Coq
TYPES Summer School, Bertinoro, Italy

Christine Paulin-Mohring
INRIA Futurs & Université Paris Sud

August 2007
Coq: outline

- Basic notions
  - Introduction
  - History
  - Architecture
  - Vernacular
  - Basic tactics

- Extra features
  - Modules
  - Computation
  - Tactic language
  - Extraction
Outline

- Basic notions
  - Introduction
  - History
  - Architecture
  - Vernacular
  - Basic tactics

- Extra features
  - Modules
  - Computation
  - Tactic language
  - Extraction
Informations on Coq

- The Coq web site coq.inria.fr
  - Official distribution + current version of development publically available
  - Reference manual
  - Library
  - User’s contributions

- Reference book : the Coq’art by Yves Bertot and Pierre Castéran

Interactive Theorem Proving and Program Development
Coq’Art: The Calculus of Inductive Constructions

- Associated web site with exercises:
  http://www.labri.fr/perso/casteran/CoqArt
History

See preface of the reference manual

- 85 (Coquand-Huet) type-checker of the pure Calculus of Constructions + hierarchy of universes, tactics
- 88 [V4.10] first version of extraction (Prop/Set) encoded impredicative inductive definitions (Paulin)
- 89-95 [V5.6] primitive inductive definitions with recursor [V5.7] Camllight [V5.8] (co)-inductive definitions with pattern-matching and fixpoints (Paulin,Giménez) new architecture (Chet Murthy)
- 96-01 [V6.1] coercions (A. Saibi), automatic tactics (Ring, S. Boutin), inversion (C. Cornes) [V6.2] Efficient reduction (B. Barras)
- 02-03 [V7] new functional architecture (Filliâtre), Modules (Courant, Chrząszcz), tactic language (Delahaye), local definitions (Herbelin), powerful pattern-matching, new extraction (Letouzey)
- 04- [V8] Coq’art, new syntax, predicative Set, byte-code compilation (B. Grégoire) . . .
Remarkable Developments

- JavaCard architecture modeling (Trusted Logic, EAL7 certification)
- Fundamental theorem of Algebra (Constructive proof, CCorn Nijmegen)
- Four color theorem (Gonthier-Werner)
- Certified compiler for C (Leroy et al)
- Primality checker (Théry et al)
- ...
Architecture

- Implemented in OCAML
- Batch compilation: coqc
- Interfaces: coqide (lablgtk), Proof General (emacs), web...
- Kernel
  - strong rules for inductive definitions
  - complete pattern-matching, guarded definitions
  - modules
  - possible efficient byte-code reduction
- Extensions
  - universe inference
  - coercions, implicit arguments
  - advanced pattern-matching
  - notations, tactic language, functional definitions
  - extraction
Basic notions

Basic terms

Prop, Set, Type

\[ \forall x, y, B \quad \forall (x, y : A) z, B \\]
\[ A \rightarrow B \rightarrow C \quad (A \rightarrow B) \rightarrow C \]

\[ \text{fun} \ x, y \Rightarrow t \quad \text{fun} \ (x, y : A) z \Rightarrow t \]

\[ t \times y \quad t \ (u \ x) \]

match \ t \ with \ .. \ end

let \ x := t \ in \ u

fix \ f \ (x : A) : B := t \]
Basic notions

▶ Propositional logic

\[ A \land B \quad A \lor B \quad A \leftrightarrow B \quad \sim A \]

▶ predicate logic

exists x, P
x=y

▶ data-types

A * B \quad (x, y)
A + B

▶ arithmetic (nat, Z ...)

0 45 n+m n*m
Basic Vernacular

Enriching the environments

**Definition** `name : type := term.`

**Variable/Hypothesis** `name : type`.

(co)inductive definition

**Inductive/CoInductive** `name params : type := constructors.`

**Fixpoint** `name params {struct arg}: type := term.`

**CoFixpoint** `name params : type := term`.

Bottom-up development

**Lemma/Theorem** `name : type`.

**Save/Qed/Defined**.

**Save/Qed**: Opaque constants not unfolded in computation.
Libraries

▶ Section mechanism

Section name.
  Variable ...
  Hypothesis ...
  Definition ...
  Lemma ...
End name.

▶ Loading precompiled libraries

Require Export library.

