For Generalised Algebraic Theories, Two Sorts Are Enough

Niyousha Najmaei, École Polytechnique
with **Samy Avrillon, Ambrus Kaposi, Ambroise Lafont and Johann Rosain**

Deducteam Seminar, LMF
23 April, 2026

Preprint: `https://arxiv.org/abs/2601.19426`

Problem: I Want To Use (Q)IITs

Rocq:

- Does not support IITs natively.
- IITs can be reduced to indexed inductive types Kaposi, Kovács, and Lafont 2019; Sestini 2023, but in extensional/observational type theory.
- QIITs do not have native support.

Cubical Agda:

- Supports a wider class of inductive types natively.
- Okay! I will use Cubical Agda.

The QIIT Specification of MLTT

Type theory

$\mathsf{Con}_- : \mathbb{N} \to \mathsf{Set}$
$\mathsf{Ty}_- : \mathbb{N} \to \mathsf{Con}_i \to \mathsf{Set}$
$\mathsf{Sub} : \mathsf{Con}_i \to \mathsf{Con}_j \to \mathsf{Set}$
$\mathsf{Tm} : (\Gamma : \mathsf{Con}_i) \to \mathsf{Ty}_j\,\Gamma \to \mathsf{Set}$

Substitution calculus

$\cdot : \mathsf{Con}_0$
$- \triangleright - : (\Gamma : \mathsf{Con}_i) \to \mathsf{Ty}_j\,\Gamma \to \mathsf{Con}_{i\sqcup j}$
$-[-] : \mathsf{Ty}_j\,\Delta \to \mathsf{Sub}\,\Gamma\,\Delta \to \mathsf{Ty}_j\,\Gamma$
$\mathsf{id} : \mathsf{Sub}\,\Gamma\,\Gamma$
$- \circ - : \mathsf{Sub}\,\Theta\,\Delta \to \mathsf{Sub}\,\Gamma\,\Theta \to \mathsf{Sub}\,\Gamma\,\Delta$
$\epsilon : \mathsf{Sub}\,\Gamma\,\cdot$
$-, - : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \mathsf{Tm}\,\Gamma\,(A[\sigma]) \to \mathsf{Sub}\,\Gamma\,(\Delta \triangleright A)$
$\pi_1 : \mathsf{Sub}\,\Gamma\,(\Delta \triangleright A) \to \mathsf{Sub}\,\Gamma\,\Delta$
$\pi_2 : (\sigma : \mathsf{Sub}\,\Gamma\,(\Delta \triangleright A)) \to \mathsf{Tm}\,\Gamma\,(A[\pi_1\,\sigma])$
$-[-] : \mathsf{Tm}\,\Delta\,A \to (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \mathsf{Tm}\,\Gamma\,(A[\sigma])$
$[\mathsf{id}] : A[\mathsf{id}] = A$
$[\circ] : A[\sigma \circ \delta] = A[\sigma][\delta]$
$\mathsf{ass} : (\sigma \circ \delta) \circ \nu = \sigma \circ (\delta \circ \nu)$
$\mathsf{idl} : \mathsf{id} \circ \sigma = \sigma$
$\mathsf{idr} : \sigma \circ \mathsf{id} = \sigma$
$\cdot\eta : (\sigma : \mathsf{Sub}\,\Gamma\,\cdot) = \epsilon$
$\triangleright\beta_1 : \pi_1\,(\sigma, t) = \sigma$
$\triangleright\beta_2 : \pi_2\,(\sigma, t) = t$
$\triangleright\eta : (\pi_1\,\sigma, \pi_2\,\sigma) = \sigma$
$, \circ : (\sigma, t) \circ \delta = (\sigma \circ \delta, t[\delta])$

Function space

$\Pi : (A : \mathsf{Ty}_i\,\Gamma) \to \mathsf{Ty}_j\,(\Gamma \triangleright A) \to \mathsf{Ty}_{i\sqcup j}\,\Gamma$
$\mathsf{lam} : \mathsf{Tm}\,(\Gamma \triangleright A)\,B \to \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B)$
$\mathsf{app} : \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \to \mathsf{Tm}\,(\Gamma \triangleright A)\,B$
$\Pi\beta : \mathsf{app}\,(\mathsf{lam}\,t) = t$
$\Pi\eta : \mathsf{lam}\,(\mathsf{app}\,t) = t$
$\Pi[] : (\Pi\,A\,B)[\sigma] = \Pi\,(A[\sigma])\,(B[\sigma^\uparrow])$
$\mathsf{lam}[] : (\mathsf{lam}\,t)[\sigma] = \mathsf{lam}\,(t[\sigma^\uparrow])$

Sigma

$\Sigma : (A : \mathsf{Ty}_i\,\Gamma) \to \mathsf{Ty}_j\,(\Gamma \triangleright A) \to \mathsf{Ty}_{i\sqcup j}\,\Gamma$
$-, - : (u : \mathsf{Tm}\,\Gamma\,A) \to \mathsf{Tm}\,\Gamma\,(B[\langle u\rangle]) \to \mathsf{Tm}\,\Gamma\,(\Sigma\,A\,B)$
$\mathsf{proj}_1 : \mathsf{Tm}\,\Gamma\,(\Sigma\,A\,B) \to \mathsf{Tm}\,\Gamma\,A$
$\mathsf{proj}_2 : (t : \mathsf{Tm}\,\Gamma\,(\Sigma\,A\,B)) \to \mathsf{Tm}\,\Gamma\,(B[\langle \mathsf{proj}_1\,u\rangle])$
$\Sigma\beta_1 : \mathsf{proj}_1\,(u, v) = u$
$\Sigma\beta_2 : \mathsf{proj}_2\,(u, v) = v$
$\Sigma\eta : (\mathsf{proj}_1\,t, \mathsf{proj}_2\,t) = t$
$\Sigma[] : (\Sigma\,A\,B)[\sigma] = \Sigma\,(A[\sigma])\,(B[\sigma^\uparrow])$
$, [] : (u, v)[\sigma] = (u[\sigma], v[\sigma])$

