Beyond Intuition: Nontrivial Equivalences in Category Theory and Homotopy Type Theory

Douglas C. Youvan

doug@youvan.com

August 6, 2025

Mathematics often rewards intuition, but the abstractions of Category Theory and Homotopy Type Theory (HoTT) challenge even the most seasoned thinkers. These frameworks reveal structural equivalences that collapse distinctions deeply ingrained in conventional reasoning. In the Curry–Howard–Lambek correspondence, a proof in logic, a program in computation, and a morphism in a category are not merely analogous—they are the same structure. The Univalence Axiom in HoTT dissolves the traditional boundary between equality and isomorphism, declaring equivalent types to be identical. Equality itself becomes richer, reinterpreted as a path space with higher-dimensional homotopies. Even the empty type is recast as a universal construction, the initial object from which exactly one morphism flows to any other object. Products and coproducts exchange roles in dual categories; objects can be completely reconstructed from their relationships via the Yoneda Lemma. These nonintuitive equivalences do more than surprise—they expand the horizon of what counts as "the same" and invite mathematicians, logicians, and computer scientists to navigate a landscape where intuition must be rebuilt from structural foundations.

Keywords: Category Theory, Homotopy Type Theory, Univalence Axiom, Curry–Howard–Lambek correspondence, Yoneda Lemma, identity types, initial object, products and coproducts, higher groupoids, type theory, logic and computation, structural equivalence. A collaboration with GPT-4o.

I. Introduction

A. Motivation: How abstract mathematics overturns everyday assumptions

Much of mathematical learning builds upon the reinforcement of intuition: numbers behave predictably, shapes follow geometric rules, and algebraic manipulations preserve familiar relationships. Yet at higher levels of abstraction, these expectations can collapse. Category Theory and Homotopy Type Theory (HoTT) expose the fragility of many "obvious" distinctions—such as equality versus equivalence, structure versus content, or syntax versus semantics. The result is often a kind of cognitive dissonance: structures that appeared unrelated turn out to be manifestations of the same principle, while distinctions assumed to be absolute dissolve under the lens of abstraction. These shifts are not merely philosophical; they fundamentally alter the way problems are formulated and solved.

B. Scope: Category Theory and Homotopy Type Theory (HoTT) as frameworks for deep structural equivalences

This paper focuses on equivalences that emerge naturally within Category Theory and HoTT—frameworks designed to distill mathematics to its compositional and type-theoretic essence. Category Theory reveals the structural relationships between mathematical objects through morphisms, universal properties, and dualities, while HoTT integrates type theory with the intuitions of homotopy theory, interpreting types as spaces and equalities as paths. Together, they form a unified language in which disparate domains of mathematics can be compared, merged, and transformed without loss of rigor.

C. The element of surprise: why these equivalences matter in both theory and application

The equivalences discussed here are "nonintuitive" not because they are difficult to prove, but because they directly challenge the mental models formed in earlier stages of mathematical development. Discovering that a computer program, a logical proof, and a categorical morphism are formally identical changes how one reasons about software verification. Accepting that isomorphic objects can be treated as identical in HoTT influences both foundational debates in mathematics

and practical formalization in proof assistants. Even the re-interpretation of equality as a space has ripple effects in topology, logic, and computer science. These insights extend beyond pure theory—they guide programming language design, influence the semantics of computation, and inspire new approaches to unifying mathematical fields.

II. The Curry–Howard–Lambek Correspondence

A. Logic, computation, and category theory as a single structure

The Curry–Howard–Lambek correspondence unites three seemingly distinct realms—formal logic, typed computation, and category theory—under a single structural framework. In propositional logic, an implication $A \Rightarrow B$ expresses that assuming A allows us to deduce B. In typed λ-calculus, a function type $A \to B$ expresses that given a term of type A, one can compute a term of type B. In a cartesian closed category, a morphism $A \to B$ is an arrow from object A to object B satisfying composition laws. The correspondence reveals that these are not analogous in a loose sense—they are mathematically equivalent: propositions are types, proofs are programs, and proofs/programs correspond to morphisms in an appropriate category.

