N (8, V, V)))
```