# TD 1 – Structures de données inductives

**Exercice 1** – *Entiers positifs et négatifs*

Les définitions inductives suivantes modélisent avec le type `Nat` les entiers positifs (à partir de 0 codé par `Z`) et avec le type `NNat` les entiers négatifs (à partir de 0 codé par o majuscule).

```
Inductive Nat:=
| Z: Nat
| S: Nat -> Nat

Inductive NNat:=
| O: NNat
| P: NNat -> NNat
```

Ainsi, la valeur `S(S(S(Z)))` représente l'entier 3 et `P(P(P(O)))` représente l'entier -3.

**1.** Prouver qu'il y a un nombre infini de valeurs de type `NNat`.

**2.** Donner la définition d'une fonction `posToNeg: Nat -> NNat` qui à partir d'un entier positif (argument `Nat`) calcule un entier négatif (type `NNat`) ayant la même magnitude (ou valeur absolue).

**3.** On ajoute le type des booléens aux définitions ci-dessus :

```
Inductive Boolean:=
| false
| true
```

Définir la fonction `isNegOf: Nat * NNat -> Boolean` qui reçoit un entier positif (de type `Nat`) et un entier négatif (de type `NNat`), et renvoie **`true`** si les arguments sont des entiers opposées, et **`false`** autrement.

**4.** Prouver la propriété suivante :

$$(\forall \mathtt{x} : \mathtt{Nat}, \exists \mathtt{y} : \mathtt{NNat}, \mathtt{isNegOf}(\mathtt{x}, \mathtt{y}) = \mathtt{true})$$

**Exercice 2**

Dans cet exercice, on dénote par $n$ le terme `S(S(...S(0)))` où `S` apparait $n$ fois. Soit la définition inductive suivante qui modélise une collection d'entiers :

```
Inductive Col:=
| empty : Col
| add   : Nat * Col -> Col
```

La valeur `empty` représente la collection vide et `add 3 (add 2 (add 1 empty))` représente la collection ayant les éléments 3, 2 et 1. Donnez les définitions inductives des fonctions suivantes :

**1.** `size: Col -> Nat` qui renvoie la taille (nombre d'éléments) de la collection en argument.

2. `union: Col * Col -> Col` qui renvoie la collection ayant comme éléments l'union des éléments des collections en argument.

3. `mem: Nat * Col -> Nat` qui renvoie le un naturel représentant le nombre d'occurrences du naturel en argument dans la collection donnée.

4. `toSet: Col -> Col` qui calcule une collection qui modélise l'ensemble des éléments de la collection en arguments. (Indication : `toSet(add(1,add(1,empty))) = add(1,empty)`.)
   Cette fonction doit respecter la spécification suivante :

$$(\forall \mathtt{c} : \mathtt{col}, \forall \mathtt{x} : \mathtt{Nat},\ \mathtt{mem}(\mathtt{x}, \mathtt{toSet}(\mathtt{c})) \leqslant 1)$$

$$(\forall \mathtt{c} : \mathtt{col}, \forall \mathtt{x} : \mathtt{Nat},\ \mathtt{mem}(\mathtt{x}, \mathtt{toSet}(\mathtt{c})) > 0 \iff \mathtt{mem}(\mathtt{x}, \mathtt{c}) > 0)$$

   où l'opération $\leqslant 1$ signifie que le terme `Nat` a au plus une application du constructeur `S`.

5. Prouver la première des propriétés ci-dessus.

**Exercice 3** – *Représentation des entiers bit-à-bit*

On considère une représentation plus efficace des entiers, où on a trois constructeurs :

1. Soit il s'agit de l'entier "zéro",
2. Soit on a le double d'un entier,
3. Soit on a le double d'un entier plus un.

Ainsi le nombre 4 est modélisé par le terme :

`(twice(twice(twice+(zero)))),`

et le nombre 5 est modélisé par :

`(twice+(twice(twice+(zero)))),`

1. Définir un type inductif pour les entiers en binaire a l'aide des constructeurs suivants :

```
Inductive Bin:=
| zero  : ___________
| twice : ___________
| twice+ : ___________
```

2. Définir par induction la fonction `incr: Bin -> Bin` pour incrémenter l'entier donné.

3. Définir par induction la fonction `binToNat: Bin -> Nat` qui transforme l'entier en binaire vers un entier dans `Nat` (en base 1).