Modal Homotopy Type Theory

David Corfield

University of Kent

25 January, 2021

- Modal homotopy type theory: The prospect of a new logic for philosophy

- Modal homotopy type theory: The prospect of a new logic for philosophy

Note that modal homotopy type theory emerges out of *categorical logic* (category theory – type theory), which has been developed over many decades.

Categorical logic in general offers many tools for analytic philosophy.

- Modal homotopy type theory: The prospect of a new logic for philosophy

Note that modal homotopy type theory emerges out of *categorical logic* (category theory – type theory), which has been developed over many decades.

Categorical logic in general offers many tools for analytic philosophy.

However, HoTT/UF is a particularly interesting system.

HoTT

- Another foundation for mathematics, which describes a different 'shape' natively – rather than a set, a 'homotopy type'.
- But it does so as an integral system, directly comparable to (untyped) first-order or higher-order logic.

HoTT

- Another foundation for mathematics, which describes a different ‘shape’ natively – rather than a set, a ‘homotopy type’.
- But it does so as an integral system, directly comparable to (untyped) first-order or higher-order logic.
- The relationship between logic and mathematics is altered. Propositions constitute a kind of type, as do sets.

HoTT

- Another foundation for mathematics, which describes a different 'shape' natively – rather than a set, a 'homotopy type'.
- But it does so as an integral system, directly comparable to (untyped) first-order or higher-order logic.
- The relationship between logic and mathematics is altered. Propositions constitute a kind of type, as do sets.
- HoTT is an intrinsically structuralist language, suited to reasoning up to equivalence. The 'homotopification' of mathematics.

HoTT

- Another foundation for mathematics, which describes a different ‘shape’ natively – rather than a set, a ‘homotopy type’.
- But it does so as an integral system, directly comparable to (untyped) first-order or higher-order logic.
- The relationship between logic and mathematics is altered. Propositions constitute a kind of type, as do sets.
- HoTT is an intrinsically structuralist language, suited to reasoning up to equivalence. The ‘homotopification’ of mathematics.
- Modal variants are interesting too. They allow a great deal of theory to be expressed synthetically.

Peeling back the layers

- Modal homotopy type theory

is

- Modal homotopy dependent type theory

Peeling back the layers

- Modal homotopy type theory

is

- Modal homotopy dependent type theory

My book explores these components in conceptual terms. Other sources are available (e.g., the 'HoTT Book') for a formal treatment.

Why types?

- Question words: who?, when?, where?, which A?, how?, why?

Why types?

- Question words: who?, when?, where?, which A?, how?, why?
- Objects exist, events occur, properties obtain,...

Why types?

- Question words: who?, when?, where?, which A?, how?, why?
- Objects exist, events occur, properties obtain,...
- Computer languages, theorem-provers.

Why types?

- Question words: who?, when?, where?, which A?, how?, why?
- Objects exist, events occur, properties obtain,...
- Computer languages, theorem-provers.

A type may be viewed as a set of clothes (or a suit of armor) that protects an underlying untyped representation from arbitrary or unintended use. It provides a protective covering that hides the underlying representation and constrains the way objects may interact with other objects. In an untyped system untyped objects are naked in that the underlying representation is exposed for all to see. (Cardelli, L., Wegner, P., On Understanding Types, Data Abstraction, and Polymorphism, 1985, p. 474)

Philosophical interest

Surprisingly little work has been done on type theory meets analytic philosophy of language/metaphysics.

> *...we are in the dark about the nature of philosophical problems and methods if we are in the dark about types and categories. (Ryle, "Categories", 1938)*

(For some further thoughts, my last few blog posts: https://golem.ph.utexas.edu/category/david.html)

Protection from unintended use

Given a function $f : A \to B$ and an element $a : A$, we apply the function and obtain $f(a) : B$.

Protection from unintended use

Given a function $f : A \to B$ and an element $a : A$, we apply the function and obtain $f(a) : B$.

We don't form terms $f(c)$ on the understanding that c may turn out to be an element of A. Don't form 'the mass of the entity I'm thinking about.'

Protection from unintended use

Given a function $f : A \to B$ and an element $a : A$, we apply the function and obtain $f(a) : B$.

We don't form terms $f(c)$ on the understanding that c may turn out to be an element of A. Don't form 'the mass of the entity I'm thinking about.'

Similarly, a predicate only applies to elements of a specific type.

Protection from unintended use

Given a function $f : A \to B$ and an element $a : A$, we apply the function and obtain $f(a) : B$.

We don't form terms $f(c)$ on the understanding that c may turn out to be an element of A. Don't form 'the mass of the entity I'm thinking about.'

