EPITA – Ing 3
Preuve de Programmes en Coq – COQ

TP 2 : Arithmétique & Induction

©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 : Identités sur les entiers

New Tactics Unlocked !
`induction`, `rewrite`

Q1 – Montrez que 0 est l'élément neutre de l'addition :

```
Proposition plus_n_0 :
  forall n: nat, n+0=n.
```

Vous pouvez le faire par induction sur n.

Q2 – Montrez que pour tout nombre `n`, l'additionner à lui-même est égal à le multiplier par 2.

```
Proposition double_is_plus:
  forall n : nat, n+n=2*n.
```

Vous pourrez utiliser la proposition précédente.

Q3 – Montrez la proposition suivante :

```
Proposition add_succ_l :
  forall n m: nat, S n + m = S (n + m).
```

Il ne sera pas necessaire de faire un raisonnement par induction.

Q4 – Montrez la proposition suivante par induction sur `n` :

```
Proposition add_succ_r :
  forall n m: nat, n + S m = S (n + m).
```

Exercice II : Parité

New Tactics Unlocked !
`contradiction`

Nous allons à présent définir un prédicat récursif sur les entiers, puis prouver des propriétés sur notre implémentation. On rappelle que les entiers sont un type défini inductivement (dont vous pouvez voir la définition à l'aide de la commande `Print nat`.).

L'example suivant illustre la syntaxe des fonctions récursives et du *pattern-matching* :

```
Fixpoint fact (n:nat) : nat :=
  match n with
  | 0 => 1
  | S m => n *fact m end.
```

Q1 – Définissez un prédicat `even`, qui prend un entier et qui retourne une **Proposition** (type `Prop` et non pas `bool` !). Votre fonction retournera `True` si et seulement si celui-ci est pair. Vous pourrez définir votre prédicat de façon récursive (à l'aide de la commande `Fixpoint`), en raisonnant sur les cas `0`, `S 0` et `S (S n)` à l'aide d'un *pattern-matching*.

Q2 – Prouvez par induction que pour toute paire de nombres successifs, au moins un est pair :

```
Proposition one_of_two_succ_is_even:
  forall n : nat, (even n) \/ (even (S n)).
```

Q3 – Prouvez par induction que si un nombre est pair, son successeur ne l'est pas.

```
Proposition but_not_both :
  forall n : nat, even n -> ~ (even (S n)).
```

Q4 – Prouvez par induction que tout nombre de la forme `n*2` est pair.

```
Proposition double_is_even :
  forall n : nat, even (n*2).
```

Q5 – Prouvez **sans utiliser d'induction** mais en ré-utilisant vos résultats précédents que tout nombre de la forme `2*n+1` n'est pas pair. La tactique `assert` ajoute une nouvelle hypothèse à l'objectif actuel et un nouveau sous-objectif avant pour prouver l'hypothèse. Vous pourrez vous en servir pour ajouter l'hypothèse que `2*n` est pair, avant de réutiliser les résultats précédents.

```
Proposition succ_double_is_odd :
  forall n : nat, ~(even (S (n*2))).
```

Exercice III : Induction d'ordre 2

Lors de l'utilisation de l'induction, nous supposons que $P(k)$ est vraie pour prouver $P(k+1)$. En induction d'ordre n, on suppose que tous les $P(k-n), \ldots, P(k-1), P(k)$ sont vrais pour prouver $P(k+1)$.

Q1 – Prouvez le principe d'induction suivant :

```
Proposition pair_induction :
  forall (P : nat -> Prop),
    P 0 -> P 1 -> (forall n, P n -> P (S n) -> P (S (S n))) ->
      forall x, P x.
```

Indice : cette preuve est assez originale. Plutôt que de prouver Px, il est plus facile de prouver d'abord un résultat plus fort (à l'aide de la tactique `assert`), à savoir $Px \wedge P(Sx)$ puis de déduire notre but initial à partir de ce résultat. Pourquoi ? Pour faire une induction sur une paire d'entiers consécutifs x, Sx plutot que sur x. Cela aura pour effet de générer une hypothèse d'induction plus utile.

Q2 – La tactic `induction` peut être paramétrée par un principe d'induction en faisant `induction n using custom_induction`. Définissez la proposition `even_sum` qui ennonce : *"La somme de deux entiers paires est paire"*, puis prouvez la. Pour ce faire utilisez le principe d'induction `pair_induction`.

Exercice IV : Suivre une spécification

New Tactics Unlocked !
`exists`

Conseil : vous pouvez pour cette exercice utiliser la commande `Search` pour rechercher et utiliser des théorème de la librairie standard de coq. N'oubliez pas d'importer les module `Arith` dans lequel ils sont définis, en faisant `Require Import Arith`.

Considérons la défintion suivante :

```
Definition mystery (f : nat -> nat) : Prop :=
  exists t, t > 0 /\ forall x, f x = f (x+t).
```

`mystery` prends une fonction unaire `f` sur les entiers et construit une propriété sur `f`.

Q1 – Définissez une fonction qui satisfait cette propriété et prouvez que c'est bien le cas.

Q2 – Donnez la spécification pour deux fonctions (`f : nat -> nat`) et (`g : nat -> nat`) telles que `f` est l'inverse de `g`.

Q3 – Appliquez et prouvez la proposition sur deux fonctions de votre choix.