Computer Verified Mathematics using Coq

MEDDAYS 2018, February 14

Cyril Cohen
for the MARELLE team
Coq the bug killer

• **What is Coq?**
  - A Purely Functional Programming Language (like OCaml),
  - An Interactive Proof Assistant,

• Why do we use Coq?
  - To develop software without errors (e.g. CompCert)
  - To write mathematical proofs without errors (e.g. Four Colors Theorem, Odd Order Theorem)

• How does one use Coq?
  Describe four components:
  - data (e.g. numbers, lists, polynomials),
  - operations (e.g. addition, matrix product),
  - properties (e.g. associativity, Cayley-Hamilton theorem),
  - and their proofs

• Coq is widely adopted (U. Paris, Inria, CNRS, Princeton, Yale, MIT, Cornell, U. Penn., U. Wash, Galois, MPI,)

• ACM Awards: Programming language, Software System 2013

Cyril Cohen – Computer Verified Mathematics using Coq – MEDDAYS 2018, February 14
Coq the bug killer

• **What is Coq?**
  - A Purely Functional Programming Language (like OCaml),
  - An Interactive Proof Assistant,

• **Why do we use Coq?**
  - To develop software **without errors** (e.g. CompCert)
  - To write mathematical proofs **without errors**
    (e.g. Four Colors Theorem, Odd Order Theorem)
Coq the bug killer

- **What is Coq?**
  - A Purely Functional Programming Language (like OCaml),
  - An Interactive Proof Assistant,

- **Why do we use Coq?**
  - To develop software **without errors** (e.g. CompCert)
  - To write mathematical proofs **without errors** (e.g. Four Colors Theorem, Odd Order Theorem)

- **How does one use Coq?** Describe four components:
  - data (e.g. numbers, lists, polynomials),
  - operations (e.g. addition, matrix product),
  - properties (e.g. associativity, Cayley-Hamilton theorem),
  - and their proofs
Coq the bug killer

• **What is Coq?**
  - A Purely Functional Programming Language (like OCaml),
  - An Interactive Proof Assistant,

• **Why do we use Coq?**
  - To develop software **without errors** (e.g. CompCert)
  - To write mathematical proofs **without errors** (e.g. Four Colors Theorem, Odd Order Theorem)

• **How does one use Coq?** Describe four components:
  - data (e.g. numbers, lists, polynomials),
  - operations (e.g. addition, matrix product),
  - properties (e.g. associativity, Cayley-Hamilton theorem),
  - and their proofs

• **Coq is widely adopted** (U. Paris, Inria, CNRS, Princeton, Yale, MIT, Cornell, U. Penn., U. Wash. Galois, MPI,)

• **ACM Awards**: Programming language, Software System 2013
The formal verification landscape

Three main categories of software formal verification:

- Static program analysis
  **Automated** verification of low level features
  \(\text{(e.g. Lint, Astree, \ldots)}\)

- Program logics for conventional Programming Languages
  **Manual** annotation, **(Semi-)Automatic** verification
  \(\text{(e.g. Why3, KeY, \ldots)}\)

- Dedicated language for algorithms and proofs
  **Manual** specification, **Interactive** verification
  \(\text{(e.g. Coq, HOL, \ldots)}\)
The formal verification landscape

Three main categories of software formal verification:

- Static program analysis
  Automated verification of low level features
  (e.g. Lint, Astree, ...)

- Program logics for conventional Programming Languages
  Manual annotation, (Semi-)Automatic verification
  (e.g. Why3, KeY, ...)

- Dedicated language for algorithms and proofs
  Manual specification, Interactive verification
  (e.g. Coq, HOL, ...)

Cyril Cohen – Computer Verified Mathematics using Coq – MEDDAYS 2018, February 14
History of Higher-Order Logic theorem provers

De Bruijn Authomath (1967) and Milner’s LCF system (1972).

HOL family
- HOL
- HOL88
- ..., HOL4
- Isabelle/HOL
- HOL-Light
- ...

