Homotopical trinitarianism: A perspective on homotopy type theory

Michael Shulman[1]

[1]University of San Diego

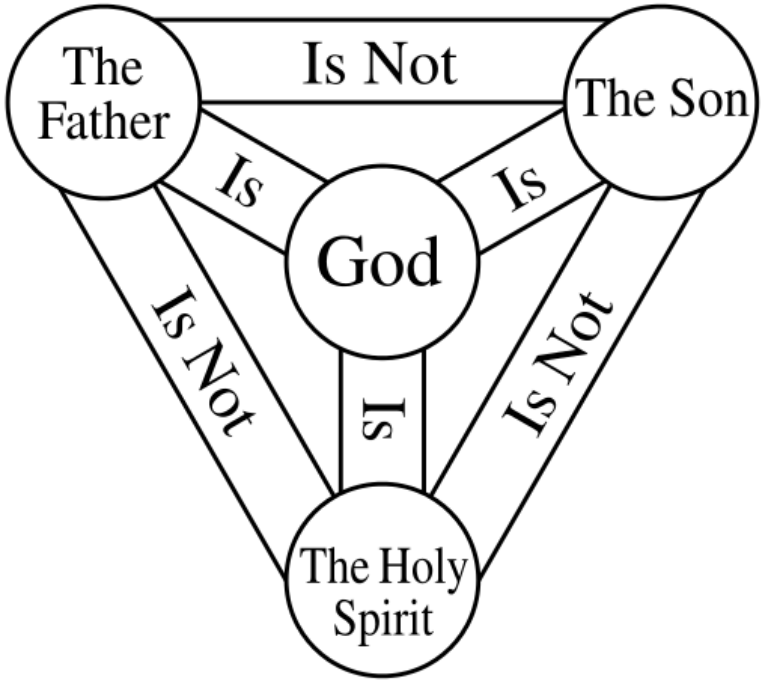
Thursday, January 11, 2018
Joint Mathematics Meetings
San Diego, CA

This talk is dedicated to the memory of
Vladimir Voevodsky (1966–2017)

without whose insight and courage
we would not be where we are today.

Outline

Christian trinitarianism



Go, therefore, and make disciples of all nations, baptizing them in the name of the Father, and of the Son, and of the Holy Spirit. *Matthew 28:19*

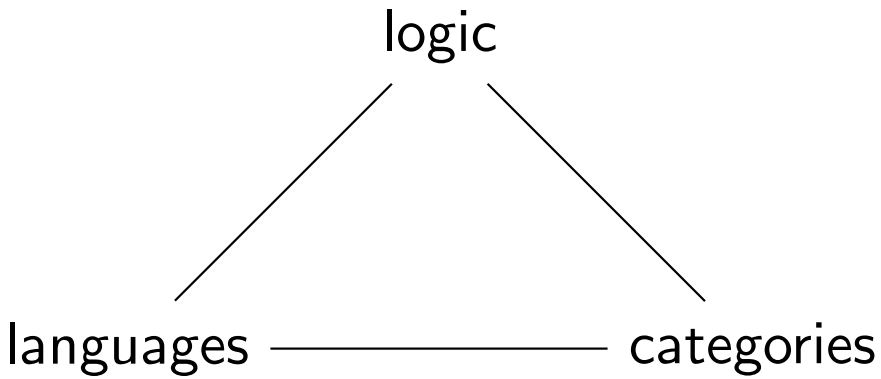
Other triple deities

> *Triads of gods appear very early, at the primitive level. . . . Arrangement in triads is an archetype in the history of religion, which in all probability formed the basis of the Christian Trinity.*
>
> *C. G. Jung. A Psychological Approach to the Dogma of the Trinity.*

- Brahma, Vishnu, Shiva
- Om, Tat, Sat
- Zeus, Poseidon, Hades
- Osiris, Isis, Horus
- Odin, Freyr, Thor
- Aglaea, Euphrosyne, Thalia
- Alekto, Megaera, Tilphousia
- . . .

And in popular culture

Computational trinitarianism

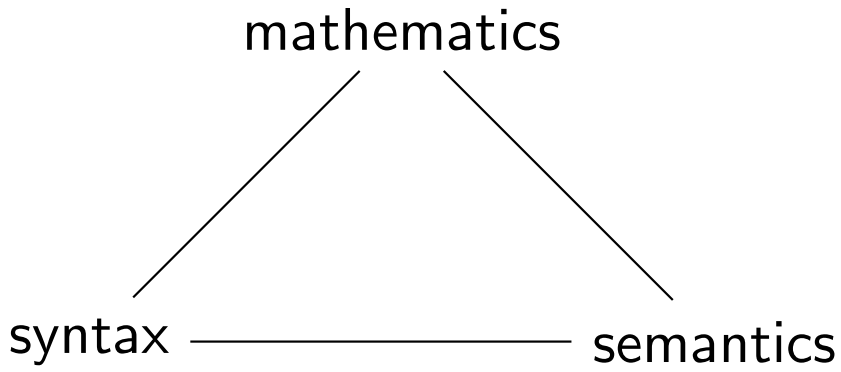


The central dogma of computational trinitarianism holds that Logic, Languages, and Categories are but three manifestations of one divine notion of computation. . . .

. . . any concept arising in one aspect should have meaning from the perspective of the other two. If you arrive at an insight that has importance for logic, languages, and categories, then you may feel sure that you have elucidated an essential concept of computation—you have made an enduring scientific discovery.

–Bob Harper

Computational trinitarianism

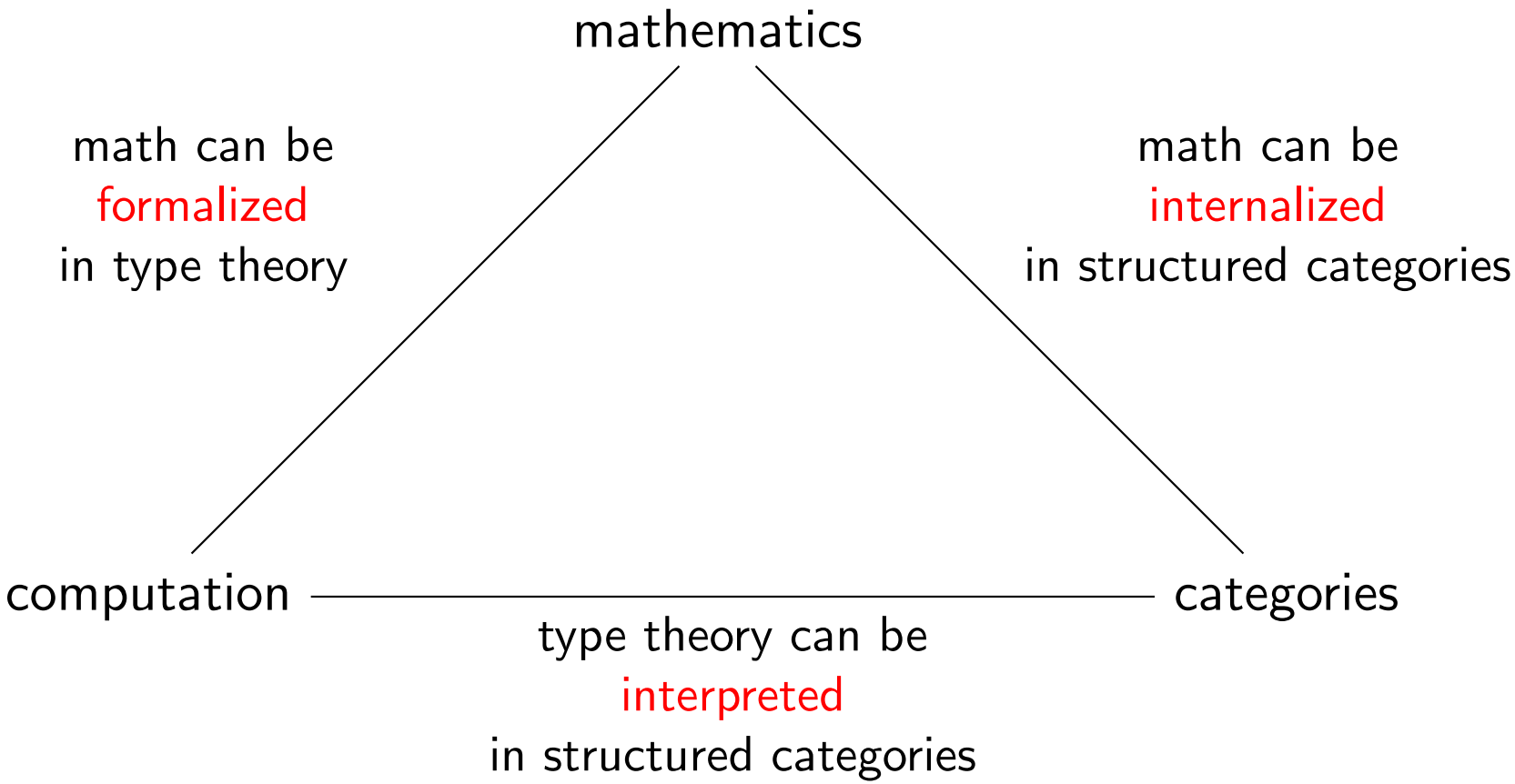


The central dogma of computational trinitarianism holds that Logic, Languages, and Categories are but three manifestations of one divine notion of computation. . . .

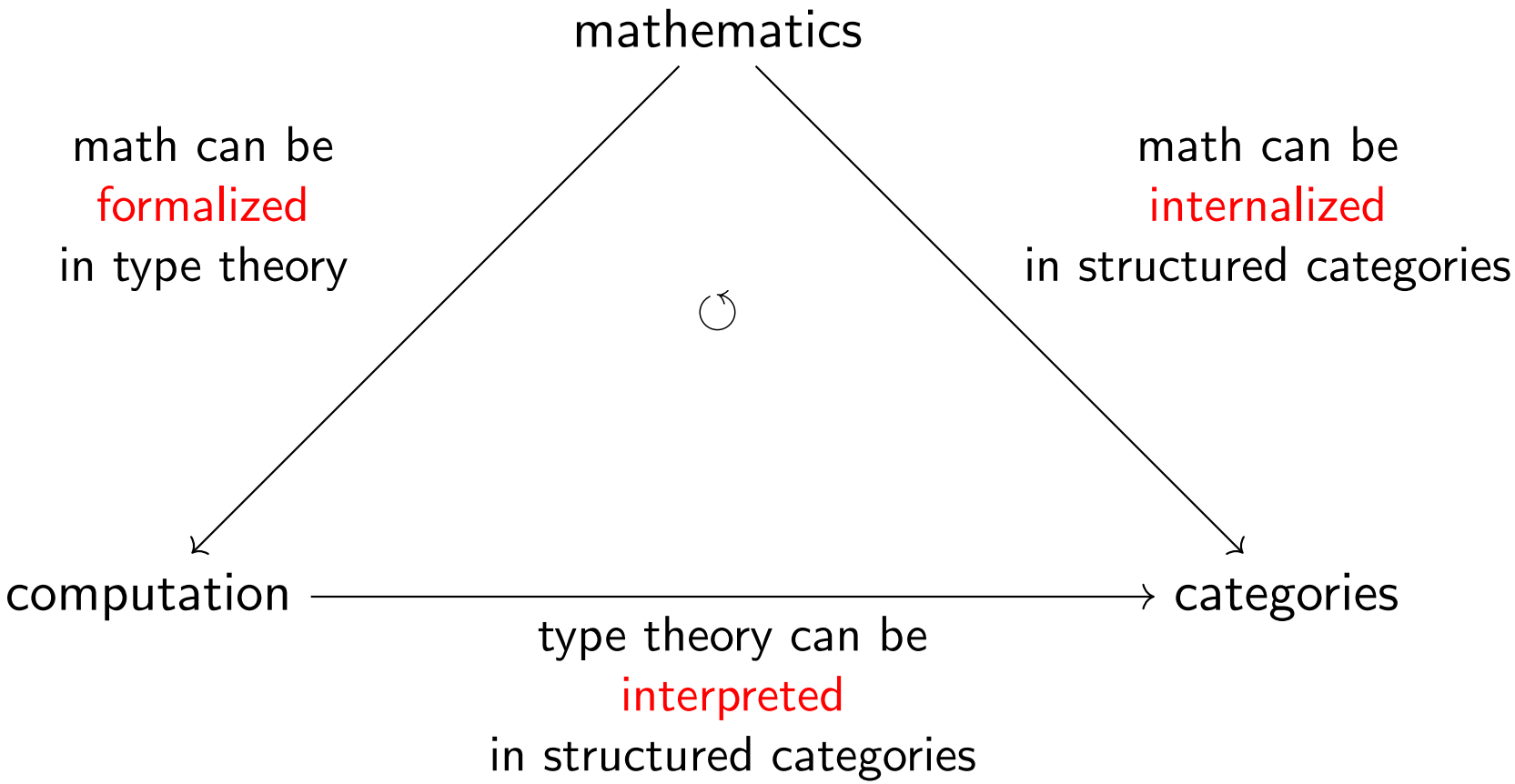
. . . any concept arising in one aspect should have meaning from the perspective of the other two. If you arrive at an insight that has importance for logic, languages, and categories, then you may feel sure that you have elucidated an essential concept of computation—you have made an enduring scientific discovery.

