Report on the development of Coq

August 2009

Outline

The Coq development

Support for third-party extensions

The web site

Cocorico

The user contributions

Coq 8.2

Coq 8.3

The current research directions

The tactics and $\mathcal{L}_{tac}$

The library

Using Coq for teaching

Miscellaneous questions

Discussion

The Coq development

Born in 1984 (Gérard Huet, Thierry Coquand, Christine Paulin)

About 40 contributors over 25 years

Main code developers for Coq 8.2 were Matthieu Sozeau, Hugo Herbelin, Arnaud Spiwack, Élie Soubiran, Bruno Barras, Jean-Marc Notin with contributions from Frédéric Besson, Vincent Siles, Pierre Letouzey, Stéphane Glondu, Pierre Courtieu, Julien Forest, Jean-Christophe Filliâtre, Julien Narboux, Yves Bertot, Nicolas Tabareau, Lionel Mamane.

Main developers for the library were Pierre Letouzey, Evgeny Makarov, Laurent Théry, Benjamin Grégoire with sponsored contributions from Aaron Bohannon, Russell O'Connor, Cezary Kaliscyk, Milad Niqui.

Various gracious bug-fixes provided on coq-bugs.

Many suggestions from intensive users on Coq-Club and coq-bugs.

Coq is typically a collaborative system with its associated drawbacks (different styles of programming, redundant code by accident, dead code).

Open source at http://gforge.inria.fr/coq (*svn* tree). Probable move to *git* at some time.

Support for third-party extensions

The system of plugins introduced with Coq 8.2 facilitates the support for third-party extensions such as `Ssreflect`, but...

Currently no true documentation (the code is the documentation)

APIs not stable

Modularity: make of Coq a library of components (unification, proof engine, type-checker?)

- Done: `coqchk` checker is an stand-alone `vo`-files type-checker

- In progress: CoqIDE split out of Coq (use of a communication protocol)

The web site

New web site released a few months ago.

Cocorico

It is a wiki: anyone can contribute!

The user contributions

We strongly encourage users to submit their formalisation, tactics or plugins!

- It adds visibility
- It ensures that the development will be ported to new versions of Coq by the development team
- It helps the development team to evaluate the degree of incompatibilities introduced by some modification of Coq
- It provides a test of non-regression and a benchmark for the evolution of Coq

The new repository (thanks to J.-M. Notin) provides browsing of sources. We plan to have a dedicated *library* section.

Coq 8.2

The most striking features were

- the introduction of *type classes*

- the support for *plugins* (thanks to Objective Caml new dynamic binding)

Less visible because more in the continuous extensions of previous works are

- improved type inference

- improved tactics (including setoid rewriting)

- smoother module system

- extended decision of arithmetics (`micromega`)

- the stand-alone stand-alone type-checker

Unfortunately unfinished is the reworking of the arithmetical libraries.

Coq 8.3

Release of Coq 8.3 is planned for the end of Autumn.

It will be a relatively small release whose most important feature will be the introduction of *structured proof scripts*.

In addition, there will be various improvements of existing features (tactics, library, module system, ...).

The current research directions

Towards Coq as a dependently-typed programming language (DTPL)

- Improving the evaluation machine
- Improved support for extensional equations in dependent typing
- Improving the type inference
- Certifying the chain of extraction from Coq to O'Caml, to assembly language

Implicit calculus of inductive constructions (currently a parallel track)

A still smoother and "high-level" everyday use of the system

Other foundational questions

Suspended problems:

- Proof-irrelevance

Other open problems:

- Is there a need for smoother termination evidence for (co-)recursive functions in the underlying theory or are `measure` and `wf` enough?

- Commutative cuts (i.e. having an application or an external `match` enter an inner `match`)?

The underlying evaluation machine

Planned features:

- persistent arrays in the bytecode call-by-value machine (in progress)

- pattern-matching with support for inversion equality (in progress)

- support for monadic views of native imperative features (long term)

Improving the type inference

Planned features:

- support for eta

- support for a form “unification hints” and more (e.g. unification modulo AC, modulo isomorphic notions)

Support for extensional equations in dependent typing

Make the use of *dependent types* easier by supporting provable equations in the conversion. E.g., could we be eventually able to type

```
forall (l1:vector n1) (l2:vector n2), rev (l1++l2) = rev l1 ++ rev l2
```

Dependent pattern-matching à la Agda (and more)

- `Program` and `Equations`

- Being able to see equations of the form `x=t` as local definitions

The tactics and $\mathcal{L}_{tac}$

There are many tactics for interactive or automatic reasoning, and a language to write new tactics but:

- Often, something is missing (e.g. `auto` is not able to `destruct` conjunctions, nor to efficiently `discriminate`), ...

- Numerous informal requests for extending $\mathcal{L}_{tac}$... (see Cocorico)

The library

The Coq library is a collection of files of *different authors* written at *different times*. There is no uniformity neither in the way definitions are written, nor in the naming conventions.

The proofs often denote a use of tactics from many years ago.

Reasoning on decidable notions using computation is under-developed.

My feeling is that we did not find the right *development model* for the library yet. In particular, we are expecting from the dutch MathWiki project for stimulating the developments of libraries in collaborative way.

We plan to set up a “policy” for the integration of third-party contributions

The library: a continuous evolution or new start

Should we progressively evolve to a better-structured library with compatibility wrappers?

Should we provide a one-shot translation tool for setting up a new more consistent library?

Should we renounce to compatibility?

Are tutorials missing?

There are often *different ways* to formalise the same thing.

There are *different styles of tactic scripts* depending on whether we put the accent on concision, contextual robustness, readibility, ...

Shall we recommend formalisation styles? Compare the tutorials or textbooks from

- Yves Bertot and Pierre Castéran (Coq'Art)
- Yves Bertot's (Coq in a Hurry)
- Benjamin Pierce's course notes
- Adam Chlipala's "Certified Programming with Dependent Types" notes
- The Mathematical Components' library design choices

Using Coq for teaching

There have been many experiences of teaching Coq or using Coq for teaching: see Cocorico's dedicated page.

Miscellaneous questions?

Is a nominal package needed? (e.g. is the U. Penn binders toolkit enough or not?)

Should we investigate the C-zar mathematical style language further?

Next Coq workshop

Plan to go with ITP (July 2010, Edinburgh).

Discussion

Any opinion on the compatibility vs improvements dialectic? Especially for the tactics and the library.

Any opinion on how to improve the development third-party contributions?

Any opinion on the wiki?