# 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,

# 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
  (*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, ...)

# 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, ...)

# History of Higher-Order Logic theorem provers

Family started by De Bruijn Authomath (1967) and Milner's LCF system (1972).

| **HOL family** | **Dependent Type Theory family** |
|---|---|
| HOL, HOL88, ..., HOL4 | Coq |
| Isabelle/HOL | Agda |
| HOL-Light | 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.
```

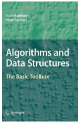
# A simple example

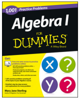
```
(* Data *)
Inductive list (A : Type) : Type :=
    nil : list A | cons : A -> list A -> list A

(* 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 // mul1mx 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

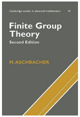
- Elementary arithmetics:

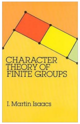
  Divisibility, means, primes, binomials...

- Indexed iterated operations:

  $\Sigma$, $\Pi$, $\bigoplus$,...

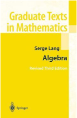
# Finite group theory

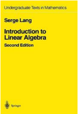



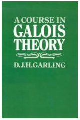




- Elementary concepts:
  Order, morphisms, permutations, quotient, characteristic subgroups, series, products, commutators, presentations,...
- Elementary theory:
  Lagrange, isomorphisms, Sylow, Hall, Jordan-Hölder theorems, structure of abelian groups, theory of various characteristic subgroups,...
- Representation theory of finite groups
  Schur, Maschke, Jacobson density, Clifford, Wedderburn components,...
- Character theory of finite groups
  irreducible constituants, 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 *[Mahboubi 2005, Cohen and Mahboubi 2012, Djalal 2018]*
- 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]*
- Cylindrical algebraic decomposition *[Mahboubi 2005, Cohen and Mahboubi 2012, Djalal 2018]*
- 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

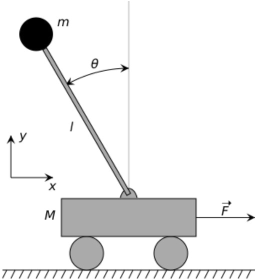



Figure: Cart pendulum (credits wikipedia)

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

R. Affeldt and C. Cohen. Formal Foundations of 3D Geometry to Model Robot Manipulators. In *Conference on Certified Programs and Proofs 2017*, Paris, France, Jan. 2017.

Y. Bertot, L. Rideau, and L. Théry. Distant decimals of $\pi$. *Journal of Automated Reasoning*, pages 1–45, 2017. URL `https://hal.inria.fr/hal-01582524`.

S. Boldo, C. Lelay, and G. Melquiond. Coquelicot: A user-friendly library of real analysis for Coq. *Mathematics in Computer Science*, 9(1):41–62, 2015. doi: 10.1007/s11786-014-0181-1.

G. Cano. *Interaction between linear algebra and analysis in formal mathematics*. Theses, Université Nice Sophia Antipolis, Apr. 2014.

C. Cohen and A. Mahboubi. Formal proofs in real algebraic geometry: from ordered fields to quantifier elimination. *Logical Methods in Computer Science*, 8(1:02):1–40, Feb. 2012. doi: 10.2168/LMCS-8(1:02)2012.

# Bibliography II

C. Cohen and D. Rouhling. A Formal Proof in Coq of LaSalle's Invariance Principle. In *Interactive Theorem Proving*, Brasilia, Brazil, Sept. 2017. doi: 10.1007/978-3-319-66107-0_10. URL https://hal.inria.fr/hal-01612293.

B. Djalal. A Constructive Formalisation of Semi-algebraic Sets and Functions. In J. Andronick and A. Felty, editors, *Certified Programs and Proofs*, Los Angeles, California, United States, Jan. 2018. URL https://hal.inria.fr/hal-01643919.

J.-F. Dufourd and Y. Bertot. Formal study of plane Delaunay triangulation. In L. Paulson and M. Kaufmann, editors, *Interactive Theorem Proving*, volume 6172, pages 211–226, Edinburgh, United Kingdom, July 2010. Springer.

G. Gonthier. The four colour theorem: Engineering of a formal proof. In D. Kapur, editor, *Computer Mathematics, 8th Asian Symposium, ASCM 2007, Singapore, December 15-17, 2007. Revised and Invited Papers*, volume 5081 of *Lecture Notes in Computer Science*, page 333. Springer, 2007. ISBN 978-3-540-87826-1. doi: 10.1007/978-3-540-87827-8_28.

# Bibliography III

G. Gonthier, A. Mahboubi, and E. Tassi. A Small Scale Reflection Extension for the Coq system. Research Report RR-6455, Inria Saclay Ile de France, 2007–2016.

G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, S. L. Roux, A. Mahboubi, R. O'Connor, S. O. Biha, I. Pasca, L. Rideau, A. Solovyev, E. Tassi, and L. Théry. A machine-checked proof of the odd order theorem. In *4th International Conference on Interactive Theorem Proving (ITP 2013), Rennes, France, July 22–26, 2013*, volume 7998 of *LNCS*, pages 163–179, 2013.

N. Julien and I. Pasca. Formal verification of exact computations using Newton's method. In *Theorem Proving in Higher Order Logics*, volume 5674, pages 408–423, Munich, Germany, Aug. 2009. Springer. doi: 10.1007/978-3-642-03359-9_28.

# Bibliography IV

R. Lozano, I. Fantoni, and D. J. Block. Stabilization of the inverted pendulum around its homoclinic orbit. *Systems & Control Letters*, 40(3):197 – 204, 2000. ISSN 0167-6911. doi: http://dx.doi.org/10.1016/S0167-6911(00)00025-6.

A. Mahboubi. Programming and certifying a CAD algorithm in the Coq system. In *Dagstuhl Seminar 05021 - Mathematics, Algorithms, Proofs*, Dagstuhl, Germany, Jan. 2005. Dagstuhl Online Research Publication Server.

D. Pichardie and Y. Bertot. Formalizing convex hull algorithms. In R. J. Boulton and P. B. Jackson, editors, *Theorem Proving in Higher Order Logics, 14th International Conference, TPHOLs 2001, Edinburgh, Scotland, UK, September 3-6, 2001, Proceedings*, volume 2152 of *Lecture Notes in Computer Science*, pages 346–361. Springer, 2001. ISBN 3-540-42525-X. doi: 10.1007/3-540-44755-5_24.

# Bibliography V

D. Rouhling. A Formal Proof in Coq of a Control Function for the Inverted Pendulum. In *CPP 2018 - 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 1–14, Los Angeles, United States, Jan. 2018. doi: 10.1145/3167101. URL https://hal.inria.fr/hal-01639819.