Unit

$\top : \mathsf{Ty}_0\,\Gamma$
$\mathsf{tt} : \mathsf{Tm}\,\Gamma\,\top$
$\top\eta : (t : \mathsf{Tm}\,\Gamma\,\top) = \mathsf{tt}$
$\top[] : \top[\sigma] = \top$
$\mathsf{tt}[] : \mathsf{tt}[\sigma] = \mathsf{tt}$

Empty

$\bot : \mathsf{Ty}_0\,\Gamma$
$\mathsf{exfalso} : \mathsf{Tm}\,\Gamma\,\bot \to \mathsf{Tm}\,\Gamma\,C$
$\bot[] : \bot[\sigma] = \bot$
$\mathsf{exfalso}[] : (\mathsf{exfalso}\,t)[\sigma] = \mathsf{exfalso}\,(t[\sigma])$

Sum

$- + - : \mathsf{Ty}_i\,\Gamma \to \mathsf{Ty}_j\,\Gamma \to \mathsf{Ty}_{i\sqcup j}\,\Gamma$
$\mathsf{inj}_1 : \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,(A + B)$
$\mathsf{inj}_2 : \mathsf{Tm}\,\Gamma\,B \to \mathsf{Tm}\,\Gamma\,(A + B)$
$\mathsf{case} : (P : \mathsf{Ty}_i\,(\Gamma \triangleright A + B)) \to \mathsf{Tm}\,(\Gamma \triangleright A)\,(P[\mathsf{wk}, \mathsf{inj}_1\,\mathsf{vz}]) \to \mathsf{Tm}\,(\Gamma \triangleright B)\,(P[\mathsf{wk}, \mathsf{inj}_2\,\mathsf{vz}]) \to (t : \mathsf{Tm}\,\Gamma\,(A + B)) \to \mathsf{Tm}\,\Gamma\,(P[\langle t\rangle])$
$+\beta_1 : \mathsf{case}\,P\,u\,v\,(\mathsf{inj}_1\,t) = u[\langle t\rangle]$
$+\beta_2 : \mathsf{case}\,P\,u\,v\,(\mathsf{inj}_2\,t) = v[\langle t\rangle]$
$+[] : (A + B)[\sigma] = A[\sigma] + B[\sigma]$
$\mathsf{inj}_1[] : (\mathsf{inj}_1\,t)[\sigma] = \mathsf{inj}_1\,(t[\sigma])$
$\mathsf{inj}_2[] : (\mathsf{inj}_2\,t)[\sigma] = \mathsf{inj}_2\,(t[\sigma])$
$\mathsf{case}[] : (\mathsf{case}\,P\,u\,v\,t)[\sigma] = \mathsf{case}\,(P[\sigma^\uparrow])\,(u[\sigma^\uparrow])\,(v[\sigma^\uparrow])\,(t[\sigma])$

Coquand universes

$\mathsf{U}_- : (i : \mathbb{N}) \to \mathsf{Ty}_{i+1}\,\Gamma$
$\mathsf{El} : \mathsf{Tm}\,\Gamma\,\mathsf{U}_i \to \mathsf{Ty}_i\,\Gamma$
$\mathsf{c} : \mathsf{Ty}_i\,\Gamma \to \mathsf{Tm}\,\Gamma\,\mathsf{U}_i$
$\mathsf{U}\beta : \mathsf{El}\,(\mathsf{c}\,A) = A$
$\mathsf{U}\eta : \mathsf{c}\,(\mathsf{El}\,a) = a$
$\mathsf{U}[] : \mathsf{U}_i[\sigma] = \mathsf{U}_i$
$\mathsf{El}[] : (\mathsf{El}\,a)[\sigma] = \mathsf{El}\,(a[\sigma])$

Booleans

$\mathsf{Bool} : \mathsf{Ty}_0$
$\mathsf{true} : \mathsf{Tm}\,\Gamma\,\mathsf{Bool}$
$\mathsf{false} : \mathsf{Tm}\,\Gamma\,\mathsf{Bool}$
$\mathsf{if} : (P : \mathsf{Ty}_i\,(\Gamma \triangleright \mathsf{Bool})) \to \mathsf{Tm}\,\Gamma\,(P[\langle\mathsf{true}\rangle]) \to \mathsf{Tm}\,\Gamma\,(P[\langle\mathsf{false}\rangle]) \to (t : \mathsf{Tm}\,\Gamma\,\mathsf{Bool}) \to \mathsf{Tm}\,\Gamma\,(P[\langle t\rangle])$
$\mathsf{Bool}\beta_{\mathsf{true}} : \mathsf{if}\,P\,u\,v\,\mathsf{true} = u$
$\mathsf{Bool}\beta_{\mathsf{false}} : \mathsf{if}\,P\,u\,v\,\mathsf{false} = v$
$\mathsf{Bool}[] : \mathsf{Bool}[\sigma] = \mathsf{Bool}$
$\mathsf{true}[] : \mathsf{true}[\sigma] = \mathsf{true}$
$\mathsf{false}[] : \mathsf{false}[\sigma] = \mathsf{false}$
$\mathsf{if}[] : (\mathsf{if}\,P\,u\,v\,t)[\sigma] = \mathsf{if}\,(P[\sigma^\uparrow])\,(u[\sigma])\,(v[\sigma])\,(t[\sigma])$

Natural numbers

$\mathsf{Nat} : \mathsf{Ty}_0$

From Ambrus' repo.