–Bob Harper

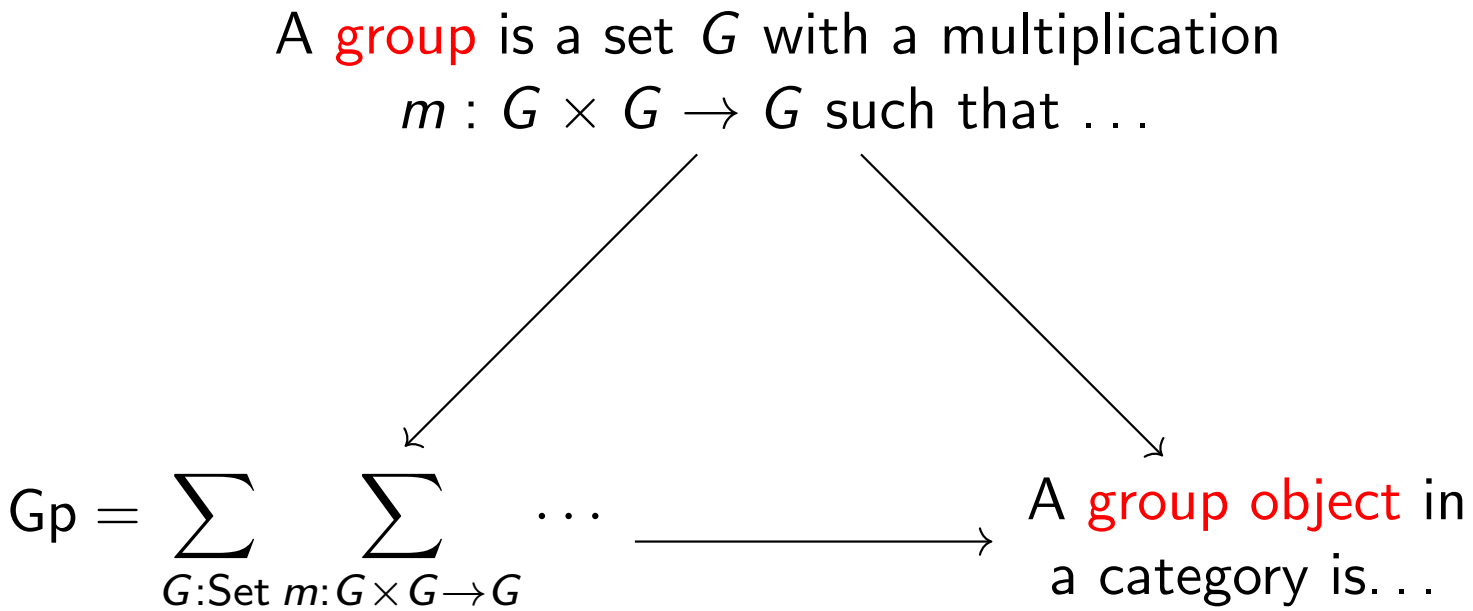
Computational trinitarianism



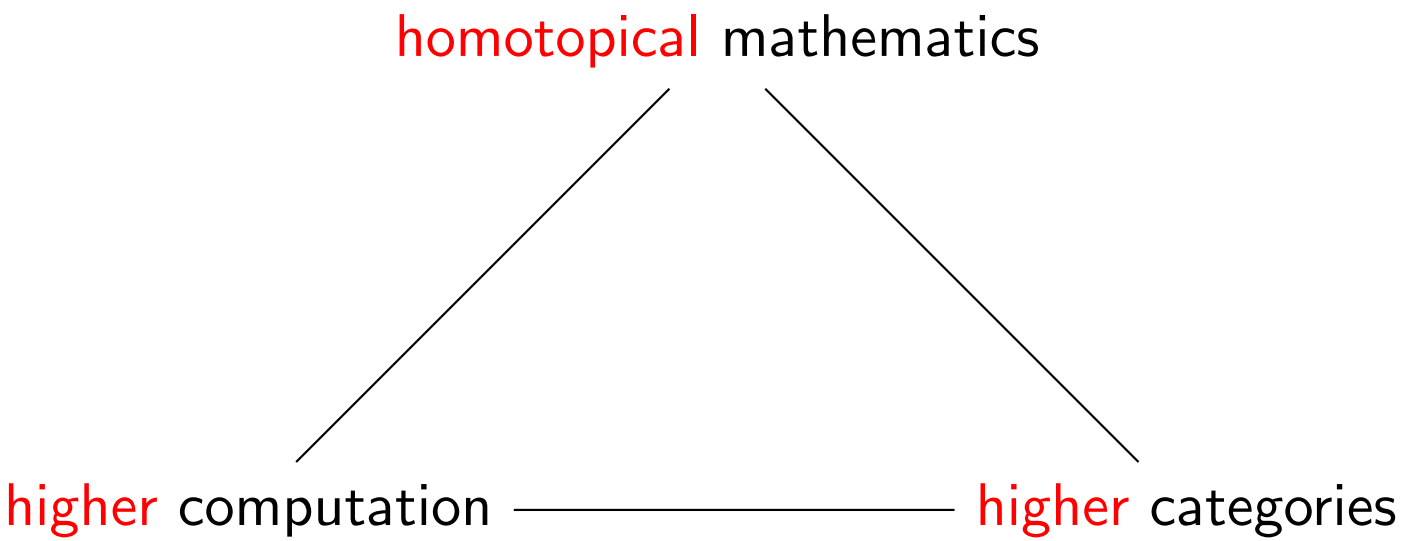
Computational trinitarianism



A trinitarian example



Homotopical Trinitarianism



Outline

Prehistory: type theory before homotopy

- Russell: the first type theory
- Brouwer: constructive critique of classical mathematics
- Church: λ-calculus, a formal model of computation
- LISP, ML, Haskell: programming languages based on λ-calc
- Curry, Howard: propositions as types, logic is type theory
- Bishop: practical constructive mathematics
- Martin-Löf: constructive dependent type theory
- PRL: type theory as a programming language
- Martin-Löf, Constable: types that come with equality
- Constable, Awodey–Bauer: squash/bracket types
- Lawvere-Tierney: internal logic of elementary toposes
- Seely: Martin-Löf type theory in LCC categories
- Coq, Agda: proof assistants using Martin-Löf type theory
- Odd-order, Flyspeck: formalized mathematics in type theory

Prehistory: type theory before homotopy

- Russell: the first type theory
- Brouwer: constructive critique of classical mathematics
- Church: λ-calculus, a formal model of computation
- LISP, ML, Haskell: programming languages based on λ-calc
- Curry, Howard: propositions as types, logic is type theory
- Bishop: practical constructive mathematics
- Martin-Löf: constructive dependent type theory
- PRL: type theory as a programming language
- Martin-Löf, Constable: types that come with equality
- Constable, Awodey–Bauer: squash/bracket types
- Lawvere-Tierney: internal logic of elementary toposes
- Seely: Martin-Löf type theory in LCC categories
- Coq, Agda: proof assistants using Martin-Löf type theory
- Odd-order, Flyspeck: formalized mathematics in type theory

Prehistory: type theory before homotopy

- Russell: the first type theory
- Brouwer: constructive critique of classical mathematics
- Church: λ-calculus, a formal model of computation
- LISP, ML, Haskell: programming languages based on λ-calc
- Curry, Howard: propositions as types, logic is type theory
- Bishop: practical constructive mathematics
- Martin-Löf: constructive dependent type theory
- PRL: type theory as a programming language
- Martin-Löf, Constable: types that come with equality
- Constable, Awodey–Bauer: squash/bracket types
- Lawvere-Tierney: internal logic of elementary toposes
- Seely: Martin-Löf type theory in LCC categories
- Coq, Agda: proof assistants using Martin-Löf type theory
- Odd-order, Flyspeck: formalized mathematics in type theory

Prehistory: type theory before homotopy

- Russell: the first type theory
- Brouwer: constructive critique of classical mathematics
- Church: λ-calculus, a formal model of computation
- LISP, ML, Haskell: programming languages based on λ-calc
- Curry, Howard: propositions as types, logic is type theory
- Bishop: practical constructive mathematics
- Martin-Löf: constructive dependent type theory
- PRL: type theory as a programming language
- Martin-Löf, Constable: types that come with equality
- Constable, Awodey–Bauer: squash/bracket types
- Lawvere-Tierney: internal logic of elementary toposes
- Seely: Martin-Löf type theory in LCC categories
- Coq, Agda: proof assistants using Martin-Löf type theory
- Odd-order, Flyspeck: formalized mathematics in type theory

A lightning summary of dependent type theory

- The basic objects are types and their elements, written $t : A$.
- Each element belongs intrinsically to a unique type.
- Types are formed by operations $A \times B$, $A + B$, $A \to B$, etc.
- Types themselves are the elements of some universe type $\mathcal{U}$.
- A proposition (statement) is represented by a type, whose element (if any) is a "witness" of its truth.
- Logical operations $\wedge, \vee, \Rightarrow, \exists, \forall$ are induced by type operations $\times, +, \to, \sum, \prod$.
- The syntax of elements is also a programming language that can be executed, and used by a computer to verify proofs.
- Categorical semantics interprets types as objects, elements as morphisms (whose domain is the *context*), and propositions as subobjects.

The problem of equality

Originally, type theorists thought of types as behaving like the sets in formal set theory.

Question

By propositions-as-types, "$x = y$" is a type. What is it?

A priori, we would like it to satisfy:

- Proof-irrelevance: can't be "two different equalities" $x = y$.
- Function extensionality: if $f(x) = g(x)$ for all x, then $f = g$.
- Propositional extensionality: if $P \Rightarrow Q$ and $Q \Rightarrow P$ for propositions P, Q, then $P = Q$.
- Function comprehension: if for all x there is a unique y such that $R(x, y)$, then there is f such that $R(x, f(x))$ for all x.

We can assert all of these as axioms, but doing so breaks the computational nature of type theory.

Approaches to equality

1. Intensional Type Theory: Equality is inductively generated by reflexivity ("the smallest reflexive relation"). Not proof-irrelevant, no function or propositional extensionality.
2. Variations: Proof-irrelevant, still no function or propositional extensionality.
3. PRL and Observational Type Theory: Equality is defined recursively over types. Proof-irrelevant, function extensionality. Can get propositional extensionality for a separate type of propositions, but that breaks function comprehension.

None satisfactorily answers: what is an equality between types?

(Note the "axiom of extensionality" from ZFC doesn't apply: an element of one type can never be an element of another type.)

Categorical and homotopical model

Theorem (Hoffmann–Streicher, 1998)

Intensional type theory has a model in groupoids, with $x = y$ interpreted as $\hom(x, y)$, which can have more than one element.

Theorem (Awodey–Warren, 2009)

The inductive generation of equality by reflexivity is the same as the lifting property of a path object in a Quillen model category.

In particular, intensional type theory should have a model in any sufficiently nice Quillen model category, including groupoids but also "homotopy spaces" (∞-groupoids).

Outline

The need for formalization

From *The Origins and Motivations of Univalent Foundations:*

...my paper "Cohomological Theory of Presheaves with Transfers,"... was written... in 1992–93. [Only] In 1999–2000... did I discover that the proof of a key lemma in my paper contained a mistake and that the lemma, as stated, could not be salvaged....

