Overview of the Coq Proof Assistant

Nicolas Magaud
School of Computer Science and Engineering
The University of New South Wales

Guest lecture
Theorem Proving
Outline

• Some Theoretical Background
  • Constructive Logic
  • Curry-Howard Isomorphism

• The Coq Proof Assistant
  • Specification Language: Inductive Definitions
  • Proof Development
  • Practical Use and Demos
Constructive Logic

• Also known as Intuitionistic Logic.

• Does not take the excluded middle rule $A \lor \neg A$ into account!

• Pierce law: $(P \Rightarrow Q) \Rightarrow P \Rightarrow P$

• A proof (of existence) of $\{f \mid P(f)\}$ actually provides an executable function $f$.

• Application: extraction of programs from proofs

  $\forall a : \text{nat}, \forall b : \text{nat}, \exists q : \text{nat}, r : \text{nat} \mid a = q * b + r \land 0 \leq r < b$

From this proof, we can compute $q$ and $r$ from $a$ and $b$. 
Natural Deduction

- Propositional Logic (implication fragment)

\[
\begin{align*}
\Gamma, A & \vdash B \\
\Gamma, A & \Rightarrow B & \Rightarrow I \\
\Gamma & \vdash B & \Rightarrow E \\
\Gamma & \vdash A \Rightarrow B & \Gamma & \vdash A \\
\end{align*}
\]

- Rules for the other Connectives

\[
\begin{align*}
\Gamma & \vdash A & \Gamma & \vdash B \\
\Gamma & \vdash A \land B & \land I \\
\Gamma & \vdash A & \Gamma & \vdash B \\
\Gamma & \vdash A \land B & \land E_1 \\
\Gamma & \vdash A & \\
\Gamma & \vdash B & \land E_2 \\
\Gamma & \vdash A \lor B \\
\Gamma & \vdash A \lor B & \lor I_1 \\
\Gamma & \vdash B & \lor I_2 \\
\Gamma & \vdash A \lor B & \Gamma & \vdash A & \Gamma & \vdash C & \Gamma, B & \vdash C & \lor E \\
\Gamma & \vdash C \\
\Gamma, A & \vdash \text{False} & \Gamma & \vdash A & \Gamma & \vdash \neg A \\
\Gamma & \vdash \neg A & \neg E \\
\Gamma, A & \vdash \text{False} & \Gamma & \vdash \text{False} & \neg E \\
\Gamma & \vdash \text{False} & \text{False}_E \\
\Gamma & \vdash A \\
\end{align*}
\]
Semantics - Interpretation of a Logic (I)

- Tarski semantics
- Boolean interpretation of the logic

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>A \land B</th>
<th>A \lor B</th>
<th>A \Rightarrow B</th>
<th>\neg A \equiv A \Rightarrow False</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

0 0 1 1 0
Semantics - Interpretation of a Logic (II)

- Heyting-Kolmogorov semantics
  - A proof of $A \Rightarrow B$ is a function which for any proof of $A$ yields a proof of $B$.
  - A proof of $A \land B$ is a pair featuring a proof of $A$ and a proof of $B$.
  - A proof of $A \lor B$ is a pair $(i, p)$ with $(i = 0$ and $p$ a proof of $A)$ or $(i = 1$ and a proof of $B)$.
  - A proof of $\forall x. A$ is a function which for any object $t$ builds a proof of $A[t/x]$.
- It looks like computing and λ-calculus, doesn’t it?
Curry-Howard Isomorphism

- A formula (statement) in the logic is represented as a type in the \( \lambda \)-calculus.
- A proof of a formula \( A \) is a term of type \( A \).

