MPRI 2-7-2: Proof Assistants

Matthieu Sozeau

Dec 5, 2022
Goals

- Learn the basics of using Coq
  - Specification language (Gallina)
  - Modelization
  - Tactics
- Study the underlying theory (Type Theory)
  - Formalism: Calculus of Inductive Constructions, deriving from Martin Löf’s TT and System F.
  - Features: (co)-inductive types and universes.
- More in-depth theory: take 2-7-1!
  - Meta-theory: extraction, strong normalization, paradoxes.
Lectures:
- 8 lectures of 3h (1h30 theory + 1h30 TPs)
- Teachers: Matthieu Sozeau (MS, 4), Yannick Forster (YF, 2) and Théo Winterhalter (TW, 2)

Evaluation:
- A written exam (3h), (date: Mar 8).
- 2 exercises (modelisation and proofs in Coq), written report
Program

- Dec 5 (MS): $\lambda$-calculus, Curry-Howard. Propositional logic
- Dec 12 (MS): CC: Dependent types, universes and polymorphism, impredicative encodings.
- Christmas Break: *Project handout*.
- Jan 2: *No lecture*
- Jan 9 (MS): CIC: theory of inductive types and families.
- Jan 17 (MS): Advanced inductive types: inductive families, derived properties, positivity, guard condition.
- Jan 23 (YF): Prop vs Type and extraction, well-founded recursion, proof techniques / Equations
- Jan 30 (YF): Metatheory, axioms and models
- Feb 6 (TW): Functional, dependently-typed programming
- Feb 13 (TW): Metaprogramming? *Project due*

Exam: Feb 27
Installing and using Coq


- Web: one can always use JSCoq’s scratchpad See top-right corner quick help for keybindings.
- Local: Linux, Mac and Windows: use the Coq Platform 2022.09.0 installers with extended level to get the METAcoq package.
- Coq might also be available through your package manager, e.g. opam.

IDEs:
- Beginners are invited to use CoqIDE, which is bundled with Coq.
- VScoq for VSCode/VSCodium, ProofGeneral for Emacs, Coqtail for Vim.
Learning Coq

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

- **Coq’Art** (Y. Bertot, P. Casteran)
- **Software Foundations** series (B. Pierce *et al.*)
- **Certified Programming with Dependent Types** (A. Chlipala)
- ...

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

- **Wiki**: https://github.com/coq/coq/wiki, list: coq-club@inria.fr
- **Forums**: Coq Zulip, Coq Discourse, Stack Overflow
- **Video tutorials** on YouTube (A. Bauer)
Overview

1. Proof Assistants
2. First-order logic
3. Untyped $\lambda$-calculus
4. Simply typed $\lambda$-calculus
Proofs on computers

Doing proofs with computers requires:

- A language to represent objects and operations: integers, functions, sets, ADTs...
- 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
  ≠ 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 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)
- Undecidability Library (Y. Forster)
- . . .
First-order logic

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

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

Natural deduction rules: introduction/elimination rules

\[
\frac{A \in \Delta}{\Delta \vdash A} (Ax) \quad \frac{\Delta \vdash \top}{\top - I} \quad \frac{\Delta \vdash \bot}{\bot - E}
\]
First-order: natural deduction rules

Conjunction

\[ \Delta \vdash A \quad \Delta \vdash B \quad \therefore \Delta \vdash A \land B \] (\land - I)  
\[ \Delta \vdash A \quad \therefore \Delta \vdash A \land B \] (\land - E_1)  
\[ \Delta \vdash A \land B \quad \therefore \Delta \vdash B \] (\land - E_2)

Implication

\[ \Delta, A \vdash B \quad \therefore \Delta \vdash A \Rightarrow B \] (\Rightarrow - I)  
\[ \Delta \vdash A \Rightarrow B \quad \Delta \vdash A \quad \therefore \Delta \vdash B \] (\Rightarrow - E)

Disjunction

\[ \Delta \vdash A \quad \therefore \Delta \vdash A \lor B \] (\lor - I_1)  
\[ \Delta \vdash B \quad \therefore \Delta \vdash A \lor B \] (\lor - I_2)  
\[ \Delta \vdash A \lor B \quad \Delta, A \vdash C \quad \Delta, B \vdash C \quad \therefore \Delta \vdash C \] (\lor - E)  
\[ \Delta \vdash A \lor \neg A \quad \therefore \] (EM)
First-order: natural deduction rules

Universal quantification

\[ \frac{\Delta \vdash A(x) \quad x \text{ fresh}}{\Delta \vdash \forall x. A(x)} \quad (\forall - I) \]

\[ \frac{\Delta \vdash A(x)}{\Delta \vdash A(t)} \quad (\forall - E) \]

