Coq

Laurent Théry, Pierre Letouzey, and Georges Gonthier

Formalizations by Laurent Théry <laurent.thery@sophia.inria.fr>, Pierre Letouzey <Pierre.Letouzey@lri.fr> and Georges Gonthier <gonthier@microsoft.com>. Answers by Laurent Théry.

4.1 Statement

```
irrational (sqrt 2%nat).
```

4.2 Definitions

Definition of `irrational`

```
Definition irrational (x : R) : Prop :=
   forall (p : Z) (q : nat), q <> 0 ->  x <> (p / q)%R.
```

4.3 Proof

```
Require Import ArithRing.
Require Import Wf_nat.
Require Import Peano_dec.
Require Import Div2.
Require Import Even.
```

Properties of `div2` and `double` (these theorems should be in `Div2.v`)

```
Theorem double_div2: forall (n : nat),  div2 (double n) = n.
simple induction n; auto with arith.
intros n0 H.
rewrite double_S; pattern n0 at 2; rewrite <- H; simpl; auto.
Qed.

Theorem double_inv: forall (n m : nat), double n = double m ->  n = m.
intros n m H; rewrite <- (double_div2 n); rewrite <- (double_div2 m); rewrite H;
 auto.
Qed.

Theorem double_mult_l: forall (n m : nat),  double (n * m) = double n * m.
unfold double; auto with arith.
Qed.

Theorem double_mult_r: forall (n m : nat),  double (n * m) = n * double m.
unfold double; intros; ring.
Qed.
```

If the power to the 2 of a number is even, then this number is even

```
Theorem even_is_even_times_even: forall (n : nat), even (n * n) ->  even n.
intros n H; (try case (even_or_odd n)); auto.
intros; apply even_mult_inv_r with (1 := H); auto.
Qed.
```

F. Wiedijk (Ed.): The Seventeen Provers of the World, LNAI 3600, pp. 28–35, 2006.

Useful fact $4 \cdot (n/2) \cdot (n/2) = n \cdot n$ if n is even

```
Theorem main_thm_aux:
 forall (n : nat), even n ->  double (double (div2 n * div2 n)) = n * n.
intros; rewrite double_mult_l; rewrite double_mult_r;
 (repeat rewrite <- even_double); auto.
Qed.
```

Main theorem
We do the proof of the theorem by well founded induction: Suppose that we have $n \cdot n = 2 \cdot p \cdot p$

if $n = 0$ then $p = 0$
if $n \neq 0$ then
- n is even ($n = 2n'$) and p is even ($p = 2p'$)
- we have $n' \cdot n' = 2 \cdot p' \cdot p'$ and $n' < n$
- by the induction hypothesis we have $p' = 0$
- so $p = 0$

```
Theorem main_thm: forall (n p : nat), n * n = double (p * p) ->  p = 0.
intros n; pattern n; apply lt_wf_ind; clear n.
intros n H p H0.
case (eq_nat_dec n 0); intros H1.
generalize H0; rewrite H1; case p; auto; intros; discriminate.
assert (H2: even n).
apply even_is_even_times_even.
apply double_even; rewrite H0; rewrite double_div2; auto.
assert (H3: even p).
apply even_is_even_times_even.
rewrite <- (double_inv (double (div2 n * div2 n)) (p * p)).
apply double_even; rewrite double_div2; auto.
rewrite main_thm_aux; auto.
assert (H4: div2 p = 0).
apply (H (div2 n)).
apply lt_div2; apply neq_0_lt; auto.
apply double_inv; apply double_inv; (repeat rewrite main_thm_aux); auto.
rewrite (even_double p); auto; rewrite H4; auto.
Qed.
```

Coercions from `nat` and `Z` to `R`

```
Require Import Reals.
Require Import Field.
Coercion INR : nat >-> R.
Coercion IZR : Z >-> R.
```

Definition of irrational

```
Definition irrational (x : R) : Prop :=
   forall (p : Z) (q : nat), q <> 0 ->  x <> (p / q)%R.
```

Final theorem

```
Theorem irrational_sqrt_2: irrational (sqrt 2%nat).
intros p q H H0; case H.
apply (main_thm (Zabs_nat p)).
replace (Div2.double (q * q)) with (2 * (q * q));
 [idtac | unfold Div2.double; ring].
case (eq_nat_dec (Zabs_nat p * Zabs_nat p) (2 * (q * q))); auto; intros H1.
case (not_nm_INR _ _ H1); (repeat rewrite mult_INR).
rewrite <- (sqrt_def (INR 2)); auto with real.
rewrite H0; auto with real.
assert (q <> 0%R :> R); auto with real.
field; auto with real; case p; simpl; intros; ring.
Qed.
```

