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)

- **17/9 (BB)**: First-order logic, $\lambda$-calculus, Simple types
- **24/9 (BB)**: Dependent types, Universes
- **1/10 (MS)**: CIC and general inductive types.
- **8/10 (MS)**: Theory of inductive types: inductive families, positivity, termination. *exercise handout*
- **15/10 (BB)**: Advanced inductive types.
- **22/10 (BB)**: Homotopy Type Theory.
- no lecture on 29/10
- **5/11 (MS)**: Modelization of mathematical structures.
- **12/11 (MS)**: Proof by reflexion: boolean, computational. *exercise due*

Exam: 3/12
Installing Coq


- 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

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

Natural deduction rules: intro/elim rules

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

Conjunction

\[
\begin{align*}
\Delta \vdash A & \quad \Delta \vdash B \\
\Delta & \vdash A \land B \quad (\land - I) \\
\Delta \vdash A & \quad \Delta \vdash A \land B \quad (\land - E_1) \\
\Delta & \vdash B \quad \Delta \vdash A \land B \quad (\land - E_2)
\end{align*}
\]

Implication

\[
\begin{align*}
\Delta A \vdash B & \quad (\Rightarrow - I) \\
\Delta & \vdash A \Rightarrow B \\
\Delta \vdash A & \quad \Delta \vdash A \Rightarrow B \quad (\Rightarrow - E)
\end{align*}
\]

Disjunction

\[
\begin{align*}
\Delta \vdash A & \quad (\lor - I_1) \\
\Delta & \vdash A \lor B \\
\Delta \vdash B & \quad (\lor - I_2) \\
\Delta \vdash A \lor B & \quad (\lor - E) \\
\Delta \vdash A \lor B & \quad \Delta A \vdash C \quad \Delta B \vdash C \quad (\lor - E) \\
\Delta & \vdash C \\
\Delta & \vdash A \lor \neg A \quad (EM)
\end{align*}
\]

(EM)
First-order: natural deduction rules

Universal quantification

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

Existential quantification

\[
\frac{\Delta \vdash A(t)}{\Delta \vdash \exists x. A(x)} (\exists - I) \quad \frac{\Delta \vdash \exists x. A(x) \vdash C \quad x \text{ fresh}}{\Delta \vdash C} (\exists - E)
\]
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>$P \ t1 \ldots \ tn$</td>
<td>split</td>
<td>destruct &lt;hyp&gt;</td>
</tr>
<tr>
<td>$A \land B$</td>
<td>$A \ \land \ B$</td>
<td>left, right</td>
<td>destruct &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 &lt;hyp&gt;</td>
</tr>
<tr>
<td>$\exists x. A(x)$</td>
<td>exists $x$, $A$</td>
<td>exist &lt;term&gt;</td>
<td></td>
</tr>
</tbody>
</table>

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

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

- extends first-order terms with binders
  \[ \Lambda ::= x \mid t_1 \; t_2 \mid \lambda x. \; t \mid c \; \text{where } c \text{ is a constant symbol} \]
- 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 $\beta$ 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
  $\lambda x. P(x)$ 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, $\Delta$ set of assumptions):

\[
\frac{\Delta \vdash P \ t}{\Delta \vdash \exists P} \quad \frac{\Delta \vdash \exists P \quad \Delta; (P \ x) \vdash C}{\Delta \vdash C} (x \text{ fresh})
\]
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 \( (A \lor \neg A) \).
Simply-typed $\lambda$-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 $\tau \rightarrow \tau'$ the type of functions from $\tau$ to $\tau'$.

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

\[
\begin{align*}
(x : \tau) &\in \Gamma  &\Gamma \vdash t : \tau \rightarrow \tau' &\Gamma \vdash u : \tau  &\Gamma ; (x : \tau) \vdash t : \tau' \\
\Gamma \vdash x : \tau  &\Gamma \vdash t u : \tau' &\Gamma \vdash \lambda x : \tau.t : \tau \rightarrow \tau'
\end{align*}
\]
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:

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

(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