Examen du 11 Janvier 2023

Durée 1h30

Téléphones, calculettes et ordinateurs interdits. Tous les documents sont autorisés. Dans chaque question, vous pouvez utiliser les fonctions demandées dans les questions et les exercices précédents même si vous ne les avez pas définies. Vous pouvez utiliser les fonctions prédéfinies de la librairie standard (par exemple `List.mem`*), sauf spécifié autrement dans la question. Le barème est donné sur 28 : la note finale sur 20 sera* $\frac{n \times 20}{28}$ *ou n est le total de points obtenus. Inscrire votre numéro d'anonymat sur votre copie.*

Rappel. La représentation binaire *inversée* d'un entier naturel n est la séquence de bits $b_0 b_1 \cdots b_N$ telle que $n = \sum_{i=0}^{N} b_i 2^i$ (b_N est le bit de poids fort et b_0 est le bit de poids faible).

Notez que $b_0 =$ `n mod 2`, et que $(b_1 \cdots b_N)$ représente l'entier `n / 2`.

Exemple : 110101 est la représentation binaire inversée de l'entier 43 puisque :

$$43 = (1 \times 2^0) + (1 \times 2^1) + (0 \times 2^2) + (1 \times 2^3) + (0 \times 2^4) + (1 \times 2^5) = 1 + 2 + 0 + 8 + 0 + 32$$

Exercice 1 (2+2+3=7 points, *Représentation binaire inversée d'un entier*)**.**

On encode une représentation binaire inversée d'un entier par une liste de booléens, où `true` correspond au bit 1 et `false` au bit 0. En représentation binaire inversée, les bits de poids faibles sont en début de liste (c'est-à-dire à gauche).

Exemples :

— 2, en binaire inversé 01, correspond à `[false; true]`

— 43, en binaire inversé 110101, correspond à `[true; true; false; true; false; true]`

On remarquera que si l est une liste de booléens représentant un entier i, alors la liste `l @ [false]` obtenue en ajoutant `false` à la fin de la liste l représente le même entier i : en effet, ajouter un 0 en position de poids fort ne change pas l'entier représenté (car cela correspond à l'ajout d'un 0 non significatif). Il existe donc plusieurs représentations binaires inversées d'un même entier.

Exemples : les listes suivantes représentent toutes l'entier 6

```
[false; true; true]
[false; true; true; false]
[false; true; true; false; false]
```

On note `bin` le type OCaml des listes de booléens :

```
type bin = bool list
```

1. (2 points) Définir une fonction de signature `bin_to_int (i : bin) : int` calculant un entier à partir de sa représentation binaire inversée.

   Exemples :

   ```
   bin_to_int [] = 0
   bin_to_int [false; true; true] = 6
   bin_to_int [true; true; false; true; false; true] = 43
   ```

   Une solution

   ```
   let rec bin_to_int (b : bin) : int =
     match b with
     | [] -> 0
     | a :: b -> (if a then 1 else 0) + 2 * bin_to_int b
   ```

2. (2 points) Définir une fonction de signature `int_to_bin (i : int) : bin` calculant la représentation binaire inversée d'un entier.

   Exemples :
   ```
   int_to_bin 6 = [false; true; true]
   int_to_bin 43 = [true; true; false; true; false; true]
   ```

   Une solution
   ```
   let rec int_to_bin (i : int) : bin =
     if i = 0 then []
     else if i mod 2 = 0 then false :: int_to_bin (i / 2)
                         else true  :: int_to_bin (i / 2)
   ```

3. (3 points) Définir une fonction de signature `comp_bin (l : bin) (n : int) : bin` qui construit la liste de booléens obtenue en ajoutant à la fin de la liste `l` les booléens `false` pour obtenir une liste de longueur `n`. Si `n` est strictement inférieur à la longueur de `l`, alors la fonction `comp_bin` lève l'exception `Invalid_argument` ; si `n` est égal à la longueur de `l`, alors la fonction `comp_bin` retourne la liste `l`.

