Coq Coq Correct!
Verification of Type Checking and Erasure for Coq, in Coq

Matthieu Sozeau
Théo Winterhalter

joint work with
Jakob Botsch Nielsen
Yannick Forster
Simon Boulier
Nicolas Tabareau
Motivation

Coq

CompCert

DeepSpec

4-colour theorem

... 

Verified C Compiler (Executable)

Verified Web Server (Executable)

Verified Colouring Program

...
What do you trust?

Ideal Coq

Implemented Coq

Trusted Core
What do you trust?

- Dependent Type Checker (18kLoC, 30+ years)
- Inductive Families w/ Guard Checking
- Universe Cumulativity and Polymorphism
- ML-style Module System
- KAM, VM and Native Conversion Checkers
- OCaml’s Compiler and Runtime
The Reality

Unspecified
Ideal Coq

Buggy
Implemented Coq
The Reality

- Reference Manual roughly specifies on paper the basic core metatheory. The rest is (at best) in various papers and PhD theses, e.g. module system, treatment of eta-conversion, guard condition, SProp....

- Discrepancies with the actual implementation

- Combination of features not worked-out in detail. E.g. cumulative inductive types + let-bindings in parameters of inductives???
### 354 lines (314 sloc) | 16.7 KB

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Preliminary compilation of critical bugs in stable releases of Coq</td>
</tr>
<tr>
<td>2</td>
<td>WORK IN PROGRESS WITH SEVERAL OPEN QUESTIONS</td>
</tr>
<tr>
<td>6</td>
<td>To add: #7723 (vm_compute universe polymorphism), #7695 (modules and typing constructions)</td>
</tr>
<tr>
<td>10</td>
<td>component: &quot;match&quot; summary: substitution missing in the body of a let introduced: ? impacted released versions: V8.3-V8.3pl2, V8.4-V8.4pl4 impacted development branches: none impacted coqchk versions: ? fixed in: master/trunk/v8.5 (e583a79b5, 22 Nov 2015, Herbelin), v8.5 found by: Herbelin</td>
</tr>
</tbody>
</table>

~ 1 critical bug every year

**The Reality**

**Implemented Coq**

**Trusted Core**
Our Goal: Improving Trust

Ideal Coq

Implemented Coq

Trusted Theory

~ 1 critical bug every year
Coq in MetaCoq

Part I: Coq’s Calculus PCUIC

Part II: Verified Coq

MetaCoq
Formalization of
Coq in Coq
JAR’20

Implemented Coq

Trusted Theory
What we have...

```coq
fix vrev {A : Type@{i}} {n m : nat} (v : vec@{i} A n) (acc : vec@{i} A m) :=
match v in vec _ _ n return vec@{i} A (n + m) with
| vnil ⇒ acc
| vcons a n v' ⇒
  let idx := S n + m in
  coerce (vec A) idx (e : n + S m = idx) (vrev v' (vcons a m acc))
end.
```

vrev_term : term :=
tFix [[
  dname := nNamed "vrev" ;
  dtype := tProd (nNamed "A") (tSort (Universe.make" (Level.Level "Top.160", false) []))
  (tProd (nNamed "n") (tInd[{| inductive_mind := "Coq.Init.Datatypes.nat";
    inductive_ind := 0 |}] []))
  (tProd (nNamed "m") (tInd[...] ...
What we have...

```coq
fix vrev {A : Type[@{i}]} {n m : nat} (v : vec[@{i}] A n) (acc : vec[@{i}] A m) :=
  match v in vec _ n return vec[@{i}] A (n + m) with
  | vnil => acc
  | vcons a n v' =>
    let idx := S n + m in
    coerce (vec A) idx (e : n + S m = idx) (vrev v' (vcons a m acc))
  end.
```
...and what we don’t

(\textbf{fun } x \Rightarrow f \ x) \equiv f \ (x \notin f)

\eta\text{-conversion (WIP)}

\textbf{list \ nat} : \textbf{Set}
\textbf{list \ Type@i} : \textbf{Type@i}

«template» polymorphism

\textbf{Module } M <: S. \textbf{Definition } t := \textbf{nat}. \textbf{End } M.

module system

No existential or named variables (yet)
### Definitions in Contexts

\[
(x : T := t) \in \Gamma \\
\Gamma \vdash x \rightarrow t
\]

### General Substitution

\[
\Gamma \vdash \text{let } x : T := t \text{ in } b \rightarrow b' [x := t]
\]

