A Tale of Two Beasts: Coq & OCaml

Pierre Letouzey
πr² team

...@pps.univ-paris-diderot.fr
...@inria.fr

13 Sept. 2012
Coq?

- **What**: a proof assistant.
Coq ?

- **What**: a proof assistant.
  - a type-checker / compiler for the CIC logic / language

- When: since 1985 (almost as old as Caml)
- Who: Inria teams, French univs, contributors worldwide
- Licence: LGPL
- Size: 200 Kloc (OCaml sources) + 200 Kloc (Coq stdlib)
- Users: thousands according to the mailing-list
- Feats: 4-color theorem, Compcert certified C compiler, ...
Coq?

- **What**: a proof assistant.
  - a type-checker / compiler for the CIC logic / language
- **When**: since 1985 (almost as old as Caml)
Coq ?

- **What**: a proof assistant.
  - a type-checker / compiler for the CIC logic / language
- **When**: since 1985 (almost as old as Caml)
- **Who**: Inria teams, French univs, contributors worldwide
Coq?

- **What**: a proof assistant.
  - a type-checker / compiler for the CIC logic / language
- **When**: since 1985 (almost as old as Caml)
- **Who**: Inria teams, French univs, contributors worldwide
- **Licence**: LGPL
Coq?

- **What**: a proof assistant.
  - a type-checker / compiler for the CIC logic / language
- **When**: since 1985 (almost as old as Caml)
- **Who**: Inria teams, French univs, contributors worldwide
- **Licence**: LGPL
- **Size**: 200 Kloc (OCaml sources) + 200 Kloc (Coq stdlib)

Users: thousands according to the mailing-list

Feats: 4-color theorem, Compcert certified C compiler, ...
Coq?

- **What**: a proof assistant.
  - a type-checker / compiler for the CIC logic / language
- **When**: since 1985 (almost as old as Caml)
- **Who**: Inria teams, French univs, contributors worldwide
- **Licence**: LGPL
- **Size**: 200 Kloc (OCaml sources) + 200 Kloc (Coq stdlib)
- **Users**: thousands according to the mailing-list

Feats: 4-color theorem, Compcert certified C compiler, ...
Coq?

- **What:** a proof assistant.
  - a type-checker / compiler for the CIC logic / language
- **When:** since 1985 (almost as old as Caml)
- **Who:** Inria teams, French univs, contributors worldwide
- **Licence:** LGPL
- **Size:** 200 Kloc (OCaml sources) + 200 Kloc (Coq stdlib)
- **Users:** thousands according to the mailing-list
- **Feats:** 4-color theorem, Compcert certified C compiler, ...
Why OCaml?

Roughly, Coq’s main job is to play with ASTs.
Why OCaml?

Roughly, Coq’s main job is to play with ASTs

Coq terms: formulas / functions / proof fragments…
Why OCaml?

Roughly, Coq’s main job is to play with ASTs.

Coq terms: formulas / functions / proof fragments...

Many phases:

- parse the user textual input (constreexpr)
- resolve constant names
- expand notations (glob_constr)
- fill implicit arguments
- compile to primitive pattern-matching
- type-check (constr)
Why OCaml?

Roughly, Coq’s main job is to play with ASTs

Coq terms: formulas / functions / proof fragments...

Many phases:
- parse the user textual input (constrexpr)
- resolve constant names
- expand notations (glob_constr)
- fill implicit arguments
- compile to primitive pattern-matching
- type-check (constr)

Extra operations: compute, display, interactive build (proofs)
Why OCaml?

Roughly, Coq’s main job is to play with ASTs.

Coq terms: formulas / functions / proof fragments...

Many phases:

- Parse the user textual input (constrexp)
- Resolve constant names
- Expand notations (glob_constr)
- Fill implicit arguments
- Compile to primitive pattern-matching
- Type-check (constr)

Extra operations: compute, display, interactive build (proofs)

Other ASTs: commands, tactics, ...
Why OCaml?

Manipulating ASTs: right in the scope of ML dialects!

What we appreciate in OCaml:

▶ a fast compiler producing fast native code
▶ convenient dev tools (toplevel, debugger, ...)
▶ decent trade-off between stability and novelties
▶ ...

Overall, we’re really happy OCaml users
Advanced features?

Use of advanced OCaml features: not so much

- rectypes: yes
- serialization: a lot
- lazyness: a bit
- objects: barely (in the GUI coqide, via lablgtk)
- 1st class modules: no
- GADT: no
- Native-code dynlink: yes!
Advanced features?

To help compiling Coq on many systems:

- We try to support relatively old OCaml versions
  - Today’s requirement: \( \geq 3.11.2 \).
  - Roughly : what is available in Debian Stable.
Advanced features?

To help compiling Coq on many systems:

- We try to support relatively old OCaml versions
  - Today’s requirement: \( \geq 3.11.2 \).
  - Roughly: what is available in Debian Stable.

