Unfolding FOLDS
HoTT/UF Workshop
Sept. 9, 2017

Matthew Weaver and Dimitris Tsementzis

Princeton
Rutgers
The Syntax of Syntax

• Type theory has a rich syntax...
• ...which is why we love it!
• ...and is also what makes everything difficult
The Syntax of Syntax

• We often encounter the situation where we can define a construct in the metatheory, but not internally

• **Challenge:** Let’s make type theory express its own metatheory

• **Bonus Challenge:** Let’s do so in a way that is well-typed and preserves logical consistency
Let's make type theory eat itself!
The Syntax of Syntax

• Meta-programming and reflection are already everywhere

Tactic languages in proof assistants:

```coq
Definition PreShv_to_slice_is_funct : is_functor PreShv_to_slice_data. 
  Proof. 
  split; [intros X | intros X Y Z f g]; 
  apply eq_mor_slicecat; 
  apply (nat_trans_eq has_homsets_HSET); 
  unfold PreShv_to_slice_ob_nat , PreShv_to_slice_ob_funct_fun; 
  intro c; 
  apply funextsec; intro p; 
  now rewrite tppr. 
Defined.
```
The Syntax of Syntax

- Meta-programming and reflection are already everywhere

Generic programming over datatypes:

data Tm = Var TName 
  | App Tm Tm
  | Lam (Bind (TName, Embed Tm) Tm)
  | Pi (Bind (TName, Embed Tm) Tm)
  | Type
  | Kind
  deriving (Show, Generic, Typeable)
The Syntax of Syntax

- Meta-programming and reflection are already everywhere

Reflection of abstract syntax:

```
idNat : Nat -> Nat
idNat = %runElab (do intro `{{x}}
  fill (Var `{{x}})
solve)
```
The Syntax of Syntax

• Meta-programming and reflection are already everywhere

Classical mathematics:

\[ \frac{d}{dx} x^n = nx^{n-1} \]
The Syntax of Syntax

• In many cases, it is an untrusted extension of the theory that can break its good properties
The Syntax of Syntax

• We define a univalent type theory that can safely manipulate and interpret (some of) its own syntax

• Using this, we propose a novel approach to defining the type of semi-simplicial types

• We also describe a general framework to describe the semantics of reflection in type theory
What are Semi-Simplicial Types?

- A “0-dimensional triangle” is a point
- A “1-dimensional triangle” is a line
- A “2-dimensional triangle” is a triangle
- A “3-dimensional triangle” is a pyramid/tetrahedron made from 4 triangles, etc…
What are Semi-Simplicial Types?

• Consider a type of points $T_0$

• For any two terms (i.e. points) $x$ and $y$ in $T_0$, there is a type $T_1 \times x \times y$ of lines between $x$ and $y$

• For any three points $x$, $y$, and $z$, and three lines $a : T_1 \times x \times y$, $b : T_1 \times y \times z$, and $c : T_1 \times x \times z$, there is a type $T_2 \ a \ b \ c$ of triangles outlined by $a$, $b$ and $c$

• etc…
What are Semi-Simplicial Types?

\[ \Sigma T_0 : \text{Type}, \]

\[ \Sigma T_1 : (\Pi x y : T_0, \text{Type}), \]

\[ \Sigma (T_2 : \Pi (x y z : T_0) (a : T_1 x y) (b : T_1 y z) (c : T_1 x z), \text{Type}), \]

e tc...
What are Semi-Simplicial Types?

• The type of \( n \)-truncated semi-simplicial types (\( \text{sst}_n \)) is given by \( \Sigma T_0, \Sigma T_1, \ldots, T_n \)

• It is a known result that type of semi-simplicial types is the homotopy limit of \( \text{sst}_n \) over \( n : \mathbb{N} \) [ACS15]

• The homotopy limit is constructed with the following syntax where where \( \pi_n \) is the obvious projection from \( \text{sst}_{n+1} \) to \( \text{sst}_n \):

\[
\sum \prod (x : \Pi_{(n : \mathbb{N})} \text{sst}_n) (n : \mathbb{N}) \pi_n x_{n+1} = x_n
\]
What are Semi-Simplicial Types?

• Defining the function \( \text{sst} : \mathbb{N} \to \text{Type} \) picking out the \( n \)-truncated semi-simplicial type proves challenging:

  • All the dependencies in the types require proving equalities on terms of arbitrary types...

  • …which require proving equalities on proofs of equalities of terms of arbitrary types...