B. Implications for proofs, programs, and morphisms

This identification collapses traditional barriers between disciplines. A proof in constructive logic is simultaneously a program that can be executed and a morphism obeying categorical laws. Conversely, a functional program can be interpreted as a constructive proof of its type signature. This provides a direct bridge between formal verification in computer science, theorem proving in logic, and structural reasoning in mathematics. For example, “running” a proof corresponds to executing a program, while “optimizing” a program corresponds to finding a more efficient proof or morphism.

C. Examples of practical translations between domains

- **Logic → Programming:** A proof of $A \wedge B \Rightarrow B \wedge A$ in natural deduction corresponds to a program that swaps components of a pair, and in category theory to the symmetry morphism of a product object.
- **Programming → Logic:** A typed function in Haskell, `swap :: (A, B) -> (B, A)`, embodies a constructive proof that $A \times B \cong B \times A$.
- **Category Theory → Programming Languages:** The categorical concept of a functor underlies polymorphic types such as Haskell's `map` function, which applies a transformation to every element of a container while preserving structure.

These translations demonstrate that the Curry–Howard–Lambek correspondence is not a metaphorical bridge—it is a formal isomorphism of structures that allows movement between reasoning, computation, and categorical abstraction without loss of meaning.

III. Univalence: Isomorphic = Identical

A. The traditional distinction between equality and isomorphism

In classical mathematics, equality is treated as a strict identity: two objects are equal if they are exactly the same entity. Isomorphism, by contrast, expresses structural sameness without literal identity—two sets, groups, or spaces can be isomorphic if there exists a bijection or structure-preserving map between them, yet they remain distinct objects in the formal sense. This distinction often forces mathematicians to carry around the baggage of "up to isomorphism" disclaimers, even when the differences are purely notational or incidental. For example, $\mathbb{Z}_2$ defined as integers modulo 2 and $\{0, 1\}$ with addition mod 2 are structurally identical as groups but are not strictly equal in traditional set theory.

B. The Univalence Axiom in HoTT

Homotopy Type Theory, with the Univalence Axiom introduced by Vladimir Voevodsky, collapses the gap between equality and equivalence (isomorphism). The axiom states that for types A and B, the identity type $(A = B)$ is equivalent to the type of equivalences $A \simeq B$. In other words, to be equivalent is to be equal, not merely similar. This reframing integrates the principle of "mathematics up to isomorphism" into the foundational logic itself. The universe of types behaves like a homotopy space, where paths between points (equalities) correspond exactly to equivalences.

C. Philosophical and technical consequences

Philosophically, univalence reflects a structuralist stance: objects are defined entirely by their structural relationships, not by their intrinsic "labels" or representations. This aligns closely with the ethos of category theory, where objects are understood through their morphisms. Technically, univalence simplifies formalization in proof assistants such as Coq or Agda, because theorems and definitions no longer need to be duplicated for "isomorphic but not equal" cases—one can freely substitute equivalent types in any context. This also eliminates certain sources of accidental complexity in formal proofs, making them more compositional and modular. The consequence is a foundation where the working mathematician's informal habit of identifying isomorphic structures is not only permissible but logically sanctioned.

IV. The Empty Type and the Initial Object

A. Role of the empty type in type theory

In type theory, the *empty type*—often denoted $\bot$ or Empty—is a type with no inhabitants. It represents an impossible or contradictory proposition: to inhabit it would require proving a falsehood. Because it has no elements, no term can be constructed with this type in a consistent system. Despite its apparent triviality, the empty type plays a crucial role in the structure of type theory. It defines the bottom of the type hierarchy, forming the logical counterpart of "false" in the propositions-as-types paradigm. Its very emptiness makes it a universal source for defining vacuous or absurd functions: from nothing, anything follows.

B. Category-theoretic interpretation as initial object

In category theory, the empty type corresponds to the *initial object*—an object 0 such that for every object A in the category, there exists exactly one morphism $0 \to A$. This single morphism is the "vacuous" map, existing not because of a constructive process but because no counterexample can be produced: there are no elements in the source to map incorrectly. In the category Set, the initial object is the empty set, and the unique function from the empty set to any set is simply

the empty function. This universal property defines the initial object abstractly, without reference to internal structure.