Similarly, a predicate only applies to elements of a specific type.

Questions concerning the value of ill-formed terms/types "do not arise" (Collingwood, Strawson).

Dependent types

The big departure now is to allow types to be *dependent*:

- $x : A \vdash B(x) : Type$.

Dependent types

The big departure now is to allow types to be *dependent*:

- $x : A \vdash B(x) : Type$.

This allows for the natural treatment of the common situation of a function $f : B \to A$, where we consider the inverse images of each $a : A$.

Assigning players to their team, for $t : Team$, $Player(t)$ collects players of the same team.

Why dependent types?

Let k be a field, V a finite-dimensional vector space over k, and f an endomorphism of V. Then define $E(V, k, f)$, the eventual image of f, as the vector space which is the intersection of all $f^n(V)$. Show $f(E) = E$.

Why dependent types?

Let k be a field, V a finite-dimensional vector space over k, and f an endomorphism of V. Then define $E(V, k, f)$, the eventual image of f, as the vector space which is the intersection of all $f^n(V)$. Show $f(E) = E$.

$k : Field, V : FinVect(k), f : Endo(V, k) \vdash E(V, k, f) : SubVect(V, k)$

Why dependent types?

Let k be a field, V a finite-dimensional vector space over k, and f an endomorphism of V. Then define $E(V, k, f)$, the eventual image of f, as the vector space which is the intersection of all $f^n(V)$. Show $f(E) = E$.

$k : Field, V : FinVect(k), f : Endo(V, k) \vdash E(V, k, f) : SubVect(V, k)$

Let x be an author, y one of their books, and z a character from this book. The question arises as to whether z's appearance in y is autobiographical.

Why dependent types?

Let k be a field, V a finite-dimensional vector space over k, and f an endomorphism of V. Then define $E(V, k, f)$, the eventual image of f, as the vector space which is the intersection of all $f^n(V)$. Show $f(E) = E$.

$k : Field, V : FinVect(k), f : Endo(V, k) \vdash E(V, k, f) : SubVect(V, k)$

Let x be an author, y one of their books, and z a character from this book. The question arises as to whether z's appearance in y is autobiographical.

$x : Author, y : Book(x), z : Character(x, y) \vdash Autobiographical(x, y, z) : Prop$

Why dependent types?

Let k be a field, V a finite-dimensional vector space over k, and f an endomorphism of V. Then define $E(V, k, f)$, the eventual image of f, as the vector space which is the intersection of all $f^n(V)$. Show $f(E) = E$.

$k : \mathit{Field}, V : \mathit{FinVect}(k), f : \mathit{Endo}(V, k) \vdash E(V, k, f) : \mathit{SubVect}(V, k)$

Let x be an author, y one of their books, and z a character from this book. The question arises as to whether z's appearance in y is autobiographical.

$x : \mathit{Author}, y : \mathit{Book}(x), z : \mathit{Character}(x, y) \vdash \mathit{Autobiographical}(x, y, z) : \mathit{Prop}$

$x : \mathit{Author}, y : \mathit{Book}(x), z : \mathit{Character}(x, z) \vdash a : \mathit{Autobiographical}(x, y, z)$

Judgement structure

- $x : A, y : B(x), z : C(x, y) \vdash d : D(x, y, z)$

Judgement structure

- $x : A, y : B(x), z : C(x, y) \vdash d : D(x, y, z)$

We may consider first-order logic (FOL) as a dependent type-theory, but a very restricted one.

Judgement structure

- $x : A, y : B(x), z : C(x, y) \vdash d : D(x, y, z)$

We may consider first-order logic (FOL) as a dependent type-theory, but a very restricted one.

Typed FOL has one layer of non-dependent types (sets), all other types are dependent propositions – predicates, relations.

Judgement structure

- $x : A, y : B(x), z : C(x, y) \vdash d : D(x, y, z)$

We may consider first-order logic (FOL) as a dependent type-theory, but a very restricted one.

Typed FOL has one layer of non-dependent types (sets), all other types are dependent propositions – predicates, relations.

- $x : A, y : B, z : C, P(x), Q(y, z), R \vdash S(x, y, z)$

Note that we typically don't mention elements of propositions, but take the presence of these propositions to indicate that they hold.

Now in *untyped* (unityped) FOL, we only allow one non-dependent type, and then omit to mention it.

- $x : A, y : B, z : C, P(x), Q(y, z), R \vdash S(x, y, z)$

Now in *untyped* (unityped) FOL, we only allow one non-dependent type, and then omit to mention it.

