Mathematical Structuralism and the Univalent Foundations

Homotopy Type Theory as Structuralist Foundations

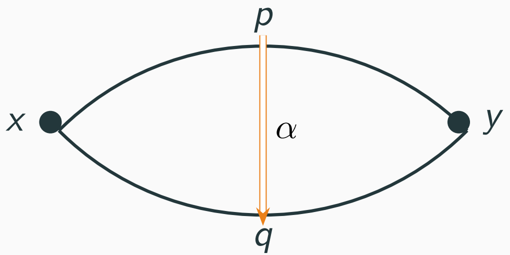
Min Cheol Seo

`mail@mincheolseo.com`

5th Korea Logic Day

14 Jan, 2026

Department of Philosophy
Sungkyunkwan University



1. Motivation: A Tale of Two Naturals

Section 1/8

Outline

Motivation: A Tale of Two Naturals

nat (Peano)

```
Theorem nat_comm :
  forall n m : nat,
    n + m = m + n.
```

- The “usual” naturals
- Proofs by induction / recursion

N (binary)

```
Goal forall n m : N,
  n + m = m + n.
Proof.
  Fail apply nat_comm.
Abort.
```

Error (summary): expected N, found nat.

- More machine-friendly naturals

Problem?

Same mathematics, but **different types** $\Rightarrow$ no direct reuse.

So what? (Not a mere engineering annoyance)

- In practice, we can always *patch* this: conversions, bridges, duplicated lemmas, automation.

So what? (Not a mere engineering annoyance)

- In practice, we can always *patch* this: conversions, bridges, duplicated lemmas, automation.
- But notice what the patch is doing: it is **managing representations**, not proving new mathematics.

So what? (Not a mere engineering annoyance)

- In practice, we can always *patch* this: conversions, bridges, duplicated lemmas, automation.
- But notice what the patch is doing: it is **managing representations**, not proving new mathematics.
- So the real question is not "can we fix it?" but:
  - What, exactly, is the **mathematical content** shared by both encodings?
  - And why doesn't the system give a **canonical route** for reusing proofs?

So what? (Not a mere engineering annoyance)

- In practice, we can always *patch* this: conversions, bridges, duplicated lemmas, automation.
- But notice what the patch is doing: it is **managing representations**, not proving new mathematics.
- So the real question is not "can we fix it?" but:
  - What, exactly, is the **mathematical content** shared by both encodings?
  - And why doesn't the system give a **canonical route** for reusing proofs?

Philosophical motivation

Multiple "equally good" representations can do the same job. So what makes them the same in the relevant sense—and why doesn't reuse follow automatically?

Outline

2. What Numbers Could Not Be?

Section 2/8

Outline

Benacerraf's Arbitrariness Problem

- Arithmetic can be implemented in many ways (e.g. different set-theoretic reductions).
- These implementations can agree on all **arithmetical truths**.

Benacerraf's Arbitrariness Problem

- Arithmetic can be implemented in many ways (e.g. different set-theoretic reductions).
- These implementations can agree on all **arithmetical truths**.
- Yet, if we identify numbers with particular sets, the identity claim becomes **arbitrary**:
  - Why should 3 be *this* set rather than *that* set?
  - Nothing in arithmetic seems to settle the choice.

Benacerraf's Arbitrariness Problem

- Arithmetic can be implemented in many ways (e.g. different set-theoretic reductions).
- These implementations can agree on all **arithmetical truths**.
- Yet, if we identify numbers with particular sets, the identity claim becomes **arbitrary**:
  - Why should 3 be *this* set rather than *that* set?
  - Nothing in arithmetic seems to settle the choice.

Benacerraf's moral I

The pressure is not to pick *the* right representative, but to articulate what counts as **content** across equally good representations.

Junk: when the foundational language is too expressive

- In ZF-style foundations, $\in$ is primitive.
- If numbers are implemented as sets, the language can form questions like:

$$1 \in 3 \,? \qquad 2 \in 4 \,?$$

Junk: when the foundational language is too expressive

- In ZF-style foundations, $\in$ is primitive.
- If numbers are implemented as sets, the language can form questions like:

$$1 \in 3\ ? \qquad 2 \in 4\ ?$$

- These are not arithmetical questions; they track **coding artifacts**.
- Across equally good implementations, their truth-values can diverge.

