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:

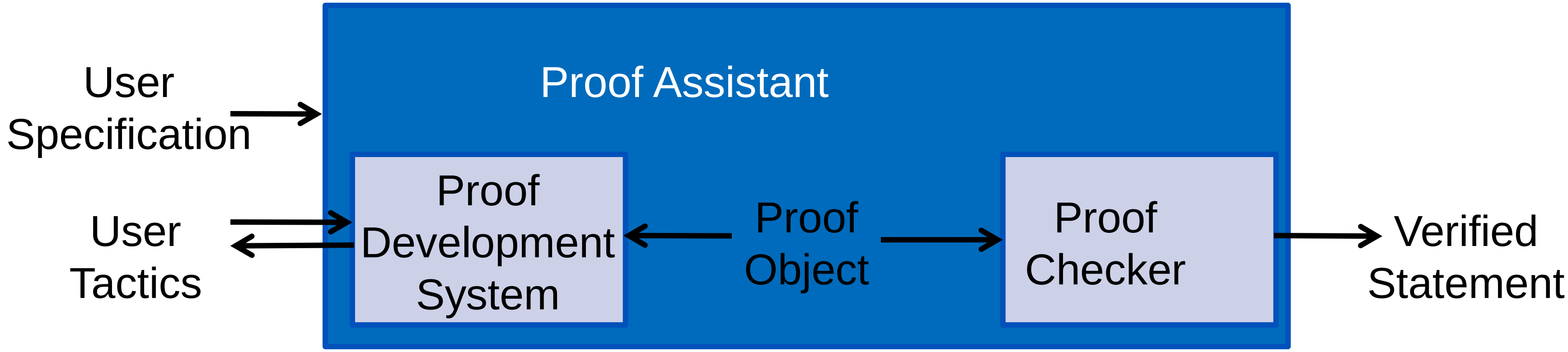
- Computations:
  - *numerical computations ("calculators")*
  - *symbolic computations (computer algebra systems)*
- Reasoning:

Automated theorem provers	Proof assistants
Fully automated System delivers a proof Specialized	Interactive Human delivers a proof Highly general

- Under development: combining computations and reasoning in one system (e.g. Maple mode for Coq)

Proof Assistants

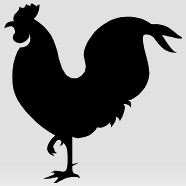
- General structure (details may vary)



- An error in the checker invalidates the whole approach!
- De Bruijn principle: we shall prefer systems with a small, reliable checker

Presentation Outline

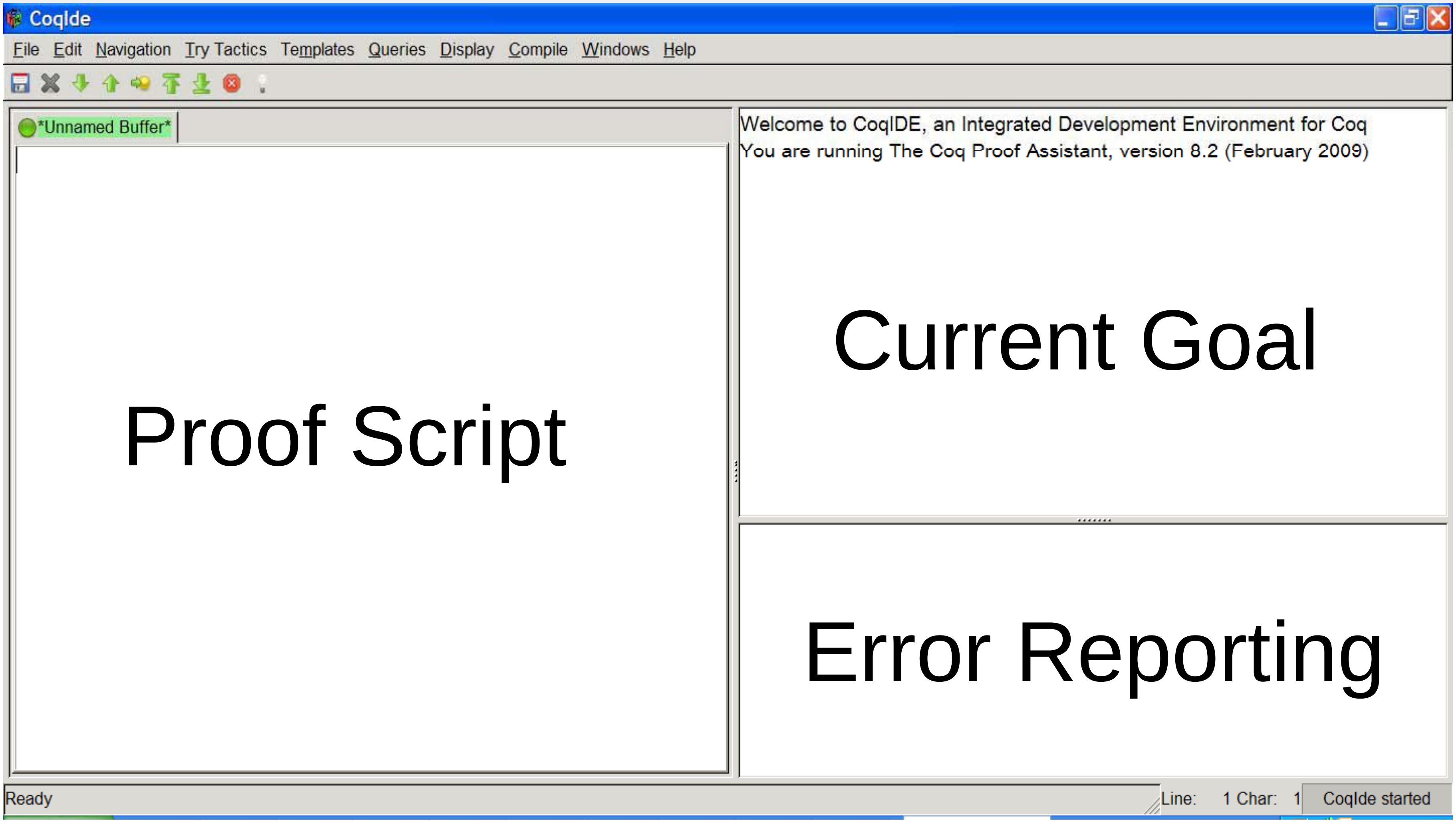
- Overview
  - Computer in Proofs
  - Proof Assistant
- Coq
  - Introduction
  - The Coq Proof Assistant
  - Programming With Coq
- Demo
- References



