

# **Dependent Type Theory**

Summary note of Angiuli & Gratzer's Dependent Type Theory.

**Hojune Lee**

School of Computing, KAIST

This is my summary note of KAIST CS520 Graduated PL course of 2025 Fall. Instructor is Prof. Hongseok Yang, and all the materials are based on textbook by Angiuli & Gratzer's Dependent Type Theory.

# Contents

<b>1</b>	<b>Context and Substitutions</b>	<b>4</b>
1.1	Intuitions . . . . .	4
1.2	Contexts Judgements . . . . .	6
1.3	Substitution Judgements . . . . .	6
<b>2</b>	<b>Internalizing Judgemental Structure : <math>\Pi, \Sigma, \text{Eq}, \text{Unit}</math></b>	<b>14</b>
2.1	Basic Intuitions . . . . .	14
2.2	Dependent Sums . . . . .	15
2.3	Dependent Products . . . . .	20
2.4	Extensional Equality . . . . .	25
2.5	Unit Type . . . . .	27
<b>3</b>	<b>Inductive Types : Void, Bool, Nat</b>	<b>28</b>
3.1	Basic Intuitions . . . . .	28
3.2	Void : The empty type . . . . .	29
3.3	Booleans . . . . .	36
3.4	Natural Number . . . . .	41
<b>4</b>	<b>Further topics of Chapter 3</b>	<b>46</b>
4.1	Uniqueness and Extensional Equality . . . . .	46
4.2	Large Elimination . . . . .	49

# Chapter 1

## Context and Substitutions

### 1.1 Intuitions

In simple type theory, every thing was quite clear. This is because that, if we once fix the rules for types in simple type theory, then all possible types are determined immediately by inductive way. We can informally think this as ‘propositional logic’-like type construction. However, many things are different in dependent type system. Let’s think dependent type system as ‘first order logic’-like type construction. Imagine the formulas in first order logic as types in dependent type theory. I hope that this is proper intuitive thinking for the main differences between two type systems.

Once we imagine FOL formula-style type, it means that each type can contain variables and constants as its representation. It means that for each variables which are appeared in type representation, we must know what are the types of each variables. That is Context’s role.

Now, we need to construct the intuition for contexts and substitutions. Contexts are ‘wolrds’ of terms and types. And substitutions are ‘traffic way’ between each worlds( contexts ). Then naturally, following questions are arising.

#### Questions

1. Is this world ( Context ) well-formed ?
2. In this world ( Context )  $\Gamma$ , what terms of type  $A$  are well-formed ?
3. In this world ( Context )  $\Gamma$ , what types are well-formed ?
4. Between 2 worlds ( Context )  $\Delta$  and  $\Gamma$ , is traffic way ( substitution )  $\gamma$  well-formed ?
5. In this world ( Context ), which terms/types/substitutions are equivalent?

Actually, this is all about ‘judgement rules’ referred in Notation 2.3.1. of textbook. Formally, we can write format for above judgements.

## Formal Representations for Judgements

1.  $\vdash \Gamma \text{ cx}$
2.  $\Gamma \vdash a : A$
3.  $\Gamma \vdash A \text{ type}$
4.  $\Delta \vdash \gamma : \Gamma$
5.  $\Gamma \vdash a = a' : A, \quad \Gamma \vdash A = A' \text{ type}, \quad \Delta \vdash \gamma = \gamma' : \Gamma$

In this chapter, we'll discuss the rules for above judgements. Now, it's time to define explicit judgement rules that we referred.

## 1.2 Contexts Judgements

Construction 1.2.1.

$$\frac{}{\vdash \mathbf{1} \text{ cx}} \quad \frac{\vdash \Gamma \text{ cx} \quad \Gamma \vdash A \text{ type}}{\vdash \Gamma.A \text{ cx}}$$

Here,  $\mathbf{1}$  is empty context and each context is just list of types. (No variable names) It means that we use De Bruijn index, i.e. in each context, index automatically determines the variable in context.

## 1.3 Substitution Judgements

However, before that the most important one is understanding direction of traffic way (from now on, substitution).

Definition 1.3.1.

If  $\Delta \vdash \gamma : \Gamma$ , then  $\gamma$  is substitution from  $\Delta$  to  $\Gamma$ . i.e.

$$\gamma : \Delta \rightarrow \Gamma$$

However, it's role is send types and terms of  $\Gamma$  into  $\Delta$ . (Note: Direction is important)

### 1.3. Intuitions for direction of substitutions

To understand why here use this notation, see following example. Imagine that contexts are ‘sets’ and substitutions are function between them. There are 2 sets  $\Delta, \Gamma$  and mapping  $\gamma : \Delta \rightarrow \Gamma$ . Suppose that there is a function  $g : \Gamma \rightarrow \mathbb{R}$ , which is defined on set  $\Gamma$ . How can we ‘use’ this function in  $\Delta$  set? One way is that,

$$g : \Gamma \rightarrow \mathbb{R} \implies g \circ \gamma : \Delta \rightarrow \mathbb{R}$$

Then we can ‘use’ the function  $g$  in domain  $\Delta$ . This exactly corresponds in our notation. For  $\Delta \vdash \gamma : \Gamma$ , when we define  $\gamma : \Delta \rightarrow \Gamma$ ,

$$\Gamma \vdash A \text{ type} \implies \Delta \vdash A[\gamma] \text{ type}$$

Since  $A$  is type in  $\Gamma$  and the direction of substitution is  $\gamma : \Delta \rightarrow \Gamma$ , we say that  $A[\gamma]$  is pull-backed type of  $A$  through  $\gamma : \Delta \rightarrow \Gamma$ .

