Type Theory with Paige North 7/10

Jason Schuchardt

July 12, 2019

1 Product Types

Generalize the function type.

1.1 In Sets:

For a one element set $1 = \{*\}$, and a set X we have

$$\{\text{elements of } X\} \longleftrightarrow \{\text{functions } 1 = \{*\} \to X\} \longleftrightarrow X \longleftrightarrow \prod_{x\in\{*\}} X.$$

For a two element set, 2, and a set X, we have

$$\{\text{ordered pairs of } X\} \longleftrightarrow \{\text{functions } 2 \to X\} \longleftrightarrow X \times X \longleftrightarrow \prod_{x\in 2} X.$$

Then

$$\{\text{sequences in } X\} \longleftrightarrow \{\text{functions } \mathbb{N} \to X\} \longleftrightarrow X^{\mathbb{N}} \longleftrightarrow \prod_{x\in\mathbb{N}} X.$$

One way to think of functions is as tuples. For functions $A \to X$, we can think of these as tuples in X whose entries are labeled elements of A.

If we have an indexed family $E(b)$ over B, then we can form

$$\prod_{b\in B} E(b),$$

the set of generalized tuples x, where $x_b \in E(b)$.

This is in bijection with sections of the map

$$\pi : \coprod_{b\in B} E(b) \to B.$$

A *section* of the map π is a map

$$s : B \to \coprod_{b\in B} E(b)$$

such that $\pi s = 1_B$. In other words, $s(b) \in \pi^{-1}(b)$ for all b.

This is the concept that we want to generalize in type theory. If E is trivially indexed by B, then

$$\prod_{b\in B} E \simeq \left\{\text{sections of } \coprod_{b\in B} E \simeq B \times E \to B\right\} \simeq \mathrm{Hom}(B, E).$$

This is the sense in which we are generalizing functions.

2 Π-types

We start with the rules for Π-types.

1. Π-formation
$$\frac{\Gamma \vdash B : U \qquad \Gamma, x : B \vdash E(x) : U}{\Gamma \vdash \prod_{x:B} E(x) : U}$$

2. Π-introduction
$$\frac{\Gamma, x : B \vdash e(x) : E(x)}{\Gamma \vdash \lambda x.e(x) : \prod_{x:B} E(x)}$$

3. Π-elimination
$$\frac{\Gamma \vdash f : \prod_{x:B} E(x) \qquad \Gamma \vdash b : B}{\Gamma \vdash fb : E(b)}$$

4. Π-computation
$$\frac{\Gamma, x : B \vdash e(x) : E(x) \qquad \Gamma \vdash b : B}{\Gamma \vdash (\lambda x.e(x))b = e(b) : E(b)}$$

5. Π-uniqueness
$$\frac{\Gamma \vdash f : \prod_{x:B} E(x)}{\Gamma \vdash \lambda x.fx = f : \prod_{x:B} E(x)}$$

2.1 ∧-types

We can form ∧-types out of Π-types in the exact same way as we form ∨-types from Σ-types.

We have
$$\vdash S_0 : U \qquad \vdash S_1 : U,$$
so
$$b : \mathbb{B} \vdash S(b) : U.$$
Then
$$\vdash \prod_{b:\mathbb{B}} S(b) : U.$$
Then if $\vdash s_0 : S_0$, $\vdash s_1 : S_1$, $b : \mathbb{B} \vdash s(b) : S(b)$, and we have
$$\vdash \lambda b.s(b) : \prod_{b:\mathbb{B}} S(b).$$
Lastly, we get the π_1, π_2 maps as
$$\frac{\vdash p : \prod_{b:\mathbb{B}} S(b)}{\vdash p0 : S(0) \qquad \vdash p1 : S(1)}$$

We'll denote $S_0 \wedge S_1$ by $S_0 \times S_1$ from now on, as we're a bit more interested in the set interpretation than the logic interpretation.

2.2 $\implies$-types

Starting with types $\vdash S : U$ and $\vdash T : U$, we have $x : S \vdash T : U$, and we can form

$$\vdash \prod_{x:S} T : U.$$

Then we have

$$\frac{x : S \vdash f(x) : T}{\vdash \lambda x.f(x) : \prod_{x:S} T}.$$

From now on, we'll write this as $\rightarrow$, since, once again, the set interepretation is somehow closer to what we're doing now.

2.3 Logic interpretation

To prove $\prod_{x:S} T(x)$, we have to prove $T(x)$ for all $x : S$. Looks like $\forall_{x:S} T(x)$. "For all $x \in S$, $T(x)$ holds."

As with the Σ-type/$\exists$ correspondence, this product $\forall$ type is somehow stronger than the logic $\forall$.

Life hack. Read Π as $\forall$, and Σ as $\exists$. This will make formulas in type theory make much more sense to you.

3 Returning to Id-types.

They form an equivalence relation. A relation on T is a dependent type $s : T, t : T \vdash R(s,t) : U$. We can think of a relation as a function $R : T \times T \rightarrow U$.