- We also tend to avoid external dependencies
  - Special case of the GUI (coqide): lablgtk, sourceview, ...
Extensibility

An important design choice: extensibility

- Extensible syntax
- Extensible set of commands and tactics
Extensibility

An important design choice: extensibility

- Extensible syntax
- Extensible set of commands and tactics

For simple extensions, directly in a Coq script file:

- Notation, Tactic Notation
- Ltac: a dedicated ad-hoc tactic language
Extensibility

An important design choice: extensibility

- Extensible syntax
- Extensible set of commands and tactics

For simple extensions, directly in a Coq script file:

- Notation, Tactic Notation
- Ltac: a dedicated ad-hoc tactic language

For more advanced needs, some extra OCaml code:

- Loadable plugins (native-code dynlink)
Extensible grammar : Camlp4

Camlp4 provides an extensible LL(1) parsing engine
▶ Not ideal (some factorization of rules by hand)
▶ Alternative ? GLR ?
Extensible grammar: Camlp4

Camlp4 provides an extensible LL(1) parsing engine
  ▶ Not ideal (some factorization of rules by hand)
  ▶ Alternative? GLR?

Two flavors: Camlp4 / Camlp5
Extensible grammar : Camlp4

Camlp4 provides an extensible LL(1) parsing engine
  ▶ Not ideal (some factorization of rules by hand)
  ▶ Alternative ? GLR ?

Two flavors : Camlp4 / Camlp5

In practice:
  ▶ Grammar rules are declared in OCaml files
  ▶ These OCaml files need pre-processing (syntax extension)
  ▶ In the Coq sources : *.ml4
Build System

Coq’s Makefile is notoriously horrible
Some OCaml files use quotations:
  ▶ They embed some pieces of tactic or constr syntax
  ▶ Finding deps of these files require the Coq parser!
  ▶ Many phases in the build...
  ▶ We currently rely on some gnu-make ”features”:
    ▶ iterated rebuild attempt of included files ...
OCaml is also hard to handle correctly in ”make”:
  ▶ multi-sources (.ml / .mli), but not always
  ▶ multi-targets (.cmo / .cmi)
Build System

An experimental attempt at using Omahbuild

- way shorter than Makefile
- but quite tricky as well
- efficiency issues (work-in-progress)

Many other recent build tools to try (omake, scons, ...)

Refactoring

Invaluable help of strong typing (and abstract types).
Example of the ongoing reform:

▶ from pyramidal accumulation organization ...
▶ ... to component-based organization with type interfaces
▶ Lighter Coq parser grammar.cma
  ▶ No need to have the "kernel" type-checker in it!

Ultimate goal:

▶ independent parser ?
▶ Coq as a library usable as backend by other tools (API !).
Computations

Many reduction engines on Coq terms:

▸ **Yesterday**: a basic one done in OCaml: `reduction.ml`

▸ **Today**: one bytecode runtime in C: `coq_interp.c`

▸ **Tomorrow**: calls to ocamlopt + nat dynlink + run
Dyn

Example of some dirty tricks (Obj.magic)

A priori unsafe, but very isolated and dynamically checked

Now 1st class modules and/or GADT might probably help.

If it works, don’t fix it!
OCaml upgrades

Almost never a big deal.

An exception: reversed Set.fold in 3.09

▶ ... following discussion on a Coq certification of Set!

Btw, nice Caml-Coq interactions: certifications of new Set.filter...
Portability

Most Coq development done on Linux (+ MacOS).

The Windows port is a pain.
  ▶ Cross-compilation
  ▶ Unix compatibility (signals), etc etc
Delicate programming aspects

Hash-consing

- In a persistent way, with very deep AST
- For instance 1000 in nat is (S (S (S ... 0)))).
- Hashtbl.hash not enough, revised version by Y. Régis-Gianas
Delicate programming aspects

Generic comparison

- `Pervasives.compare` is both a blessing and a curse
- Same with generic serialization
  `(Pervasives.output_value)` used in compiled `.vo`
- Example: universes level.
  - Algebraic int variables with constraints between them.
  - Variable names = int not enough (multi-session)
  - Variable names = `(modname, int)` : order matters!
Delicate programming aspects

Exceptions

- Many bug reports: Uncaught exception (e.g. Not_found)
- Irritating, even if fixed after 5 min in ocamldebug
- Coq’s jungle of exceptions isn’t “backtrace”-friendly
- Nasty effects of (try ... with _ ---> ...)
- Some static analysis could probably help ...
Code quality

Old & large code base with few maintainers: cleanup needed!

- New OCaml warnings...
- Static analysis tools, dead code detector (Oug of M. Guesdon),
- Code quality tools by X. Clerc
- Revue and re-evaluate type cheating (Obj.magic)
Code safety

One of Coq paradox:

- we promote specs+certifications when using Coq
- but Coq itself uses a rather old-style dev method (tests...)

Hopefully all code isn’t critical

- the ”kernel” type-checker
- but even parsing or display shouldn’t be neglected

An external checker, working on compiled files.
Certifying (parts of) Coq is a research topic (Bootstrap)
Conclusion

- Coq, an old software still in active development
- A quite unique large-scale experimentation field