Cubical Type Theory: a constructive interpretation of the univalence axiom

Anders Mörtberg

(jww C. Cohen, T. Coquand, and S. Huber)

Institute for Advanced Study, Princeton

February 23, 2016

Introduction

Goal: provide a computational justification for notions from Homotopy Type Theory and Univalent Foundations, in particular the univalence axiom and higher inductive types

Specifically, design a type theory with good properties (normalization, decidability of type checking, etc.) where the univalence axiom computes and which has support for higher inductive types

Homotopy Type Theory

Univalent Foundations of Mathematics

The Univalent Foundations Program
Institute for Advanced Study

Homotopy Type Theory

Builds on connection between type theory and homotopy theory

Adds extensionality concepts to (intensional) type theory:

- Function extensionality: two functions are equal if they produce the same output for all inputs
- Quotient types
- Can transport code and proofs between structures (univalence)
- Higher inductive types

Type theory

Type theory uses the syntax of λ-calculus, hence it can be seen as a functional programming language

There are many proof assistants based on variations of type theory: Coq, Agda, Nuprl, Idris, Lean...

Programs and proofs are written in the same languange

HoTT is typically defined as an extension of intensional type theory (e.g. Martin-Löf type theory or Calculus of constructions)

BHK interpretation

Proofs are first-class citizens (just like functions in FP): an even natural number is a pair of the number and a proof that it is even

BHK interpretation

Proofs are first-class citizens (just like functions in FP): an even natural number is a pair of the number and a proof that it is even

This gives an interpretation of the logical connectives. For instance an implication:

$$P \to Q$$

is a **function** that maps a proof of P to a proof of Q

BHK interpretation

Proofs are first-class citizens (just like functions in FP): an even natural number is a pair of the number and a proof that it is even

This gives an interpretation of the logical connectives. For instance an implication:

$$P \to Q$$

is a **function** that maps a proof of P to a proof of Q

Propositions form a "sub-universe" of the types and logical connectives are encoded by the general operations on types

Equality/Identity types

```
Inductive eq (A : Type) (a : A) : A -> Type :=
  refl : eq A a a

Notation (a = b) := (eq A a b).
Notation 1_a := (refl a).

Lemma eq_sym (A : Type) (a b : A) : a = b -> b = a.

Lemma eq_trans (A : Type) (a b c : A) : a = b -> b = c -> a = c.

Lemma eq_trans_refl_l (A : Type) (a b : A) (p : a = b), eq_trans 1_a p = p.

Lemma eq_trans_refl_r (A : Type) (a b : A) (p : a = b), eq_trans p 1_b = p.

...
```

Equality: transport

```
Definition transport (A : Type) (P : A -> Type)
  (a b : A) (p : a = b) : P a -> P b := ...
```

"'Leibniz identity of indiscernibles": there cannot be separate objects or entities that have all their properties in common, that is, entities a and b are identical if every predicate possessed by a is also possessed by b and vice versa

Equality: higher dimensional structure

As the first parameter to eq is a type one can plug in another eq:

$$\begin{aligned} a\ b &: A \\ p\ q &: a = b \\ \alpha &: p = q \\ &\vdots \end{aligned}$$

Equality is proof relevant, and has the structure of an ∞-groupoid...

Homotopy type theory

"In algebraic topology, homotopy theory is the study of homotopy groups; and more generally of the category of topological spaces and homotopy classes of continuous mappings."

Homotopy type theory

"In algebraic topology, homotopy theory is the study of homotopy groups; and more generally of the category of topological spaces and homotopy classes of continuous mappings."

Type theory	Homotopy theory
A Type	A Space
$a : A$	a is a point in A
$p : a = b$	p is a path between a and b in A
$\alpha : p = q$	α is a homotopy between p and q
$\vdots$	$\vdots$

Univalence axiom

Equivalence of types, Equiv A B, is a generalization of bijection of sets

Univalence axiom (Voevodsky): equality of types is equivalent to equivalence of types:

$$\text{Equiv}\ (A = B)\ (\text{Equiv}\ A\ B)$$

I particular we get a map:

$$\text{Equiv}\ A\ B \to A = B$$

Univalence axiom: consequences

Can prove extensionality for functions:

```
Lemma funext (A B : Type) (f g : A -> B)
             (H : forall a, f a = g a), f = g.
```

