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

Any type family gives rise to a dependent function type:

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

Also need the *congruence rule*:

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

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:

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

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 `Lean`, `Haskell`, `ML`, … – we write "$f\ x$" for "$f(x)$").

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

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:

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

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

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

Construction of the *identity function*

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

-

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

foobar

Bibliography

Bibliography