TP 3 : Preuves sur les Listes

©2024 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, 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 !*

Exercice I : Concaténation

Les propositions à démontrer ne sont pas données dans cette exercice, vous allez devoir les *déduire* vous même, à partir des énnoncés.

N'oubliez pas de charger le module `List`, et les notations qui y sont définies en faisant :

```
Require Import List.
Import ListNotations.
```

Q1 – Prouvez que la concaténation à gauche d'une liste `l` avec la liste vide (`nil`) donne `l`.

Q2 – Prouvez que la concaténation à droite d'une liste `l` avec la liste vide (`nil`) donne `l`.

Q3 – Prouvez que la concaténation de listes est associative.

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`, les indices commençant à 0 (autrement dit, `nth 0 l` retournera le premier element de la liste).
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` $\geq$ `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_len_app1 :
  forall (A:Set) (l1 l2 : list A),
  nth (length l1) (l1 ++ l2) = nth 0 l2.
```

Q7 – Montrez la proposition suivante :

```
Proposition nth_len_app2 :
  forall (A:Set) (l1 l2: list A) (i: nat),
    (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 : Lorsque l'on applique `induction` sans introduire au préalable des variables, l'hypothèse d'induction, `IH`, est quantifiée universellement sur les variables qui n'ont pas encore été introduites. Cela rend `IH` plus générale et souvent plus utile dans les preuves.

— 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 la fonction `rev` est involutive :

```
Lemma rev_involutive:
  forall (A:Set) (l: list A), rev (rev l) = l.
```

Vous pouvez essayer de le faire par induction, mais vous devriez échouer : 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`.

Q1 – Nous allons donc prouver des résultats annexes. Montrer que la fonction `rev` est distributive par rapport à la concaténation.

Q2 – Montrez la proposition suivante :

```
Lemma rev_involution :
  forall (A:Set) (xs ys: list A),
    rev ((rev xs) ++ ys) = rev ys ++ xs.
```

Reprenons à présent la preuve du lemme `rev_involutive`. Il y a (au moins) deux manières de prouver le résultat :

Q3 – **Sans faire d'induction**, mais à l'aide de ré-écritures (`simpl`, `rewrite`). Vous devrez vous servir du résultat (plus fort) `rev_involution`, ainsi que des résultats de l'exercice 1.

Q4 – Faites une seconde preuve, par induction cette fois-ci, ou vous simplifierez `rev (rev l1 ++ a :: nil)` à l'aide du lemme de distributivité de la question 1.