Séminaire Coq

Introduction et utilisation basique

Maxime Folschette

`http://www.irccyn.ec-nantes.fr/~folschet/coq/exos1-vide.v`

27 novembre 2012

Coq = assistant de preuves formelles

Preuve = Démonstration
= Raisonnement propre à établir une vérité
à partir de propositions initiales

Exemple

Les chats ont des poils et Socrate est un chat, donc Socrate a des poils.

Exemple de raisonnement faux

Tous ceux qui écoutent du Black Metal ont les cheveux longs.
J'ai les cheveux longs donc j'écoute du Black Metal.

Exemple

Je suis à Londres $\Rightarrow$ Je suis en Angleterre
$\not\Leftrightarrow$ Je ne suis pas à Londres $\Rightarrow$ Je ne suis pas en Angleterre

Démonstration par récurrence, par l'absurde, ...

Coq = assistant de preuves formelles

Distinction formel / informel
Preuve informelle = Lisible par un être humain

MQ : Il y a autant d'entiers naturels pairs que d'entiers naturels impairs.

1. À chaque entier naturel pair, on peut associer son successeur qui est impair. (Pourquoi ?)
   De même, à chaque entier naturel impair, on peut associer son prédécesseur qui est pair. (Pourquoi ?)
2. À chaque entier naturel pair, on peut associer son successeur qui est impair. (Pourquoi ?)
   De même pour tout entier naturel impair. (Vraiment ?)
3. C'est trivial. (Alors montrez-le.)

Distinction formel / informel
Preuve informelle = Lisible par un être humain
Preuve formelle = Objet mathématique, lisible par une machine

MQ : Il y a autant d'entiers naturels pairs que d'entiers naturels impairs.

[MQ : $\exists f : P \to Q$ bijective, avec :
$P = \{n \in \mathbb{N} \mid \exists k \in \mathbb{N}, n = 2k\} \wedge Q = \{m \in \mathbb{N} \mid \exists k \in \mathbb{N}, m = 2k+1\}$]
Soit $n \in P$. Alors $\exists k \in \mathbb{N}, n = 2k$, ainsi $n+1 = 2k+1 \in Q$.
Soit $m \in Q$. Alors $\exists l \in \mathbb{N}, m = 2l+1$, ainsi $m-1 = 2l \in P$.
[Montrer aussi que $m \geq 1$ sans quoi on pourrait avoir $m-1 < 0$.]
[Pour cela, montrer que Q est minoré et trouver son minimum.]
Soient : $f = \begin{cases} P & \to & Q \\ n & \to & n+1 \end{cases}$ et $g = \begin{cases} Q & \to & P \\ m & \to & m-1 \end{cases}$
Soit $m \in Q$. On a : $f \circ g(m) = f(g(m)) = f(m-1) = (m-1)+1 = m$.
Soit $n \in P$. On a : $g \circ f(n) = g(f(n)) = g(n+1) = (n+1)-1 = n$.
Ainsi, f est une bijection de P dans Q (de réciproque g). CQFD.

Coq = assistant de preuves formelles

Assistant = Coq ne crée pas de démonstration, il se contente de vérifier celle que vous écrivez.

Coq est le correcteur de khôle qui hurle dès que vous essayez de l'embrouiller.

Applications de Coq

- Démonstration de théorèmes complexes
  - c'est rare car la formulation est difficile
  - mais de plus en plus fréquent pour s'assurer de la véracité d'une preuve
  - et c'est enrichissant pour la communauté scientifique

Exemple

Théorème des quatre couleurs

- Preuve de programmes
  - s'assurer qu'un programme possède le comportement voulu
  - s'assurer qu'un comportement indésirable n'est pas possible
  - de plus en plus fréquent

Exemple

Électronique embarquée et critique (voiture, avion, navettes...)

Le langage de définitions de Coq est **Gallina** (inspiré d'OCaml). Il permet de définir :

- des fonctions $\rightarrow$ sur lesquelles porteront les preuves,
- des propositions $\rightarrow$ qui pourront servir d'énoncés de théorèmes,
- de lancer une démonstration...

C'est un langage purement fonctionnel

- Vraiment purement fonctionnel
- Transparence référentielle (pas d'exception, pas d'E/S...)

$\Rightarrow$ Permet d'écrire des maths

Les propositions

Les propositions permettent d'écrire des assertions mathématiques.

