WIKIPEDIA
The Free Encyclopedia

Coq (software)

Coq is an interactive theorem prover first released in 1989. It allows for expressing mathematical assertions, mechanically checks proofs of these assertions, helps find formal proofs, and extracts a certified program from the constructive proof of its formal specification. Coq works within the theory of the calculus of inductive constructions, a derivative of the calculus of constructions. Coq is not an automated theorem prover but includes automatic theorem proving tactics (procedures) and various decision procedures.

The Association for Computing Machinery awarded Thierry Coquand, Gérard Huet, Christine Paulin-Mohring, Bruno Barras, Jean-Christophe Filliâtre, Hugo Herbelin, Chetan Murthy, Yves Bertot, and Pierre Castéran with the 2013 ACM Software System Award for Coq.

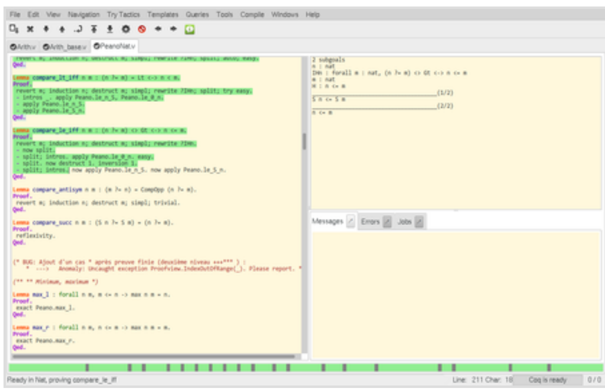
The name "Coq" is a wordplay on the name of Thierry Coquand, Calculus of Constructions or "CoC" and follows the French computer science tradition of naming software after animals (*coq* in French meaning rooster).[2] On October 11th, 2023, the development team announced that Coq will be renamed "The Rocq Prover" in the coming months, and has started updating the code base, website and associated tools.[3]

Coq (software)

Developer(s)	The Coq development team
Initial release	1 May 1989 (version 4.10)
Stable release	8.19.1[1] / 4 March 2024
Repository	github.com/coq/coq (https://github.com/coq/coq)
Written in	OCaml
Operating system	Cross-platform
Available in	English
Type	Proof assistant
License	LGPLv2.1
Website	coq.inria.fr (https://coq.inria.fr/)

Overview

When viewed as a programming language, Coq implements a dependently typed functional programming language;[4] when viewed as a logical system, it implements a higher-order type theory. The development of Coq has been supported since 1984 by INRIA, now in collaboration with École Polytechnique, University of Paris-Sud, Paris Diderot University, and CNRS. In the 1990s, ENS Lyon was also part of the project. The development of Coq was initiated by Gérard Huet and Thierry Coquand, and more than 40 people, mainly researchers, have contributed features to the core system since its inception. The implementation team has successively been coordinated by Gérard Huet, Christine Paulin-Mohring, Hugo Herbelin, and Matthieu Sozeau. Coq is mainly implemented in OCaml with a bit of C. The core system can be extended by way of a plug-in mechanism.[5]



An interactive proof session in CoqIDE, showing the proof script on the left and the proof state on the right.

The name *coq* means 'rooster' in French and stems from a French tradition of naming research development tools after animals.[6] Up until 1991, Coquand was implementing a language called the Calculus of Constructions and it was simply called CoC at this time. In 1991, a new implementation based on the extended Calculus of Inductive Constructions was started and the name was changed from CoC to Coq in an indirect reference to Coquand, who developed the Calculus of Constructions along with Gérard Huet and contributed to the Calculus of Inductive Constructions with Christine Paulin-Mohring.[7]

Coq provides a specification language called Gallina[8] ("hen" in Latin, Spanish, Italian and Catalan). Programs written in Gallina have the weak normalization property, implying that they always terminate. This is a distinctive property of the language, since infinite loops (non-terminating programs) are common in other programming languages,[9] and is one way to avoid the halting problem.

