Notes on univalent foundations with Coq

Type theory

N. Raghavendra

14 February, 2020

Abstract

This document is about some basic concepts of type theory. It is a part of a set of notes on univalent foundations with Coq.

Contents

1 Introduction 1
2 Universes 2
3 Pi types 3
4 Function types 5
5 Some basic functions 6
6 Composition of functions 7
7 Families as functions 8
8 Sigma types 9
9 The product of two types 10
10 The empty type 11
11 The unit type 11
12 The boolean type 12
13 The coproduct of two types 13
14 Natural numbers 14
15 Paths 15

1 Introduction

Type theory is a system of foundations for mathematics. There is only one primitive notion in this system, the concept of an *object*. We are allowed to make only one kind of statement about objects, that one object is an *element* of another. The theory provides certain objects, certain ways of constructing new objects from old ones, and certain statements about elements of these objects.

For any two objects x and X, we write

$$x : X$$

to indicate that x is an element of X.

We will assume that we know what it means to say that two things, such as two objects, are the *same*. The notation

$$x \equiv y$$

means that the objects x and y are the same.

2 Universes

Type theory begins by giving an infinite sequence

$$\mathcal{U}_0, \mathcal{U}_1, \ldots$$

of objects called *universes*. We will call the index i the *grade* of the universe $\mathcal{U}_i$.

We are given that the sequence of universes has the following two properties:

1. (*Hierarchy*) The object $\mathcal{U}_i$ is an element of $\mathcal{U}_{i+1}$ for every i. In the notation introduced above,

$$\mathcal{U}_i : \mathcal{U}_{i+1}.$$

2. (*Cumulativity*) Every element of $\mathcal{U}_i$ is an element of $\mathcal{U}_{i+1}$ also. In symbols, if $X : \mathcal{U}_i$, then $X : \mathcal{U}_{i+1}$.

Let us now fix one of the given universes $\mathcal{U}_i$, and denote it by $\mathcal{U}$. An element of this fixed universe $\mathcal{U}$ is called a *type*.

A *family* of types on a type

$$X : \mathcal{U}$$

is an assignment that attaches a type

$$Y_x : \mathcal{U}$$

to every element $x : X$. We denote this family by

$$(Y_x)_{x : X}.$$

The type X is called the *base* of the family. If it is obvious from the context, we may simply write

$$(Y_x)$$

for the family.

3 Pi types

Let us now look at a family of types

$$(Y_x)_{x:X}.$$

According to the above definition, this means that

$$X : \mathcal{U},$$

and we are given a type

$$Y_x : \mathcal{U}$$

for each element $x : X$.

A *selection* for this family is an assignment f that attaches an element

$$f(x) : Y_x$$

to every element $x : X$. The object $f(x)$ is called the *value* of f at x.

For every family

$$(Y_x)_{x:X},$$

we are given a type

$$\Pi(x : X).\, Y_x,$$

whose elements are exactly the selections for the family. This type is called the *Pi type* of the family.

The definition of a type implies that the Pi type is an element of $\mathcal{U}$:

$$\Pi(x : X).\, Y_x : \mathcal{U}.$$

We may just write

$$\Pi x.\, Y_x$$

for the Pi type if the base type X is obvious.

Suppose we are given an element

$$y_x : Y_x$$

for every element $x : X$. We then have a selection for the family, whose value at $x : X$ is y_x. We denote this selection by

$$\lambda(x : X).y_x.$$

A selection defined like this is called a *λ-abstraction*, or just an *abstraction*.

This definition means that

$$(\lambda(x : X).y_x)(a) \equiv y_a$$

for all $a : X$. We may just write

$$\lambda(x).y_x$$

for the abstraction if the base type is obvious.

If we start with a selection f for the family, we can take y_x to be $f(x)$ in the above description, and get another selection

$$\lambda(x : X).f(x).$$

Both f and this selection are assignments that attach the same value

$$f(x) : Y_x$$

to every element $x : X$. So the two selections are the same:

$$f \equiv \lambda(x : X).f(x).$$

This relation is called *η-conversion* for selections. The abstraction on its right hand side is called the *η-expansion* of f.

