

Méthodes formelles de vérification (MFVerif)

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 0) et `NNat` représente les entiers négatifs (à partir de 0).

```
Inductive Nat :=
| 0 : Nat
| S : Nat -> Nat
```

```
Inductive NNat :=
| 0 : NNat
| P : NNat -> NNat
```

Ainsi, la valeur `S(S(S(0)))` représente l'entier 3 et `P(P(P(0)))` 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 :

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

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 :

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

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

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

Ainsi le numéro 4 est donné par la suite des constructeurs :

$$(\text{double_de}(\text{double_de}(\text{double_plus}(\text{zero}))))),$$

et le numéro 5 est donné par :

$$(\text{double_plus}(\text{double_de}(\text{double_plus}(\text{zero}))))),$$

1. Définir un type inductif pour les entiers en binaire à 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`).