C. Logical principle of *ex falso* and its computational analog

The logical maxim *ex falso quodlibet*—"from falsehood, anything follows"—is the direct logical expression of the empty type's properties. If a proof of ⊥ exists, then for any proposition P, a proof of P can be derived. In computational terms, this corresponds to a function from Empty to any type A. In functional programming languages like Agda or Haskell (with extensions), such a function is defined without cases—there are no possible inputs to handle, so the type checker accepts it without requiring implementation. This captures the idea that an impossible situation trivially satisfies any requirement. The empty type's power, therefore, lies not in what it contains but in how its absence of content forces universal, structure-preserving behavior in logic, computation, and category theory.

V. Products, Coproducts, and Duality

A. Definitions in Set and general categories

In the category **Set**, the *product* $A \times B$ is the familiar Cartesian product: the set of all ordered pairs (a, b) where $a \in A$ and $b \in B$. It comes equipped with projection functions $\pi_1 : A \times B \to A$ and $\pi_2 : A \times B \to B$, and satisfies a universal property: for any object X with morphisms $f : X \to A$ and $g : X \to B$, there exists a unique morphism $\langle f, g \rangle : X \to A \times B$ making the relevant diagrams commute. The *coproduct* $A + B$ in **Set** is the disjoint union: all elements from A tagged as "left" and all elements from B tagged as "right." It has canonical injections $\iota_1 : A \to A + B$ and $\iota_2 : B \to A + B$, and satisfies a dual universal property for morphisms from $A + B$ into a target object. In general categories, these definitions extend abstractly: products are limits of diagrams, coproducts are colimits, and their specifics depend on the category's objects and morphisms.

B. Role reversal in opposite categories

Duality in category theory is achieved by reversing the direction of all morphisms, producing the *opposite category* $\mathcal{C}^{op}$. Under this reversal, every statement about limits becomes a statement about colimits, and vice versa. In particular, products in $\mathcal{C}$ become coproducts in $\mathcal{C}^{op}$, and coproducts in $\mathcal{C}$ become products in $\mathcal{C}^{op}$. This role reversal illustrates that "product" and "coproduct" are not inherently tied to multiplication and addition in the everyday sense, but are instead determined entirely by morphism structure and universal properties. For example, in **Set**, the product and coproduct look very different—ordered pairs vs. tagged unions—but categorically they are dual manifestations of the same pattern when viewed through the lens of arrow reversal.

C. Structural vs. material differences in mathematical objects

The categorical perspective shifts focus from the "material" content of objects (e.g., the elements of a set) to their "structural" role in a network of morphisms. Two objects may look very different at the element level yet play categorically identical roles if their universal properties match. The product–coproduct duality emphasizes that much of mathematics can be understood purely structurally: it is the existence and uniqueness of certain morphisms—not the internal representation of the objects—that determines their identity in the category. This abstraction is powerful because it applies equally well in categories with no element-like notion at all, such as categories of topological spaces, groups, or even categories themselves. The material differences only matter when working within a specific concrete category, but structurally, product and coproduct are just two sides of the same universal coin.

VI. Identity Types as Paths

A. Interpretation of equality as a space in HoTT

In traditional logic and set theory, equality is a binary relation: two elements are either equal or they are not. Homotopy Type Theory (HoTT) replaces this static conception with a richer interpretation—equality between two elements $x, y : A$ is itself a *type*, denoted $x =_A y$. The inhabitants of this type are called *identifications* or *paths*, and each such path can be thought of as a continuous deformation from x to y in the "space" represented by A. This viewpoint is inspired by homotopy theory in topology, where paths between points in a space encode fundamental connectivity information. Equality is thus no longer a truth value, but a mathematical object that can be examined, manipulated, and transformed in its own right.

B. Higher paths and homotopies between homotopies

The path-based interpretation of equality in HoTT naturally extends into *higher-dimensional* structures. Just as points can be connected by paths, paths themselves can be connected by *homotopies*—continuous deformations between one path and another. These homotopies can have their own higher homotopies, leading to a hierarchy of identifications:

- 0-paths: points (elements of a type)
- 1-paths: identifications between points (terms of $x=y$ $x = y$)
- 2-paths: identifications between identifications (homotopies)
- n-paths: higher identifications, continuing without bound.
  This infinite hierarchy mirrors the structure of ∞-groupoids, where objects, morphisms, 2-morphisms, and higher morphisms coexist with coherent composition rules. Types in HoTT are thus understood as ∞-groupoids, making higher-dimensional symmetry and equivalence intrinsic to the foundations.

