Computational Semantics of Cartesian Cubical Type Theory

Carlo Angiuli

September 19, 2019

Carnegie Mellon University

Dependent type theory

Dependent type theory is a language for mathematical reasoning.

Implemented in many proof assistants (software for developing and checking proofs) — Agda, Coq, Lean, Nuprl, . . .

- Four color theorem [Gonthier 2008]
- Feit–Thompson theorem [Gonthier *et al.* 2013]

Dependent type theory

Dependent type theory is a language for mathematical reasoning.

Implemented in many proof assistants (software for developing and checking proofs) — Agda, Coq, Lean, Nuprl, …

- Four color theorem [Gonthier 2008]
- Feit–Thompson theorem [Gonthier *et al.* 2013]
- CompCert C compiler [Leroy 2009]
- Multi-Paxos [Schiper *et al.* 2014]
- mbedTLS HMAC-DRBG [Ye *et al.* 2017]

Dependent type theory

Proof assistants reduce mathematical proofs to primitive inferences.

$$\frac{\Gamma, a : A \vdash M : B(a)}{\Gamma \vdash \lambda a.M : (a : A) \longrightarrow B(a)} \qquad \frac{\Gamma \vdash M : (a : A) \longrightarrow B(a) \qquad \Gamma \vdash N : A}{\Gamma \vdash M\, N : B(N)}$$

Dependent type theory

Proof assistants reduce mathematical proofs to primitive inferences.

$$\frac{\Gamma, a : A \vdash M : B(a)}{\Gamma \vdash \underbrace{\lambda a.M : (a : A) \longrightarrow B(a)}_{\text{dependent function}}} \qquad \frac{\Gamma \vdash M : (a : A) \longrightarrow B(a) \qquad \Gamma \vdash N : A}{\Gamma \vdash M\,N : B(N)}$$

$(n : \mathbb{N}) \longrightarrow \mathsf{List}(n)$

Dependent type theory

Proof assistants reduce mathematical proofs to primitive inferences.

$$\frac{\Gamma, a : A \vdash M : B(a)}{\Gamma \vdash \underbrace{\lambda a.M : (a : A) \to B(a)}}$$

$$\frac{\Gamma \vdash M : (a : A) \to B(a) \qquad \Gamma \vdash N : A}{\Gamma \vdash M\,N : B(N)}$$

dependent function

$(n : \mathbb{N}) \to \mathsf{List}(n)$

proof of $\forall a \in A.B(a)$

$\forall n \in \mathbb{N}.\mathsf{isEven}(2n)$

Dependent type theory

Proof assistants reduce mathematical proofs to primitive inferences.

$$\frac{\Gamma, a : A \vdash M : B(a)}{\Gamma \vdash \underbrace{\lambda a.M : (a : A) \to B(a)}} \qquad \frac{\Gamma \vdash M : (a : A) \to B(a) \qquad \Gamma \vdash N : A}{\Gamma \vdash M\, N : B(N)}$$

dependent function $\qquad (n : \mathbb{N}) \to \mathsf{List}(n)$

proof of $\forall a \in A. B(a) \qquad \forall n \in \mathbb{N}.\mathsf{isEven}(2n)$

Dependent types play the role of both sets (*e.g.*, functions) and propositions (*e.g.*, $\forall$).

Identity types

$\mathrm{Id}_A(a, b)$ — a and b are equal in A [Martin-Löf 1975].

- Equality is reflexive — $\mathrm{refl}(a) : \mathrm{Id}_A(a, a)$
- "Everything respects equality" — if it holds for $\mathrm{refl}(a)$, it holds for any $p : \mathrm{Id}_A(a, b)$

Identity types

Everything respects equality $\implies$ symmetry, transitivity, and coercion.

Identity types

Everything respects equality $\implies$ symmetry, transitivity, and coercion.

$$\mathsf{refl}(a) : \mathsf{Id}_A(a, a)$$

$$\rightsquigarrow \; p : \mathsf{Id}_A(a, b)$$

$$\mathsf{Id}_A(b, a)$$

Identity types

Everything respects equality $\implies$ symmetry, transitivity, and coercion.

$$\mathsf{refl}(a) : \mathsf{Id}_A(a, a) \quad \overset{p \,:\, \mathsf{Id}_A(a,b)}{\rightsquigarrow} \quad \mathsf{Id}_A(b, a)$$

$$p : \mathsf{Id}_A(a, b) \quad \overset{q \,:\, \mathsf{Id}_A(b,c)}{\rightsquigarrow} \quad \mathsf{Id}_A(a, c)$$

Everything respects equality $\Longrightarrow$ symmetry, transitivity, and coercion.

$$\mathsf{refl}(a) : \mathsf{Id}_A(a, a) \quad \overset{p \,:\, \mathsf{Id}_A(a,b)}{\rightsquigarrow} \quad \mathsf{Id}_A(b, a)$$

$$p : \mathsf{Id}_A(a, b) \quad \overset{q \,:\, \mathsf{Id}_A(b,c)}{\rightsquigarrow} \quad \mathsf{Id}_A(a, c)$$

$$\lambda a.a : A \to A \quad \overset{p \,:\, \mathsf{Id}_{\mathsf{Type}}(A,B)}{\rightsquigarrow} \quad \mathsf{coerce}(p) : A \to B$$

Identity types

Everything respects equality $\implies$ symmetry, transitivity, and coercion.

