# Type Theory with Paige North 7/10

Jason Schuchardt

July 11, 2019

## 1 Dependent Type Theory

The syntax rules were passed out in class. They come from a standard reference.

What is dependent type theory?

For example, we can consider the type $\textsc{List}(T)$. We can consider a pair of lists and send them to a list of pairs

$$([1,2,3],[3,2,1]) \mapsto [(1,3),(2,2),(3,1)]$$

We could then add them together and get the list $[3,4,3]$, and then add these together to get 10. However, we can't do this, because we can't talk about the length of a list in the type.

We want a type $\textsc{List}(n,T)$ for each $n:\mathbb{N}$ and type $T$ such that the "union over $n$" is $\textsc{List}(T)$.

We talked about a type/proposition "12 is composite," but this is awkward. We're usually more interested in the predicate, " ____ is composite," or perhaps with a variable, "$n$ is composite."

Types should also be dependent on hypotheses/variables.
**Example 1.1.** I.e., we want to be able to write something like this,

$$\frac{T \ \textsc{type}}{n:\mathbb{N} \vdash \textsc{List}(n,T) \ \textsc{type}},$$

where the type depends on the parameter $n$. Similarly, we might write our is composite predicate as

$$n:\mathbb{N} \vdash \text{isComp}(n) \ \textsc{type}$$

*Note.* A family of sets $T(x)$ indexed by $\Gamma$ produces a natural function $\bigsqcup_{x\in\Gamma} T(x) \to \Gamma$, which takes an element of the disjoint union and returns the index of the set it came from. Conversely, given a function $\pi : T \to \Gamma$, we get a family $\pi^{-1}(x)$ over $\Gamma$.

This gives a 'bijection'

$$\{\text{families of sets indexed by } \Gamma\} \longleftrightarrow \{\text{functions with codomain } \Gamma\}$$

(ignoring size issues).
**Example 1.2.** For the list type, we have the following:

$$\text{for } n:\mathbb{N},\ \textsc{List}(\text{n,T}) \longleftrightarrow \bigcup_{n\in\mathbb{N}} \textsc{List}(n,T) \to \mathbb{N}$$

| Type | Programming | Logic | Set |
|---|---|---|---|
| $x : \Gamma \vdash T(x)$ TYPE | A program that takes input $x$ "(Given a $n : \mathbb{N}$ print all factors of $n$)" | A predicate, "$n$ is prime" | A family of sets $T(x)$ indexed by $\Gamma$. |

# 2 Definition of Dependent Type Theory

Per Martin-Löf (70s-80s). There are four judgements:

1. Is a type in a context:
$$\Gamma \vdash T \text{ TYPE}$$
2. Type equality in a context:
$$\Gamma \vdash S = T$$
3. Is a term of a type in a context:
$$\Gamma \vdash t : T$$
4. Term equality in a context:
$$\Gamma \vdash s = t : T$$

In the context formation rules that we had before, we had

$$\frac{}{\varnothing \text{ ctxt}}, \quad \frac{\Gamma \text{ ctxt} \qquad T \text{ TYPE}}{\Gamma, x : T \text{ ctxt}}.$$

Now $T$ might depend on $\Gamma$, so we should change this to:

$$\frac{}{\varnothing \text{ ctxt}}, \quad \frac{\Gamma \text{ ctxt} \qquad \Gamma \vdash T \text{ TYPE}}{\Gamma, x : T \text{ ctxt}}.$$

## 2.1 The types

### 2.1.1 $\bot$ type

$$\frac{}{\bot \text{ TYPE}} \quad \bot\text{-formation}$$

Before we had

$$\frac{T \text{ TYPE} \quad \Gamma \vdash f : \bot}{\Gamma \vdash}$$

Now we need to allow it to depend on the context

$$\frac{\Gamma, x : \bot \vdash C(x) \text{ TYPE} \quad \Gamma \vdash f : \bot}{\Gamma \vdash j_{c,f} : C(f)} \quad \bot\text{-elimination}$$

### 2.1.2 $\top$ type

1. $\top$-formation
$$\frac{}{\top \text{ TYPE}}$$
2. $\top$-introduction
$$\frac{}{* : \top}$$

3. $\top$-elimination
$$\frac{\Gamma, x : \top \vdash C(x) \text{ TYPE} \qquad \Gamma \vdash c : C(*)}{\Gamma, x : \top \vdash j_{C,c}(x) : C(x)}$$

4. $\top$-computation
$$\frac{\Gamma, x : \top \vdash C(x) \text{ TYPE} \qquad \Gamma \vdash c : C(*)}{\Gamma, x : \top \vdash j_{C,c}(*) = c : C(*)}$$

In general, we have these four sorts of rules, describing forming the type, introducing terms of the type, and elimination and computation rules describing how to break down the type.

The elimination rule says "If you have a predicate $C$ on terms of a type, then if you want to prove $C$ always holds, then it suffices to prove it just for the canonical terms of your type."
*Question.* We notice that the rules do not say that all elements of $\top$ are equal to the canonical element.

Answer: This is true, and we don't want this. However, we will soon be able to prove that they are equal in a different sense.

