Introduction to Coq Proof Assistant

Qian Hu, M.Sc

McMaster University

March 3, 2010
Presentation Outline

- Overview
  - Computer Assistance in Proofs
  - Proof Assistant

- Coq
  - Introduction
  - The Coq Proof Assistant
  - Programming With Coq

- Demo

- References
Computer Assistance in Proofs

Proof consists of:
- reasoning (derivation rules) and
- computations (reductions).

Computers can assist with both:
- Computations:
  - numerical computations ("calculators")
  - symbolic computations (computer algebra systems)
- Reasoning:
  - Under development: combining computations and reasoning in one system (e.g. Maple mode for Coq)

<table>
<thead>
<tr>
<th>Automated theorem provers</th>
<th>Proof assistants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully automated</td>
<td>Interactive</td>
</tr>
<tr>
<td>System delivers a proof</td>
<td>Human delivers a proof</td>
</tr>
<tr>
<td>Specialized</td>
<td>Highly general</td>
</tr>
</tbody>
</table>
Proof Assistants

- General structure (details may vary)

- An error in the checker invalidates the whole approach!

- De Bruijn principle: we shall prefer systems with a small, reliable checker
Presentation Outline

- Overview
  - Computer in Proofs
  - Proof Assistant

- Coq
  - Introduction
  - The Coq Proof Assistant
  - Programming With Coq

- Demo

- References
Introduction

Coq: Short Intro

- Theorem prover developed in France
  - Name is the French word for rooster
  - Lots of library & tool support
- Available for all major platforms (Linux, MS Win, OSX) on the web at http://coq.inria.fr
- Written in the language O’Caml
- Working with Coq: interface
  - CoqIDE (User-friendly but not stable for Windows)
  - ProofGeneral (uses Xemacs, somehow more difficult to use but stable)
CoqIDE

Introduction

Proof Script

Current Goal

Error Reporting
The Coq Proof Assistant

Mechanisms

Coq provides mechanisms for

- Writing (encoding) specifications
- Developing interactively new proofs
- Batch checking of existing proofs
- Reusing previously developed proofs and specifications
- Extracting programs from proofs
CIC

- The formalism (metalanguage) of Coq is the *Calculus of (Co)Inductive Constructions* (CIC), a conservative extension of $\lambda C$ – $\lambda C$ plus ‘inductive types’

<table>
<thead>
<tr>
<th>$\lambda C$</th>
<th>Coq</th>
</tr>
</thead>
<tbody>
<tr>
<td>$^p$</td>
<td>Prop</td>
</tr>
<tr>
<td>$^s$</td>
<td>Set</td>
</tr>
<tr>
<td>$\square$</td>
<td>Type</td>
</tr>
<tr>
<td>$\lambda x : A.M$</td>
<td>fun $x : A =&gt; M$</td>
</tr>
<tr>
<td>$\Pi X : A.M$</td>
<td>forall $x : A, M$</td>
</tr>
<tr>
<td>$\rightarrow$</td>
<td>$\longrightarrow \rightarrow$</td>
</tr>
</tbody>
</table>
Curry-Howard Isomorphism

- Central Principle: Systems are represented (encoded) via the Curry-Howard isomorphism
- Propositions as types, Proofs as terms

\[
\begin{align*}
\text{proof} & \iff \text{term} \\
\text{Proposition} & \iff \text{type} \\
\text{proof checking} & \iff \text{type checking} \\
\text{proving / proof search} & \iff \text{term search} \\
\text{program, algorithm} & \iff \text{term} \\
\text{specification} & \iff \text{type} \\
\text{program} & \iff \text{proof}
\end{align*}
\]
Tactic-based System

- CIC is quite powerful, so automatic proof generation is quite limited

- Instead, a user provides hints in the form of proof scripts

- Proof scripts are lists of tactics, which guide Coq in generating the proof
Declarations

- Variables can be declared as follows:
  
  Coq < Variable n : nat.
  
  n is assumed

- One may assume properties for declared variables:
  
  Coq < Hypothesis Pos_n : n>0.
  
  Pos_n is assumed
Definitions

- Attach a name to an expression

Coq < Definition three := 3.
three is defined

- Definition Functions

Coq < Definition add3 (x : nat) := x + 3.
add3 is defined
Check

- The command \texttt{Check} produces the type of its argument.

\begin{verbatim}
Coq < \texttt{Check 0}.
0
  : nat
\end{verbatim}

\begin{verbatim}
Coq < \texttt{Check 2 + 3}.
2 + 3
  : nat
\end{verbatim}
Libraries

- When you start Coq the Core library is loaded at start.
- It defines many basic notions and notations (e.g. Set, nat, plus, +, <).
- For more involved properties and definitions one may need to load other libraries. E.g.:

  Coq < Require Import Arith.
Some Useful Tactics

<table>
<thead>
<tr>
<th></th>
<th>=&gt;</th>
<th>for all</th>
<th>(\land)</th>
<th>(\lor)</th>
<th>exist</th>
<th>not</th>
<th>=</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypothesis</td>
<td>apply</td>
<td>apply</td>
<td>elim</td>
<td>elim</td>
<td>elim</td>
<td>elim</td>
<td>rewrite</td>
</tr>
<tr>
<td>goal</td>
<td>intros</td>
<td>intros</td>
<td>split</td>
<td>Left or right</td>
<td>Exists v</td>
<td>intros</td>
<td>reflexivity</td>
</tr>
</tbody>
</table>
Proving $A \rightarrow A$

Coq < **Parameter** $A B C : \text{Prop.}$
A is assumed
B is assumed
C is assumed

Coq < **Lemma l : A -> A.**
1 subgoal

```
===========================================
A -> A

l < intro x.
1 subgoal

x : A
===========================================
A
```

l < exact x.
Proof completed.

l < Qed.
intro x.
exact x.
l is defined

Coq < **Check l.**

```
l :
    A -> A
```
Proving $A \land B \rightarrow B \land A$

Coq < Lemma and_commutative : $A \land B \rightarrow B \land A$.  
1 subgoal

$\vdash A \land B \rightarrow B \land A$

and_commutative < intro. 
1 subgoal

$H : A \land B$

$\vdash B \land A$

and_commutative < elim H. 
1 subgoal

$H : A \land B$

$\vdash B \land A$

and_commutative < intros. 
1 subgoal

$H : A \land B$

$H0 : A$

$H1 : B$

$\vdash A$

and_commutative < exact H1. 
1 subgoal

$H : A \land B$

$H0 : A$

$H1 : B$

$\vdash B \land A$

and_commutative < exact H0. 
Proof completed.
Demo

Demo 1

More examples in the Demo 2
References 1


- Yves Bertot, *Coq in A Hurry*, 2005


References 2

- Yves Bertot, *A Short Presentation of Coq*, 2008
- Furio Honsell, *Interactive Proof Assistants Based on Type Theory: Coq*, 2001
- Martin Henz and Aquinas Hobor, *Automated Theorem Proving*
- Wiki, *Curry–Howard correspondence*
The End

Thank You!