Proof term of `main_thm`

```
main_thm =
fun n : nat =>
lt_wf_ind n
  (fun n0 : nat => forall p : nat, n0 * n0 = Div2.double (p * p) -> p = 0)
  (fun (n0 : nat)
     (H : forall m : nat,
          m < n0 -> forall p : nat, m * m = Div2.double (p * p) -> p = 0)
     (p : nat) (H0 : n0 * n0 = Div2.double (p * p)) =>
   match Peano_dec.eq_nat_dec n0 0 with
   | left H1 =>
       let H2 :=
         eq_ind_r (fun n : nat => n * n = Div2.double (p * p) -> p = 0)
           match p as n return (0 * 0 = Div2.double (n * n) -> n = 0) with
           | O => fun H2 : 0 * 0 = Div2.double (0 * 0) => H2
           | S n0 =>
               fun H2 : 0 * 0 = Div2.double (S n0 * S n0) =>
               let H3 :=
                 eq_ind (0 * 0)
                   (fun ee : nat =>
                    match ee with
                    | O => True
                    | S _ => False
                    end) I (Div2.double (S n0 * S n0)) H2 in
               False_ind (S n0 = 0) H3
           end H1 in
       H2 H0
   | right H1 =>
       let H2 :=
         even_is_even_times_even n0
           (double_even (n0 * n0)
              (eq_ind_r (fun n : nat => n = Div2.double (div2 n))
                 (eq_ind_r
                    (fun n : nat => Div2.double (p * p) = Div2.double n)
                    (refl_equal (Div2.double (p * p)))
                    (double_div2 (p * p))) H0)) in
       let H3 :=
         even_is_even_times_even p
           (eq_ind (Div2.double (div2 n0 * div2 n0))
              (fun n : nat => even n)
```

```
            (double_even (Div2.double (div2 n0 * div2 n0))
               (eq_ind_r
                  (fun n : nat =>
                   Div2.double (div2 n0 * div2 n0) = Div2.double n)
                  (refl_equal (Div2.double (div2 n0 * div2 n0)))
                  (double_div2 (div2 n0 * div2 n0))))
            (p * p)
            (double_inv (Div2.double (div2 n0 * div2 n0))
               (p * p)
               (eq_ind_r (fun n : nat => n = Div2.double (p * p)) H0
                  (main_thm_aux n0 H2)))) in
      let H4 :=
        H (div2 n0) (lt_div2 n0 (neq_0_lt n0 (sym_not_eq H1)))
          (div2 p)
          (double_inv (div2 n0 * div2 n0) (Div2.double (div2 p * div2 p))
             (double_inv (Div2.double (div2 n0 * div2 n0))
                (Div2.double (Div2.double (div2 p * div2 p)))
                (eq_ind_r
                   (fun n : nat =>
                    n =
                    Div2.double
                      (Div2.double (Div2.double (div2 p * div2 p))))
                   (eq_ind_r (fun n : nat => n0 * n0 = Div2.double n) H0
                      (main_thm_aux p H3)) (main_thm_aux n0 H2)))) in
      eq_ind_r (fun p0 : nat => p0 = 0)
        (eq_ind_r (fun n1 : nat => Div2.double n1 = 0) (refl_equal 0) H4)
        (even_double p H3)
  end)
    : forall n p : nat, n * n = Div2.double (p * p) -> p = 0
```