The QIIT Specification of MLTT

```
Con : Set
Sub : Con → Con → Set
Ty  : Con → Set
Tm  : (Γ : Con) → Ty Γ → Set
...
-- constructors:
_[_] : Ty Γ → Sub Δ Γ → Ty Δ
id   : Sub Γ Γ
...
-- path constructor:
[id] : A [ id ] ≡ A
```

- Multiple inductive types being defined together
- The inductive types are indexed over each other
- Path constructors

A Family Encoding Is Used

Altenkirch, Kaposi, and Xie 2025 use a "family" encoding:

```
U : Set
El   : U → Set
-- all four sorts encoded as codes:
Con : U
Sub : El Con → El Con → U
Ty  : El Con → U
Tm  : (Γ : El Con) → El (Ty Γ) → U
...
```

Altenkirch and Scoccola 2020 also use such a family encoding for defining the HIT of integers.

Family Encoding: Recipe

```
Con : Set
Sub : Con → Con → Set
Ty  : Con → Set
Tm  : (Γ : Con) → Ty Γ → Set
...
_[_] : Ty Γ → Sub Δ Γ → Ty Δ
...
```

Family Encoding: Recipe

1. Add sorts $U : \mathsf{Set}, \mathsf{El} : U \to \mathsf{Set}$ to the GAT

U : Set

El : U ⟶ Set

```
Con : Set
Sub : Con → Con → Set
Ty  : Con → Set
Tm  : (Γ : Con) → Ty Γ → Set
...
_[_] : Ty Γ → Sub Δ Γ → Ty Δ
...
```

Family Encoding: Recipe

1. Add sorts $U : \mathsf{Set}, \mathsf{El} : U \to \mathsf{Set}$ to the GAT
2. Replace Set in the original sorts with U

```
U : Set
El  : U → Set
Con : ~~Set~~ U
Sub : Con → Con → ~~Set~~ U
Ty  : Con → ~~Set~~ U
Tm  : (Γ : Con) → Ty Γ → ~~Set~~ U
...
_[_] : Ty Γ → Sub Δ Γ → Ty Δ
...
```

Family Encoding: Recipe

1. Add sorts $U : \mathsf{Set}, \mathsf{El} : U \to \mathsf{Set}$ to the GAT
2. Replace Set in the original sorts with U
3. Insert El in front of all the original sorts

```
U : Set
El  : U → Set
Con : U
Sub : El Con → El Con → U
Ty  : El Con → U
Tm  : (Γ : El Con) → El (Ty Γ) → U
...
_[_] : El (Ty Γ) → El (Sub Δ Γ) → El (Ty Δ)
...
```

Family Encoding: Recipe

1. Add sorts $U : \mathsf{Set}, \mathsf{El} : U \to \mathsf{Set}$ to the GAT
2. Replace Set in the original sorts with U
3. Insert El in front of all the original sorts

```
U : Set
El  : U → Set
Con : U
Sub : El Con → El Con → U
Ty  : El Con → U
Tm  : (Γ : El Con) → El Ty Γ → U
...
_[_] : El Ty Γ → El Sub Δ Γ → El Ty Δ
...
```

Why Use The Family Encoding

Rocq:

- Does not support IITs natively.
- IITs can be recovered via reduction to indexed inductive types Kaposi, Kovács, and Lafont 2019, but this requires extra work, UIP and funext.
- QIITs do not have native support.

Cubical Agda:

- Supports a wider class of inductive types natively.
- Does **not** support **sort equations** in QIITs.
- Does **not** support **interleaved constructors of different sorts**.
- Does **not** support **interleaved sorts and constructors**.

The family encoding bypasses these restrictions in Cubical Agda.

Why Use These Encodings: Interleaved constructors

Cubical Agda does **not** support interleaving constructors of different sorts.

```
...
-- constructor of sort Ty:
_[_] : Ty Γ → Sub Δ Γ → Ty Δ
-- constructor of sort Sub:
_∘_ : Sub Δ Γ → Sub Θ Δ → Sub Θ Γ
...
-- this equality of sort Ty uses the _∘_ constructor of sort Sub:
[∘] : A [ γ ∘ δ ] = A [ γ ] [ δ ]
-- this constructor of sort Sub, uses the _[_] constructor of sort Ty:
_,_ : (g : Sub Δ Γ) → Tm Δ (A [ g ]) → Sub Δ (Γ , A)
...
```

Why Use These Encodings: Interleaved constructors

After the family encoding:

```
U  : Set
El : U → Set
...
Ty  : El Con → U
Sub : El Con → El Con → U
-- constructors now all target U, no interleaving:
_[_] : El (Ty Γ) → El (Sub Δ Γ) → El (Ty Δ)
_∘_ : El (Sub Δ Γ) → El (Sub Θ Δ) → El (Sub Θ Γ)
...
[∘] : A [ γ ∘ δ ] = A [ γ ] [ δ ]
_,_ : (g : El (Sub Δ Γ)) → El (Tm Δ (A [ g ])) → El (Sub Δ (Γ , A))
...
```

Why Use These Encodings: Sort Equations

Example: Extending MLTT with a Russell universe; adding a type of types.

$$\frac{\Gamma \ \mathsf{ctx}}{\Gamma \vdash \mathsf{Univ}_\Gamma \ \mathsf{type}} \qquad \frac{\Gamma \vdash A : \mathsf{Univ}_\Gamma}{\Gamma \vdash A \ \mathsf{type}}$$

Why Use These Encodings: Sort Equations

Example: Extending MLTT with a Russell universe; adding a type of types.

$$\frac{\Gamma \text{ ctx}}{\Gamma \vdash \mathsf{Univ}_\Gamma \text{ type}} \qquad \frac{\Gamma \vdash A : \mathsf{Univ}_\Gamma}{\Gamma \vdash A \text{ type}}$$

Extending the GAT of MLTT with a Russell universe adds a constructor and a **sort equation** expressing that terms of type $\mathsf{Univ}\,\Gamma$ are identified by types in Γ.

$$\mathsf{Univ} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \qquad \text{and} \qquad \mathsf{Tm}\,\Gamma\,(\mathsf{Univ}\,\Gamma) =_{\mathsf{Set}} \mathsf{Ty}\,\Gamma$$

Sort equations are **not** accepted by Cubical Agda.