   Exemples :
   `comp_bin [false; true; true] 2` lève l'exception `Invalid_argument`
   ```
   comp_bin [true; false] 2 = [true; false]
   comp_bin [true; false] 4 = [true; false; false; false]
   ```

   Une solution
   ```
   let rec comp_bin (l : bin) (n : int) : bin =
     match l,n with
     | [],0 -> l
     | [],n -> false :: (comp_bin [] (n-1))
     | h::t,n -> if n=0 then raise (Invalid_argument "comp_bin")
                        else h::(comp_bin t (n-1))
   ```

Exercice 2 (3+3+3+2+2+2+3+3=21 points, *Arbres équilibrés et ensemble de nombres*)**.**

Un arbre binaire est dit *équilibré* si tous les chemin (à partir de la racine jusqu'à une feuille) sont de même longueur.

Exemple : l'arbre ci-dessous à gauche est équilibré, mais l'arbre ci-dessous à droite ne l'est pas (on représente les nœuds internes par des ○, et les feuilles par des □).

Notez qu'un arbre équilibré de hauteur k a exactement 2^k feuilles.

Dans cet exercice, la hauteur $h(t)$ d'un arbre binaire t est 0 si t est une feuille, et $1+\max(h(g),h(d))$ si t est un arbre avec un fils gauche g et un fils droit d. Si t est un arbre binaire équilibré alors $h(g)=h(d)$, et dans ce cas $h(t)=1+h(g)=1+h(d)$.

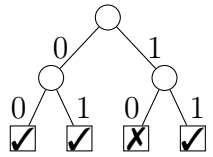
On souhaite représenter un ensemble d'entiers par un arbre binaire équilibré dont les nœuds internes ne sont pas étiquetés et les feuilles sont étiquetées par des booléens.

Un arbre binaire équilibré de hauteur k représente le sous-ensemble des entiers de $\{0, \ldots, 2^k - 1\}$ dont une représentation binaire inversée représente un chemin de la racine à une feuille marquée `true` tel qu'à chaque étape du cheminement, le bit 0 indique de descendre sur le sous-arbre gauche tandis que le bit 1 indique de descendre sur le sous-arbre droit.

Exemple : Dans l'arbre ci-contre ✓ représente le booléen `true`, ✗ représente le booléen `false`. Cet arbre représente l'ensemble $\{0, 2, 3\}$ puisque 00, 01, et 11 sont les représentations binaires inversées des entiers 0, 2 et 3. Il s'agit bien d'un sous ensemble de $\{0, 1, 2, 3\}$.

Afin de représenter de tels arbres en OCaml on définit le type `int_set` ci-contre. L'arbre de l'exemple ci-dessus est représenté par la valeur OCaml `ex` définie ci-contre. Les bits 0 et 1 qui figurent sur les arêtes correspondent aux indications de chemin (0 pour l'arête gauche et 1 pour l'arête droite) mais ne font pas partie de la représentation des arbres.
On suppose définies les exceptions `Not_perfect` et `No_fit`.

```
(* Un arbre représentant
   l'ensemble {0;2;3} *)
```




```
type int_set =
  | N of int_set * int_set
  | E of bool

let ex =
    N(N(E true,
        E true),
      N(E false,
        E true))

exception Not_perfect
exception No_fit
```

1. (3 points) Définir une fonction de signature `hauteur_equilibre (t: int_set): int` qui calcule la hauteur de l'arbre `t` si `t` est un arbre équilibré et lève l'exception `Not_perfect` sinon. Attention l'arbre `E(...)` est de hauteur 0.

   Une solution

   ```
   let rec hauteur_equilibre (t: int_set): int =
     match t with
     | E _ -> 0
     | N(g, d) ->
       let x = hauteur_equilibre g in
       let y = hauteur_equilibre d in
       if x <> y then raise Not_perfect
       else x+1
   ```

   Exemples :
   `hauteur_equilibre ex = 2`
   `hauteur_equilibre (N(N(E true,E true),E false))` lève l'exception `Not_perfect`

2. (3 points) Définir une fonction de signature `mem (x: int) (t: int_set): bool` permettant de tester si l'entier `x` appartient à l'ensemble représenté par l'arbre `t`.

