Méthodes formelles de vérification (MF)
TD n° 3 : Preuve de programmes fonctionnels

Exercice 1 :

Lesquelles des relations suivantes sont bien fondées ?

1. ($\mathbb{R}$, $<$)

2. ($(1/2, 1]$, $<$)

3. ($\mathbb{N}$, $\leqslant$)

4. ($\mathbb{N}$, $<$)

5. ($\mathbb{Z}$, $>$)

6. (`Nat`, `Lt`)

où
```
Lt(0   , S m)  = true
Lt(n   , 0)    = false
Lt(S n, S m)   = Lt(n,m)
```

7. (`List[Nat]`, `Smaller`)

où
```
Smaller(l::ls ,[])   = false
Smaller([]    ,_)    = true
Smaller(l::ls ,m::ms) = Smaller(ls,ms)
```

8. Donner une définition d'ordre bien fondé pour les naturels binaires définis au TD1 (`Bin`).

Exercice 2 :

On considère le type des arbres binaires de recherche (ABR) qui sont des arbres binaires (`BinTree'`) respectant la propriété inductive suivante :

```
Inductive BinTree :=
| Leaf : Int -> BinTree
| Node : Int × BinTree × BinTree -> BinTree

ABR(Leaf(n))              = true
ABR(Node(x,left,right)) = (∀ y Find(left,y) ⇒ y < x) ∧
                          (∀ z Find(right,z) ⇒ x < z) ∧
                          ABR(left) ∧ ABR(right)
```

1. Spécifier la fonction `Flatten` (en utilisant `Mem` et `Find`) pour les ABR.

2. Prouver que l'implémentation de `flatten` donnée au TD1 satisfait la spécification (sinon corrigez-la et prouvez-la).

Exercice 3 :

Prouver la correction des fonctions `Partition` et `FusionSort` du TD2.

Exercice 4 :

Compléter les preuves laisses comme exercice au cours.

1. Etant donnée `Append : List[*] -> List[*] -> List[*]`, où

```
Spec_Append(l1,l2,l) =
     Append(l1,l2) = l ⇒
        |l|=|l1|+|l2| ∧
        (∀ i ∈ ℕ, (0 ⩽ i ⩽ |l1|) ⇒ l[i] = l1[i]) ∧
                (∀ i ∈ ℕ, (0 ⩽ i ⩽ |l2|) ⇒ l[|l1|+i] = l2[i])
```

et

```
Append([],l)     = l
Append(a::l1,l2) = a::(Append(l1,l2))
```

prouver que :

```
(∀ l1, l2, l, Append(l1,l2) = l ⇒ Spec_Append(l1,l2,l))
```

2. Etant donnée `Insert : * -> List[*] -> List[*]`, où

```
Spec_Insert(a,l,l') = Insert(a,l) = l' ⇒
                        Ordered(l') ∧ (Ms(l') = Sg(a) ∪ Ms(l))
```

et

```
Insert(a,[])   = a::[]
Insert(a,b::l) = if a ⩽ b then a::(b::l) else b::(Insert(a,l))
```

prouver que :

```
(∀ a, l, l', Insert(a,l) = l' ⇒ Spec_Insert(a,l,l')
```

3. Etant donnée `qsort : List[*] -> List[*]`, où

```
Spec_qsort(l,l') = qsort(l) = l' ⇒
                     Ordered(l') ∧ (Ms(l') = Sg(a) ∪ Ms(l))
```

et

```
qsort([])   = []
qsort(a::l) = let (l1, l2) = split (a, l) in
                 Append(qsort(l1),(a::qsort(l2))
```

avec `split` étant la fonction qui divise la liste donnée en deux, où la première contient les éléments plus petits que `a`, et la deuxième les autres.

a. Donner une spécification et une implémentation pour `split`.

b. En supposant la correction de `split` prouver que pour tout `l` et `l'`, `Spec_qsort(l,l')`