Why Use These Encodings: Sort Equations

After the family encoding:

$$\mathsf{Tm}\,\Gamma\,(\mathsf{Univ}\,\Gamma) =_{\mathsf{Set}} \mathsf{Ty}\,\Gamma \qquad \text{becomes} \qquad \mathsf{Tm}\,\Gamma\,(\mathsf{Univ}\,\Gamma) =_{U} \mathsf{Ty}\,\Gamma$$

```
U    : Set
El   : U → Set
Con  : U
Sub  : El Con → El Con → U
Ty   : El Con → U
Tm   : (Γ : El Con) → El (Ty Γ) → U
...
Univ : (Γ : El Con) → El (Ty Γ)
-- sort equation becomes equality in U
UEq  : Tm Γ (Univ Γ) = Ty Γ
...
```

Does It Always Work?

What do we mean by "it works"?

We want a construction that takes the encoded (Q)IIT and:

- gives a model of the original QIIT
- equips this model with a suitable recursion principle (aka initiality principle)

In categorical terms: a functor $\mathsf{Models}(\mathsf{Enc}(\Gamma)) \to \mathsf{Models}(\Gamma)$ that preserves initial objects.

Does It Always Work?

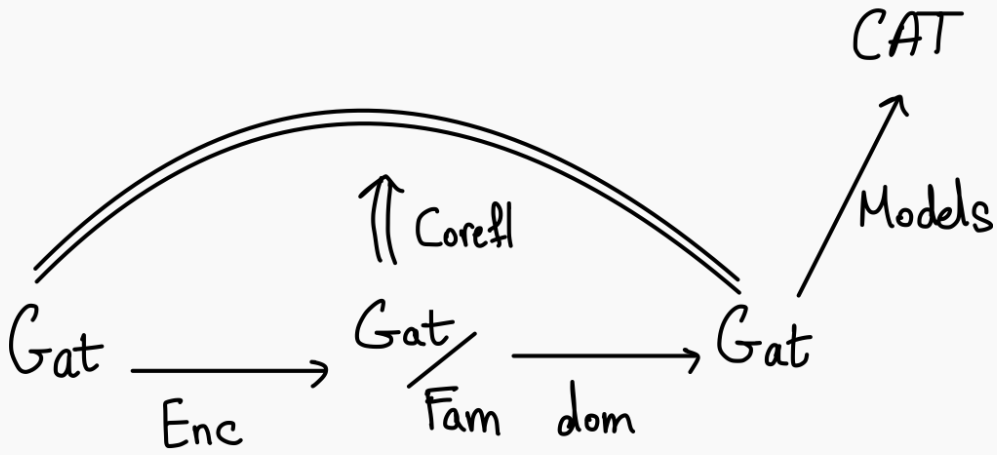
- Sestini 2023 defined a reduction of family inductive-inductive types to W-types in observational type theory, and *conjectured* the existence of a general family encoding to justify focusing on the family IITs.

It Always Works!

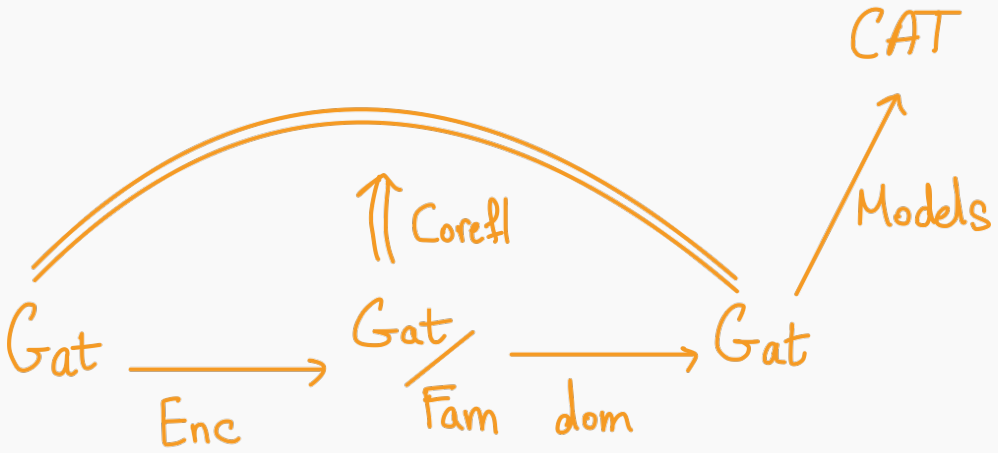
We generalise from (Q)IIT specifications to arbitrary generalised algebraic theories (GATs) and show that "it always works!":

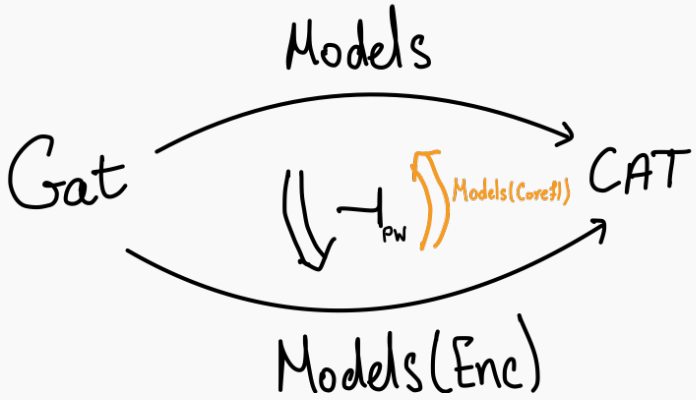
1. There is a fully faithful functor $\mathsf{Enc} : \mathsf{Gat} \to \mathsf{Gat}/\mathsf{Fam}$ doing the family encoding.
2. There is a **strict coreflection** $\mathsf{Models}(\Gamma) \underset{\longleftarrow}{\overset{\longrightarrow}{\bot}} \mathsf{Models}(\mathsf{Enc}(\Gamma))$, i.e. adjunction whose unit is the identity.
   $\Rightarrow$ the initial model of $\mathsf{Enc}(\Gamma)$ yields the initial model of Γ