C. Consequences for mathematics as synthetic homotopy theory

By internalizing the language of paths and homotopies into type theory, HoTT becomes a *synthetic* approach to homotopy theory—meaning that theorems about spaces can be derived directly from type-theoretic reasoning without recourse to external topological models. This unification has profound implications: topology and logic are no longer separate domains linked by analogy, but are two aspects of the same foundational framework. For example, the fundamental group of a type emerges from the structure of its 1-paths and 2-paths, while higher homotopy groups are encoded in higher paths. This makes HoTT not just an alternative foundation for mathematics, but one that bakes in geometric and homotopical intuition at the ground level, enabling proofs and constructions that would otherwise require intricate topological machinery to appear naturally in the logic itself.

VII. The Yoneda Lemma: Objects as Relationships

A. Statement and intuition behind the lemma

The Yoneda Lemma is one of the central results in category theory, often summarized as: *an object is completely determined by how all other objects map into it.* Formally, for a locally small category $\mathcal{C}$, the lemma states that for any object A and any functor $F : \mathcal{C} \to \mathbf{Set}$, there is a natural isomorphism

$$\mathrm{Nat}(\mathrm{Hom}(-, A), F) \cong F(A).$$

Here, $\mathrm{Hom}(-, A)$ is the *representable functor* mapping each object X to the set of morphisms $X \to A$. The intuition is that by knowing every possible morphism into A (and how those morphisms compose), you have enough information to fully reconstruct the behavior of A within the category—without ever "looking inside" it. This makes Yoneda a categorical embodiment of the idea that "you are defined by your relationships."

B. Reconstructing objects from hom-sets

In practical terms, the Yoneda Lemma asserts that the representable functor $\mathrm{Hom}(-, A)$ encodes all categorical information about A. If two objects have naturally isomorphic hom-functors, then they are isomorphic as objects. This principle allows one to replace reasoning about an object directly with reasoning about the functor of morphisms into it, often a simpler or more tractable perspective. In concrete categories like **Set**, the set $\mathrm{Hom}(X, A)$ is just the set of functions from X to A, and the natural transformations to other functors preserve the algebraic and compositional structure inherent in A. Thus, by analyzing how an object interacts with all possible "test objects" X, one can recover the structure of A itself.

C. Applications to logic, computation, and topology

In logic, Yoneda can be interpreted as a generalization of the idea that the meaning of a proposition is given by the collection of all contexts in which it can be proven. In computation, it underlies the concept of "free theorems" in functional programming: polymorphic functions are constrained entirely by their type signatures because those signatures correspond to natural transformations between functors. For example, in Haskell, the type `forall a. [a] -> [a]` implies that the function cannot inspect or alter the elements—only rearrange them—because its behavior is determined solely by morphism structure. In topology, the lemma supports the use of representable functors in algebraic topology, such as in the definition of cohomology theories, where spaces are studied entirely through the continuous maps into them. Across domains, Yoneda reframes the essence of

an entity as the totality of its interactions, providing a powerful unifying perspective that sidesteps the need for internal inspection.

VIII. Distributivity and Its Limits

A. The familiar algebraic distributive law in Set

In the category Set, products and coproducts correspond to familiar operations on sets: products are Cartesian products, and coproducts are disjoint unions. The distributive law familiar from basic algebra holds:

$$A \times (B + C) \cong (A \times B) + (A \times C).$$

This isomorphism can be seen explicitly: pairing an element $a \in A$ with either a left-tagged element $b \in B$ or a right-tagged element $c \in C$ is equivalent to having either a left-tagged pair (a, b) or a right-tagged pair (a, c). This correspondence is bijective, natural in all three variables, and relies on the "element-wise" construction possible in **Set**. In such a setting, distributivity feels inevitable because it directly reflects our everyday algebraic experience with multiplication over addition.

