Université Paris Diderot - M1 Informatique Année 2016–2017

Preuves assistées par ordinateur – TD n° 2

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 basée sur `gtk`
- via `proofgeneral`, qui est un mode pour `emacs`
- ou éventuellement en lançant `coqtop`, une boucle d'interaction textuelle à la `ocaml`

Chaque méthode a ses aficionados (même la dernière). Pour utiliser `proofgeneral` sur les machines de l'UFR, il suffit d'ajouter la ligne `(load-file "~letouzey/.emacs-coq")` à votre `~/.emacs` puis lancer `emacs` sur un fichier Coq (d'extension `.v`).

Une fois lancées, les deux interfaces `coqide` et `proofgeneral` proposent une disposition assez similaire : le fichier en cours d'édition est à gauche, tandis que les preuves en cours seront affichées en haut à droite, et les messages du système en bas à droite (réponses de Coq ou messages d'erreurs).

Commandes Coq

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 :

```
Check 0.
Check S.
Check nat.
Print nat.
SearchAbout nat.
```

```
Check 2 + 2 = 5.
Check forall x, exists y, x = 2 * y \/ x = 2 * y + 1.
Definition id := fun (A : Set) (x : A) => x.
Check id.
Check id nat 7.
```

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 marquera le début de la preuve par la commmande `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 doit alors utiliser la commande `Qed`[1] pour conclure la preuve et repasser au mode « commande ». 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. assumption.
  - intros. destruct H. split; assumption.
Qed.
```

Lorsque les tactiques sont séparées par des « `.` » Coq va alors les exécuter pas-à-pas. On peut aussi utiliser le « `;` » pour « chaîner » des tactiques. Ainsi `split;assumption` fait agir `assumption` sur les deux sous-buts créés par `split`.

Devant des tactiques, on peut éventuellement placer une « puce », c'est-à-dire un des marqueurs `-` ou `+` ou `*`. Ces puces sont optionnelles mais aident grandement à hiérarchiser la preuve en cours en délimitant chaque sous-partie. Il s'agit d'une nouveauté de Coq 8.4, ne pas les utiliser avec les versions antérieures.

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`).

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

Exercice 1 – Calcul propositionnel Poser l'existence de variables propositionnelles `A B C` via la commande `Parameter A B C : Prop`, puis prouver en Coq les formules suivantes :

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

Exercice 3 – Reprise Refaire en Coq les exercices 1 et 4 de la feuille de TD n^{o}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.
```