MetaCoq Quotation

Partial Work Towards Löb's Theorem

Jason Gross

Context: Löb's Theorem

$$\Box(\Box X \to X) \to \Box X$$

□“X” := { t : Ast.term & Σ ;;; [] |- t : <% X %> }

“If you can prove that X is true whenever X is provable, then you can prove X”

Context: Löb's Theorem: Most Challenging Building Blocks

Decidable Equality of ASTs: $\forall\ (x\ y : \square X), \{x = y\} + \{x \neq y\}$

Quotation: $\square A \to \square\square A$

Everything else* is comparatively straightforward;
details (and Agda formalization) available upon request.

**With the possible exception of the equational reduction laws about quotation*

□“X” := { t : Ast.term & Σ ;;; [] |- t : <% X %> }
$\square(\square X \to X) \to \square X$

MetaCoq Quotation

□“X” := { t : Ast.term & Σ ;;; [] |- t : <% X %> }

Quotation: □A → □□A   (*aka cojoin of the lax monoidal semicomonad* □)

cojoin a := (tApp <% existT %> [quote_term a.1; quote_typing a.2];

quote_existT_typing quote_term_well_typed quote_typing_well_typed)

quote_term   : Ast.term → Ast.term

quote_typing   : (Σ ;;; Γ |- t : T) → Ast.term

quote_term_well_typed   : (Σ ;;; Γ |- t : T) → Σ ;;; [] |- quote_term t : <% Ast.term %>

quote_typing_well_typed   : ∀ (pf : Σ ;;; Γ |- t : T),

Σ ;;; [] |- quote_typing pf : tApp <% typing %> [quote Σ; quote Γ; quote t; quote T]

□“X” := { t : Ast.term & Σ ;;; [] |- t : <% X %> }

□(□X → X) → □X

MetaCoq Quotation: Code Walkthrough

https://github.com/MetaCoq/metacoq/tree/main/quotation/theories#readme

Code Stats:

≈ 8 kloc

wakatime 220 hrs 24 mins

Please ask questions and tell me what you want to see!

MetaCoq Quotation: Future Work

□“X” := { t : Ast.term & Σ ;;; [] |- t : <% X %> }

Quotation: □A → □□A   (*aka cojoin of the lax monoidal semicomonad* □)

cojoin a := (tApp <% existT %> [quote_term a.1; quote_typing a.2];
    quote_existT_typing quote_term_well_typed quote_typing_well_typed)

quote_term   : Ast.term → Ast.term

quote_typing   : (Σ ;;; Γ |- t : T) → Ast.term

quote_term_well_typed   : (Σ ;;; Γ |- t : T) → Σ ;;; [] |- quote_term t : <% Ast.term %>

quote_typing_well_typed   : ∀ (j : Σ ;;; Γ |- t : T),
    Σ ;;; [] |- quote_typing j : tApp <% typing %> [quote Σ; quote Γ; quote t; quote T]

□“X” := { t : Ast.term & Σ ;;; [] |- t : <% X %> }
□(□X → X) → □X

MetaCoq Quotation: Future Work: Anticipated Major Work

1. Universe polymorphism design
2. Producing unsquashed typing derivations with a safechecker variant
   (but Théo Winterhalter indicated on Zulip he might be doing this)
3. Safechecker work deduplication: abstracting over Gallina context variables
4. Using the same axioms in the metatheory and object theory
   (also the same inductives and environment definitions)
5. Performance scaling with size of term?
   (we need to be able to quote and safecheck cojoin (well-typed quotation) itself)

Partial work at JasonGross/metacoq@coq-8.16+quotation-typing

Context: Löb's Theorem: Building Blocks I

□“X” := { t : Ast.term & Σ ;;; [] |- t : <% X %> }

I. *Syntax forms a Cartesian Category:*

id	$: \square(A \to A)$
_;_	$: \square(A \to B) \to \square(B \to C) \to \square(A \to C)$
dup	$: \square(A \to A \times A)$
×map	$: \square(A \to X) \to \square(B \to Y) \to \square(A \times B \to X \times Y)$
qtt	$: \square\mathrm{unit}$

N.B. With a bit of extra work, we could also strip the outer box off of all of these, and interpret the “→” outside of the outer box to mean ≈ tactic function

Context: Löb's Theorem: Building Blocks II

□“X” := { t : Ast.term & Σ ;;; [] |- t : <% X %> }

I. *Syntax forms a Cartesian Category (id, compose, ×, 1)*
II. *Diagonal lemma premises*

S := <% { T : Type & T } %>

ϕ : □((S × □S) → □“Type”) (*by decidable equality with* = (*“□S → Type”; _*))

ϕ^{-1} : □(□S → “Type”) → □S (*straightforward*)

(N.B. There might be minor mistakes in translation from Agda here, I haven’t fully worked out the details)

Context: Löb's Theorem: Building Blocks III

□“X” := { t : Ast.term & Σ ;;; [] |- t : <% X %> }

I. *Syntax forms a Cartesian Category (id, compose, ×, 1)*
II. *Diagonal lemma premises (built from decidable equality of ASTs)*
III. *Löbian premises*

S := Δ(□S → X) or Δ(□(S → X)) or Δ(□S → □X), depending on proof

ϕ : □((S × □S) → □X) (*from diagonal lemma*)

ϕ^{-1} : □(□S → X) → □S (*from diagonal lemma*)

□(□X → X) → □X

Context: Löb's Theorem: Building Blocks IV

□“X” := { t : Ast.term & Σ ;;; [] |- t : <% X %> }

I. *Syntax forms a Cartesian Category (id, compose, ×, 1)*
II. *Diagonal lemma premises (built from decidable equality of ASTs)*
III. *Löbian premises (from the diagonal lemma mostly)*
IV. □ *is a lax monoidal semicomonad*

□-map	: □(A → B) → □(□A → □B)	(aka quote + function application)
□×-codistr	: □((□ A × □ B) → □(A × B))	(aka <% <% pair %> %>)
□1-codistr	: □(unit → □unit)	(aka <% <% tt %> %>)
cojoin	: □(□A → □□A)	

□(□X → X) → □X

Scratchpad

$$X : \square\text{“Type”}$$

$$\square(\square X \to X) \to \square X$$

$$\square(\square\square\text{“False”} \to \square\text{“False”}) \to \square\square\text{“False”}$$

Can’t get $\square\square A \to \square A$