Coq

LASER 2011 Summerschool
Elba Island, Italy

Christine Paulin-Mohring
`http://www.lri.fr/~paulin/LASER`

Université Paris Sud & INRIA Saclay - Île-de-France

September 2011

The proof assistant COQ

- An environment for developing mathematical facts:
  - defining objects
    (integers, sets, trees, functions, programs . . .)
  - make statements (predicates)
  - write proofs
- The compiler checks the correctness:
  - of definitions (well-formed sets, terminating functions . . .)
  - of proofs
- The environment helps with:
  - advanced notations
  - proof search
  - modular developments
  - program extraction

Examples done with Coq

- Mathematics
  - Fundamental theorem of Algebra (Barendregt et al)
  - Feit-Thompson theorem on finite groups (INRIA-Microsoft Research)
- Mixing maths and programs
  - Four color theorem (Gonthier-Werner)
  - Primality checker (Théry et al)
  - A Wave Equation Resolution Scheme (Boldo et al)
- Programming environments with proofs
  - JavaCard architecture (Gemalto-Trusted Logic, EAL7 certification)
  - Certified optimizing compiler for C (Leroy et al)
  - Formal Proofs for Computational Cryptography (Barthe et al)
  - Ynot library: imperative programs-separation logic (Morrisett and al)

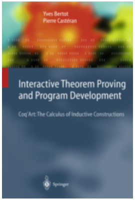
Related systems

- Coq is a proof assistant similar to HOL (Isabelle/HOL, HOL4,HOL-light), PVS, ...

- Coq is based on intuitionistic type theory:
  - Similar to Epigram, Matita, ... also Agda, NuPrl ...
  - Intentional behavior:
    functions are programs that can be computed (not binary relations).
  - Strong correspondance between proofs and programs.

Practical informations on COQ

- The Coq web site coq.inria.fr
  - Official distribution (multi-platform), Reference manual
  - Libraries and User's contributions
- Book : the Coq'art by Yves Bertot and Pierre Castéran



Interactive Theorem Proving and Program Development

Coq'Art: The Calculus of Inductive Constructions

Series: Texts in Theoretical Computer Science.

http://www.labri.fr/perso/casteran/CoqArt

- See also:
  - *Software foundations* by B. Pierce and al.
    http://www.cis.upenn.edu/~bcpierce/sf/
  - *Certified Programming with Dependent Types* by A. Chlipala.
    http://adam.chlipala.net/cpdt/

Two levels architecture

COQ environment		COQ kernel
▶ notations ▶ extended language ▶ libraries ▶ tactics ▶ user extensible	compiled to	▶ limited language ▶ few rules ▶ expressive

`5=2+3` becomes (using `Z_scope`)

```
@eq Z
   (Zpos (xI (xO xH)))
   (Zplus (Zpos (xO xH)) (Zpos (xI xH)))
```

Using Coq for program verification

- Express "*program* p *is correct*" as a mathematical statement in Coq and prove it!
  Can be hard but proof is safe.
- Program your favorite program analyser (model-checking, abstract interpretation,...) in Coq, prove it correct and use it !
  A big investment, but automatic result for each program instance.
- Represent program p as a Coq term t and the specification as a type T such that $t : T$ implies p is correct.
  Works well for functional (possibly monadic) programs.
- Use an external tool to generate proof obligations and then Coq to solve obligations
  Less safe approach but can deal with undecidable fragments

Using COQ for program verification

- Express "*program* p *is correct*" as a mathematical statement in COQ and prove it!
  Can be hard but proof is safe.
- Program your favorite program analyser (model-checking, abstract interpretation,...) in COQ, prove it correct and use it !
  A big investment, but automatic result for each program instance.
- Represent program p as a COQ term t and the specification as a type T such that $t : T$ implies p is correct.
  Works well for functional (possibly monadic) programs.
- Use an external tool to generate proof obligations and then COQ to solve obligations
  Less safe approach but can deal with undecidable fragments

Using Coq for program verification

- Express "*program p is correct*" as a mathematical statement in Coq and prove it!
  Can be hard but proof is safe.
- Program your favorite program analyser (model-checking, abstract interpretation,...) in Coq, prove it correct and use it !
  A big investment, but automatic result for each program instance.
- Represent program p as a Coq term t and the specification as a type T such that $t : T$ implies p is correct.
  Works well for functional (possibly monadic) programs.
- Use an external tool to generate proof obligations and then Coq to solve obligations
  Less safe approach but can deal with undecidable fragments