This story got me scared. Starting from 1993, multiple groups of mathematicians studied my paper at seminars and used it in their work and none of them noticed the mistake.... A technical argument by a trusted author, which is hard to check and looks similar to arguments known to be correct, is hardly ever checked in detail.

Higher-categorical mathematics

From *The Origins and Motivations of Univalent Foundations:*

I was working on a new development, which I called 2-theories, [and] getting more and more uncertain about how to proceed. . . . I really enjoyed discovering new structures that were not direct extensions of structures in lower dimensions.

But to do the work at the level of rigor and precision I felt was necessary would take an enormous amount of effort and would produce a text that would be very hard to read. And who would ensure that I did not forget something and did not make a mistake, if even the mistakes in much more simple arguments take years to uncover?

Towards univalent foundations

From *The Origins and Motivations of Univalent Foundations:*

. . . it soon became clear that the only long-term solution was somehow to make it possible for me to use computers to verify my abstract, logical, and mathematical constructions. . . . The primary challenge that needed to be addressed was that the foundations of mathematics were unprepared for the requirements of the task.. . . existing foundations could not be used to directly express statements about such objects as, for example, the ones in my work on 2-theories.

Voevodsky as a trinitarian

From *The Origins and Motivations of Univalent Foundations:*

. . . let me suppose that any foundation for mathematics adequate both for human reasoning and for computer verification should have the following three components.

The first component is a formal deduction system: a language and rules of manipulating sentences in this language that are purely formal, such that a record of such manipulations can be verified by a computer program. The second component is a structure that provides a meaning to the sentences of this language in terms of mental objects intuitively comprehensible to humans. The third component is a structure that enables humans to encode mathematical ideas in terms of the objects directly associated with the language.

Voevodsky as a trinitarian

From *The Origins and Motivations of Univalent Foundations:*

. . . let me suppose that any foundation for mathematics adequate both for human reasoning and for computer verification should have the following three components.

The first component is a formal deduction system: a language and rules of manipulating sentences in this language that are purely formal, such that a record of such manipulations can be verified by a computer program. The second component is a structure that provides a meaning to the sentences of this language in terms of mental objects intuitively comprehensible to humans. The third component is a structure that enables humans to encode mathematical ideas in terms of the objects directly associated with the language.

Voevodsky as a trinitarian

From *The Origins and Motivations of Univalent Foundations:*

. . . let me suppose that any foundation for mathematics adequate both for human reasoning and for computer verification should have the following three components.

The first component is a formal deduction system: a language and rules of manipulating sentences in this language that are purely formal, such that a record of such manipulations can be verified by a computer program. The second component is a structure that provides a meaning to the sentences of this language in terms of mental objects intuitively comprehensible to humans. The third component is a structure that enables humans to encode mathematical ideas in terms of the objects directly associated with the language.

Voevodsky as a trinitarian

From *The Origins and Motivations of Univalent Foundations:*

. . . let me suppose that any foundation for mathematics adequate both for human reasoning and for computer verification should have the following three components.

The first component is a formal deduction system: a language and rules of manipulating sentences in this language that are purely formal, such that a record of such manipulations can be verified by a computer program. The second component is a structure that provides a meaning to the sentences of this language in terms of mental objects intuitively comprehensible to humans. The third component is a structure that enables humans to encode mathematical ideas in terms of the objects directly associated with the language.

The simplicial model

Theorem (Voevodsky)

Intensional type theory has a model in Kan simplicial sets.

The equality types are path spaces, as in Awodey–Warren.

Theorem (Voevodsky)

This model has a universe (type of types).

Thus we can ask: what is equality of types in this model?

Theorem (Voevodsky)

The type $A = B$ in the simplicial model is equivalent to the space $\mathrm{hEquiv}(A, B)$ *of homotopy equivalences from A to B.*

Thus, U is an object classifier as for Rezk–Lurie ∞-topoi.

Type-theoretic univalence

Theorem (Voevodsky)

The simplicial model satisfies internally the univalence axiom:
The canonical map $(A = B) \to \mathrm{Equiv}(A, B)$ *is an equivalence.*

- Finally answers "what should an equality of types be?"
- By analogy to function and propositional extensionality, univalence is typal extensionality.
- Hoffmann-Streicher noticed univalence for discrete groupoids, but full univalence required a further insight:

Definition (Voevodsky)

$f : A \to B$ is an equivalence* if its fiber $\sum_{x:A}(f(x) = y)$ is contractible for each $y : B$.

* Voevodsky called it a "weak equivalence".

Funext and homotopy levels

Theorem (Voevodsky)

Univalence implies function extensionality, propositional extensionality, and function comprehension.

Univalence contradicts proof-irrelevance in general, but we can turn a failed conjecture into a definition:

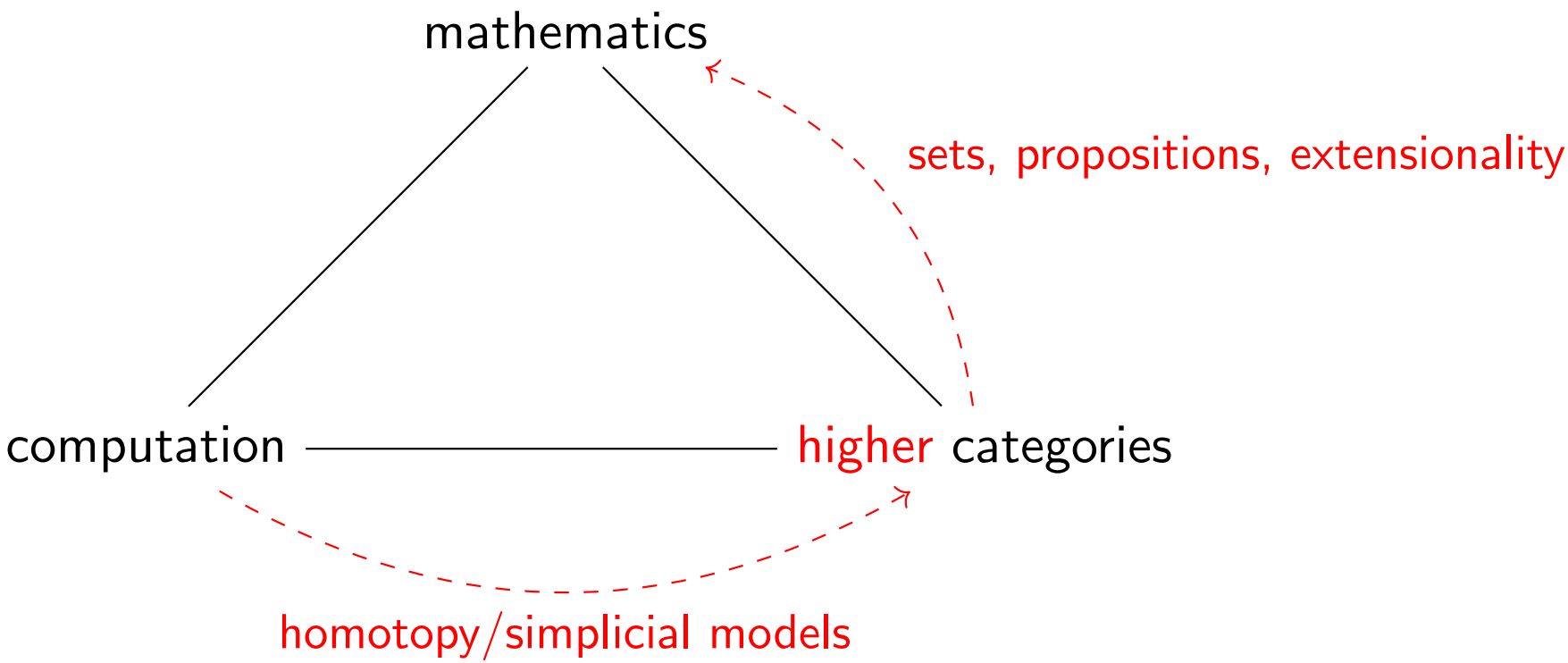
Definition (Voevodsky)

A type A is contractible, or a (-2)-type, if $\sum_{x:A} \prod_{y:A} (x = y)$.
It is an $(n{+}1)$-type* if $x = y$ is an n-type for all $x, y : A$.
A proposition is a (-1)-type, a set is a 0-type.

The sets are the types whose equality is proof-irrelevant; most "ordinary" mathematics lives in the world of sets.

* Voevodsky said "A has h-level $n + 2$" instead of "A is an n-type".

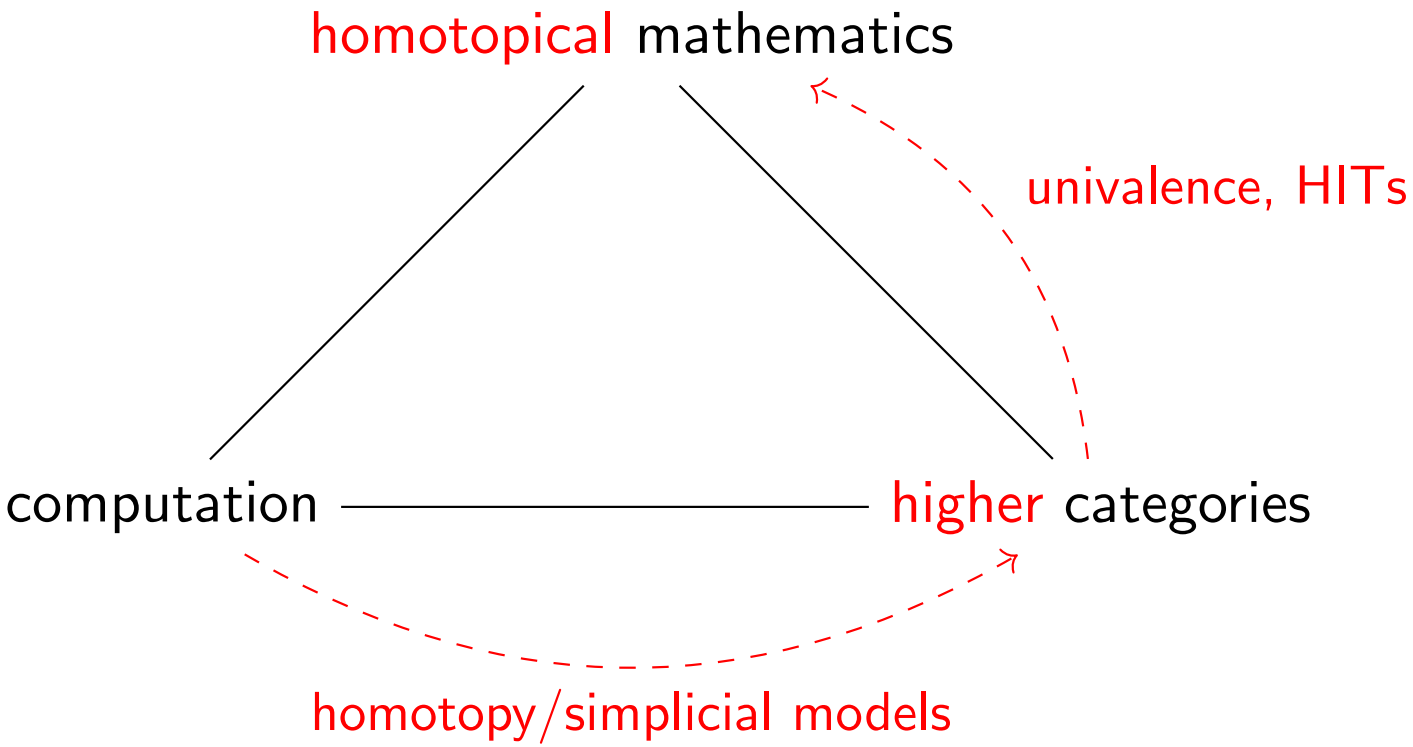
Trinitarian status report, 2009



Homotopy type theory, 2010–2013

- Oberwolfach: higher inductive types (HITs)
- Shulman: $\pi_1(S^1) = \mathbb{Z}$
- IAS: special year on HoTT/UF
- Licata: encode-decode method
- Synthetic homotopy theory: $\pi_n(S^n)$, van Kampen, etc.
- Aczel: informal type theory
- The homotopy type theory book

Trinitarian status report, 2013



Outline

Computation in Book HoTT

In Book HoTT, univalence is an axiom. This can cause evaluation to get "stuck".

