TP 1 : Logique Booléenne

©2022 Ghiles Ziat
ghiles.ziat@epita.fr

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 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 pouvez de plus 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, vous n'avez pas le droit d'utiliser les résultats de la librairie standard

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

Pour rappel, 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 :

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

Proof.
  intro.
  reflexivity.
Qed.
```

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

Q1 – Montrez que :

```
forall A B : Prop, ((A ∨ B) → False) → (A → False).
```

Q2 – Montrez que :

```
forall A B : Prop, ((A ∨ B) → False) → (B → False).
```

Exercice III : Lois de De Morgan (Booleens)

Q1 – Prouvez les deux égalités suivantes sur les booleens.

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

Q2 –

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

Exercice IV : Lois de De Morgan (Propositions)

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

$$\overline{(P \vee Q)} \leftrightarrow (\overline{P}) \wedge (\overline{Q})$$

Nous allons prouver cette équivalence en la décomposant en deux implications.

Q1 – Prouver la proposition suivante :

```
Proposition de_morgan_1 :
  forall P Q, ~(P ∨ Q) → ~P ∧ ~Q.
```

Vous pourrez appliquer les résultats de l'exercice précédent 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 lemmes precedent, prouvez à présent la loi de De Morgan qui s'écrit en Coq de la façon suivante :

```
Proposition de_morgan :
  forall P Q, ~P ∧ ~Q ↔ ~(P ∨ Q).
```

Indice : L'équivalence `P ↔ Q` est en fait du sucre syntaxique pour `P → Q ∧ Q → P`.

Q4 – Peut-on prouver la proposition suivante ?

```
Proposition de_morgan :
  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 est toujours prudent de supposer un axiome de décidabilité (c'est-à-dire une instance de tiers exclu) pour un `Prop P` particulier. Pourquoi ? Car la négation dun 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 :

```
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.