Cubical Type Theory: a constructive interpretation of the univalence axiom

Anders Mörtberg – Inria Sophia-Antipolis

Cubical Type Theory

Goal: provide a computational justification for **Homotopy Type Theory** and **Univalent Foundations**

We have designed a type theory where **univalence** computes and with support for **higher inductive types**

From the point of view of type theory this work is mainly about **equality**

Equality/Identity types in type theory

```
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 is proof relevant

Equality: transport

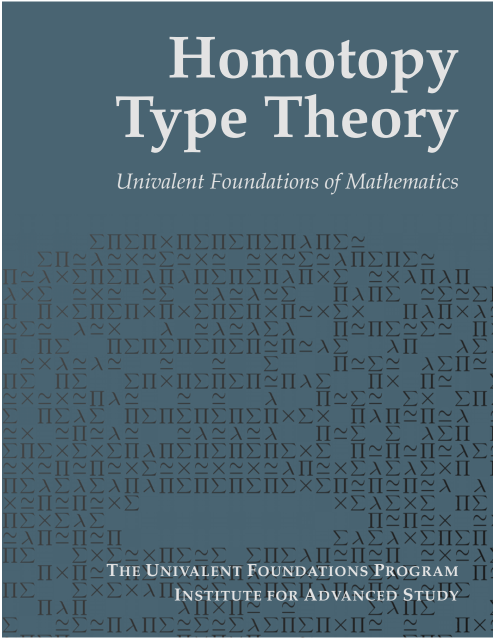
```
Definition transport (A : Type) (P : A -> Type)
  (a b : A) (p : a = b) : P a -> P b := ...
```

"Leibniz Indiscernibility of Identicals": identical objects satisfy the same properties

Problems with equality in type theory

- Not possible to prove that pointwise equal functions are equal (function extensionality)
- Not easy to define quotients (“setoid nightmare”)
- What is the equality between types, i.e. what is the equality for Type?

Solution: Homotopy Type Theory and Univalent Foundations

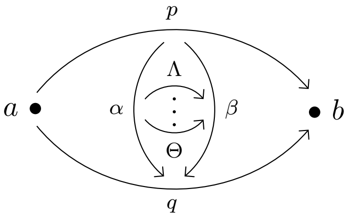

Homotopy Type Theory
Univalent Foundations of Mathematics
The Univalent Foundations Program
Institute for Advanced Study

Homotopy type theory

"**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

$a, b : \mathsf{A}$
$p, q : a = b$
$\alpha, \beta : p = q$
$\Lambda, \Theta : \alpha = \beta$
$\vdots$



Voevodsky's univalence axiom

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

Univalence axiom: equality of types is equivalent to equivalence of types

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

I particular we get a map:

$$\text{univalence\_inv} : \text{Equiv}\ A\ B \to A = B$$

Univalence axiom: consequences

Can prove function extensionality:

