Coq 8.5 at work

M. Sozeau, M. Dénès – Inria

CoqPL
January 23rd 2016
St Petersburg, FL

Announcement

COQ 8.5 is released (at last)!

coq.inria.fr/coq-85

Coq 8.5

- A year of beta-testing (3 betas), due to cross-cutting, “deep” features.
- Many improvments in interaction, semantics of the new proof engine constructs and usability of universes.
- More external contributions than ever: pull requests are welcome!

Next episode: code streamlining, tactics and interface accessibility. Expected deprecation of features and factoring of similar functionalities.

Coq 8.5 at work

1. Coq 8.5 features
   - Incremental development – E. Tassi
   - New proof engine – A. Spiwack
   - Universe polymorphism – M. Sozeau
   - Native compilation – M. Dénès & B. Grégoire
   - Fast record projections – M. Sozeau
   - Misc – Coq dev team & contributors
   - `opam`
   - Performance
2. Coq future
3. Focus on universe polymorphism

Incremental development

Enrico Tassi

Asynchronous and parallel processing of definitions. Separate compilation.
Huge gain in user productivity.

$-/++$ *Not optional, backwards-compatible*

$+++$ *Faster interaction and parallel compilation*

New proof engine

Arnaud Spiwack

New proof engine – A. Spiwack

Expressive and clear proof-search semantics, dependent subgoals, managment of subgoals.

−/++ *Not optional, backwards-compatibility layer*

−/+ *0-15% time overhead, unnoticeable as the rest of the system got faster/Opportunities to have faster primitive tactics*

Universe polymorphism

Matthieu Sozeau

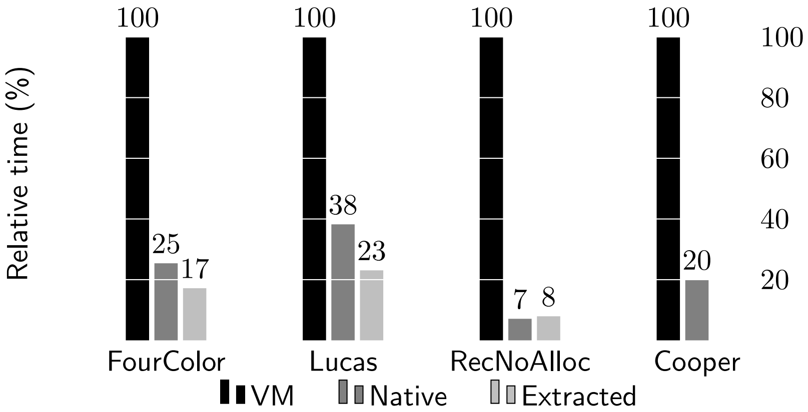
Truly polymorphic definitions and inductives, cleaner kernel.

$=/++$ *Kernel change - impacts the ML hacker only*.
Backwards-compatibility layer.

$=+/+$ *Comparable or better performance, more expressive*

Native compilation

Maxime Dénès & Benjamin Grégoire


100
100
100
100
100
80
60
40
20
Relative time (%)
25
17
38
23
7
8
20
FourColor
Lucas
RecNoAlloc
Cooper
VM
Native
Extracted

Native compilation – M. Dénès & B. Grégoire

Down to assembly through OCaml.
Useful for large reflection proofs.

$+$ *Optional*

$++/-$ *Faster at runtime, compilation is slow*

Fast record projections

Matthieu Sozeau

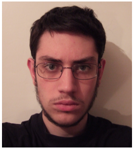
Faster conversion and type-checking, smaller memory footprint.

$+/+-$ *Optional, backwards-compatibility layer, small source-level incompatibilities.*

$+^{\omega}$ *Exponentially better performance*

Misc – Coq dev team & contributors

- Interfaces, documentation, and OCaml best practices (P.M. Pédrot, …).
- Tactics in terms (P.M. Pédrot)
- Module system simplifications (P. Letouzey)
- Tactic improvements (e.g. intro patterns) (H. Herbelin, P. Letouzey, . . .)
- More expressive guard condition (P. Boutillier, H. Herbelin)
- Rewriting with strategies (M. Sozeau)




- Coq, any version, git included
- Packages: Ssreflect, Mathematical Components, Containers, Coccinelle, Ergo, …
- Submit a pull request! Try! Test!
- Caution: recommended setup still in discussion.

opam.ocaml.org
https://github.com/coq/opam-coq-archive

Performance

Through careful profiling, many hotspots identified and optimized (thanks to P.M. Pédrot).