We want to prove that two terms are equal. In type theory, prove means construct a term of a type, so we should be constructing a term of an appropriate type. Let's define that type now.

## 2.2 The type Id

1. Id-formation
$$\frac{\Gamma \vdash T \text{ TYPE}}{\Gamma, s : T, t : T \vdash \mathrm{Id}_T(s,t) \text{ TYPE}}$$

2. Id-introduction
$$\frac{\Gamma \vdash T \text{ TYPE}}{\Gamma, t : T \vdash \mathrm{refl}_t : \mathrm{Id}_T(t,t)}$$

3. Id-elimination
$$\frac{\Gamma, s : T, t : T, p : \mathrm{Id}_T(s,t) \vdash C(s,t,p) \text{ TYPE} \qquad \Gamma, t : T \vdash c(t) : C(t,t,\mathrm{refl}_t)}{\Gamma, s : T, t : T, p : \mathrm{Id}_T(s,t) \vdash j(s,t,p) : C(s,t,p)}$$

4. Id-computation
$$\frac{\Gamma, s : T, t : T, p : \mathrm{Id}_T(s,t) \vdash C(s,t,p) \text{ TYPE} \qquad \Gamma, t : T \vdash c(t) : C(t,t,\mathrm{refl}_t)}{\Gamma, t : T \vdash j(t,t,\mathrm{refl}_t) = c(t) : C(t,t,\mathrm{refl}_t)}$$

## 2.3 Relations in sets

We are trying to emulate relations in sets. The idea is that "Equality is the smallest reflexive relation."

Given some function/relation $R \xrightarrow{i} X \times X$, we say that it is reflexive if there exists $r : X \to R$ such that $ir = \Delta$, i.e, if $ir(x) = \Delta(x) := (x, x)$.

For example, we have
$$\mathbb{R} \to [\leq] \hookrightarrow \mathbb{R} \times \mathbb{R},$$
where $[\leq] = \{(x, y) \in \mathbb{R} \times \mathbb{R} : x \leq y\}$. Then the left hand map is $r \mapsto (r, r)$.

The map $i$ produces a family of sets $i^{-1}(x, y)$ indexed by $X \times X$. If $i$ is an injection, then these preimages are either empty or contain exactly one element.

We can define equality to be the smallest reflexive relation. I.e., given a set $X$, we have the relation,

$$X \xrightarrow{\mathrm{id}_X} X \xrightarrow{\Delta} X \times X,$$

$$x \mapsto x \mapsto (x, x)$$

Then

$$\Delta^{-1}(x, y) = \begin{cases} \varnothing & \text{if } x \neq y \\ \{x\} & \text{if } x = y. \end{cases}$$

Given a reflexive relation, we get a function

$$\Delta^{-1}(x, y) \to i^{-1}(x, y).$$

In other words, given a reflexive relation, $i : R \to X \times X$, we get a map $f : X \to R$ such that $if = \Delta$. Indeed, this is obvious, $f$ is the map $r$ from the definition of reflexivity.

## 2.4 Relations in type theory

A relation on $T$ is a type

$$x : T, y : T \vdash R(x, y) \text{ TYPE}.$$

"Says that $s$ and $t$ are related if there is a term in $R(s, t)$."

We will say that $R$ is reflexive if

$$t : T \vdash \rho_t : R(t, t).$$

Now Id is the smallest reflexive relation, in the sense that

$$x : T, y : T, p : \mathrm{Id}_T(x, y) \vdash ? : R(x, y)$$

for any reflexive relation $R$.

We begin with saying $R(x, y)$ is a type,

$$x : T, y : T, p : \mathrm{Id}_T(x, y) \vdash R(x, y) \text{ TYPE}.$$

Then we know by reflexivity

$$x : T \vdash \rho_x : R(x, x).$$

Thus we can conclude from the Id elimination rule

$$x : T, y : T, p : \mathrm{Id}_T(x, y) \vdash j_\rho(x, y, p) : R(x, y),$$

as desired.

## 2.5 Axiom $K$ or identity reflection rule

We don't use this anymore, but it was used from the 70s-80s up until about 2000, so it's important to talk about it.

The rule says

$$\frac{\Gamma \vdash T \text{ TYPE} \qquad \Gamma \vdash s : T \qquad \Gamma \vdash t : T \qquad \Gamma \vdash p : \mathrm{Id}_T(s, t)}{\Gamma \vdash s = t : T \qquad \Gamma \vdash p = \mathrm{refl}_s : \mathrm{Id}_T(s, t)}$$

This rule is very bad, and it prevents us from doing anything useful in type theory.

We have two different notions of equality in type theory, sitting at two different levels. We have the internal equality, Id, and we have the equality judgement which always implies the Id equality.

People realized that this wasn't useful. It collapses the identity type to reflect the equality judgement. However, if we allow the identity type to have richer structure, by taking away this rule, we can capture a much broader segment of mathematics more naturally.

Without this rule, the type theory becomes naturally homotopical, whereas with this rule, we essentially have to build up homotopies from set theory and topological spaces. More on this in the future.