### Strong Reduction

\[
\Gamma, x : T := t \vdash b \rightarrow b'
\]

\[
\Gamma \vdash \text{let } x : T := t \text{ in } b \rightarrow \text{let } x : T := t \text{ in } b'
\]
Specification
Example: Call-by-Value Evaluation

\[ t \rightarrow_{\text{cbv}} v \quad b[x := v] \rightarrow_{\text{cbv}} v' \]

\[ \text{let } x : T := t \text{ in } b \rightarrow_{\text{cbv}} v' \]

\[ _ \rightarrow_{\text{cbv}} _ \quad \subseteq \quad \varepsilon \vdash _ \rightarrow _ \]
Meta-Theory

Structures

term, t, u ::= 
    | Rel (n : nat)  | Sort (u : universe) | ...

global_env, Σ ::= []
    | Σ , (kername × InductiveDecl idecl)
    | Σ , (kername × ConstantDecl cdecl)

global_env_ext ::= (global_env × universes_decl)

Γ ::= []
    | Γ , aname : term
    | Γ , aname := t : u
Meta-Theory

Judgments

\[ \Sigma ; \Gamma \vdash t \to u, \ t \to^* u \]

\[ \Sigma ; \Gamma \vdash t =_\alpha u, \ t \leq_\alpha u \]

\[ \Sigma ; \Gamma \vdash T = U, \ T \leq U \]

\[ \Sigma ; \Gamma \vdash t : T \]

\[ \text{wf} \ \Sigma, \ \text{wf\_local} \ \Sigma \ \Gamma \]

One-step reduction and its reflexive transitive closure

\( \alpha \)-equivalence + equality or cumulativity of universes

Conversion and cumulativity

\[ \iff T \to^* T' \land U \to^* U' \land T' \leq_\alpha U' \]

Typing

Well-formed global and local environments
Basic Meta-Theory

*Structural Properties*

- Traditional de Bruijn lifting and substitution operations in the spec
- Show that $\sigma$-calculus operations simulate them (à la Autosubst):
  
  $\text{ren} : (\text{nat} \to \text{nat}) \to \text{term} \to \text{term}$
  
  $\text{inst} : (\text{nat} \to \text{term}) \to \text{term} \to \text{term}$

- **Weakening** and **Substitution** from renaming and instantiation theorems
- Easier to lift to strengthening/exchange lemmas in the future
  (strengthening is not immediate here)
Universes

universe ::= Prop | SProp |
    | Type (ne_sorted_list (universe_level * nat)).

Typing  Σ ; Γ ⊢ tSort u : tSort (Universe.super u)
No distinction of algebraic universes (more general than current Coq)

universe_constraint ::=
    universe_level × ℤ × universe_level.  (u + x ≤ v)

Specification  Global set of consistent constraints, satisfy a valuation in ℕ.
  • lSet  always has level 0, smaller than any other universe.
  • Impredicative sorts are separate from the predicative hierarchy.
Universes
Basic Meta-Theory

Global environment weakening
Monotonicity of typing under context extension: universe consistency is monotone.

Universe instantiation
Easy, de Bruijn level encoding of universe variables (no capture)

Implementation
Longest simple paths in the graph generated by the constraints $\phi$, with source $lSet$

$$\forall l, \text{lsp } \phi \ l \ l = 0 \iff \text{satisfiable } \phi \ (\lambda \ l, \text{lsp } lSet \ l)$$
Meta-Theory
The path to subject reduction

Validity
\[ \Sigma ; \Gamma \vdash t : T \]
\[ \Sigma ; \Gamma \vdash t : t\text{Sort } s \]

Context
\[ \Sigma ; \Gamma \vdash t : T \quad \Sigma \vdash \Delta \leq \Gamma \]
\[ \Sigma ; \Delta \vdash t : T \]

Conversion

Subject
\[ \Sigma ; \Gamma \vdash t : T \quad \Sigma ; \Gamma \vdash t \to u \]
\[ \Sigma ; \Gamma \vdash u : T \]

Requires transitivity of conversion/cumulativity

More generally, context cumulativity

Relies on injectivity of product types, a consequence of confluence
Confluence
The traditional way

\[ \Sigma, \Gamma \vdash t \Rightarrow u \]

One-step parallel reduction

À la Tait-Martin-Löf/Takahashi:

Diamond for \( \Rightarrow \)