- $x : A, y : B, z : C, P(x), Q(y, z), R \vdash S(x, y, z)$
- $x, y, z : A, P(x), Q(y, z), R \vdash S(x, y, z)$

Now in *untyped* (unityped) FOL, we only allow one non-dependent type, and then omit to mention it.

- $x : A, y : B, z : C, P(x), Q(y, z), R \vdash S(x, y, z)$
- $x, y, z : A, P(x), Q(y, z), R \vdash S(x, y, z)$
- $P(x), Q(y, z), R \vdash S(x, y, z)$

The domain remains implicit, and all variables range over it.

The form of a logic gives rise to a metaphysical perspective.

Untyped FOL suggests that objects belong to a domain of ‘everything’, and that then there are properties corresponding to subclasses of the domain.

There’s a clear distinction between object and property.

Type formation

After deciding on the judgement structure, we now need to provide ways to form types: 0, 1, $A \times B$, $A + B$, $[A, B]$,...

Type formation

After deciding on the judgement structure, we now need to provide ways to form types: 0, 1, $A \times B$, $A + B$, $[A, B]$,...

Type formation is specified by rules saying when we can introduce a type, how to construct terms of that type, how to use or "eliminate" terms of that type, and how to compute when we combine the constructors with the eliminators.

	type theory	**category theory**
	syntax	semantics
	natural deduction	universal construction
	product type	***product***
type formation	$\dfrac{\vdash A : \mathrm{Type} \qquad \vdash B : \mathrm{Type}}{\vdash A \times B : \mathrm{Type}}$	$A, B \in \mathscr{C} \Rightarrow A \times B \in \mathscr{C}$
term introduction	$\dfrac{\vdash a : A \qquad \vdash b : B}{\vdash (a,b) : A \times B}$	$\begin{tikzcd} & Q \arrow[dl, "a"'] \arrow[d, "(a,b)"] \arrow[dr, "b"] & \\ A & A \times B & B \end{tikzcd}$
term elimination	$\dfrac{\vdash t : A \times B}{\vdash p_1(t) : A} \qquad \dfrac{\vdash t : A \times B}{\vdash p_2(t) : B}$	$\begin{tikzcd} & Q \arrow[d, "t"] & \\ A & A \times B \arrow[l, "p_1"'] \arrow[r, "p_2"] & B \end{tikzcd}$
computation rule	$p_1(a,b) = a \quad p_2(a,b) = b$	$\begin{tikzcd} & Q \arrow[dl, "a"'] \arrow[d, "(a,b)"] \arrow[dr, "b"] & \\ A & A \times B \arrow[l, "p_1"'] \arrow[r, "p_2"] & B \end{tikzcd}$

It's important to understand how type formation expresses universal properties via adjunctions. (Cf. 'Categorical Harmony and Path Induction' Patrick Walsh)

It's important to understand how type formation expresses universal properties via adjunctions. (Cf. 'Categorical Harmony and Path Induction' Patrick Walsh)

We should identify logic with a "scheme of interlocking adjoints" (Lawvere 1973, p. 142)

We also need to extend the product and function types to dependent types.

We also need to extend the product and function types to dependent types.

Consider the role of conjunction:

- $A, B \vdash C$, therefore $A \& B \vdash C$.

We also need to extend the product and function types to dependent types.

Consider the role of conjunction:

- $A, B \vdash C$, therefore $A \& B \vdash C$.

Similarly, there is the type formation known as dependent sum (or dependent pair)

- $y : Author, z : Book(y)$
- $(y, z) : \sum_{x:Author} Book(x)$

We also need to extend the product and function types to dependent types.

Consider the role of conjunction:

- $A, B \vdash C$, therefore $A \& B \vdash C$.

Similarly, there is the type formation known as dependent sum (or dependent pair)

- $y : Author, z : Book(y)$
- $(y, z) : \sum_{x:Author} Book(x)$

Pairing with a predicate gives 'An author who is ...'

- $y : Author, z : British(y)$,
- $(y, z) : \sum_{x:Author} British(x)$

- $A \vdash B$, therefore $\vdash A \to B$.

Then dependent product (or dependent function)

- $x : Author \vdash f(x) : Book(x)$
- $\vdash f : \prod_{x:Author} Book(x)$

- $A \vdash B$, therefore $\vdash A \to B$.

Then dependent product (or dependent function)

- $x : Author \vdash f(x) : Book(x)$
- $\vdash f : \prod_{x:Author} Book(x)$

In the case of a proposition:

- $x : Author \vdash f(x) : British(x)$
- $\vdash f : \prod_{x:Author} British(x)$

Anaphoric pronouns

- Whenever someone's child dishonours them, they punish them for it.

