Computational Higher-Dimensional Type Theory

Carlo Angiuli[1] Robert Harper[1] Todd Wilson[2]

[1]Carnegie Mellon University

[2]California State University, Fresno

January 20, 2017

Homotopy Type Theory (HoTT)

Extends Martin-Löf dependent type theory with:

- Univalence axiom.
- Higher inductive types.

Captures higher-dimensional (homotopical, topological) structure.

Although this talk isn't about HoTT, let's start by reviewing it.

Homotopy Type Theory (HoTT)

Useful for constructive, mechanized (in Coq/Agda/Lean) proofs of theorems from algebraic topology and homotopy theory.

- Seifert-van Kampen theorem (Favonia, Shulman).
- Eilenberg-Mac Lane spaces (Licata, Finster).
- Mayer-Vietoris theorem (Cavallo).
- Blakers-Massey theorem (Favonia, Finster, Licata, Lumsdaine).
- Cayley-Dickson construction (Buchholtz, Rijke).

Univalence Axiom

Identity type $\mathbf{Id}_A(M, N)$ says that M, N are equal.

$\mathbf{Id}_A(M, N) \implies$ can always replace M with N.

$\mathbf{Id}_{\mathbf{Type}}(A, B) \implies$ can coerce elements of A to B.

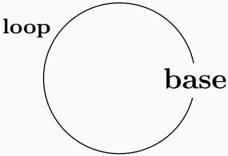
Univalence*: Any isomorphism between A, B yields $\mathbf{Id}_{\mathbf{Type}}(A, B)$.

Univalence says all isomorphisms yield proofs of identity, whose coercions are implemented by the isomorphism.

Higher Inductive Types

Inductive types with constructors for A and $\mathbf{Id}_A(M, N)$!

$$\frac{}{\Gamma \vdash \mathbf{base} : \mathbb{S}^1} \qquad \frac{}{\Gamma \vdash \mathbf{loop} : \mathbf{Id}_{\mathbb{S}^1}(\mathbf{base}, \mathbf{base})}$$

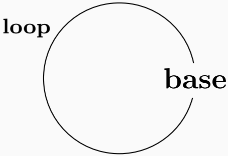


We draw this HIT as a circle because it actually behaves like one, when identity proofs are interpreted as paths.

Higher Inductive Types

Inductive types with constructors for A and $\mathbf{Id}_A(M, N)$!

$$\frac{}{\Gamma \vdash \mathbf{base} : \mathbb{S}^1} \qquad \frac{}{\Gamma \vdash \mathbf{loop} : \mathbf{Id}_{\mathbb{S}^1}(\mathbf{base}, \mathbf{base})}$$



Higher-dimensional interpretation: identity = paths.

We draw this HIT as a circle because it actually behaves like one, when identity proofs are interpreted as paths.

Propositions-as-Types Correspondence

Also known as the Curry-Howard isomorphism, or the Brouwer-Heyting-Kolmogorov explanation.

$$\text{logics} \Longleftrightarrow \text{programming languages}$$
$$\text{propositions} \Longleftrightarrow \text{types}$$
$$\text{proofs of a proposition} \Longleftrightarrow \text{programs of a type}$$

A key feature of type theory is the correspondence between proofs and programs.

Proofs as Programs?

Adding new axioms (UA, HITs) is fine in a logic, but in a PL, you can't just postulate new programs in existing types!

```
datatype bool = true | false

if ... then 0 else 1 : int
```

Axioms disrupt PAT, causing existing programs to become stuck. This ruins computation at every type.

Proofs as Programs?

Adding new axioms (UA, HITs) is fine in a logic, but in a PL, you can't just postulate new programs in existing types!

```
datatype bool = true | false | file_not_found
```

```
if file_not_found then 0 else 1 : int
```

Axioms disrupt PAT, causing existing programs to become stuck. This ruins computation at every type.

Proofs as Programs?

Adding new axioms (UA, HITs) is fine in a logic, but in a PL, you can't just postulate new programs in existing types!

```
datatype bool = true | false | file_not_found
```

```
if file_not_found then 0 else 1 : int
```

Destroys `int`!

Axioms disrupt PAT, causing existing programs to become stuck. This ruins computation at every type.