Using this one can prove that for example insertion sort and quicksort are equal as functions and rewrite with this equality

Univalence axiom: consequences

Get transport for equivalences:

```
Definition transport_equiv (P : Type -> Type) (A B : Type)
                           (p : Equiv A B) : P A -> P B :=
```

Get a new version of Leibniz's principle of the identity of indiscernibles: reasoning is invariant under equivalence

Can be used for generic programming, for example to transform fields in a database using an isomorphism

Univalence axiom: consequences

Structure identity principle: univalence lifts to structures (Coquand-Danielsson, Ahrens-Kapulkin-Shulman)

```
Definition transport_monoid (P : Monoid -> Type)
  (A B : Monoid) (p : EquivMonoid A B) : P A -> P B := ...
```

Can be used for program and data refinements: can prove properties on the monoid of unary natural numbers by computing with the monoid of binary natural numbers

Univalence axiom: problems

The univalence axiom can be added to Coq as an Axiom:

```
Definition eqweqmap (A B : Type) (p : A = B) : Equiv A B :=

Axiom univalence (A B : Type), is_equiv (eqweqmap A B)).
```

By doing this Coq looses its good computational properties, in particular one can construct terms that are **stuck**

Solution: define a new type theory in which the univalence axiom computes, in other words where there is a term that realizes it

Cubical Type Theory

An extension of dependent type theory which allows the user to directly argue about n-dimensional cubes (points, lines, squares, cubes etc.) representing equality proofs

Based on a model in cubical sets formulated in a constructive metatheory

The univalence axiom is provable in the system

We have a prototype implementation in HASKELL

Cubical Type Theory

Extends dependent type theory with:

1. Path types
2. Kan composition operations
3. Glue types
4. Higher inductive types

Base type theory

$$
\begin{array}{lcl}
\Gamma, \Delta & ::= & () \mid \Gamma, x : A \\
t, u, A, B & ::= & x \mid \lambda x : A.\ t \mid t\ u \mid (x : A) \to B \\
& \mid & (t, u) \mid t.1 \mid t.2 \mid (x : A) \times B \\
& \mid & 0 \mid \mathsf{s}\ u \mid \mathsf{natrec}\ t\ u \mid \mathsf{N}
\end{array}
$$

with η for functions and pairs

Path types: the interval

Path types provides a convenient syntax for reasoning about (higher) equality proofs

Formal representation of the interval, $\mathbb{I}$:

$$r, s \quad ::= \quad 0 \mid 1 \mid i \mid 1 - r \mid r \wedge s \mid r \vee s$$

The context can contain variables in the interval:

$$\Gamma, \Delta \quad ::= \quad \ldots \mid \Gamma, i : \mathbb{I}$$

$$\frac{\Gamma \vdash}{\Gamma, i : \mathbb{I} \vdash} \; (i \notin \mathrm{dom}(\Gamma))$$

Path types

The intuition is that a type in a context with n names corresponds to an n-dimensional cube:

$() \vdash A$	$\bullet\ A$
$i : \mathbb{I} \vdash A$	$A(i0) \xrightarrow{A} A(i1)$
$i : \mathbb{I}, j : \mathbb{I} \vdash A$	$\begin{array}{ccc} A(i0)(j1) & \xrightarrow{A(j1)} & A(i1)(j1) \\ {\scriptstyle A(i0)}\uparrow & A & \uparrow{\scriptstyle A(i1)} \\ A(i0)(j0) & \xrightarrow[A(j0)]{} & A(i1)(j0) \end{array}$
$\vdots$	$\vdots$

Path types: syntax

$$\begin{array}{rcl} t, u, A, B & ::= & \ldots \\ & | & \mathsf{Path}\ A\ t\ u \mid \langle i \rangle\ t \mid t\ r \end{array}$$

Path abstraction, $\langle i \rangle\ t$, binds the name i in t

Path application, $t\ r$, applies a term t to an element $r : \mathbb{I}$

This is similar to the notion of name-abstraction in nominal sets

Path types

We define 1_a : Path A a a as $\langle i \rangle$ a, which corresponds to a proof of reflexivity:

