Méthodes formelles de vérification (MF)
TD n° 1 : Structures des données inductives

Exercice 1 :

On considère les types de données suivantes, où `Nat` représente les entiers positifs (à partir de `O`) et `NNat` représente les entiers négatifs (à partir de `O`).

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

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

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

1. Prouver qu'il y a un numéro infini des valeurs du type `NNat`.

2. Donner une fonction `posToNeg : Nat -> NNat` que reçoit un entier positif (`Nat`) et renvoie un entier négatif de la même magnitude (`NNat`).

3. On ajoute le type des booléens :

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

Définir la fonction `isNegOf` qui reçoit un entier positif (`Nat`) et un entier négatif (`NNat`), et renvoi `true` si les entières saisies sont opposées, et `false` autrement. Le type de la fonction est donné par : `isNegOf: Nat * NNat -> Boolean`.

4. Prouver la propriété :

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

Exercice 2 :

Étant donné le type des collections de naturels :

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

où la valeur `empty` représente la collection vide, et `add 3 (add 2 (add 1 empty))` représente la collection $\{3, 2, 1\}$. [1]

1. Donner la fonction `mem : Nat * col -> Nat` qu'indique combien de fois le naturel donné ce trouve dans la collection donnée.

1. Le numéro n est en fait un raccourci pour la valeur `S(S(...S(0)))` où `S` apparait n fois.

2. Donner la fonction `union : col * col -> col` qui renvoie l'union des collections données.

3. Donner la fonction `size : col -> Nat` qui renvoie la taille de la collection donnée.

4. Donner la fonction `toEnsemble : col -> col` qui reçoit une collection, et renvoie un ensemble. Cette fonction doit respecter la spécification suivante :

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

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

5. Prouver la premièr des propriétés ci-dessus.

Exercice 3 : Representation 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 "zero",
2. Soit on a le double d'un entier,
3. Soit on a le double d'un entier plus un.

Ansi le numero 4 est donné par la suite des constructeurs :
(double_de(double_de(double_plus(zero)))),
et le numero 5 est donné par :
(double_plus(double_de(double_plus(zero)))),

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

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

2. Donner une fonction `incr : Bin -> Bin` pour incrémenter l'entier donné.

3. Donner une fonction `binToNat : Bin -> Nat` qui transforme l'entier en binaire donnée vers un entier en unaire (`Nat`).