MPRI 2-7-2: Proof Assistants

Bruno Barras, Matthieu Sozeau

Sep 13, 2019

Goals

- Learn the basics of using Coq
  - Specification language (Gallina)
  - Modelization
  - Tactics
- Study the underlying theory (Type Theory)
  - Formalisms: Calculus of Inductive Constructions
  - Features: inductive types.
- More in-depth theory: take 2-7-1!
  - Meta-theory: extraction, strong normalization, paradoxes.

Organization

Lectures:

- 8 lectures of 3h
- Teachers: Bruno Barras (4), Matthieu Sozeau (4)

Evaluation:

- A written exam (3h). Coef:2
- 1 exercise/project (modelization and proofs in Coq) Written report, Coef: 1

Program (provisional)

- 13/9 (BB): First-order logic, λ-calculus, Simple types
- 20/9 (BB): Dependent types, Universes
- 27/9 (BB): CIC and general inductive types.
- 4/10 (BB): Theory of inductive types: inductive families, positivity, termination.
- 11/10 (MS): Advanced inductive types. *exercise handout*
- 18/10 (MS): Modelization of mathematical structures.
- 25/10 (MS): Proof by reflexion : boolean, computational.
- 8/11 (MS): Homotopy Type Theory. *exercise due*

Installing Coq

See `http://coq.inria.fr`.

- Linux: packages of major distributions, or opam
- Mac: precompiled binaries (dmg) or opam
- Windows: precompiled binaries (installer)

Beginners are invited to use CoqIDE.

Using ProofGeneral (emacs) is also possible.

Learning Coq

This module is just an initiation. See `http://coq.inria.fr/documentation` for other methods:

- Coq'Art (Y. Bertot, P. Casteran)
- Software Foundations (B. Pierce)
- Certified Programming with Dependent Types (A. Chlipala)
- ...

Help (see `http:/coq.inria.fr/community`):

- Wiki: cocorico, list: coq-club
- Forums: Discourse, Gitter, Stack Overflow
- Video tutorials (A. Bauer)

Overview

Proofs on computers

Doing proofs with computers requires:

- A language to represent objects and operations:
  integers, functions, sets, ...
- A language to represent properties of objects :
  first-order logic, higher-order logic.
- A method to construct/verify proofs
  basic rules + a way to mechanize them.

Approach based on higher-order logic:

- typed lambda-calculus for representing objects and properties
  $\neq$ set theory (first order)
- tactics or well-typed proof terms for building and verifying proofs.

Examples of case studies

In the Coq proof assistant but analogous examples in Isabelle/HOL

- Formalisation of semantics of JavaCard, certification of security functionalities (Thales, Trusted Labs)
- Proof of the 4-colors theorem (G. Gonthier, B. Werner - Inria - Microsoft Research)
- Proof of the Feit-Thompson theorem (G. Gonthier et al. - Inria - Microsoft Research)
- Development of a certified C compiler producing optimized code (Compcert, X. Leroy)
- Formalisation and reasoning on floating-point number arithmetic (S. Boldo, G. Melquiond . . .)
- Development of certified static analysers (D. Pichardie)
- . . .

First-order logic

Terms: $x \mid f(t_1, \cdots, t_n)$ (f function symbol)

Formulae:
$P(t_1, \cdots, t_n) \mid \top \mid \bot \mid A \wedge B \mid A \vee B \mid A \Rightarrow B \mid \forall x.\, A(x) \mid \exists x.\, A(x)$
(P predicate symbol)

Natural deduction rules: intro/elim rules

$$\frac{A \in \Delta}{\Delta \vdash A}(Ax) \quad \frac{}{\Delta \vdash \top}(\top - I) \quad \frac{\Delta \vdash \bot}{\Delta \vdash C}(\bot - E)$$

First-order: natural deduction rules

Conjunction

$$\frac{\Delta \vdash A \quad \Delta \vdash B}{\Delta \vdash A \wedge B}(\wedge - I) \qquad \frac{\Delta \vdash A \wedge B}{\Delta \vdash A}(\wedge - E_1) \qquad \frac{\Delta \vdash A \wedge B}{\Delta \vdash B}(\wedge - E_2)$$

Implication

$$\frac{\Delta A \vdash B}{\Delta \vdash A \Rightarrow B}(\Rightarrow - I) \qquad \frac{\Delta \vdash A \Rightarrow B \quad \Delta \vdash A}{\Delta \vdash B}(\Rightarrow - E)$$

Disjunction

$$\frac{\Delta \vdash A}{\Delta \vdash A \vee B}(\vee - I_1) \qquad \frac{\Delta \vdash B}{\Delta \vdash A \vee B}(\vee - I_2)$$

$$\frac{\Delta \vdash A \vee B \quad \Delta A \vdash C \quad \Delta B \vdash C}{\Delta \vdash C}(\vee - E) \qquad \left(\frac{}{\Delta \vdash A \vee \neg A}(EM)\right)$$