Existential quantification

\[ \frac{\Delta \vdash A(t)}{\Delta \vdash \exists x. A(x)} \quad (\exists - I) \]

\[ \frac{\Delta, A(x) \vdash C \quad x \text{ fresh}}{\Delta, \exists x. A(x) \vdash C} \quad (\exists - E) \]
First-order logic in Coq

Syntax:

<table>
<thead>
<tr>
<th>FOL</th>
<th>Coq</th>
<th>Intro-rule</th>
<th>Elim-rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P(t_1, \ldots, t_n) )</td>
<td>( P \ t_1 \ldots \ t_n )</td>
<td>split</td>
<td>destruct ( \text{&lt;hyp&gt;} )</td>
</tr>
<tr>
<td>( A \land B )</td>
<td>( A /\ B )</td>
<td>left, right</td>
<td>destruct ( \text{&lt;hyp&gt;} )</td>
</tr>
<tr>
<td>( A \lor B )</td>
<td>( A \ \lor \ B )</td>
<td>trivial</td>
<td>contradiction</td>
</tr>
<tr>
<td>( \top, \bot )</td>
<td>True, False</td>
<td>intro</td>
<td>apply</td>
</tr>
<tr>
<td>( A \Rightarrow B )</td>
<td>( A \rightarrow B )</td>
<td>intro</td>
<td>apply</td>
</tr>
<tr>
<td>( \forall x. A(x) )</td>
<td>forall ( x, A )</td>
<td>intro</td>
<td>destruct ( \text{&lt;hyp&gt;} )</td>
</tr>
<tr>
<td>( \exists x. A(x) )</td>
<td>exists ( x, A )</td>
<td>exist ( \text{&lt;term&gt;} )</td>
<td>destruct ( \text{&lt;hyp&gt;} )</td>
</tr>
</tbody>
</table>

Exercices...
Untyped $\lambda$-calculus: genesis

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

- extends first-order terms with binders

\[ \Lambda ::= x \mid t_1 \mid t_2 \mid \lambda x. \ t \mid c \] where $c$ is a constant symbol

- An equality rule: $\alpha$-equivalence

\[ (\lambda x. \ t) =_\alpha (\lambda y. \ t[y/x]) \quad y \text{ fresh in } t \]

- A computation rule: $\beta$-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 $\alpha$ and $\beta$ 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**: $\land$, $\lor$, $\Rightarrow$, $\top$, $\bot$, $\neg$, $\forall$, $\exists$. ($A \land B$ written $\land A \land B$)
- $\lambda$-abstractions in formulae:
  \[ \forall x. P(x) \text{ is written } \forall (\lambda x. P(x)) \]
- $\lambda$-abstractions in terms: functions, comprehension scheme
  \[ \lambda x. P(x) \text{ denotes the “set” (or collection) of all individuals (e.g. sets) that satisfy } P, \text{ and application } (t_1 \ t_2) \text{ denotes membership } t_2 \in t_1. \]

Inference rules (natural deduction style, $\Delta$ set of assumptions):

\[
\begin{align*}
\Delta \vdash P \ t & \quad & \Delta \vdash \exists P \quad \Delta; (P \ x) \vdash C \\
\Delta \vdash \exists \ P & \quad & \Delta \vdash C (x \text{ fresh})
\end{align*}
\]
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 \( (A \lor \neg A) \).
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 $\tau \rightarrow \tau'$ the type of functions from $\tau$ to $\tau'$.

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

$$
\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x : \tau} \quad \frac{\Gamma \vdash t : \tau \rightarrow \tau' \quad \Gamma \vdash u : \tau}{\Gamma \vdash tu : \tau'} \quad \frac{\Gamma ; (x : \tau) \vdash t : \tau'}{\Gamma \vdash \lambda x : \tau.t : \tau \rightarrow \tau'}
$$
Church’s Higher-Order Logic (HOL) uses two base types:

- $\iota$ the type of individuals (e.g. sets)
- $o$ the type of logical formulae (propositions)

Constants:

\[
\begin{align*}
\top, \bot &: o \\
\neg &: o \rightarrow o \\
\Rightarrow, \land, \lor &: o \rightarrow o \rightarrow o \\
\forall_\tau \exists_\tau &: (\tau \rightarrow o) \rightarrow o \\
=_{\tau}: \tau \rightarrow \tau \rightarrow o
\end{align*}
\]

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

\[ \delta = \lambda x. \neg(x\ x) \] cannot be well-typed (since \( \tau \neq (\tau \rightarrow \tau') \))

HOL is a consistent logic: \( \not\vdash \bot \)

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

- Dependent types
- Universes