# The Internal Language of Univalent Categories

Niels van der Weide

# Type Theory

- In this talk, we study type theory
- Frequently, one wants to have a nice metatheory: normalization, canonicity, ...
- This is to guarnatee correct implementations
- However, it tends to be quite complicated to prove such properties

### Semantical Methods

- Semantical methods are a common method to prove metatheoretical properties
- The ideas originated from Tait (strong normalization of the simply typed lambda calculus) and it has been refined to logical relations and categorical gluing
- Note: these proofs can still be rather complicated
- Hence, we want to formalize semantical methods
- Benefits: correctness, usability in larger developments, modularity, collaboration

### Problem

- Big problem: to formalize the metatheory of type theory in a proof assistant
- We focus on the categorical semantics and proofs
- In this talk, we look at internal language theorems that relate categorical models and type theories

# Internal Languages

- ► A celebrated result by Clairambault and Dybjer says that extensional type theory with ∏-types and ∑-types is the internal language of locally Cartesian closed categories<sup>1</sup>
- ▶ Precisely: the bicategories of democratic categories with families with ∏, ∑, and =<sub>ext</sub> and of locally Cartesian closed categories are biequivalent

This theorem is nice, because

- it gives **soundness**: the language can be *interpreted*
- it gives completeness: every model generates a language
- ▶ it deals with translations: functoriality

<sup>1</sup>Originally by Seely, but Seely's proof incorrectly dealt with substitution

### Foundations

- To formalize the internal language theorem, we need suitable foundations
- Since our focus is on categorical semantics, we would like to use foundations whose strength is formalizing with category theory
- Enter: univalent foundations

### Univalent Foundations

- Univalent foundations: extension of Martin-Löf type theory with the univalence axiom
- Univalence axiom: equivalence of types is equality (whenever two types are equivalent, then they have the same properties)
- In recent years, univalent foundations has been established as convenient language to reason about category theory
- Hence, univalent foundations provide a good framework to formally study the metatheory of type theory

# Category Theory in Univalent Foundations

- There are two flavors of category in univalent foundations: strict categories and univalent categories
- Strict categories: invariant under isomorphism
- Univalent categories: invariant under adjoint equivalence
- Note: univalent categories are more common than strict ones (sets and presheaves)
- This suggests that there are two views on the internal language theorem

## The Internal Language Theorem in UF

- The proof by Clairambault and Dybjer of the internal language theorem can almost verbatim be formalized for strict categories
- However, for univalent categories it is more subtle
- To reason about sets and presheaves, one needs this theorem for univalent categories<sup>2</sup>
- Main question of this talk: what is the internal language of univalent categories?

<sup>&</sup>lt;sup>2</sup>The alternative would be using iterative sets (Gylterud and Stenholm)



Introduction to Univalent Categories

An Introduction to Comprehension Categories

The Main Theorem

### Introduction to Univalent Categories

An Introduction to Comprehension Categories

The Main Theorem

What was univalent foundations again?

### Definition

Let A and B be types. By path induction, we define a map **idtoequiv**<sub>A,B</sub> that sends paths A = B to equivalences  $A \simeq B$ .

Axiom (Univalence Axiom)

The map  $idtoequiv_{A,B}$  is an equivalence of types for all A and B.

## Consequences of the Univalence Axiom

- Whenever two types are equivalent, then they share the same properties
- It is not that case that whenever p, q : x = y, that we also have p = q (for example, the universe)

# Sets in Univalent Foundations

#### Definition

A type A is a **set** if for all x, y : A and all p, q : x = y, we have that p = q.

# Sets in Univalent Foundations

### Definition

A type A is a **set** if for all x, y : A and all p, q : x = y, we have that p = q.

Intuitively:

- Think of a type A as a space
- Terms: points
- Proofs of x = y: paths in A
- Set: given by the points

## Categories

### Definition

A category<sup>3</sup>  $\mathcal{C}$  is given by

- a type  $C_0$  of objects (we write C instead of  $C_0$ )
- for all objects x, y : C a set  $x \to y$  of morphisms
- ▶ for all objects x : C an **identity morphism**  $id_x : x \to x$
- ▶ for all morphisms  $f : x \to y$  and  $g : y \to z$ , a composition  $f \cdot g : x \to z$

such that the composition is associative and the identity is neutral with regards to composition

<sup>&</sup>lt;sup>3</sup>The HoTT book says *precategory* for this notion, but I don't

# Univalent Categories

### Definition

Let x and y be objects in a category C. By path induction, we define a map **idtoiso**<sub>x,y</sub> that sends paths x = y to isomorphisms  $x \cong y$ .