Anaphoric pronouns

- Whenever someone's child dishonours them, they punish them for it.
- $x : Person, y : Child(x), z : Dishonour(x, y) \vdash p : Punish(x, y, z)$

Identity types

Given a type A and two elements $a, b : A$, we can form the identity type $Id_A(a, b)$.

Identity types

Given a type A and two elements $a, b : A$, we can form the identity type $Id_A(a, b)$.

We don't impose the restriction that any two proofs of identity, p, q, in such a type $Id_A(a, b)$ are equal. This allows us to say that a type is a proposition, a set, a groupoid, ...

Identity types

Given a type A and two elements $a, b : A$, we can form the identity type $Id_A(a, b)$.

We don't impose the restriction that any two proofs of identity, p, q, in such a type $Id_A(a, b)$ are equal. This allows us to say that a type is a proposition, a set, a groupoid, ...

$$X : Type \vdash isContr(X) \equiv \sum_{(x : X)} \prod_{(y : X)} Id_X(x, y) : Type.$$

Identity types

Given a type A and two elements $a, b : A$, we can form the identity type $Id_A(a, b)$.

We don't impose the restriction that any two proofs of identity, p, q, in such a type $Id_A(a, b)$ are equal. This allows us to say that a type is a proposition, a set, a groupoid, ...

$$X : Type \vdash isContr(X) \equiv \sum_{(x : X)} \prod_{(y : X)} Id_X(x, y) : Type.$$

We also have a hierarchy of universes of types, $Type_i$.

Univalent universes are such that the types $Id_{Type}(A, B)$ and $Equiv(A, B)$ are equivalent.

Univalent universes are such that the types $Id_{Type}(A, B)$ and $Equiv(A, B)$ are equivalent.

From an equivalence, f, between A and B, we have an element, $ua(f) : Id_{Type}(A, B)$, which we use to transfer constructions on A to B.

Univalent universes are such that the types $Id_{Type}(A, B)$ and $Equiv(A, B)$ are equivalent.

From an equivalence, f, between A and B, we have an element, $ua(f) : Id_{Type}(A, B)$, which we use to transfer constructions on A to B.

'*ua*' stands for 'univalence axiom', not given as a rule. Cubical type theory looks to specify univalence in a computational way.

Univalent universes are such that the types $Id_{Type}(A, B)$ and $Equiv(A, B)$ are equivalent.

From an equivalence, f, between A and B, we have an element, $ua(f) : Id_{Type}(A, B)$, which we use to transfer constructions on A to B.

'*ua*' stands for 'univalence axiom', not given as a rule. Cubical type theory looks to specify univalence in a computational way.

We also allow the formation of HITs (e.g., the type of natural numbers, quotients of sets by relations, the circle, etc.), and we have HoTT.

FOL as a restricted type theory

Type-formation in the restricted FOL setting:

- 0 appears as the proposition False.
- $\times$ appears as conjunction.
- $\sum$ appears as existential quantification.
- $\prod$ appears as universal quantification.
- Function types of propositions correspond to implication.
- Elements of function types in $[domain^n, domain]$ are n-ary functions.
- The identity type appears as equality on the domain.

FOL is a restricted form of dependent type theory, which then requires expressive power to be restored, say, via set theory.

Compare with purchasing a sports car, fitting it with a speed-limiter and than bolting on an external engine.

FOL is a restricted form of dependent type theory, which then requires expressive power to be restored, say, via set theory.

Compare with purchasing a sports car, fitting it with a speed-limiter and than bolting on an external engine.

HoTT is the internal language of elementary $(\infty, 1)$-toposes. It allows *synthetic homotopy theory* with no need to formalize simplicial sets, etc.

Cohomology, group actions,...

FOL is a restricted form of dependent type theory, which then requires expressive power to be restored, say, via set theory.

Compare with purchasing a sports car, fitting it with a speed-limiter and than bolting on an external engine.

HoTT is the internal language of elementary $(\infty, 1)$-toposes. It allows *synthetic homotopy theory* with no need to formalize simplicial sets, etc.

Cohomology, group actions,...

It can be made classical.

Of course, just as with FOL we may define FOL-theories by specifying predicates, constants, functions, and axioms, so with HoTT we may provide HoTT-theories by specifying generating types, terms, and axioms.

'*Type theory*' is very ambiguous.

Little has been said about applying it.

Applied type theory

Applied type theory is everywhere in computer science, e.g., in the use of databases.