Roadmap



Roadmap


CAT
Models
Corefl
Gat
Gat
Gat
Enc
Fam
dom


Models
Gat
CAT
Models(Corefl)
PW
Models(Enc)

Contributions

1. Adapting Uemura's universal property for FinGat to Gat
   - Our main technical tool
2. Family encoding functor $\mathrm{Enc} : \mathrm{Gat} \to \mathrm{Gat}/\mathrm{Fam}$, mapping any GAT to a family GAT with sorts $U : \mathrm{Set}$ and $El : U \to \mathrm{Set}$
3. Characterisation of $\mathrm{Models}(\mathrm{Enc}(\Gamma))$ as a kind of comma category "$F_\Gamma/_c Fam$"
4. Consequence of point 3: a strict coreflection $\mathrm{Models}(\Gamma) \underset{\longleftarrow}{\overset{\longrightarrow}{\bot}} \mathrm{Models}(\mathrm{Enc}(\Gamma))$
5. Proving the family encoding functor is fully faithful
   - For this we show the existence of initial models

Universal Property of (Finite) GATs

Uemura's Characterisation

Informally, FinGat is the category of finite GATs and GAT morphisms (substitutions).

Theorem (Uemura 2022)

FinGat, *equipped with* $p_G : (A : \mathrm{Set},\ a : A) \to (A : \mathrm{Set})$, *is* ***bi-initial*** *among categories with finite limits and a choice of an "exponentiable morphism"*[1].

We use this universal property as our main tool.

[1] exponentiable object in a slice

Uemura's Characterisation — Consequence

Key consequence: to define a functor out of FinGat, it suffices to show that the target has finite limits and check that the morphism we want p_G to be mapped to is exponentiable — existence follows from bi-initiality.

Example (The *model functor*)

Let's define a functor

$$\mathsf{Models}(-) \;:\; \mathsf{FinGat} \longrightarrow \mathsf{CAT}$$

mapping each theory Γ to its **category of models** $\mathsf{Models}(\Gamma)$.

We want $p_G : (A : \mathsf{Set},\ a : A) \to (A : \mathsf{Set})$ to be mapped to $\mathsf{PtdSet} \to \mathsf{Set}$. So we check that $\mathsf{PtdSet} \to \mathsf{Set}$ is exponentiable.

Bi-initiality then yields a functor:

$$\mathsf{Models}(-) \;:\; \mathsf{FinGat} \longrightarrow \mathsf{CAT}$$

Intuition

For more intuition about Uemura's characterisation, for example the relation between context extension and p_G: see Uemura 2022 and this TYPES abstract.

`https://niyoushanajmaei.github.io/abstracts/twosort_abs.pdf`

Extension to Infinite GATs

Uemura 2022: **FinGat**, equipped with $p_G : (A : \mathrm{Set},\ a : A) \to (A : \mathrm{Set})$, is bi-initial among categories with **finite limits** and a choice of an "exponentiable morphism".

Theorem

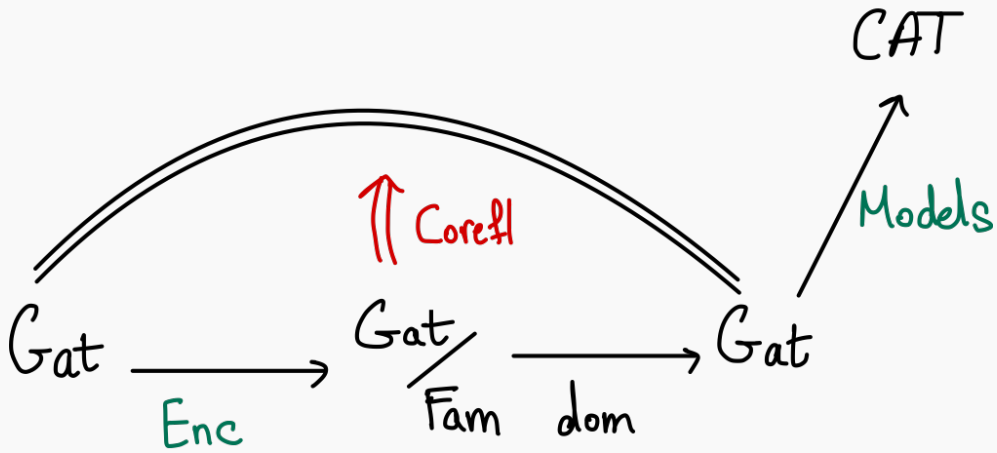
Gat, *equipped with* $p_G : (A : \mathrm{Set},\ a : A) \to (A : \mathrm{Set})$, *is bi-initial among categories with* ***limits*** *and a choice of an "exponentiable morphism".*

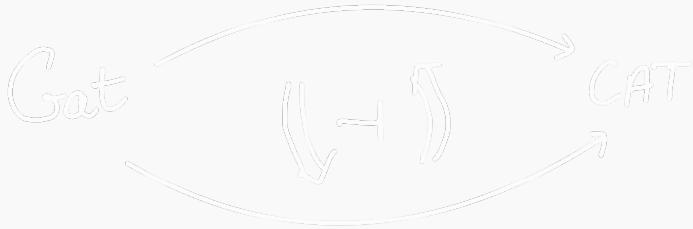
Proof idea

We use that Gat is the Pro-completion of FinGat.