We will use the following compact notation for iterated Pi types. If

$$(Y_x)_{x : X}$$

is a family of types, and if for every $x : X$, we are given a family

$$(Z_{xy})_{y : Y_x},$$

we will denote

$$\Pi(x : X).\,(\Pi(y : Y_x).\,Z_{xy})$$

by

$$\Pi(x : X)(y : Y_x).\,Z_{xy}.$$

If we are given an element

$$z_{xy} : Z_{xy}$$

for every $x : X$ and $y : Y_x$, we will denote the selection

$$\lambda(x : X).(\lambda(y : Y_x).z_{xy})$$

by

$$\lambda(x : X)(y : Y_x).z_{xy}.$$

We will drop the types X and Y_x from the notation if they are obvious, and simply write

$$\Pi xy.\,Z_{xy}$$

and

$$\lambda xy.z_{xy}.$$

We will use the same kind of notation even when there are more than two indices.

We will also use the convention that the scope of the Pi operator in any expression is the entire right of the expression unless stopped by parentheses. So an expression like

$$\Pi(y : X).\, x = y$$

means

$$\Pi(y : X).\, (x = y),$$

not

$$(\Pi(y : X).\, x) = y.$$

4 Function types

Any two types X and Y give us a family of types

$$(Y_x)_{x\,:\,X}$$

on X, where

$$Y_x \equiv Y$$

for all $x : X$. We call this the *constant family* on X with *value* Y.

A selection for this family is called a *function* from X to Y. It is an assignment f that attaches an element

$$f(x) : Y$$

to every element $x : X$.

The Pi type of the family is denoted by

$$X \to Y.$$

It is called the *function type* of X and Y. It is an element of the fixed universe $\mathcal{U}$. Its elements are exactly the functions from X to Y.

For any function

$$f : X \to Y,$$

the type X is called the *domain* of f and is denoted by $\mathrm{Dom}(f)$, while Y is called the *codomain* of f and is denoted by $\mathrm{Cod}(f)$.

Given an element

$$y_x : Y$$

for every element $x : X$, we have the abstraction

$$\lambda(x : X).y_x : X \to Y,$$

which is a function with the property that

$$(\lambda(x : X).y_x)(a) \equiv y_a$$

for all $a : X$.

Conversely, every function

$$f : X \to Y$$

is the same as its abstraction:

$$f \equiv \lambda(x : X).f(x).$$

This fact is called *η-conversion* for functions.

5 Some basic functions

We will see many examples of selections and functions as we go on. For now, we will describe just a couple of functions that we need immediately.

The *identity function* of a type X is a standard example of a function. It is the function

$$\mathrm{id}_X : X \to X$$

defined by

$$\mathrm{id}_X(x) \equiv x$$

for all $x : X$. In terms of abstractions,

$$\mathrm{id}_X \equiv \lambda(x : X).x.$$

Another class of examples of a function are the constant functions. Suppose X and Y are types, and y an element of Y. We then have a function

$$c_y : X \to Y$$

defined by

$$c_y \equiv \lambda(x : X).y.$$

It is called the *constant function* on X with value y. As y runs over Y, we get an element

$$c : Y \to X \to Y$$

given by

$$c \equiv \lambda(y : Y).c_y.$$

We are following the convention that the arrow operator is right associative. So

$$Y \to X \to Y$$

means

$$Y \to (X \to Y),$$

not

$$(Y \to X) \to Y.$$

6 Composition of functions

The *composite* of two functions

$$f : X \to Y, \quad g : X \to Z$$

is the function

$$g \circ f : X \to Z$$

defined by

$$g \circ f \equiv \lambda(x : X).g(f(x)).$$

When we vary f and g, we get an element

$$\phi : (X \to Y) \to (Y \to Z) \to X \to Z$$

given by

$$\phi \equiv \lambda(f : X \to Y)(g : Y \to Z).g \circ f.$$

We are using the convention that the scope of the lambda operator, as with the Pi operator, is the entire right of the expression unless stopped by parentheses. So