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


CoqIde
File Edit Navigation Try Tactics Templates Queries Display Compile Windows Help
Unnamed Buffer
Welcome to CoqIDE, an Integrated Development Environment for Coq
You are running The Coq Proof Assistant, version 8.2 (February 2009)
Current Goal
Proof Script
Error Reporting
Ready
Line: 1 Char: 1
CoqIde started

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) λC – λC plus 'inductive types'

λC	Coq
*p	Prop
*s	Set
□	Type
λx : A.M	fun x : A => M
ΠX : A.M	forall x : A, M
→	—>

Curry-Howard Isomorphism

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

proof	⇔	term
Proposition	⇔	type
proof checking	⇔	type checking
proving / proof search	⇔	term search
program, algorithm	⇔	term
specification	⇔	type
program	⇔	proof

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 **Check** produces the type of its argument.

Coq < ***Check O***.
0
: nat

Coq < **Check 2 + 3.**
2 + 3
: nat

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

	=>	for all	/\	\/	exist	not	=
Hypothesis	apply	apply	elim	elim	elim	elim	rewrite
goal	intros	intros	split	Left or right	Exists v	intros	reflexivity

Proving A -> A

```
Coq < Parameter A B C : Prop.
A is assumed
B is assumed
C is assumed

Coq < Lemma I : A -> A.
1 subgoal

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

I < intro x.
1 subgoal

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

```
I < exact x.
Proof completed.

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

Coq < Check I.
I
     : A -> A
```

Proving A /\ B -> B /\ A

```
Coq < Lemma and_commutative : A ∧ B -> B ∧ A.
1 subgoal

  ============================
   A ∧ B -> B ∧ A

and_commutative < intro.
1 subgoal

  H : A ∧ B
  ============================
   B ∧ A

and_commutative < elim H.
1 subgoal

  H : A ∧ B
  ============================
   A -> B -> B ∧ A

and_commutative < intros.
1 subgoal

  H : A ∧ B
  H0 : A
  H1 : B
  ============================
   B ∧ A
```

```
and_commutative < split.
2 subgoals

  H : A ∧ B
  H0 : A
  H1 : B
  ============================
   B

subgoal 2 is:
 A

and_commutative < exact H1.
1 subgoal

  H : A ∧ B
  H0 : A
  H1 : B
  ============================
   A

and_commutative < exact H0.
Proof completed.
```

Demo

Demo 1

More examples in the Demo 2

References 1

- Coq: http://coq.inria.fr
- The Coq development team. The Coq proof Assistant Reference Msnual, Ecole Polytechnique, INRIA, 2009
- Yves Bertot, *Coq in A Hurry*,*2005*
- Yves Bertot, Pierre Castéran, *Coq'Art: The Calculus of Inductive Constructions. Springer-Verlag,2004*
- Eduardo Giménez, Pierre Castéran, *A Tutorial on [Co-]Inductive Types in Coq, 2006*

References 2

- Adam Koprowski, *Introduction to Coq--Proving With Computer Assistance, 2007*
- 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!