$$\mathrm{refl}(a) : \mathrm{Id}_A(a, a) \;\rightsquigarrow_{p : \mathrm{Id}_A(a, b)}\; \mathrm{Id}_A(b, a)$$

$$p : \mathrm{Id}_A(a, b) \;\rightsquigarrow_{q : \mathrm{Id}_A(b, c)}\; \mathrm{Id}_A(a, c)$$

$$\lambda a.a : A \to A \;\rightsquigarrow_{p : \mathrm{Id}_{\mathrm{Type}}(A, B)}\; \mathrm{coerce}(p) : A \to B$$

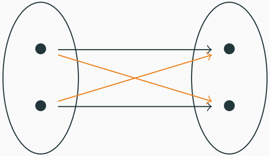
$\not\Longrightarrow$ UIP — there's only one proof of $\mathrm{Id}_A(a, b)$ [Hofmann, Streicher 1998].

Univalence

Univalence axiom [Voevodsky 2010] — for any $f : A \xrightarrow{\sim} B$,

- $\mathsf{univalence}(f) : \mathsf{Id}_{\mathsf{Type}}(A, B)$
- $\mathsf{coerce}(\mathsf{univalence}(f)) : A \to B$ applies f [Licata 2016]

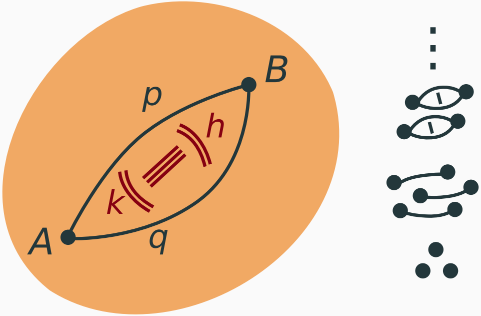
Contradicts UIP — $\mathsf{univalence}(\rightrightarrows) \neq \mathsf{univalence}(\times)$.



Higher-dimensional structure

Contradicts UIPIP, UIPIPIP… [Kraus, Sattler 2015].

This higher-dimensional structure is the same as seen in topology [Voevodsky 2012].

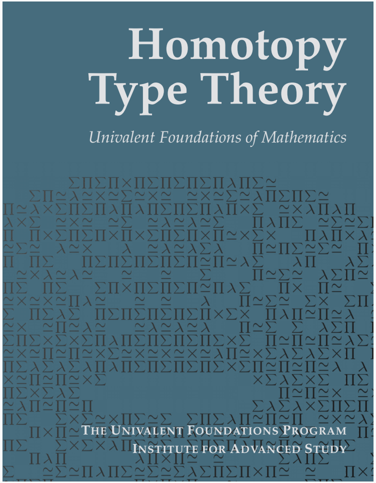


(Image credit: Favonia)

Homotopy type theory

Use type theory + univalence + higher inductive types
to prove theorems about topological spaces!

- Seifert–van Kampen theorem [Favonia, Shulman 2016]
- Blakers–Massey theorem [Favonia *et al.* 2016]
- Gysin sequence [Brunerie 2016]
- Serre spectral sequence [van Doorn 2018]



Computation I

Where do programs enter the picture?

Proof assistants can extract programs from proofs (*e.g.*, CompCert).

		ML Program	**Behavioral Spec.**
$f : (n : \mathbb{N}) \to \mathsf{List}(n)$	$\rightsquigarrow$	$f : \mathbb{N} \to \mathsf{List}$	where $\mathsf{length}(f(n)) = n$
$p : \mathsf{Id}_A(a, b)$	$\rightsquigarrow$		where $a = b$

Univalence $\implies$ p needed at runtime $\implies$ standard extraction is broken.

Computation II

Proof assistants use computation to silently discharge many equations.

$$\mathsf{append} : \mathsf{List}(n) \longrightarrow \mathsf{List}(m) \longrightarrow \mathsf{List}(n+m)$$

$$? : \mathsf{Id}_{\mathsf{List}(2)}(\underbrace{\mathsf{append}\ [\star]\ [\star]}_{:\ \mathsf{List}(1+1)}, [\star, \star])$$

If we can't compute with univalence, proofs using it are quite bureaucratic.

Canonicity

For these reasons, we want a computational semantics for a type theory with univalence — a way to regard proofs involving univalence as programs. Summarized by:

Theorem (Canonicity): Every $\cdot \vdash n : \mathbb{N}$ computes to, and is equal to, a concrete numeral.

Thesis statement

Higher-dimensional types classify higher-dimensional programs extensionally according to their behaviors.

I describe Cartesian cubical type theory ($\times$ 2) and present its computational semantics.

Published at POPL 2017 [A., Harper, Wilson] and CSL 2018 [A., Favonia, Harper].

Implemented in two proof assistants (`http://github.com/RedPRL`):

- **RedPRL** [A., Cavallo, Favonia, Harper, Sterling 2018]
- **`redtt`**

Outline

In the rest of this talk:

- Cartesian cubical type theory
- Computational semantics
- Taking stock

Cartesian cubical type theory

Coercion

$$\mathrm{coercion} : \mathrm{Id}_{\mathsf{Type}}(A, B) \to A \to B$$

Ordinary type theory — at runtime, only $\mathrm{coercion}(\mathrm{refl}(A)) = \lambda a.a : A \to A$.

With univalence — other arguments possible; coercion must do something!