Family Encoding by Bi-initiality

Roadmap





Family Encoding

Theorem

There exists a fully faithful functor

$$\mathrm{Enc} \;:\; \mathrm{Gat} \longrightarrow \mathrm{Gat/Fam}$$

mapping any GAT Γ *to a GAT* $\mathrm{Enc}(\Gamma)$ *with sorts* $(U : \mathrm{Set},\ El : U \to \mathrm{Set})$.

Proof idea (without fully faithfulness)

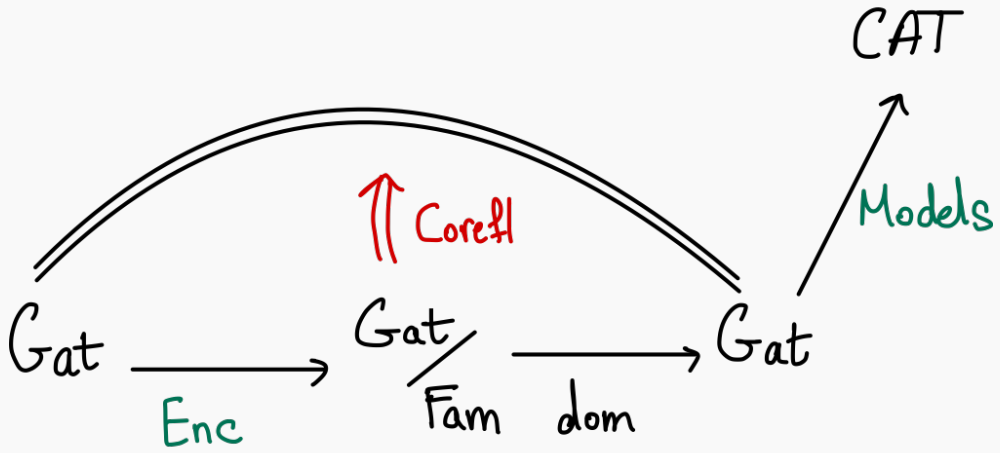
Using bi-initiality of Gat: p_G should be mapped to the morphism

$$(U : \mathrm{Set},\ El : U \to \mathrm{Set},\ A : U,\ a : El\ A) \longrightarrow (U : \mathrm{Set},\ El : U \to \mathrm{Set},\ A : U).$$

We show that this is exponentiable.

The Coreflector Morphism

The functor Enc lands in the slice Gat/Fam, so every $\mathrm{Enc}(\Gamma)$ comes equipped with a projection $\mathrm{Enc}(\Gamma) \to \mathrm{Fam}$. But we can also define **a morphism** $\mathrm{Coref}\mathrm{l}_\Gamma : \mathrm{Enc}(\Gamma) \to \Gamma$ **back to the *original* theory**.



Example of the Family Encoding Functor and Coreflector

Example (Enc(A : Set))

The functor Enc maps the theory $\Gamma := (A : \mathrm{Set})$ to:

$$U : \mathrm{Set}, \quad El : U \to \mathrm{Set}, \quad A : U.$$

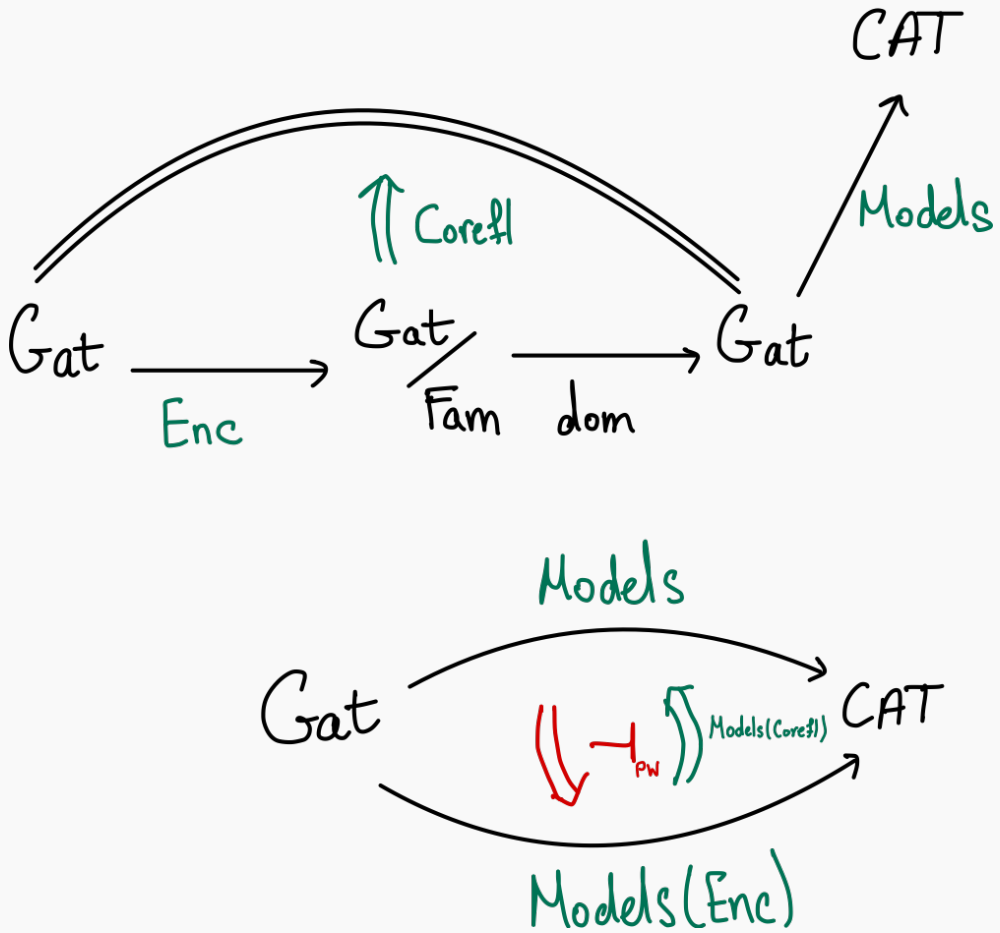
Its coreflector morphism is the following morphism:

$$\mathrm{Corefl} : (U : \mathrm{Set},\ El : U \to \mathrm{Set},\ A : U) \longrightarrow (A : \mathrm{Set})$$

which interprets the sort $A : \mathrm{Set}$ of Γ as the term $El\, A : \mathrm{Set}$ in $\mathrm{Enc}(\Gamma)$.