Junk: when the foundational language is too expressive

- In ZF-style foundations, $\in$ is primitive.
- If numbers are implemented as sets, the language can form questions like:

$$1 \in 3\ ? \qquad 2 \in 4\ ?$$

- These are not arithmetical questions; they track **coding artifacts**.
- Across equally good implementations, their truth-values can diverge.

Benacerraf's moral II

Junk is not "bad taste"; it is a **language-design** issue: what your foundation makes expressible once you commit to a representation.

3. Structuralism as a Constraint on Language

Section 3/8

Outline

Structuralism in 60 seconds

Core thought

Mathematics is primarily about **structural roles and relations**, not about the **thisness** (*haecceity*) of particular representatives.

Structuralism in 60 seconds

Core thought

Mathematics is primarily about **structural roles and relations**, not about the **thisness** (*haecceity*) of particular representatives.

- Two representations can be "the same for mathematics" even if they are not literally identical.
- So the slogan is: **isomorphic/equivalent structures should be treated as the same**.

Structuralism in 60 seconds

Core thought

Mathematics is primarily about **structural roles and relations**, not about the **thisness** (*haecceity*) of particular representatives.

- Two representations can be "the same for mathematics" even if they are not literally identical.
- So the slogan is: **isomorphic/equivalent structures should be treated as the same**.

How I will read this today

Not as an ontological thesis first, but as a constraint on what our **foundational language** should count as meaningful.

From representation problems to language constraints

Two symptoms (Section 2)

- **Arbitrariness:** many equally good representatives ⇒ identity claims look arbitrary.
- **Junk:** expressive primitives + fixed representation ⇒ non-mathematical questions proliferate.

Diagnostic shift

- Which statements *track structural content* (not coding artefacts)?
- When structures count as "the same", how should *proofs/constructions move*?

Up-shot: read structuralism *not first as ontology*, but as a *constraint on meaningful foundational language*.

C1 is standard—but how do we implement it?

C1 (informal)

If $A \approx B$ (isomorphic/equivalent), then content-allowed sentences should not distinguish A from B.

C1 is standard—but how do we implement it?

C1 (informal)

If $A \approx B$ (isomorphic/equivalent), then content-allowed sentences should not distinguish A from B.

- **A natural attempt:** weaken the language so that "junk" becomes inexpressible.
- **Example:** ETCS replaces primitive $\in$ with structural primitives (objects/arrows).

C1 is standard—but how do we implement it?

C1 (informal)

If $A \approx B$ (isomorphic/equivalent), then content-allowed sentences should not distinguish A from B.

- **A natural attempt:** weaken the language so that "junk" becomes inexpressible.
- **Example:** ETCS replaces primitive $\in$ with structural primitives (objects/arrows).

But: C1 is not automatic

Even without $\in$, **primitive equality** + **naming** can reintroduce haecceity. So we still need a principled boundary for "content-allowed" language.

ETCS-style haecceity: “representative-picking” without $\in$

Setup (purely categorical vocabulary)

Let A, B be objects with an isomorphism $e : A \cong B$.

Assume we have a named arrow (a global element) $o : 1 \to A$.

ETCS-style haecceity: “representative-picking” without $\in$

Setup (purely categorical vocabulary)

Let A, B be objects with an isomorphism $e : A \cong B$.

Assume we have a named arrow (a global element) $o : 1 \to A$.

A haecceitistic formula

Let $\mathrm{cod} : \mathrm{Arr} \to \mathrm{Obj}$ be “codomain”. Define

$$\varphi(x) \;\equiv\; \mathrm{cod}(o) = x.$$

ETCS-style haecceity: “representative-picking” without $\in$

Setup (purely categorical vocabulary)

Let A, B be objects with an isomorphism $e : A \cong B$.

Assume we have a named arrow (a global element) $o : 1 \to A$.

A haecceitistic formula

Let $\mathrm{cod} : \mathrm{Arr} \to \mathrm{Obj}$ be “codomain”. Define

$$\varphi(x) \;\equiv\; \mathrm{cod}(o) = x.$$

- Then $\varphi(A)$ holds *by construction* (since $\mathrm{cod}(o) = A$).
- But $\varphi(B)$ is not forced by $A \cong B$: in many models where $A \neq B$ (strict object-identity), $\varphi(B)$ fails.

ETCS-style haecceity: "representative-picking" without $\in$

Setup (purely categorical vocabulary)

Let A, B be objects with an isomorphism $e : A \cong B$.