B. Categorical perspective on when distributivity holds

From a categorical standpoint, distributivity is not automatic—it is a special property of certain categories. A category is called *distributive* if it has finite products and finite coproducts, and the canonical morphism

$$A \times (B + C) \to (A \times B) + (A \times C)$$

is an isomorphism for all A, B, C. This property depends entirely on the interaction between the universal properties of products and coproducts, not on any element-level reasoning. Many familiar categories, such as **Set** and **Poset**, are distributive. However, distributivity in this abstract sense is not guaranteed even in categories that have both products and coproducts—extra structure or constraints are often required to ensure it. The categorical viewpoint thus generalizes the algebraic law, while making clear that it is contingent, not universal.

C. Counterexamples and structural failures

There are well-known categories in which distributivity fails. For example, in the category of commutative rings (**CRing**), the product is the Cartesian product of rings, while the coproduct is the tensor product over $\mathbb{Z}$. These operations do not satisfy distributivity in the sense above, because the interaction between ring multiplication and addition of ideals does not yield a canonical isomorphism between $R \times (S \oplus T)$ and $(R \times S) \oplus (R \times T)$. Similarly, in categories with non-cartesian monoidal products—such as **Vect** (vector spaces over a field) when considered with the tensor product as monoidal structure—the coproduct (direct sum) does not generally distribute over the tensor product without additional assumptions. These counterexamples show that distributivity is not a formal inevitability, but a property tied to specific structural compatibilities. Recognizing its failure is crucial in advanced settings, as many intuitive "algebraic" manipulations valid in **Set** break down elsewhere, forcing a more careful, property-driven approach to reasoning.

IX. Negative Types and Logical Negation

A. The type-theoretic encoding of negation as functions to the empty type

In type theory, particularly under the propositions-as-types correspondence, logical negation is encoded in an elegant and uniform way. If A is a type, its negation $\neg A$ is defined as the function type $A \to \bot$, where $\bot$ is the empty type. Intuitively, to "prove" $\neg A$ means to demonstrate that any assumed instance of A would lead to a contradiction—i.e., to a term of the empty type, which is impossible in a consistent system. This definition mirrors the logical idea that $\neg A$ asserts the impossibility of A, but it is expressed entirely in constructive, type-theoretic terms. No special logical operator is required; negation arises from the same function-space construct that describes ordinary mappings between types.

B. Relation to classical and constructive logic

In classical logic, negation interacts with the Law of Excluded Middle (LEM), which states $A \vee \neg A$ for all propositions A. However, in constructive type theory, LEM is not assumed by default. Here, $\neg A$ still means $A \to \bot$, but $\neg\neg A$ does not automatically collapse to A unless one assumes classical principles such as double-negation elimination. This subtle difference leads to richer distinctions between "refuting A" and "constructively producing A." In HoTT, this constructive nature is preserved even in higher-type settings, where negation can apply to propositions about paths, homotopies, and higher structures. Thus, the type-theoretic encoding of negation integrates seamlessly into both constructive and classical frameworks, with the choice of logic determining how far equivalences like $\neg\neg A \Rightarrow A$ can be pushed.

C. Implications for programming languages and proof assistants

In functional programming languages grounded in type theory (e.g., Agda, Idris, Coq), the encoding $\neg A = A \to \bot$ has direct computational meaning. It says that a program of type $\neg A$ is a function that, if given an input of type A, can produce an output in the empty type—which is impossible to construct unless the program can exploit a contradiction in A's structure. This makes negation a tool for *proof by refutation* in mechanized reasoning: to prove that a type is uninhabited, one writes a function that maps any would-be inhabitant to a contradiction. In practice, this enables proof assistants to unify logical and computational reasoning: the same mechanisms that ensure type safety in programs also guarantee logical soundness in proofs. Moreover, because negation is just a function type, it enjoys all the compositional and higher-order properties of ordinary functions, making it a flexible and powerful building block for formal reasoning systems.

X. Types as ∞-Groupoids

A. HoTT's interpretation of types as higher-dimensional groupoids

One of the foundational shifts in Homotopy Type Theory (HoTT) is the interpretation of types as ∞-*groupoids*. In this view, a type is not just a collection

of elements—it comes equipped with a rich hierarchy of identifications. At the base level, elements of a type are the *points* (0-cells). Equalities between points are *paths* (1-cells), equalities between paths are *homotopies* (2-cells), and this continues indefinitely to higher and higher identifications. These identifications satisfy groupoid-like properties (existence of inverses, composition, and identity morphisms) at every level. The "∞" in ∞-groupoid reflects the fact that this hierarchy extends without bound, making every type a higher-dimensional algebraic object, not merely a set. This perspective dissolves the traditional boundary between equality and structure, revealing that even "sameness" is itself structured and multi-layered.