Coercion

Computes by cases on the proof of $\mathrm{Id}_{\mathsf{Type}}(A, B)$. (Not on A, B!)

$$\mathrm{univalence}(f) : \mathrm{Id}_{\mathsf{Type}}(A, B)$$

Coercion

Computes by cases on the proof of $\mathrm{Id}_{\mathrm{Type}}(A, B)$. (Not on A, B!)

$$\mathrm{univalence}(f) : \mathrm{Id}_{\mathrm{Type}}(A, B)$$

$$\mathrm{refl}(A \longrightarrow B) : \mathrm{Id}_{\mathrm{Type}}(A \longrightarrow B, A \longrightarrow B)$$

$$p : \mathrm{Id}_{\mathrm{Type}}(B, B') \rightsquigarrow$$

$$\mathrm{Id}_{\mathrm{Type}}(A \longrightarrow B, A \longrightarrow B')$$

Coercion

Computes by cases on the proof of $\mathrm{Id}_{\mathrm{Type}}(A, B)$. (Not on A, B!)

$$\mathrm{univalence}(f) : \mathrm{Id}_{\mathrm{Type}}(A, B)$$

$$\mathrm{refl}(A \to B) : \mathrm{Id}_{\mathrm{Type}}(A \to B, A \to B)$$

$$p : \mathrm{Id}_{\mathrm{Type}}(B, B') \rightsquigarrow$$

$$\mathrm{Id}_{\mathrm{Type}}(A \to B, A \to B')$$

$$\mathrm{refl}(\mathrm{List}(n)) : \mathrm{Id}_{\mathrm{Type}}(\mathrm{List}(n), \mathrm{List}(n))$$

$$p : \mathrm{Id}_{\mathbb{N}}(n, m) \rightsquigarrow$$

$$\mathrm{Id}_{\mathrm{Type}}(\mathrm{List}(n), \mathrm{List}(m))$$

Coercion

Computes by cases on the proof of $\mathrm{Id}_{\mathrm{Type}}(A, B)$. (Not on A, B!)

$$\mathrm{univalence}(\mathrm{reverse}) : \mathrm{Id}_{\mathrm{Type}}(\mathrm{List}(n), \mathrm{List}(n))$$

$$\mathrm{refl}(A \longrightarrow B) : \mathrm{Id}_{\mathrm{Type}}(A \longrightarrow B, A \longrightarrow B)$$

$$p : \mathrm{Id}_{\mathrm{Type}}(B, B') \rightsquigarrow$$

$$\mathrm{Id}_{\mathrm{Type}}(A \longrightarrow B, A \longrightarrow B')$$

$$\mathrm{refl}(\mathrm{List}(n)) : \mathrm{Id}_{\mathrm{Type}}(\mathrm{List}(n), \mathrm{List}(n))$$

$$p : \mathrm{Id}_{\mathbb{N}}(n, m) \rightsquigarrow$$

$$\mathrm{Id}_{\mathrm{Type}}(\mathrm{List}(n), \mathrm{List}(m))$$

Paths

Where to put the proof that A equals B? In between them!

$$\begin{array}{ccc} A & \xrightarrow{\quad p(x) \quad} & B \\ \| & & \| \\ p(0) & & p(1) \end{array}$$

p is a "continuous function" out of $[0, 1] \subset \mathbb{R}$.

Interval and path types

Cubical type theories add an interval $\mathbb{I}$, path types...

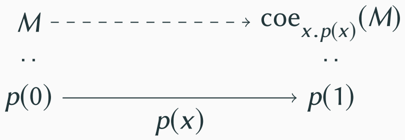
$$\frac{}{\Gamma \vdash 0 : \mathbb{I}} \qquad \frac{}{\Gamma \vdash 1 : \mathbb{I}} \qquad \frac{}{\Gamma, x : \mathbb{I} \vdash x : \mathbb{I}}$$

$$\frac{\Gamma, x : \mathbb{I} \vdash M(x) : A}{\Gamma \vdash \langle x \rangle M(x) : \mathrm{Path}_A(M(0), M(1))} \qquad \frac{\Gamma \vdash M : \overbrace{\mathrm{Path}_A(a_0, a_1)}^{\{p : \mathbb{I} \to A \mid p(0) = a_0 \wedge p(1) = a_1\}} \qquad \Gamma \vdash r : \mathbb{I}}{\Gamma \vdash M\, r : A}$$

Coercion

…and a new coercion operation.

$$\frac{\Gamma, x : \mathbb{I} \vdash p(x) : \mathrm{Type} \qquad \Gamma \vdash M : p(0)}{\Gamma \vdash \mathrm{coe}_{x.p(x)}(M) : p(1)}$$



Coercion

…and a new coercion operation.

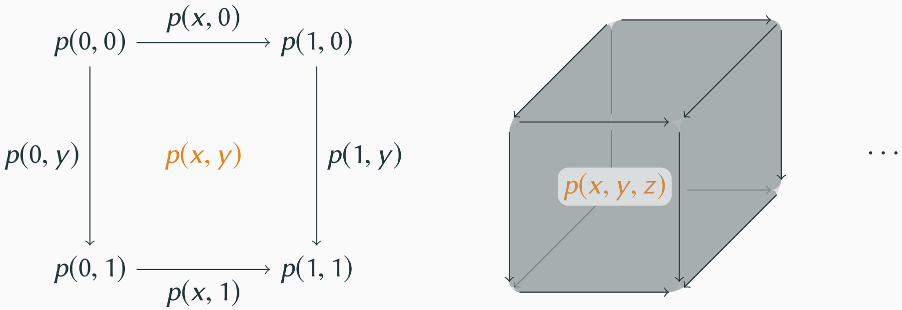
$$\frac{\Gamma \vdash r, r' : \mathbb{I} \qquad \Gamma, x : \mathbb{I} \vdash p(x) : \mathsf{Type} \qquad \Gamma \vdash M : p(r)}{\Gamma \vdash \mathsf{coe}^{r \rightsquigarrow r'}_{x.p(x)}(M) : p(r')}$$

