Méthodes formelles de vérification (MF)
TD n° 2 : Spécifications en logique du premier ordre

Exercice 1 :

Étant donnée la signature (`Nat`, `Bin`, {`binToNat`}, $\{=, \neq\}$) où `Nat`, `Bin` et `binToNat` sont comme vu lors du TD1. Donner la spécification de la fonction `natToBin` qui est l'inverse de la fonction `binToNat`.

Exercice 2 :

On considère le type des séquences d'entiers $Seq[\texttt{Int}]$ dont les constructeurs sont

$\epsilon : Seq[\texttt{Int}]$, la séquence vide,

$\bullet$: $\texttt{Int} \times Seq[\texttt{Int}] \rightarrow Seq[\texttt{Int}]$, l'ajout en tête (à gauche) d'un élément à une séquence.

Etant donnée une séquence s, on désigne par $|s|$ la longueur de la séquence s. On considère que $|\epsilon| = 0$. Pour tout entier naturel $i \leqslant |s|$, on désigne par $s[i]$ l'élément se trouvant à la position i dans s.
Soit la fonction

$$\texttt{Fusion} : Seq[\texttt{Int}] \times Seq[\texttt{Int}] \rightarrow Seq[\texttt{Int}]$$

qui étant données deux séquences triées s et s', fusionne ces séquences. (On suppose qu'il est possible d'avoir plusieurs occurrences d'un même entier dans une séquence.)

1. Ecrire une spécification formelle de la fonction `Fusion` en logique de premier ordre reliant ses données et son résultat.
2. Donner une implémentation récursive de cette spécification.

On considère la fonction

$$\texttt{Partition} : Seq[\texttt{Int}] \rightarrow Seq[\texttt{Int}] \times Seq[\texttt{Int}]$$

dont la spécification est la suivante :

$$\texttt{Partition}(s) = (s_1, s_2) \iff$$
$$(|s| = |s_1| + |s_2|) \wedge (\forall i, 2i \leqslant |s| \Rightarrow s[2i] = s_1[i]) \wedge (\forall i, 2i < |s| \Rightarrow s[2i+1] = s_2[i])$$

3. Définir une fonction récursive de tri `FusionSort` basée sur les fonctions `Partition` et `Fusion`.
4. Donner une implémentation récursive de la fonction `Partition`.

Exercice 3 :

On considère le type des arbres binaires des entiers :

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

On considère la fonction `Flatten : BinTree -> List[Int]` qui prend un arbre en argument, et renvoie la liste des éléments dans les feuilles de l'arbre.

1. On peut donner la spécification de la fonction `Flatten` en supposant une signature avec les fonctions suivantes ? :
   - `Find : BinTree -> Int -> Bool`
   - `Mem : List[*] -> * -> Bool`

2. Donner la spécification complète de `Flatten`. (NB : On peut utiliser les `Multiset`).