In type theory, and functional programming, it is often better to instead think about $R : T \rightarrow (T \rightarrow U)$. These two types are equivalent in a way that we can't really talk about yet.

We can define the type of relations on T.

$$\mathrm{Rel}(T) := T \rightarrow T \rightarrow U.$$

Note that we are not writing parentheses. Functions types are assumed to associate to the left. Then

$$T : U \vdash \mathrm{Rel}(T) = T \rightarrow T \rightarrow U : U.$$

Given $a : T$ and $b : T$, we have

$$\mathrm{Id}_T(a,b) : U.$$

This gives us a dependent type

$$x : T, y : T \vdash \mathrm{Id}_T(x,y) : U.$$

Lambda abstracting gives us

$$x : T \vdash \lambda y.\mathrm{Id}_T(x,y) : \prod_{y:T} U,$$

and again, we have

$$\vdash \lambda x.\lambda y.\mathrm{Id}_T(x,y) : \prod_{x:T}\prod_{y:T} U.$$

As mentioned before, we'll write this last type as $T \rightarrow T \rightarrow U = \mathrm{Rel}(T)$. We'll call this term $\mathrm{Id}_T := \lambda x.\lambda y.\mathrm{Id}_T(x,y)$.

How do we show that something is reflexive? We want to define a type isRefl so that the type being inhabited corresponds to a proof of reflexivity:

$$T : U, R : \mathrm{Rel}(T) \vdash \mathrm{isRefl}(R) : U$$

$$s, t : T \vdash R(s,t) : U$$

$$t : T \vdash \mathrm{refl}_t : R(t,t) \vdash \lambda t.\mathrm{refl}_t : \prod_{t:T} \mathrm{Rel}(t,t).$$

This last type should be our predicate isRefl:

$$\mathrm{isRefl}(R) := \prod_{t:T} R(t,t).$$

Then to prove that Id is reflexive, we have

$$T : U, t : T \vdash \lambda t.\mathrm{refl}_t : \prod_{t:T} \mathrm{Id}(t,t),$$

and this type is precisely isRefl(Id).

Now for symmetry, once again, we want to define a predicate type $\mathrm{isSym}(R)$. Translating from logic, it should be

$$\mathrm{isSym}(R) := \prod_{s,t:T} R(s,t) \to R(t,s).$$

Now let's prove that Id_T is symmetric. we start with $s, t : T, p : \mathrm{Id}_T(s,t) \vdash \mathrm{Id}_T(t,s) : U$. Now $t : T \vdash \mathrm{refl}_t : \mathrm{Id}_T(t,t)$, so by the elimination rule for the identity type,

$$s, t : T, p : \mathrm{Id}_T(s,t) \vdash j_{\mathrm{refl}_t}(s,t,p) : \mathrm{Id}_T(t,s) : U.$$

Then by lambda abstracting, we get

$$\vdash \lambda s, t, p.j_{\mathrm{refl}_t}(s,t,p) : \prod_{s,t:T} \prod_{p:\mathrm{Id}_T(s,t)} \mathrm{Id}_T(t,s) = \mathrm{isSym}(R).$$

Finally, we just need to prove transitivity. This time the type should be

$$\prod_{r,s,t:T} R(r,s) \to R(s,t) \to R(r,t).$$

Now we prove that the identity type is transitive. We start with $r, s, t : T$, $p : \mathrm{Id}_T(r,s)$, $q : \mathrm{Id}_T(s,t)$, and we want to get $\mathrm{Id}_T(r,t)$.

It suffices to prove

$$t : T, r, s : T, p : \mathrm{Id}_T(r,s) \vdash ? : \mathrm{Id}_T(s,t) \to \mathrm{Id}_T(r,t) : U.$$

$$t : T \vdash \lambda x.x : \mathrm{Id}_T(r,t) \to \mathrm{Id}_T(r,t),$$

so if we have

$$t : T, r, s : T, p : \mathrm{Id}_T(r,s) \vdash j_{\lambda x.x}(t,r,s,p) : \mathrm{Id}_T(s,t) \to \mathrm{Id}_T(r,t)$$

then we can lambda abstract, getting

$$\lambda r, s, t, p.j_{\lambda x.x}(t,r,s,p) : \prod_{r,s,t} \mathrm{Id}_T(r,s) \to \mathrm{Id}_T(s,t) \to \mathrm{Id}_T(r,t) = \mathrm{isTrans}(\mathrm{Id}_T).$$

Lastly, we can define an equivalence relation in the type theory, we can define

$$T : U, R : \mathrm{Rel}(T) \vdash \mathrm{isEquivRel}(R) := \mathrm{isRefl}(R) \times \mathrm{isSym}(R) \times \mathrm{isTrans}(R) : U.$$

Then we have

$$T : U \vdash ((r,s),t) : \mathrm{isEquivRel}(\mathrm{Id}_T).$$