Assume we have a named arrow (a global element) $o : 1 \to A$.

A haecceitistic formula

Let $\mathrm{cod} : \mathrm{Arr} \to \mathrm{Obj}$ be "codomain". Define

$$\varphi(x) \;\equiv\; \mathrm{cod}(o) = x.$$

- Then $\varphi(A)$ holds *by construction* (since $\mathrm{cod}(o) = A$).
- But $\varphi(B)$ is not forced by $A \cong B$: in many models where $A \neq B$ (strict object-identity), $\varphi(B)$ fails.

Point

So "no $\in$" does *not* by itself block representative-picking. The culprit is **primitive**

Transition: from ETCS to identity-sensitive language design

- ETCS can *weaken* one major junk-generator ($\in$), but haecceity can re-enter.

Transition: from ETCS to identity-sensitive language design

- ETCS can *weaken* one major junk-generator ($\in$), but haecceity can re-enter.
- Diagnosis: if your foundational language lets you write "this object is *that very object*", C1 will be fragile.

Transition: from ETCS to identity-sensitive language design

- ETCS can *weaken* one major junk-generator ($\in$), but haecceity can re-enter.
- Diagnosis: if your foundational language lets you write "this object is *that very object*", C1 will be fragile.
- So a more radical design move suggests itself:
    - Restrict (or reconstruct) **object-identity** in the language, rather than taking it as primitive.

Transition: from ETCS to identity-sensitive language design

- ETCS can *weaken* one major junk-generator ($\in$), but haecceity can re-enter.
- Diagnosis: if your foundational language lets you write "this object is *that very object*", C1 will be fragile.
- So a more radical design move suggests itself:
  - Restrict (or reconstruct) **object-identity** in the language, rather than taking it as primitive.

Next

This motivates treating structuralism as **explicit constraints** (C1/C2), and then asking what kind of language can actually realise them.

4. Structuralist Language: Two Constraints

Section 4/8

Outline

Two constraints on a structuralist foundation

C1: Content invariance

Fix a notion of structural sameness $\approx$ (iso/equiv/...).
Content-allowed statements should not distinguish $x \approx y$.

$$x \approx y \Rightarrow (\varphi(x) \leftrightarrow \varphi(y))$$

C2: Canonical transfer

Not only truth, but *constructions* must move: definitions, lemmas, witnesses, proofs.

$$\mathrm{Tr}_P : (X \approx Y) \times P(X) \to P(Y)$$

Two constraints on a structuralist foundation

C1: Content invariance

Fix a notion of structural sameness $\approx$ (iso/equiv/...).
Content-allowed statements should not distinguish $x \approx y$.

$$x \approx y \Rightarrow (\varphi(x) \leftrightarrow \varphi(y))$$

C2: Canonical transfer

Not only truth, but *constructions* must move: definitions, lemmas, witnesses, proofs.

$$\mathrm{Tr}_P : (X \approx Y) \times P(X) \to P(Y)$$

Key contrast

C1 is about **truth-values**.

C2 is about **reuse and stability of reasoning**.

C2 is not “same proof script” (a precise reading)

What C2 does not say

It does *not* demand that the **same syntactic proof text** works across presentations.

C2 is not "same proof script" (a precise reading)

What C2 does not say

It does *not* demand that the **same syntactic proof text** works across presentations.

What C2 does say

Given evidence $e : X \approx Y$, there should be a **canonical and coherent transport** of structure/content across e.

$$f_Y \;:=\; e \circ f_X \circ e^{-1} \qquad\qquad R_Y(\vec{y}) \;:\Longleftrightarrow\; R_X(e^{-1}(\vec{y})).$$

C2 is not “same proof script” (a precise reading)

What C2 does not say

It does *not* demand that the **same syntactic proof text** works across presentations.

What C2 does say

Given evidence $e : X \approx Y$, there should be a **canonical and coherent transport** of structure/content across e.

$$f_Y \;:=\; e \circ f_X \circ e^{-1} \qquad\qquad R_Y(\vec{y}) \;:\Longleftrightarrow\; R_X(e^{-1}(\vec{y})).$$

Heuristic

C2 is a **rule for moving meaning and constructions**, not a demand for textual reuse.

C2 as a design spec + practice-based criteria

Methodological stance

C2 is not a metaphysical conclusion of structuralism. It is a **specification for a foundational language** meant to support structural practice.