"Squash" lemma

\[ c \quad \Rightarrow \quad c \]
\[ c \quad \rightarrow \quad c \quad \Rightarrow \quad c \quad \rightarrow^* \quad c \]
Takahashi’s Trick

\[ \rho : \text{term} \rightarrow \text{term} \]

An optimal one-step parallel reduction function.
The triangle property

\(
\begin{array}{c}
\rho(t) \\
\downarrow \\
\end{array}
\begin{array}{c}
t \\
\uparrow \\
\end{array}
\begin{array}{c}
u \\
\end{array}
\end{array}
\)
The triangle property
Confluence
For a theory with definitions in contexts

$$\Sigma \vdash \Gamma, t \Rightarrow \Delta, u$$

One-step parallel reduction, including reduction in contexts.

$$\Sigma \vdash \Gamma, x := t \Rightarrow \Delta, x := t', \Sigma \vdash (\Gamma, x := t), b \Rightarrow (\Delta, x := t'), b'$$

$$\Sigma \vdash \Gamma, (\text{let } x := t \text{ in } b) \Rightarrow \Delta, (\text{let } x := t' \text{ in } b')$$

$$\rho : \text{context} \to \text{term} \to \text{term}$$

$$\text{pctx} : \text{context} \to \text{context}$$
Principality and changing equals for equals

Definition principality \{\Sigma \Gamma t\} : (welltyped \Sigma \Gamma t : Prop) →
\Sigma (P : term), \Sigma ; \Gamma \vdash t : P \times principal_type \Sigma \Gamma t P

\Sigma ; \Gamma \vdash t : T  \quad \Sigma ; \Gamma \vdash u : U
\Sigma \vdash u \leq_{\alpha_{\text{noind}}} t

Informally: (well-typed) smaller terms have more types than larger ones.

Justifies the change tactic up-to cumulativity (excluding inductive type cumulativity).
Cumulativity and Prop

\[ \Sigma ; \Gamma \vdash T \sim U \]

Conversion identifying all predicative universes (hence larger than cumulativity).

\[ \Sigma ; \Gamma \vdash t : T \quad \Sigma ; \Gamma \vdash u : U \]

\[ \Sigma \vdash u \leq_\alpha t \]

\[ \Sigma ; \Gamma \vdash T \sim U \]

Informally: for two well-typed terms, if they are syntactically equal up-to cumulativity of inductive types, then they live in the same hierarchy (Prop, SProp or Type).

Required for erasure correctness
Trusted Theory Base

Assumptions

‣ The specifications of typing, reduction and cumulativity
~ 500 LoC from scratch (verified and testable)

‣ Guard Conditions. Oracles:
  check_fix : global_env → context → fixpoint → bool
  + preservation by renaming/instantiation/equality/reduction

‣ Strong Normalization (not provable thanks to Gödel, but also not used in the preceding results). Consistency and canonicity follow easily.

Axiom normalisation :
  ∀ Γ t, welltyped Σ Γ t → Acc (cored (fst Σ) Γ) t.
Verifying Type-Checking
Conversion

Objective

Input

\[ u : A \quad v : B \]

Output

\[ (u \equiv v) + (u \neq v) \]
Conversion

Objective

Input

\[ u : A \]

\[ v : B \]

Output

\[ (u \equiv v) + (u \not\equiv v) \]

\[
isconv : \forall \Sigma \Gamma (u \; v \; A \; B : \text{term}),
\]

\[ (\Sigma ; \Gamma \vdash u : A) \]

\[ (\Sigma ; \Gamma \vdash v : B) \]

\[ (\Sigma ; \Gamma \vdash u \equiv v) + \]

\[ (\Sigma ; \Gamma \vdash u \equiv v \bot) \]
Conversion

Algorithm

\[ u : A \xrightarrow{\text{whnf}} u' \]

\[ v' \xleftarrow{\text{whnf}} v : B \]
Conversion

Algorithm

\[ u : A \]

\[ u' \equiv v' \]

\[ v : B \]
Conversion

Algorithm

\[ u : A \]

\[ v : B \]

\[ u' \equiv \_ \]

\[ v' \equiv \_ \]

\[ \text{match} \ u' , \ v' \]

\[ \lambda (x : A_1). t_1 \]

\[ \lambda (x : A_2). t_2 \]

\[ \equiv \]

\[ A_1 \]

\[ A_2 \]

\[ B_1 \]

\[ B_2 \]

\[ \text{match} \ u' , \ v' \]

\[ \Pi (x : A_1). B_1 \]

