

## Méthodes formelles de vérification (MF)

### TD n° 2 : Spécifications en logique du premier ordre

#### Exercice 1 :

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

#### Exercice 2 :

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

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

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

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

Soit la fonction

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

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

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

dont la spécification est la suivante :

$$\text{Partition}(s) = (s_1, s_2) \iff$$

$$(|s| = |s_1| + |s_2|) \wedge (\forall i, 2i \leq |s| \Rightarrow s[2i] = s_1[i]) \wedge (\forall i, 2i < |s| \Rightarrow s[2i + 1] = s_2[i])$$

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`).

**Exercice 4 :**

On utilise la définition d'arbre binaire de l'Exercice 3 pour définir les Arbres Binaires de Recherche (ABR).

1. Donnez l'implémentation d'une fonction pour vérifier si un `BinTree` est un ABR.
2. Donnez la propriété qu'indique qu'un arbre binaire est bien un ABR.<sup>1</sup>
3. Donnez l'implémentation des fonctions `Find` et `Insert` pour les ABR.
4. Prouvez la correction de la fonction `Insert`.
5. Donnez l'implémentation de la traversée de l'arbre dans l'ordre avec le type

`InOrder` : `BinTree -> List[Int]`

6. Prouvez que si l'arbre d'entrée est un ABR la liste produite par `InOrder` est ordonnée.

---

1. La spécification même pourrait être récursive.