

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

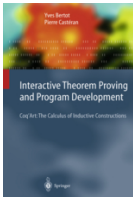
- ▶ JavaCard architecture (Gemalto-Trusted Logic, EAL7 certification)
- ▶ Certified optimizing compiler for C (Leroy et al)
- ▶ Formal Proofs for Computational Cryptography (Barthe et al)
- ▶ Ynot library: imperative programs-separation logic (Morrisett and al)

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

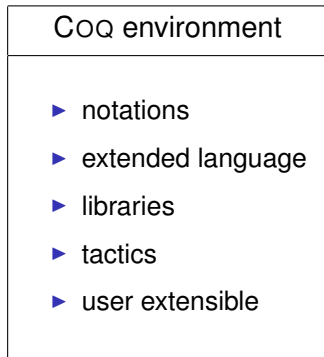


Interactive Theorem Proving and Program Development
Coq'Art: The Calculus of Inductive Constructions
Series: Texts in Theoretical Computer Science.

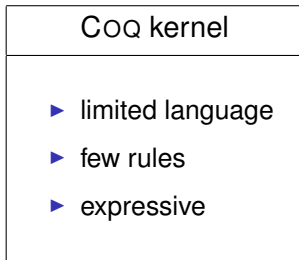
<http://www.labri.fr/perso/casteran/CoqArt>

- ▶ 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



compiled to



$5=2+3$ 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

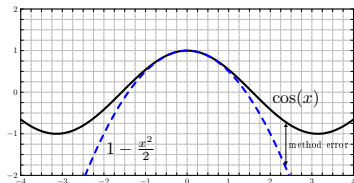
Outline

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

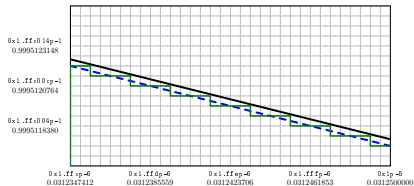
Example of C program verification

Approximate cosine 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

Proof obligations	Alt-Ergo 0.92.3	Z3 2.3 (SS)	Gappa 0.15.0	Statistics
Function my_cosine Default behavior	✗	✗	✗	1/2
1. assertion	?	✗	?	
2. postcondition	?	✗	●	
Function my_cosine Safety	✗	✗	✓	3/3
1. check FP overflow	◆	●	●	
2. check FP overflow	◆	●	●	
3. check FP overflow	✗	✗	●	

```
my_cosine ensures default po 1
x_0: single
HI: abs_real(single_value(x_0)) <= 0x1.p-5

abs_real((1.0 - single_value(x_0) * single_value(x_0) * 0.5 -
cos(single_value(x_0)))) <= 0x1.p-24

/*@ requires \abs(x) <= 0x1p-5; // $x \in [-1/32,1/32]$
@ ensures \abs(\result - \cos(x)) <= 0x1p-23;
@ // total error
@*/
float my_cosine(float x) {
  //@ assert \abs(1.0 - x*x*0.5 - \cos(x)) <= 0x1p-24;
  // method error
  return 1.0f - x * x * 0.5f;
}
```

Timeout 10 Pretty Printer | file: mycos2.c VC: assertion

Generating COQ goals

The screenshot shows the Coq IDE interface with a file named `mycos2_why.v` open. The editor contains the following Coq code:

```
(* Why obligation from file "mycos2.c", line 6, characters 13-53: *)
(*Why goal*) Lemma my_cosine_ensures_default_po_1 :
  forall (x_0 : single),
  forall (HW_1 : (* JC 3 *) (Rle (Rabs (single_value x_0)) (1 / 32)%R)),
  (* JC 13 *)
  (Rle
   (Rabs
    (Rminus
     (Rminus
      (1)%R (Rmult (Rmult (single_value x_0) (single_value x_0)) (05 / 10)%R))
      cos (single_value x_0))))
   (1 / 16777216)%R).
Proof.
intuition.
(* FILL PROOF HERE *)
interval with (i_bisect_diff (single_value x_0)).
Save.
```