C2 as a design spec + practice-based criteria

Methodological stance

C2 is not a metaphysical conclusion of structuralism. It is a **specification for a foundational language** meant to support structural practice.

- **Practice criterion 1 (“up to isomorphism”):**
  Mathematicians routinely treat isomorphic presentations as interchangeable. That norm implicitly presupposes robust transfer of constructions.

C2 as a design spec + practice-based criteria

Methodological stance

C2 is not a metaphysical conclusion of structuralism. It is a **specification for a foundational language** meant to support structural practice.

- **Practice criterion 1 ("up to isomorphism"):** Mathematicians routinely treat isomorphic presentations as interchangeable. That norm implicitly presupposes robust transfer of constructions.
- **Practice criterion 2 (large-scale formalization / social practice):** If we want reusable libraries and collaborative formalization, non-canonical transfer becomes a scalability bottleneck.

C2 as a design spec + practice-based criteria

Methodological stance

C2 is not a metaphysical conclusion of structuralism. It is a **specification for a foundational language** meant to support structural practice.

- **Practice criterion 1 ("up to isomorphism"):**
  Mathematicians routinely treat isomorphic presentations as interchangeable. That norm implicitly presupposes robust transfer of constructions.
- **Practice criterion 2 (large-scale formalization / social practice):**
  If we want reusable libraries and collaborative formalization, non-canonical transfer becomes a scalability bottleneck.

Up-shot

So C2 is justified as a **language-engineering requirement** whose success is measured against mathematical practice.

C2 in everyday mathematics (one intuition)

A familiar pattern

Same object, different presentation $\Rightarrow$ we expect a **canonical rule** to move data/proofs across presentations.

C2 in everyday mathematics (one intuition)

A familiar pattern

Same object, different presentation $\Rightarrow$ we expect a **canonical rule** to move data/proofs across presentations.

- Linear algebra: change of basis changes notation, but comes with a canonical transformation.
- Graphs: relabeling changes names, but an isomorphism canonically transports witnesses.

C2 in everyday mathematics (one intuition)

A familiar pattern

Same object, different presentation $\Rightarrow$ we expect a **canonical rule** to move data/proofs across presentations.

- Linear algebra: change of basis changes notation, but comes with a canonical transformation.
- Graphs: relabeling changes names, but an isomorphism canonically transports witnesses.

What C2 demands

Transfer must not be merely possible; it should be **canonical** and **coherent**.

5. The Univalent Foundations

Section 5/8

Outline

HoTT/UF: what is the package?

Two layers

- **Syntax (MLTT):** dependent types + identity types ($x = y$).
- **Semantics (homotopy):** interpret types as spaces / ∞-groupoids.

HoTT/UF: what is the package?

Two layers

- **Syntax (MLTT):** dependent types + identity types ($x = y$).
- **Semantics (homotopy):** interpret types as spaces / ∞-groupoids.

Why it matters for us

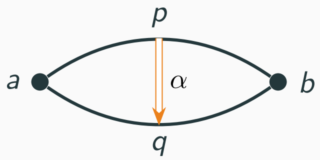
In HoTT/UF, **equality is structured** (not just a truth-value), and it comes with a built-in mechanism for **transport**.

- This is exactly the kind of mechanism C2 was asking for.
- Univalence then extends it from $(=)$ to $(\approx)$.

The ∞-groupoid viewpoint (one diagram, one moral)

Reading a type A

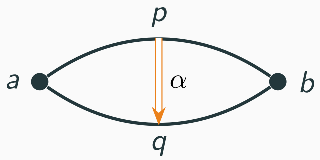
- terms $a : A$ are **points**
- proofs $p : a = b$ are **paths**
- proofs $\alpha : p = q$ are **homotopies** (paths between paths)
- and so on $\Rightarrow$ **higher equalities**



The ∞-groupoid viewpoint (one diagram, one moral)

Reading a type A

- terms $a : A$ are **points**
- proofs $p : a = b$ are **paths**
- proofs $\alpha : p = q$ are **homotopies** (paths between paths)
- and so on $\Rightarrow$ **higher equalities**



Moral for structuralism

"Sameness" is not a bare predicate: it has **internal coherence data**. This is why HoTT/UF is a natural habitat for C2-style constraints.

Identity types: equality as an object you can use

Identity type (informal)

For $a, b : A$, the type $(a = b)$ is the type of **identifications** of a and b. A term $p : a = b$ is a **witness** of equality.