Using Coq for program verification

- Express "*program p is correct*" as a mathematical statement in Coq and prove it!
  Can be hard but proof is safe.
- Program your favorite program analyser (model-checking, abstract interpretation,...) in Coq, prove it correct and use it !
  A big investment, but automatic result for each program instance.
- Represent program p as a Coq term t and the specification as a type T such that $t : T$ implies p is correct.
  Works well for functional (possibly monadic) programs.
- Use an external tool to generate proof obligations and then Coq to solve obligations
  Less safe approach but can deal with undecidable fragments

Coq: outline of the lectures

- Introduction
- Basics of Coq system
- Using simple inductive definitions
- Functional programming with Coq
- Automating proofs

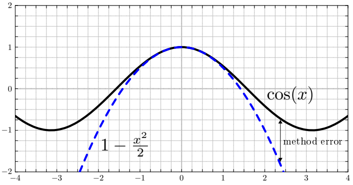
Outline

- Introduction
  - What is COQ ?
  - Example
- Basics of COQ system
- Using simple inductive definitions
- Functional programming with COQ
- Automating proofs

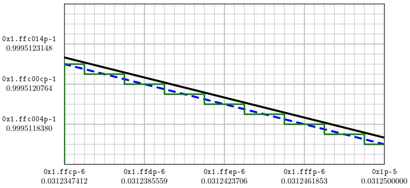
Example of C program verification

Approximate cosinus function near 0 using floating point numbers.

```
float my_cosine(float x) {
  return 1.0f - x * x * 0.5f;
}
```



Method error



Floating point error near $\frac{1}{32}$

Using Coq for C program verification

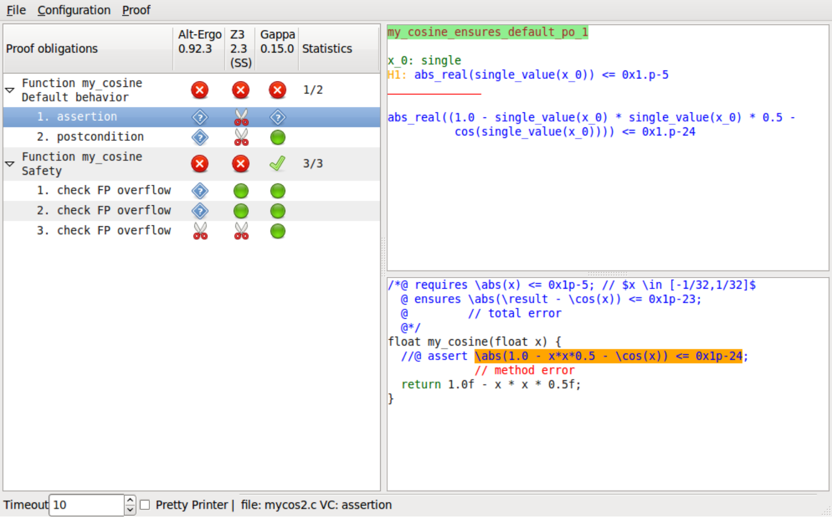
Code with specification (using real numbers):

```
/*@ requires \abs(x) <= 0x1p-5;
  @ ensures \abs(\result - \cos(x)) <= 0x1p-23;
  @*/
float my_cosine(float x) {
  //@ assert \abs(1.0 - x*x*0.5 - \cos(x)) <= 0x1p-24;
  return 1.0f - x * x * 0.5f;
}
```

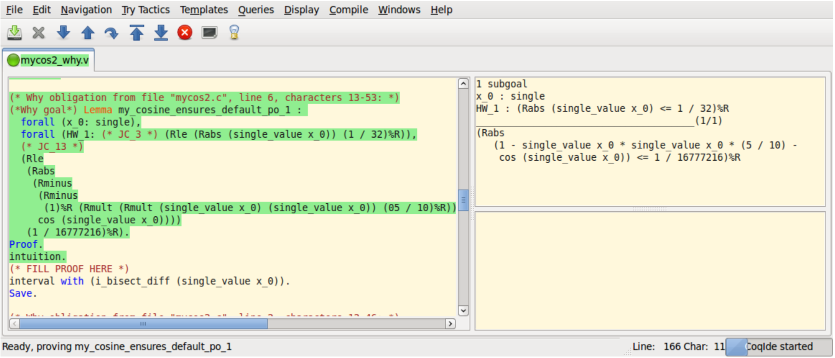
Frama-C/Why/Coq

Generating conditions

```
frama-c -jessie mycos2.c
```



Generating Coq goals



Generating COQ goals

Certified version of automated tools