   Indication : l'entier `x` appartient à l'ensemble représenté par l'arbre `t` si la liste de booléens correspondant à une représentation binaire inversée de `x` est un chemin de la racine de `t`

   — à une feuille étiquetée par le booléen `true`

   — ou bien à un nœud interne n à partir duquel on accède à une feuille étiquetée par le booléen `true` en se déplaçant uniquement vers la gauche depuis le nœud n (puisque si l est une liste de booléens représentant un entier, alors la liste obtenue en ajoutant un nombre quelconque de booléens `false` à la fin de la liste l représente le même entier)

   La fonction `mem` pourra donc utiliser une fonction auxiliaire dont le paramètre est la représentation binaire inversée du paramètre `n` de `mem`.

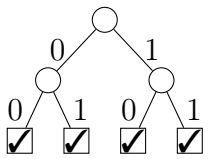
Exemples :
`mem 2 ex` = `true`
`mem 1 ex` = `false`

Une solution

```
let mem (x: int) (t: int_set): bool =
  let rec doit (b : bin) (t : int_set) : bool =
    match b, t with
    | [], E b -> b
    | [], N (g,d) -> doit [] g
    | a :: b, N (g,d) -> if not a then doit b g else doit b d
    | _ -> false
  in
  doit (int_to_bin x) t
```

3. (3 points) Définir une fonction de signature `insert_exn (t: int_set) (x: int):int_set` qui construit l'arbre représentant l'ensemble des entiers $F_t \cup \{x\}$ où F_t est l'ensemble des entiers représenté par `t` ; l'arbre construit doit avoir la même hauteur que l'arbre `t`. Si `x` ne peut être inséré dans `t` sans changer la hauteur de `t`, alors la fonction `insert_exn` lèvera l'exception `No_fit`. L'insertion n'est donc possible que si elle peut se faire uniquement en changeant l'étiquette d'une feuille.

Exemples :
`insert_exn ex 1` =



`(insert_exn ex 25)` lève l'exception `No_fit`

Indication : on pourra calculer la liste de booléens correspondant à une représentation binaire inversée de `x`, et suivre le chemin de la racine de `t` jusqu'à une feuille comme dans la question précédente (c'est-à-dire en se déplaçant à gauche dans l'arbre lorsque la liste de booléens est vide).

Une solution

```
(* dans le style de mem *)
let insert_exn (t: int_set) (x: int) : int_set =
  let rec doit (t : int_set) (b : bin) : int_set =
    match b, t with
    | [], E b -> E true
    | [], N (g,d) -> N (doit g [], d)
    | a :: b, N (g,d) -> if not a then N (doit g b,d) else N(g,doit d b)
    | _ -> raise No_fit
  in
  doit t (int_to_bin x)
```

4. (2 points) Définir une fonction de signature `increase_height (t: int_set) : int_set` qui construit un arbre de hauteur $h(\texttt{t})+1$ et qui représente exactement le même ensemble d'entiers que l'arbre `t`.

Indication : il s'agit d'ajouter un fils gauche et un fils droit à chaque feuille de l'arbre `t`

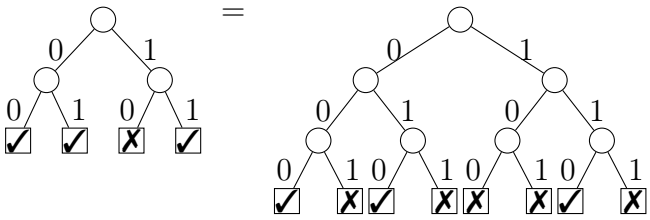
— si cette feuille est étiquetée par le booléen `true` alors l'entier correspondant au chemin menant à

cette feuille appartient à l'ensemble représenté par `t` et il suffit alors que le fils gauche ajouté soit étiqueté par le booléen `true` (puisqu'un déplacement vers la gauche correspond au booléen `false` sur le chemin et qu'ajouter le booléen `false` à la fin d'un chemin ne change pas la valeur de l'entier représenté par le chemin) et que le fils droit ajouté soit étiqueté par le booléen `false` (pour ne pas ajouter d'entier à l'ensemble représenté par `t`)