$$= M \quad \text{when } r = r'$$

$$\begin{array}{ccc} M & \dashrightarrow & \mathsf{coe}^{0 \rightsquigarrow 1}_{x.p(x)}(M) \\ \vdots & & \vdots \\ p(0) & \xrightarrow[p(x)]{} & p(1) \end{array}$$

Hypercubes

$\mathbb{I}$ makes it cubical because $p(x_1, \ldots, x_n)$ forms an n-dimensional hypercube.



Cubical models of type theory

	Structure on $\mathbb{I}$		. . . on coercion
BCH [Bezem, Coquand, Huber 2013]	★	*(affine)*	★★
CCHM [Cohen, C., H., Mörtberg 2016]	★ ★ ★	$\min(r, r')$, $\max(r, r')$, $(1 - r)$	★
Cartesian [A., Favonia, Harper 2017]	★★	*(structural)*	★ ★ ★

Cubical models of type theory

	Structure on $\mathbb{I}$		…on coercion
BCH [Bezem, Coquand, Huber 2013]	★	*(affine)*	★★
CCHM [Cohen, C., H., Mörtberg 2016]	★ ★ ★	$\min(r, r')$, $\max(r, r')$, $(1 - r)$	★
Cartesian [A., Favonia, Harper 2017]	★★	*(structural)*	★ ★ ★

Contribution: Model full univalent type theory using structural $\mathbb{I}$.
[Coquand 2014; Brunerie, Licata 2014; Awodey 2016; *et al.*]

Coercion — function types

How does coercion compute? Suppose $p(x) := A(x) \to B(x)$.

(Requires $1 \rightsquigarrow 0$; dependent functions require $1 \rightsquigarrow x$ and $1 \rightsquigarrow 1 = \mathrm{id}_A$.)

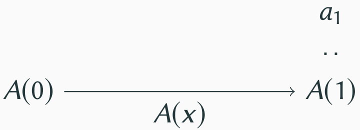
$$\frac{x : \mathbb{I} \vdash A(x) : \mathrm{Type} \qquad x : \mathbb{I} \vdash B(x) : \mathrm{Type} \qquad f : A(0) \to B(0)}{\mathrm{coe}^{0 \rightsquigarrow 1}_{x.A(x) \to B(x)}(f) : A(1) \to B(1)}$$

Coercion — function types

How does coercion compute? Suppose $p(x) := A(x) \longrightarrow B(x)$.

(Requires $1 \rightsquigarrow 0$; dependent functions require $1 \rightsquigarrow x$ and $1 \rightsquigarrow 1 = \mathrm{id}_A$.)

$$\frac{x : \mathbb{I} \vdash A(x) : \mathrm{Type} \qquad x : \mathbb{I} \vdash B(x) : \mathrm{Type} \qquad f : A(0) \longrightarrow B(0)}{\mathrm{coe}^{0 \rightsquigarrow 1}_{x.A(x) \longrightarrow B(x)}(f) : A(1) \longrightarrow B(1)}$$



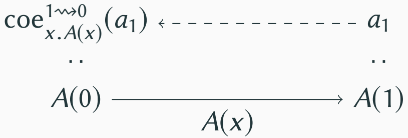
$:= \lambda a_1.$

Coercion — function types

How does coercion compute? Suppose $p(x) := A(x) \to B(x)$.

(Requires $1 \rightsquigarrow 0$; dependent functions require $1 \rightsquigarrow x$ and $1 \rightsquigarrow 1 = \mathrm{id}_A$.)

$$\frac{x : \mathbb{I} \vdash A(x) : \mathrm{Type} \qquad x : \mathbb{I} \vdash B(x) : \mathrm{Type} \qquad f : A(0) \to B(0)}{\mathrm{coe}^{0 \rightsquigarrow 1}_{x.A(x) \to B(x)}(f) : A(1) \to B(1)}$$



$:= \lambda a_1. \qquad \mathrm{coe}^{1 \rightsquigarrow 0}_{x.A(x)}(a_1)$

Coercion — function types

How does coercion compute? Suppose $p(x) := A(x) \longrightarrow B(x)$.

(Requires $1 \rightsquigarrow 0$; dependent functions require $1 \rightsquigarrow x$ and $1 \rightsquigarrow 1 = \mathrm{id}_A$.)

$$\frac{x : \mathbb{I} \vdash A(x) : \mathrm{Type} \qquad x : \mathbb{I} \vdash B(x) : \mathrm{Type} \qquad f : A(0) \longrightarrow B(0)}{\mathrm{coe}^{0 \rightsquigarrow 1}_{x.A(x) \to B(x)}(f) : A(1) \longrightarrow B(1)}$$

$$:= \lambda a_1. \qquad f\ \mathrm{coe}^{1 \rightsquigarrow 0}_{x.A(x)}(a_1)$$

$$f\ \mathrm{coe}^{1 \rightsquigarrow 0}_{x.A(x)}(a_1)$$

$$B(0) \xrightarrow[B(x)]{} B(1)$$

Coercion — function types

How does coercion compute? Suppose $p(x) := A(x) \to B(x)$.

