

Tutoriels Coq

Nicolas Margulies nicolas.margulies@lmf.cnrs.fr
Théo Vignon theo.vignon@lmf.cnrs.fr

La deuxième partie du projet se concentre sur la preuve formelle. Étant donné la pénibilité d'écrire des preuves entièrement formelles à la main, de nombreux logiciels appelés assistants de preuve (proof assistants/interactive theorem provers en anglais) permettent de spécifier des objets et des théorèmes, puis d'écrire des scripts de preuve, la charge de la vérification de la correction de la preuve revenant au logiciel. Il en existe plusieurs: Agda, Coq, Isabelle/HOL, Lean, Il en existe également qui sont spécialisés dans un domaine particulier: cryptographie (Proverif, cryptoverif, easycrypt, tamarin, squirrel, ...), preuve de programmes (why3, Iris, ...).

Dans ce cours, nous nous concentrerons sur Coq (qui est au passage en train de se renommer Rocq). La logique sur laquelle il se base est le calcul des constructions inductives. Coq distingue un langage permettant d'exprimer des théorèmes et des structures de données (gallina), et un langage de script de preuve (Ltac).

Exercice 0 — Installer Coq

Pour installer Coq sur votre ordinateur, suivez les instructions sur [cette page](#). Votre choix est assez libre entre le paquet opam et la Coq platform.

Coq est conçu pour être un assistant de preuve interactif. Ainsi, il est **FORTEMENT** recommandé d'avoir un éditeur de texte qui implémente le mode interactif du langage. La plupart des éditeurs de texte possèdent une extension idoine. Voici quelques exemples (d'autres sont disponibles [sur le site](#)) :

- [CoqIDE](#) est fourni avec Coq et est basique en tant qu'éditeur de texte, mais au moins possède un support Coq sans plus de configuration.
- Vim avec [Coqtail](#)
- Emacs avec [Proof General](#), et [Company Coq](#) en option (ou juste le module coq de [Doom Emacs](#))
- VSCode / VSCodium avec soit [VSCoq](#), soit [coq-lsp](#).

Séance 1 — Logique propositionnelle et du premier ordre

Cette séance concerne les fichiers `propositional.v` et `firstorder.v`. Elle s'intéresse à l'utilisation de Coq pour faire de la déduction naturelle.

Séance 2 — Entiers et types inductifs

La séance 2 s'intéressera à l'utilisation des entiers et à la notion de types inductifs (qui sert d'équivalent aux types de données algébriques d'OCaml). Elle concerne les fichiers `nats.v` et `inductives.v`.

Séance 3 — Listes et récursion

La séance 3 s'intéressera aux définitions et structures de données récursives. Elle concerne les fichiers `lists.v`, `measure.v` et `acc.v`.

Séance 4 — Extraction et décidabilité

La séance 4 s'intéresse à la notion de décidabilité d'un prédicat, et la fonction de Coq d'extraire du code OCaml qui calcule effectivement les valeurs obtenues par un théorème constructif. Les fichiers concernés sont `decidability.v` et `extraction.v`.

Séance 5 — Ltac et automatisation

La séance 5 s'intéresse à l'efficacité de l'écriture de preuve. Elle abordera plus en détail le langage Ltac, l'utilisation de la librairie standard et l'automatisation des preuves. Elle concerne les fichiers `ltac.v` et `auto.v`.

Avant la séance 5, essayez de limiter votre utilisation de la librairie standard et de l'automatisation de preuves. La plupart des exercices ont déjà été prouvés dans la librairie standard, mais leur intérêt pédagogique est limité si vous vous contentez d'aller chercher leur nom au lieu d'effectivement faire la preuve. De même, les premiers exercices peuvent probablement tous se résoudre en une seule tactique de haut niveau, mais cela ne vous permettra pas d'apprendre les bases de la preuve en Coq¹.

¹Une règle à avoir en tête est de ne laisser l'automatisation travailler que si vous y arriveriez facilement sans.