$$\lambda(f : X \to Y)(g : Y \to Z).g \circ f$$

means

$$\lambda(f : X \to Y)(g : Y \to Z).(g \circ f),$$

not

$$(\lambda(f : X \to Y)(g : Y \to Z).g) \circ f.$$

The definition of composition implies that for any three functions

$$f : X \to Y, \quad g : X \to Z, \quad h : Z \to W,$$

we have

$$((h \circ g) \circ f)(x) \equiv (h \circ (g \circ f))(x) \equiv h(g(f(x)))$$

for all $x : X$. So by η-conversion we get the *associativity law*

$$(h \circ g) \circ f \equiv h \circ (g \circ f).$$

The same kind of verification gives the *identity law*

$$f \circ \mathrm{id}_X \equiv \mathrm{id}_Y \circ f \equiv f$$

for every function $f : X \to Y$.

7 Families as functions

We have defined a family of types

$$(Y_x)_{x:X}$$

to be an assignment that attaches to every element $x : X$ a type Y_x, that is to say an element Y_x of the fixed universe $\mathcal{U}$. According to the definition of a function in Section 4 above, this is just a function from X to $\mathcal{U}$. The value Y_x of this function at any $x : X$ is the same as the value of the abstraction

$$\lambda(x : X).Y_x$$

at x. So by η-conversion we have

$$(Y_x)_{x:X} \equiv \lambda(x : X).Y_x.$$

From now we will view a family of types on a type X as a function

$$F : X \to \mathcal{U}.$$

This family would be written as

$$(F(x))_{x:X}$$

in the earlier notation.

One point that has been fudged above is that $\mathcal{U}$ is not a type according to the definition that a type means an element of $\mathcal{U}$. The statement

$$\mathcal{U} : \mathcal{U}$$

leads to a contradiction called Girard's paradox. So we have not yet defined the notion of a function from X to $\mathcal{U}$.

To fill this gap suppose that $\mathcal{U} \equiv \mathcal{U}_i$, and denote $\mathcal{U}_{i+1}$ by $\mathcal{U}_+$. The hierarchy property of universes implies that $\mathcal{U} : \mathcal{U}_+$. Everything we have said until now will be true if we take $\mathcal{U}_+$ to be our fixed universe instead of $\mathcal{U}$. But then $\mathcal{U}$ would be a type because it is an element of the fixed universe $\mathcal{U}_+$. The cumulativity of universes guarantees that the element X of $\mathcal{U}$ is an element of $\mathcal{U}_+$, so it is also a type with respect to $\mathcal{U}_+$. The notion of a function from X to $\mathcal{U}$ is now given by the definition, in Section 4, of a function from one type to another.

8 Sigma types

A *pair* for a family

$$F : X \to \mathcal{U}$$

is an object t that consists of two other objects: an element $x : X$ called the *first component* of t, and an element $y : F(x)$ called the *second component* of t. Such a pair t is denoted by

$$(x, y).$$

We are given a type

$$\Sigma(x : X).\, F(x),$$

whose elements are exactly the pairs for the family F. It is called the *Sigma type* of the family. By the convention that a type means an element of the fixed universe, the Sigma type is an element of $\mathcal{U}$. We may simply write

$$\Sigma x.\, F(x)$$

for the Sigma type if the base type X is obvious.

We have a function

$$\mathrm{pr}_1 : \Sigma(x : X).\, F(x) \to X$$

defined by

$$\mathrm{pr}_1(x, y) \equiv y$$

for all $x : X$ and $y : F(x)$. This function is called the *first projection*.

We also have a selection

$$\mathrm{pr}_2 : \Pi(t : \Sigma(x : X).\, F(x)).\, F(\mathrm{pr}_1(t))$$

called the *second projection*. It is defined by

$$\mathrm{pr}_2(x, y) \equiv y$$

for all $x : X$ and $y : F(x)$.