Example

The equivalence $f : \mathbb{N} \to \mathbb{N}$ that swaps 0 and 1 induces, by univalence, $p : \mathbb{N} = \mathbb{N}$. Then $p_*(0)$ can be proven to equal 1, but it doesn't evaluate to it in the programming language sense.

In plain MLTT we have canonicity, e.g. any term of type $\mathbb{N}$ evaluates to some "canonical-form" numeral $\mathrm{succ}(\mathrm{succ}(\ldots(0)))$.

Conjecture (Voevodsky)

Any term of type $\mathbb{N}$ in MLTT + univalence can be proven to equal some numeral.

A trinitarian view of cubical type theory

Cubical type theory exists at all three vertices:

1. A model of type theory in cubical sets that works in a constructive metatheory.
2. A programming language that implements canonicity for univalence and higher inductive types.
3. A calculus of cubes that simplifies higher-homotopical reasoning and synthetic homotopy theory.

NB: There are different kinds of cubical type theory; here are some notable milestones and some people who worked on them:

- Bezem-Coquand-Huber: first constructive cubical set model
- Polonsky, Altenkirch-Kaposi: incomplete theories with univalence-by-definition
- Cohen-Coquand-Huber-Mörtberg: cubical model/type theory with connections
- Angiuli-Hou-Harper, Huber: cubical type theory as a programming language
- Angiuli-Brunerie-Coquand-Hou-Harper-Licata: cartesian model/type theory

Cubical type theory: a different approach to equality

Central innovation of cubical type theory

An equality between $x, y : A$ is a path $p : \mathbb{I} \to A$ defined on an "interval", with $p(0) = x$ and $p(1) = y$.

This allows us to have our cake and eat it too:

- Specifying a type's equalities is a special case of specifying its elements: an equality is a family of elements depending on $r : \mathbb{I}$.
- Often, the correct paths are simply pointwise elements, so no special rules for equality are needed. E.g. we get funext for free.
- For quotients, HITs, and univalence, we add extra elements depending on $r : \mathbb{I}$.

Challenge

Univalence ought to be a "definition" of equality in $\mathcal{U}$, like function extensionality for $A \to B$. But existing cubical type theories need auxiliary "glue types" to ensure univalence; can this be avoided?

Transport

What makes equality, equality?

Indiscernibility of identicals or transport: If $p : x = y$ and $b : B(x)$, we have $p_*(b) : B(y)$.

- In ZFC, this is an axiom of first-order logic.
- In ML type theory, this is roughly the induction principle, from the free generation of equality by reflexivity.

In cubical type theory, like MLTT, we have a basic operation p_*, but it computes differently.

- In MLTT, $p_*(b)$ only evaluates (to b) when p is reflexivity.
- In CTT, $p_*(b)$ evaluates by recursion on the type family B.

Computing transport

Again

In cubical TT, $p_*(b)$ evaluates* by recursion on the type family B.

- If $(b, c) : B(x) \times C(x)$, then $p_*(b, c) \equiv (p_* b, p_* c)$.
- If $f : B(x) \to C(x)$, then $p_*(f)(b) \equiv p_*(f(p_*^{-1}(b)))$.
- If $q : f(x) =_B g(x)$, then $p_*(q) \equiv f(p^{-1}) \bullet q \bullet g(p)$ (which in turn evaluates by recursion on B).

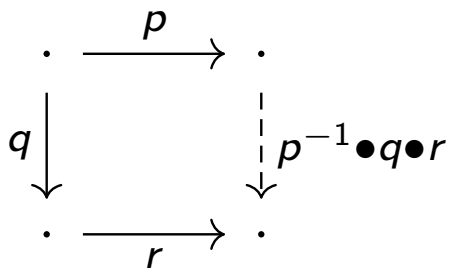
In practice, this enables more computation than evaluating on reflexivity, and we can still prove that $\mathrm{refl}_*(b) = b$.

- We recover canonicity: if $p : \mathbb{N} = \mathbb{N}$ swaps 0 and 1, then $p_*(0)$ can compute to 1 even though p is not refl.
- Formalizing mathematics is often easier: many "by hand" steps become automated evaluations.

* except in some cases, like composition in a HIT, where it is a canonical form and doesn't evaluate further.

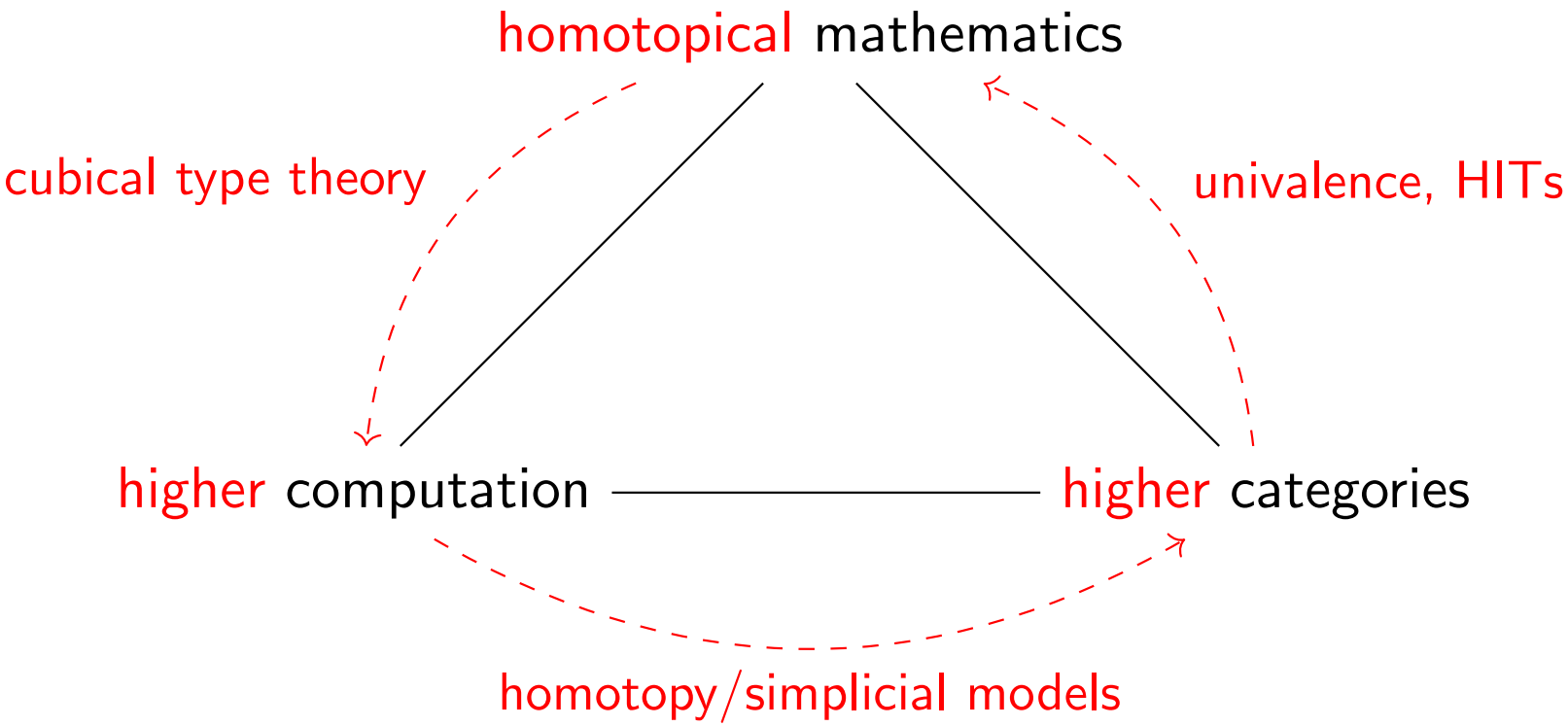
So why "cubical"?

1. Paths of paths of paths... are defined on cubes $\mathbb{I} \times \mathbb{I} \times \mathbb{I} \cdots$
2. The natural path composition $(p, r)_*(q)$ is a cubical "box-filling":



   and similarly for higher-dimensional versions.
3. Historically, the syntax was invented by abstracting from a model in cubical sets.

Trinitarian status report, 2018



Outline

Synthetic homotopy theory

- In analytic homotopy theory, homotopy types are defined using sets (topological spaces, simplicial sets).
- In synthetic homotopy theory, homotopy types are basic foundational objects described by rules and axioms.

Known

$\pi_1(S^1)$, $\pi_n(S^n)$, $\pi_3(S^2)$, Freudenthal suspension, Blakers–Massey theorem (Hou-Finster-Licata-Lumsdaine), Steenrod squares (Brunerie), Serre spectral sequence (Shulman-van Doorn-etc.), real projective spaces (Buchholtz-Rijke), ...

Frontiers

More spectral sequences, Steenrod operations, localization and completion [MRC project], spectra and homology [MRC project],...

Synthetic homotopy theory

Synthetic homotopy theory is constructive by default, meaning:

- No law of excluded middle or axiom of choice; but
- Results are interpretable in any ∞-topos.

In Book HoTT and Cubical TT, moreover:

- No point-set-level equality; everything is up to homotopy.
- Infinite coherence structures have to be finitely encoded.

Book HoTT also requires tedious "path algebra", which in Cubical TT are computation steps [MRC project].

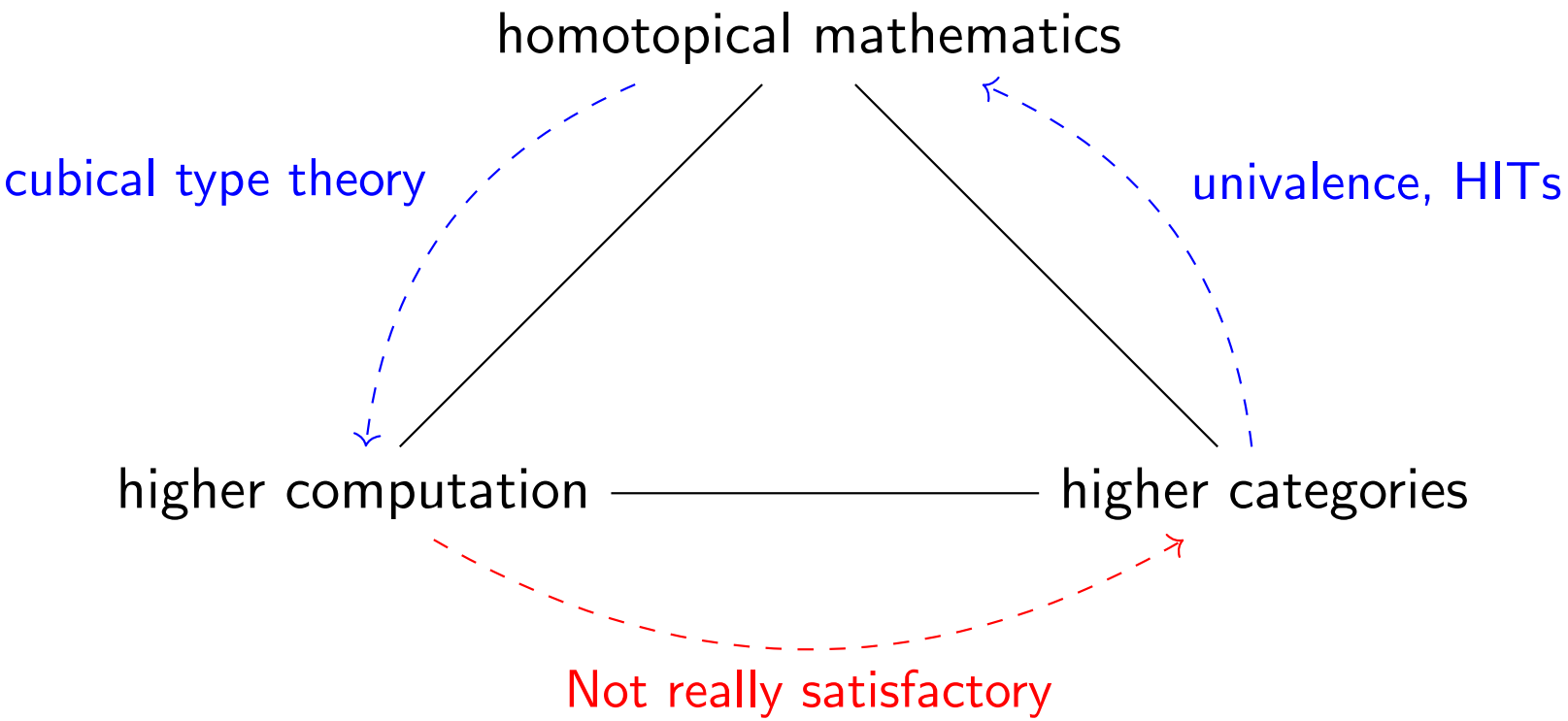
From set-based to univalent mathematics

HoTT can also serve as a foundation for non-homotopical mathematics, using 0-types ("h-sets") in place of ZF-sets.