Dependent Type Theory family
- Coq
- Agda
- Lean
- ...

Able to model and reason about arbitrary algorithms

- **Undecidable**: human intervention required
- **Expressive**: Operating systems, Smartcard, Cryptography, Information theory, Compilers, Computer arithmetic, Linear Algebra, Group Theory, ...
Mathematical Components

A library for Mathematics in Coq

Origins:

• Created for the Four Colour Theorem [Gonthier 2007] and
• Extended for the Odd Order Theorem [Gonthier et al. 2013]

Contents:

• Containers, Basic datatypes, Elementary arithmetic, ... 
• Finite graph theory, depth-first search, ... 
• Finite group theory, Representation theory, Character theory, ... 
• Algebraic structures, Linear Algebra, Galois Theory, ... 
• Advanced group theory, ... 
• Quantifier Elimination for Real Closed Fields, ... 
• ...
A simple example

(* Data *)
Inductive list (A : Type) : Type :=
  nil : list A | cons : A -> list A -> list A

(* Operation *)
Fixpoint cat s1 s2 :=
  if s1 is x :: s1' then x :: s1' ++ s2 else s2
where "s1 ++ s2" := (cat s1 s2) : seq_scope.

(* Properties *)
Lemma cat0s s : [::] ++ s = s.
(* Proof *)
Proof. by []. Qed.
A simple example

(* Data *)
Inductive list (A : Type) : Type :=
    nil : list A | cons : A -> list A -> list A

(* Operation *)
Fixpoint cat s1 s2 :=
    if s1 is x :: s1' then x :: s1' ++ s2 else s2
where "s1 ++ s2" := (cat s1 s2) : seq_scope.

(* Properties *)
Lemma cat0s s : [::] ++ s = s.
(* Proof *)
Proof. by []. Qed.

Lemma catA s1 s2 s3 : s1 ++ s2 ++ s3 = (s1 ++ s2) ++ s3.
Proof. by elim: s1 => //≡ x s1 ->. Qed.
A more complicated example

Theorem normal_spectral_subproof \(\{n\}\) \(\{A: 'M[C]_n\}\):
reflect
(exists2 sp: 'M_n * 'rV_n,
    sp.1 \is unitary &
    A = invmx sp.1 *m diag_mx sp.2 *m sp.1)
(A \is normalmx).
Proof.
apply: (iffP normalmxP); last first.
  move=> [[/= P D] P_unitary --]].
  rewrite !trmx_mul !map_mxM !mulmxA inv_unitary //.
  rewrite !trmxCK ![\*m P] *m [\_mulmxtVK //.
  by rewrite --[X in X *m P]mulmxA tr_diag_mx map_diag_mx diag_mxC mulmxA.
move=> /cotrigonalization2 [P Punitary /andP]].
set D := _ *m A *m _ => Dtriangular Dtc_triangular.
exists (P, \row_i D i i) => //.
have Punit : P \in unitmx by rewrite unitary_unit.
apply: (@row_full_inj _ _ _ P); rewrite ?row_full_unit //.
apply: (@row_free_inj _ _ _ (invmx P)); rewrite ?row_free_unit ?unitmx_inv //.
rewrite !mulmxA mulmxV // mulmx mulmxK //.
apply/matrixP=> i j; rewrite [D]lock ![in RHS]mxE --lock --val_eqE.
have [lt_ij|lt_ji|val_inj<--] := ltngtP; rewrite mulr0n.
  by rewrite (triangularP _).
suff : D^t* j i = 0 by rewrite !mxE => /eqP; rewrite conjC_eq0 => /eqP.
rewrite !trmx_mul !map_mxM inv_unitary // trmxCK --(@inv_unitary _ P) //.
by rewrite mulmxA (triangularP _).
Qed.
Combinatorics and numbers

- Containers, basic datatypes:
  Lists, tuples, finite types, functions, sets, graphs...