Proofs as Programs?

Exactly what happens with UA+HITs in HoTT: new $\mathbf{Id}_A(M, N)$ proofs not handled by the $\mathbf{Id}$ eliminator!

Inconvenient, even if you only care about logic.

Brunerie Constant

Guillaume Brunerie successfully computed an invariant as $\mathbb{Z}/k\mathbb{Z}$ where $\cdot \vdash k : \mathbb{N}$ (14 pages, 2013).

Required a PhD thesis (129 pages, 2016) to show $k = 2$.

Propositions-as-types $\implies$ k computes to 2!

Computational Cubical Type Theory

We define a (non-HoTT) higher-dimensional type theory for which propositions-as-types works. Core idea is to extend:

Nuprl, Constable, *et al.* (1985–). Computational type theory.

Constructive Mathematics and Computer Programming, Martin-Löf (1979). Meaning explanations of type theory.

Computational Type Theory

Given a programming language $M \Downarrow V$, types are defined as classifications of programs according to their behavior.

$$\cdot \gg M \in \mathbf{bool} \iff M \Downarrow \mathbf{true} \text{ or } M \Downarrow \mathbf{false}$$

$$\cdot \gg M \in A \to B \iff M \Downarrow \lambda a.M' \wedge \forall N \in A,\ M'[N/a] \in B$$

Closely related to logical relations and to refinements!

We adopt the $\gg$ and $\in$ notation to avoid confusion with other type theories.

Computational Type Theory

The familiar rules of type theory hold relative to these definitions!

$$\frac{M \in \mathbf{bool} \to \mathbf{bool} \quad N \in \mathbf{bool}}{M\ N \in \mathbf{bool}}$$

Computational Type Theory

The familiar rules of type theory hold relative to these definitions!

$$\frac{M \in \mathbf{bool} \to \mathbf{bool} \quad N \in \mathbf{bool}}{M\ N \in \mathbf{bool}}$$

$$\Updownarrow$$

$$\frac{\begin{array}{c} M \Downarrow \lambda a.M' \ \wedge \forall N' \in \mathbf{bool},\ M'[N'/a] \in \mathbf{bool} \\ N \Downarrow \mathbf{true} \text{ or } \mathbf{false} \end{array}}{M\ N \Downarrow \mathbf{true} \text{ or } \mathbf{false}}$$

Computational Type Theory

Constructive (à la Brouwer): truth is defined by algorithms.

- Not defined by enumerating proof rules.
- Programs have many types, some more obvious than others! (Ranges from “read the program” to “prove a theorem.”)

Types Internalize Judgments

Types internalize concepts present in the judgmental framework.

$$\frac{A \text{ true} \quad B \text{ true}}{A \wedge B \text{ true}}$$

$$\frac{A \text{ true}}{A \vee B \text{ true}} \qquad \frac{B \text{ true}}{A \vee B \text{ true}}$$

Writing multiple premises to a rule implicitly invokes conjunction; writing multiple rules with the same conclusion implicitly invokes disjunction.

Types Internalize Judgments

Originally, closed $\mathbf{Id}_A(M, N)$ determined by equality judgment.

In HoTT,

- $\mathbf{Id}_{\mathbb{S}^1}(\mathbf{base}, \mathbf{base})$ determined by definition of $\mathbb{S}^1$.
- $\mathbf{Id}_{\mathbf{Type}}(A, B)$ determined by isomorphisms.

What judgmental concept does the HoTT identity type internalize?

Path Judgments

Canonicity for 2-Dimensional Type Theory, Licata and Harper (POPL 2012): Define a judgment for paths.

$$\Gamma \vdash M : A$$

$$\Gamma \vdash P : M \simeq N : A$$

We can organize iterated path judgments cubically.

Path Judgments

Canonicity for 2-Dimensional Type Theory, Licata and Harper (POPL 2012): Define a judgment for paths.

$$\Gamma \vdash M : A$$

$$\Gamma \vdash P : M \simeq N : A$$

$$\Gamma \vdash H : P \simeq Q : M \simeq N : A$$

We can organize iterated path judgments cubically.

