

$\top$ $\Sigma_{x:A} B(x)$ $\text{Eq}(A, a, b)$ $A/\top$

A Canonical Normal Form Theorem for the Type Theory of Regular Categories

Riccardo Borsetto, University of Verona

SSTT 2026 · Ljubljana

Internal languages turn diagrams into syntax

Syntax/semantics (*Sabelli and Maietti, 2026*)

Category $\mathcal{C}$ on one side; formal theory $\text{Int}(\mathcal{C})$ on the other.

$$\mathcal{C} \models \varphi \iff \text{Int}(\mathcal{C}) \vdash \varphi$$

Why use it?

1. Translate a categorical problem into syntax.
2. Reason by derivations.
3. Translate the result back.

Maietti's hierarchy locates the regular level

Modular correspondence (*Maietti, 2005*)

Categorical structure $\leftrightarrow$ extensional dependent type theory; universal properties become type constructors.

Synt(T) category presented by a theory T

Int($\mathcal{C}$) internal type theory of a category $\mathcal{C}$

$\mathcal{C} \simeq \text{Synt}(\text{Int}(\mathcal{C}))$ and $T \simeq \text{Int}(\text{Synt}(T))$, *level by level*.

$$\begin{array}{ccccccc} \mathbf{lex} & \subset & \mathbf{regular} & \subset & \mathbf{lexensive} & \subset & \mathbf{pretopoi} & \subset & \mathbf{topoi} \\ \top, \Sigma, \text{Eq} & & +A/\top & & +\perp, \text{sums} & & + \text{quotients} & & +\Pi, \Omega \end{array}$$

The regular structure reads as type formers

Calculus	Regular category
context / closed type	object
dependent type	object over the context
dependent mono-type	subobject over the context
term	section
$\top$	terminal object
Σ	composition of context maps
Eq	equaliser
$A/\top$	stable image of $A \rightarrow \top$

The syntax has four judgement forms

$$\Gamma \vdash A \text{ type}$$
$$\Gamma \vdash A = B$$
$$\Gamma \vdash a : A$$
$$\Gamma \vdash a =_A b$$

Equality reflection

$$\frac{\Gamma \vdash p : \text{Eq}(A, a, b)}{\Gamma \vdash a =_A b}$$

Beyond structural context rules (weakening and substitution admissible), each type former has formation, introduction, elimination, congruence, computation and corresponding equality rules.

The canonical normal form theorem

Theorem

Every derivable closed judgement has a canonical form with a canonical proof: normal at the end, formation/introduction last.

Closed type canonical forms have head $\top$, $\Sigma_{x:A} B(x)$, $\text{Eq}(A, a, b)$ or $A/\top$; equality judgements normalise to the equality rule for that head.

canonical type head

$\top$

$\Sigma_{x:A} B(x)$

$\text{Eq}(A, a, a)$

$A/\top$

closed term computes to

$t \Rightarrow \star$

$t \Rightarrow \langle a, b \rangle$

$t \Rightarrow \text{refl}$

$t \Rightarrow [a]$

Canonical forms give three consequences

Introductory form

Example: if the canonical term is $\langle a, b \rangle$, its type has canonical head Σ .

Canonicity

For a closed $t : A$:

- if A has head $\top$, then $t \Rightarrow \star$
- if A has head Σ , then $t \Rightarrow \langle a, b \rangle$
- if A has head Eq , then $t \Rightarrow \text{refl}$
- if A has head $/\top$, then $t \Rightarrow [a]$

Consistency

No-confusion principle:
closed type equality
normalises, so distinct
canonical heads cannot
be judgementally equal.
For example, $\top$ cannot
equal a canonical Σ type.

The computability theorem

What $\text{Comp}(J)$ records

Comp is mutual over the four judgement forms. Its clauses package:

- associated derivable judgements;
- evaluation to a normal form;
- correctness of that evaluation;
- computability of canonical parts;
- stability under computable substitutions.

Fundamental theorem

Every derivable judgement is computable:

$$\mid \text{Deriv}(J) \Rightarrow \text{Comp}(J)$$

- Closed case: terms evaluate.
- Open case: for $\sigma \models \Gamma$ assigning closed computable terms, $\Gamma \vdash a : A$ implies $\text{Comp}(\vdash a(\sigma) : A(\sigma))$.

