TME 2 : Preuves sur les listes

1 Prise en main

Comme lors du dernier TME, nous utiliserons *ProofGeneral*, qui est une extension de l'éditeur de texte Emacs. Téléchargez le squelette du fichier `tme2.v` à l'adresse suivante : `rmonat.fr/data/teaching/MU5IN554/tme2.v`. Lancez `emacs tme2.v`.

ProofGeneral doit se lancer avec `coq-mode` actif. Pour rappel, les raccourcis les plus courants de *ProofGeneral* sont les suivants :

- `C-c C-l` réorganise les fenêtres d'Emacs pour montrer toutes les informations.
- `C-c C-n` avance d'une commande dans la preuve courante.
- `C-c C-u` recule d'une commande dans la preuve courante.
- `C-c C-RET` avance dans la preuve jusqu'à la position du curseur.

Les tactiques suivantes sont les seules nécessaires pour ce TME :

- `induction x` lance une preuve par induction structurelle sur `x`. Dans le cas des entiers, cela revient à faire une preuve par récurrence.
  N'hésitez pas à structurer vos preuves par induction avec des – :

```
induction x.
- (* Base *)
- (* Hérédité *)
```

- `intro` ajoute comme hypothèse la première variable quantifiée universellement dans le but de preuve. `intros` est une variante qui répète `intro` autant que possible.
- `simpl` applique le processus d'évaluation symbolique sur le but de preuve.
- `reflexivity` invoque la réflexivité de l'égalité (sur des buts comme `x = x`) pour terminer une preuve.
- `rewrite l` réécrit l'égalité définie dans le lemme `l` (ou l'hypothèse `l`) dans le but de preuve. La réécriture instancie par elle-même les arguments du lemme. Par défaut, elle remplace le membre de gauche de l'égalité par celui de droite.
  Il est possible d'appliquer une réécriture en sens inverse avec `rewrite <- l`.
  Il est aussi possible de faire la réécriture dans une des hypothèses avec `rewrite l in H`.
- `apply l`. Essaie de prouver le but courant en utilisant le lemme ou l'hypothèse `l`.

2 Preuves basiques

Nous allons commencer par faire des preuves basiques, proches de celles faites sur les entiers lors du TME 1.

1. Montrer que la concaténation des listes est associative.
2. Montrer que la concaténation à gauche d'une liste `l` avec la liste vide (`nil`) donne `l`.
3. Montrer que la concaténation à droite d'une liste `l` avec la liste vide (`nil`) donne `l`.

3 Preuves sur les tailles

1. Prouver que la taille de la concaténation de deux listes est la somme des tailles des listes : `length (xs ++ ys) = length xs + length ys`.
2. Prouver que `rev` préserve la taille de la liste. Pour faciliter certaines preuves sur les entiers, n'hésitez pas à chercher les résultats dans la librairie standard, via par exemple
   `Search (_ + 1 = S _).`
3. Dans cette question, nous voulons prouver deux lemmes admis durant le cours, afin de prouver la correction de `nth`.
   (a) Montrer que `nth (length xs) (xs ++ ys) = nth 0 ys`.
   (b) Montrer que `(i < length xs) -> nth i (xs ++ ys) = nth i xs`. Dans le cas de base, une preuve par l'absurde s'avère nécessaire. Vous pouvez utiliser la tactique `exfalso` pour démarrer une preuve par l'absurde, et chercher ensuite avec `Search` un lemme de la bibliothèque standard pour conclure.
   Dans le cas inductif, il peut être intéressant de faire une disjonction de cas sur `i` en utilisant la tactique `destruct i`, et d'utiliser un lemme arithmétique de la librairie standard.

4 Contenu des listes et `rev`

Dans cette partie, nous allons montrer que `rev` est idempotente : `rev (rev xs) = xs`.

1. Essayer de faire la preuve par récurrence structurelle sur `xs`.
2. La preuve précédente n'est pas faisable directement. Dans le cas d'induction du constructeur de listes, il faut prouver que `rev (rev xs ++ a :: nil) = a :: xs`, avec pour seule hypothèse `rev (rev xs) = xs`. Il y a deux manières de prouver le résultat :
   — Vous pouvez prouver un résultat plus fort dans la preuve par récurrence :
   `rev (rev xs ++ ys) = rev ys ++ xs`.
   — Vous pouvez prouver un lemme intermédiaire sur `rev` :
   `rev (xs ++ ys) = rev ys ++ rev xs`.
   Cela vous permettra de simplifier `rev (rev xs ++ a :: nil) = a :: xs` et de finir la preuve de la question (1).