(Requires $1 \rightsquigarrow 0$; dependent functions require $1 \rightsquigarrow x$ and $1 \rightsquigarrow 1 = \mathrm{id}_A$.)

$$\frac{x : \mathbb{I} \vdash A(x) : \mathsf{Type} \qquad x : \mathbb{I} \vdash B(x) : \mathsf{Type} \qquad f : A(0) \to B(0)}{\mathrm{coe}^{0 \rightsquigarrow 1}_{x.A(x) \to B(x)}(f) : A(1) \to B(1)}$$

$$:= \lambda a_1.\mathrm{coe}^{0 \rightsquigarrow 1}_{x.B(x)}(f\ \mathrm{coe}^{1 \rightsquigarrow 0}_{x.A(x)}(a_1))$$

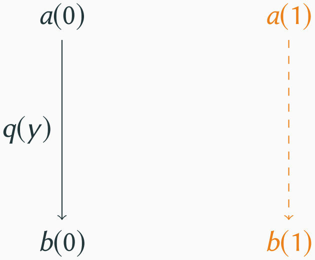
$$f\ \mathrm{coe}^{1 \rightsquigarrow 0}_{x.A(x)}(a_1) \dashrightarrow \mathrm{coe}^{0 \rightsquigarrow 1}_{x.B(x)}(f\ \mathrm{coe}^{1 \rightsquigarrow 0}_{x.A(x)}(a))$$

$$B(0) \xrightarrow[B(x)]{} B(1)$$

Coercion — path types

Suppose $p(x) := \mathrm{Path}_{A(x)}(a(x), b(x))$.

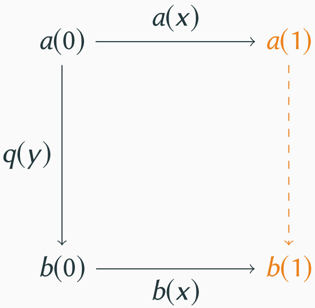
$$\frac{A : \mathrm{Type} \qquad x : \mathbb{I} \vdash a(x) : A \qquad x : \mathbb{I} \vdash b(x) : A \qquad q : \mathrm{Path}_A(a(0), b(0))}{\mathrm{coe}^{0 \rightsquigarrow 1}_{x.\mathrm{Path}_A(a(x), b(x))}(q) : \mathrm{Path}_A(a(1), b(1))}$$



Coercion — path types

Suppose $p(x) := \mathrm{Path}_{A(x)}(a(x), b(x))$.

$$\frac{A : \mathrm{Type} \qquad x : \mathbb{I} \vdash a(x) : A \qquad x : \mathbb{I} \vdash b(x) : A \qquad q : \mathrm{Path}_A(a(0), b(0))}{\mathrm{coe}^{0 \rightsquigarrow 1}_{x.\mathrm{Path}_A(a(x), b(x))}(q) : \mathrm{Path}_A(a(1), b(1))}$$

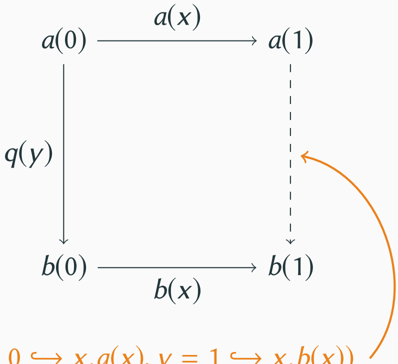


Kan composition

Composition operation extends connected partial cubes to total cubes [Kan 1955].

$$
\frac{
\begin{array}{ll}
 & \Gamma \vdash M : A \\
(\forall i) & \Gamma, \xi_i, x : \mathbb{I} \vdash N_i : A \\
(\forall i, j) & \Gamma, \xi_i, \xi_j, x : \mathbb{I} \vdash N_i = N_j : A \\
(\forall i) & \Gamma, \xi_i \vdash N_i\langle r/x \rangle = M : A
\end{array}
}{
\Gamma \vdash \mathrm{hcom}_A^{r \rightsquigarrow r'}(M; \overrightarrow{\xi_i \hookrightarrow x.N_i}) : A
}
$$

$$
= \begin{cases} M & \text{when } r = r' \\ N_i\langle r'/x \rangle & \text{when } \xi_i \end{cases}
$$



$\mathrm{hcom}_A^{0 \rightsquigarrow 1}(q(y); y = 0 \hookrightarrow x.a(x), y = 1 \hookrightarrow x.b(x))$

“Draw the rest of the owl”

Composition computes by cases on the type, mutually with coercion. Then…

“Draw the rest of the owl”

Composition computes by cases on the type, mutually with coercion. Then…

- Certain base types have free compositions and coercions [Cavallo, Harper 2019]

“Draw the rest of the owl”

Composition computes by cases on the type, mutually with coercion. Then…

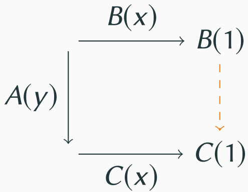
- Certain base types have free compositions and coercions [Cavallo, Harper 2019]

- Need a type former for univalence: $A \xrightarrow{V_x(f \,:\, A \xrightarrow{\sim} B)} B$

"Draw the rest of the owl"

Composition computes by cases on the type, mutually with coercion. Then...