As an example, a proof of commutativity of addition on natural numbers in Coq:

```
plus_comm =
fun n m : nat =>
nat_ind (fun n0 : nat => n0 + m = m + n0)
  (plus_n_0 m)
  (fun (y : nat) (H : y + m = m + y) =>
   eq_ind (S (m + y))
     (fun n0 : nat => S (y + m) = n0)
     (f_equal S H)
     (m + S y)
     (plus_n_Sm m y)) n
     : forall n m : nat, n + m = m + n
```

`nat_ind` stands for mathematical induction, `eq_ind` for substitution of equals, and `f_equal` for taking the same function on both sides of the equality. Earlier theorems are referenced showing $m = m + 0$ and $S(m + y) = m + Sy$.

Notable uses

Four color theorem and SSReflect extension

Georges Gonthier of Microsoft Research in Cambridge, England and Benjamin Werner of INRIA used Coq to create a surveyable proof of the four color theorem, which was completed in 2002.[10] Their work led to the development of the SSReflect ("Small Scale Reflection") package, which was a significant extension to Coq.[11] Despite its name, most of the features added to Coq by SSReflect are general-purpose features and are not limited to the computational reflection style of proof. These features include:

- Additional convenient notations for irrefutable and refutable pattern matching, on inductive types with one or two constructors
- Implicit arguments for functions applied to zero arguments, which is useful when programming with higher-order functions
- Concise anonymous arguments
- An improved `set` tactic with more powerful matching
- Support for reflection

SSReflect 1.11 is freely available, dual-licensed under the open source CeCILL-B or CeCILL-2.0 license, and compatible with Coq 8.11.[12]

Other applications

- CompCert: an optimizing compiler for almost all of the C programming language which is largely programmed and proven correct in Coq.
- Disjoint-set data structure: correctness proof in Coq was published in 2007.[13]
- Feit–Thompson theorem: formal proof using Coq was completed in September 2012.[14]

Tactic language

In addition to constructing Gallina terms explicitly, Coq supports the use of *tactics* written in the built-in language Ltac or in OCaml. These tactics automate the construction of proofs, carrying out trivial or obvious steps in proofs.[15] Several tactics implement decision procedures for various theories. For example, the "ring" tactic decides the theory of equality modulo ring or semiring axioms via associative-commutative rewriting.[16] For example, the following proof establishes a complex equality in the ring of integers in just one line of proof:[17]

```
Require Import ZArith.
Open Scope Z_scope.
Goal forall a b c:Z,
    (a + b + c) ^ 2 =
     a * a + b ^ 2 + c * c + 2 * a * b + 2 * a * c + 2 * b * c.
  intros; ring.
Qed.
```

Built-in decision procedures are also available for the empty theory ("congruence"), propositional logic ("tauto"), quantifier-free linear integer arithmetic ("lia"), and linear rational/real arithmetic ("lra").[18][19] Further decision procedures have been developed as libraries, including one for Kleene algebras[20] and another for certain geometric goals.[21]

See also

Mathematics portal

Free and open-source software portal

- Calculus of constructions
- Curry–Howard correspondence
- Intuitionistic type theory
- List of proof assistants

References

1. "Release Coq 8.19.1" (https://github.com/coq/coq/releases/tag/V8.19.1). 4 March 2024. Retrieved 14 March 2024.
2. "Alternative names · coq/coq Wiki" (https://github.com/coq/coq/wiki/Alternative-names). *GitHub*. Retrieved 3 March 2023.