First-order: natural deduction rules

Universal quantification

$$\frac{\Delta \vdash A(x) \quad x \text{ fresh}}{\Delta \vdash \forall x.\, A(x)}(\forall - I) \qquad \frac{\Delta \vdash \forall x.\, A(x)}{\Delta \vdash A(t)}(\forall - E)$$

Existential quantification

$$\frac{\Delta \vdash A(t)}{\Delta \vdash \exists x.\, A(x)}(\exists - I) \qquad \frac{\Delta\, A(x) \vdash C \quad x \text{ fresh}}{\Delta\, \exists x.\, A(x) \vdash C}(\exists - E)$$

First-order logic in Coq

Syntax:

FOL	Coq	Intro-rule	Elim-rule
$P(t_1, \ldots, t_n)$	`P t1 .. tn`		
$A \wedge B$	`A /\ B`	`split`	`destruct <hyp>`
$A \vee B$	`A \/ B`	`left, right`	`destruct <hyp>`
$\top, \bot$	`True, False`	`trivial`	`contradiction`
$A \Rightarrow B$	`A -> B`	`intro`	`apply`
$\forall x.A(x)$	`forall x, A`	`intro`	`apply`
$\exists x.A(x)$	`exists x, A`	`exist <term>`	`destruct <hyp>`

Exercices...

Untyped λ-calculus: genesis

Church (1930s) proposed a notation for logical formulae:

- extends first-order terms with binders

  $$\Lambda ::= x \mid t_1\ t_2 \mid \lambda x.\, t \mid c \quad \text{where } c \text{ is a constant symbol}$$

- A computation rule: β-reduction

  $$(\lambda x.\, t_1)\ t_2 \to_\beta t_1[t_2/x]$$

  capture once and for all the binding constructions.
- Formulae equal up to β are identified.

Note: not seen at this point as a universal computational model (such as Turing machines)

A notation for higher-order logic

Used as a notation for higher-order logic (for both formulae and terms):

- Symbols: $\wedge$, $\vee$, $\Rightarrow$, $\top$, $\bot$, $\neg$, $\forall$, $\exists$. ($A \wedge B$ written $\wedge\ A\ B$)
- λ-abstractions in formulae:

$$\forall x.\, P(x) \text{ is written } \forall\ (\lambda x.\, P(x))$$

- λ-abstractions in terms: functions, comprehension scheme $\lambda x.\, P(x)$ denotes the "set" (or collection) of all individuals (e.g. sets) that satisfy P, and application $(t_1\ t_2)$ denotes membership $t_2 \in t_1$.

Inference rules (natural deduction style, Δ set of assumptions):

$$\frac{\Delta \vdash P\ t}{\Delta \vdash \exists\ P} \qquad \frac{\Delta \vdash \exists P \quad \Delta;\ (P\ x) \vdash C}{\Delta \vdash C}(x \text{ fresh})$$

A paradox (Kleene-Rosser, 1935)

As in naive set theory, we can build the "set of sets not belonging to themselves": $\delta = \lambda x.\neg(x\ x)$

... and whether it belongs to itself is paradoxical

$$\delta\ \delta \to_\beta \neg(\delta\ \delta)$$

Exercise: prove $\vdash \bot$, without using excluded-middle ($A \vee \neg A$).

Simply-typed λ-calculus

Church (1940) fixed the paradox by forbidding terms that do not follow a typing discipline.

Types are either

- one of the base types (to be defined),
- or $\tau \to \tau'$ the type of functions from τ to τ'.

Typing rules ($\Gamma \vdash t : \tau$)

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma \vdash t : \tau \to \tau' \quad \Gamma \vdash u : \tau}{\Gamma \vdash t\ u : \tau'} \qquad \frac{\Gamma; (x : \tau) \vdash t : \tau'}{\Gamma \vdash \lambda x{:}\,\tau.t : \tau \to \tau'}$$

Church's Higher-Order Logic (HOL)

Church's Higher-Order Logic (HOL) uses two base types:

- ι the type of individuals (e.g. sets)
- o the type of logical formulae (propositions)

Constants:

$$\top \bot : o \quad \neg : o \to o \quad \Rightarrow \wedge \vee : o \to o \to o$$
$$\forall_\tau \exists_\tau : (\tau \to o) \to o \quad =_\tau : \tau \to \tau \to o$$

(first-order quantifiers are $\forall_\iota$ and $\exists_\iota$)

Metatheory of HOL

$\delta = \lambda x.\neg(x\ x)$ cannot be well-typed (since $\tau \neq (\tau \to \tau')$)

HOL is a consistent logic: $\not\vdash \bot$

Proof assistants HOL (HOL4 HOL-Light) and Isabelle/HOL use variants of this formalism.

Next week...

Next week:

- Dependent types
- Universes