#### Definition

A category is called **univalent** if the map  $idtoiso_{x,y}$  is an equivalence of types for all x and y.

# Compare to the Univalence Axiom

### Definition

Let A and B be types. By path induction, we define a map **idtoequiv**<sub>A,B</sub> that sends paths A = B to equivalences  $A \simeq B$ .

Axiom (Univalence Axiom)

The map  $idtoequiv_{A,B}$  is an equivalence of types for all A and B.

### And Strict Categories

#### Definition

A category is called **strict** if the type of objects is strict.

There are several advantages to univalent categories

1. Category theory often studies categories up to **adjoint equivalence**. Properties of univalent categories are invariant under equivalence, whereas for strict categories, they are invariant under isomorphism.

There are several advantages to univalent categories

- 1. Category theory often studies categories up to **adjoint equivalence**. Properties of univalent categories are invariant under equivalence, whereas for strict categories, they are invariant under isomorphism.
- 2. Univalence allows us to do induction on adjoint equivalences, and this simplifies some proofs.

There are several advantages to univalent categories

- 1. Category theory often studies categories up to **adjoint equivalence**. Properties of univalent categories are invariant under equivalence, whereas for strict categories, they are invariant under isomorphism.
- 2. Univalence allows us to do induction on adjoint equivalences, and this simplifies some proofs.
- 3. For univalent categories, one can constructively prove that every fully faithful and essentially surjective functor is an equivalence, but not for strict categories.

There are several advantages to univalent categories

- 1. Category theory often studies categories up to **adjoint equivalence**. Properties of univalent categories are invariant under equivalence, whereas for strict categories, they are invariant under isomorphism.
- 2. Univalence allows us to do induction on adjoint equivalences, and this simplifies some proofs.
- 3. For univalent categories, one can constructively prove that every fully faithful and essentially surjective functor is an equivalence, but not for strict categories.
- 4. Univalent categories are more common than strict categories. For example, the categories of sets, presheaves, and sheaves are univalent, but not strict.

## What do we want?

- We want to study the internal language of univalent categories
- The ultimate goal is to develop a type theory that we can use to reason about univalent categories
- Analogous theorem: Martin-Löf type theory is the internal language of locally Cartesian closed categories (Clairambault and Dybjer)

Introduction to Univalent Categories

An Introduction to Comprehension Categories

The Main Theorem

## Categories with Families?

- Clairambault and Dybjer use categories with families (CwF) to prove their internal language theorem
- Note: in a CwF, the types form a set
- However, in the set model of type theory, types in the empty context are given by sets
- Since the collection of sets is not a set, we do not use CwFs in our work

## Our Requirements

We are looking for a **categorical structure** in which we can interpret dependent type theory such that

- the types do not have to form a set (this would invalidate the set/presheaf/sheaf model)
- we use universal properties to express substitution (this simplifies dealing with coherence)

## Our Requirements

We are looking for a **categorical structure** in which we can interpret dependent type theory such that

- the types do not have to form a set (this would invalidate the set/presheaf/sheaf model)
- we use universal properties to express substitution (this simplifies dealing with coherence)

We shall see that **comprehension categories** satisfy these requirements.

## Our Requirements

We are looking for a **categorical structure** in which we can interpret dependent type theory such that

- the types do not have to form a set (this would invalidate the set/presheaf/sheaf model)
- we use universal properties to express substitution (this simplifies dealing with coherence)

We shall see that **comprehension categories** satisfy these requirements.

But... what are they?

In the beginning, there were hyperdoctrines

- The notion of comprehension category is inspired by hyperdoctrines
- Hyperdoctrines were introduced as a categorical model for first-order predicate logic
- Hyperdoctrines come in many different flavors to incorporate different type formers and forms of logic
- We shall discuss a basic notion of hyperdoctrines

# Hyperdoctrines, formally

### Definition

- A hyperdoctrine is given by
  - $\blacktriangleright$  a category  ${\cal C}$  with finite products
  - ▶ a presheaf  $P : C^{op} \rightarrow Lat$

# Hyperdoctrines, formally

### Definition

A hyperdoctrine is given by

- a category C with finite products
- ▶ a presheaf  $P : C^{op} \rightarrow Lat$

#### Explanation:

- ▶ objects in C are **contexts**, morphisms are **substitutions**
- elements of  $P(\Gamma)$  are **formulas** in context  $\Gamma$
- the action of P on morphisms: substitution of formulas
- the lattice operations give connectives