The Semantic Translation

Roadmap


CAT
Models
Corefl
Gat
Gat
Gat
Enc
Fam
dom
Models
Gat
CAT
Models(Coreflˡ)
PW
Models(Enc)

Models of the Reduced GAT: Key Ingredient for Coreflection

We define the family functor $F_\Gamma : \mathsf{Models}(\Gamma) \to \mathsf{Fam}$ using bi-initiality of Gat.

Theorem

Given Γ *with family functor* $F_\Gamma : \mathsf{Models}(\Gamma) \to \mathsf{Fam}$, *we have* $\mathsf{Models}(\mathsf{Enc}(\Gamma))$ *is iso to the category* $F_\Gamma/_c\mathsf{Fam}$, *where objects are are "cartesian" morphisms of families of the form*

$$F_\Gamma(M) \to (U', El')$$

for some model $M \in \mathsf{Models}(\Gamma)$ *and family* (U', El').

Proof idea

By bi-initiality of Gat.

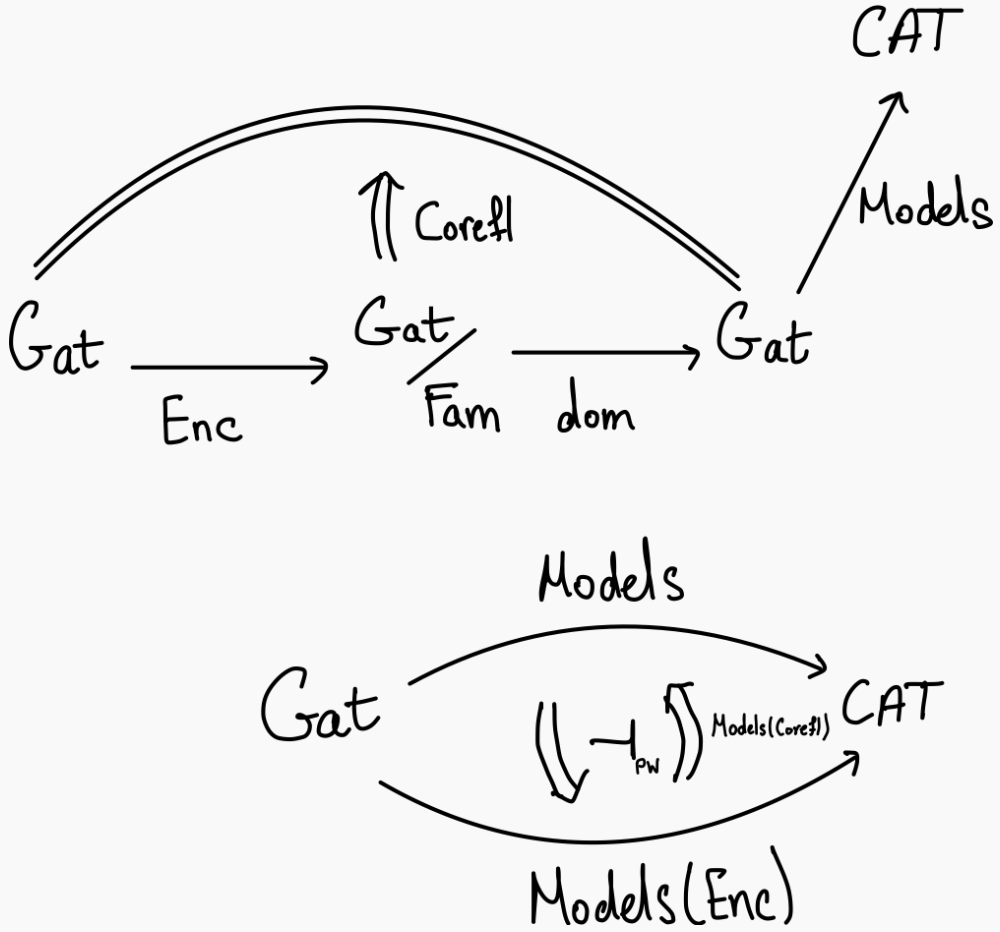
The Coreflection

Now that we have $\mathsf{Models}(\mathsf{Enc}(\Gamma)) \cong F_\Gamma/_c\mathsf{Fam}$, we can define the coreflection.

$$\mathsf{Models}(\Gamma) \underset{\mathsf{Models}(\mathsf{Coref}\mathsf{l}_\Gamma)}{\overset{\longrightarrow}{\underset{\longleftarrow}{\bot}}} \mathsf{Models}(\mathsf{Enc}(\Gamma))$$

- The **right adjoint** $\mathsf{Models}(\mathsf{Coref}\mathsf{l}_\Gamma)$ maps a cartesian morphism $F_\Gamma(M) \to (U', El')$ to its underlying model M.
- The **left adjoint** maps a model M to the identity $F_\Gamma(M) \xrightarrow{\mathrm{id}} F_\Gamma(M)$.
- The unit is the **identity**: *strict* coreflection.
- Round-trip starting from any model of Γ is the identity.

Recap


CAT
Models
Corefl
Gat
Gat
Gat
Enc
Fam
dom
Models
Gat
CAT
Models(Corefl)
PW
Models(Enc)

Thank you for listening! Questions?

References i


Altenkirch, Thorsten, Ambrus Kaposi, and Szumi Xie (2025). **"The Groupoid-syntax of Type Theory is a Set"**. In: *CoRR* abs/2509.14988. DOI: 10.48550/ARXIV.2509.14988. arXiv: 2509.14988. URL: https://doi.org/10.48550/arXiv.2509.14988.