In the other direction, for every $a : X$ there is a function

$$\sigma_a : F(a) \to \Sigma(x : X).\, F(x)$$

defined by

$$\sigma_a(y) \equiv (a, y)$$

for $y : F(a)$. It is called the *canonical function* from $F(a)$ to the Sigma type.

The definitions of the two projections imply that

$$(x, y) \equiv (\mathrm{pr}_1(x, y), \mathrm{pr}_2(x, y))$$

for every pair (x, y) for the family F. In other words,

$$t \equiv (\mathrm{pr}_1(t), \mathrm{pr}_2(t))$$

for every element t of $\Sigma(x : X).\, F(x)$. This relation is called *η-conversion* for Sigma types.

As in the case of Pi types, we use a shortened notation for iterated Sigma types. If

$$F : X \to \mathcal{U}$$

is a family of types, and if for every $x : X$, we are given a family

$$G_x : F(x) \to \mathcal{U},$$

we will denote

$$\Sigma(x : X).\, (\Sigma(y : F(x)).\, G_x(y))$$

by

$$\Sigma(x : X)(y : F(x)).\, G_x(y).$$

We use a similar notation for iterated Sigma types with more than two indices.

9 The product of two types

Just as function types are the Pi types of constant families, product types are the Sigma types of constant families.

Suppose X and Y are two types. We can then take the family in the description of Sigma types in Section 8 to be the constant family on X with value Y. We denote the Sigma type of this family by $X \times Y$:

$$X \times Y \equiv \Sigma(x : X).\, Y.$$

It is called the *product* of X and Y. The elements of $X \times Y$ are exactly the pairs for the family. These are objects of the form

$$(x, y),$$

where $x : X$ and $y : Y$.

We have the projections

$$\mathrm{pr}_1 : X \times Y \to X, \quad \mathrm{pr}_2 : X \times Y \to Y.$$

They are defined by

$$\mathrm{pr}_1(x, y) \equiv x, \quad \mathrm{pr}_2(x, y) \equiv y$$

for all $x : X$ and $y : Y$.

For any $a : X$ and $b : Y$, there are functions

$$\sigma_a : Y \to X \times Y, \quad \tau_b : X \to X \times Y$$

defined by

$$\sigma_a(y) \equiv (a, y), \quad \tau_b(x) \equiv (x, b)$$

for all $x : X$ and $y : Y$. As we vary a and b, these functions give rise to functions

$$\sigma : X \to Y \to X \times Y, \quad \tau : Y \to X \to X \times Y.$$

These two latter functions are called the *pairing* functions.

The η-conversion for Sigma types implies that

$$t \equiv (\mathrm{pr}_1(t), \mathrm{pr}_2(t))$$

for all $t : X \times Y$. This fact is called *η-conversion* for product types.

10 The empty type

We are given a type $\mathbf{0}$ called the *empty type*. We are also given for every family

$$F : \mathbf{0} \to \mathcal{U}$$

on $\mathbf{0}$, a selection ϕ_F for F, called the *canonical selection* for F.

Now if X is any type, we can take F to be the constant family with domain $\mathbf{0}$ and value X. We then get a function

$$\phi_X : \mathbf{0} \to X,$$

which is called the *canonical function* from $\mathbf{0}$ to X.

11 The unit type

We are given a type $\mathbf{1}$ called the *unit type*, and an element

$$\star : \mathbf{1}$$

called the *canonical element* of $\mathbf{1}$.

For every family

$$F : \mathbf{1} \to \mathcal{U}$$

on $\mathbf{1}$, and for every element

$$a : F(\star),$$

we are given a selection $\phi_{F,a}$ for F with the property that

$$\phi_{F,a}(\star) \equiv a.$$

We call $\phi_{F,a}$ the selection for F *induced* by the element a.