4.4 Another Formalization: Using the Binary Representation of the Integers

```
Require Import BinPos.
Open Scope positive_scope.

Ltac mysimpl := simplify_eq; repeat rewrite Pmult_xO_permute_r.

Theorem main_thm: forall p q: positive, 2*(q*q)<>p*p.
Proof.
induction p; simpl; intro; mysimpl.
destruct q; mysimpl; firstorder.
Qed.

Require Import Reals Field.
Open Scope R_scope.

(* IPR: Injection from Positive to Reals *)
(* Should be in the standard library, close to INR and IZR *)

Definition IPR (p:positive):= (INR (nat_of_P p)).
Coercion IPR : positive >-> R.

Lemma mult_IPR : forall p q, IPR (p * q) = (IPR p * IPR q)%R.
unfold IPR; intros; rewrite nat_of_P_mult_morphism; auto with real.
Qed.
Lemma IPR_eq : forall p q, IPR p = IPR q -> p = q.
unfold IPR; intros; apply nat_of_P_inj; auto with real.
Qed.
```

```
Lemma IPR_nonzero : forall p, IPR p <> 0.
unfold IPR; auto with real.
Qed.
Hint Resolve IPR_eq IPR_nonzero.
(* End of IPR *)

Ltac myfield := field; rewrite <- mult_IPR; auto.

Lemma main_thm_pos_rat : forall (p q:positive), 2 <> (p/q)*(p/q).
Proof.
red; intros.
assert (2*(q*q)=p*p).
  rewrite H; myfield.
clear H; change 2 with (IPR 2) in H0.
apply main_thm with p q; auto.
repeat rewrite <- mult_IPR in H0; auto.
Qed.

Coercion IZR : Z >-> R.

Lemma main_thm_rat : forall (p:Z)(q:positive), 2 <> (p/q)*(p/q).
Proof.
destruct p; simpl; intros.
replace (0 / q * (0 / q)) with 0.
discrR.
field; rewrite <- mult_IPR; auto.
exact (main_thm_pos_rat p q).
replace (INR (nat_of_P p)) with (IPR p); auto.
replace (- p / q * (- p / q)) with ((p/q)*(p/q)); try myfield.
exact (main_thm_pos_rat p q).
Qed.

Definition irrational (x:R) : Prop := forall (p:Z)(q:positive), x <> (p/q).

Lemma Sqrt2_irr : irrational (sqrt 2).
Proof.
 red; intros p q H.
 assert (H1: 2 = sqrt 2 * sqrt 2).
   symmetry; apply sqrt_sqrt; auto with real.
 rewrite H in H1; apply main_thm_rat with p q; auto.
Qed.
```

4.5 Another Formalization: Coq in the Style of Georges Gonthier

```
Section Sqrt2.

Variable R : real_model.
Coercion Local fracR := (fracr R).

Theorem sqrt2_irrational : ~(EX f : frac | `f = sqrt 2`).
Proof.
Move=> [f Df]; Step [Hf22 H2f2]: `(mulf f f) = F2`.
  Apply: (eqr_trans (fracr_mul ? ? ?)); Apply: eqr_trans (fracrz R (Znat 2)).
  By Apply: eqr_trans (square_sqrt (ltrW (ltr02 R))); Apply mulr_morphism.
Step Df2: (eqf F2 (mulf f f)) By Apply/andP; Split; Apply/(fracr_leqPx R ? ?).
Move: f Df2 {Hf22 H2f2 Df} => [d m]; Rewrite: /eqf /= -eqz_leq; Move/eqP.
Rewrite: scalez_mul -scalez_scale scalez_mul mulzC {-1 Zpos}lock /= -lock.
Step []: (Zpos (S d)) = (scalez d (Znat 1)).
  By Apply esym; Apply: eqP; Rewrite scalez_pos; Elim d.
Step [n []]: (EX n | (mulz (Zpos n) (Zpos n)) = (mulz m m)).
```

```
  Case: m => [n | n]; LeftBy Exists n.
  By Exists (S n); Rewrite: -{1 (Zneg n)}oppz_opp mulz_oppl -mulz_oppr.
Pose i := (addn (S d) n); Move: (leqnn i) {m}; Rewrite: {1}/i.
Elim: i n d => // [i Hrec] n d Hi Dn2; Move/esym: Dn2 Hi.
Rewrite: -{n}odd_double_half double_addnn !zpos_addn; Move/half: n (odd n) => n.
Case; [Move/((congr oddz) ? ?) | Move/((congr halfz) ? ?)].
  By Rewrite: !mulz_addr oddz_add mulzC !mulz_addr oddz_add !oddz_double.
Rewrite: addOn addnC -addnA addOz mulz_addr !halfz_double mulzC mulz_addr.
Case: n => [|n] Dn2 Hi; LeftBy Rewrite: !mulz_nat in Dn2.
Apply: Hrec Dn2; Apply: (leq_trans 3!i) Hi; Apply: leq_addl.
Qed.

End Sqrt2.
```

4.6 System

What is the home page of the system?

<http://pauillac.inria.fr/coq/>

What are the books about the system? The Coq'Art book

> Yves Bertot and Pierre Castéran, *Interactive Theorem Proving and Program Development, Coq'Art: The Calculus of Inductive Constructions*, Texts in Theoretical Computer Science. An EATCS Series, 2004, 469 pp., ISBN 3-540-20854-2.

provides a pragmatic introduction to the development of proofs and certified programs using Coq. Its web page is:

<http://www.labri.fr/Perso/~casteran/CoqArt/>

Otherwise a reference manual and a tutorial are available at:

<http://pauillac.inria.fr/coq/doc/main.html>
<http://pauillac.inria.fr/coq/doc/tutorial.html>

What is the logic of the system? Coq is based on the *Calculus of Inductive Construction*, a lambda calculus with a rich type system with dependent types.

