Cubical Type Theory: a constructive interpretation of the univalence axiom

Anders Mörtberg – Inria Sophia-Antipolis
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.


...
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

“*Homotopy theory* is the study of homotopy groups; and more generally of the category of topological spaces and homotopy classes of continuous mappings”

<table>
<thead>
<tr>
<th>Type theory</th>
<th>Homotopy theory</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Type</td>
<td>A Space</td>
</tr>
<tr>
<td>$a, b : A$</td>
<td>$\bullet$</td>
</tr>
<tr>
<td>$p, q : a = b$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>$\alpha, \beta : p = q$</td>
<td>$\Lambda$</td>
</tr>
<tr>
<td>$\Lambda, \Theta : \alpha = \beta$</td>
<td>$\Theta$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$p$</td>
<td>$\beta$</td>
</tr>
<tr>
<td>$q$</td>
<td>$b$</td>
</tr>
</tbody>
</table>
Voevodsky’s univalence axiom

Equivalence of types, $\text{Equiv } A B$, is a generalization of bijection of sets

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

\[
\text{univalence} : \text{Equiv } (A = B) \rightarrow (\text{Equiv } A B)
\]

In particular we get a map:

\[
\text{univalence_inv} : \text{Equiv } A B \rightarrow A = B
\]
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 \((P : \text{Type} \to \text{Type}) (A B : \text{Type}) (p : \text{Equiv} A B) : P A \to P B := \ldots\)

This can be seen as a new version of Leibniz’s principle: reasoning is invariant under equivalence
Structure identity principle: univalence lifts to structures
(Coquand-Danielsson, Ahrens-Kapulkin-Shulman)

Definition transport_monoid \( (P : \text{Monoid} \to \text{Type}) \)
\( (A \ B : \text{Monoid}) \ (p : \text{EquivMonoid} \ A \ B) : P \ A \to P \ B := \ldots \)

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
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 $\eta$ 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:

\[
\Gamma \vdash \\
\Gamma, i : \mathbb{I} \vdash
\]

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

\[
r, s ::= 0 \mid 1 \mid i \mid 1 - r \mid r \land s \mid r \lor s
\]

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

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

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

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

\[
\begin{array}{ccc}
A(i/0)(j/1) & \xrightarrow{A(j/1)} & A(i/1)(j/1) \\
\uparrow & & \uparrow \\
A(i/0) & \xrightarrow{A} & A(i/1) \\
\downarrow & & \downarrow \\
A(i/0)(j/0) & \xrightarrow{A(j/0)} & A(i/1)(j/0)
\end{array}
\]

and so on...
Path types: rules

\[
\frac{\Gamma \vdash A}{\Gamma \vdash \langle \bar{i} \rangle \cdot t : \text{Path} \ A \ t(i/0) \ t(i/1)}
\]

\[
\frac{\Gamma \vdash A \quad \Gamma, i : \mathbb{I} \vdash t : A}{\Gamma \vdash (\langle \bar{i} \rangle \cdot t) \cdot r = t(i/r) : A}
\]

\[
\frac{\Gamma \vdash t : \text{Path} \ A \ u_0 \ u_1}{\Gamma \vdash t \ 0 = u_0 : A}
\]

[Further rules follow]
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 $r : \mathbb{I}$

$$a \xrightarrow{p} b \quad b \xrightarrow{\langle i \rangle p \ (1-i)} a$$
Path types are great! (function extensionality)

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

\[
p : (x : A) \to \text{Path} \; B \; (f \; x) \; (g \; x)
\]

we can prove that the functions are equal by:

\[
\langle i \rangle \; \lambda x : A. \; p \; x \; i : \text{Path} \; ((x : A) \to B) \; f \; g
\]
Path types are great! (maponpaths)

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

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

satisfying definitionally:

$$\text{ap } \text{id} \ p = p$$
$$\text{ap } (f \circ g) \ p = \text{ap } f \ (\text{ap } g \ p)$$

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:

\[ a \xrightarrow{p} b \quad b \xrightarrow{q} c \]

We do this by computing the dashed line in:

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 $\Gamma, \varphi$ where $\varphi$ is a “face formula” representing a subset of the faces of a cube

$$\varphi, \psi ::= 0_F | 1_F | (i = 0) | (i = 1) | \varphi \land \psi | \varphi \lor \psi$$
Partial types

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

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

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

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

$\begin{array}{c}
\bullet \\
A(i/0) \\
\downarrow \\
\bullet A(j/0)
\end{array}$

$\begin{array}{c}
\bullet \\
A(i/1) \\
\downarrow \\
\bullet
\end{array}$

$\begin{array}{c}
\uparrow \\
j \\
i
\end{array}$
Partial elements

Any judgment valid in a context $\Gamma$ is also valid in a restriction $\Gamma, \varphi$

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

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

$$
\frac{
\Gamma \vdash b : A \\
\Gamma, \varphi \vdash a : A \\
\Gamma, \varphi \vdash a = b : A
}{
\Gamma \vdash b : A[\varphi \mapsto a]
}$$
Box principle

We can now formulate the box principle in type theory:

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

- \(a_0\) is the bottom
- \(u\) is the sides
- \(\text{comp}^i A[\varphi \mapsto u] \ a_0\) is the lid

Equality judgments for \(\text{comp}^i A[\varphi \mapsto u] \ a_0\) are defined by cases on \(A\)
Composition operations: example

With composition we can justify transitivity of path types:

\[
\begin{align*}
\Gamma \vdash p : \text{Path } A \ a \ b \quad & \quad \Gamma \vdash q : \text{Path } A \ b \ c \\
\Gamma \vdash \langle i \rangle \ \text{comp}^j \ A \ [(i = 0) \mapsto a, (i = 1) \mapsto q \ j] \ (p \ i) : \text{Path } A \ a \ c
\end{align*}
\]
Cast as a composition

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

$$\begin{array}{c}
\Gamma, i : \mathbb{I} \vdash A \\
\Gamma \vdash a : A(i/0)
\end{array} \quad \frac{\Gamma \vdash \text{cast}^i A \ a = \text{comp}^i A \ [] \ a : A(i/1)}{
\Gamma \vdash \text{cast}^i A \ a}
$$

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 \( \text{nat} \) be unary natural numbers and \( \text{binnat} \) be binary natural numbers. We have an equivalence

\[
e: \text{nat} \rightarrow \text{binnat}
\]

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

\[
\text{nat} \xrightarrow{P} \text{binnat}
\]
Example: unary and binary numbers

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

\[
\begin{array}{ccc}
\text{nat} & \overset{P}{\longrightarrow} & \text{binnat} \\
\downarrow & & \downarrow \\
e & \vdash & \in \\
\text{binnat} & \overset{\text{id}}{\longrightarrow} & \text{binnat}
\end{array}
\]

We write

\[
P = \langle i \rangle \text{Glue binnat} [(i = 0) \leftrightarrow (\text{nat, } e), (i = 1) \leftrightarrow (\text{binnat, } \text{id})]
\]
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 \( e : \text{Equiv} \ A \ B \) and \( a : A \):

\[
\text{Path} \ B \ (\text{cast} \ (\text{ua} \ e) \ a) \ (e.1 \ a)
\]
Univalence

Given $e : \text{Equiv } A \ B$ we can define the term

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

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
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...)

Inria
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 $\mathbb{Z}$-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
- $T \simeq S^1 \times S^1$ (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!