Taking F to be a constant family, for every type

$$X : \mathcal{U},$$

and every element

$$a : X,$$

we get a function

$$\phi_a : \mathbf{1} \to X$$

such that

$$\phi_a(\star) \equiv a.$$

It is called the function from $\mathbf{1}$ to X *induced* by a.

We also have a function

$$\psi_X : X \to \mathbf{1}$$

defined by

$$\psi_X(x) \equiv \star$$

for all $x : X$. It is called the *canonical function* from X to $\mathbf{1}$.

12 The boolean type

We are given a type $\mathbf{B}$ called the *boolean type*, and two elements

$$\mathbf{t} : \mathbf{B}, \quad \mathbf{f} : \mathbf{B},$$

which are not the same. These two elements are called the *canonical elements* of $\mathbf{B}$.

For every family

$$F : \mathbf{B} \to \mathcal{U},$$

and elements

$$a : F(\mathbf{t}), \quad b : F(\mathbf{f}),$$

we are given a selection $\phi_{F,a,b}$ for F with the property that

$$\phi_{F,a,b}(\mathbf{t}) \equiv a, \quad \phi_{F,a,b}(\mathbf{f}) \equiv b.$$

It is called the selection for F *induced* by a and b.

If we assume that F is a constant type, for every type

$$X : \mathcal{U},$$

and elements

$$a : X, \quad b : X,$$

we get a function

$$\phi_{a,b} : \mathbf{B} \to X$$

such that

$$\phi_{a,b}(\mathbf{t}) \equiv a, \quad \phi_{a,b}(\mathbf{f}) \equiv b.$$

We again call $\phi_{a,b}$ the function from $\mathbf{B}$ to X *induced* by a and b.

13 The coproduct of two types

Suppose X and Y are types. We are then given a type

$$X \sqcup Y.$$

We are also given two functions

$$\mathrm{i}_1 : X \to X \sqcup Y, \quad \mathrm{i}_2 : Y \to X \sqcup Y.$$

They are called the *canonical functions* from X and Y to $X \sqcup Y$. Objects of the form $\mathrm{i}_1(x)$ or $\mathrm{i}_2(y)$, where $x : X$ and $y : Y$, are called the *canonical elements* of $X \sqcup Y$.

For every family

$$F : X \sqcup Y \to \mathcal{U},$$

every selection f for the family

$$F \circ \mathrm{i}_1 : X \to \mathcal{U},$$

and every selection g for the family

$$F \circ \mathrm{i}_2 : Y \to \mathcal{U},$$

we are given a selection $\phi_{F,f,g}$ for F with the property that

$$\phi_{F,f,g}(\mathrm{i}_1(x)) \equiv f(x), \quad \phi_{F,f,g}(\mathrm{i}_2(y)) \equiv g(y)$$

for all $x : X$ and $y : Y$. We call $\phi_{F,f,g}$ the selection for F *induced* by f and g.

Now take F to be the constant family on $X \sqcup Y$ with value Z, where Z is any type. Consider any two functions

$$f : X \to Z, \quad g : Y \to Z.$$

The above provision then gives us a function

$$\phi_{f,g} : X \sqcup Y \to Z$$

such that

$$\phi_{f,g}(\mathrm{i}_1(x)) \equiv f(x), \quad \phi_{f,g}(\mathrm{i}_2(y)) \equiv g(y)$$

for all $x : X$ and $y : Y$. It is called the function from $X \sqcup Y$ to Z *induced* by f and g.

14 Natural numbers

We are given a type $\mathbf{N}$, an element

$$\mathrm{O} : \mathbf{N},$$

and a function

$$\mathrm{S} : \mathbf{N} \to \mathbf{N}.$$

The elements of $\mathbf{N}$ are called the *natural numbers*. The natural number O is called *zero*. For any $n : \mathbf{N}$, the natural number $\mathrm{S}(n)$ is called the *successor* of n.

