

Preuves assistées par ordinateur – TD n° 3

Premiers pas en Coq

La documentation du système Coq est consultable en ligne : <http://coq.inria.fr/doc/>

Démarrage du système Il existe actuellement trois manières principales d'utiliser Coq :

- via `coqide`, une interface graphique pour `gtk2`
- via `proofgeneral`, qui est un mode pour `emacs`
- via `coqtop`, une boucle d'interaction textuelle à la `ocaml`

Chaque méthode a ses aficionados (même la dernière). Nous allons ici utiliser `proofgeneral`. Pour cela, ajouter la ligne (`load-file "~letouzey/.emacs-coq"`) à votre `.emacs` puis lancer `emacs` sur un fichier Coq (d'extension `.v`). Après le traitement de la première commande, apparaît en haut à droite une zone où seront affichées les preuves en cours, et en bas à droite une zone de messages, soit des réponses du système, soit des erreurs.

En Coq, une commande est formée d'un nom de commande (commençant par une majuscule), éventuellement suivie d'un ou plusieurs arguments, et terminée par un point. Exemples :

<code>Check 0.</code>	<code>Check 2 + 2 = 5.</code>
<code>Check S.</code>	<code>Check forall x, exists y, x = 2 * y \ / x = 2 * y + 1.</code>
<code>Check nat.</code>	<code>Definition id := fun (A : Set) (x : A) => x.</code>
<code>Print nat.</code>	<code>Check id.</code>
<code>SearchAbout nat.</code>	<code>Check id nat 7.</code>

Après avoir tapé quelques commandes, soumettez-les à Coq, en utilisant l'un des moyens de navigation (icônes, menus ou raccourcis clavier). Observez le déplacement de la zone colorée marquant la partie du fichier déjà exécutée.

Passage au mode preuve L'utilisateur déclare son intention d'effectuer une preuve avec une commande de la forme

```
Lemma and_commut :
  forall A B : Prop, A /\ B <-> B /\ A.
```

On notera que cette commande donne un nom (ici : `and_commut`) au lemme, ce qui permettra de le référencer par la suite. (On peut remplacer `Lemma` par `Fact`, `Proposition` ou `Theorem`). Sans que cela soit obligatoire, on peut marquer le début de la preuve par la commande `Proof`.

Sous-buts et tactiques Une fois le mode preuve lancé, la zone supérieure gauche affiche en permanence un ou plusieurs sous-buts (*subgoals*) qu'il s'agit de démontrer. Ces sous-buts sont essentiellement des séquents de la déduction naturelle écrits verticalement : les hypothèses (nommées) sont en haut, et la conclusion figure en bas sous un trait. Dans la partie haute figurent également des déclarations de variables.

La preuve se fait à l'aide de *tactiques* (distinguées des commandes par une minuscule initiale), qui effectuent des transformations plus ou moins complexes sur le but courant. Par exemple, la tactique `intro` effectuée sur un but de la forme `A -> B` introduit une hypothèse `H : A` dans le

contexte, et remplace la conclusion par B. À chaque règle d'inférence de la déduction naturelle correspond une ou plusieurs tactiques, mais certaines tactiques permettent également d'effectuer des morceaux de preuve plus complexes, comme par exemple la résolution de contraintes linéaires en arithmétique de Presburger (tactique `omega`).

Les tactiques sont susceptibles d'engendrer de nouveaux sous-buts (correspondant aux prémisses), ou au contraire de faire disparaître le but courant (lorsque celui-ci est résolu). La preuve est terminée lorsqu'il n'y a plus de sous-but à démontrer. On repasse alors au mode « commande » avec la commande `Qed`¹. Voici par exemple une preuve complète pour l'énoncé précédent.

```
Lemma and_commut :
  forall A B : Prop, A /\ B <-> B /\ A.
Proof.
  intros; split; intros.
  destruct H; split; assumption.
  destruct H; split; assumption.
Qed.
```

Un tel «script de preuve», une fois sauvé dans un fichier `mes_preuves.v`, peut ensuite être «compilé» via la commande unix `coqc mes_preuves.v`. Si les preuves que contient ce fichier sont correctes, un fichier compilé `mes_preuves.vo` est alors produit, qui permettra de recharger ces preuves rapidement par la suite (cf. la commande `Require Import`).

Exercice 1 – Calcul propositionnel Établir en Coq les formules suivantes :

1. `forall A : Prop, A -> A`
2. `forall A B C : Prop, (A -> B) -> (B -> C) -> A -> C`
3. `forall A B : Prop, A /\ B <-> B /\ A`
4. `forall A B : Prop, A \/ B <-> B \/ A`
5. `forall A B C : Prop, (A /\ B) /\ C <-> A /\ (B /\ C)`
6. `forall A B C : Prop, (A \/ B) \/ C <-> A \/ (B \/ C)`
7. `forall A : Prop, A -> ~~A`
8. `forall A B : Prop, (A -> B) -> ~B -> ~A`
9. `forall A : Prop, ~~(A \/ ~A)`

Exercice 2 – Calcul des prédicats Après avoir effectué les déclarations suivantes

```
Parameter X Y : Set.
Parameter A B : X -> Prop.
Parameter R : X -> Y -> Prop.
```

établir en Coq les formules suivantes :

1. `(forall x, A x /\ B x) <-> (forall x, A x) /\ (forall x, B x)`
2. `(exists x, A x \/ B x) <-> (exists x, A x) \/ (exists x, B x)`
3. `(exists y, forall x, R x y) -> forall x, exists y, R x y`

1. *Quod erat demonstrandum*, le « CQFD » latin.

Exercice 3 – Reprise Refaire en Coq les exercices 1 et 4 de la feuille de TD n°1. Pour simuler la règle de raisonnement par l'absurde, on pourra déclarer l'axiome suivant :

```
Axiom not_not_elim : forall A : Prop, ~~A -> A.
```