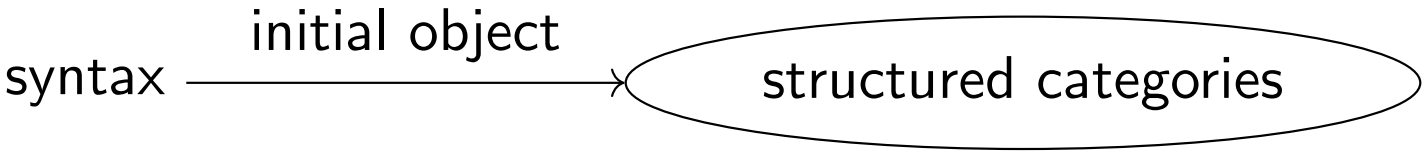
- For the most part, univalent mathematics is not very different.
- Sets behave structurally, as in ETCS: no global $\in$.
- Constructive by default: no LEM or AC unless added.
- HITs allow free constructions without transfinite induction.
- A little easier to formalize in a computer proof assistant than many other foundations.
- "Global choice" is inconsistent with univalence.
- Category theory forces us up the homotopical ladder one rung at a time: the type of sets is a 1-type, not a 0-type (Ahrens-Kapulkin-Shulman)

Outline

Trinitarian status report, 2018



Functorial semantics

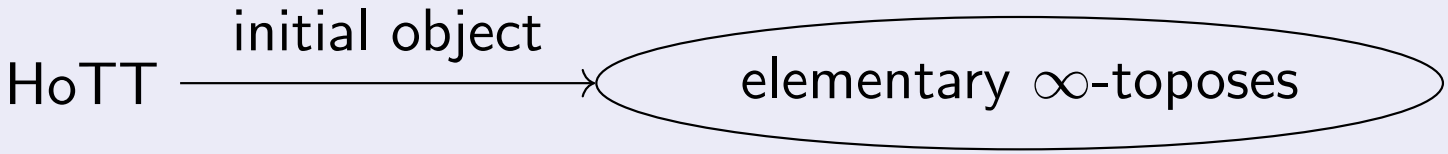


STLC	cartesian closed cats
MILL	closed monoidal cats
LL	$*$-autonomous cats
EDTT	locally CC cats
IHOL	elementary toposes
$\vdots$	$\vdots$

The interpretation of syntax in a category is given by the unique structure-preserving functor out of the initial structure category built from syntax.

Higher functorial semantics

Dream



Problems with this include:

- We don't know what an elementary ∞-topos is. [MRC project]
- Current technology requires strictifying ∞-categories.
- Even the strict initiality theorem isn't proven.

Strictification

Currently, we interpret syntax into an $(\infty, 1)$-category by:

1. Presenting with a Quillen model category.
2. The category of fibrant objects is a "tribe".
3. Building the initial tribe out of syntax.

Problems with this include:

- Model categories only work for the Grothendieck case.
- Unknown whether universes and HITs can always also be strictified (Shulman, Cisinski, Lumsdaine-Shulman).
- Want syntax to be initial among $(\infty, 1)$-categories, not just tribes (Kapulkin-Szumiło) [MRC project]
- If HoTT were also the metatheory, such strictification would definitely not be always possible.
- Cubical type theory adds extra computation rules, hence more strictification is required.

Computation and categories seem to pull in different directions: syntax wants more computation rules, ∞-categories want fewer.

A possible way forward

Challenge

Regard the canonical forms as an inductively defined structure on their own, without any strict equality of the usual sort, and interpret them directly into ∞-categories.

To interpret a non-canonical form, first compute it fully. Thus "computational equality" happens only in the syntax, and doesn't have to be represented by anything strict in the models.

This does work in very simple cases. Doing it in general would be very complicated, but it would be a triumph of trinitarianism.

The initiality conjecture

. . . It is a long standing conjecture that the syntactic category of the MLTT is the initial [tribe] . . . Thomas Streicher proved an analog of this conjecture for a much more simple type theory. . . it remains the only substantially non-trivial analog of this conjecture known. . . . Proving this conjecture in a way that will also enable us to prove its analogs for yet more complex type theories. . . is the most important, from my point of view, goal that needs to be achieved in the development of the UF and HoTT.

Nobody doubts this conjecture is true, or has much doubt about how to prove it. But no one has done it either: it would be very tedious and boring, and everyone knows it's true anyway. (-:0

The initiality conjecture

. . . It is a long standing conjecture that the syntactic category of the MLTT is the initial [tribe] . . . Thomas Streicher proved an analog of this conjecture for a much more simple type theory. . . it remains the only substantially non-trivial analog of this conjecture known. . . . Proving this conjecture in a way that will also enable us to prove its analogs for yet more complex type theories. . . is the most important, from my point of view, goal that needs to be achieved in the development of the UF and HoTT.

Nobody doubts this conjecture is true, or has much doubt about how to prove it. But no one has done it either: it would be very tedious and boring, and everyone knows it's true anyway. (-:0

Initiality: (Why) is it important?

Why do we need a general theory of dependent type theories and their initiality theorems?

1. Type theory is a constantly evolving subject: we don't ever expect a final answer to "the" type theory.
2. Which type theories does Streicher's method work for? Maybe experts know, at least in the Potter Stewart* sense, but can they communicate that knowledge to students and outsiders?
3. A general theory of "Potter Stewart type theories" could enable many other general theorems, like strictification.
4. Not even a graduate student would want to carefully prove initiality for only one type theory like Book HoTT. So if we want the theorem to actually get proven, even in only one case, we need to do it in generality!

(Peter LeFanu Lumsdaine, personal communication)

* "I shall not today attempt further to define the kinds of material I understand to be embraced within that shorthand description ['hard-core pornography'], and perhaps I could never succeed in intelligibly doing so. But I know it when I see it." (*Jacobellis v. Ohio*)

Outline

The problem of infinite objects

One of the major remaining foundational problems is:

The problem of infinite objects

Higher category theory and homotopy theory need infinitely coherent structures: $(\infty, 1)$-categories, A_∞-spaces, E_∞-rings, ...
In general, we don't know how to define these in HoTT.

- Classical definitions use point-set level equality, either directly (model categories) or one level up ($(\infty, 1)$-categories), but that's not available in HoTT.
- Sometimes infinite coherence can be finitely encoded: contractibility, equivalences, idempotents, even ∞-groupoids.
- Other times we can cheat, e.g. define an "∞-group" to be its classifying space, a pointed connected type.

A type system with two identity types

From *A type system with two identity types:*

...Several approaches lead to constructions which at some point require some object expression to have type $T(m)$ while its actual type is $T'(m)$. Here m is a natural parameter and remarkably for each individual m one has $T(m) \stackrel{d}{=} T'(m)$.

However the length of the reduction sequence one needs to perform to connect $T(m)$ with $T'(m)$ grows with m. Therefore in the context where m is a variable T is not definitionally equal to T'.

Possible solution: *to make it possible to prove definitional equality by induction.*

Two-level type theories

Idea (Voevodsky, Bauer, Annenkov-Capriotti-Kraus)

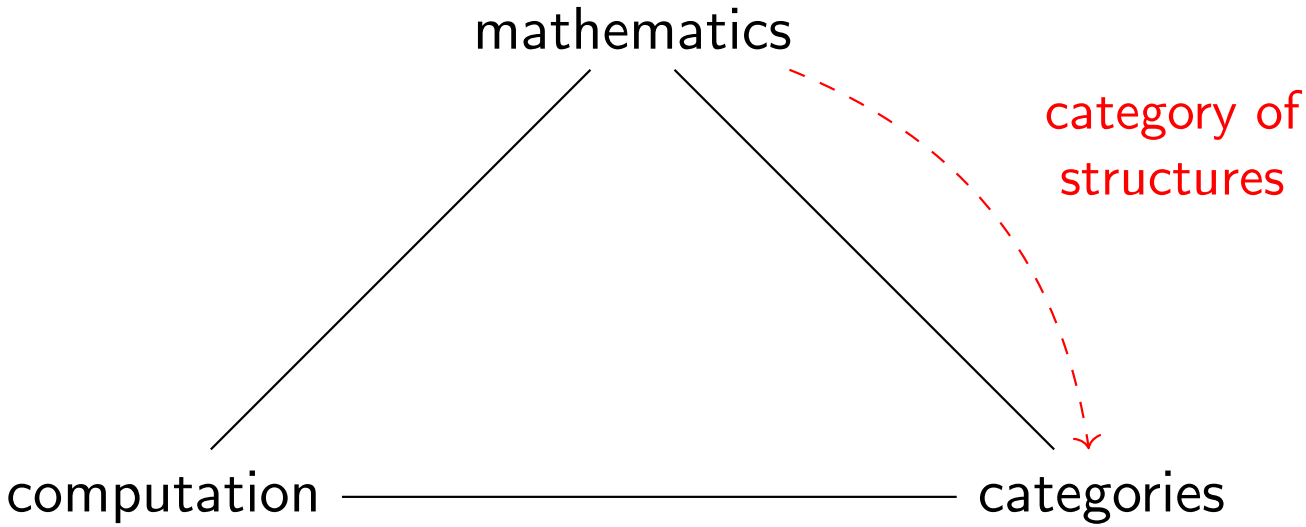
Separate exact equality type, which is a "pretype", and univalent path type, which is a "fibrant type".

- Can define some higher coherence structures.
- Tied to current semantic technology (model categories).
- Apparently gives up some foundational homotopy-invariance.
- Definitions are no easier than in analytic homotopy theory, and in some ways they are harder (e.g. tracking fibrancy).

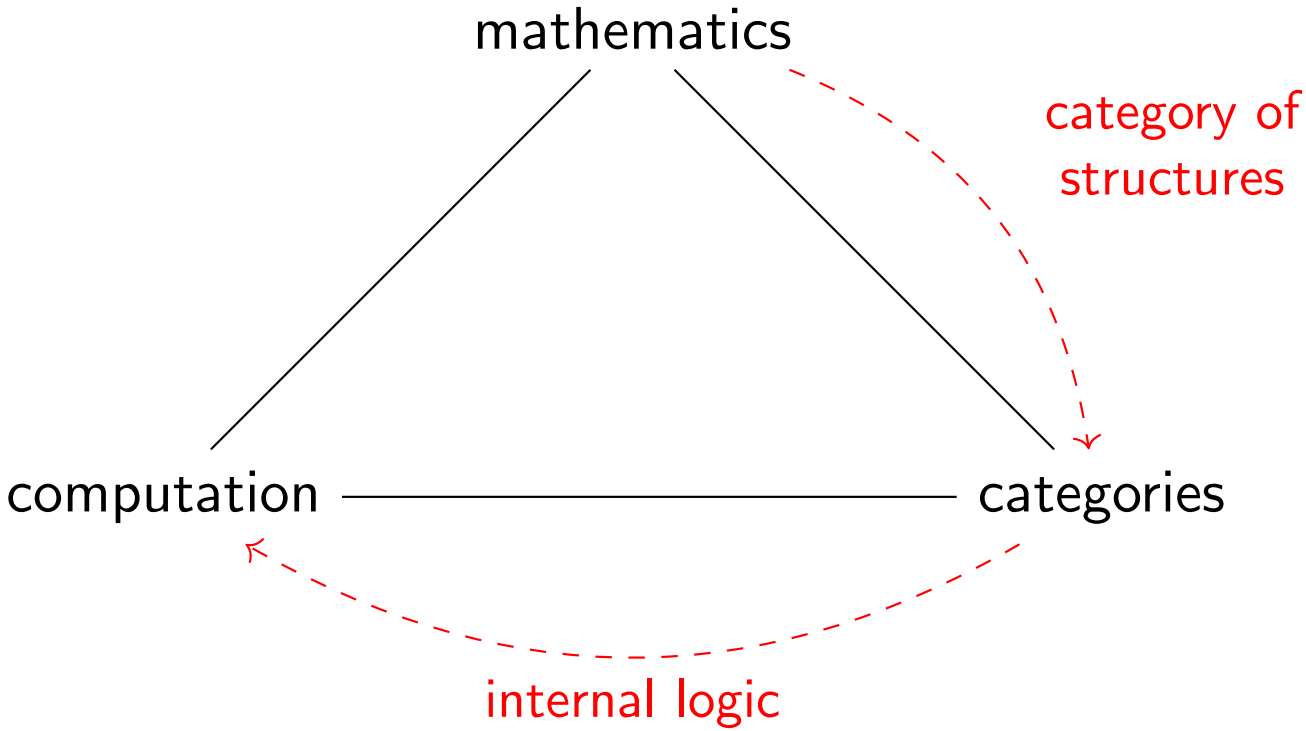
One possible perspective

Two-level type theory is a blunt weapon (but less blunt than ZFC). It works, but more elegant is to seek synthetic descriptions of more and more higher structures, like HoTT's synthetic ∞-groupoids.

