TD - Définitions inductives

1 Représentation binaire des entiers

On souhaite construire un type inductif représentant les entiers représentés en binaire. On commence par définir un type inductif, nommé `positive`, qui aura 3 constructeurs permettant de construire les entiers 1, $2x$ à partir de x et $2x+1$ à partir de x. On notera que ce type ne permet de décrire que les entiers strictement positifs. Nous verrons plus loin comment l'étendre pour y intégrer 0.

6- Ecrire la définition de `positive` en `Coq`. Définir 4 constantes `un`, `trois`, `quatre` et `sept`, représentant respectivement les nombres positifs 1, 3, 4 et 7.

7- Programmer une fonction récursive Sp : `positive` $\rightarrow$ `positive` de calcul du successeur (au sens des entiers de Peano) d'un nombre de type `positive`.

8- Programmer la fonction `f` : $x \mapsto 2 * x - 1$ sur le type `positive`. En déduire une manière de programmer la fonction prédécesseur Pp sur le type `positive`. On prendra la convention que le prédecesseur de 1 est 1.

9- Ecrire le principe d'induction associé à la définition du type inductif `positive`.

10- Démontrer en `Coq` le théorème suivant : $\forall p$: `positive`, $(Sp\ (f\ x)) =$ `times2` x, où `times2` est le constructeur correspondant à la fonction $x \mapsto 2 * x$. On précisera bien toutes les étapes de la démonstration, notamment les réductions intervenant lors de l'application de la tactique `simpl`.

11- (facile) Définer un nouveau type inductif `bin` permettant de décrire tous les entiers binaires positifs y compris 0. On pourra réutiliser le type inductif `positive` dans cette définition. Ecrire le principe d'induction correspondant à cette définition inductive.

12- Déduire des questions précédentes une fonction Sb : `bin` $\rightarrow$ `bin` de calcul du successeur.

13- Programmer une fonction d'addition sur le type `positive`. On déclarera une fonction `pplus` de type `positive` $\rightarrow$ `positive` $\rightarrow$ `bool` $\rightarrow$ `positive` où les deux premiers arguments de type `positive` sont les nombres à additionner et le troisième argument est un booléen indiquant à chaque appel récursif la présence ou non d'une retenue. Initialement la fonction est appelée avec l'argument booléen à `false`, i.e. pas de retenue.