\[ \Pi (x : A_2). B_2 \]

\[ \equiv \]

\[ A_1 \]

\[ A_2 \]

\[ B_1 \]

\[ B_2 \]
Conversion

Completeness

\[
\lambda(x:A_1). t_1 \quad \lambda(x:A_2). t_2 \Rightarrow A_1 \equiv A_2 \land B_1 \equiv B_2
\]

\[
\Pi(x:A_1). B_1 \quad \Pi(x:A_2). B_2 \Rightarrow A_1 \equiv A_2 \land B_1 \equiv B_2
\]
Conversion
Completeness

\[ \prod(x: A_1). B_1 \equiv \prod(x: A_2). B_2 \Rightarrow A_1 \neq A_2 \]
Conversion

Completeness

we conclude

\[ \Pi(x:A_1). B_1 \not\equiv \Pi(x:A_2). B_2 \]

using inversion lemmata and confluence
Conversion

\[
\begin{align*}
\lambda(x:A_1). t_1, &\quad \lambda(x:A_2). t_2 \Rightarrow A_1 \\
\Pi(x:A_1). B_1, &\quad \Pi(x:A_2). B_2 \Rightarrow A_1
\end{align*}
\]

\[
\begin{align*}
\lambda(x:A_1). t_1, &\quad \lambda(x:A_2). t_2 \Rightarrow A_1 \\
\Pi(x:A_1). B_1, &\quad \Pi(x:A_2). B_2 \Rightarrow A_1
\end{align*}
\]

\[
\begin{align*}
\lambda(x:A_1). t_1, &\quad \lambda(x:A_2). t_2 \Rightarrow A_1 \\
\Pi(x:A_1). B_1, &\quad \Pi(x:A_2). B_2 \Rightarrow A_1
\end{align*}
\]

\[
\begin{align*}
\lambda(x:A_1). t_1, &\quad \lambda(x:A_2). t_2 \Rightarrow A_1 \\
\Pi(x:A_1). B_1, &\quad \Pi(x:A_2). B_2 \Rightarrow A_1
\end{align*}
\]
Weak head reduction

Objective

Input:
- \( u \) term

Output:
- \( v \) term
Weak head reduction

Objective

Input

\(u\)  
term

Output

\(v\)  
\(u \, ? \, v\)

\(v\)  
term

Prop
Weak head reduction

Objective

Input

\[ u \]

term

Output

\[ v \]

\[ u \text{ ? } v \]

term

Prop

\[
\text{weak}_\text{head}_\text{reduce} : \forall (u : \text{term}), \sum (v : \text{term}), u \text{ ? } v
\]
Weak head reduction

Example

\[ \text{Definition } \text{foo} := \lambda(x:\text{nat}). \, x. \]

foo 0
Definition \( \text{foo} := \lambda(x: \text{nat}). x \).
Weak head reduction

Example

Definition foo := \(\lambda(x:\text{nat}). \ x\).

\[\lambda(x:\text{nat}).x\theta\]

foo \rightarrow \lambda(x:\text{nat}).x
Weak head reduction

Example

Definition $\text{foo} := \lambda(x:\text{nat}).\ x$. 

Input $u$  

Output $v$ $u$ ?

$0$

$\text{foo} \rightarrow \lambda(x:\text{nat}).x$
Weak head reduction

Example

Definition foo := \(\lambda(x:\text{nat}).x\).

foo 0 \rightarrow (\lambda(x:\text{nat}).x) 0 \rightarrow 0
Weak head reduction

Termination

Input

u

Output

v

u ? v
Weak head reduction

Termination

Input

\( u \pi_1 \)

Output

\( v \pi_2 \)

\( u \quad ? \quad v \)
Weak head reduction

Termination

Input

Output

\[ u \quad \pi_1 \quad \rightarrow \quad v \quad \pi_2 \]
Weak head reduction

Termination

\[ \text{foo 0} \quad \text{foo 0} \quad \lambda(x: \text{nat}).x \quad 0 \quad 0 \]
Weak head reduction

Termination

\[ \text{foo 0} \]

\[ \lambda(x:\text{nat}).x \text{ 0} \]

\[ (\lambda(x:\text{nat}).x) \text{ 0} \rightarrow \text{ 0} \]
Weak head reduction