- Numbers:
  Naturals, integers, modular arithmetics, rationals

- Elementary arithmetics:
  Divisibility, means, primes, binomials...

- Indexed iterated operations:
  \( \Sigma, \Pi, \bigoplus, \ldots \)
Finite group theory

• Elementary concepts:
  Order, morphisms, permutations, quotient, characteristic subgroups, series, products, commutators, presentations,…

• Elementary theory:
  Lagrange, isomorphisms, Sylow, Hall, Jordan-Hölder theorems, structure of abelian groups, theory of various characteristic subgroups,…

• Representation theory of finite groups
  Schur, Maschke, Jacobson density, Clifford, Wedderburn components,…

• Character theory of finite groups
  irreducible constituents, product and norm, virtual characters, inertia groups,…
Algebra

- Abstract algebra infrastructure
  - rings, integral domains, fields, modules, vector spaces, matrices, polynomials, finite fields, algebraic numbers,…
- Linear algebra
  - Decomposition, (auto)morphisms, rank, resultants, Cayley-Hamilton, modules,…
- Elementary Galois theory
  - Field extensions, primitive element theorem, splitting fields, Galois groups, Galois norm, Hilbert’s 90 theorem, fundamental theorem,…
A Motivating Application for MARELLE

Long Term Goal: *Formal Verification of Robot Design*

Critical application: robot-*human* interaction (rescue, health care,...)

**Guarantees:**

- absence of software runtime error
- the mathematics involved are correct
- the robot satisfies some safety properties (expressed with the same mathematics)

**Beyond guarantees:**

- forces more disciplined programming
- certified ground for more advanced mathematics
MARELLE efforts

- Convex Hulls [Pichardie and Bertot 2001]
- Cylindrical algebraic decomposition
- Exact computations using Newton’s method
  [Julien and Pasca 2009]
- Formalized one step in Plane Delaunay computation
  [Dufourd and Bertot 2010]
- Perron-Frobenius Theorem [Cano 2014]
- 3D Geometry For Robot Manipulators [Affeldt and Cohen 2017]
- Inverted pendulum [Cohen and Rouhling 2017, Rouhling 2018]
- Motion planning and Computational Geometry [Bertot (wip)]
- Numerical precision [Bertot et al. 2017]
MARELLE efforts

- Convex Hulls [Pichardie and Bertot 2001]
- Exact computations using Newton’s method [Julien and Pasca 2009]
- Formalized one step in Plane Delaunay computation [Dufourd and Bertot 2010]
- Perron-Frobenius Theorem [Cano 2014]
- 3D Geometry For Robot Manipulators [Affeldt and Cohen 2017]
- Inverted pendulum [Cohen and Rouhling 2017, Rouhling 2018]
- Motion planning and Computational Geometry [Bertot (wip)]
- Numerical precision [Bertot et al. 2017]
Inverted Pendulum

CoQ Libraries involved:

- Coquelicot Real Analysis Library
  [Boldo et al. 2015]
- Mathematical Components
  [Gonthier et al. 2007–2016]

Reference: [Lozano et al. 2000]

Example: trajectories are contained in the positive limit set.

Lemma invariant_pos_limit_set V (F : V -> V) (x : R -> V) : is_sol F x -> is_invariant (pos_limit_set x).
Conclusion

Lessons learnt

• Textbook mathematics are full of holes and mistakes.
• The devil is in the details.
• Do not let Coq guide you, guide Coq.
• …
Conclusion

Lessons learnt

• Textbook mathematics are full of holes and mistakes.
• The devil is in the details.
• Do not let Coq guide you, guide Coq.
• ...

What our team efforts are about:

• Making formal proofs of mathematical facts
Conclusion

Lessons learnt

- Textbook mathematics are full of holes and mistakes.
- The devil is in the details.
- Do not let Coq guide you, guide Coq.
- ...

What our team efforts are about:

- Making formal proofs of mathematical facts
- Provide a reusable Mathematics Library
- Provide tools and methods to make proofs faster
Bibliography I


Bibliography II


