TP 3 : Preuves sur les Listes

©2022 Ghiles Ziat
ghiles.ziat@epita.fr

La documentation complète de l'assistant de preuve Coq est disponible à l'url : `https://coq.inria.fr/refman/index.html`. En particulier, l'index des différentes tactiques vous sera très utile : `https://coq.inria.fr/refman/coq-tacindex.html`

Nous utiliserons *CoqIDE*, qui est un environnement de développement pour Coq. Son objectif principal est de permettre aux utilisateurs d'éditer des scripts Coq et d'avancer et de reculer dans ceux-ci.

Lancez `coqide tmen.v`. Les raccourcis les plus courants de *CoqIde* sont les suivants :

- `Ctrl + down` avance d'une commande dans la preuve courante.
- `Ctrl + up` recule d'une commande dans la preuve courante.
- `Ctrl + right` avance dans la preuve jusqu'à la position du curseur.

La documentation complete est disponible à l'url :
`https://coq.inria.fr/refman/practical-tools/coqide.html`

Solution de secours :
jsCoq, qui est un environnement Web interactif pour Coq, est disponible à l'url
`https://coq.vercel.app/`.
vous ne pourrez cependant pas sauvegarder vos preuves et devrez en faire une copie manuellement

Conseils :

N'hésitez pas à admettre une preuve et à passer à la suivante si vous êtes bloqués. Vous pouvez faire ça à l'aide de la tactique `admit`. Il vous faudra alors sortir du mode preuve en faisant `Admitted` plutôt que `Qed`, mais vous pourrez tout de même ré-utiliser les résultats admis. *Cela implique qu'admettre une proposition fausse rend tout le systeme incohérent, donc faites attention !*

Vous pourrez utiliser la commande `Print`, qui affiche la définition correspondant à l'identifiant passé en paramètre. Par exemple :

```
Print nat.
Print bool.
Print plus. Print Nat.add.
Print mult. Print Nat.mul.
```

Vous pourrez aussi utiliser la commande `Check`, qui affiche le type du terme passé en paramètre.

Par exemple :

```
Check 0.
Check (0+0=0).
```

Exercice I : Concaténation

Q1 – Ennoncez et prouvez que la concaténation de listes est associative.

Q2 – Ennoncez et prouvez que la concaténation à gauche d'une liste `l` avec la liste vide (`nil`) donne `l`.

Q3 – Ennoncez et prouvez que la concaténation à droite d'une liste `l` avec la liste vide (`nil`) donne `l`.

Exercice II : Tailles

Q1 – Définissez une fonction `length` qui prend une liste (polymorphe) et retourne sa taille.

Q2 – Prouver que la taille de la concaténation de deux listes est égale à la somme des tailles des listes :

```
Proposition concat_length_sum :
  forall (A:Set) (xs ys: list A),
  length (xs ++ ys) = length xs + length ys.
```

Q3 – Definissez une fonction `rev`, qui prend une liste $[a; b; c; \ldots; y; z]$ et retourne la liste renversée $[z; y; \ldots; c; b; a]$

Q4 – Ennoncez et 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 la commande `Search`.

Q5 – Définissez une fonction `nth` qui prend une liste `l` et un entier `n` et retourne le `n`-ième élément de `l`. Attention, votre fonction devra retourner une valeur optionnelle, étant donné que le n-ième élément d'une liste n'est pas défini dans le cas ou `n` > `length l`. Elle aura donc comme squelette le code suivant :

```
Fixpoint nth {A:Set} (i:nat) (xs:(list A)) : (option A) :=
  (* fill here *)
```

Q6 – Montrez la proposition suivante :

```
Proposition nth_concat :
  forall l1 l2,
  nth (length l1) (l1 ++ l2) = nth 0 l2.
```

Q7 – Montrez la proposition suivante :

```
Proposition nth_concat :
```

```
forall l1 l2,
(i < length l1) → nth i (l1 ++ l2) = nth i l1.
```

Indices :

— Vous avez probablement pris l'habitude de commencer vos preuves par `intros`. Cette preuve devrait être l'occasion de comprendre que c'est n'est pas toujours judicieux.

— Il peut être intéressant de faire une induction sur `l1`. Dans le cas de base, vous devriez tomber sur une contradiction. 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.

Exercice III : Contenu des listes et `rev`

Dans cette partie, nous allons montrer que `rev` est involutive : `rev (rev l1) = l1`. Vous pouvez essayer de le faire par induction, mais vous devriez échouer.

Q1 – Cette preuve n'est pas faisable directement. Dans le cas d'induction du constructeur de listes, il faut prouver que `rev (rev l1 ++ a :: nil) = a :: l1`, avec pour seule hypothèse `rev (rev l1) = l1`. Il y a (au moins) deux manières de prouver le résultat :

— Vous pouvez prouver un résultat plus fort dans la preuve par récurrence :
`rev (rev l1 ++ l2) = rev l2 ++ l1`.

— Vous pouvez prouver un lemme intermédiaire sur `rev` : `rev (l1 ++ l2) = rev l2 ++ rev l1`. Cela vous permettra de simplifier `rev (rev l1 ++ a :: nil) = a :: l1` et de finir la preuve.