- Performance closer to 8.3, despite the many new features: checking of universes, STM layer, more expressive proof engine.
- Hash-consing is used more pervasively: smaller memory footprint and proof objects.

Faster universe algorithm scheduled for next version (J.H. Jourdan).

Coq future

Improving the development

Coq 8.5 showed the limits of the current development model.

We are working on improving the development process:

- Predictible short release cycles
- Opening up to external contributions
- Communicating on forthcoming evolutions

Coq consortium

A consortium of academic and industrial users of Coq is being built. Its role will be:

- Coordinating the engineering effort on Coq
- Sharing resources
- Providing premium support to members
- Collecting annual membership fees and allocating resources through a steering committee of members

Contact us!

`maxime.denes@inria.fr`
`yves.bertot@inria.fr`

Universe Polymorphism

Polymorphic Universes

- Allow generic developments over universe levels.
- As with typical ambiguity, can be made entirely implicit.
- Explicit mode for careful control of universe instances.
- Compatible with asynchronous checking of proofs and fast conversion algorithms (`vm_compute` and `native_compute`).

Typical ambiguity

Working with explicit universe indices is cumbersome, annotations pervade definitions and proofs.

$\Rightarrow$ Allow *typical ambiguity* (first used by Russell in Principia).

Idea: write `Type` to mean any type that "fits" (keeps the system consistent).

- On paper: let the reader infer levels for universes and check consistency.
- On computer: let the computer infer levels and check consistency in the background.

Floating universes

Formally, translate from anonymous $\texttt{Type}$s to explicit $\texttt{Type}_i$s.
But in general many i's can work!

$$\texttt{Definition}\ \mathrm{id}\ (A : \texttt{Type})\ (a : A) := a.$$

$$\leadsto\ \vdash \mathrm{id} : \Pi(A : \texttt{Type}_0),\ A \to A : \texttt{Type}_1$$

or

$$\leadsto\ \vdash \mathrm{id} : \Pi(A : \texttt{Type}_1),\ A \to A : \texttt{Type}_2$$

or . . . ?

$\Rightarrow$ universe variables

Floating universes and constraints

Consistency ensured by giving an assignment of natural numbers to universe variables, satisfying *constraints*. New judgment $\vdash_{float}$

$$\textsc{Type-intro}\quad \frac{\vdash_{float} \Gamma \quad (i, j \in \mathbb{L})}{\Gamma \vdash_{float} \texttt{Type}_i : \texttt{Type}_j \leadsto i < j}$$

$$\textsc{Type-prod}\quad \frac{\Gamma \vdash_{float} A : \texttt{Type}_i \qquad \Gamma, x : A \vdash B : \texttt{Type}_j}{\Gamma \vdash_{float} \Pi x : A.B : \texttt{Type}_k \leadsto \max(i, j) \leq k}$$

Without polymorphism

Floating levels + cumulativity give a *restricted form* of polymorphism:

$$\texttt{Definition}\ \mathrm{id}\ (A : \texttt{Type})\ (a : A) := a$$

$$\rightsquigarrow\ \vdash \mathrm{id} : \Pi(A : \texttt{Type}_l),\ A \to A : \texttt{Type}_{l+1}$$

$\Rightarrow$ l is not quantified at the definition level here, it is *global*:

$$\nvdash \mathrm{id}\ (\Pi(A : \texttt{Type}_l),\ A \to A)\ \mathrm{id} : \tau$$

Because $l + 1 \not\leq l$. However l can go “up” as far as required.

With polymorphism

Real, bounded polymorphism:

$$\texttt{Polymorphic Definition}\ \mathrm{id}\ (A : \texttt{Type})\ (a : A) := a$$

$$\underline{\mathrm{id}}_l : \Pi(A : \texttt{Type}_l),\ A \to A$$

$\Rightarrow l$ is quantified at the definition level now and we can *instantiate* it at each application:

$$l < k \vdash_{poly} \underline{\mathrm{id}}_k\ \underline{\mathrm{id}}_l : \Pi(A : \texttt{Type}_l),\ A \to A$$

Constraint checking

Constraints are generated once at refinement time outside the kernel. The kernel just checks that the constraints are consistent and sufficient to typecheck the terms.

$$\text{universe context} \quad \Psi \quad ::= \quad \overrightarrow{i} \vDash \Theta$$