- Certain base types have free compositions and coercions [Cavallo, Harper 2019]
- Need a type former for univalence: $A \xrightarrow{V_x(f \,:\, A \xrightarrow{\sim} B)} B$
- Need a type former for compositions of types:



“Draw the rest of the owl”

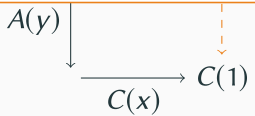
Composition computes

- Certain base types

- Need a type former

$$
\frac{
\begin{gathered}
\widetilde{N}_i[w,z] := \mathsf{coe}^{s'\langle w/x\rangle \rightsquigarrow z}_{z.B_i\langle w/x\rangle}(\mathsf{coe}^{r \rightsquigarrow w}_{x.B_i\langle s'/z\rangle}(M)) \\
\widetilde{T} := \overrightarrow{\xi_i\langle r/x\rangle \hookrightarrow z.\mathsf{coe}^{z \rightsquigarrow s\langle r/x\rangle}_{z.B_i\langle r/x\rangle}(\mathsf{coe}^{s'\langle r/x\rangle \rightsquigarrow z}_{z.B_i\langle r/x\rangle}(M))} \\
\widetilde{O}[z] := \mathsf{hcom}^{s'\langle r/x\rangle \rightsquigarrow z}_{A\langle r/x\rangle}(\mathsf{cap}^{s\langle r/x\rangle \leftsquigarrow s'\langle r/x\rangle}(M; \overrightarrow{\xi_i\langle r/x\rangle \hookrightarrow z.B_i\langle r/x\rangle}); \widetilde{T}) \\
\widetilde{P} := \mathsf{com}^{r \rightsquigarrow r'}_{x.A}(\widetilde{O}[s\langle r/x\rangle]; \overrightarrow{\xi_i \hookrightarrow x.\widetilde{N}_i[x,s]|_{(x\#\xi_i)}}, s = s' \hookrightarrow x.\mathsf{coe}^{r \rightsquigarrow x}_{x.A}(M)|_{(x\#s,s')}) \\
\widetilde{Q}_k[z] := \mathsf{com}^{s\langle r'/x\rangle \rightsquigarrow z}_{z.B_k\langle r'/x\rangle}(\widetilde{P}; \overrightarrow{\xi_i \hookrightarrow z.\widetilde{N}_i[r',z]|_{(x\#\xi_i)}}, r = r' \hookrightarrow z.\mathsf{coe}^{s'\langle r'/x\rangle \rightsquigarrow z}_{z.B_k\langle r'/x\rangle}(M)) \\
\widetilde{H} := \mathsf{hcom}^{s\langle r'/x\rangle \rightsquigarrow s'\langle r'/x\rangle}_{A\langle r'/x\rangle}(\widetilde{P}; \overrightarrow{\xi_i\langle r'/x\rangle \hookrightarrow z.\mathsf{coe}^{z \rightsquigarrow s\langle r'/x\rangle}_{z.B_i\langle r'/x\rangle}(\widetilde{Q}_i[z])}, r = r' \hookrightarrow z.\widetilde{O}[z]) \\
\widetilde{C} := \mathsf{box}^{s\langle r'/x\rangle \rightsquigarrow s'\langle r'/x\rangle}(\widetilde{H}; \overrightarrow{\xi_i\langle r'/x\rangle \hookrightarrow \widetilde{Q}_i[s'\langle r'/x\rangle]})
\end{gathered}
}{
\Psi \mid \Gamma \vdash \mathsf{coe}^{r \rightsquigarrow r'}_{x.\mathsf{hcom}^{s \rightsquigarrow s'}_{\mathsf{Type}_j}(A; \overrightarrow{\xi_i \hookrightarrow z.B_i})}(M) = \widetilde{C} : (\mathsf{hcom}^{s \rightsquigarrow s'}_{\mathsf{Type}_j}(A; \overrightarrow{\xi_i \hookrightarrow z.B_i}))\langle r'/x\rangle
}
$$

- Need a type former for compositions of types:



Computational semantics

Semantics

Dependent type theory + $\mathbb{I}$ + coercion + composition + univalence + $\cdots$ = ?

- Is this consistent?
- Does this give computational meaning to univalence?

Yes and yes — we give a computational semantics [Martin-Löf 1979; Allen 1987].

(Denotational semantics formalized in Agda [A., Brunerie, Coquand, Favonia, Harper, Licata].)

Ordinary computational semantics

$$\cdot \vdash M : A$$

- Interpret every closed proof M as a program

- Define an untyped functional programming language ($M \longmapsto M'$ and M val)
- The interpretation (extraction) can delete annotations

$$\frac{M \longmapsto M'}{M\ N \longmapsto M'\ N} \qquad \frac{}{(\lambda a.M)\ N \longmapsto M[N/a]} \qquad \cdots$$

Ordinary computational semantics

$\cdot \vdash M : A$

- Interpret every closed proof M as a program
- Interpret every closed type A as a behavioral specification

---

- Types are binary ("logical") relations on programs ($M \doteq N \in A$)
- Closed under evaluation — if $M \doteq M \in A$ then $M \Downarrow V$ and $M \doteq V \in A$
- For observable types, return an answer — $M \doteq N \in \text{bool}$ iff $M, N \Downarrow \text{true}$ or $M, N \Downarrow \text{false}$
- Consider only the closed instances of open terms ($a : A \gg M(a) \doteq N(a) \in B(a)$)

Cubical programming language

Cubical programs include coercion/composition.