▶ Modules
Basic help

**Check**  *term*.
**Print**  *ident*.
**SearchAbout**  *ident*.
**Eval**  *compute in term*.
**Print**  **LoadPath**.
Basic tactics

- exact `term`; assumption
- intros
- apply `term`; constructor
- case `term`; elim `term`
- change `term`; unfold `term`; simpl
- fix `n`; cofix
Composed tactics

Inductive types

- inversion \textit{term}; induction \textit{term}
- equality : rewrite \textit{term}; injection \textit{term}; discriminate

Automatic tactics

- Database search: auto, trivial
- Decision procedure: tauto, firstorder, omega
- Propositional simplification: intuition
- Reflexive tactic: ring

Tacticals

- \textit{tac}_1; \textit{tac}_2  \textit{tac}; [\textit{tac}_1|\ldots|\textit{tac}_n]
- try \textit{tac}, \textit{tac}_1||\textit{tac}_2
Outline

- Basic notions
  - Introduction
  - History
  - Architecture
  - Vernacular
  - Basic tactics

- Extra features
  - Modules
  - Computation
  - Tactic language
  - Extraction
Modules

- Extension of Ocaml modules systems to type theory
  - Modules are not terms
  - Interfaces contains both abstract and explicit declarations.
  - Parameterized Modules (functors)

- Meta-theory by J. Courant (98)
- Implementation by J. Chrzaszcz (02)
- Non logical features : notations, Hint databases, Extraction are compatible with the module system.

Demo.
Computation

- Computation is part of type-checking (verification of convertibility)
  \[ \Gamma \vdash U : s \quad \Gamma \vdash t : T \quad T \equiv U \]
  \[ \Gamma \vdash t : U \]

- Internal programming language (functional kernel of CAML)
- Possibility to write complex programs and use them in proofs
  - Four colors theorem
  - Reflexive tactics
- Efficient reduction technics (byte-code compiling, B. Grégoire)
- Ongoing work on proof-irrelevance

Demo.
Reflexive tactics

- We have a function $s2bool : \forall AB, A + B \rightarrow bool$ and a proof
  $strue : \forall p : A + B, s2bool p = true \rightarrow A$

- If there is a (closed) proof $thm$ of type $\forall x : A, (P x) + Q$ then for closed term $a$:

  $strue(thm a)(refleq true) : P a$

Proof is well-typed when $s2bool(thm a) \equiv true$

- Problems
  - $a : A$ should be closed (reification), but we also prefer to keep $a$ small
  - $thm$ should be proved and should reduce efficiently

- Application
  - Ring, (R)Omega, Setoid Rewrite...
  - Interfaces between Coq and other systems using traces.
Example of reflection

**Inductive** form : Set :=

  T | F | Var : nat -> form

| Conj : form -> form -> form.

**Def** env := list Prop.

**Def** find_env (e:env) (n:nat) : Prop := ...

**Fixpoint** interp (e:env) (f:form) struct f :=

  match f with
  T => True | F => False

  | Conj A B => interp e A /\\ interp e B

  | Var n => find_env e n

  end.

**Eval compute in**

  (interp ((1=1)::(0=0)::nil)

  (Conj (Var 0) (Conj (Var 1) (Var 1))))).

  = 1 = 1 /\\ 0 = 0 /\\ 0 = 0 : Prop
Simplification function

**Def** simplform : form -> form.

**Lemma** simplcorrect :
    
    forall e f, interp e (simplform f) -> interp e f.

In order to use the simplification

- Find *e* and *f* such that *interp e f* is convertible with the goal (reification)
- Apply *simplcorrect*
- Compute *simplform* and *interp*
Tactic language

Ltac designed by D. Delahaye

- A way to write complex tactics without writing ML code.
- A specific language
  - specific patterns for matching goals (non-linear)
    ```
    match goal with
      id:?A / ?B |- ?A => case id; trivial
      | _ => idtac
    end
    ```
  - match goal with
  - specific notion of backtracking
    - patterns are tried until the right-hand side succeeds
  - specific constructions: **fresh name, type of term** ...

Demo
Extraction

- Distinction between \textsf{Prop} and \textsf{Set} given by the user
- Parts in \textsf{Prop} are erased
- Extraction of Ocaml programs (P. Letouzey)
  - Termination problems \texttt{fun} \texttt{x : ⊥ ⇒ t}
  - Typing problems
    - polymorphism
    - generalised inductive types
    - dependent types: \texttt{if b then 0 else "hello"}
- Efficiency

Demo
Proposed exercises

http://www.lri.fr/~paulin/TypesSummerSchool/exercises.html