TP 1 : Logiques Booléenne & Propositionnelle

©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 !*

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

```
Print nat. Print plus. Print Nat.add.
```

Vous pouvez voir les définitions des notations infixes à l'aide de la commande : `Locate`. Par exemple :

```
Locate "<->".
```

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

Par exemple :

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

Pour ce premier TP, il est suffisant de travailler avec les tactiques :
`intro`, `intros`, `apply`, `assumption`, `left`, `right`, `destruct`, `simpl`, `reflexivity`, `unfold` et `split`

On peut prouver un théorème en coq en faisant suivre sa définition par la commande `Proof.`, suivie d'un ensemble de tactiques permettant de résoudre le but de preuve. La commande `Qed` permet de sortir du mode preuve et de revenir au mode commande lorsque tous les sous-buts ont été résolus :

```
Proposition reflex: forall n:nat, n=n.

Proof.
  intro.
  reflexivity.
Qed.
```

Exercice I : *Modus Ponens*, Transitivité, *Modus Tollens*

Q1 – Le *Modus Ponens* est une règle logique qui selon deux propositions "A" et "B" consiste à affirmer une implication *"si A alors B"*, à poser ensuite l'antécédent "et A" pour en déduire le conséquent *"donc B"*. En Coq, cette règle s'écrit :

```
Proposition modus_ponens :
  forall A B : Prop, (A -> B) -> A -> B.
```

Prouvez la règle ci-dessus.

Q2 – Ennoncez et prouvez la transitivité de l'implication.

Q3 – Le *Modus Tollens* est une règle logique qui selon deux propositions "A" et "B" consiste à affirmer que l'implication *"si A alors B"* et la négation du conséquent *"B"* entrainent la négation de l'antécédent "A".

Ennoncez et prouvez cette règle.

Exercice II : Elimination gauche et droite pour la disjonction

.

On va montrer que si on ne peut avoir ni A ni B, alors en particulier on ne peut pas avoir A (resp. B).

Q1 – Montrez :

```
Proposition a_or_b_false_left:
  forall A B : Prop, ((A \/ B) -> False) -> (A -> False).
```

Q2 – Montrez :

```
Proposition a_or_b_false_right:
  forall A B : Prop, ((A \/ B) -> False) -> (B -> False).
```

Exercice III : Lois de De Morgan (Booleens)

.

Les lois de De Morgan peuvent être enoncées de la façon suivante :

$$\neg(P \vee Q) \leftrightarrow (\neg P) \wedge (\neg Q) \quad \text{et} \quad \neg(P \wedge Q) \leftrightarrow (\neg P) \vee (\neg Q)$$

Q1 – Prouvez l'égalité suivante sur les booleens. Vous pourrez faire un raisonnement par cas à l'aide de la tactique `destruct`

```
Proposition de_morgan_bool_1 :
  forall a b:bool, negb (orb a b) = andb (negb a) (negb b).
```

Q2 – Faites de même pour la proposition suivante.

```
Proposition de_morgan_bool_2 :
  forall a b:bool, negb (andb a b) = orb (negb a) (negb b).
```

Exercice IV : Lois de De Morgan (Propositions)

Dans le monde des propositions, on ne peut plus faire de raisonnement par cas parceque les propositions ne sont pas toutes définies inductivement ! Nous allons donc prouver cette équivalence en la décomposant en deux implications.

Q1 – Prouver la proposition suivante : Notez que $\neg P$ est du sucre syntaxique pour $P \implies False$. Vous pouvez commencer votre preuve par `unfold "~"`, qui déroule syntaxiquement la négation. C'est un simple désucrage.

```
Proposition de_morgan_1 :
  forall P Q, ~(P \/ Q) -> ~P /\ ~Q.
```

Vous pourrez appliquer les résultats des exercices précédents à l'aide la tactique `apply`.

Si vous rencontrez l'erreur : *Error : unable to find an instance for the variable v.*, cela signifie que Coq ne sait pas quelle variable faire apparaitre lors de la transformation du but. Il est possible de le préciser en ajoutant la construction `with` à la tactique `apply`, par exemple : `apply myprop with (v := P)`.

Q2 – Prouver la proposition suivante :

```
Proposition de_morgan_2 :
  forall P Q,  ~P /\ ~Q -> ~(P \/ Q).
```

Q3 – En ré-utilisant les résultats précedent, prouvez à présent la loi de De Morgan qui s'énonce :

```
Proposition de_morgan_1_2 :
  forall P Q, ~P /\ ~Q <->  ~(P \/ Q).
```

Indice : L'équivalence `P <-> Q` est une forme de conjonction pour `(P -> Q) /\ (Q -> P)`.

Q4 – **Peut-on** prouver la proposition suivante ? Indice : Vous pouvez essayer, mais ne perdez pas trop de temps dessus.

```
Proposition de_morgan_classic:
  forall P Q, ~P \/ ~Q <->  ~(P /\ Q).
```

Exercice V : Consistence et Irréfutabilité

Prouver la cohérence de Coq avec l'axiome général du tiers exclu nécessite un raisonnement compliqué qui ne peut pas être effectué au sein même de Coq. Cependant, le théorème suivant implique qu'il n'est jamais faux de supposer un axiome de décidabilité (c'est-à-dire une instance de tiers exclu) pour un `Prop P` particulier. Pourquoi ? Car la négation d'un tel axiome conduit à une contradiction.

Si $\neg(P \vee \neg P)$ était prouvable, alors, par la proposition `de_morgan`, $\neg P \wedge P$ serait prouvable, ce qui serait une contradiction.

Q1 – Prouver la proposition suivante. Indice : Vous devrez utiliser deux fois la même hypothèse.

```
Proposition excluded_middle_irrefutable:
  forall P:Prop, ~ ~ (P \/ ~ P).
```

Q2 – En déduire que l'axiome de la double négation implique le tiers exclus :

```
Proposition double_neg_implies_excluded :
  (forall P: Prop, ~ ~ P -> P) -> (forall P, (P \/ ~ P)).
```

Q3 – Prouvez que si l'on a le tiers exclus, alors il est possible de prouver la dernière loi de De Mogan :

```
Proposition excluded_implies_demorgan:
  (forall P : Prop, P \/ ~ P) ->
  (forall P Q : Prop, ~ (P /\ Q) -> ~ P \/ ~ Q).
```

Indice : cette preuve necessite un peu d'immagination. Qu'obtient on si on applique le principe du tiers exclus à l'une de nos proposition ?

Q4 – Finalement, enoncez et prouvez que le principe de double négation implique la dernière loi de De Morgan.