To case on the path, we must evaluate terms containing interval variables!

$$\frac{M \longmapsto M'}{M\ N \longmapsto M'\ N} \qquad \frac{}{(\lambda a.M)\ N \longmapsto M[N/a]} \qquad \frac{A \longmapsto A'}{\mathrm{coe}^{r \rightsquigarrow r'}_{x.A}(M) \longmapsto \mathrm{coe}^{r \rightsquigarrow r'}_{x.A'}(M)}$$

$$\frac{}{\mathrm{coe}^{0 \rightsquigarrow 1}_{x.A(x) \to B(x)}(M) \longmapsto \lambda a_1.\mathrm{coe}^{0 \rightsquigarrow 1}_{x.B(x)}(f\ \mathrm{coe}^{1 \rightsquigarrow 0}_{x.A(x)}(a_1))}$$

$$\frac{}{\mathrm{coe}^{0 \rightsquigarrow 1}_{x.\mathrm{Path}_A(a(x), b(x))}(q) \longmapsto \mathrm{hcom}^{0 \rightsquigarrow 1}_{A}(q(y); y = 0 \hookrightarrow x.a(x), y = 1 \hookrightarrow x.b(x))} \qquad \dots$$

Cubical logical relations

Cubical behavioral specifications range over programs with interval variables.

$$M \doteq N \in A\,[x_1, \ldots, x_n]$$

Cubical logical relations

Cubical behavioral specifications range over programs with interval variables.

$$\underbrace{a_1 : A_1, \ldots, a_n : A_n}_{\text{(extensionally)}} \gg M \doteq N \in A\,[x_1, \ldots, x_n]$$

Coherence

These specifications must be closed under evaluation and interval substitutions.

$$M \doteq M \in A\,[\Psi] \implies M \Downarrow V \text{ and } M \doteq V \in A\,[\Psi]$$
$$M(x) \doteq N(x) \in A\,[\Psi, x] \implies M(0) \doteq N(0) \in A\,[\Psi]$$

Thus,

$$M(x) \doteq M(x) \in A\,[\Psi, x] \implies \begin{array}{c} M(x) \doteq V(x) \in A\,[\Psi, x] \\ M(0) \doteq V(0) \in A\,[\Psi] \\ M(0) \doteq V' \in A\,[\Psi] \end{array}$$

Coherence

These specifications must be closed under evaluation and interval substitutions.

$$M \doteq M \in A\,[\Psi] \implies M \Downarrow V \text{ and } M \doteq V \in A\,[\Psi]$$
$$M(x) \doteq N(x) \in A\,[\Psi, x] \implies M(0) \doteq N(0) \in A\,[\Psi]$$

Thus,

$$M(x) \doteq M(x) \in A\,[\Psi, x] \implies \begin{array}{c} M(x) \doteq V(x) \in A\,[\Psi, x] \\ M(0) \doteq V(0) \in A\,[\Psi] \\ M(0) \doteq V' \in A\,[\Psi] \end{array} \implies V(0) \doteq V' \in A\,[\Psi]$$

Proving coherence

Much of the difficulty involves proving coherence of evaluation and interval substitution.

- Ordinary steps commute on the nose with interval substitution. (Cubically stable.)
- To show these commute for M, construct a substitution-indexed family of its reducts, and prove they are pairwise extensionally equal. (Coherent expansion.)

Canonicity

Booleans (and HITs!) are indeed observable.

Theorem: Every $M \in \text{bool}\ [\cdot]$ computes to, and is equal to, true or false. ($\approx$ [Huber 2016])

Theorem: Every $M \in \mathbb{S}^1\ [\cdot]$ computes to, and is equal to, base.

Contribution: Refinement of composition operation ("validity" restriction) makes higher inductive types also observable. [Vezzosi, Mörtberg, Abel 2019]

Exact equality

These specifications naturally account for extensional/exact equality $M \doteq N \in A\ [\cdot]$.
(Unlike paths, these equations do not appear in extracts.)

Can we have an exact equality type? No, it doesn't support coercion!

$$\star \in \mathsf{Eq}_{\mathsf{Type}}(A, A)\ [\cdot]$$

$$\rightsquigarrow \ \mathsf{univalence}(f : A \xrightarrow{\sim} B)$$

$$\mathsf{Eq}_{\mathsf{Type}}(A, B)\ [\cdot] \ ✗$$

Two-level type theory

We present a two-level type theory [Voevodsky 2013; Altenkirch, Capriotti, Kraus 2016]:

- Pretypes — respect only exact equality
- Kan types — respect paths and exact equality

Exact equality is seemingly needed for some mathematical constructions.

Implementing exact equality

Practical question — what principles about exact equality do we expose?

“Equality reflection” precludes type-checking proofs (as in Coq, Agda, **redtt**, …).

Nuprl/**RedPRL** include reflection; instead of type-checking, one directly manipulates (cubical) extracts and behavioral specifications as a program logic.

Contribution: **RedPRL** is the first (only) implementation of a two-level type theory with canonicity.

Taking stock

Contribution

We present Cartesian cubical type theory, a univalent dependent type theory whose proofs have computational meaning.

Second such, after the cubical type theory of [Cohen, Coquand, Huber, Mörtberg 2016].

Try these out in **RedPRL**, **redtt**, and Cubical Agda!

What's all this good for? I

First answer — the homotopy type theory project has led mathematicians to use type theory as a formal language for homotopy theory.