Path Judgments

Canonicity for 2-Dimensional Type Theory, Licata and Harper (POPL 2012): Define a judgment for paths.

$$\Gamma \vdash M : A$$

$$\Gamma \vdash P : M \simeq N : A$$

$$\Gamma \vdash H : P \simeq Q : M \simeq N : A$$

We can organize iterated path judgments cubically.

Path Judgments

Canonicity for 2-Dimensional Type Theory, Licata and Harper (POPL 2012): Define a judgment for paths.

$$\Gamma \vdash M : A$$

$$\Gamma \vdash P : M \simeq N : A$$

$$\Gamma \vdash H : P \simeq Q : M \simeq N : A$$

We can organize iterated path judgments cubically.

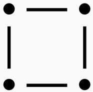
Path Judgments

Canonicity for 2-Dimensional Type Theory, Licata and Harper (POPL 2012): Define a judgment for paths.

$$\Gamma \vdash M : A$$

$$\Gamma \vdash P : M \simeq N : A$$

$$\Gamma \vdash H : P \simeq Q : M \simeq N : A$$



We can organize iterated path judgments cubically.

Cubical Programs

Cubes. Kan (1955), Bezem, Coquand, Huber (2014).

Programs representing points, lines, squares, cubes. . .

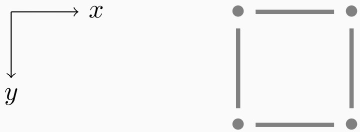
n-dimensional programs parametrized by n dimension variables.

- $\mathbf{base}$ is a point (no dimensions).
- $\mathbf{loop}_x$ is a line (one dimension, x).

Cubical Programs

Imagine a square M as a map $M(x, y) : [0, 1]^2 \to \mathbf{Term}$.

Substituting for a dimension computes an aspect.

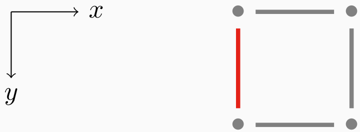


Dimension substitutions compute aspects (faces, diagonals) of cubes. Substitution satisfies expected geometric laws.

Cubical Programs

Imagine a square M as a map $M(x, y) : [0, 1]^2 \to \mathbf{Term}$.

Substituting for a dimension computes an aspect.



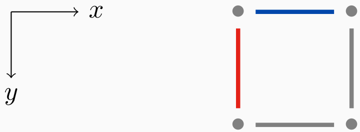
$M\langle 0/x\rangle$

Dimension substitutions compute aspects (faces, diagonals) of cubes. Substitution satisfies expected geometric laws.

Cubical Programs

Imagine a square M as a map $M(x, y) : [0, 1]^2 \to \mathbf{Term}$.

Substituting for a dimension computes an aspect.



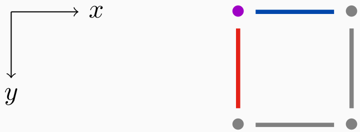
$M\langle 0/x\rangle$ $M\langle 0/y\rangle$

Dimension substitutions compute aspects (faces, diagonals) of cubes. Substitution satisfies expected geometric laws.

Cubical Programs

Imagine a square M as a map $M(x, y) : [0, 1]^2 \to \mathbf{Term}$.

Substituting for a dimension computes an aspect.



$$M\langle 0/x\rangle\langle 0/y\rangle \;=\; M\langle 0/y\rangle\langle 0/x\rangle$$

Dimension substitutions compute aspects (faces, diagonals) of cubes. Substitution satisfies expected geometric laws.

Cubical Programs

Can evaluate programs of any dimension.

$$\frac{}{\mathbf{base\ val}} \qquad \frac{}{\mathbf{loop}_x\ \mathbf{val}}$$

$$\frac{}{\mathbf{loop}_0 \longmapsto \mathbf{base}} \qquad \frac{}{\mathbf{loop}_1 \longmapsto \mathbf{base}}$$

expected

The bottom rules ensure that the faces of $\mathbf{loop}_x$ are both $\mathbf{base}$.

Cubical Judgments

Judgments at every dimension.

M is a point	$\Gamma \gg M \in A\ [\varnothing]$
... line	$\Gamma \gg M \in A\ [x]$
... square	$\Gamma \gg M \in A\ [x, y]$
... cube	$\Gamma \gg M \in A\ [x, y, z]$

Cubical Judgments

The cubical judgments

$$\Gamma \gg A \doteq B \ \mathbf{pretype}\ [\Psi]$$

$$\Gamma \gg M \doteq N \in A\ [\Psi]$$

are defined by the cubical meaning explanations.