  • …and then proving equalities on proofs of equalities of proofs of equalities of terms of arbitrary types...

  • …etc…
What is FOLDS?

• First Order Logic with Dependent Sorts: FOL where sorts can be indexed by elements of other sorts (i.e. dependent types)

• A FOLDS Signature is a context of dependent sorts (equivalently a Finite Inverse Category)

• Example: Cat
  
  \( O : \text{Sort} \)
  
  \( A : O \times O \rightarrow \text{Sort} \)
  
  \( I : \Pi x : O, A \times x \rightarrow \text{Sort} \)

• Note: The type of \( n \)-truncated semi-simplicial types is such a signature with a sort for each dimension
Our Theory

• TT+I is a type theory that includes:
  • \(\Pi\)-types, \(\Sigma\)-types, id-types, \(\mathbb{N}\), 1
  • A type \(\text{Sig}\) of FOLDS signatures
  • An interpretation function \(I : \text{Sig} \to \text{Type}\)
Our Theory

• The type \( \text{Sig} \) of well-formed FOLDS signatures is built using the following:

  • \( \text{Sig} : \text{Type} \) is a list of well-formed contexts \( \text{Ctx} \), each representing a sort by its dependencies

  • \( \text{Ctx} : \text{Sig} \rightarrow \text{Type} \) is a list of sorts previously defined in the signature

• **Example:** representing \( \text{Cat} : \text{Sig} \)

  \[
  \text{Cat} := \text{O} : \cdot, \text{A} : (c : \text{O}, d : \text{O}), \text{I} : (x : \text{O}, i : \text{A x x})
  \]
Our Theory

- The type **Sig** of well-formed FOLDS signatures is built using the following:
  - **Sig : Type** is a list of well-formed contexts **Ctx**, each representing a sort by its dependencies
  - **Ctx : Sig → Type** is a list of sorts previously defined in the signature
  - ...a couple other helper types

- **Sig** and **Ctx** are both h-sets (along with the other types)

- Complete definition is a quotient-inductive-inductive type in Agda à la type theory in type theory [AK16]
Our Theory

- The type $\text{Sig}$ of well-formed FOLDS signatures is built using the following:
  - $\text{Sig} : \text{Type}$ is a list of well-formed contexts $\text{Ctx}$, each representing a sort by its dependencies
  - $\text{Ctx} : \text{Sig} \rightarrow \text{Type}$ is a list of sorts previously defined in the signature

- The interpretation function $I : \text{Sig} \rightarrow \text{Type}$ is defined as

  $$I(\Gamma_0, \Gamma_1, \ldots, \Gamma_n) := \Sigma(T_0:[\Gamma_0]\rightarrow\text{Type}), \Sigma(T_1:[\Gamma_1]\rightarrow\text{Type}), \ldots, [\Gamma_n]\rightarrow\text{Type}$$
Our Theory

• The interpretation function $I : \text{Sig} \to \text{Type}$ is defined as

$$I(\Gamma_0, \Gamma_1, \ldots, \Gamma_n) := \Sigma(T_0: \llbracket \Gamma_0 \rrbracket \to \text{Type}), \Sigma(T_1: \llbracket \Gamma_1 \rrbracket \to \text{Type}), \ldots, \llbracket \Gamma_n \rrbracket \to \text{Type}$$

• Example: representing $\text{Cat}$ in $\text{Sig}$

$$\text{Cat} := O : \cdot, A : (c : O, d : O), I : (x : O, i : A \times x)$$

$$I(\text{Cat}) := \Sigma(O : \text{Type}), \Sigma(A : O \times O \to \text{Type}), (\Sigma(x : O), A(x, x)) \to \text{Type}$$
Defining Semi-Simplicial Types

1. Define \( \text{sst}' : \mathbb{N} \to \text{Sig} \), picking out the \( n \)-truncated semi-simplicial type leveraging the strictness of \( \text{Sig} \) and \( \text{Ctx} \).

2. \( \text{sst} : \mathbb{N} \to \text{Type} \coloneqq \text{I} \circ \text{sst}' \)

3. \[ \sum \prod_{(x : \prod_{(n : \mathbb{N})} \text{sst}_n)} (n : \mathbb{N}) \pi_n x_{n+1} = x_n \]
How is this Reflection?

- **Sig** is a datatype representing the abstract syntax of the types corresponding to well-formed FOLDS signatures.

- **I** is the interpretation function decoding terms of **Sig** into the types they represent.