Closed computability reads off canonical forms

Closed-term clauses

- $\text{Comp}(\vdash t : \top)$ iff $t \Rightarrow \star$
- $\text{Comp}(\vdash t : \Sigma_{x:A} B(x))$ iff $t \Rightarrow \langle a, b \rangle$ with $\text{Comp}(\vdash a : A)$ and $\text{Comp}(\vdash b : B(a))$
- $\text{Comp}(\vdash t : \text{Eq}(A, a, b))$ iff $t \Rightarrow \text{refl}$ with $\text{Comp}(\vdash a =_A b)$
- $\text{Comp}(\vdash t : A/\top)$ iff $t \Rightarrow [a]$ for some $\text{Comp}(\vdash a : A)$

Extraction

For a closed term judgement, $\text{Comp}(\vdash t : A)$ unfolds to evaluation at the head constructor of A .

$$\text{Deriv}(J) \Rightarrow \text{Comp}(J) \Rightarrow (t \Rightarrow \text{canonical})$$

The new case is the terminal quotient

Introduction

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash [a] : A/\top}$$

Computability follows the representative:

$$\frac{\text{Comp}(\Gamma \vdash a : A)}{\text{Comp}(\Gamma \vdash [a] : A/\top)}$$

Equality introduction

The quotient identifies any two representatives:

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : A}{\Gamma \vdash [a] =_{A/\top} [b]}$$

$A/\top$ remembers that a representative exists, not which one.

Quotient elimination needs coherence

The eliminator El_Q has a target family C , a representative case l , an argument p , and a coherence premise:

$$\frac{\begin{array}{c} \Gamma, z : A/\top \vdash C(z) \text{ type} \\ \Gamma \vdash p : A/\top \\ \Gamma, a : A \vdash l(a) : C([a]) \\ \Gamma, a : A, b : A \vdash l(a) =_{C([a])} l(b) \end{array}}{\Gamma \vdash \text{El}_Q(l, p) : C(p)}$$

The last premise is the new burden: use $[a] =_{A/\top} [b]$ to transport $l(a)$ to the fibre $C([b])$, then prove $l(a) =_{C([b])} l(b)$.

The direct Agda proof exposes a cycle

Direct encoding in standard Agda

- Previous attempts: old Rocq/Coq partial implementation and a Cubical Agda RAM-hungry non-terminating one
- Comp is presented as an inductive family of derivable judgements.
- Preservation of Comp under substitution is proven by recursion on derivations, with a syntactic size measure.
- The cycle does not convince Agda's termination checker.

$$\Gamma \vdash t : A \quad \wedge \quad \sigma \models \Gamma \quad \Rightarrow \quad \text{Comp}(\vdash t(\sigma) : A(\sigma))$$

$\top$, Σ and Eq go through directly; only El_Q makes the block genuinely circular.

Where El_Q creates the loop

Representative independence

If the quotient input is represented by a or by b , the representative case must give the same result after transport:

$$l(a) =_{C([a])} l(b)$$

After substitution, proving this requires substituting representatives into both the target family C and the representative case l .

Computable environments

The substitution lemma is stated using computable environments:

$$\sigma \models \Gamma \Rightarrow \text{substitute into } C, l$$

But $\sigma \models \Gamma$ is not a smaller premise of the El_Q rule. The direct proof therefore calls back into the same mutual block.

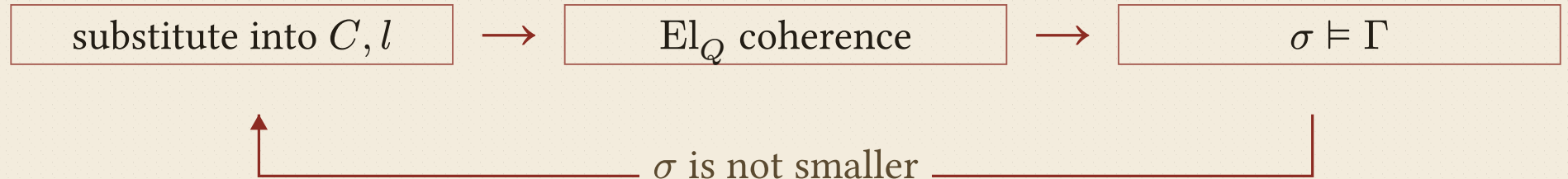
The cycle mixes two measures

No single measure decreases around the loop

The Σ -family clause substitutes into a fibre, so it wants a type-depth decrease:

$$\text{Comp}(\vdash t : \Sigma_{x:A} B(x)) \Rightarrow \text{Comp}(\vdash a : A) \wedge \text{Comp}(\vdash b : B(a))$$

Ordinary eliminators want derivation-size decrease. El_Q needs both, and the environment edge does not decrease in the right component.



Syntactic measures still miss type structure

Three failed repairs

1. Refactoring helper lemmas does not remove the recursive cycle.
2. Isolating the Σ -family substitution fixes positivity only.
3. The remaining call is smaller by type depth, not by derivation size.

