Overview on Formalization - Type Theory part 2

George McNinch

2024-04-08

A derivation about renaming variables

We are going to derive the inference rule

$$\frac{\Gamma, x : A, \Delta \vdash \mathscr{J}}{\Gamma, x' : A, \Delta[x'/x] \vdash \mathscr{J}[x'/x]} x'/x$$

which essentially says "we an replace variable $x : A$ by variable $x' : A$".

The derivation uses *substitution*, *weakening* and the *generic element*:

$$\frac{\dfrac{\Gamma \vdash A \text{ type}}{\Gamma, x' : A \vdash x' : A}\delta \quad \dfrac{\Gamma \vdash A \text{ type} \quad \Gamma, x : A, \Delta \vdash \mathscr{J}}{\Gamma, x' : A, x : A, \Delta \vdash \mathscr{J}}W}{\Gamma, x' : A, \Delta[x'/x] \vdash \mathscr{J}[x'/x]} S$$

Dependent function types

Let b be a section of a family B over A in context Γ. Thus

$$\Gamma, x : A \vdash b(x) : B(x).$$

Two points of view here:

- can think of b as a *"program"* $x \mapsto b(x)$ that takes as input $x : A$ and produces a term $b(x) : B(x)$.
- or, can view $b(x)$ as just a "choice of element" in each $B(x)$ for $x : A$. i.e. $x \mapsto b(x)$ is a *dependent function*

The *type* of all such dependent functions is called the *dependent function type*, written:

$$\Pi_{(x:A)} B(x)$$

Part of what we must formulate in our type theory are inference rules guaranteeing that these types are well-formed. For this we need various sorts of *rules*:

- *formation rule* specifying how we may form dependent function types
- *introduction rule* specifying how to introduce new *terms* of dependent function types
- *elimination rule* specifying how to *use* arbitrary terms of dependent function types
- *computation rules* specifying how introduction rules and elimination rules interact

For the first three of these rules, we also need rules asserting that the constructions play nice with judgmental equality.

Π-formation rule

$$\frac{\Gamma, x : A \vdash B(x) \text{ type}}{\Gamma \vdash \Pi_{(x:A)} B(x) \text{ type}} \Pi$$

Also need the *congruence rule*:

$$\frac{\Gamma \vdash A \doteq A' \qquad \Gamma, x : A \vdash B(x) \doteq B'(x) \text{ type}}{\Gamma \vdash \Pi_{(x:A)} B(x) \doteq \Pi_{(x:A')} B'(x)} \ \Pi\text{-eq}$$

Π-introduction rule

This rule specifies that we may "construct" dependent functions provided that we have constructed a section:

$$\frac{\Gamma, x : A \vdash b(x) : B(x)}{\Gamma \vdash \lambda x.b(x) : \Pi_{(x:A)} B(x)} \ \lambda$$

We use the notation $\lambda x.b(x)$ for the "dependent function". This introduction rule is also called the λ-*abstraction rule*.

Again, we need to know that λ-abstraction respects judgmental equality:

$$\frac{\Gamma, x : A \vdash b(x) \doteq b'(x) : B(x)}{\Gamma \vdash \lambda x.b(x) \doteq \lambda x.b'(x) : \Pi_{(x:A)} B(x)} \ \lambda\text{-eq}$$

Π-elimination rule

The way to use a dependent function is to evaluate it at any an element of the domain type. The Π-elimination rule is thus sometimes called the *evaluation rule*.

$$\frac{\Gamma \vdash f : \Pi_{(x:A)} B(x)}{\Gamma, x : A \vdash f(x) : B(x)} \ \text{ev}$$

(Note that in practice – e.g. in type-theory based programming languages `Lean`, `Haskell`, `ML`, $\ldots$ – we write "$f\ x$" for "$f(x)$").

$$\frac{\Gamma \vdash f \doteq f' : \Pi_{(x:A)} B(x)}{\Gamma, x : A \vdash f(x) \doteq f'(x) : B(x)} \ \text{ev-eq}$$

Π-computation rule

We need to postulate rules controlling our functions.

The β-*rule* stipulates that $\lambda x.b(x)$ behaves in the way that we understand the function $x \mapsto b(x)$.

$$\frac{\Gamma, x : A \vdash b(x) : B(x)}{\Gamma, x : A \vdash (\lambda y.b(y))(x) \doteq b(x) : B(x)} \ \beta$$

The second rule – known as the η-*rule* – postulates that the *only* elements of $\Pi_{(x:A)} B(x)$ are dependent functions.

$$\frac{\Gamma \vdash f : \Pi_{(x:A)} B(x)}{\Gamma \vdash \lambda x.f(x) \doteq f : \Pi_{(x:A)} B(x)} \ \eta$$

Ordinary function types

We obtain ordinary functions as a special case of dependent functions. Let's describe the setting.