```
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 := ...
```

This can be seen as a new version of Leibniz's principle: reasoning is invariant under equivalence

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 type theory as an axiom:

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

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

This is consistent by Voevodsky's simplicial set model

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

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

Each type has a *"cubical"* structure

Cubical Type Theory

Extends dependent type theory (with η for functions and pairs) with:

1. **Path types**
2. **Composition operations**
3. **Glue types (univalence)**
4. Identity types
5. Higher inductive types

Path types

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

Contexts can contain variables in the interval:

$$\frac{\Gamma \vdash}{\Gamma, i : \mathbb{I} \vdash}$$

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

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

i, j, k... formal symbols/names representing directions/dimensions

Path types

$i : \mathbb{I} \vdash A$ corresponds to a line:

$$A(i/0) \xrightarrow[i]{\quad A \quad} A(i/1)$$

$i : \mathbb{I}, j : \mathbb{I} \vdash A$ corresponds to a square:

$$\begin{tikzcd}
A(i/0)(j/1) \arrow[r, "A(j/1)"] & A(i/1)(j/1) \\
A(i/0)(j/0) \arrow[u, "A(i/0)"] \arrow[r, "A(j/0)"'] \arrow[ur, phantom, "A"] & A(i/1)(j/0) \arrow[u, "A(i/1)"']
\end{tikzcd}
\qquad
\begin{tikzcd}
\cdot \\
\cdot \arrow[u, "j"] \arrow[r, "i"'] & \cdot
\end{tikzcd}$$

and so on...

Path types: rules

$$\frac{\Gamma \vdash A \qquad \Gamma, i : \mathbb{I} \vdash t : A}{\Gamma \vdash \langle i \rangle\ t : \mathsf{Path}\ A\ t(i/0)\ t(i/1)} \qquad \frac{\Gamma \vdash t : \mathsf{Path}\ A\ u_0\ u_1 \qquad \Gamma \vdash r : \mathbb{I}}{\Gamma \vdash t\ r : A}$$

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

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

Path types

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

$$t(i/0) \xrightarrow{\langle i \rangle\ t} t(i/1)$$

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

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

Path types are great! (function extensionality)

Given (dependent) functions $f, g : (x : A) \to B$ and that are pointwise equal:

$$p : (x : A) \to \mathsf{Path}\ B\ (f\ x)\ (g\ x)$$

we can prove that the functions are equal by:

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

Path types are great! (maponpaths)

Given $f : A \to B$ and $p :$ Path $A\ a\ b$ we can define:

$$\texttt{ap}\ f\ p = \langle i \rangle\ f\ (p\ i) : \text{Path}\ B\ (f\ a)\ (f\ b)$$

satisfying definitionally:

$$\begin{array}{llll} \texttt{ap} & \mathrm{id} & p & = & p \\ \texttt{ap} & (f \circ g) & p & = & \texttt{ap}\ f\ (\texttt{ap}\ g\ p) \end{array}$$

This way we get new ways for reasoning about equality: inline ap, funext, symmetry... with new definitional equalities

Composition operations

We want to be able to compose paths:

$$a \xrightarrow{p} b \qquad\qquad b \xrightarrow{q} c$$

We do this by computing the dashed line in:

$$\begin{tikzcd}
a \arrow[r, dashed] & c \\
a \arrow[u, "a"] \arrow[r, "p"'] & b \arrow[u, "q"']
\end{tikzcd}$$

In general this corresponds to computing the missing sides of n-dimensional cubes

Composition operations

Box principle: any open box has a lid

Cubical version of the Kan condition for simplicial sets:

"Any horn can be filled"

First formulated by Daniel Kan in *"Abstract Homotopy I"* (1955) for cubical complexes

Context restrictions

To formulate this we need syntax for representing partially specified n-dimensional cubes

We add context **restrictions** Γ, φ where φ is a "face formula" representing a subset of the faces of a cube

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

Partial types

If $\Gamma, \varphi \vdash A$ then A is a **partial type** of extent φ

A partial type $i : \mathbb{I}, (i = 0) \vee (i = 1) \vdash A$ corresponds to:

$$A(i/0) \bullet \qquad\qquad \bullet A(i/1)$$

A partial type $i\ j : \mathbb{I}, (i = 0) \vee (i = 1) \vee (j = 0) \vdash A$ corresponds to:

$$\begin{tikzcd}
\bullet & \bullet \\
\bullet \arrow[u, "A(i/0)"] \arrow[r, "A(j/0)"'] & \bullet \arrow[u, "A(i/1)"']
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
{} \\
\cdot \arrow[u, "j"] \arrow[r, "i"'] & {}
\end{tikzcd}$$

Partial elements

Any judgment valid in a context Γ is also valid in a restriction Γ, φ

$$\frac{\Gamma \vdash A}{\Gamma, \varphi \vdash A}$$

If $\Gamma \vdash A$ and $\Gamma, \varphi \vdash a : A$ then a is a **partial element** of A of extent φ.
We write $\Gamma \vdash b : A[\varphi \mapsto a]$ for:

$$\Gamma \vdash b : A \qquad \Gamma, \varphi \vdash a : A \qquad \Gamma, \varphi \vdash a = b : A$$

Box principle

We can now formulate the box principle in type theory:

$$\frac{\Gamma, i : \mathbb{I} \vdash A \qquad \Gamma \vdash a_0 : A(i/0)[\varphi \mapsto u(i/0)] \qquad \Gamma, \varphi, i : \mathbb{I} \vdash u : A}{\Gamma \vdash \mathsf{comp}^i\ A\ [\varphi \mapsto u]\ a_0 : A(i/1)[\varphi \mapsto u(i/1)]}$$

- a_0 is the bottom
- u is the sides
- $\mathsf{comp}^i\ A\ [\varphi \mapsto u]\ a_0$ is the lid

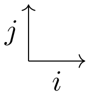
Equality judgments for $\mathsf{comp}^i\ A\ [\varphi \mapsto u]\ a_0$ are defined by cases on A

Composition operations: example

With composition we can justify transitivity of path types:

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

$$\begin{tikzcd} a \arrow[r, dashed] & c \\ a \arrow[u, "a"] \arrow[r, "p\ i"'] & b \arrow[u, "q\ j"'] \end{tikzcd}$$



Cast as a composition

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

$$\frac{\Gamma, i : \mathbb{I} \vdash A \qquad \Gamma \vdash a : A(i/0)}{\Gamma \vdash \mathsf{cast}^i\ A\ a = \mathsf{comp}^i\ A\ []\ a : A(i/1)}$$

$$a \bullet \qquad\qquad\qquad \bullet\ \mathsf{cast}^i\ A\ a$$

$$A(i/0) \xrightarrow[i]{A} A(i/1)$$

Using this we can define `transport`, path induction...

Glue types

We extend the system with **Glue types**, these allow us to:

- Define composition for the universe
- Prove univalence

Composition for these types is the most complicated part of the system

Example: unary and binary numbers

Let `nat` be unary natural numbers and `binnat` be binary natural numbers. We have an equivalence

$$e : \texttt{nat} \to \texttt{binnat}$$

and we want to construct a path P with $P(i/0) = \texttt{nat}$ and $P(i/1) = \texttt{binnat}$:

$$\texttt{nat} \overset{P}{\dashrightarrow} \texttt{binnat}$$

Example: unary and binary numbers

P should also store information about e, we achieve this by “glueing”:

$$
\begin{tikzcd}
\texttt{nat} \arrow[r, dashed, "P"] \arrow[d, "e"', "\wr"] & \texttt{binnat} \arrow[d, "\wr"', "\texttt{id}"] \\
\texttt{binnat} \arrow[r, "\texttt{binnat}"'] & \texttt{binnat}
\end{tikzcd}
$$

We write

$$P = \langle i \rangle \ \mathsf{Glue}\ \texttt{binnat}\ [(i = 0) \mapsto (\texttt{nat}, e), (i = 1) \mapsto (\texttt{binnat}, \texttt{id})]$$

Univalence?

What do we need to prove univalence?

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

By an observation of Dan Licata it suffices to define a function:

$$\text{ua} : \text{Equiv}\ A\ B \to \text{Path U}\ A\ B$$

such that for any e : Equiv A B and $a : A$:

$$\text{Path}\ B\ (\text{cast}\ \ (\text{ua}\ e)\ a)\ (e.1\ a)$$

Univalence

Given e : Equiv A B we can define the term

$$\mathsf{ua} : \mathsf{Path}\ \mathsf{U}\ A\ B = \langle i \rangle\ \mathsf{Glue}\ B\ [(i=0) \mapsto (A, e), (i=1) \mapsto (B, \mathtt{id}_B)]$$

which satisfies the necessary computation rule

Univalence is hence provable in the system, but it is often more convenient to work with the Glue types directly

cubicaltt

We have a prototype implementation written in HASKELL:

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

The implementation contains an evaluator, typechecker, parser, etc, but it has no "fancy" features of modern proof assistants (unification, implicit arguments, type classes...)

Computing with univalence: bool = bool