<table>
<thead>
<tr>
<th>Logic</th>
<th>( \lambda )-calculus</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma, A \vdash B ) [\frac{\Gamma \vdash A \Rightarrow B}{\Gamma \vdash A \vdash B} ] ( \Gamma \vdash \lambda x : A.t : A \rightarrow B ) [\frac{\Gamma \vdash t : A \rightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash (t \ a) : B} ]</td>
<td>( \frac{\Gamma \vdash (t \ a) : B}{\Gamma \vdash a, b : A \times B} \quad \frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \text{fst} \ t : A} )</td>
</tr>
</tbody>
</table>
Curry-Howard (II)

- Dependent types: from $A \rightarrow B$ to $\forall x : A.(B x)$

- More Curry-Howard:

$$
\frac{\Gamma \vdash A}{\Gamma \vdash \forall x.A} \quad \frac{x \notin \Gamma}{\text{Γ, } x : A \vdash M : B} \quad \frac{\Gamma \vdash (\Pi x : A.B) : s}{\Gamma \vdash \lambda x : A.M : \Pi x : A.B}
$$

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

- $\lambda$-cube: classification of $\lambda$-calculi

- Calculus of Constructions (CC): the most expressive calculus in the $\lambda$-cube (polymorphism, dependent types and higher-order)

- Calculus of Inductive Constructions: CC plus Inductive Definitions and Recursion Operators (fixpoint and pattern matching)
Outline

- Some Theoretical Background
  - Constructive Logic
  - Curry-Howard Isomorphism
- The Coq Proof Assistant
  - Specification Language: Inductive Definitions
  - Proof Development
  - Practical Use and Demos
The Coq Proof Assistant

• Main Features
  • Interactive Theorem Proving
  • Powerful Specification Language
    (includes dependent types and inductive definitions)
  • Tactic Language to Build Proofs
  • Type-checking Algorithm to Check Proofs

• More concrete stuff
  • 3 sorts to classify types: Prop, Set, Type
  • Inductive definitions are primitive
  • Elimination mechanisms on such definitions
Examples of Applications of Dependent Types

- Lists and Vectors

\[
\text{append} : \forall n : \text{nat.}(\text{list } n) \rightarrow \forall m : \text{nat.}(\text{list } m) \rightarrow (\text{list } n + m)
\]

- Integer Square Root

\[
\forall n : \text{int. } 0 \leq n \rightarrow \\
\exists s, r : \text{int. } 0 \leq s \land 0 \leq r \land n = s^2 + r \land s^2 \leq n < (s + 1)^2
\]

- printf (single expression)

\[
\text{printf} : \forall t : \text{type. } t \rightarrow \text{unit}
\]
An Inductive Definition

- A mean to Reason about it
  \[ ∀P : \text{nat} → \text{Prop}, \quad P\ 0 → (∀n : \text{nat}, \quad P\ n → P\ (S\ n)) → ∀n : \text{nat}, \quad P\ n \]
- What about Computing?
  We need something like Gödel recursion operator in System T:
  \[ R_a : a → (\text{nat} → a → a) → \text{nat} → a \]
  equipped with the following rules:
  \[ R_a\ v0\ vr\ 0 → v0 \]
  \[ R_a\ v0\ vr\ (S\ p) → vr\ p\ (R_a\ v0\ vr\ p) \]
  This is achieved using Pattern Matching and Structural Recursion.
Logic Connectives as Inductive Definitions (I)

Inductive True: Prop := I: True.
Inductive False: Prop :=.

\[\text{False_ind : \forall P : Prop, False \rightarrow P}\]

Inductive and \((A : \text{Prop}) (B : \text{Prop}) : \text{Prop} :=\)
\[\text{conj : A \rightarrow B \rightarrow A \wedge B}\]

\[\text{and_ind : \forall A B P : Prop, (A \rightarrow B \rightarrow P) \rightarrow A \wedge B \rightarrow P}\]

Inductive or \((A : \text{Prop}) (B : \text{Prop}) : \text{Prop} :=\)
\[\text{or_introl : A \rightarrow A \lor B} \mid \text{or_intror : B \rightarrow A \lor B}\]

\[\text{or_ind : \forall A B P : Prop, (A \rightarrow P) \rightarrow (B \rightarrow P) \rightarrow A \lor B \rightarrow P}\]
Logic Connectives as Inductive Definitions (II)