Constraint checking

Constraints are generated once at refinement time outside the kernel. The kernel just checks that the constraints are consistent and sufficient to typecheck the terms.

$$\text{universe context} \quad \Psi \quad ::= \quad \overrightarrow{i} \vDash \Theta$$

Elaboration in bidirectionl fashion:

- Inference: $\Gamma; \Psi \vdash t \Uparrow \rightsquigarrow \Psi' \vdash t' : T$
- Checking: $\Gamma; \Psi \vdash t \Downarrow T \rightsquigarrow \Psi' \vdash t' : T$

Constraint checking

Constraints are generated once at refinement time outside the kernel. The kernel just checks that the constraints are consistent and sufficient to typecheck the terms.

$$\text{universe context} \quad \Psi \quad ::= \quad \overrightarrow{i} \vDash \Theta$$

Elaboration in bidirectionl fashion:

- Inference: $\Gamma; \Psi \vdash t \Uparrow \rightsquigarrow \Psi' \vdash t' : T$
- Checking: $\Gamma; \Psi \vdash t \Downarrow T \rightsquigarrow \Psi' \vdash t' : T$

$$\textsc{Check-Type}\ \frac{\theta \vdash \texttt{Type}_{i+1} \leq T \rightsquigarrow \theta'}{\Gamma; us \vDash \theta \vdash \texttt{Type} \Downarrow T \rightsquigarrow us, i \vDash \theta' \vdash \texttt{Type}_i : T}$$

Introducing universe polymorphic definitions

Suppose a top-level `Definition` $\mathrm{c} : T := t$.

Introducing universe polymorphic definitions

Suppose a top-level `Definition` $\mathrm{c} : T := t$.

1. $\Gamma; \vdash T \Uparrow \rightsquigarrow \Psi \vdash T' : s$

Introducing universe polymorphic definitions

Suppose a top-level `Definition` $\mathrm{c} : T := t$.

1. $\Gamma; \vdash T \Uparrow \rightsquigarrow \Psi \vdash T' : s$
2. $\Gamma; \Psi \vdash t \Downarrow T' \rightsquigarrow \Psi' \vdash t' : T'$

Introducing universe polymorphic definitions

Suppose a top-level `Definition` $\mathrm{c} : T := t$.

1. $\Gamma; \vdash T \Uparrow \rightsquigarrow \Psi \vdash T' : s$
2. $\Gamma; \Psi \vdash t \Downarrow T' \rightsquigarrow \Psi' \vdash t' : T'$
3. Add $\mathrm{c} : \forall\ \Psi', T' := t'$ to the environment.
4. Each use of c carries a universe instance: $\mathrm{c}_{\vec{l}} : T'[\overrightarrow{l/i}]$

Guiding principle and main difficulty:
Constants are transparent, indistinguishable from their bodies.

Unification

Unification of id_i and id_j:

`Definition` $U2 := \texttt{Type}_i$.

`Definition` $U1 : U2 := \texttt{Type}_j \leadsto j < i$

`Definition` $U0 : U1 := \texttt{Type}_k \leadsto k < j$

`Definition` $U02 : U2 := U0 \leadsto k < i$

$$\mathrm{id}_i\ U02 \ \sim\ \mathrm{id}_j\ U0 \leadsto i = j$$

But:

$$\mathrm{id}_i\ U02 \rightarrow^* (U0 \rightarrow U0) \leftarrow^* \mathrm{id}_j\ U0$$

Unification and Conversion

$$\dfrac{\overrightarrow{as} =_{\psi}^{=} \overrightarrow{bs} \qquad \psi \models \overrightarrow{u} = \overrightarrow{v}}{\underline{\mathrm{c}}_{\overrightarrow{u}}\ \overrightarrow{as} =_{\psi}^{R} \underline{\mathrm{c}}_{\overrightarrow{v}}\ \overrightarrow{bs}}\ \textsc{Conv-FO}$$

Uses backtracking (Ziliani & Sozeau, ICFP'15).

