MPRI 2-7-2: Proof Assistants

Bruno Barras, Matthieu Sozeau

Dec 8, 2016

Goals

- Learn the basics of using Coq
  - Specification language (Gallina)
  - Modelization
  - Tactics
- Study the underlying theory (Type Theory)
  - Formalisms: Martin Löf's TT, Calculus of Constructions.
  - Features: inductive types.
  - Meta-theory: extraction, strong normalization, paradoxes.

Organization

Lectures:

- 8 lectures of 3h (1h30 theory + 1h30 TPs)
- Teachers: Bruno Barras (4), Matthieu Sozeau (4)

Evaluation:

- A written exam (3h), (date: 3/2/17).
- 2 exercises, written report

Program

- 12/8 (BB): λ-calculus, Curry-Howard isomorphism, dependent types.
- 12/15 (BB): Polymorphism, Calculus of Constructions, Pure Type Systems.
- 1/5 (BB): CIC and general inductive types.
- 1/12 (BB): Theory of inductive types: inductive families, positivity. *1st exercise handout*.
- 1/19 (MS): Advanced inductive types : singletons, sort restrictions, extraction and dependently-typed programming.
- 1/26 (MS): Modelization of mathematical structures. *1st ex. due, 2nd exercise handout*
- 2/2 (MS): Proof by reflexion : boolean, computational.
- 2/9 (MS): Homotopy Type Theory. *2nd ex. due*
- 2/16, 2/23: no lectures.

Installing Coq

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

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

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:

- Video tutorials (A. Bauer)
- Wiki: cocorico, list: coq-club, irc, ...

Plan

Proof Assistants

Untyped λ-calculus

Simply typed λ-calculus

Curry-Howard isomorphism

Dependent types

Proofs on computers

For doing proofs with computers we need:

- A language to represent objects : 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 builing 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 (Gemalto, 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)
- ...

Untyped λ-calculus: genesis

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

- extends first-order terms ($x \mid f(t_1, \cdots, t_n)$) 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

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

  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: $\wedge$, $\vee$, $\Rightarrow$, $\top$, $\bot$, $\neg$, $\forall$, $\exists$.
- 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

$$\delta\ \delta \rightarrow_\beta \neg(\delta\ \delta)$$

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 $\exists_\iota$)

Proof assistants HOL and Isabelle/HOL use variants of this formalism.

Metatheory of HOL

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

HOL is a consistent logic: $\nvdash \bot$

Brouwer-Heyting-Kolmogorov interpretation of proofs

Given a proposition A, what *are* the proofs of A ?

Case of conjunction:

$$\frac{\Delta \vdash A \quad \Delta \vdash B}{\Delta \vdash A \wedge B} \qquad \frac{\Delta \vdash A \wedge B}{\Delta \vdash A} \qquad \frac{\Delta \vdash A \wedge B}{\Delta \vdash B}$$

$\Rightarrow$ A proof of $A \wedge B$ could be a couple of a proof of A and a proof of B.

BHK interpretation: disjunction

Disjunction in natural deduction:

$$\frac{\Delta \vdash A}{\Delta \vdash A \lor B} \qquad \frac{\Delta \vdash B}{\Delta \vdash A \lor B} \qquad \frac{\Delta \vdash A \lor B \quad \Delta; A \vdash C \quad \Delta; B \vdash C}{\Delta \vdash C}$$

Tentatively: a proof of $A \lor B$ (in the empty context) is either a proof of A or a proof of B.

But...

BHK interpretation: disjunction, limitations

2 objections to the BHK interpretation of disjunction:

- In non-empty context:
  $A \vee B \vdash B \vee A$ holds but whether there is a proof of A or a proof of B depends on whether it is also the case of $A \vee B$.
- The excluded-middle rule:
  there is a proof of $\vdash A \vee \neg A$ for all A but there exists neither a proof of A nor a proof of $\neg A$ when A is undecidable.