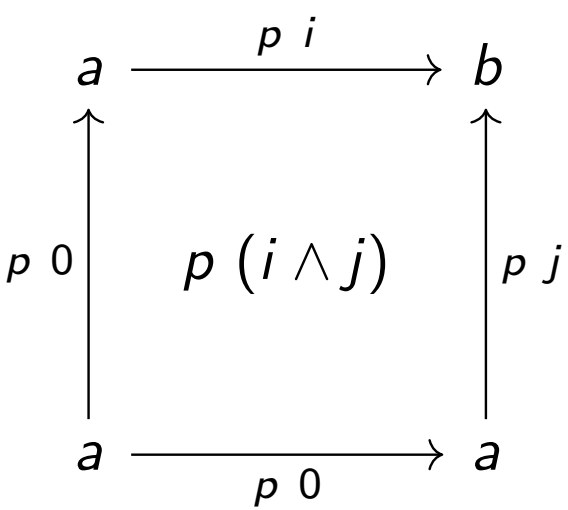
$$a \xrightarrow{1_a} a$$

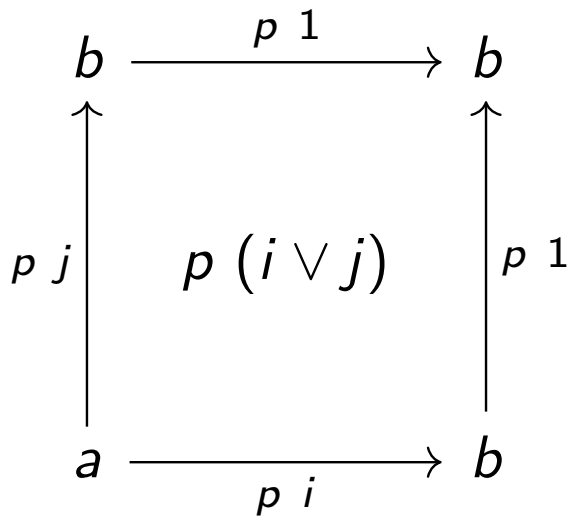
Given p : Path A a b we have p $0 = a$, p $1 = b$ and

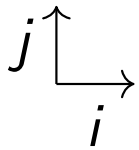
$$b \xrightarrow{p\ (1-i)} a$$

Path types: connections

Given p : Path A a b we can build







Path types: rules

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash t : A \qquad \Gamma \vdash u : A}{\Gamma \vdash \mathsf{Path}\ A\ t\ u} \qquad \frac{\Gamma \vdash A \qquad \Gamma, i : \mathbb{I} \vdash t : A}{\Gamma \vdash \langle i \rangle\ t : \mathsf{Path}\ A\ t(i0)\ t(i1)}$$

$$\frac{\Gamma \vdash t : \mathsf{Path}\ A\ u_0\ u_1 \qquad \Gamma \vdash r : \mathbb{I}}{\Gamma \vdash t\ r : A} \qquad \frac{\Gamma \vdash A \qquad \Gamma, i : \mathbb{I} \vdash t : A}{\Gamma \vdash (\langle i \rangle\ t)\ r = t(i/r) : A}$$

$$\frac{\Gamma \vdash t : \mathsf{Path}\ A\ u_0\ u_1}{\Gamma \vdash t\ 0 = u_0 : A} \qquad \frac{\Gamma \vdash t : \mathsf{Path}\ A\ u_0\ u_1}{\Gamma \vdash t\ 1 = u_1 : A}$$

$$\frac{\Gamma, i : \mathbb{I} \vdash t\ i = u\ i : A}{\Gamma \vdash t = u : \mathsf{Path}\ A\ u_0\ u_1}$$

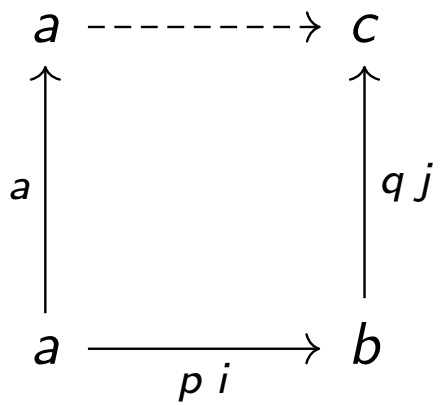
Path types: function extensionality

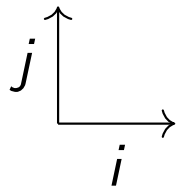
Function extensionality for path types can be proved as:

$$\frac{\Gamma \vdash f, g : (x : A) \to B \qquad \Gamma \vdash p : (x : A) \to \mathsf{Path}\ B\ (f\ x)\ (g\ x)}{\Gamma \vdash \langle i \rangle\ \lambda x : A.\ p\ x\ i : \mathsf{Path}\ ((x : A) \to B)\ f\ g}$$