Closed Cubical Judgments

$$A \qquad \mathbf{pretype}\ [\Psi]$$

means $A \Downarrow A_0$,

and we specify the canonical Ψ -elements of A_0 , and
when two canonical Ψ -elements of A_0 are equal,

ψ is an arbitrary dimension substitution from Ψ to Ψ'.

Closed Cubical Judgments

$$A \quad \mathbf{pretype}\ [\Psi]$$

means $\forall\psi : \Psi' \to \Psi,\ \ A\psi \Downarrow A_0$,

and we specify the canonical Ψ'-elements of A_0 , and when two canonical Ψ'-elements of A_0 are equal,

---

ψ is an arbitrary dimension substitution from Ψ to Ψ'.

Closed Cubical Judgments

$$A \doteq B \ \textbf{pretype}\ [\Psi]$$

means $\forall \psi : \Psi' \to \Psi,\ \ A\psi \Downarrow A_0$ and $B\psi \Downarrow B_0$,

and we specify the canonical Ψ'-elements of A_0 (resp., B_0), and when two canonical Ψ'-elements of A_0 (resp., B_0) are equal,

and the canonical Ψ'-elements of A_0 and B_0 are the same, with the same equality.

ψ is an arbitrary dimension substitution from Ψ to Ψ'.

Closed Cubical Judgments

$$M \in A\ [\Psi]$$

means $\forall\psi : \Psi' \to \Psi$, $M\psi \Downarrow M_0$,

and M_0 is a canonical Ψ'-element of A_0 (where $A\psi \Downarrow A_0$).

The highlighted condition only makes sense if we presuppose that $A\ \mathbf{pretype}\ [\Psi]$.

Closed Cubical Judgments

$$M \qquad \in A\ [\Psi]$$

presupposing $A\ \mathbf{pretype}\ [\Psi]$,

means $\forall\psi : \Psi' \to \Psi$, $M\psi \Downarrow M_0$,

and M_0 is a canonical Ψ'-element of A_0 (where $A\psi \Downarrow A_0$).

The highlighted condition only makes sense if we presuppose that $A\ \mathbf{pretype}\ [\Psi]$.

Closed Cubical Judgments

$$M \doteq N \in A\ [\Psi]$$

presupposing A $\mathbf{pretype}$ $[\Psi]$,

means $\forall\psi : \Psi' \to \Psi$, $M\psi \Downarrow M_0$ and $N\psi \Downarrow N_0$,

and M_0 and N_0 ~~is a~~ are equal canonical Ψ'-elements of A_0 (where $A\psi \Downarrow A_0$).

The highlighted condition only makes sense if we presuppose that A $\mathbf{pretype}$ $[\Psi]$.

Open Cubical Judgments

$$c : C \gg A \doteq B \ \textbf{pretype}\ [\Psi]$$

$$\text{when } C \ \textbf{pretype}\ [\Psi],$$
$$, \forall M \quad \in C \quad [\Psi\ ],$$
$$A \quad [M/c] \doteq B \quad [M\ /c]\ \textbf{pretype}\ [\Psi\ ].$$

$$c : C \gg N \doteq N' \in A\ [\Psi]$$

$$\text{when } C \ \textbf{pretype}\ [\Psi],$$
$$, \forall M \quad \in C \quad [\Psi\ ],$$
$$N \quad [M/c] \doteq N' \quad [M\ /c] \in A \quad [M/c]\ [\Psi\ ].$$

Open judgments mean that, for all equal elements of C, the corresponding closed judgments hold.

Open Cubical Judgments

$$c : C \gg A \doteq B \ \mathbf{pretype}\ [\Psi]$$

when C $\mathbf{pretype}\ [\Psi]$,
$, \forall M \doteq M' \in C \quad [\Psi\ ],$
$A \ \ [M/c] \doteq B \ \ [M'/c]\ \mathbf{pretype}\ [\Psi\ ].$

$$c : C \gg N \doteq N' \in A\ [\Psi]$$

when C $\mathbf{pretype}\ [\Psi]$,
$, \forall M \doteq M' \in C \quad [\Psi\ ],$
$N \ \ [M/c] \doteq N' \ \ [M'/c] \in A \ \ [M/c]\ [\Psi\ ].$

Open judgments mean that, for all equal elements of C, the corresponding closed judgments hold.