Conclusions:

- The former remark is not an issue since the property still holds for proofs in the empty context
- The latter shows this interpretation is only valid in intuitionistic logic

BHK interpretation: implication, quantifiers

- A proof of $A \Rightarrow B$ is not a proof of $\neg A \vee B$, but:
  a function producing a proof of B given any proof of A.
- A proof of $\forall x.\, P(x)$ is
  a function producing a proof of $P(e)$ for all individual e
- A proof of $\exists x. P(x)$ is
  a couple of a witness e and a proof of $P(e)$

Note: the interpretation of the existential suffers the same remarks as disjunction regarding excluded-middle.

BHK interpretation: conclusions

Propositional case:

- $[\![A \wedge B]\!] = [\![A]\!] \times [\![B]\!]$
- $[\![A \vee B]\!] = [\![A]\!] + [\![B]\!]$
- $[\![A \Rightarrow B]\!] = [\![A]\!] \rightarrow [\![B]\!]$

This interpretation is sound w.r.t. intuitionistic logic and can be expressed in the simply typed λ-calculus with cartesian product and disjoint sum.

λ-calculus with cartesian product and disjoint sum

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x \; : \; A} \qquad \frac{\Gamma; (x : A) \vdash t : B}{\Gamma \vdash \lambda x : A.t \; : \; A \to B} \qquad \frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash u : A}{\Gamma \vdash t\, u \; : \; B}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash (a, b) : A \times B} \qquad \frac{\Gamma \vdash p : A \times B}{\Gamma \vdash \pi_1(p) : A} \qquad \frac{\Gamma \vdash p : A \times B}{\Gamma \vdash \pi_2(p) : B}$$

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash \iota_1(a) : A + B} \qquad \frac{\Gamma \vdash b : B}{\Gamma \vdash \iota_2(b) : A + B}$$

$$\frac{\Gamma \vdash p : A + B \quad \Gamma; (a : A) \vdash f : C \quad \Gamma; (a : B) \vdash g : C}{\Gamma \vdash p[\iota_1(a) \Rightarrow f \mid \iota_2(b) \Rightarrow g] : C}$$

Intuitionisitc propositional logic in natural deduction

$$\frac{A \in \Gamma}{\Gamma \vdash A} \qquad \frac{\Gamma; A \vdash B}{\Gamma \vdash A \to B} \qquad \frac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B}$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}$$

$$\frac{\Gamma \vdash A \vee B \quad \Gamma; A \vdash C \quad \Gamma; B \vdash C}{\Gamma \vdash C}$$

Curry-Howard isomorphism

The analogy shows that the typed λ-calculus correspond to the inference rules of intuitionisitc propositional logic decorated with the proof-object as in the BHK interpretation.

Logic	λ-calculus
proposition	type
proof	term
M is a proof of T	$\vdash M : T$

Extending the Curry-Howard isomorphism to first-order logic

Examples:

- For the universal quantification, a proof $p : \forall_\tau x.\, x = x$ is a function such that $p\ u\ :\ u = u$ and $p\ v\ :\ v = v$ for u and v elements of τ.
  The type of p is not of the form $\tau \to \tau'$
- For the existential quantification, a proof $p : \exists_{\mathbb{Z}} x.\, x^2 = 4$ could be a pair $(2, q)$ with $q\ :\ 2^2 = 4$, or a pair $(-2, q')$ with $q'\ :\ (-2)^2 = 4$.
  Again, the type of p is not of the form $\mathbb{Z} \times \tau'$.

Note: types appear in terms, terms appear in types. It is convenient to use the same syntax for both syntactic categories.

Types are terms typed by special constants, called sorts. Coq uses mainly 2 sorts: `Prop` and `Type` (with all types of `Prop` being also types of `Type`).

Dependent product

$$\frac{\Gamma; (x : A) \vdash M : B}{\Gamma \vdash \lambda x : A.M \ : \ \Pi x : A.B} \qquad \frac{\Gamma \vdash M : \Pi x : A.B \quad \Gamma \vdash N : A}{\Gamma \vdash M\ N \ : \ B[N/x]}$$

This forms a formalism called $\lambda\Pi$. It is at the basis of many formalisms of Type Theory:

- ELF
- Martin Löf's Type Theory

Next week...

Next week:

- Polymorphism: system F
- Calculus of Constructions
- Pure Type Systems