Licence STS, semestre 4 — 2015–16
Mathématiques pour l'Informatique 2 — 5 février 2016
http://www.lri.fr/~paulin/MathInfo2

TP 1 - Eléments de logique

Le squelette de la séance est disponible à l'adresse `http://www.lri.fr/~paulin/MathInfo2/tp1.v`. On ouvrira le fichier `tp1.v` à l'aide de la commande `coqide`, on pourra alors le compléter dans la fenêtre de script de CoqIDE et utiliser le bouton ⇓ pour commencer la preuve.

Exercice 1 (Logique propositionnelle)
On va démontrer en Coq que certaines formules simples de la logique propositionnelles sont valides (c'est-à-dire toujours vraies, quelles que soient les valeurs des variables P, Q et R).

Ces énoncés vous sont donnés en syntaxe Coq. Il vous suffit de les écrire dans la fenêtre de script de CoqIDE et de les charger (bouton ⇓) pour commencer la preuve.

Astuce : on commence (très) souvent une preuve par la tactique `intros` qui permet d'introduire successivement les connecteurs et quantificateurs (universels) principaux de la conclusion du but courant.

A la fin d'une preuve, lorsque tous les buts ont été prouvés, il est nécessaire de demander à Coq de vérifier l'ensemble de la preuve, à l'aide de la commande `Qed`.

On commence par introduire les variables qui apparaitront dans les formules.
`Variables P, Q, R : Prop.`

1. `Theorem un : (P /\ Q) -> R -> (Q /\ R).`
   Quelle est la tactique Coq correspondant à la règle d'introduction de la conjonction ?
2. `Theorem curry : (P -> Q -> R) <-> (P /\ Q -> R).`
3. `Theorem contra : (P -> Q) -> ((~ Q) -> (~ P)).`
   Attention, la tactique `intros` n'introduit pas la négation, il faut utiliser `intro` à la suite.
4. `Theorem quatre : (P \/ (Q /\ R)) -> P \/ Q.`
5. `Theorem circuit : (P <-> (~ P)) -> False.`

Exercice 2 (Quantificateurs)
Nous nous intéressons, dans cet exercice, à la façon de prouver des formules qui contiennent des quantificateurs.

1. Nous allons travailler avec un prédicat quelconque sur les entiers naturels. On peut le définir de la manière suivante : `Variable p : nat -> Prop.`
   Prouvez les théorèmes suivants :
   - `Theorem fimpe : (forall x:nat, p x) -> exists y:nat, p y.`
     L'implication inverse est-elle démontrable ?
   - `Theorem notexists : (forall x:nat, ~ p x) <-> ~ (exists y:nat, p y).`
2. Considérons maintenant un prédicat à deux arguments (autrement dit une *relation*) :
   `Variable rel : nat -> nat -> Prop.`
   Prouvez le théorème suivant :
   `Theorem effe : (exists x, forall y, rel x y) -> (forall y, exists x, rel x y).`
   (Notez que l'on n'a pas précisé les types des variables dans les quantifications de cet exemple, car Coq peut les deviner (les *inférer*) tout seul.)
   L'implication inverse est-elle démontrable ?
3. Nous allons maintenant nous intéresser à la relation "être plus petit que". Cette relation peut se définir en Coq sous la forme d'une relation définie par une formule existentielle :
   `Definition le (x y:nat) : Prop := exists z:nat, y = x + z.`
   a. Prouvez que l'addition est croissante pour la relation `le` ("less or equal") :
   `Theorem plus_croissante : forall x y:nat, le x (x+y).`
   Quelle tactique doit-on utiliser pour déplier une définition ?

b. On dispose en Coq d'une tactique permettant de prouver des buts simples d'arithmétique linéaire (sans multiplication). Cette tactique s'appelle `omega`. Pour pouvoir l'utiliser, vous devez charger le *module* qui la contient :

```
Require Import Omega.
```