Least-commitment principle

Use two kinds of universe level variables during elaboration:

- Polymorphic constants get elaborated with fresh flexible argument levels that can be unified.
- Typical ambiguity (e.g. `Type`) creates rigid variables.
- User-given levels are rigid

Design choice

Universe instances are levels: Suppose

$$\mathrm{id} : \forall i, \Pi A : \texttt{Type}_i, A \rightarrow A$$

Levels only, adding constraint if an algebraic would appear:

$$\Gamma; \vdash \mathrm{id}\ \texttt{Type} \Uparrow \rightsquigarrow i\ j \vDash i < j \vdash \mathrm{id}_j\ \texttt{Type}_i : \texttt{Type}_i \rightarrow \texttt{Type}_i$$

and not:

$$\Gamma; \vdash \mathrm{id}\ \texttt{Type} \Uparrow \rightsquigarrow i \vdash \mathrm{id}_{i+1}\ \texttt{Type}_i : \texttt{Type}_i \rightarrow \texttt{Type}_i$$

Minimization

That's *a lot* of fresh universe variables!!

Typical example:

$$\Gamma; \vdash \mathrm{id}\ \mathsf{true} \Uparrow \rightsquigarrow i_{\mathsf{f}} \vDash \mathtt{Set} \leq i \vdash @\mathrm{id}_i\ \mathsf{bool}\ \mathsf{true} : \mathsf{bool}$$

Minimization

That's *a lot* of fresh universe variables!!

Typical example:

$$\Gamma; \vdash \mathrm{id}\ \mathsf{true} \Uparrow\ \rightsquigarrow i_{\mathsf{f}} \vDash \mathtt{Set} \leq i \vdash @\mathrm{id}_i\ \mathsf{bool}\ \mathsf{true} : \mathsf{bool}$$

We'd want: $@\mathrm{id}_{\mathtt{Set}}$ bool true : bool, no new universe, no additional constraint, just as general.

Minimization

That's *a lot* of fresh universe variables!!

Typical example:

$$\Gamma; \vdash \mathrm{id}\ \mathsf{true} \Uparrow \rightsquigarrow i_{\mathsf{f}} \vDash \mathtt{Set} \leq i \vdash @\mathrm{id}_i\ \mathsf{bool}\ \mathsf{true} : \mathsf{bool}$$

We'd want: $@\mathrm{id}_{\mathtt{Set}}\ \mathsf{bool}\ \mathsf{true} : \mathsf{bool}$, no new universe, no additional constraint, just as general.

$\Rightarrow$ Minimization: compute a minimal set of universe variables.

See Cardelli's greedy algorithm for $F^{\leq}$ inference, local type inference (Pierce & Turner).

- Only applies to flexible variables.

Minimization, results

Correctness proof: easy, preservation of local solutions.

Of course this is *not* endangering the consistency of Coq!

Theorem (*Conservativity*)

Unfolding universe polymorphic definitions gives correct typings in the original system. Might just not be the most general ones if minimization did anything. For inductives, each instantiation is a new copy.

Dealing with Prop

Let $\underline{\text{false}}_i : \texttt{Type}_{i+1} \triangleq (\Pi A : \texttt{Type}_i, A : \texttt{Type}_{\max(i+1,i)})$.

But $\underline{\text{false}}_{\texttt{Prop}} \to^* \Pi A : \texttt{Prop}, A$, of type $\texttt{Prop}$ by impredicativity (and $\texttt{Type}_{\texttt{Prop}+1}$ still).

Dealing with Prop

Let $\underline{\text{false}}_i : \mathtt{Type}_{i+1} \triangleq (\Pi A : \mathtt{Type}_i, A : \mathtt{Type}_{\max(i+1,i)})$.

But $\underline{\text{false}}_{\mathtt{Prop}} \to^* \Pi A : \mathtt{Prop}, A$, of type Prop by impredicativity (and $\mathtt{Type}_{\mathtt{Prop}+1}$ still).

Fact: Cannot handle the implicit $\mathtt{Prop} \leq \mathtt{Type}$ rule and impredicativity precisely and efficiently (models of proof-irrelevance have a similar issue).

Ideal Solution: Use an explicit coercion.
Current Solution: Forbid instantiation of a polymorphic level with Prop. Compatible with an explicit coercion.

Universe declarations

This restriction gives clear semantics for universe declarations:

- A toplevel, global universe i is always $>$ `Set`.
- A local universe in a polymorphic definition is always $\geq$ `Set`. It can get collapsed to `Set` during type inference.
- Naturally enforces the invariant that there is no universe between `Prop` and `Set` (or below `Prop`!).

Explicit Universes

DEMO

Summary

- More functional, trustable implementation.
- User-level control on generated universes and constraints (simplification, declaration...).
- Elaboration/tactics become universe aware (earlier error messages).

The End