In our experience, type theories with good computational properties are easier to use.

Serves as a language for topological spaces — equivariant Cartesian cubical model [Awodey, Cavallo, Coquand, Riehl, Sattler].

What's all this good for? II

Second answer — cubes solve some longstanding difficulties with equality connectives.

- Function extensionality, unlike identity types
- Good account of dependent equality ("pathovers")
- Well-behaved quotients (better with observable HITs!)
- Univalence permits invariant view of mathematics [Awodey 2014]

We've backported cubes to non-univalent type theory — XTT [Sterling, A., Gratzer 2019].

Thanks!

(`cs.cmu.edu/~cangiuli`)

- Chapters 1 and 5 — Big picture
- Chapter 2 — Ordinary computational semantics
- Chapter 3 — Cubical type theories
- Chapter 4 — All main theorems
- Appendix A — Program logic ≈ **RedPRL**
- Appendix B — **`redtt`** core calculus

Coercion — dependent function types

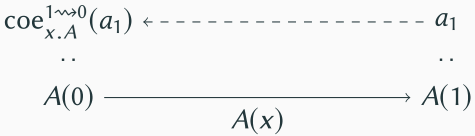
Suppose $p(x) := (a : A(x)) \to B(x, a)$. (Uses $1 \rightsquigarrow x$ and $1 \rightsquigarrow 1$.)

$$\frac{x : \mathbb{I} \vdash A(x) : \mathrm{Type} \qquad x : \mathbb{I}, a : A \vdash B(x, a) : \mathrm{Type} \qquad f : (a_0 : A(0)) \to B(0, a_0)}{\mathrm{coe}^{0 \rightsquigarrow 1}_{x.A(x) \to B(x,a)}(f) : (a_1 : A(1)) \to B(1, a_1)}$$

Coercion — dependent function types

Suppose $p(x) := (a : A(x)) \to B(x, a)$. (Uses $1 \rightsquigarrow x$ and $1 \rightsquigarrow 1$.)

$$\frac{x : \mathbb{I} \vdash A(x) : \mathrm{Type} \qquad x : \mathbb{I}, a : A \vdash B(x, a) : \mathrm{Type} \qquad f : (a_0 : A(0)) \to B(0, a_0)}{\mathrm{coe}^{0 \rightsquigarrow 1}_{x.A(x) \to B(x,a)}(f) : (a_1 : A(1)) \to B(1, a_1)}$$



$$:= \lambda a_1. \qquad \mathrm{coe}^{1 \rightsquigarrow 0}_{x.A(x)}(a_1)$$

Coercion — dependent function types

Suppose $p(x) := (a : A(x)) \to B(x, a)$. (Uses $1 \rightsquigarrow x$ and $1 \rightsquigarrow 1$.)

$$\frac{x : \mathbb{I} \vdash A(x) : \mathrm{Type} \qquad x : \mathbb{I}, a : A \vdash B(x,a) : \mathrm{Type} \qquad f : (a_0 : A(0)) \to B(0, a_0)}{\mathrm{coe}^{0 \rightsquigarrow 1}_{x.A(x) \to B(x,a)}(f) : (a_1 : A(1)) \to B(1, a_1)}$$

$$:= \lambda a_1. \qquad f\ \mathrm{coe}^{1 \rightsquigarrow 0}_{x.A(x)}(a_1)$$

$$f\ \mathrm{coe}^{1 \rightsquigarrow 0}_{x.A(x)}(a_1) \dashrightarrow$$

$$B(0, \mathrm{coe}^{1 \rightsquigarrow 0}_{x.A(x)}(a_1)) \qquad B(1, a_1)$$

Coercion — dependent function types

Suppose $p(x) := (a : A(x)) \to B(x, a)$. (Uses $1 \rightsquigarrow x$ and $1 \rightsquigarrow 1$.)

$$\frac{x : \mathbb{I} \vdash A(x) : \mathsf{Type} \qquad x : \mathbb{I}, a : A \vdash B(x, a) : \mathsf{Type} \qquad f : (a_0 : A(0)) \to B(0, a_0)}{\mathsf{coe}^{0 \rightsquigarrow 1}_{x.A(x) \to B(x,a)}(f) : (a_1 : A(1)) \to B(1, a_1)}$$

$$:= \lambda a_1.\mathsf{coe}^{0 \rightsquigarrow 1}_{x.B(x, \mathsf{coe}^{1 \rightsquigarrow x}_{x.A}(a_1))}(f\ \mathsf{coe}^{1 \rightsquigarrow 0}_{x.A(x)}(a_1))$$

$$\begin{array}{ccc} f\ \mathsf{coe}^{1 \rightsquigarrow 0}_{x.A(x)}(a_1) & \dashrightarrow & \\ \vdots & & \vdots \\ B(0, \mathsf{coe}^{1 \rightsquigarrow 0}_{x.A(x)}(a_1)) & \xrightarrow[B(x, \mathsf{coe}^{1 \rightsquigarrow x}_{x.A}(a_1))]{} & B(1, a_1) \\ & & \parallel \\ & & \mathsf{coe}^{1 \rightsquigarrow 1}_{x.A}(a_1) \end{array}$$

Graphics test

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt [ut labore et dolore magna aliqua]. Utenimadminimveniam, **RedPRL**

quis nostrud: exercitation ullamco laboris nisi ut aliquip ex ea commodo [consequat].

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum *dolore* ★★