Prouvez que la relation `le` est réflexive :

```
Theorem le_refl : forall x:nat, le x x.
```

c. Prouvez que la relation `le` est transitive :

```
Theorem le_trans : forall (x y z:nat), le x y /\ le y z -> le x z.
```

Exercice 3 (Vérités et mensonges)

Sur l'île des purs et des pires, il y a des habitants.

```
Variable habitant : Set.
```

Ces habitants sont soit des purs, soit des pires.

```
Variable pur : habitant -> Prop.
Variable pire : habitant -> Prop.
Axiom pur_ou_pire : forall x, pur x \/ pire x.
```

Les purs disent toujours la vérité alors que les pires mentent toujours.

```
Variable say : habitant -> Prop -> Prop.
Axiom ax_pur : forall x P, pur x -> say x P -> P.
Axiom ax_pire : forall x P, pire x -> say x P -> ~P.
```

1. Montrez que si `a` et `b` sont deux habitants de l'île et que `a` dit que `b` est pire, alors au moins l'un des deux est pire.

   Astuce : introduisez le résultat intermédiaire du fait que `a` est soit pur soit pire, en utilisant la tactique `assert`, puis faîtes un raisonnement par cas.
2. Montrez que si `a` dit que `a` et `b` sont tous les deux pires, alors `a` est pire.
3. Montrez que si `a` dit `Faux` alors c'est un pire.
4. Que peut-on dire si un habitant dit qu'il est lui-même un pire ?

Exercice 4 (Relations)

Dans cet exercice, nous considèrerons des relations binaires sur un ensemble A, qui sont représentées en Coq par des objets de type $A \to A \to \text{Prop}$.

```
Variable A : Set.
```

L'objectif de cet exercice est de formaliser les propriétés concernant les relations. Par exemple, on peut caractériser les relations *totales* (telles que *tout élément de A est en relation avec un autre élément de A*) par la définition Coq suivante :

```
Definition RTot (R : A -> A -> Prop) := forall x, exists y, R x y.
```

1. De la même façon, formalisez les notions de
   - relation fonctionnelle (`RFun`) : un objet a n'est pas en relation avec deux objets b et c différents ;
   - relation injective (`Inj`) : deux objets différents a et b ne sont pas en relation avec le même objet c ;
   - relation surjective (`Surj`) : pour tout objet b, il existe un objet a tel que a est en relation avec b.
2. On définit l'inverse d'une relation `R` de la manière suivante :

   ```
   Definition Inv (R : A -> A -> Prop) (x y : A) : Prop := (R y x).
   ```

   Démontrez que les propriétés d'injectivité et de fonctionnalité sont échangées par passage à l'inverse :
   - `Theorem inj_fun_inv : forall R, Inj R -> RFun (Inv R).`
   - `Theorem fun_inj_inv : forall R, RFun R -> Inj (Inv R).`
3. Montrez que l'inverse d'une relation totale est une surjection :

- `Theorem tot_surj_inv : forall R, RTot R -> Surj (Inv R).`

Exercice 5 (Fonctions)
On va maintenant s'intéresser plus particulièrement aux fonctions totales d'un ensemble A dans lui-même. En Coq, les fonctions de A dans A sont représentées par les objets de type $A \to A$ (notez qu'en Coq, il n'existe que des fonctions totales).
Si `f` est un objet de type `A -> A` et `x` est un objet de type `A`, alors l'application de `f` à `x` est un objet de type `A` et est notée `f x` (ou bien avec des parenthèses : `(f x)`).
On formalise l'injectivité d'une fonction comme suit :

```
Definition FInj (f : A -> A) := forall x y, f x = f y -> x = y.
```

1. Formalisez de même la surjectivité (`FSurj`).
2. Définissez la composition de fonctions :
   - `Definition Comp (f : A -> A) (g : A -> A) x := (* remplir ici *).`
3. Démontrez qu'injectivité et surjectivité se composent :
   - `Theorem comp_inj : forall f g, FInj f -> FInj g -> FInj (Comp f g).`
   - `Theorem comp_surj : forall f g, FSurj f -> FSurj g -> FSurj (Comp f g).`