```
data bool = false | true

negBool : bool → bool = split
  false → true
  true → false

negBoolK : (b : bool) → Path bool (negBool (negBool b)) b = split
  false → <i> false
  true → <i> true

negBoolEquiv : equiv bool bool =
  (negBool,gradLemma bool bool negBool negBool negBoolK negBoolK)

negBoolEq : Path U bool bool =
  <i> Glue bool [ (i = 0) ↦ (bool,negBoolEquiv)
                , (i = 1) ↦ (bool,idEquiv bool) ]

> cast negBoolEq true
EVAL: false
```

Computing with univalence

We have implemented many more examples:

- Unary and binary numbers
- Fundamental group of the circle (compute winding numbers)
- Voevodsky's impredicative set quotients
- Dan Grayson's definition of the circle using Z-torsors and a proof that it is equivalent to the HIT circle (Rafaël Bocquet)
- Structure identity principle for categories (Rafaël Bocquet)
- Universe categories and C-systems, proof that two equivalent universe categories give two equal C-systems (Rafaël Bocquet)
- $\mathbb{Z}$ as a HIT
- $\mathbb{T} \simeq \mathbb{S}^1 \times \mathbb{S}^1$ (Dan Licata, 60 LOC)
- ...

Normal form of univalence

```
module nthmUniv where

import univalence

nthmUniv : (t : (A X : U) → Id U X A → equiv X A) (A : U)
  (X : U) → isEquiv (Id U X A) (equiv X A) (t A X) = \(t : (A X : U)
  → (IdP (<!0> U) X A) → (Sigma (X → A) (λ(f : X → A) → (y : A)
  → Sigma (Sigma X (λ(x : X) → IdP (<!0> A) y (f x))) (λ(x : Sigma X
  (λ(x : X) → IdP (<!0> A) y (f x))) → (y0 : Sigma X (λ(x0 : X) →
  IdP (<!0> A) y (f x0))) → IdP (<!0> Sigma X (λ(x0 : X) → IdP (<!0>
  A) y (f x0))) x y0)))) → λ(A x : U) → ...
```

It takes 8min to compute it, it is about 12MB and it takes 50 hours to typecheck it!

Current and future work

- Normalization and decidability of typechecking (S. Huber's PhD thesis contains canonicity proof)
- Formalize correctness of the model (Orton/Pitts has formalized large parts in Agda in a more general framework, and we are working with M. Bickford to formalize the whole model in Nuprl)
- General formulation and semantics of higher inductive types
- Implement a new, or extend an existing, proof assistant with cubical features (experimental implementation of cubical Agda by A. Vezzosi)

Thank you for your attention!