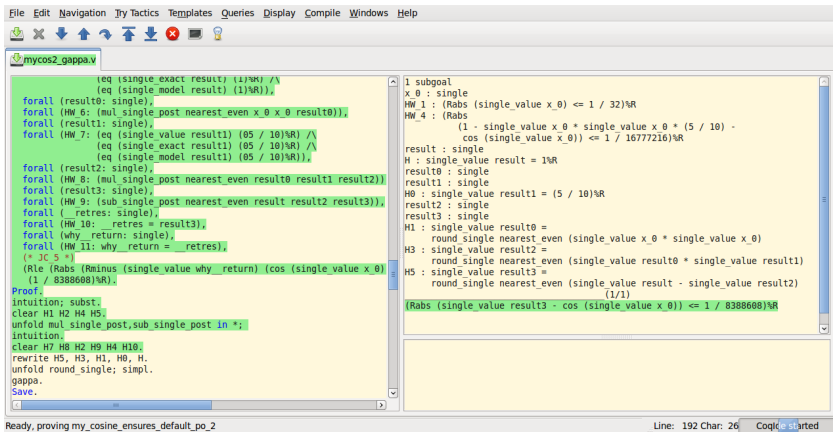
The right-hand pane displays the generated subgoal:

```
1 subgoal
x_0 : single
HW_1 : (Rabs (single_value x_0) <= 1 / 32)%R
(1/1)
(Rabs
 (1 - single_value x_0 * single_value x_0 * (5 / 10) -
  cos (single_value x_0)) <= 1 / 16777216)%R
```

The status bar at the bottom indicates: `Ready, proving my_cosine_ensures_default_po_1` and `Line: 166 Char: 11 CoqIDE started`.

Generating COQ goals

Certified version of automated tools



The screenshot shows a Coq IDE window titled "mycos2_gappa.v". The left pane contains a Coq proof script, and the right pane shows the current goal state.

```
File Edit Navigation Try Tactics Templates Queries Display Compile Windows Help
mycos2_gappa.v
(eq (single_exact result) (1)%R) /\
(eq (single_model result) (1)%R)),
forall (result0 : single),
forall (HW_6: (mul single_post nearest_even x_0 x_0 result0)),
forall (result1 : single),
forall (HW_7: (eq (single_value result1) (05 / 10)%R) /\
(eq (single_exact result1) (05 / 10)%R) /\
(eq (single_model result1) (05 / 10)%R))),
forall (result2 : single),
forall (HW_8: (mul single_post nearest_even result0 result1 result2)),
forall (result3 : single),
forall (HW_9: (sub single_post nearest_even result result2 result3)),
forall (retres : single),
forall (HW_10: retres = result3),
forall (why_return : single),
forall (HW_11: why_return = retres),
(* JC S *)
(Rle (Rabs (Rminus (single_value why_return) (cos (single_value x_0)
(1 / 8388608)%R)).
Proof.
intuition; subst.
clear H1 H2 H4 H5.
unfold mul_single_post, sub_single_post in *.
intuition.
clear H7 H8 H2 H9 H4 H10.
rewrite H5, H3, H1, H0, H.
unfold round_single; simpl.
gappa.
Save.
```

The right pane shows the goal state:

```
1 subgoal
x_0 : single
HW_1 : (Rabs (single_value x_0) <= 1 / 32)%R
HW_4 : (Rabs
(1 - single_value x_0 * single_value x_0 * (5 / 10) -
cos (single_value x_0)) <= 1 / 1677216)%R
result : single
H : single_value result = 1%R
result0 : single
result1 : single
H0 : single_value result1 = (5 / 10)%R
result2 : single
result3 : single
H1 : single_value result0 =
round_single nearest_even (single_value x_0 * single_value x_0)
H3 : single_value result2 =
round_single nearest_even (single_value result0 * single_value result1)
H5 : single_value result3 =
round_single nearest_even (single_value result - single_value result2)
(1/1)
(Rabs (single_value result3 - cos (single_value x_0)) <= 1 / 8388608)%R
```

Ready, proving my_cosine_ensures_default_po_2 Line: 192 Char: 26 CoqIDE started