Altenkirch, Thorsten and Luis Scoccola (2020). **"The Integers as a Higher Inductive Type"**. In: *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*. Ed. by Holger Hermanns et al. ACM, pp. 67–73. DOI: 10.1145/3373718.3394760. URL: https://doi.org/10.1145/3373718.3394760.

References ii

- Kaposi, Ambrus, András Kovács, and Ambroise Lafont (2019). **"For Finitary Induction-Induction, Induction Is Enough".** In: *25th International Conference on Types for Proofs and Programs, TYPES 2019, Oslo, Norway, June 11-14, 2019*. Ed. by Marc Bezem and Assia Mahboubi. Vol. 175. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 6:1–6:30. DOI: 10.4230/LIPICS.TYPES.2019.6. URL: https://doi.org/10.4230/LIPIcs.TYPES.2019.6.
- Sestini, Filippo (Dec. 2023). **"Bootstrapping extensionality".** PhD thesis. URL: https://eprints.nottingham.ac.uk/74363/.
- Uemura, Taichi (2022). **"The universal exponentiable arrow".** In: *Journal of Pure and Applied Algebra* 226.7, p. 106991. ISSN: 0022-4049. DOI: https://doi.org/10.1016/j.jpaa.2021.106991. URL: https://www.sciencedirect.com/science/article/pii/S002240492100332 7.

Uemura's Characterisation

Definition

A **CartExp-category** is a category with finite limits equipped with a chosen *exponentiable* morphism $p : Y \to X$ (i.e. the pullback functor $p^* : \mathcal{C}/X \to \mathcal{C}/Y$ has a right adjoint).

A **CartExp-functor** preserves finite limits, the chosen morphism, and the right adjoint.

Informally: FinGat has finite GATs as objects and substitutions as morphisms.

Theorem (Uemura 2022)

FinGat, *equipped with* $p_G : (A : \mathrm{Set},\ a : A) \to (A : \mathrm{Set})$, *is* ***bi-initial*** *in the 2-category of CartExp-categories: for any CartExp-category* (C, p') *there exists a CartExp-functor* FinGat $\to C$ *sending* p_G *to* p', *unique up to unique isomorphism.*

Example: Transitive Graphs

The GAT of transitive graphs:

```
V : Set
E : V → V → Set
T : (v1 v2 v3 : V) → E v1 v2 → E v2 v3 → E v1 v3
```

The Family Functor: A Key Ingredient

Using bi-initiality of Gat, we get the family functor $F_\Gamma : \mathrm{Models}(\Gamma) \to \mathrm{Fam}$.

Example (A : Set)

The family functor of $(A : \mathrm{Set})$ maps a set X to the family (U, El) defined by $U = \{*\}$ and $El(*) = X$.

Example (Transitive Graphs)

For transitive graphs, F_Γ maps a model (V, E, T) to the family (U_G, El_G) defined by $U_G = \{*\} + V \times V$, $El_G(*) = V$, and $El_G(a, b) = E(a, b)$.

Example of the Family Encoding Functor and Coreflector

Example (Enc(**Transitive Graphs**))

The functor Enc maps the GAT of transitive graphs to:

$$U : \mathrm{Set}, \quad El : U \to \mathrm{Set}, \quad V : U, \quad E : El\ V \to El\ V \to U$$

$$T : \prod_{v_1, v_2, v_3 : El\ V} El(E\ v_1\ v_2) \to El(E\ v_2\ v_3) \to El(E\ v_1\ v_3)$$

The coreflector morphism is the GAT morphism $\mathrm{Corefl}_\Gamma : \mathrm{Enc}(\Gamma) \longrightarrow \Gamma$ which interprets:

- the sort $V : \mathrm{Set}$ of Γ as the term $El\ V : \mathrm{Set}$ in $\mathrm{Enc}(\Gamma)$,
- the sort $E : V \to V \to \mathrm{Set}$ of Γ as the term $\lambda\, v_1\ v_2.\ El(E\ v_1\ v_2) : \mathrm{Set}$ in $\mathrm{Enc}(\Gamma)$,
- the operator Enc of Γ as the operator Enc of $\mathrm{Enc}(\Gamma)$.

Example of the Characterisation of Models(Enc(Γ))

Example (What is Models(Enc((A : Set))) via $F_{A:Set}/_c\mathsf{Fam}$)

For $\Gamma := (A : \mathrm{Set})$, an object of $F_\Gamma/_c\mathsf{Fam}$ consists of:

- a set X,
- a family (U', El')
- a function $f : \{*\} \to U'$ such that $El'(f(*)) = X$.

Giving f is choosing an element $A' \in U'$ with $El'(A') = X$. So (U', El', A') is precisely a model of $T(A : \mathrm{Set}) = (U : \mathrm{Set},\ El : U \to \mathrm{Set},\ A : U)$.

Also note that, the round-trip Models(Γ) → Models(Enc(Γ)) → Models(Γ) starting from X gives back X.

Example of the Characterisation of Models(Enc(Γ))

Example (Models(Enc((Transitive Graphs))))

For $\Gamma :=$ Transitive Graphs, an object of $F_\Gamma/_c\mathrm{Fam}$ consists of:

- a transitive graph $G = (V, E, T)$,
- a family (U', El')
- a function $f : \{*\} + V \times V \to U'$ such that $El'(f(*)) = V$ and $El'(f(a, b)) = E(a, b)$ for all $a, b \in V$.

Giving f is choosing:oh

- an element $V' \in U'$ with $El'(V') = V$,
- a function $E' : V \times V \to U'$ with $El'(E'(a, b)) = E(a, b)$.

So (U', El', V', E', T) is precisely a model of Enc(Γ). Also note that, the round-trip Models(Γ) → Models(Enc(Γ)) → Models(Γ) starting from G gives back G.