Termination

\[ \text{foo } 0 \rightarrow (\lambda(x:\text{nat}).x) \ 0 \]

\[ (\lambda(x:\text{nat}).x) \ 0 \rightarrow 0 \]
Weak head reduction

Termination

foo 0 → (λ(x:nat).x) 0

foo 0 → foo 0

λ(x:nat).x 0 → 0

foo 0 ⇐ foo

(λ(x:nat).x) 0 → 0
Weak head reduction

Termination

foo 0 \to (\lambda(x:\text{nat}).x) 0

foo 0 \to foo

foo 0 \to (\lambda(x:\text{nat}).x) 0 \to 0

Lexicographic order of \[ and \{

1
Weak head reduction

Termination

\[ \text{foo} \ 0 \rightarrow (\lambda(x: \text{nat}).x) \ 0 \]

\[ \text{foo} \ 0 \rightarrow (\lambda(x: \text{nat}).x) \ 0 \rightarrow 0 \]

Lexicographic order of ?? and □
Weak head reduction

Termination

Lexicographic order of and ⊏
Weak head reduction
Termination

Lexicographic order of \( ? \) and \( \sqsubseteq \)
Weak head reduction

Termination

but \( p.1 \neq p \)

Lexicographic order of \(?\) and \(\sqsubset\)
Weak head reduction

Termination

\[ p.1 > p.1 \]

and \( p.1 = p.1 \)

Lexicographic order of \(?\) and \(\sqsubseteq\)
Weak head reduction

Termination

fix f (n:nat). t end n

Lexicographic order of ? and ⊏
Weak head reduction

Termination

\[ \text{fix } f \ (n:\text{nat}) \ . \ t \ \text{end } n \]

Lexicographic order of \(?\) and \(\sqsubseteq\)
Weak head reduction

Termination

```latex
\texttt{fix } f \texttt{ (n:nat). t end n}
```

Lexicographic order of [?] and ⊏
Weak head reduction

Termination

\[
\text{fix } f \ (n:\text{nat}). \ t \ \text{end} \ n
\]
Weak head reduction

Termination

Lexicographic order of \(\text{f n}\) and \(\text{f n}\)
Weak head reduction

Termination

Lexicographic order of $f_n$ and an order on positions
Weak head reduction

Termination

Lexicographic order of \( f_n \) and an order on positions
Weak head reduction

Termination

Lexicographic order of $u\pi_1$ and an order on positions
Weak head reduction

Termination

\[ \langle u \pi_1, \text{stack\_pos } u \pi_1 \rangle > \langle v \pi_2, \text{stack\_pos } v \pi_2 \rangle \]

\[ \text{pos (u } \pi_1) \]

\[ \text{pos (v } \pi_2) \]

Lexicographic order of ?? and an order on positions
Weak head reduction

Termination

\[
\langle u \pi_1, \text{stack_pos } u \pi_1 \rangle > \langle v \pi_2, \text{stack_pos } v \pi_2 \rangle
\]

pos (u \pi_1)

pos (v \pi_2)

Dependent lexicographic order of \(\pi\) and an order on positions
Type Checking

Weak head reduction

Conversion
Type Checking

- Weak head reduction
- Cumulativity
- Inference
Type Checking

Inference

Cumulativity

Weak head reduction

Infer $t$ → Check $B \leq A$

Check $t : A$ →

Check $B \leq A$
Type Checking

Weak head reduction

Cumulativity

Inference

Infer $t : B$ → Check $B \leq A$

Check $t : A$

MetaCoq Check foo.

ITP’21 Bidirectional type checking for completeness
A little success story

Spec/Proof/Program co-design for the new `match` representation in Coq (PR #13563 by P.M. Pédrot, CEP #34 by H. Herbelin).

‣ MetaCoq => typechecking of case on cumulative inductive types is incomplete

‣ Failure of subject reduction in Coq.

‣ “Quick” fix requires strengthening which in turn is not provable without subject reduction, leading to a messy meta-theory. Also incompatible with eta-conversion.

‣ The new representation solves all these issues and reflects the high-level user syntax more faithfully. It’s win/win/win!
Benefits of Bidirectional Type-Checking

- MetaCoq => typechecking of case on cumulative inductive types is incomplete

- Failure of subject reduction in Coq.

- “Quick” fix requires strengthening which in turn is not provable without subject reduction, leading to a messy meta-theory. Also incompatible with eta-conversion.