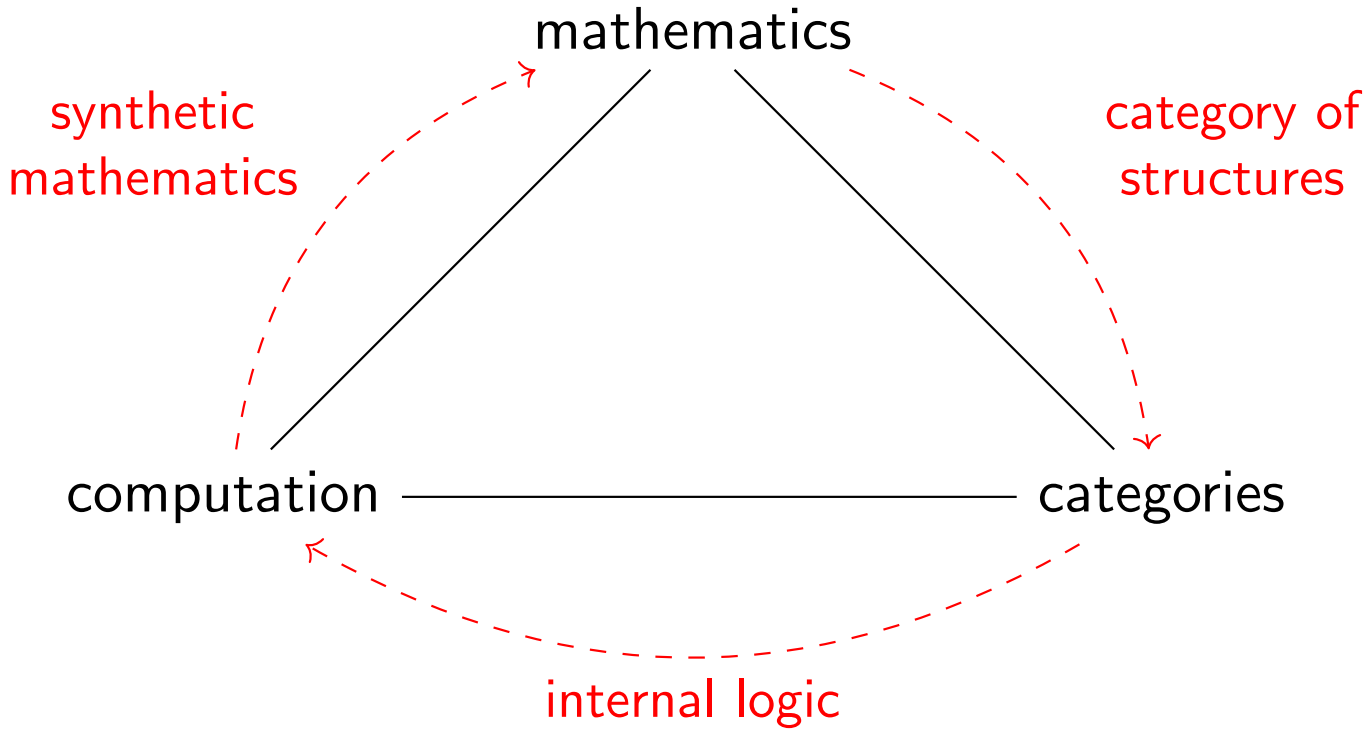
A trinitarian approach to enhanced type theories



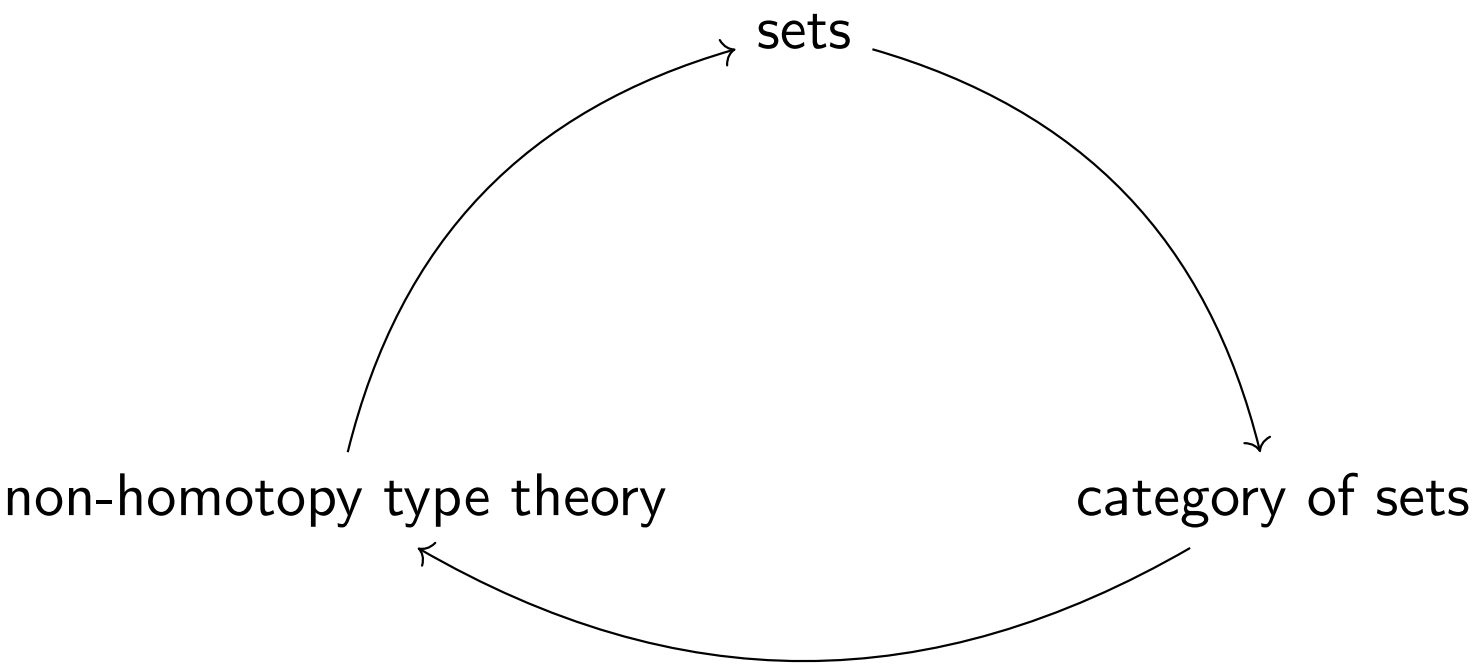
A trinitarian approach to enhanced type theories



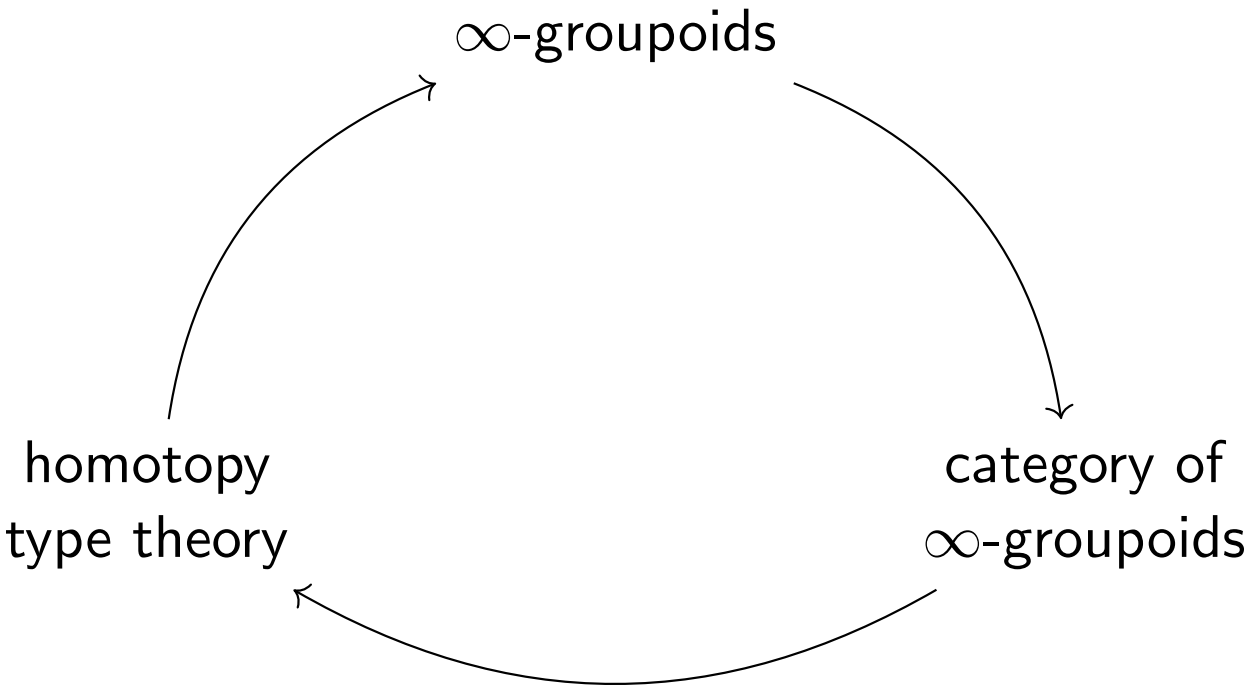
A trinitarian approach to enhanced type theories