Les symboles :

- /\, \/ : conjonction, disjonction
- ->, <-> : implication, équivalence
- =, <> : égalité, inégalité
- <, <=, >, >=, : comparaisons (entiers naturels seulement)

Contrairement aux fonctions, elles sont figées (non simplifiables).

Elles peuvent représenter des assertions fausses ! Une proposition, contrairement à un théorème, n'est pas suivie d'une démonstration et peut énoncer n'importe quoi de syntaxiquement correct.

Définition d'un théorème

```
Theorem <nom> : <proposition>.
Proof.
  <tactiques et conclusions>
Qed.
```

Environnement de preuves

```
  Contexte
  ================
   Objectif actif

Objectifs en attente
```

- Définir un théorème lance l'environnement de preuves.
- L'environnement de preuves contient :
  - un ensemble d'objectifs (propositions à prouver pour résoudre la démonstration) — un seul objectif est actif à la fois,
  - un contexte (hypothèses).
- Au départ, le seul objectif est l'énoncé du théorème et le contexte est vide.
- Les tactiques permettent de résoudre la démonstration. Elles peuvent avoir trois effets :
  - modifier l'objectif courant,
  - créer un nouvel objectif (rajouter une étape dans la démonstration),
  - supprimer l'objectif courant (résoudre l'étape en cours).
- On peut clore une démonstration (`Qed.`) lorsqu'il ne reste plus d'objectif.

Définitions récursives

Type paramétré = Type dépendant d'un autre type.

Une `option` sur le type `X` est :

- soit un élément qui ne contient aucune valeur → `None`
- soit un élément qui contient une valeur de type `X` → `Some x`

Type récursif = Type qui fait référence à lui-même.

Un entier naturel `nat` (arithmétique de Peano) est :

- soit l'élément nul → `O`
- soit le successeur d'un entier naturel → `S n`

Les listes :

Une liste d'éléments de type `X` (`list X`) est :

- soit la liste vide → `[]`
- soit un élément de `X` (tête) et une liste de `X` (queue) → `h :: t`

Fonction récursive = Fonction qui fait référence à elle-même.

→ Problème : quid des fonctions récursives qui ne terminent pas ?
Quel résultat, Quel type ?
⇒ Mathématiquement non défini

→ Solution : forcer la terminaison
Décroissance structurelle = Une fonction ne peut s'appeler elle-même qu'avec au moins un argument strictement inférieur structurellement
X **est structurellement inférieur à** Y = On peut construire Y à partir de X
⇒ On finit toujours dans un cas dégénéré

Exemple

La fonction `length` s'appelle elle-même sur la queue de la liste, qui est structurellement plus petite. (On peut construire la liste de départ à partir de sa tête et de sa queue.)

Récurrence sur $\mathbb{N}$ (scolaire)

Si on parvient à montrer P_0 et $\forall n \in \mathbb{N}, P_n \Rightarrow P_{n+1}$, alors on a : $\forall n \in \mathbb{N}, P_n$.
Autrement dit : $(P_0 \wedge \forall n \in \mathbb{N}, P_n \Rightarrow P_{n+1}) \Rightarrow \forall n \in \mathbb{N}, P_n$.

Récurrence sur "`nat`" (structurelle)

$P(\texttt{O}) \Rightarrow (\forall \texttt{n} \in \texttt{nat}, P(\texttt{n}) \Rightarrow P(\texttt{S n})) \Rightarrow \forall \texttt{n} \in \texttt{nat}, P(\texttt{n})$

Récurrence sur "`list X`" (structurelle)

$P(\texttt{[]}) \Rightarrow (\forall \texttt{x} \in \texttt{X}, \forall \texttt{l} \in \texttt{list X}, P(\texttt{l}) \Rightarrow P(\texttt{x :: l})) \Rightarrow \forall \texttt{l} \in \texttt{list X}, P(\texttt{l})$

avec associativité à droite de "$\Rightarrow$" :

$$A \Rightarrow B \Rightarrow C \quad \equiv \quad A \Rightarrow (B \Rightarrow C) \quad \equiv \quad (A \wedge B) \Rightarrow C$$

Un petit mot sur Curry-Howard

```
H : P -> Q
f : P -> Q
```

Séminaire Coq

Introduction et utilisation basique

Maxime Folschette

http://www.irccyn.ec-nantes.fr/~folschet/coq/

27 novembre 2012

http://coq.inria.fr/
http://www.cis.upenn.edu/~bcpierce/sf/
http://www.coursera.org/course/progfun
http://www.labri.fr/perso/casteran/CoqArt/index.html