What is the implementation architecture of the system? The system is written in ocaml (a dialect of ML). Following Curry-Howard's isomorphism, the kernel of the system is a type-checking algorithm that checks the correctness of proofs.

What does working with the system look like? The system has a specification language called Gallina. It allows the user to write its own specification by developing theories. Theories are built from axioms, hypotheses, parameters, lemmas, theorems and definitions of constants, functions, predicates and sets. Proofs are constructed interactively using the usual LCF tactics approach.

User may interact with the system using the standard shell window but there are also three available graphical user-interfaces:

- CoqIde an integrated gtk-based user interface
- Proof General <http://zermelo.dcs.ed.ac.uk/~proofgen/> an Emacs-based interface
- Pcoq <http://www-sop.inria.fr/lemme/pcoq/> a java-based interface

What is special about the system compared to other systems? First of all, the logic of Coq is very expressive allowing to define rich mathematical objects. Second, Coq manipulates explicit proof objects. A consequence is that the integrity of the system only relies on the correct implementation of a typechecking algorithm. Finally, a program extractor synthesizes computer programs obeying their formal specifications written as logical assertions in the language.

What are other versions of the system? There is only one supported implementation of the system. The current version is 8.0.

Who are the people behind the system? The main developers of the system are from the Logical group at INRIA France (<http://logical.inria.fr/>).

What are the main user communities of the system? The main user communities are in France (INRIA, LIX, ENS, LRI) and in Holland (Nijmegen).

What large mathematical formalizations have been done in the system? The user contributions are listed at the following address:

http://pauillac.inria.fr/coq/contribs-eng.html.

Here are some relevant ones:

- A proof of the four colour theorem by Georges Gonthier, in collaboration with Benjamin Werner
- Constructive Category Theory by Amokrane Saïbi
- Rational Numbers represented as Stern-Brocot Trees by Milad Niqui
- Elements of Constructive Geometry, Group Theory, and Domain Theory by Gilles Kahn
- High School Geometry and Oriented Angles of Vectors in the Plane by Frédérique Guilhot
- Basics notions of algebra by Loïc Pottier
- Fundamental Theorem of Algebra by Herman Geuvers, Freek Wiedijk, Jan Zwanenburg, Randy Pollack, Henk Barendregt
- Proof of Buchberger's algorithm by Laurent Théry, Henrik Persson
- Rem Theorem in Baire space by Henk Barendregt
- Real analysis by Micaela Mayero (standard library)
- A Proof of the Three Gap Theorem (Steinhaus Conjecture) by Micaela Mayero

What representation of the formalization has been put in this paper? What is presented is the exact script one has to feed Coq with so it accepts the final theorem.

After the script a proof term of one of the lemmas is shown.

What needs to be explained about this specific proof? In this proof we have decided to use as much as possible the notions that were already present in the system. The predicates *even* and *odd* are mutually defined in the theory `Even`. The function *div2* and *double* are defined in `Div2`. The key point of the main proof (`main_thm`) is the application of the well founded induction *lt_wf_ind* (second line of the script) whose statement is:

$$\forall p, P.\,(\forall n.\,(\forall m.\,m < n \to (P\ m)) \to (Pn)) \to (P\ p)$$

The reals are defined in the standard library `Reals`.

What needs to be explained about the second proof? The second formalization has been inspired by the Minlog entry (page 151). It takes advantage of the *positive* datatype of Coq, that encodes strictly positive numbers in a binary way. This allows to easily check whether a number is even or not, and also to stick to normal induction instead of well-founded induction.

What needs to be explained about the third proof? This formalization uses a few basic libraries of the Colour Theorem proof, including a construction of the classic reals, which has been extended with a definition of the square root (not shown). The type `frac` is the representation of the rational numbers used in the construction of the real numbers.

The proof script uses the extended Coq v7 tactics developed for the Four Colour Theorem proof. It is self-contained: the first four lines reduce the problem from $\mathbb{R}$ to $\mathbb{Q}$, the next two from $\mathbb{Q}$ $\mathbb{Z}$, the next five from $\mathbb{Z}$ to $\mathbb{N}$, the next two lines set up an induction on the size of the fraction, which is completed in the last six lines.