Pre-homotopy type theory

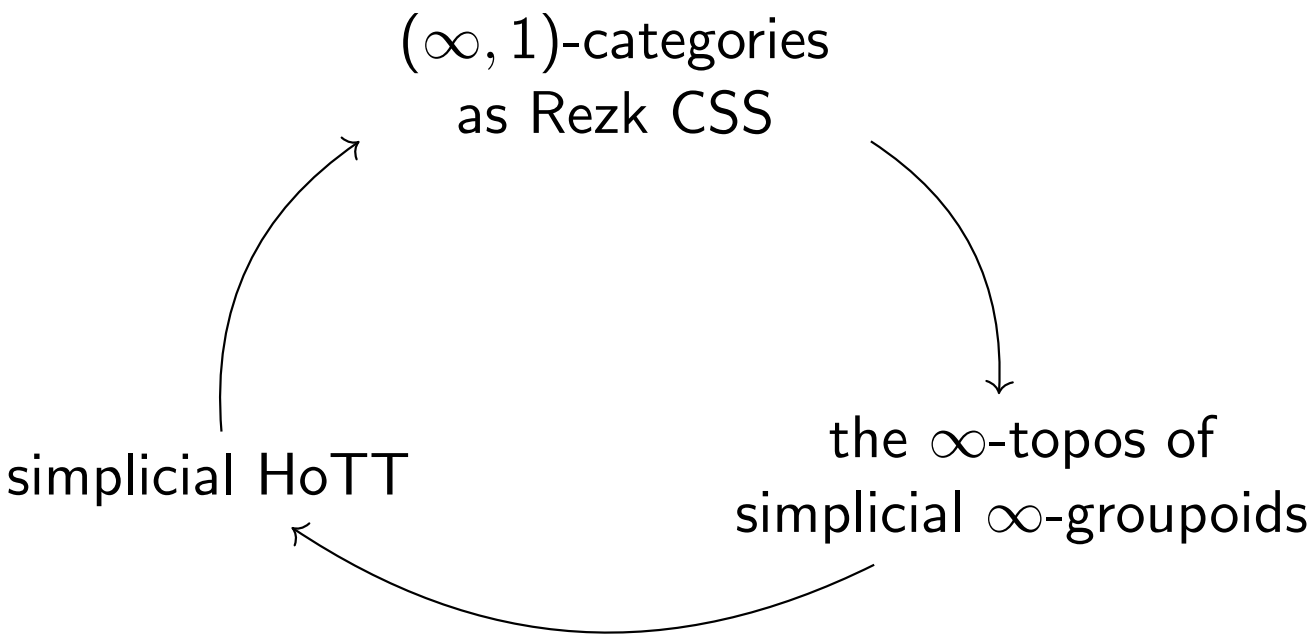


An ahistorical approach to homotopy type theory



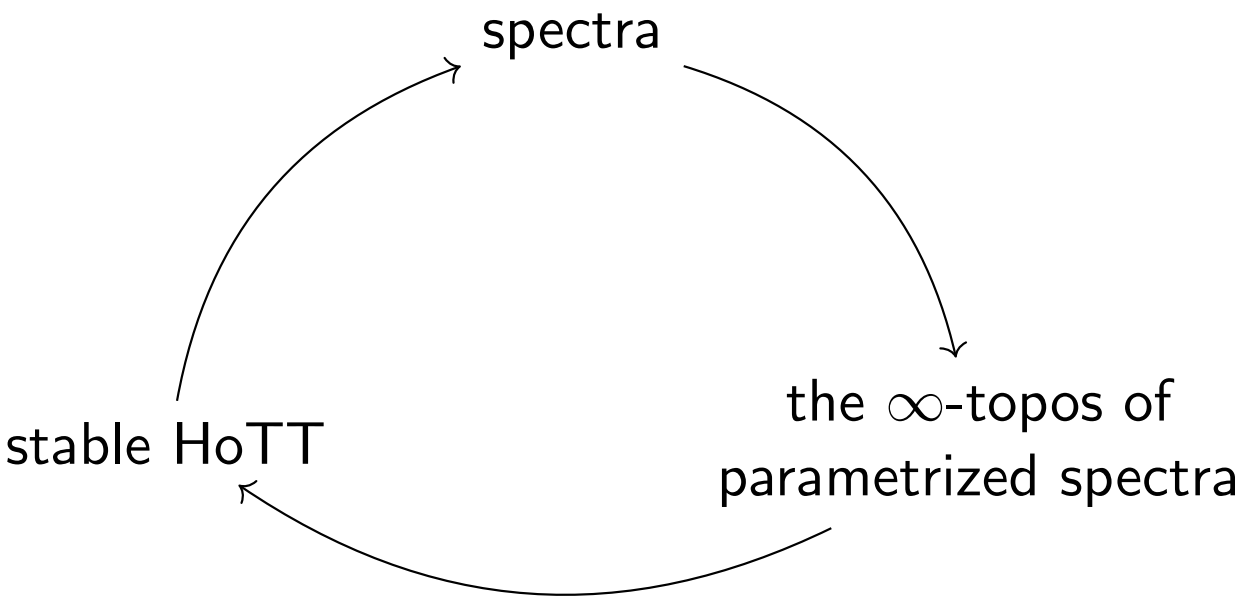
Instead of types behaving like sets, now types behave like ∞-groupoids.

Directed homotopy type theory



- Types act like simplicial spaces, with $(\infty, 1)$-categories (Rezk types) an internally finitely defined subclass.
- Coherent ∞-adjunctions, ∞-limits, etc. are finitely definable.
- The Yoneda lemma is "directed transport".
- Riehl-Shulman, [MRC project].

Stable homotopy type theory



Towards cohesion

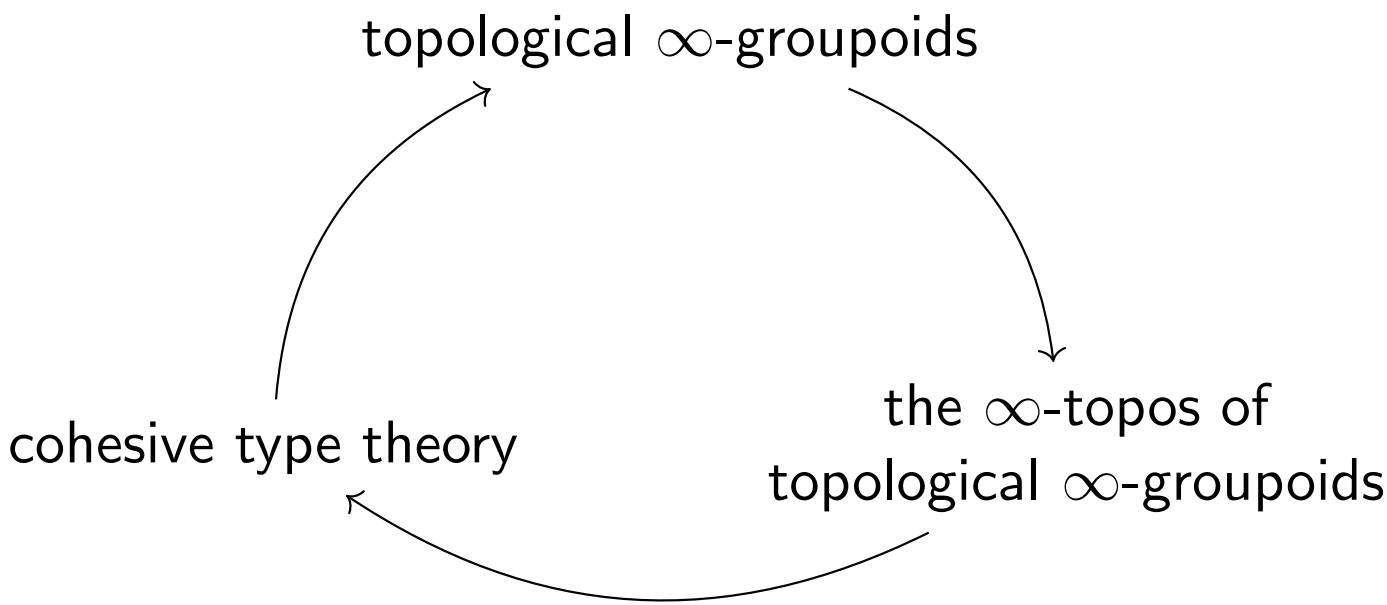
Problem

Synthetic homotopy theory disentangles ∞-groupoids from topological spaces. But many applications of classical homotopy theory use topological presentations to extract non-homotopical information: Brouwer and Lefschetz fixed-point theorems, fundamental theorem of algebra, Borsuk-Ulam theorem, ...

Possible solutions:

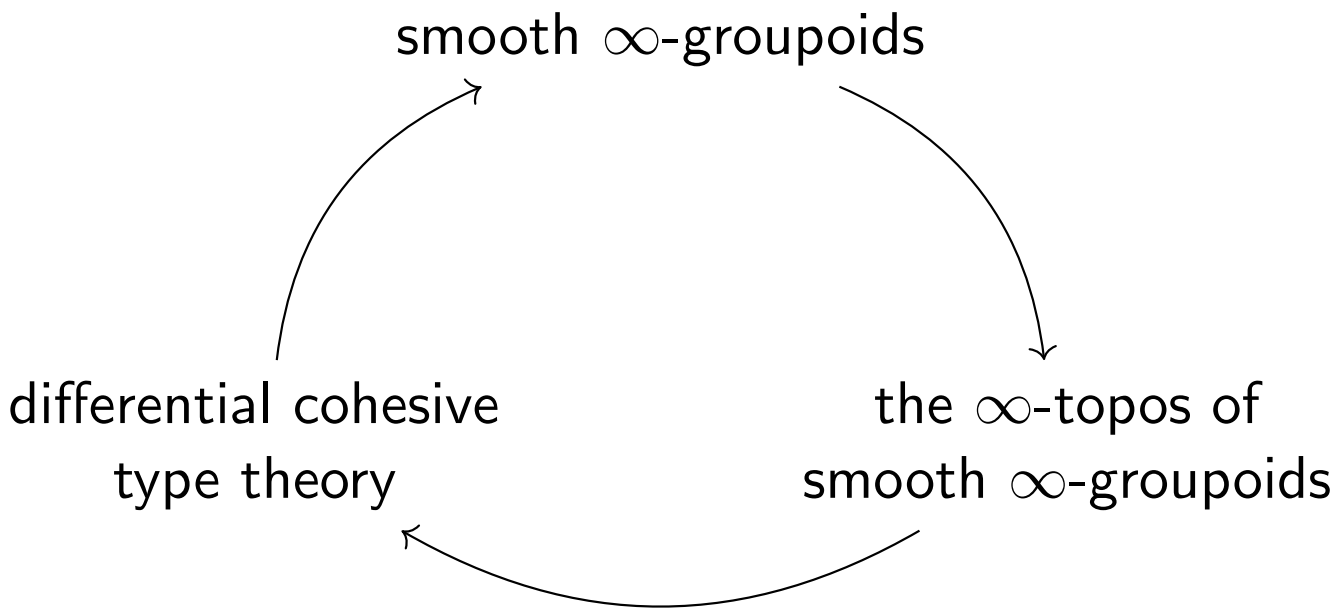
1. Define the fundamental ∞-groupoid (or fundamental type) of a topological space internally in HoTT. This is an infinite object, but might be possible in two-level type theory.
2. Make topology and fundamental ∞-groupoid (or shape) synthetic as well, described by rules and axioms.

Cohesive homotopy type theory



- Types have both a topology and an ∞-groupoid structure.
- The shape $\int$ reflects into topologically-discrete types.
- Can prove many topological applications of homotopy theory.
- Shulman, [MRC project]

Differential cohesive homotopy type theory



- Types have smooth structure and ∞-groupoid structure.
- Schreiber, Wellen, [MRC project]

Cohesion: a trinitarian success story

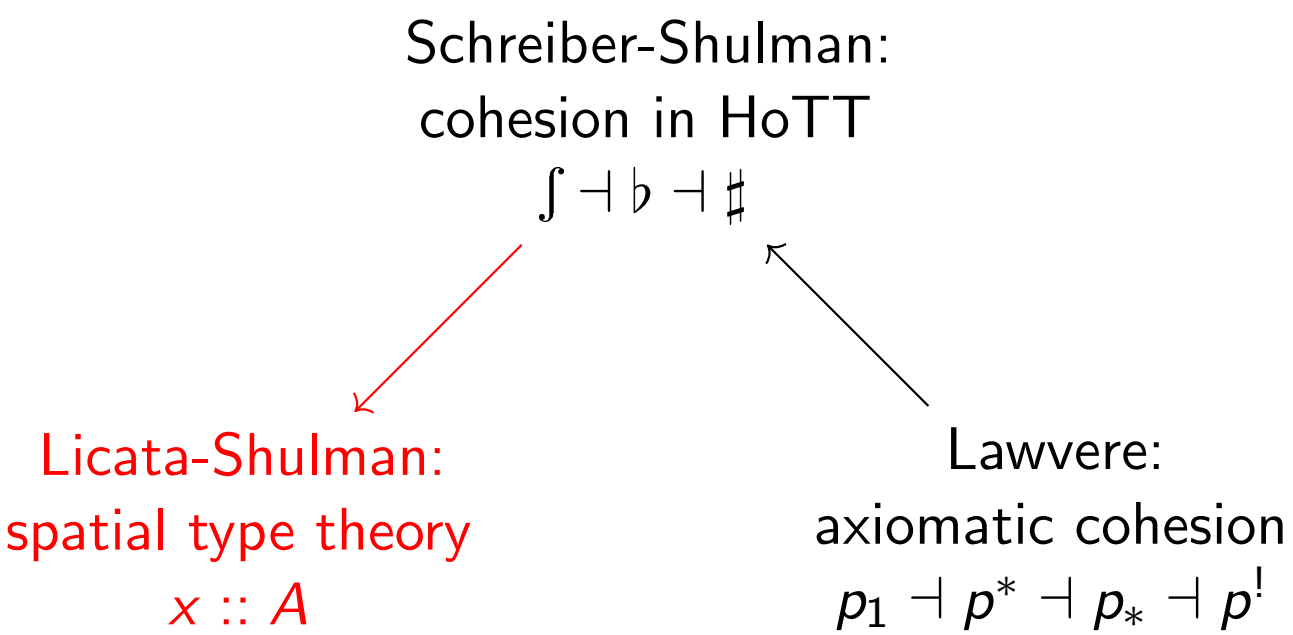
Lawvere:
axiomatic cohesion
$p_1 \dashv p^* \dashv p_* \dashv p^!$