Identity types: equality as an object you can use

Identity type (informal)

For $a, b : A$, the type $(a = b)$ is the type of **identifications** of a and b. A term $p : a = b$ is a **witness** of equality.

- This makes equality *first-class*: you can quantify over it and compute with it.
- Higher equalities $(p = q)$ are also internal objects, enabling coherence control.

Identity types: equality as an object you can use

Identity type (informal)

For $a, b : A$, the type $(a = b)$ is the type of **identifications** of a and b. A term $p : a = b$ is a **witness** of equality.

- This makes equality *first-class*: you can quantify over it and compute with it.
- Higher equalities $(p = q)$ are also internal objects, enabling coherence control.

Why we care

C2 needs evidence-sensitive transfer. In HoTT, such evidence is literally $p : a = b$.

Path induction (J): the core rule for reasoning about identity

Path induction (informal statement)

To prove something about an arbitrary $p : x = y$, it suffices to prove it in the case $p \equiv \mathrm{refl}_x : x = x$.

Path induction (J): the core rule for reasoning about identity

Path induction (informal statement)

To prove something about an arbitrary $p : x = y$, it suffices to prove it in the case $p \equiv \mathrm{refl}_x : x = x$.

Schematic form

Given a family $C : \prod_{x,y:A}(x = y) \to \mathcal{U}$,
if you have $c : \prod_{x:A} C(x, x, \mathrm{refl}_x)$, then you get

$$J(c) : \prod_{x,y:A} \prod_{p:x=y} C(x, y, p).$$

Path induction (J): the core rule for reasoning about identity

Path induction (informal statement)

To prove something about an arbitrary $p : x = y$, it suffices to prove it in the case $p \equiv \mathrm{refl}_x : x = x$.

Schematic form

Given a family $C : \prod_{x,y:A}(x = y) \to \mathcal{U}$,
if you have $c : \prod_{x:A} C(x, x, \mathrm{refl}_x)$, then you get

$$J(c) : \prod_{x,y:A} \prod_{p:x=y} C(x, y, p).$$

Punchline

This is what makes **transport canonical and coherent** (no ad hoc choices).

Transport: C2 for definable families comes "for free"

Transport (key construction)

Let $P : A \to \mathcal{U}$ and $p : x = y$. Then there is a canonical map

$$\mathrm{transport}_P(p, -) : P(x) \to P(y).$$

Transport: C2 for definable families comes "for free"

Transport (key construction)

Let $P : A \to \mathcal{U}$ and $p : x = y$. Then there is a canonical map

$$\mathrm{transport}_P(p, -) : P(x) \to P(y).$$

- $\mathrm{transport}_P(\mathrm{refl}_x, -)$ is judgmentally the identity.
- Transport respects composition of paths $\Rightarrow$ coherence "built-in".

Transport: C2 for definable families comes "for free"

Transport (key construction)

Let $P : A \to \mathcal{U}$ and $p : x = y$. Then there is a canonical map

$$\mathrm{transport}_P(p, -) : P(x) \to P(y).$$

- $\mathrm{transport}_P(\mathrm{refl}_x, -)$ is judgmentally the identity.
- Transport respects composition of paths $\Rightarrow$ coherence "built-in".

Connection to our constraints

For $(=)$, HoTT already implements the core of **C2: canonical transfer**.

6. Univalence Axiom

Section 6/8

Outline

The gap: we need transport along $\approx$, not only along $=$

- Structuralism works with a notion of sameness $\approx$ (iso/equiv/...).
- But HoTT's built-in transport is along **identity** ($=$).

The gap: we need transport along $\approx$, not only along $=$

- Structuralism works with a notion of sameness $\approx$ (iso/equiv/...).
- But HoTT's built-in transport is along **identity** ($=$).
- So: how do we get **canonical transport along equivalence**?

The gap: we need transport along $\approx$, not only along $=$

- Structuralism works with a notion of sameness $\approx$ (iso/equiv/...).
- But HoTT's built-in transport is along **identity** ($=$).
- So: how do we get **canonical transport along equivalence**?

This is exactly what Univalence provides

It turns equivalence into a source of identity.