The direct proof terminates mathematically, but the decrease is hidden in semantic type structure, so Agda's termination checker does not accept it.

Computability now recurses on type structure

Reverse the indexing

The final proof replaces the inductive family Comp by a recursive function comp on type structure: for each closed type A , a predicate on closed terms of A .

$$\text{comp}_{A(t)} \quad := \quad \text{comp}(\vdash t : A)$$

No positivity obligation

A function has no positivity check, so the Σ clause may quantify over computability values freely:

$$\text{comp}(\vdash t : \Sigma_{x:A} B(x)) \quad \Leftrightarrow \quad \exists ab. t \Rightarrow \langle a, b \rangle \wedge \text{comp}(\vdash a : A) \wedge \text{comp}(\vdash b : B(a))$$

Each unfolding shrinks the type depth

Type structure recursive

Each unfolding strictly reduces type depth:

- $d(A) < d(\Sigma_{x:A} B(x))$
- $d(B(a)) = d(B) < d(\Sigma_{x:A} B(x))$
- substitution preserves the relevant depth bounds.

Well-founded recursion

Defined by well-founded recursion on type depth. The choice of accessibility proof does not affect the result; it is handled by computation, so no function-extensionality axiom is needed.

Computability must respect type formers

Computable equality

Closed terms are only one component:

$$\text{comp}(\vdash t : A)$$

The full theorem also needs equality:

- $\text{comp}(\vdash A = C)$
- $\text{comp}(\vdash t =_A u)$

Where equality is used

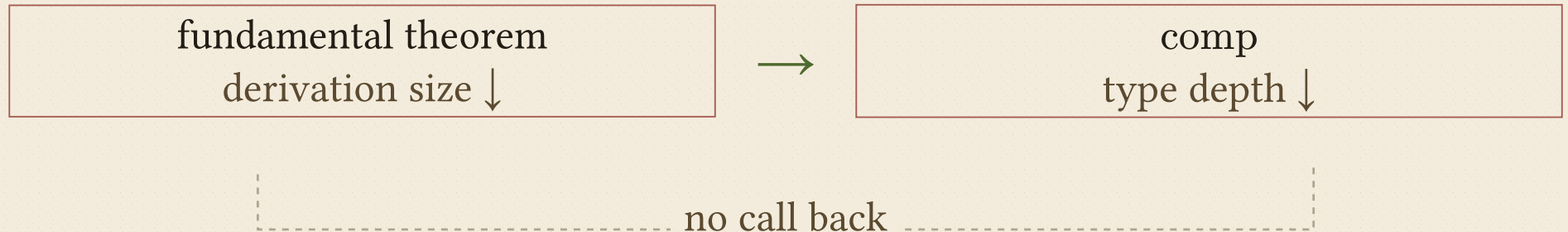
- Type conversion uses computable type equality.
- Eq: by reflection, computability of $p : \text{Eq}(A, a, b)$ gives computable equality of a and b : $\text{comp}(\vdash a =_A b)$; the proof normal form is refl.
- El_Q : after retyping $C([a]) = C([b])$, coherence asks for:

$$\text{comp}(\vdash l(a) =_{C([b])} l(b))$$

The computability theorem no longer loops

Two definitions, two measures

- comp is a function on type structure, with its own recursion on type depth.
- The fundamental theorem recurses on derivation size and only calls comp.
- comp never calls back, so there is no shared block to measure.
- This separation is what Agda's termination checker accepts.



Current work and longer-term directions

Exact and pretopos steps

- Lextensive sums add $\perp$ and disjoint finite sums; Comp splits on ι_1 and ι_2 .
- Full quotients replace $A/\top$ by A/R ; use witness judgements:

$$\Gamma \vdash r : R(a, b)$$

Topos steps

- Π : computable functions preserve computable inputs.
- Ω : predicates are classified by canonical proposition codes. Code canonicity is the hard part.

Longer term

$(\infty, 1)$ -categories: compare with cubical/HoTT canonicity and canonical forms; see other talks here at SSTT.

References and code

Repository `github.com/eapiova/regcat-normal-form`

References

- [1] M. E. Maietti. *Modular correspondence between dependent type theories and categories including pretopoi and topoi*. *Math. Struct. Comp. Sci.* 15(6):1089-1149, 2005.
- [2] P. Sabelli and M. E. Maietti. *The Internal Language of Quasi-toposes*. Slides, 112th PSSL, Nottingham, 2026.
- [3] S. Valentini. *Meta-mathematical aspects of Martin-Löf's type theory*. PhD thesis, Nijmegen, 2000.
- [4] W. W. Tait. *Intensional interpretations of functionals of finite type I*. *J. Symb. Logic* 32(2):198-212, 1967.