3. "Coq roadmap 069" (https://github.com/coq/ceps/blob/coq-roadmap/text/069-coq-roadmap.md#change-of-name-coq---the-rocq-prover). *GitHub*.
4. A short introduction to Coq (http://coq.inria.fr/a-short-introduction-to-coq)
5. Avigad, Jeremy; Mahboubi, Assia (3 July 2018). *Interactive Theorem Proving: 9th International Conference, ITP 2018, Held as ...* (https://books.google.com/books?id=I-tiDwAAQBAJ&q=%22coq%22,%22plugins%22,%22extended%22&pg=PA21) Springer. ISBN 9783319948218. Retrieved 21 October 2018.
6. "Frequently Asked Questions" (https://github.com/coq/coq/wiki/Presentation). *GitHub*. Retrieved 2019-05-08.
7. "Introduction to the Calculus of Inductive Constructions" (https://hal.inria.fr/hal-01094195/document). Retrieved 21 May 2019.
8. Adam Chlipala. "Certified Programming with Dependent Types": "Library Universes" (http://adam.chlipala.net/cpdt/html/Universes.html).
9. Adam Chlipala. "Certified Programming with Dependent Types": "Library GeneralRec" (http://adam.chlipala.net/cpdt/html/GeneralRec.html). "Library InductiveTypes" (http://adam.chlipala.net/cpdt/html/InductiveTypes.html).
10. Gonthier, Georges (2008). "Formal Proof—The Four-Color Theorem" (https://www.ams.org/notices/200811/tx081101382p.pdf) (PDF). *Notices of the American Mathematical Society*. **55** (11): 1382–1393. MR 2463991 (https://mathscinet.ams.org/mathscinet-getitem?mr=2463991).
11. Gonthier, Georges; Mahboubi, Assia (2010). "An introduction to small scale reflection in Coq" (https://jfr.unibo.it/article/view/1979). *Journal of Formalized Reasoning*. **3** (2): 95–152. doi:10.6092/ISSN.1972-5787/1979 (https://doi.org/10.6092%2FISSN.1972-5787%2F1979).
12. "The Mathematical Components Library 1.11.0" (https://github.com/math-comp/math-comp/releases/tag/mathcomp-1.11.0). *GitHub*.
13. Conchon, Sylvain; Filliâtre, Jean-Christophe (2007). "A persistent union-find data structure". In Russo, Claudio V.; Dreyer, Derek (eds.). *Proceedings of the ACM Workshop on ML, 2007, Freiburg, Germany, October 5, 2007* (https://www.lri.fr/~filliatr/puf/). Association for Computing Machinery. pp. 37–46. doi:10.1145/1292535.1292541 (https://doi.org/10.1145%2F1292535.1292541).
14. "Feit-Thompson theorem has been totally checked in Coq" (https://web.archive.org/web/20161119094854/http://www.msr-inria.fr/news/feit-thomson-proved-in-coq/). Msr-inria.inria.fr. 2012-09-20. Archived from the original (http://www.msr-inria.fr/news/feit-thomson-proved-in-coq/) on 2016-11-19. Retrieved 2012-09-25.
15. Kaiser, Jan-Oliver; Ziliani, Beta; Krebbers, Robbert; Régis-Gianas, Yann; Dreyer, Derek (2018-07-30). "Mtac2: typed tactics for backward reasoning in Coq" (https://doi.org/10.1145%2F3236773). *Proceedings of the ACM on Programming Languages*. **2** (ICFP): 78:1–78:31. doi:10.1145/3236773 (https://doi.org/10.1145%2F3236773). hdl:21.11116/0000-0003-2E8E-B (https://hdl.handle.net/21.11116%2F0000-0003-2E8E-B).
16. Grégoire, Benjamin; Mahboubi, Assia (2005). "Proving Equalities in a Commutative Ring Done Right in Coq". In Hurd, Joe; Melham, Tom (eds.). *Theorem Proving in Higher Order Logics: 18th International Conference, TPHOLs 2005, Oxford, UK, August 22–25, 2005, Proceedings* (https://link.springer.com/chapter/10.1007/11541868_7). Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. pp. 98–113. doi:10.1007/11541868_7 (https://doi.org/10.1007%2F11541868_7). ISBN 978-3-540-31820-0.
17. "The ring and field tactic families — Coq 8.11.1 documentation" (https://coq.inria.fr/doc/V8.11.1/refman/addendum/ring.html). *coq.inria.fr*. Retrieved 2023-12-04.