# Hyperdoctrines, formally

### Definition

A hyperdoctrine is given by

- a category C with finite products
- ▶ a presheaf  $P : C^{op} \rightarrow Lat$

#### Explanation:

- ▶ objects in C are **contexts**, morphisms are **substitutions**
- elements of  $P(\Gamma)$  are **formulas** in context  $\Gamma$
- the action of P on morphisms: substitution of formulas
- the lattice operations give connectives

For simplicity, we shall focus on presheaves  $\mathcal{C}^{op} \to \textbf{Set}$ 

Remember our first requirement

the types do not have to form a set (this would invalidate the set/presheaf/sheaf model)

This is not satisfied since lattices have a set of elements. So: we must **categorify** the definition of hyperdoctrines

# Categorifying Hyperdoctrines

Category Theory	Bicategory Theory
Functor	Pseudofunctor
Set	Cat
Functor from $\mathcal{C}^{op}$ to $\mathbf{Set}$	Pseudounctor from $\mathcal{C}^{op}$ to <b>Cat</b>

### However...

- The definition of pseudofunctor is quite complicated
- Why? We have to write down all coherences and the definition is 2-categorical
- Precisely: 5 pieces of data and 7 laws
- This is why we want to use universal properties: for those, the coherences follow automatically

The main idea: Grothendieck construction

Theorem

Functions  $A \rightarrow B$  are the same as families of types Y over B.

Theorem

Functions  $A \rightarrow B$  are the same as families of types Y over B.

Proof.

Given  $f : A \rightarrow B$ , define the family that sends b : B to its fiber:

$$\sum_{a:A} b = f(a)$$

Theorem

Functions  $A \rightarrow B$  are the same as families of types Y over B.

Proof.

Given  $f : A \rightarrow B$ , define the family that sends b : B to its fiber:

$$\sum_{a:A} b = f(a)$$

Given a type family Y over B, define the total space  $\int Y$  of Y:

$$\sum_{b:B} Y(b)$$

Then we have a function  $\int Y \to B$  (first projection).

Theorem

Functions  $A \rightarrow B$  are the same as families of types Y over B.

Proof.

Given  $f : A \rightarrow B$ , define the family that sends b : B to its fiber:

$$\sum_{a:A} b = f(a)$$

Given a type family Y over B, define the total space  $\int Y$  of Y:

$$\sum_{b:B} Y(b)$$

Then we have a function  $\int Y \rightarrow B$  (first projection). The Grothendieck construction: this, but for categories

### Enter Grothendieck

Suppose, we have  $F : \mathcal{C}^{op} \to \mathbf{Cat}$ . Define the category  $\int F$ :

- Objects: pairs of x : C and  $\overline{x} : F(x)$
- Morphisms from  $(x, \overline{x})$  to  $(y, \overline{y})$ : pairs of  $f : x \to y$  and  $\overline{x} \to F(f)(\overline{y})$

### Enter Grothendieck

Suppose, we have  $F : \mathcal{C}^{op} \to \mathbf{Cat}$ . Define the category  $\int F$ :

- Objects: pairs of x : C and  $\overline{x} : F(x)$
- Morphisms from  $(x, \overline{x})$  to  $(y, \overline{y})$ : pairs of  $f : x \to y$  and  $\overline{x} \to F(f)(\overline{y})$

Note:

- We have x : C and y : C and  $f : x \to y$
- $\overline{x}$  : F(x) and  $\overline{y}$  : F(y)
- F(f) is a functor from F(y) to F(x) (contravariant)
- So:  $F(f)(\overline{y}) : F(x)$
- So:  $F(x) \to F(f)(\overline{y})$  is well-typed (morphism in F(x))

### Enter Grothendieck

Suppose, we have  $F : \mathcal{C}^{op} \to \mathbf{Cat}$ . Define the category  $\int F$ :

- Objects: pairs of x : C and  $\overline{x} : F(x)$
- Morphisms from  $(x, \overline{x})$  to  $(y, \overline{y})$ : pairs of  $f : x \to y$  and  $\overline{x} \to F(f)(\overline{y})$

Note:

- We have x : C and y : C and  $f : x \to y$
- $\overline{x}$  : F(x) and  $\overline{y}$  : F(y)
- F(f) is a functor from F(y) to F(x) (contravariant)