B. Equivalence between type theory and homotopy theory

Under the *homotopy interpretation* of type theory, types correspond to spaces, elements to points, identity types to path spaces, and higher identity types to higher homotopies. This correspondence is not just metaphorical—it is formal: Martin-Löf type theory can be interpreted in homotopy-theoretic models such as Kan simplicial sets, where the rules of type theory exactly mirror the rules for manipulating spaces up to homotopy. Conversely, one can work *synthetically* in HoTT and recover theorems of classical homotopy theory without ever referring to point-set topology. The Univalence Axiom strengthens this bridge by identifying equivalence of types with equality, ensuring that the type-theoretic universe behaves like a homotopy-invariant mathematical cosmos.

C. Unification of logic, topology, and algebra

The ∞-groupoid interpretation allows logic, topology, and algebra to be expressed in a single coherent framework. In logic, propositions are types with at most one point up to homotopy (*h-propositions*), and proofs are paths. In topology, types represent spaces, with homotopy groups arising naturally from iterated identity types. In algebra, ∞-groupoids capture higher-categorical symmetries and coherence data found in structures such as braided monoidal categories and higher operads. This unification means that theorems about paths and homotopies in topology are directly usable as logical lemmas, while algebraic coherence conditions emerge from the same equality-as-path framework. HoTT's treatment of types as ∞-groupoids thus does more than reinterpret

mathematics—it provides a universal stage on which logic, geometry, and algebra are facets of the same underlying structure, allowing results in one domain to translate, often unexpectedly, into others.

XI. Conclusion

A. Summary of surprising equivalences

Category Theory and Homotopy Type Theory reveal a series of deep structural identifications that overturn familiar distinctions from everyday mathematics. The Curry–Howard–Lambek correspondence equates proofs, programs, and morphisms; the Univalence Axiom collapses the boundary between equality and equivalence; and identity types reinterpret equality as a hierarchy of paths and higher homotopies. The empty type emerges as the initial object, embodying the principle of *ex falso*, while products and coproducts reveal a duality that becomes literal in opposite categories. The Yoneda Lemma reframes objects as nothing more (and nothing less) than their relationships, distributivity shows itself to be a contingent property rather than a universal truth, and logical negation appears as a simple function type to the empty type. Finally, the view of types as ∞-groupoids unites logic, topology, and algebra in a single conceptual framework.

B. Reflection on the reshaping of intuition through abstract frameworks

These equivalences are "nonintuitive" precisely because they clash with deeply embedded conceptual habits. In traditional mathematical education, equality is absolute, objects are examined internally, and operations are interpreted in concrete terms. Abstract frameworks like Category Theory and HoTT demand that intuition be rebuilt from structural properties rather than element-level content. Over time, these abstractions replace the old mental models with a more flexible, compositional understanding—one where apparent differences between domains collapse into expressions of the same universal patterns. This shift is not merely aesthetic; it alters how one approaches problem-solving, proof design, and even the boundaries between disciplines.

C. Potential research directions: bridging deep theory with practical computation
The next frontier lies in translating these theoretical insights into tools that can be applied at scale in mathematics, computer science, and beyond. Proof assistants grounded in HoTT, such as Coq and Agda, already demonstrate how univalence, higher-dimensional identity, and constructive negation can make formal verification both more expressive and more faithful to working mathematicians' habits. In programming language theory, categorical semantics can inform the design of languages that natively reflect logical and homotopical structure. In pure mathematics, the unification of logic, topology, and algebra under the ∞-groupoid paradigm offers unexplored pathways for synthetic reasoning. Bridging these deep theoretical results with high-performance automated reasoning systems, domain-specific languages, and large-scale formalization projects will not only expand the reach of abstract mathematics but also embed its insights directly into the computational infrastructure of science and engineering.

Epilogue: Human Perception and Thought Using Category Theory and Homotopy Type Theory (HoTT)

---

1. Objects Would Be Seen Only Through Their Relationships
Instead of perceiving "things" as self-contained entities, humans would naturally see them as nodes in a web of morphisms. An apple wouldn't be "an apple" in isolation; it would be the intersection of functions, mappings, and transformations—its color in light-space, its role in a food chain, its position in economic and cultural categories. The *Yoneda Lemma* wouldn't be an abstract theorem; it would be their perceptual reality.