Univalence (statement)

Univalence (slogan)

For types $A, B : \mathcal{U}$, identity is equivalent to equivalence:

$$(A = B) \simeq (A \simeq B).$$

Univalence (statement)

Univalence (slogan)

For types $A, B : \mathcal{U}$, identity is equivalent to equivalence:

$$(A = B) \simeq (A \simeq B).$$

- A term $p : A = B$ gives an equivalence (by transport).
- Univalence adds (roughly) the converse: an equivalence gives a path.

Univalence (statement)

Univalence (slogan)

For types $A, B : \mathcal{U}$, identity is equivalent to equivalence:

$$(A = B) \simeq (A \simeq B).$$

- A term $p : A = B$ gives an equivalence (by transport).
- Univalence adds (roughly) the converse: an equivalence gives a path.

Structuralist reading

Univalence internalizes the principle: “equivalent structures count as equal”.

Univalence $\Rightarrow$ transport along equivalence (the C2 engine for $\approx$)

From equivalence to transport

Assume $e : A \simeq B$. By univalence, obtain a path $p : A = B$. Then for any $P : \mathcal{U} \to \mathcal{V}$ we get

$$\mathrm{Tr}_P(e, -) \ := \ \mathrm{transport}_P(p, -) : P(A) \to P(B).$$

Univalence $\Rightarrow$ transport along equivalence (the C2 engine for $\approx$)

From equivalence to transport

Assume $e : A \simeq B$. By univalence, obtain a path $p : A = B$. Then for any $P : \mathcal{U} \to \mathcal{V}$ we get

$$\mathrm{Tr}_P(e, -) \;:=\; \mathrm{transport}_P(p, -) : P(A) \to P(B).$$

- Canonical: depends only on e via p (no extra choices).
- Coherent: inherits coherence laws from path induction.

Univalence $\Rightarrow$ transport along equivalence (the C2 engine for $\approx$)

From equivalence to transport

Assume $e : A \simeq B$. By univalence, obtain a path $p : A = B$. Then for any $P : \mathcal{U} \to \mathcal{V}$ we get

$$\mathrm{Tr}_P(e, -) \ := \ \mathrm{transport}_P(p, -) : P(A) \to P(B).$$

- Canonical: depends only on e via p (no extra choices).
- Coherent: inherits coherence laws from path induction.

This matches C2 as we defined it

Evidence-sensitive, canonical, coherent transfer under $\approx$.

C1 and C2 become internal lemmas (clean payoff)

Lemma-form C2

For any $P : \mathcal{U} \to \mathcal{V}$,

$$e : A \simeq B \Rightarrow \mathrm{Tr}_P(e, -) : P(A) \to P(B).$$

- "Reuse" becomes definable transport.
- Coherence is inherited (not bolted on).

Lemma-form C1

For any $P : \mathcal{U} \to \mathrm{Prop}$,

$$e : A \simeq B \Rightarrow (P(A) \leftrightarrow P(B)).$$

- Content invariance follows from transport.
- "Up to equivalence" is built into meaning.

C1 and C2 become internal lemmas (clean payoff)

Lemma-form C2

For any $P : \mathcal{U} \to \mathcal{V}$,

$$e : A \simeq B \Rightarrow \mathrm{Tr}_P(e, -) : P(A) \to P(B).$$

- "Reuse" becomes definable transport.
- Coherence is inherited (not bolted on).

Lemma-form C1

For any $P : \mathcal{U} \to \mathrm{Prop}$,

$$e : A \simeq B \Rightarrow (P(A) \leftrightarrow P(B)).$$

- Content invariance follows from transport.
- "Up to equivalence" is built into meaning.

Why HoTT/UF?

Because it is a foundational language where the structuralist constraints (C1 invariance, C2 canonical transfer) are **implemented**, not merely postulated.

7. HoTT as Structuralist Heaven?

Section 7/8

Outline

What HoTT/UF delivers (and what it does not)

What we gained

- A built-in notion of **evidence-sensitive, coherent transport** (via identity).
- Univalence: **equivalence becomes a source of transport** $\Rightarrow$ C1/C2 internalized.

What HoTT/UF delivers (and what it does not)

What we gained

- A built-in notion of **evidence-sensitive, coherent transport** (via identity).
- Univalence: **equivalence becomes a source of transport** $\Rightarrow$ C1/C2 internalized.

But "implementation" $\neq$ "automatic eraser"

UF provides a principled mechanism, not a guarantee that all practical burdens disappear.

Limitation 1: propositional vs definitional equality

The gap

Univalence typically yields **propositional** equality (paths), while rewriting/computation in proof assistants often relies on **definitional** equality.

