Coq, an overview

Damien Pous (Plume team)

Fontenay aux roses, 24 Juin 2016
What is Coq?

- A “proof assistant”
- A “formal proof management system” (from Coq webpage)
- A programming language
- A specification language
- An interactive prover
- A project initiated by Thierry Coquand in 1984, and still under active development...
What is Coq useful for?

- Formally “certify” existing programs/libraries
- Build “certified” software
- Prove or certify mathematical theorems
What Coq is not?

- A fast/distributed/object-oriented programming language
- A Turing-complete programming language
- A model-checker
- A proof checker
- An automatic prover
- An oracle
- Something easy to work with
Mathematical proofs can be arbitrarily complex (and thus difficult to find), but checking them should remain simple.
Mathematical proofs can be arbitrarily complex (and thus difficult to find), but checking them should remain simple.

Once we know what is a proof...
Principles - Curry-Howard correspondance

“proofs are programs”

\[(A \rightarrow B) \land (B \rightarrow C) \land A \rightarrow C\]
Principles - Curry-Howard correspondance

“proofs are programs”

\[ p : (A \to B) \land (B \to C) \land A \to C \]
\[ (f, g, x) \mapsto g(f(x)) \]
“proofs are programs”

\[ p : (A \rightarrow B) \land (B \rightarrow C) \land A \rightarrow C \]

\[ (f, g, x) \mapsto g(f(x)) \]

property \( P \)
proof \( p \)
proof-checking \( p \vdash P \)

type \( T \)
(implementation)

term \( t \)

\( p \vdash t : T \)
(type-checking)
Outline

Quick tour of syntax and basic principles

Three kinds of use
   Prove/certify mathematical theorems
   Certify existing software
   Build certified software

Conclusions
Let’s play with Coq.
What did we learn?

- There is a single language (gallina), for:
  - programs/functions,
  - specifications,
  - proofs.

This is a purely functionnal programming language.
What did we learn?

▶ There is a single language (**gallina**), for:
  ▶ programs/functions,
  ▶ specifications,
  ▶ proofs.

This is a purely functionnal programming language.

▶ There is another language (**tactics: Ltac**):
  ▶ for building/searching proofs,
  ▶ that can be used interactively.

There are primitive tactics (**intros**, **apply**, **induction**), and rather complex ones (**tauto**, **ring**, **congruence**).
Principles - Gallina

- Checking a proof is easy: this is just type-checking... 
  ... but we need to trust the type-checker.
- Gallina is a small language,
  - for which type-checking is (easily) decidable;
  - and still remains really expressive.
- It relies on a strong theoretical background:
  - the “Calculus of Inductive Constructions”,
  - which comes from the $\lambda$-calculus.
Principles - Ltac

- Sequences of tactics do not constitute proofs: tactics produce Gallina terms that can be checked by Coq.
- We don’t need to trust tactics: any way to obtain a proof is valid since the proof will be checked.
- Proofs can actually be searched by other means than Ltac.
Outline

Quick tour of syntax and basic principles

Three kinds of use
- Prove/certify mathematical theorems
- Certify existing software
- Build certified software

Conclusions
We just proved some elementary theorems, more complex ones can be proved too!

Two major examples:
  - Georges Gonthier et al.’s proof of four-colours theorem;
  - Feit-Thompson’s theorem (finite groups classification)
Certify existing software

- Given an existing program, we might want to prove:
  - the absence of runtime errors,
  - termination,
  - behavioural correctness.

Problem: sometimes, programs are not written in Coq...

A solution: Why and Krakatoa/Caduceus tools.

(see Jean-Christophe Filliâtre' gallery of certified programs: http://why.lri.fr/examples/)
Certify existing software

Given an existing program, we might want to prove:
- the absence of runtime errors,
- termination,
- behavioural correctness.

Problem: Sometimes, programs are not written in Coq...
Certify existing software

- Given an existing program, we might want to prove:
  - the absence of runtime errors,
  - termination,
  - behavioural correctness.
- Problem: sometimes, programs are not written in Coq...

  
  (see Jean-Christophe Filliâtre’ gallery of certified programs: http://why.lri.fr/examples/)
Annotated C program

Caduceus

Why program

Why

Verification Conditions

Krakatoa

JML-Annotated Java program

Interactive provers (Coq, PVS, Isabelle/HOL, etc.)

Automatic provers (Simplify, Yices, Ergo, CVC3, etc.)
If we have to write a new program, why not writing it and certifying it within Coq?

Not so realistic, Coq is definitely too slow:
- it’s interpreted;
- integers, floats... are not ‘native’.

However, Coq programs can be extracted to other languages: OCaml, Haskell and Scheme.

This is how Xavier Leroy and Sandrine Blazy obtained their certified compiler for C:
http://compcert.inria.fr/
Outline

Quick tour of syntax and basic principles

Three kinds of use
- Prove/certify mathematical theorems
- Certify existing software
- Build certified software

Conclusions
Coq is a programming language:
  - purely functional;
  - interpreted (rather slow), but programs can be extracted to fast, compiled, languages;

Coq is an expressive specification language:
  - any mathematical property can be stated.

Coq certifies proofs by a simple type-checking algorithm.

Coq is a proof assistant:
  - the interactive mode allows us to prove a theorem progressively, by using tactics;
  - tactics can be more or less elaborated, and can be defined by the user.
Related software

- Why, Krakatoa/Caduceus,
  - for the analysis of Java and C programs.

- Isabelle/HOL
  - Larry Paulson - U. of Cambridge & Tobias Nipkow - U. of München

- PVS
  - Sam Owre, Natarajan Shankar, John Rushby

- Twelf
  - Karl Crary & Robert Harper - Carnegie Mellon U., USA
1984: Thierry Coquand and Gérard Huet implement the Calculus of Constructions (INRIA-Rocquencourt)

1991: Christine Paulin extended it to the Calculus of Inductive Constructions

2005: Georges Gonthier et al. use Coq to prove the 4-colours theorem

2008: Xavier Leroy et al. build a certified compiler for C

2012: Feit-Thompson: any finite group of odd order is solvable

Other developpers: Chet Murthy, Jean-Christophe Filliâtre, Bruno Barras, Hugo Herbelin, Assia Mahboubi and more than thirty people...

TypiCal (formerly LogiCal), ProVal and Marelle projects