---

2. Equality Would Be Path-Based, Not Binary
In their intuition, "sameness" would be about the existence of a morphism or path connecting two states, rather than strict, unqualified equality. People would accept *degrees* and *layers* of identity (as in HoTT's identity types), understanding

that two “equal” things may be equal in some dimensions and distinct in others, and that higher equalities (paths between paths) also matter. This would make disagreement and nuance far easier to navigate socially.

---

3. Logic Would Be Structural, Not Propositional
Negation wouldn’t feel like an arbitrary truth operator—it would be experienced as an actual *mapping to impossibility* (a function to the empty type). Contradictions wouldn’t just be “wrong”; they would feel like dead ends in a type space. Proofs would be constructed more like programs or geometric constructions than like verbal arguments.

---

4. Time and Change Would Be Viewed as Morphism Composition
Causality and process would be naturally represented as the composition of arrows. Instead of a linear chain of events, history and prediction would be perceived as a category of objects (states) with morphisms (transformations) connecting them. Loops, commuting diagrams, and universal properties would be *intuitive*.

---

5. Intuition Would Extend to Higher Dimensions
People would comfortably think in ∞-groupoids, where entities have not just properties, but properties of their properties, and relationships between those. They would reason about multiple layers of symmetry, deformation, and equivalence without needing explicit formalism—much like a geometer can see a rotation as “the same” object in a different orientation.

---

6. Philosophy and Science Would Merge
The distinction between “what is” and “how we reason about it” would blur. Category theory’s abstraction and HoTT’s constructive rigor would unify metaphysics, epistemology, and formal science into one discipline. Discoveries in

topology or type theory would feel as relevant to ethics and aesthetics as they are to engineering.

---

7. Social and Ethical Implications

Such humans might find it harder to cling to absolutist dogmas, since univalence teaches that equivalence *is* identity in the right context. At the same time, they could develop extremely precise, context-dependent moral and social frameworks—where “same” and “different” are never naively conflated.

