Licence STS, semestre 4 — 2015–16
Mathématiques pour l'Informatique 2 — 1-4 mars 2016
`http://www.lri.fr/~paulin/MathInfo2`

TP 2 - Récurrence

Le squelette de la séance est disponible à l'adresse `http://www.lri.fr/~paulin/MathInfo2/tp2.v`.

Exercice 1 (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 $[0 \dots n]$ 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 $[0 \dots n]$, 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 $[0 \dots n]$ 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, $n + 1$).

Exercice 2 (Parité)
En Coq, une définition par clôture se construit comme une définition `inductive`. Par exemple, la définition des entiers pairs par les deux règles suivantes :

$$\frac{}{\text{pair}(0)} \qquad \frac{\text{pair}(x)}{\text{pair}(x+2)}$$

s'écrit en Coq comme ceci :

```
Inductive pair : nat -> Prop :=
  | pair_0 : pair 0
  | pair_ss : forall x, pair x -> pair (S (S x)).
```

Chacune des règles est nommée, ce qui permet de les utiliser dans les raisonnements. Par exemple, si l'on doit prouver `pair 4` la commande `apply pair_ss` nous ramène à prouver `pair 2`.

Quand on a une hypothèse construite par clôture, comme `x : nat` ou `H : even x`, on peut raisonner dessus par induction, en utilisant la tactique `induction x` ou `induction H`. Le principe d'induction utilisé est généré automatiquement par Coq. Pour afficher celui généré pour `pair`, on utilisera la commande `Check pair_ind`.

1. Définir de la même façon le prédicat `impair`.
2. Prouver que le successeur d'un entier pair est impair, et vice versa.
   (Attention, doit-on raisonner par induction sur l'entier `x` *ou sur le prédicat* `pair x` *?)*
3. Prouver que tout entier est soit pair soit impair.