TP 2 - Relations et fonctions

Exercice 1 (Relations)
Comme nous l'avons vu dans le tp précédent, une relation entre deux ensembles A et B est représentée en Coq par un objet de type $A \to B \to \text{Prop}$. Pour plus de simplicité, nous considèrerons dans cet exercice les relations sur un même ensemble A (autrement dit, les objets de type $A \to A \to \text{Prop}$).

```
Variable A : Set.
```

L'objectif de cet exercice est de formaliser les propriétés vues en cours concernant les relations. Par exemple, on peut caractériser les relations *totales* (telles que *tout élément de A est en relation avec un autre élément de A*) par la définition Coq suivante :

```
Definition RTot (R : A -> A -> Prop) := forall x, exists y, R x y.
```

1. De la même façon, formalisez les notions de
   – relation fonctionnelle (`RFun`),
   – relation injective (`Inj`),
   – relation surjective (`Surj`).

2. On définit l'inverse d'une relation `R` de la manière suivante :

   ```
   Definition Inv (R : A -> A -> Prop) (x y : A) : Prop := (R y x).
   ```

   Démontrez que les propriétés d'injectivité et de fonctionnalité sont échangées par passage à l'inverse :
   – `Theorem inj_fun_inv : forall R, Inj R -> RFun (Inv R).`
   – `Theorem fun_inj_inv : forall R, RFun R -> Inj (Inv R).`

3. Montrez que l'inverse d'une relation totale est une surjection :
   – `Theorem tot_surj_inv : forall R, RTot R -> Surj (Inv R).`

Exercice 2 (Fonctions)
On va maintenant s'intéresser plus particulièrement aux fonctions totales d'un ensemble A dans lui-même. En Coq, les fonctions de A dans A sont représentées par les objets de type $A \to A$ (notez qu'en Coq, il n'existe que des fonctions totales).
Si `f` est un objet de type `A -> A` et `x` est un objet de type `A`, alors l'application de `f` à `x` est un objet de type `A` et est notée `f x` (ou bien avec des parenthèses : `(f x)`).

On formalise l'injectivité d'une fonction comme suit :

```
Definition FInj (f : A -> A) := forall x y, f x = f y -> x = y.
```

1. Formalisez de même la surjectivité (`FSurj`).
2. Définissez la composition de fonctions :
   – `Definition Comp (f : A -> A) (g : A -> A) x := (* remplir ici *).`
3. Démontrez qu'injectivité et surjectivité se composent :
   – `Theorem comp_inj : forall f g, FInj f -> FInj g -> FInj (Comp f g).`
   – `Theorem comp_surj : forall f g, FSurj f -> FSurj g -> FSurj (Comp f g).`

Exercice 3 (Ensembles finis)
Le but de cet exercice est de montrer le principe des pigeons de Dirichlet :

Si on répartit n pigeons dans un pigeonnier à $m < n$ cases,
alors il y a forcément une case habitée par au moins deux pigeons.

Autrement dit, par contraposée :

Si on a une fonction injective de $[0 \ldots n]$ dans $\mathbb{N}$,
alors elle prend forcément une valeur supérieure ou égale à n.

et donc on ne peut ranger ces $n+1$ objets dans moins que $n+1$ cases sans en mettre deux dans une case.

Pour simplifier, on va raisonner sur une notion plus forte que l'injectivité, et considérer des fonctions strictement croissantes (aussi appelées *strictement monotones*). Pour manipuler des fonctions sur l'intervalle $[0 \ldots n]$, on va considérer des fonctions sur $\mathbb{N}$ et ne parler que de ses valeurs sur l'intervalle en question. On se donne alors les définitions suivantes :

```
Definition FFInj (n:nat) (f:nat->nat) := forall x y, x<n -> y<n -> f(x)=f(y) -> x=y.
Definition FFMono (n:nat) (f:nat->nat) := forall x y, x<y -> y<n -> f(x)<f(y).
```

Pour raisonner sur les égalités et inégalités entre entiers naturels, on utilisera, comme dans le tp précédent, la tactique automatique `omega`. Pour cela, n'oubliez pas de charger la bibliothèque correspondante : `Require Import Omega`. Notez qu'`omega` permet également de détecter des contradictions arithmétiques dans le contexte.

1. Montrez que la stricte monotonicité implique l'injectivité :

   `Theorem mono_inj : forall n f, FFMono n f -> FFInj n f.`

   Indices : Il faut d'abord avoir en tête l'idée de la preuve ! Il sera judicieux d'énoncer le résultat intermédiaire `assert (x = y \/ x < y \/ x > y)`, qui est démontré par la tactique `omega`.

2. Démontrez que si une fonction est strictement monotone sur l'intervalle $[0 \ldots n]$ alors elle prend une valeur supérieure ou égale à n sur ce domaine :

   ```
   Theorem mono_domain :
      forall n f, FFMono (S n) f -> exists x, x<=n /\ f(x) >= n.
   ```

   Indices : Comme sur papier, il est nécessaire de faire une preuve par récurrence (on dit aussi, par induction) sur n. Après les introductions initiales, on utilisera la tactique `induction n` qui nous donnera deux buts : le cas de base (où $n = 0$) et l'hérédité (où l'on suppose l'énoncé pour n et on doit le démontrer pour son successeur $S\ n$, autrement dit, $n + 1$).