References

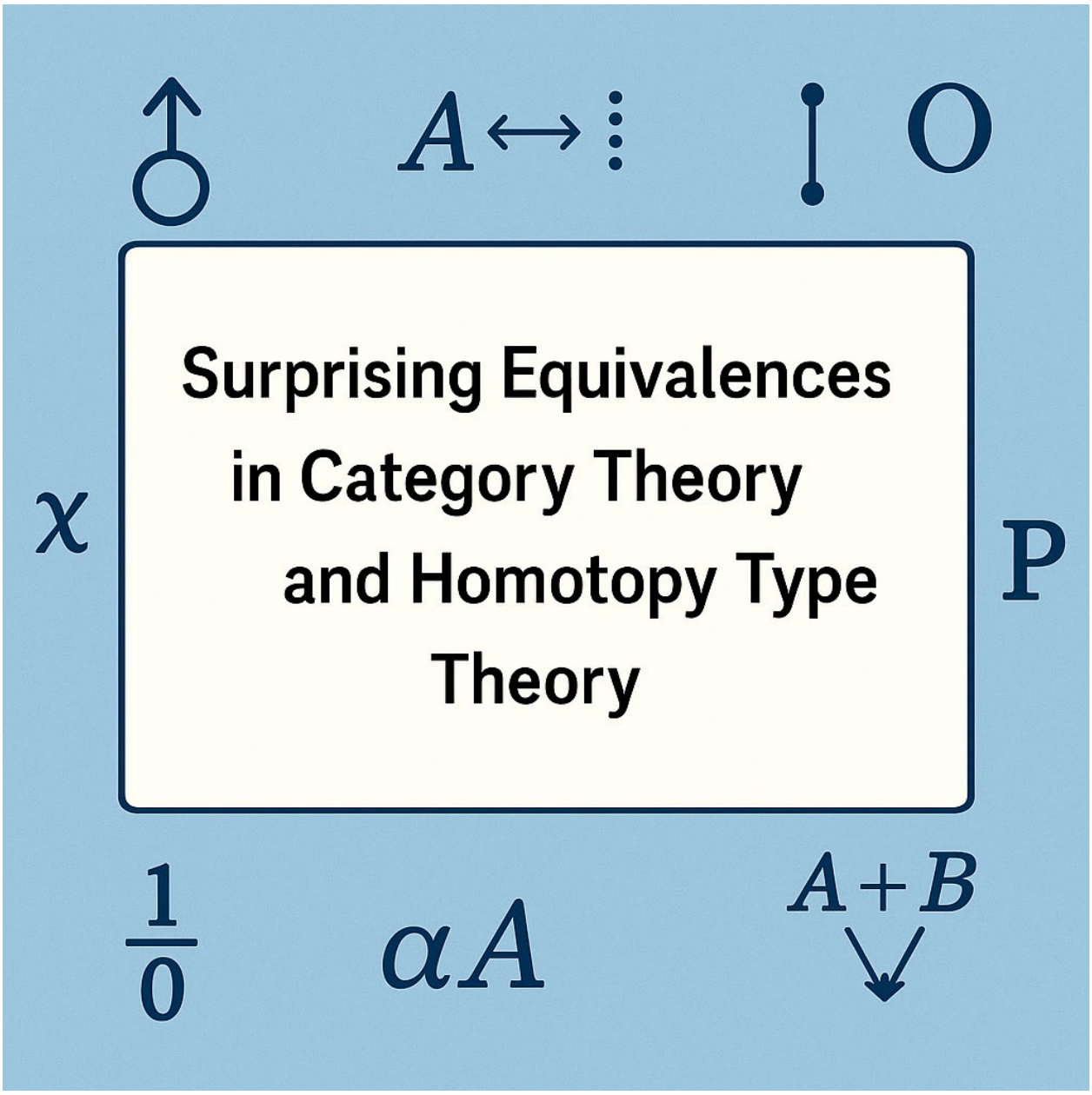
1. Awodey, S. (2010). *Category Theory* (2nd ed.). Oxford University Press.
2. Awodey, S. (2014). *Type Theory and Homotopy*. In: *Proceedings of the International Congress of Mathematicians* (ICM 2014), Vol. II, 1–29.
3. Awodey, S., & Warren, M. A. (2009). Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 146(1), 45–55.
4. Bartosz, M. (2019). *Category Theory for Programmers*. Leanpub.
5. Curry, H. B., & Feys, R. (1958). *Combinatory Logic* (Vol. 1). North-Holland.
6. Howard, W. A. (1980). The formulas-as-types notion of construction. In J. P. Seldin & J. R. Hindley (Eds.), *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism* (pp. 479–490). Academic Press.
7. Lambek, J., & Scott, P. J. (1986). *Introduction to Higher-Order Categorical Logic*. Cambridge University Press.
8. Lawvere, F. W., & Schanuel, S. H. (1997). *Conceptual Mathematics: A First Introduction to Categories*. Cambridge University Press.
9. Leinster, T. (2014). *Basic Category Theory*. Cambridge University Press.
10. Mac Lane, S. (1998). *Categories for the Working Mathematician* (2nd ed.). Springer.
11. Rijke, E., Shulman, M., & Spitters, B. (2022). Modalities in homotopy type theory. *Logical Methods in Computer Science*, 18(2), 1–57.
12. Shulman, M. (2015). Univalence for inverse diagrams and homotopy canonicity. *Mathematical Structures in Computer Science*, 25(5), 1203–1277.
13. Streicher, T. (1993). *Semantics of Type Theory: Correctness, Completeness, and Independence Results*. Birkhäuser.

14. The Univalent Foundations Program. (2013). *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study.

15. Voevodsky, V. (2015). An experimental library of formalized mathematics based on the univalent foundations. *Mathematical Structures in Computer Science*, 25(5), 1278–1294.

16. Wadler, P. (2015). Propositions as types. *Communications of the ACM*, 58(12), 75–84.

17. Yoneda, N. (1960). On Ext and exact sequences. *Journal of the Faculty of Science, University of Tokyo. Section I*, 8, 507–576.


$A \leftrightarrow$
O
Surprising Equivalences in Category Theory and Homotopy Type Theory
χ
P
$\frac{1}{0}$
αA
$A + B$