Open Cubical Judgments

$$c : C \gg A \doteq B \ \mathbf{pretype}\ [\Psi]$$

$$\text{when } C\ \mathbf{pretype}\ [\Psi],$$
$$\forall \psi : \Psi' \to \Psi,\ \forall M \doteq M' \in C\psi\ [\Psi'],$$
$$A\psi[M/c] \doteq B\psi[M'/c]\ \mathbf{pretype}\ [\Psi'].$$

$$c : C \gg N \doteq N' \in A\ [\Psi]$$

$$\text{when } C\ \mathbf{pretype}\ [\Psi],$$
$$\forall \psi : \Psi' \to \Psi,\ \forall M \doteq M' \in C\psi\ [\Psi'],$$
$$N\psi[M/c] \doteq N'\psi[M'/c] \in A\psi[M/c]\ [\Psi'].$$

Open judgments mean that, for all equal elements of C, the corresponding closed judgments hold.

Cubical Type Systems

Definition

A partial equivalence relation is a symmetric and transitive relation.

Canonical pretype equality: $\approx^{\Psi}$ is a PER over Ψ-dim'l values.

Canonical element equality: $\approx^{\Psi}_{-}$ is a $(\approx^{\Psi})$-indexed family of PERs over Ψ-dim'l values.

Cubical Type Systems

Definition

A cubical type system is a pair $(\approx^{-}, \approx^{-}_{-})$.

$$A \doteq B \ \mathbf{pretype}\ [\Psi]$$

$$\forall \psi : \Psi' \to \Psi,\ A\psi \Downarrow A_0, B\psi \Downarrow B_0,\ A_0 \approx^{\Psi'} B_0$$

$$M \doteq N \in A\ [\Psi]$$

$$\forall \psi : \Psi' \to \Psi,\ M\psi \Downarrow M_0, N\psi \Downarrow N_0,\ M_0 \approx^{\Psi'}_{A_0} N_0 \text{ where } A\psi \Downarrow A_0.$$

The judgments have meaning in any cubical type system.

Cubical Type Systems

We want a cubical type system with types!

A cubical type system has the (strict) booleans when:

- $\mathbf{bool} \approx^{\Psi} \mathbf{bool}$
- $M_0 \approx^{\Psi}_{\mathbf{bool}} N_0 \iff (M_0 = N_0 = \mathbf{true} \vee M_0 = N_0 = \mathbf{false})$

We place conditions on CTSes to ensure they have certain type formers.

Cubical Type Systems

Theorem

In every cubical type system with strict booleans,

$$\frac{}{\Gamma \gg \textbf{bool pretype}\ [\Psi]} \qquad \frac{}{\Gamma \gg \textbf{true} \in \textbf{bool}\ [\Psi]} \qquad \cdots$$

Theorem (Canonicity)

If $\cdot \gg M \in \textbf{bool}\ [\Psi]$ *then* $M \Downarrow \textbf{true}$ *or* $M \Downarrow \textbf{false}$.

Canonicity (which ensures proper PAT) here holds by definition; the hard part is proving the rules of type theory.

Coherence of Aspects

$$
\begin{array}{ccc}
 & M & \\
\swarrow & & \searrow \\
M\langle 0/x\rangle \Downarrow V & & M\langle 0/y\rangle \Downarrow V' \\
\searrow & & \swarrow \\
V\langle 0/y\rangle & \stackrel{?}{=} & V'\langle 0/x\rangle
\end{array}
$$

In the paper, we also have a coherence condition between evaluation and dimension substitution. . .

Kan Conditions

A **type** $[\Psi]$ when A **pretype** $[\Psi]$ and satisfies Kan conditions.

Generalized coercion:

$$\begin{array}{ccc} M & & \\ \rotatebox{90}{$\in$} & & \\ A\langle 0/x\rangle & \xrightarrow[A]{\hspace{4em}} & A\langle 1/x\rangle \end{array}$$

. . . and the Kan conditions, to ensure types have generalized coercion and box-filling.

