The Curious Case of Case
Correct & efficient representation of case analysis in Coq and MetaCoq

Matthieu Sozeau
Meven Lennon-Bertrand
Yannick Forster
Inria & Université de Nantes
Our Goal: Improving Trust

- Ideal Coq: Trusted Theory
- Implemented Coq: ~1 critical bug every year
The MetaCoq Project

Trusted Theory

Core Calculus: PCUIC

Equivalence

Coq's Calculus: Template-Coq

Correctness and Completeness

Verified Type Checker

Sound Erasure

λ☐

MetaCoq

CertiCoq

ConCert
A little success story

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

\[ \Sigma ; \Gamma \vdash p : \text{eq}^{i'} A' t y \]
\[ \Sigma ; \Gamma \vdash \text{fun} y e \Rightarrow P : (\forall y, \text{eq}^{i} A t y \rightarrow \text{Type}) \]
\[ \Sigma ; \Gamma \vdash b : P t \text{eq_refl} \]
\[ i' = i \quad A \leq A' \]

\[ \Sigma ; \Gamma \vdash \text{match} p \text{ as } e \text{ in } \text{eq } _ _ y \text{ return } P y e \text{ with} \]
\[ | \text{eq_refl } \Rightarrow b \text{ end } : P y p \]

Confusion in the kernel:
\( (\text{fun } (y : A) (e : \text{eq}^{i} A x y) \Rightarrow P y e) \neq (\text{fun } (y : A') (e : \text{eq}^{i'} A' x y) \Rightarrow P y e) \)
A little success story

- MetaCoq bidirectional typing completeness proof failure =>
  typechecking of case on cumulative inductive types is incomplete

- Subject reduction failure in Coq (Coq issue #13495)

- Quick & dirty fix requires strengthening, not provable by induction

\[ \Sigma ; \Gamma \vdash _ : \text{forall} (y : A), \text{eq@\{i\} } A t y \rightarrow Type \]

does not imply \[ \Sigma ; \Gamma \vdash t : A \]
without strengthening

(easy to forget when you’re just implementing!)
A little success story

\[
\Sigma ; \Gamma \vdash A : \text{Type} \quad \Sigma ; \Gamma \vdash t : A \\
\Sigma ; \Gamma, x : A, e : \text{eq@\{i\} A t x} \vdash P : \text{Type} \\
\Sigma ; \Gamma \vdash p : \text{eq@\{i\} A t u} \\
\Sigma ; \Gamma \vdash b : P[t/x] \text{ eq_refl}
\]

\[
\Sigma ; \Gamma \vdash \text{match p as e in eq@\{i\} A t x return P with} \\
| \text{eq_refl} => b \text{ end : P u p}
\]

- Case carries parameters \((A, t)\) and a universe instance \((i)\) separately + bindings names for the context. Derivation of the predicate and branch contexts on the fly.

- Completeness holds, reflects the high-level user syntax more closely

- Keeps cumulativity orthogonal to this rule

- Clear information flow in the bidirectional version
Bidirectional Type-Checking for the Win!

- Bidirectional derivations are syntax directed: Compressed and localised conversion rules.
- Trivialises correctness and completeness of type inference
- Principality follows from correctness and completeness of bidirectional typing w.r.t. "undirected" typing
- Completeness proof requires injectivity of type constructors
- Correctness proof requires transitivity of conversion
- Strengthening follows directly
The impact on MetaCoq

- Due to let-bindings in contexts, we must carry the “canonical context” of predicates and branches in PCUIC to still have well-behaved recursive definitions of renaming/substitution etc.

- This in turn requires to consider only well-scoped terms for the theory of reduction which was previously valid even on ill-scoped terms.

- Equivalence with Coq’s theory where only binding names are carried => reason on the wellformedness of the global environment from which the canonical context is built.

- Non-trivial translation before erasure to expand let-bindings in branches.
Takeaways

‣ Always define a bidirectional version of your system: it will keep you honest

‣ “Minor” implementation changes can rely on subtle assumptions about the theory or require non-trivial reorganisations of the metatheory: they can no longer go unnoticed if you formalise it!

‣ The overhead of formalising is high but can catch bugs earlier. The initial investment pays off when we want to extend the language (c.f. Meven Lennon-Bertrand’s talk on eta-conversion)