Kan composition operations

We want to be able to compose paths:





In general this corresponds to computing the missing sides of n-dimensional cubes (Kan composition)

The face lattice

In order to do this we need syntax for representing partially specified n-dimensional cubes, hence we introduce the face lattice $\mathbb{F}$:

$$\varphi, \psi \quad ::= \quad 0_{\mathbb{F}} \mid 1_{\mathbb{F}} \mid (i = 0) \mid (i = 1) \mid \varphi \wedge \psi \mid \varphi \vee \psi$$

We add context restrictions:

$$\Gamma, \Delta \quad ::= \quad \ldots \mid \Gamma, \varphi$$

$$\frac{\Gamma \vdash \varphi : \mathbb{F}}{\Gamma, \varphi \vdash}$$

Partial elements

$i : \mathbb{I}, (i = 0) \vee (i = 1) \vdash A$	$A(i0)\ \bullet \qquad\qquad \bullet\ A(i1)$
$i : \mathbb{I}, j : \mathbb{I}, (i = 0) \vee (j = 1) \vdash A$	$A(i0)(j1) \xrightarrow{A(j1)} A(i1)(j1)$ $A(i0) \uparrow$ $A(i0)(j0)$
$i : \mathbb{I}, j : \mathbb{I}, (i = 0) \vee (i = 1) \vee (j = 0) \vdash A$	$A(i0)(j1) \qquad\qquad A(i1)(j1)$ $A(i0) \uparrow \qquad\qquad \uparrow A(i1)$ $A(i0)(j0) \xrightarrow[A(j0)]{} A(i1)(j0)$

Systems

We add syntax for specifying shapes/systems:

$$\begin{array}{lcl} t, u, A, B & ::= & \ldots \\ & | & S \\ S & ::= & [\ \varphi_1\ t_1, \ldots, \varphi_n\ t_n\ ] \end{array}$$

Together with rules for specififying when these shapes are well formed:

$$\frac{\Gamma, \varphi_1 \vdash A_1 \qquad \cdots \qquad \Gamma, \varphi_n \vdash A_n \qquad \Gamma, \varphi_i \wedge \varphi_j \vdash A_i = A_j}{\Gamma \vdash [\ \varphi_1\ A_1, \ldots, \varphi_n\ A_n\ ]}$$

Kan compositions

The syntax of compositions is given by:

$$\begin{array}{rcl} t, u, A, B & ::= & \ldots \\ & | & \mathsf{comp}^i\ A\ [\varphi \mapsto u]\ a_0 \end{array}$$

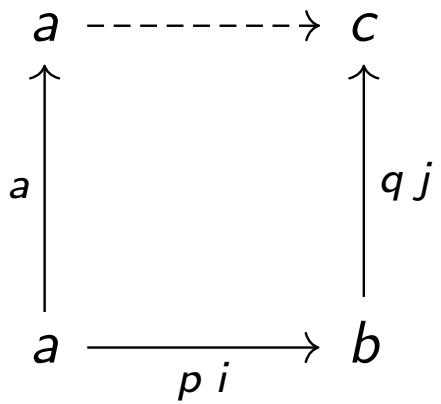
where u is a system defined on φ

$$\frac{\Gamma \vdash \varphi \qquad \Gamma, i : \mathbb{I} \vdash A \qquad \Gamma, \varphi, i : \mathbb{I} \vdash u : A \qquad \Gamma \vdash a_0 : A(i0)}{\Gamma \vdash \mathsf{comp}^i\ A\ [\varphi \mapsto u]\ a_0 : A(i1)}$$

Kan composition: example

With composition we can justify transitivity of path types:

$$\frac{\Gamma \vdash p : \mathsf{Path}\ A\ a\ b \qquad \Gamma \vdash q : \mathsf{Path}\ A\ b\ c}{\Gamma \vdash \langle i \rangle\ \mathsf{comp}^j\ A\ [(i = 0) \mapsto a, (i = 1) \mapsto q\ j]\ (p\ i) : \mathsf{Path}\ A\ a\ c}$$