Kan Conditions

A **type** $[\Psi]$ when A **pretype** $[\Psi]$ and satisfies Kan conditions.

Generalized coercion:

$$\begin{array}{ccc} M & \dashrightarrow & \mathsf{coe}^{0 \rightsquigarrow 1}_{x.A}(M) \\ \pitchfork & & \pitchfork \\ A\langle 0/x \rangle & \xrightarrow[A]{\qquad\qquad} & A\langle 1/x \rangle \end{array}$$

. . . and the Kan conditions, to ensure types have generalized coercion and box-filling.

Kan Conditions

A **type** $[\Psi]$ when A **pretype** $[\Psi]$ and satisfies Kan conditions.

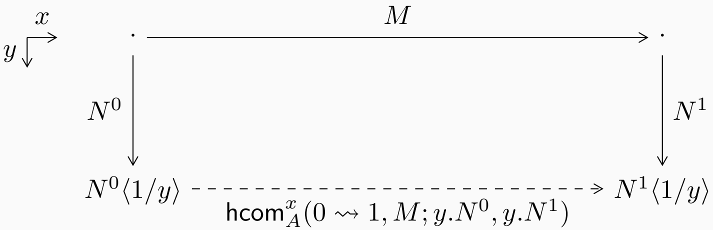
Generalized coercion:

$$\begin{array}{ccc} M & \xrightarrow{\mathsf{coe}_{x.A}^{0 \rightsquigarrow x}(M)} & \mathsf{coe}_{x.A}^{0 \rightsquigarrow 1}(M) \\ \Cap & \Cap & \Cap \\ A\langle 0/x \rangle & \xrightarrow[A]{} & A\langle 1/x \rangle \end{array}$$

. . . and the Kan conditions, to ensure types have generalized coercion and box-filling.

Kan Conditions

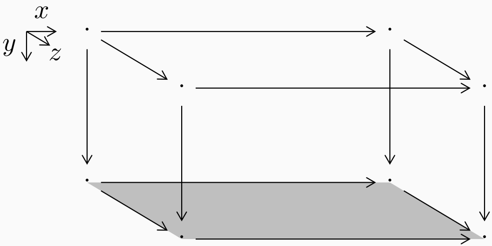
Box filling.
(Ensures symmetry, transitivity, associativity of transitivity. . .)



For any three sides of a square, the fourth exists; for any three or five sides of a cube, the sixth exists.

Kan Conditions

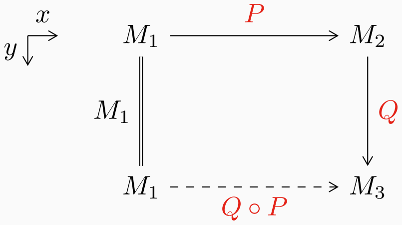
Box filling.
(Ensures symmetry, transitivity, associativity of transitivity. . .)



For any three sides of a square, the fourth exists; for any three or five sides of a cube, the sixth exists.

Kan Conditions

Proving transitivity:



So What?

Results

- A higher-dimensional type theory whose proofs run.
- Defined cubical logical relations / cubical meaning explanations / cubical realizability.
- First canonicity theorem for a higher-dimensional type theory!
  - Dependent functions, dependent pairs, identifications.
  - Some HITs (circle, weak booleans).
  - Univalence for exact isomorphisms. (New!)
  - Contains computational type theory.

Related Work

Instead of (cubical) meaning explanations, one could...

Define a logic $\Gamma \vdash M : A$ by rules (M is a formal proof of A).

To recover computation, define proof reduction for $\Gamma \vdash M : A$,

$$\Gamma \vdash M \succ N : A$$

where $\Gamma \vdash N : A$.

Related Work

Cubical type theories in the logical tradition by

- Licata and Brunerie (2014).
- Cohen, Coquand, Huber, Mörtberg (2016).
  - Has univalence and universes.
  - Proof reduction is possible, satisfies canonicity (Huber, 2016).

Future Work

- Continue implementation in RedPRL (Sterling, *et al.*).
- Full univalence and universes?
- Other HITs?

Thanks!

cs.cmu.edu/~cangiuli