Cohesion: a trinitarian success story

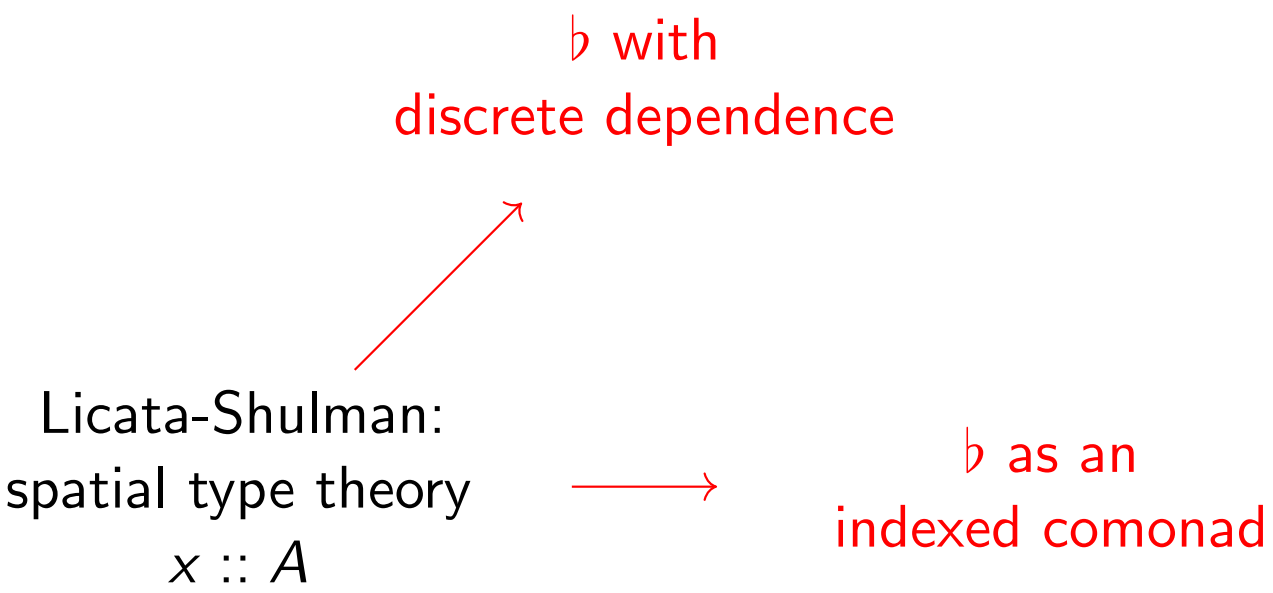
Schreiber-Shulman:
cohesion in HoTT
$\int \dashv \flat \dashv \sharp$

Lawvere:
axiomatic cohesion
$p_1 \dashv p^* \dashv p_* \dashv p^!$

Cohesion: a trinitarian success story



Cohesion: a trinitarian success story



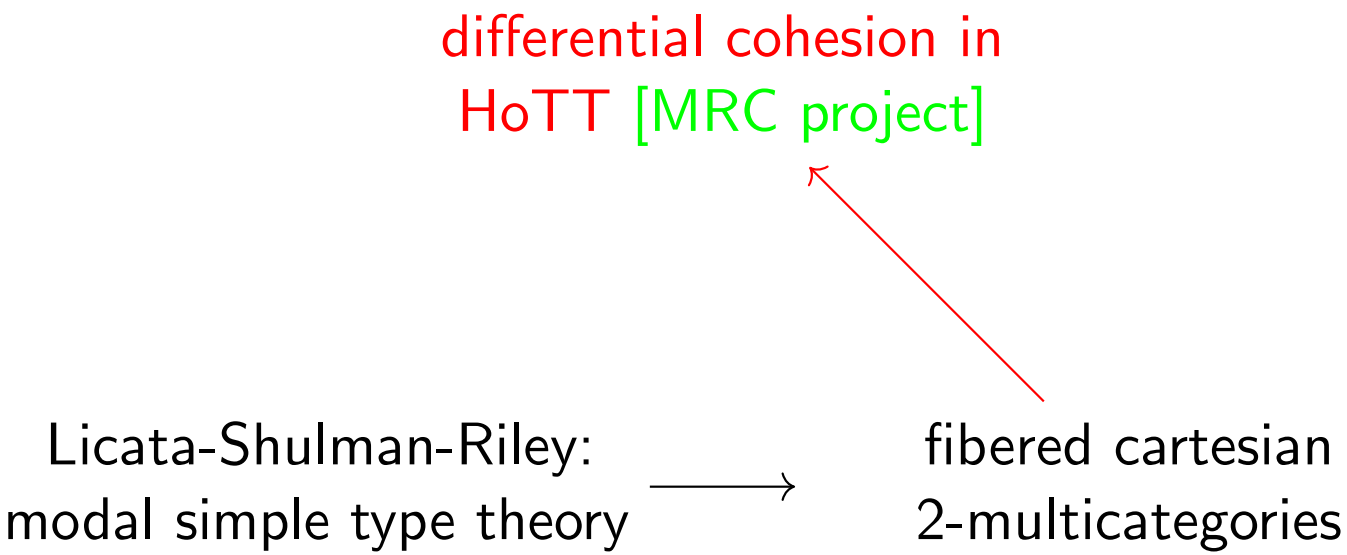
Cohesion: a trinitarian success story

Licata-Shulman-Riley:
modal simple type theory

Cohesion: a trinitarian success story

Licata-Shulman-Riley:
modal simple type theory $\longrightarrow$ fibered cartesian
2-multicategories

Cohesion: a trinitarian success story



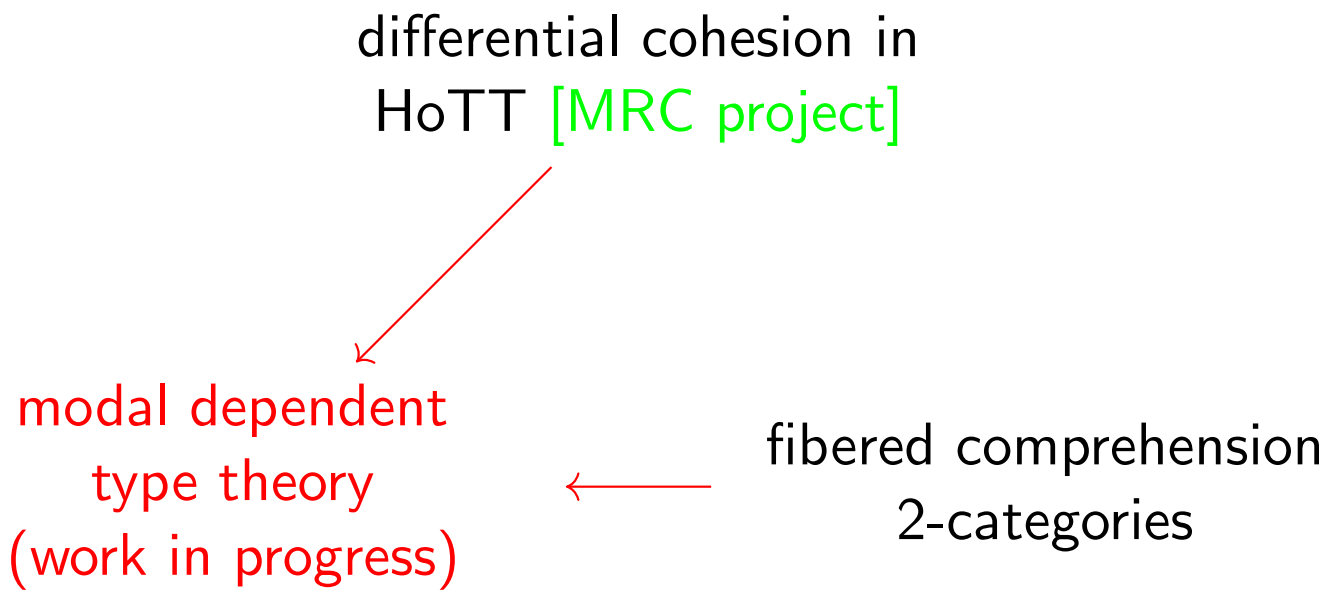
Cohesion: a trinitarian success story

differential cohesion in
HoTT [MRC project]

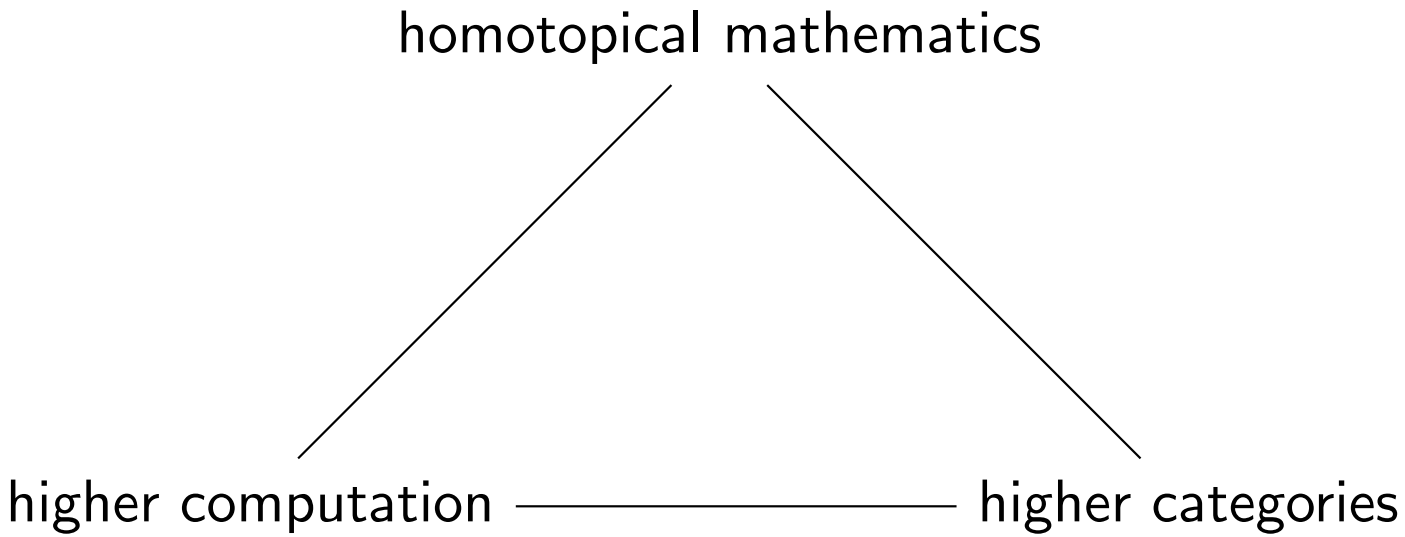
Licata-Shulman-Riley:
modal simple type theory

fibered comprehension
2-categories

Cohesion: a trinitarian success story



Homotopical Trinitarianism, bis



. . . any concept arising in one aspect should have meaning from the perspective of the other two. If you arrive at an insight that has importance for logic, languages, and categories, then you may feel sure that you have elucidated an essential concept of computation—you have made an enduring scientific discovery.