- Inductive Constructors $\equiv$ Introduction Rules
- Induction principles (_ind) $\equiv$ Elimination Rules
- Example: how to prove $\forall A, B : \text{Prop}, A \lor B \rightarrow B \lor A$ ?
  coming soon. . .
Proof Development

• Backward Reasoning
• Tactic Based Theorem Proving
• Each tactic application refines the proof term.
• Alternatively one can give a proof term directly.
• Sometimes proofs can be performed automatically.
• Eventually a proof term is produced and type-checked.
• Demo (or_commute.v)

\[ \forall A, B : \text{Prop}, A \lor B \rightarrow B \lor A \]
Equality as an Inductive Type

- No equality as a primitive notion in Coq
- Propositional Equality: Leibnitz’ equality
  \[
  \text{Inductive } \text{eq } (A : \text{Type}) (x : A) : A \rightarrow \text{Prop} := \\
  \text{refl}_\text{equal} : x = x
  \]
  \[
  \text{eq}_\text{ind} : \forall A : \text{Type}, x : A, P : A \rightarrow \text{Prop}, P x \rightarrow \forall y : A, x = y \rightarrow P y
  \]
- Terms can also be definitionally equal ($\beta\delta_i$-convertible)
- No Extensionality Property (related to extraction matters)
  \[
  \forall f, g : A \rightarrow B, \forall x : A, f \ x = g \ x \rightarrow f = g
  \]
- Rewriting relies on the substitution principle \text{eq}_\text{ind}.
Functions Definitions

• Defining (Structural Recursive) Functions
  • Functions have to be total.
  • Definition by Pattern Matching and Guarded Fixpoint
  • Allows to define all primitive recursive functions
    (and more ... e.g. Ackermann)

• Example

  Fixpoint plus (n m : nat) struct n : nat :=
    match n with
    | O => m
    | S p => S (plus p m)
  end.

• Computational Behaviour ($\iota$-reduction)

  plus O m $\xrightarrow{\iota} m$       plus (S p) m $\xrightarrow{\iota} (S (plus p m))$
Inductive definitions and Induction

- Inductive datatypes e.g. trees (see demo later)

- Inductive predicates

  Inductive le (n : nat) : nat -> Prop :=
  | le_n : n <= n
  | le_S : forall m : nat, n <= m -> n <= S m

le is a parametric inductive type representing a relation.
As an inductive type, it also comes with an induction principle:

\[ \forall n : \text{nat}, P : \text{nat} \rightarrow \text{Prop}, \]
\[ P\ n \rightarrow (\forall m : \text{nat}, n \leq m \rightarrow P\ m \rightarrow P\ (S\ m)) \rightarrow \]
\[ \forall n0 : \text{nat}, n \leq n0 \rightarrow P\ n0 \]

- Dependent Types
Proofs: some examples

- Inductive Reasoning of basic types and on a relation (tree.v)
- Induction, Inversion Principles and Case Analysis (coins.v)
- Sometimes induction is not enough: Functional Induction (mod2.v)
- A taste of Dependent Types (dep.v)
Related Tools and Challenges

- Coq has a large standard library including Integers, Reals, Sets.

- Extraction
  - Fully certified programs can be extracted from proofs.
  - from CCInd to $F\omega$
  - Actually from Coq to ML or Haskell
  - Hoare logic and correctness proofs of imperative programs
    (see http://why.lri.fr)

- Challenges:
  - More Automation (try and formalize the sum example)
  - Friendlier Handling of Dependent Types and Dependently-typed Functions
Further Reading and Exercices

- Interactive Theorem Proving and Program Development:
  Coq’Art: The Calculus of Inductive Constructions
  by Yves Bertot and Pierre Castran


- Exercices
  - http://www.labri.fr/Perso/~casteran/CoqArt/
  - ftp://ftp-sop.inria.fr/lemme/Laurent.Thery/CoqExamples/