Applied type theory

Applied type theory is everywhere in computer science, e.g., in the use of databases.

Given there are at least as many As as Cs, and a pairing of As and Bs, that there are at least as many Bs as Cs.

Applied type theory

Applied type theory is everywhere in computer science, e.g., in the use of databases.

Given there are at least as many As as Cs, and a pairing of As and Bs, that there are at least as many Bs as Cs.

From the perspective of proposition-as-some-types, this is no more philosophically puzzling than applied logic.

Perhaps you require a complete system for your basic logic, but there were other choices, such as DFOL (dependently typed first-order logic).

Perhaps you require a complete system for your basic logic, but there were other choices, such as DFOL (dependently typed first-order logic).

The choice of FOL + ZFC shapes the Anglophone philosophical terrain. We need to explore type theory in the philosophy of language and metaphysics.

Why modalities?

- Everyday: invariance under modification

Why modalities?

- Everyday: invariance under modification
- Computing: staged computation; trusted communications; resources …

Why modalities?

- Everyday: invariance under modification
- Computing: staged computation; trusted communications; resources ...
- Mathematics: synthetic axioms for continuity, smoothness, stable homotopy, ...

Why modalities?

- Everyday: invariance under modification
- Computing: staged computation; trusted communications; resources ...
- Mathematics: synthetic axioms for continuity, smoothness, stable homotopy, ...
- Philosophy: metaphysics – possible worlds, temporal logic, epistemic logic.

Why modalities?

- Everyday: invariance under modification
- Computing: staged computation; trusted communications; resources ...
- Mathematics: synthetic axioms for continuity, smoothness, stable homotopy, ...
- Philosophy: metaphysics – possible worlds, temporal logic, epistemic logic.
- Linguistics: pragmatics

S4 and monads

- $p \to \Diamond p$
- $\Diamond\Diamond p \to \Diamond p$

S4 and monads

- $p \to \Diamond p$
- $\Diamond\Diamond p \to \Diamond p$

Monads in category theory arise from adjunctions.

S4 and monads

- $p \to \Diamond p$
- $\Diamond\Diamond p \to \Diamond p$

Monads in category theory arise from adjunctions.

One natural example to consider is the adjoint triple corresponding to dependent sum and dependent product.

- $w : World \vdash A(w) : Prop$
- $\prod_{w:World} A(w), \sum_{w:World} A(w)$

S4 and monads

- $p \to \Diamond p$
- $\Diamond\Diamond p \to \Diamond p$

Monads in category theory arise from adjunctions.

One natural example to consider is the adjoint triple corresponding to dependent sum and dependent product.

- $w : World \vdash A(w) : Prop$
- $\prod_{w:World} A(w), \sum_{w:World} A(w)$

- 'For all worlds, A'.
- 'The worlds where A', truncated to 'In some world, A'.

It's very natural to apply modalities to all types rather than just propositions.

It's very natural to apply modalities to all types rather than just propositions.

- 'A choice of A for each world'.
- 'A possible A'.

It's very natural to apply modalities to all types rather than just propositions.

- 'A choice of A for each world'.
- 'A possible A'.

We can also choose a group for W.

It's very natural to apply modalities to all types rather than just propositions.

- 'A choice of A for each world'.
- 'A possible A'.

We can also choose a group for W.

More generally, we could use any $f : W \to V$, $R : S \rightrightarrows T$.

The approach of Licata, Shulman and Riley to modal dependent type theory is promising.

Synthetic Mathematics in Modal Dependent Type Theories. 6-talk tutorial joint with Felix Wellen. Hausdorff Institute Workshop on Types, Homotopy Type theory, and Verification, 2018.

(Dan Licata's site, https://dlicata.wescreates.wesleyan.edu/pubs.html)

Differential cohesive modalities allows the expression of much of modern differential cohomology, just what is required to understand connections, gauge fields, etc. as used in mathematical physics.

Differential cohesive modalities allows the expression of much of modern differential cohomology, just what is required to understand connections, gauge fields, etc. as used in mathematical physics.

> *The amount of gauge QFT notions naturally formalized here in cohesive homotopy type theory seems to be remarkable, emphasizing the value of a formal, logical, approach to concepts like smoothness and cohomology. (Urs Schreiber, Michael Shulman, Quantum Gauge Field Theory in Cohesive Homotopy Type Theory, arXiv:1408.0054)*

To conclude

We're in an exploratory phase. Modal HoTT is *disruptive technology*.

- Our understanding of the world as typed.
- A synthetic theory of structures.
- The untapped world of modal type theory.