Code is Cheap, Show Me the Proof

A Rush Introduction to Coq

Paul Zhu

July 4, 2020

Today

1. Introduction
2. Tutorials
   - Logic & Curry-Howard Correspondence
   - Functional Programming & Functional Correctness
   - Formalizing Your Theory
3. Summary

Contents

Installation

- Home page: `https://coq.inria.fr`
- Github repo: `https://github.com/coq/coq`
- CoqIDE: `https://github.com/coq/coq/releases`
- From OPAM: `https://coq.inria.fr/opam-using.html`
- From source: `https://github.com/coq/coq/blob/master/INSTALL`
- Document: `https://coq.inria.fr/refman/index.html`

Coq is a formal proof management system. It provides a formal language to write mathematical definitions, executable algorithms and theorems together with an environment for semi-interactive development of machine-checked proofs.

Applications

> *Un des points les plus remarquables de Coq est la possibilité de synthétiser des programmes certifiés à partir de preuves, et, depuis peu, des modules certifiés.*
>
> *– Le Coq'Art (V8)*

- Verified C compiler: CompCert
- Verified operating system: CertiKOS
- Four color theorem
- Gödel's incompleteness theorem
- Homotopy type theory
- Iris: a higher-order concurrent separation logic framework
- Coq in Coq

Coq Workshops

- Coq Workshops (generally colocated with ITP)
- CoqPL (colocated with POPL)
- DeepSpec (colocated with PLDI since 2017)

Why Proof?

If debugging is the process of removing bugs, then programming must be the process of putting them in.

– Edsger W. Dijkstra

Why Formal Proof?

西江月 · 数学证明题

即得易见平凡，仿照上例显然。留作习题答案略，读者自证不难。

反之亦然同理，推论自然成立。略去过程 *QED*，由上可知证毕。

– 佚名

And last, but not least, thanks to the Coq team, because without Coq there would be no proof.

– *Russell O'Connor*

Gallina

A concise primitive language for expressing logical theories, using keywords:

- `Definition`
- `Inductive` / `CoInductive`
- `Fixpoint` / `CoFixpoint`
- `Axiom`
- `Theorem` / `Lemma` / `Fact` / `Example`
- etc.

Tactic Language

An extensive (and extensible) language of tactics to write proof scripts, useful commands:

- `intros`, `rewrite`, `simpl`, `reflexivity`
- `induction`, `destruct`
- `inversion`
- `split`, `left`, `right`, `exists`
- `apply`, `exact`
- `auto`
- etc.

and a "meta language" to write macros for tactics, supporting pattern matching, composing, repeating, etc.

Vernacular

An extensive language of commands to manage the proof development environment:

- notations,
- implicit arguments, and
- type classes.

Learning Coq

Books:

- Software Foundations
- Mathematical Components
- Le Coq'Art (V8)

Courses:

- CIS 500 instructed by Benjamin Pierce at University of Pennsylvania
- See this page for more

Contents

Contents

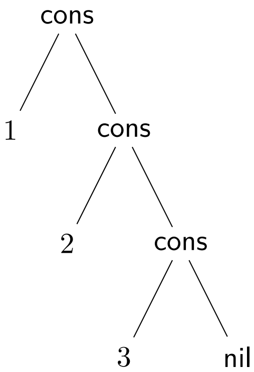
Types as Propositions

$$\boxed{a : A \iff a \text{ is a proof of } A}$$

Types	Propositions
0	$\bot$
1	$\top$
$A \times B$	$A \wedge B$
$A + B$	$A \vee B$
$A \to B$	$A \to B$
$\Pi_{x:A}\ B(x)$	$\forall x \in A, B(x)$
$\Sigma_{x:A}\ B(x)$	$\exists x \in A, B(x)$
$\mathrm{Id}_A(a, b)$	$a = b$

Contents

List



$[1, 2, 3]$

Contents

ToyLang: Syntax

$$
\begin{aligned}
\text{Term } t ::= {} & \text{zero} \\
\mid {} & \text{succ } t_1 \\
\mid {} & \text{plus } t_1 \ t_2 \\
\mid {} & \text{nil} \\
\mid {} & \text{cons } t_1 \ t_2 \\
\mid {} & \text{len } t_1 \\
\mid {} & \text{idx } t_1 \ t_2 \\
\mid {} & \text{sgt } t_1
\end{aligned}
$$

ToyLang: Value

$$\text{num-zero}\;\frac{}{\text{num zero}} \qquad \text{num-succ}\;\frac{\text{num } n}{\text{num } (\text{succ } n)}$$

$$\text{lst-nil}\;\frac{}{\text{lst nil}} \qquad \text{lst-cons}\;\frac{\text{num } n \quad \text{lst } l}{\text{lst } (\text{cons } n\ l)}$$

$$\text{value } t := \text{num } t \lor \text{lst } t$$

ToyLang: Small-Step

$$\boxed{t \to t'}$$

$$\text{ST-succ}\ \frac{t \to t'}{\text{succ}\ t \to \text{succ}\ t'}$$

$$\text{ST-plus-zero}\ \frac{\text{num}\ n}{\text{plus zero}\ n \to n} \qquad \text{ST-plus-succ}\ \frac{\text{num}\ n_1 \quad \text{num}\ n_2}{\text{plus}\ (\text{succ}\ n_1)\ n_2 \to \text{succ}\ (\text{plus}\ n_1\ n_2)}$$

$$\text{ST-plus-1}\ \frac{t_1 \to t_1'}{\text{plus}\ t_1\ t_2 \to \text{plus}\ t_1'\ t_2} \qquad \text{ST-plus-2}\ \frac{\text{num}\ t_1 \quad t_2 \to t_2'}{\text{plus}\ t_1\ t_2 \to \text{plus}\ t_1\ t_2'}$$

$$\cdots$$

ToyLang: Typing

$$\text{Type } T ::= \text{Nat} \mid \text{List}$$

$$\boxed{\vdash t : T}$$

$$\text{T-zero}\ \frac{}{\vdash \text{zero} : \text{Nat}} \qquad \text{T-succ}\ \frac{\vdash t : \text{Nat}}{\vdash \text{succ } t : \text{Nat}} \qquad \text{T-plus}\ \frac{\vdash t_1 : \text{Nat} \quad \vdash t_2 : \text{Nat}}{\vdash \text{plus } t_1\ t_2 : \text{Nat}}$$

$$\text{T-nil}\ \frac{}{\vdash \text{nil} : \text{List}} \qquad \text{T-cons}\ \frac{\vdash t_1 : \text{Nat} \quad \vdash t_2 : \text{List}}{\vdash \text{cons } t_1\ t_2 : \text{List}}$$

...

Contents

Coq CANNOT...

- prove everything automatically
- accept any function (`Fixpoint`) that actually terminates
- support classical logic directly (however, you may add axioms)

Beyond Coq

- Isabelle/HOL (set theory, classical logic)
- PVS (classical logic, refinement types)
- Agda (CuTT)
- Idris (type-driven development)
- Lean (CIC-like)
- Arend (HoTT)

Beyond Proof Assistant

- Solver-aided programming languages: Dafny, Rosette
- Software model checking framework: BLAST, CPAChecker, Ultimate Automizer, CBMC
- Modeling languages: NuSMV, Spin, TLA+, SCADE, PRISM