Limitation 1: propositional vs definitional equality

The gap

Univalence typically yields **propositional** equality (paths), while rewriting/computation in proof assistants often relies on **definitional** equality.

- You can transport along equivalences, but it may not compute "by definition".
- So reuse is principled, yet automation can still require work (rewriting steps, lemmas).

Limitation 1: propositional vs definitional equality

The gap

Univalence typically yields **propositional** equality (paths), while rewriting/computation in proof assistants often relies on **definitional** equality.

- You can transport along equivalences, but it may not compute "by definition".
- So reuse is principled, yet automation can still require work (rewriting steps, lemmas).

Up-shot

HoTT/UF improves the *theory of reuse*; engineering smoothness is an additional layer.

Limitation 2: the content-boundary problem remains

C1 is still a design choice

Even in HoTT/UF, "what counts as structural content" depends on:

- which sameness notion you adopt (equivalence, iso in a structure, etc.)
- which predicates you allow (Prop vs Type, truncation levels, etc.)

Limitation 2: the content-boundary problem remains

C1 is still a design choice

Even in HoTT/UF, "what counts as structural content" depends on:

- which sameness notion you adopt (equivalence, iso in a structure, etc.)
- which predicates you allow (Prop vs Type, truncation levels, etc.)

No free lunch

UF does not delete all junk automatically; it gives a cleaner **workshop** to articulate and enforce content constraints.

Limitation 3: "canonical" comes in strengths

Canonical transfer is not one thing

There are different targets:

- **Existence** of transport (weak)
- **Chosen** transport (constructive/canonical as a function)
- **Computational** transport (strong: good definitional behavior)

Limitation 3: “canonical” comes in strengths

Canonical transfer is not one thing

There are different targets:

- **Existence** of transport (weak)
- **Chosen** transport (constructive/canonical as a function)
- **Computational** transport (strong: good definitional behavior)

- HoTT gives coherence robustly; computation friendliness can still be subtle.
- So C2-as-spec often splits into: **coherence** vs **computation**.

Limitation 3: "canonical" comes in strengths

Canonical transfer is not one thing

There are different targets:

- **Existence** of transport (weak)
- **Chosen** transport (constructive/canonical as a function)
- **Computational** transport (strong: good definitional behavior)

- HoTT gives coherence robustly; computation friendliness can still be subtle.
- So C2-as-spec often splits into: **coherence** vs **computation**.

Take-away

UF is a major step, but "structuralist heaven" is an overstatement.

8. Conclusions

Section 8/8

Outline

Conclusions

- Proof assistants expose a genuine tension: **same mathematics, no direct reuse** (representation sensitivity).

Conclusions

- Proof assistants expose a genuine tension: **same mathematics, no direct reuse** (representation sensitivity).
- Benacerraf + Junk motivates reading structuralism as a **constraint on foundational language** (not ontology first).

Conclusions

- Proof assistants expose a genuine tension: **same mathematics, no direct reuse** (representation sensitivity).
- Benacerraf + Junk motivates reading structuralism as a **constraint on foundational language** (not ontology first).
- Structuralist constraints split into:
  - **C1:** invariance of content under $\approx$
  - **C2:** canonical, coherent transfer under $\approx$

Conclusions

- Proof assistants expose a genuine tension: **same mathematics, no direct reuse** (representation sensitivity).
- Benacerraf + Junk motivates reading structuralism as a **constraint on foundational language** (not ontology first).
- Structuralist constraints split into:
  - **C1:** invariance of content under $\approx$
  - **C2:** canonical, coherent transfer under $\approx$
- HoTT/UF provides an **implementation path**: identity $\Rightarrow$ transport, univalence $\Rightarrow$ transport along equivalence.

Conclusions

- Proof assistants expose a genuine tension: **same mathematics, no direct reuse** (representation sensitivity).
- Benacerraf + Junk motivates reading structuralism as a **constraint on foundational language** (not ontology first).
- Structuralist constraints split into:
  - **C1:** invariance of content under $\approx$
  - **C2:** canonical, coherent transfer under $\approx$
- HoTT/UF provides an **implementation path**: identity $\Rightarrow$ transport, univalence $\Rightarrow$ transport along equivalence.

Modest conclusion

UF does not finish structuralism; it turns structuralist constraints into **executable design principles**, while leaving further design choices open.

Thank you.

mail@mincheolseo.com