Kan composition: transport and equality judgments

Composition for $\varphi = 0_{\mathbb{F}}$ corresponds to transport:

$$\Gamma \vdash \mathsf{transport}^i\ A\ a = \mathsf{comp}^i\ A\ []\ a : A(i1)$$

Equality judgments for *comp* is defined by case on A, this is the main part of the system

Glue

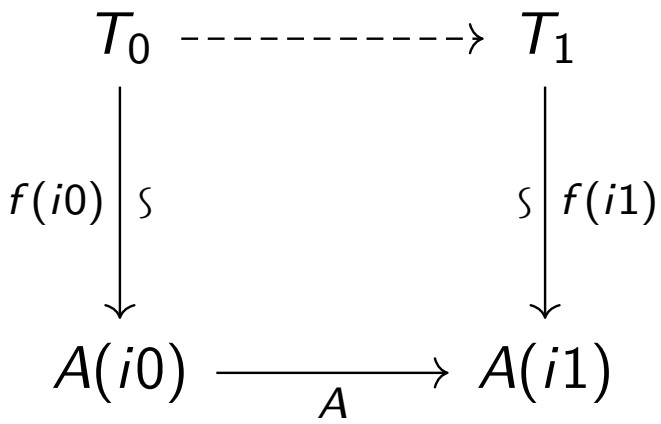
In order to be able to prove univalence we add a **glueing** operation:

$$
\begin{array}{rcl}
t, u, A, B & ::= & \ldots \\
& | & \mathsf{Glue}\ [\varphi \mapsto (T, f)]\ A \\
& | & \mathsf{glue}\ [\varphi \mapsto t]\ u \\
& | & \mathsf{unglue}\ [\varphi \mapsto (T, f)]\ u
\end{array}
$$

$$
\frac{\Gamma \vdash A \qquad \Gamma, \varphi \vdash T \qquad \Gamma, \varphi \vdash f : \mathsf{Equiv}\ T\ A}{\Gamma \vdash \mathsf{Glue}\ [\varphi \mapsto (T, f)]\ A}
$$

Glue: example

In the case $\varphi = (i = 0) \vee (i = 1)$ the glueing operation can be illustrated as the dashed line in:



Composition for glueing is the most complicated part of the system

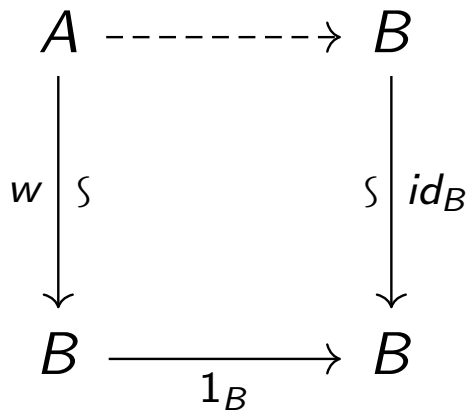
Universe and univalence

We can also add a universe U and define composition for it using composition for Glue

For univalence we need to define a map:

$$(w : \text{Equiv } A\ B) \to \text{Path U } A\ B$$

Using glue we get:



Higher inductive types

Higher inductive types generalize inductive types by allowing constructors for not only points, but also for (higher) equalities

Examples:

- Quotient types
- Propositional truncation (squash types)
- Topological spaces (circle, sphere, torus...)

Integers as a higher inductive types

```
data int = pos (n : nat)
         | neg (n : nat)
         | zeroP <i> [ (i = 0) -> pos zero
                     , (i = 1) -> neg zero ]

sucInt : int -> int = split
  pos n -> pos (suc n)
  neg n -> sucNat n
    where sucNat : nat -> int = split
            zero -> pos one
            suc n -> neg n
  zeroP @ i -> pos one
```

Summary and future work

We have an implementation in Haskell, try it:

`https://github.com/mortberg/cubicaltt/`

Future work:

- metatheory (normalization, decidability of type checking...)
- formalize correctness (wip of Mark Bickford in Nuprl)
- general formulation and semantics of higher inductive types

Thank you for your attention!