- **Note:** we only decode representations of terms, and never encode actual terms.
So, what does this theory even mean?
Decoding the Universe(s)

- **(Informal) Definition:** A universe (à la Tarski) consists of a type $U$ along with a decode function $el : U \rightarrow Type$

- Our type $\text{Sig}$ with interpretation function $I$ is such a universe!
Decoding the Universe(s)

• (incomplete) Definition: Fix a category $\mathcal{C}$. A category with families (CwF) is a model of type theory with contexts given by $\mathcal{C}$ described by the following data:

• A presheaf $\mathsf{Ty}: \mathcal{C}^{\text{op}} \to \mathbf{Set}$, where $\mathsf{Ty}(\Gamma)$ is the set of all well-formed types in context $\Gamma$

• A presheaf $\mathsf{Tm}: \int \mathsf{Ty}^{\text{op}} \to \mathbf{Set}$, where $\mathsf{Tm}(\Gamma, A)$ is the set of all well-typed terms of type $A$ in context $\Gamma$

• $\int \mathsf{Ty}$ is the category of elements of $\mathsf{Ty}$, consisting of pairs of contexts and well-formed types in that context
Decoding the Universe(s)

- **Definition:** Fix a CwF with presheaves $Ty : \mathcal{C}^{\text{op}} \to \text{Set}$, and $Tm : \int Ty^{\text{op}} \to \text{Set}$. A universe is given by
  
  • A presheaf $U : \mathcal{C}^{\text{op}} \to \text{Set}$,
  
  • A decoding natural transformation $\text{el} : U \to Ty$,
  
  • Types $U_\Gamma \in Ty(\Gamma)$ for every $\Gamma \in \mathcal{C}$ where $Tm(\Gamma, U_\Gamma) = U(\Gamma)$ and the action of morphisms on the $U_\Gamma$ is given by $U$
  
  • Definition appears as 2-level CwF derived from a universe in Paolo’s thesis [Cap17]
So, what does this theory even mean?

...we’ve also defined a 2-level type theory.
Decoding the Universe(s)

- While this notion of a universe can express adding a second universe with a strict equality on its codes of types, it doesn’t provide a way to model strictness on (representations of) terms

- Captures Sig, but not the other types used to build Sig/ their strictness
From Universes to Reflection: (Idealized) Semantics

• Definition: A *type theory with reflection* is given by

  • a category with families \((Ty, Tm)\) with universe \((U, el)\)
  
  • a presheaf \(R : \int U^{\text{op}} \to \text{Set}\)
  
  • a natural transformation \(i : el[R] \to Tm\)

• Here \(el[-]\) denotes the functor \((\int U^{\text{op}} \to \text{Set}) \to (\int Ty^{\text{op}} \to \text{Set})\) induced by \(el\)

• elements \(A_{\Gamma} \in Ty(\Gamma)\) for every \(\Gamma \in \mathcal{C}\) and \(A \in U(\Gamma)\) such that \(Tm(\Gamma, A_{\Gamma}) = R(\Gamma, A)\)
From Universes to Reflection: (Idealized) Semantics

• Haven’t yet worked out if/how TT+I is a model of a type theory with reflection

• Part of what makes Sig powerful is it has an inductor. Not (yet) generalized in semantics I’ve proposed

  • The presence of an inductor is often assumed when one thinks of reflection of abstract syntax in general

• Can this be used to model more extensive (and safe!) reflection of abstract syntax in (univalent) type theory?
Connection to 2-Level Type Theory

- 2-Level Type Theory begins with MLTT+Axiom-K, and adds a second univalent universe that decodes into MLTT
  - MLTT+Axiom-K has two equality types: the strict one with axiom-K, and the one used to decode the equality of the univalent universe
- We begin with HoTT and add a second strict universe that decodes into HoTT
  - We have a single univalent equality type
Some Future Work

- Finish defining TT+I, implement it and define the type of semi-simplicial types

- Investigate simpler theories that can define the type of semi-simplicial types and only have one notion of equality

- Investigate whether definition of type theory with reflection makes any sense

  - If so, see what other interesting theories it models
Takeaways

• Make a (nonstandard) universe in which difficult problems are easy!

• Safe/consistent reflection in (univalent) type theory is both interesting and possible


Unfolding FOLDS
HoTT/UF Workshop
Sept. 9, 2017

Matthew Weaver and Dimitris Tsementzis

Princeton
Rutgers