For every family

$$F : \mathbf{N} \to \mathscr{U},$$

every element

$$a : F(\mathrm{O}),$$

and every element

$$f : \Pi(n : \mathbf{N}).\, F(n) \to F(\mathrm{S}(n)),$$

we are given a selection $\phi_{F,a,f}$ for F such that

$$\phi_{F,a,f}(\mathrm{O}) \equiv a,$$

and

$$\phi_{F,a,f}(\mathrm{S}(n)) \equiv f(n)(\phi_{F,a,f}(n))$$

for all $n : \mathbf{N}$. We call $\phi_{F,a,f}$ the selection for F *induced* by a and f.

When we specialise this dispensation to a constant family, for every type

$$X : \mathscr{U},$$

every element

$$a : X,$$

and every function

$$f : \mathbf{N} \to X \to X,$$

we get a function

$$\phi_{a,f} : \mathbf{N} \to X$$

such that

$$\phi_{a,f}(\mathrm{O}) \equiv a,$$

and

$$\phi_{a,f}(\mathrm{S}(n)) \equiv f(n)(\phi_{a,f}(n))$$

for all $n : \mathbf{N}$. It is called the function from $\mathbf{N}$ to X *induced* by a and f.

15 Paths

Let us fix a type X, and an element $x : X$. We are then given a family of types

$$\mathrm{Path}_x : X \to \mathcal{U}$$

on X, called the *path family* of x. For any element $y : X$, we will denote the type

$$\mathrm{Path}_x(y) : \mathcal{U}$$

by

$$x = y.$$

Each of its elements is called a *path* from x to y.

An element of the type

$$x = x$$

is called a *loop* at x. We are given a loop

$$\epsilon_x : x = x.$$

It is called the *identity loop* at x.

For every element

$$F : \Pi(y : X).\, x = y \to \mathcal{U},$$

and for every element

$$a : F(x)(\epsilon_x),$$

we are given an element

$$\phi_{F,a} : \Pi(y : X)(p : x = y).\, F(y)(p)$$

such that

$$\phi_{F,a}(x)(\epsilon_x) \equiv a.$$

We say that $\phi_{F,a}(y)(p)$ is obtained by *transporting* a along the path p from x to y.

We are using here the convention that the path operator has higher precedence than the arrow. So

$$x = y \to \mathcal{U}$$

means

$$(x = y) \to \mathcal{U},$$

not

$$x = (y \to \mathcal{U}).$$

We are also ignoring the problem that the object

$$\Pi(y : X).\, x = y \to \mathcal{U}$$

is not defined as $\mathcal{U}$ is not an element of $\mathcal{U}$. However, if $\mathcal{U} \equiv \mathcal{U}_i$, and if we redo the earlier development with $\mathcal{U}_+$ in the place of $\mathcal{U}$, where $\mathcal{U}_+ \equiv \mathcal{U}_{i+1}$, then

$$\Pi(y : X).\, x = y \to \mathcal{U}$$

becomes a well-defined type for every $y : X$. This argument is like the one in Section 7 that justified the practice of looking at families as functions. We will skip such justification from now.

We can specialise the above discussion to the situation when

$$F(y)(p) \equiv G(y)$$

is independent of p. So for every family of types

$$G : X \to \mathcal{U},$$

and every element

$$a : G(x),$$

we get an element

$$\phi_{G,a} : \Pi(y : X).\, x = y \to G(y)$$

such that

$$\phi_{G,a}(x)(\epsilon_x) \equiv a.$$

Again $\phi_{G,a}(y)(p)$ is called the element obtained by *transporting* a along p.

Let us lastly look at the even more special case when G is a constant family. For every type

$$H : \mathcal{U},$$

and for every element

$$a : H,$$

we get an element

$$\phi_{H,a} : \Pi(y : X).\, x = y \to H$$

such that

$$\phi_{H,e}(x)(\epsilon_x) \equiv a.$$

If p is a path from x to y, we say then too that $\phi_{H,a}(y)(p)$ is obtained by *transporting* a along p.