— si cette feuille est étiquetée par le booléen `false` alors l'entier correspondant au chemin menant à cette feuille n'appartient pas à l'ensemble représenté par `t` et il suffit alors que les fils gauche et droit ajoutés soient étiquetés par le booléen `false` (pour ne pas ajouter d'entier à l'ensemble représenté par `t`)

Exemple :

`increase_height` =



Une solution

```
let rec increase_height (t: int_set) : int_set =
  match t with
  | E b -> N(E b, E false)
  | N(g, d) -> N(increase_height g, increase_height d)
```

5. (2 points) Déduire des deux questions précédentes une fonction de signature :

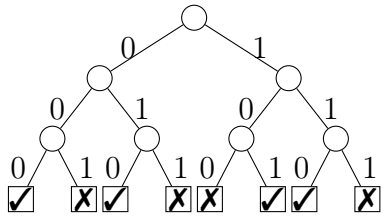
   `insert (x: int) (t: int_set) : int_set`

qui construit l'arbre représentant l'ensemble des entiers $F_t \cup \{\texttt{x}\}$ où F_t est l'ensemble des entiers représenté par `t`. La taille de l'arbre sera augmentée au besoin, autant de fois que nécessaire.

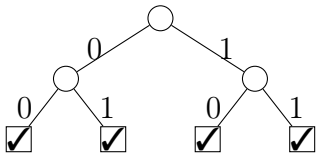
Indication : il suffit d'essayer d'ajouter l'entier `x` dans l'ensemble représenté par `t`, et si cette tentative échoue, d'essayer à nouveau sur l'arbre `t` dont la hauteur a été augmentée de 1, et ainsi de suite.

Exemples :

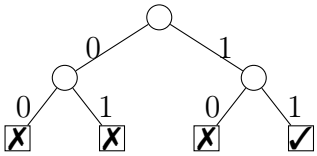
`insert 5 ex` =



`insert 1 ex` =



`insert 3 (E false)` =



Une solution

```
let rec insert (x: int) (t: int_set) : int_set =
  try insert_exn t x
  with | No_fit -> insert x (increase_height t)
```

6. (2 points) *En déduire* une fonction de signature `list_to_tree (l: int list) : int_set` qui construit l'arbre représentant l'ensemble des entiers contenus dans la liste `l`.

   Exemples :
   `list_to_tree [] = E false`
   `list_to_tree [0;2;3] = ex`

Une solution

```
let list_to_tree (l: int list) : int_set =
  List.fold_left (fun t x -> insert x t) (E false) l
```

7. (3 points) Définir une fonction de signature :

   `path_list (t : int_set) : (bin * bool) list`

   permettant de construire la liste de toutes les paires `(c,b)` telles que `c` est une liste de booléens correspondant à un chemin de la racine à une feuille de l'arbre `t` et `b` est le booléen qui étiquette cette feuille.

   Exemple :

```
path_list ex = [([false; false], true); ([false; true], true);
                ([true; false], false); ([true; true], true)]
```

Une solution

```
let rec path_list (t:int_set) : (bin * bool) list =
  match t with
  | E b -> [([],b)]
  | N(g,d) -> (List.map (fun (l,b) -> (false :: l, b)) (path_list g)) @
              (List.map (fun (l,b) -> ( true :: l, b)) (path_list d))
```

8. (3 points) En déduire une fonction de signature :

   `tree_to_list (t : int_set) : int list`

   permettant de construire la liste des entiers appartenant à l'ensemble représenté par l'arbre `t`.

   Exemple :

```
tree_to_list ex = [0; 2; 3]
```

Une solution

```
let tree_to_list (t : int_set) : int list =
  List.map
    (fun (l,_) -> bin_to_int l)
    (List.filter (fun x -> (snd x)) (path_list t))
```