Foundations of Type theory for HoTT

Siddhartha Gadgil

Department of Mathematics
Indian Institute of Science

February 11, 2015

Foundations overview

- We review the foundations of the type theory underlying homotopy type theory.
- This is a literate agda document.
- We must include a module statement, matching the file name.

```
open import Base

module Foundations where
```

Axioms and Rules

- Usual *rigorous* mathematics is based on definitions and aioms, for example *a function* $f : \mathbb{R} \to \mathbb{R}$ *is said to be continuous at* x *if*

$$\forall \epsilon > 0 \exists \delta > 0 \forall y \in \mathbb{R}(|y - x| \leq \delta \implies |f(y) - f(x)| < \epsilon).$$

- We however do not explicitly give rules saying why *a function* $f : \mathbb{R} \to \mathbb{R}$ *is said to be brown at* x *if*

$$\forall \exists \delta \forall z \in \mathbb{R}(|y - f(x)| \leq \delta \implies |f(y) - f(x)||)$$

  makes no sense.
- We thus do not give rules for
  - What is a valid expression.
  - What it represents: term (real number, set etc) or formula (in definitions, theorems).
  - Rules for deduction.
- We will instead give rules for what are valid expressions, and what are their types. We will need very few axioms.

Contexts, terms, types, universes

- A context consists of a collection of *terms*.
- Each term has a *type*, mostly unique (denoted, for example, $a : A$).
- The rules concerning a term are determined by its type.
- Types are also terms.
- A *Universe* is a type $\mathcal{U}$ so that all terms with type $\mathcal{U}$ are themselves types.

Types of rules

We have rules that:

- let us form terms from other terms.
- let us create a new context from a given context, by introducing new terms which can depend on the given context.
- give the result of substituting one term for another (with the same type) in a given term.
- allow us to make say that a term a has a specified type A.
- allow us to conclude that a pair of terms are equal (by definition).
- give a collection of universes, which are present in all contexts.

Universes

- There is a sequence of universes, $\mathcal{U}_0$, U_1, ...
- The universe $\mathcal{U}_i$ has type $\mathcal{U}_{i+1}$.
- These are cumulative, with $U_i \subset U_{i+1}$.
- If a type T has type $\mathcal{U}_i$, it also has type $\mathcal{U}_{i+1}$.

Function types

- If A and B are types, then we can form the function type $A \to B$.
- If $f : A \to B$ is a term of a function type, and $a : A$ is a term, then $f(a)$ is a term that has type B.
- We can form terms of a type $A \to B$ by using a *lambda*-expression $a \mapsto b$.
- Here b is a term of type B formed from the terms in the context together with a term a we introduce and declare to have type A, using the usual rules for forming terms.
- If $f = a \mapsto b : A \to B$, then for $a' : A$, $f(a')$ equals, by definition, the result of substituting a by a' in b.

Π-types

- A *type family* is a function $B : A \to \mathcal{U}$, where A is a type and $\mathcal{U}$ is a universe.
- Given a type family $B : A \to \mathcal{U}$, we can form the type $\Pi_{a:A} B(a)$ of dependent functions.
- Given a dependent function $f : \Pi_{a:A} B(a)$ and a term $a : A$, we can form the term $f(a)$ with type $B(a)$.
- We can form terms of a type $\Pi_{a:A} B(a)$ by using a λ-expression $a \mapsto b$, with b a term of type $B(a)$ formed from the terms in the context together with a term a we introduce and declare to be of type A.
- If $f = a \mapsto b : \Pi_{a:A} B(a)$, then for $a' : A$, $f(a')$ equals, by definition, the result of substituting a by a' in b.

Inductive types: a first look

- We can introduce into a context, simultaneously, a type W *inductively generated* by given *constructors*, and its constructors.
- The constructors for W are terms with specified types, which may depend on W.
- For example, the type $\mathbb{N}$ is inductively generated by the constructors
  - $0 : \mathbb{N}$.
  - $succ : \mathbb{N} \to \mathbb{N}$.
- In Agda, this is

```
data ℕ : Type where
  zero : ℕ
  succ : ℕ → ℕ
```

Inductive types: Lists

- For each type A, $List(A)$ is a type inductively defined by its constructors.
  - $[] : List(A)$.
  - $cons : A \to List(A) \to List(A)$.
- In Agda, this is

```
data List(A : Type) : Type where
  [] : List A
  _::_ : List A → List A → List A
```

Recursion and Induction : a first look

Families

Constructor types for an inductive type

-

Recursion for constructors having families

-

Domains of recursion

-

Recursion functions

Domains of induction

-

Induction functions