Coq,
a formal proof development environment combining logic and programming

Hugo Herbelin
Coq in a nutshell
(http://coq.inria.fr)

A logical formalism that embeds an executable typed programming language: the Calculus of Inductive Constructions [Coquand-Paulin 1988]

A small trustable kernel that checks that proofs are correct

A high-level concrete language of functions and specifications

    Inductive bool := true | false.
    Definition neg c := match c with true => false | false => true end.
    Theorem neg_involutive : forall c:bool, neg (neg c) = c.

A language of semi-interactive tactics to solve specifications and theorems

    Theorem neg_involutive : forall c:bool, neg (neg c) = c.
    Proof. induction c; auto. Qed.

Libraries on arithmetics, analysis, logic, data types, algebra

An extraction mechanism to ML, Haskell and Scheme

40 direct contributors to the archive over 24 years + a community of users
Top applications

Certification of a C compiler (Xavier Leroy and the CompCert team)
Full formalization of the 4-color theorem proof (Gonthier - Werner)
Extensive repository of constructive mathematics (Nijmegen)
Certification of a commercial JavaCard applet (Gemalto)
Exact real arithmetic
Hoare logic toolbox (Y-not, Boston & Cambridge)
SAT certification

...
The Calculus of Inductions Constructions (CIC)

Terms and types

\[ t, u, T, U ::= \text{fun } x : T \Rightarrow t \mid x \mid tt \mid \text{let } x := t \text{ in } u \mid c \]

- \( \lambda \)-calculus with definitions
- dependent types
- sorts
- inductive/coinductive types
- pattern-matching on constructors
- structural recursion

Contexts of declarations

\[ \Gamma ::= \epsilon \]

- \( \Gamma, \text{Definition } c := t : T \)
- \( \Gamma, \text{Parameter } x : T \)
- \( \Gamma, \text{Inductive } I_1(x : T_1) : T_1 := C_1 : U_1 \) with \( I_n(x : T_n) : T_n := C_n : U_n \)

Judgments

\[ \Gamma \vdash T : \text{Prop} \quad \text{means } T \text{ is a formula} \]
\[ \Gamma \vdash T : \text{Type}_n \quad \text{means } T \text{ is a type} \]
\[ \Gamma \vdash t : T \quad \text{means } t \text{ is a program or a data of type } T \text{ (if } T \text{ is a type)} \]
\[ \Gamma \vdash t : T \quad \text{means } t \text{ is a proof of } T \text{ (if } T \text{ is a formula)} \]

\( \leftrightarrow \) relies on the Curry-Howard “identity” between proofs and programs
The verification kernel

8 000 lines of Objective Caml code out of 190 000 lines for the full Coq
CIC terms provide with a notion of cross-verifiable disk-storable proof-object
Partly certified in Coq itself (Bruno Barras)
The high-level specification and programming languages

macro-generation of
induction schemes    type inference    termination evidence
↓                        ↑                        ↓

Function add (k : key) (x : elt) (s : t elt) {struct s} : t elt :=
  match s with
  | nil => (k,x) :: nil
  | (k',y) :: l =>
    match X.compare k k' with
    | LT _ => (k,x)::s
    | EQ _ => (k,x)::l
    | GT _ => (k',y) :: add k x l
  end
end.

custom notations and implicit arguments
The proof languages

A set of tactics ranging from low-level logical rules...

e.g. lemma application or induction

... to high-level automation tactics

e.g. decision of Presburger arithmetic, closed equational reasoning, semi-decision of first-order logic, approximate decision of closed real fields, decision of equality in rings and fields, ...

$L_{tac}$: a dedicated functional language for building tactics

A mathematical-style proof language

Various third-party toolboxes of tactics (written in $L_{tac}$, in Coq itself, or using ML plugins, or using pipe communicating with Coq
Lemma even_double_n: (forall m, even (double m)).
proof.
  assume m:nat.
  per induction on m.
  suppose it is 0.
    thus thesis.
  suppose it is (S mm) and thesis for mm.
    then H:(even (S (S (mm+mm)))).
    have (S (S (mm + mm)) = S mm + S mm) using omega.
    hence (even (S mm +S mm)) by H.
  end induction.
end proof.
Qed.
Other toolboxes or tactics

Gonthier's ssreflect language for reasoning on decidable structures and more

U. Penn's LNgen for reasoning on languages with binders

Connections to SML solvers and Computer Algebra Systems (Z3, simplify, maple, ...)

Nowak's toolbox for cryptographic protocols at AIST

Termination proof toolboxes

The Why, Caduceus and Krakatoa specification languages for certifying ML, C and Java programs

Y-not toolbox for reasoning in Hoare logic

...
Libraries

**Bundled with Coq**
- Peano arithmetics in $\mathbb{N}$
- Efficient machine-word-based arithmetics in $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$
- Classical real analysis
- Programming libraries: lists, finite sets, finite maps
- Logical axioms

**Third-party libraries**
- Constructive real analysis
- Algebra
- Group theory
- Programming libraries: finger trees, union-find, ...
An example
Advertising the first Asian Coq Summer School

Dates: August 24-31, 2009
Location: Tsinghua, Beijing
Organization: joint Tsinghua-INRIA team FORMES (Pr. Jean-Pierre Jouannaud and Ming Gu)
Web site: http://coqschool-asia.org/wiki/Start
Main proof assistants

Dedicated to the formalization of mathematics: Mizar

Dedicated to proof certification: HOL, ACL2

Hybrid systems: Isabelle-HOL and Isabelle-ZF, HOL-light, PVS

Hybrid systems combining a functional language and a logic: Coq, Agda2

Other systems: Twelf, Matita, ...