18. Besson, Frédéric (2007). "Fast Reflexive Arithmetic Tactics the Linear Case and Beyond" (https://link.springer.com/chapter/10.1007/978-3-540-74464-1_4). In Altenkirch, Thorsten; McBride, Conor (eds.). *Types for Proofs and Programs: International Workshop, TYPES 2006, Nottingham, UK, April 18–21, 2006, Revised Selected Papers*. Lecture Notes in Computer Science. Vol. 4502. Berlin, Heidelberg: Springer. pp. 48–62. doi:10.1007/978-3-540-74464-1_4 (https://doi.org/10.1007%2F978-3-540-74464-1_4). ISBN 978-3-540-74464-1.
19. "Micromega: solvers for arithmetic goals over ordered rings — Coq 8.18.0 documentation" (https://coq.inria.fr/doc/V8.18.0/refman/addendum/micromega.html). *coq.inria.fr*. Retrieved 2023-12-04.
20. Braibant, Thomas; Pous, Damien (2010). Kaufmann, Matt; Paulson, Lawrence C. (eds.). *An Efficient Coq Tactic for Deciding Kleene Algebras* (https://link.springer.com/chapter/10.1007/978-3-642-14052-5_13). Interactive Theorem Proving: First International Conference, ITP 2010 Edinburgh, UK, July 11-14, 2010, Proceedings. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. pp. 163–178. doi:10.1007/978-3-642-14052-5_13 (https://doi.org/10.1007%2F978-3-642-14052-5_13). ISBN 978-3-642-14052-5. S2CID 3566183 (https://api.semanticscholar.org/CorpusID:3566183).
21. Narboux, Julien (2004). "A Decision Procedure for Geometry in Coq" (https://link.springer.com/chapter/10.1007/978-3-540-30142-4_17). In Slind, Konrad; Bunker, Annette; Gopalakrishnan, Ganesh (eds.). *Theorem Proving in Higher Order Logics: 17th International Conference, TPHOLS 2004, Park City, Utah, USA, September 14–17, 2004, Proceedings*. Lecture Notes in Computer Science. Vol. 3223. Berlin, Heidelberg: Springer. pp. 225–240. doi:10.1007/978-3-540-30142-4_17 (https://doi.org/10.1007%2F978-3-540-30142-4_17). ISBN 978-3-540-30142-4. S2CID 11238876 (https://api.semanticscholar.org/CorpusID:11238876).

External links

- Official website (http://coq.inria.fr), in English
- Coq (https://github.com/coq) on GitHub, source code repository
- JsCoq Interactive Online System (https://x80.org/rhino-coq/) – allows Coq to run in a web browser, with no need to install extra software
- Alectryon (https://plv.csail.mit.edu/blog/alectryon.html) – a library to process Coq snippets embedded in documents, showing goals and messages for each Coq sentence
- Coq Wiki (https://github.com/coq/coq/wiki)
- Mathematical Components library (https://math-comp.github.io/math-comp/) – widely used library of mathematical structures, part of which is the SSReflect proof language
- Constructive Coq Repository at Nijmegen (http://corn.cs.ru.nl/)
- Math Classes (https://math-classes.github.io/)
- Coq (https://www.openhub.net/p/coq) at Open Hub

Textbooks

- The Coq'Art (http://www.labri.fr/perso/casteran/CoqArt/index.html) – a book on Coq by Yves Bertot and Pierre Castéran
- Certified Programming with Dependent Types (http://adam.chlipala.net/cpdt/) – online

and printed textbook by Adam Chlipala

- Software Foundations (http://www.cis.upenn.edu/~bcpierce/sf/) – online textbook by Benjamin C. Pierce et al.
- An introduction to small scale reflection in Coq (http://jfr.unibo.it/article/view/1979) – a tutorial on SSReflect by Georges Gonthier and Assia Mahboubi

Tutorials

- Introduction to the Coq Proof Assistant (http://video.ias.edu/univalent/appel) – video lecture by Andrew Appel at Institute for Advanced Study
- Coq Video tutorials (http://math.andrej.com/2011/02/22/video-tutorials-for-the-coq-proof-assistant/) by Andrej Bauer

---