So: 
$$F(f)(\overline{y}) : F(x)$$

• So:  $F(x) \to F(f)(\overline{y})$  is well-typed (morphism in F(x)) We have a functor  $\int F \to C$ 

# However... (again)

- Suppose, we have a functor G : E → C. Can G be constructed as in the previous slide?
- ► Nope.
- Counterexample,  $Mon \rightarrow Set$
- We need to assume that G is a **fibration**

## **Defining Fibrations**

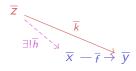
We define fibrations in two steps

- we define cartesian morphisms (expresses the universal property)
- we define fibrations

### Cartesian morphisms

#### Definition

A morphism  $\overline{f} : \overline{x} \to \overline{y}$  over  $f : x \to y$  is called **cartesian** if for all  $h : z \to \overline{x}$  and  $\overline{k} : \overline{z} \to \overline{y}$  over  $h \cdot f$  there is a unique  $\overline{h} : \overline{z} \to \overline{x}$  such that  $\overline{h} \cdot \overline{f} = \overline{k}$ .



$$z - h \rightarrow x - f \rightarrow y$$

#### Fibrations

#### Definition

We say that *F* is a **fibration** if for all  $f : x \to y$  and  $\overline{y}$  there is a cartesian morphism  $\overline{f}$  over *f*.

 $\overline{X} - \overline{f} \to \overline{y}$  $X - f \to y$ 

### Example of a Fibration

- ► Suppose, C has pullbacks
- We write C<sup>→</sup> for the arrow category of C whose objects are morphisms x → y in C
- The functor cod : C<sup>→</sup> → C sends morphisms x → y to their codomain y.
- Then cod is a fibration



### Wait... where were we?

Let's briefly recall where we are.

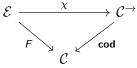
- We defined the notion of **fibration**
- Fibrations allow us to interpret dependent types and substitution
- The motivation behind fibration comes from hyperdoctrines and the Grothendieck construction (fiberwise representation of functions)
- This is the first ingredient for defining comprehension categories
- The missing part: context extension

Note: this notion is technical and hard to understand when you see it for the first time

# **Comprehension Categories**

#### Definition

A (full) comprehension category is a commuting triangle of functors



such that C has a terminal object [] and such that  $\chi$  sends cartesian morphisms to pullbacks and such that  $\chi$  is fully faithful.

Introduction to Univalent Categories

An Introduction to Comprehension Categories

The Main Theorem

# **Type Formers**

To state our main theorem, we need the following type formers in comprehension categories

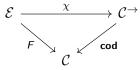
- Unit types (fiberwise terminal object)
- Binary product types (fiberwise products)
- Equalizer types (fiberwise equalizers)
- Dependent sums (*left adjoint to substitution*)

We also need **democracy** 

#### Democracy

#### Definition

Suppose that we have the following comprehension category



We say that it is **democratic** if for every context  $\Gamma : C$  there is a type  $\overline{\Gamma} : \mathcal{E}$  over [] such that  $\chi(\overline{\Gamma}) \cong \Gamma$ .

Intuitively: every context has a representative.

### The Theorem

A **DFL comprehension category** is a comprehension category that

- is democratic
- has unity types
- supports binary products
- supports equalizer types
- supports dependent sums

#### Theorem

We have a biequivalence between the bicategories of univalent DFL comprehension categories and of univalent categories with finite limits.

#### Extensions

We extended this to:

- ▶ locally Cartesian closed categories (∏-types)
- extensive categories (disjoint sum types)
- exact categories (quotient types)
- pretoposes (quotient types and disjoint sum types)
- ► ∏-pretoposes (pretopos with ∏-types)

## Usage of Univalence in the Proof

There are several points where we used univalence to simplify the proof.

- Transporting structure along equivalences
- Classifying equivalences
- Splitting: substitution laws hold up to isomorphism in fibrations

## Conclusion

We did:

- A formalization of the biequivalence between univalent comprehension categories and locally Cartesian closed univalent categories
- ► An **extension** to pretoposes, ∏-pretoposes, and elementary toposes
- The formalization is available online<sup>4</sup>

Important points:

- We used comprehension categories instead of categories with families
- Univalence made the proof simpler

<sup>&</sup>lt;sup>4</sup>https://github.com/UniMath/UniMath/tree/master/UniMath/ Bicategories/ComprehensionCat