TME 1 Preuves Arithmétiques

26 septembre 2017

1 Prise en main

Ouvrez un nouveau buffer Emacs, appelez-le `tme1.v`. *Proof General* doit se lancer, et le `coq-mode` s'appliquer. On rappelle les raccourcis claviers de *Proof General* les plus courants :

`C-c C-l` réorganise les fenêtres d'Emacs pour montrer toutes les informations.

`C-c C-n` avance d'une commande dans le script

`C-c C-u` recule d'une commande dans le script

`C-c C-RET` va au curseur

Entrez les commandes suivantes ; pour chacune, vérifiez de bien comprendre la réponse de Coq :

```
Print nat.
Print bool.
Print plus. Print Nat.add.
Print mult. Print Nat.mul.
Search (_ = _ + 0). Search (S (?X + ?Y) = ?X + S ?Y).
Compute (2+2).
Check (2+2=4).
```

2 Preuves arithmétiques

On suppose défini comme dans librairie standard de Coq les notions d'entiers, d'addition et de multiplication. Rédigez sur papier puis transcrivez en Coq les preuves des énoncés suivants. Attention : l'ordre des lemmes doit être respecté !

1. $a + (b + c) = (a + b) + c$ (associativité de l'addition)
2. $a + b = b + a$ (commutativité de l'addition) [1]
3. $(a + b) + c = (a + c) + b$
4. $a \times 0 = 0$ (zéro absorbant pour la multiplication)
5. $a \times (\mathrm{S}\ b) = a + a \times b$
6. $a \times b = b \times a$ (commutativité de la multiplication)
7. $a \times (b + c) = a \times b + a \times c$ (distributivité de la multiplication sur l'addition)
8. $a \times (b \times c) = (a \times b) \times c$ (associativité de la multiplication)

1. On rappelle que cette preuve repose sur deux lemmes, vus en cours. Ils s'appellent `plus_n_O` et `plus_n_Sm` dans la librairie standard de Coq.

3 Parité

1. La négation booléenne est définie dans la librairie standard ; son nom est `negb`. Prouvez que :

   ```
   Lemma negb_involutive : forall b, negb (negb b) = b.
   ```

2. Définir la fonction de parité booléenne `Fixpoint even(n:nat): bool := ...` par récurrence sur `n`, en utilisant `negb`.
3. Testez cette fonction avec `Compute (even 42).` et `Compute (even 43).`
4. Prouvez les lemmes :

   ```
   Lemma twice_is_even : forall n,  even (2 * n) = true.
   Lemma twice_plus_one_is_odd : forall n, even (2 * n + 1) = false.
   ```