- The new representation solves all these issues and reflects the high-level user syntax more faithfully. It’s win/win/win!
Verifying Erasure
Erasure

At the core of the **extraction** mechanism:

\[ \varepsilon : \text{term} \rightarrow \Lambda^{\square, \text{match}, \text{fix}, \text{cofix}} \]

Erases non-computational content:

- **Type erasure:**
  \[ \varepsilon (t : \text{Type}) = \square \]

- **Proof erasure:**
  \[ \varepsilon (p : P : \text{Prop}) = \square \]
Erasure

Singleton elimination principle

Erase propositional content used in computational content:

\[ \varepsilon \left( \text{match } p \text{ in } \text{eq } _\_ y \text{ with } \text{eq_refl } \Rightarrow b \text{ end} \right) = \varepsilon \left( b \right) \]

---

Definition coerce \{A\} \{B : A \to \text{Type}\} \{x\} \{y : A\}
\( (e : x = y) : P x \to P y :\)
match e with
| eq_refl \( \Rightarrow \) fun p => p
end.

fix vrev n m v acc :=
match v with
| vnil \( \Rightarrow \) acc
| vcons a n v' \( \Rightarrow \)
  let idx := S n + m in
  coerce \( \square \) idx \( \square \) (vrev v' (vcons a m acc))
end.
Erasure

Singleton elimination principle

Erase propositional content used in computational content:

$$\varepsilon \left( \text{match } p \text{ in } \text{eq } _{-} y \text{ with } \text{eq_refl } \Rightarrow b \text{ end} \right) = \varepsilon \left( b \right)$$

$$\varepsilon \left( \text{coerce} \right) \sim \text{coerce} x \ y := \left( \text{fun } p \Rightarrow p \right)$$

$$\varepsilon \left( \text{vrev} \right) \sim \text{fix vrev } n \ m \ v \ \text{acc} :=$$

$$\begin{align*}
\text{match } v \text{ with} \\
| \text{vnil} & \Rightarrow \text{acc} \\
| \text{vcons } a \ n \ v' & \Rightarrow \text{vrev } v' \left( \text{vcons } a \ m \ \text{acc} \right)
\end{align*}$$

end.
Erasure Correctness

With Canonicity and SN:

\[ \vdash t : \text{nat} \]
\[ \Rightarrow \vdash t \rightarrow n : \text{nat} \quad (n \in \mathbb{N}) \]
\[ \Rightarrow t \rightarrow_{\text{cbv}} n : \text{nat} \]
\[ \Rightarrow \varepsilon (t) \rightarrow_{\text{cbv}} n \]
Erasure Correctness

First define a non-deterministic erasure relation, then define:

\[ \varepsilon : \forall \Sigma \Gamma t \ (\text{wt} : \text{welltyped } \Sigma \Gamma t) \rightarrow \text{EAST.term} \]

Finally show that \( \varepsilon \)'s graph is in the erasure relation. Two additional optimizations:

- Remove trivial cases on singleton inductive types in Prop
- Compute the dependencies of the erased term to erase only the computationally relevant subset of the global environment. I.e. remove unnecessary proofs the original term depended on.
MetaCoq

Ideal Coq

Verified Coq

in

MetaCoq

in

Trusted Core

Implemented Coq
Summary

- **Verified Coq**
  - MetaCoq Check \( vrev \).

- **MetaCoq**('in') **in** **Verified Core**
  - Implemented Coq = Ideal Coq

- **Spec**: 30kLoC
- **Proofs**: 60kLoC
- **Comments**: 10kLoC
Perspectives

CompCert

CertiCoq

... 

Models of PCUIC

Verified C Compiler

Verified Coq Compiler

Verified Extraction to OCaml
Ongoing and future work

‣ Integration of rewrite rules (CEP #50)

‣ Interoperability of erased code with OCaml (Nomadic Labs CoqExtra project, Pierre Giraud’s PhD thesis)

‣ Full meta-theory for the SProp sort and irrelevance checking

‣ Eta-reduction and contravariant subtyping (CEP #47)

‣ Integration of a sort-polymorphism system, generalising universe polymorphism to deal more uniformly with impredicative sorts and alternative hierarchies (exceptional type theory, setoid type theory, erasable sets...) (Kenji Maillard).
Coq in MetaCoq

« Cot Cot Codet ». French, Interjection.

1. Cackle (the cry of a hen, especially one that has laid an egg).
Related Work

- Kumar et al., HOL + CakeML (JAR’16)
- Strub et al., Self-Certification of F* starting with Coq (POPL’12)
- Rahli and Anand, NuPRL in Coq (ITP’14)