Suppose that A and B are types in context Γ. We can view B as the "constant family" $B(x) = B$ over $a : A$. From this point of view, we obtain the type of functions as

$$A \to B := \Pi_{(x:A)} B(x)$$

i.e.

$$A \to B := \Pi_{(x:A)} B$$

Here is the formal derivation:

$$\frac{\dfrac{\Gamma \vdash A \text{ type} \qquad \Gamma \vdash B \text{ type}}{\Gamma, x : A \vdash B \text{ type}} \ W}{\Gamma \vdash \Pi_{(x:A)} B \text{ type}} \ \Pi$$

And here is the formal declaration of the "new notation":

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash A \text{ type} \qquad \Gamma \vdash B \text{ type}}{\Gamma, x : A \vdash B \text{ type}}\, W}{\Gamma \vdash \Pi_{(x:A)} B \text{ type}}\, \Pi}{\Gamma \vdash A \to B := \Pi_{(x:A)} B}\,\text{defn}$$

Construction of the *identity function*

Given a type A in context Γ, let's construct the identity function $\text{id} = \text{id}_A : A \to A$ using the *generic term*:

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash A \text{ type}}{\Gamma, x : A \vdash x : A}\, \delta}{\Gamma \vdash \lambda x.x : A \to A}\, \lambda\text{-intro}}{\Gamma \vdash \text{id}_A := \lambda x.x : A \to A}\,\text{defn}$$

Construction of the composition of two functions

Given types A, B, C and terms $f : A \to B$, $g : B \to C$, we can form $g \circ f : A \to C$.

In fact, write $\text{comp}(g, f)$ for $g \circ f$. Then comp is in some sense a function of two arguments g and f. Let's pause to discuss functions of multiple arguments.

Consider a function

$$f : A \to (B \to C)$$

For an argument $a : A$, $f(a) : B \to C$ so $f(a)$ is again a function. For $b : B$, we can write

$$f(a)(b) \quad \text{or} \quad f(a, b) \quad \text{or} \quad f\ a\ b$$

for the value of $f(a)$ at $b : B$.

Now we see that the type of the function comp is

$$(B \to C) \to ((A \to B) \to (A \to C))$$

Thus for $g : B \to C$, we have $\text{comp}(g) : (A \to B) \to (A \to C)$.

We are going to define comp by the rule

$$\text{comp} = \lambda g.\lambda f.\lambda x.g(f(x))$$

.

Before we give the derivation, we need a preliminary result using the *generic element*; we'll refer to this as (♣) below:

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash A \text{ type} \qquad \Gamma \vdash B \text{ type}}{\Gamma \vdash A \to B \text{ type}}\, \Pi\text{-formation}}{\Gamma, f : A \to B \vdash f : A \to B}\, \delta}{\Gamma, f : A \to B, x : A \vdash f(x) : B}\, \Pi\text{-elimination}$$

Now here is the full derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash A \text{ type} \qquad \Gamma \vdash B \text{ type}}{\Gamma, f : A \to B, x : A \vdash f(x) : B}\, ♣}{\Gamma, g : B \to C, f : A \to B, x : A \vdash f(x) : B}\, \text{W} \qquad \dfrac{\dfrac{\dfrac{\Gamma \vdash B \text{ type} \qquad \Gamma \vdash C \text{ type}}{\Gamma, g : B \to C, y : B \vdash g(y) : C}\, ♣}{\Gamma, g : B \to C, f : A \to B, y : B \vdash g(y) : C}\, \text{W}}{\Gamma, g : B \to C, f : A \to B, x : A, y : B \vdash g(y) : C}\, \text{W}}{\Gamma, g : B \to C, f : A \to B, x : A \vdash g(f(x)) : C}\, \text{subst}}{\Gamma, g : B \to C, f : A \to B \vdash \lambda x.g(f(x)) : A \to C}\, \lambda\text{-intro}}{\Gamma, g : B \to C \vdash \lambda f.\lambda x.g(f(x)) : (A \to B) \to (A \to C)}\, \lambda\text{-intro}}{\Gamma \vdash \lambda g.\lambda f.\lambda x.g(f(x)) : (C \to B) \to ((A \to B) \to (A \to C))}\, \lambda\text{-intro}$$

One can now derive a number of properties of function composition:

- associativity, i.e.

$$\frac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash g : B \to C \qquad \Gamma \vdash h : C \to D}{\Gamma \vdash (h \circ g) \circ f \doteq h \circ (g \circ f) : A \to D}$$

- left and right unit laws

$$\frac{\Gamma \vdash f : A \to B}{\Gamma \vdash \mathrm{id}_B \circ f \doteq f : A \to B}$$

$$\frac{\Gamma \vdash f : A \to B}{\Gamma \vdash f \circ \mathrm{id}_A \doteq f : A \to B}$$

Bibliography

Bibliography