First, as above we can easily define the application rule of substitution.

Construction 1.3.2.

$$\frac{\Delta \vdash \gamma : \Gamma \quad \Gamma \vdash A \text{ type}}{\Delta \vdash A[\gamma] \text{ type}} \quad \frac{\Delta \vdash \gamma : \Gamma \quad \Gamma \vdash a : A}{\Delta \vdash a[\gamma] : A[\gamma]}$$

These rules give us that we can immigrate(pull-back) terms and types of  $\Gamma$  into  $\Delta$ . And now, in our setting each contexts are different ‘worlds’. So we need to introduce explicit **weakening rule**, which intuitively means that we can bring well-formed types and terms into  $\Gamma$  into  $\Gamma.A$ , expanded context.

Construction 1.3.3.

$$\frac{\Gamma \vdash A \text{ type}}{\Gamma.A \vdash \mathbf{p} : \Gamma}$$

In diagram,

$$\Gamma.A \xrightarrow{p} \Gamma$$

Means that, we can pull-back types  $B$  and terms  $t$  in world  $\Gamma$  into  $\Gamma.A$  by write  $B[\mathbf{p}], t[\mathbf{p}]$   
Moreover, we can introduce following rules. (Actually, following rules need because our context-substitution system become a category.)

Construction 1.3.4.

$$\frac{\vdash \Gamma \text{ cx}}{\Gamma \vdash \mathbf{id} : \Gamma} \quad \frac{\Gamma_2 \vdash \gamma_1 : \Gamma_1 \quad \Gamma_1 \vdash \gamma_0 : \Gamma_0}{\Gamma_2 \vdash \gamma_0 \circ \gamma_1 : \Gamma_0}$$

Second one can be conflict when we see first. However, let's draw the diagram.

$$\begin{array}{ccc} \Gamma_2 & \xrightarrow{\gamma_1} & \Gamma_1 \\ \downarrow \gamma_0 \circ \gamma_1 & \nearrow \gamma_0 & \\ \Gamma_0 & & \end{array}$$

Then, above representation is very clear. It become same notation in our function calculus. Similarly,

Construction 1.3.5.

$$\frac{\Delta \vdash \gamma : \Gamma}{\Delta \vdash \gamma \circ \mathbf{id} = \mathbf{id} \circ \gamma = \gamma : \Gamma} \quad \frac{\Gamma_3 \vdash \gamma_2 : \Gamma_2 \quad \Gamma_2 \vdash \gamma_1 : \Gamma_1 \quad \Gamma_1 \vdash \gamma_0 : \Gamma_0}{\Gamma_3 \vdash (\gamma_0 \circ \gamma_1) \circ \gamma_2 = \gamma_0 \circ (\gamma_1 \circ \gamma_2) : \Gamma_0}$$

This two rules are also very clear when we see the diagram.

$$id \subset \Delta \xrightarrow{\gamma} \Gamma \supset id$$

$$\begin{array}{ccc}
 \Gamma_3 & \xrightarrow{\gamma_2} & \Gamma_2 \\
 (\gamma_0 \circ \gamma_1) \circ \gamma_2 \downarrow & \swarrow \gamma_0 \circ \gamma_1 & \downarrow \gamma_1 \\
 \Gamma_0 & \xleftarrow{\gamma_0} & \Gamma_1
 \end{array} \iff
 \begin{array}{ccc}
 \Gamma_3 & \xrightarrow{\gamma_2} & \Gamma_2 \\
 \gamma_0 \circ (\gamma_1 \circ \gamma_2) \downarrow & \searrow \gamma_1 \circ \gamma_2 & \downarrow \gamma_1 \\
 \Gamma_0 & \xleftarrow{\gamma_0} & \Gamma_1
 \end{array}$$

Then we can imagine following equivalent rules clearly.

Construction 1.3.6.

$$\frac{\Gamma \vdash A \text{ type}}{\Gamma \vdash A[\mathbf{id}] = A \text{ type}} \quad \frac{\Gamma \vdash a : A}{\Gamma \vdash a[\mathbf{id}] = a : A}$$

$$\frac{\Gamma_2 \vdash \gamma_1 : \Gamma_1 \quad \Gamma_1 \vdash \gamma_0 : \Gamma_0 \quad \Gamma_0 \vdash A \text{ type}}{\Gamma_2 \vdash A[\gamma_0 \circ \gamma_1] = A[\gamma_0][\gamma_1] \text{ type}}$$

$$\frac{\Gamma_2 \vdash \gamma_1 : \Gamma_1 \quad \Gamma_1 \vdash \gamma_0 : \Gamma_0 \quad \Gamma_0 \vdash a : A}{\Gamma_2 \vdash a[\gamma_0 \circ \gamma_1] = a[\gamma_0][\gamma_1] : A[\gamma_0 \circ \gamma_1]}$$

These rules are very intuitive. When we draw diagram,

$$\begin{array}{ccc}
 \Gamma_2 & \xrightarrow{\gamma_1} & \Gamma_1 \\
 \gamma_0 \circ \gamma_1 \downarrow & \swarrow \gamma_0 & \\
 \Gamma_0 & &
 \end{array}$$

For example, look the last rule.  $a$  is term in the world  $\Gamma_0$ . We want to bring this term into  $\Gamma_2$ . Bring means that, we want to use ‘function’  $a$  in  $\Gamma_2$ . i.e. such term in  $\Gamma_2$  is working same roles with  $a$  in  $\Gamma_0$ . We’ve studied in box 1.3, there are 2 ways to bring  $a$  from  $\Gamma_0$  into  $\Gamma_2$ .

$$\begin{array}{ccc}
 \Gamma_2 & \xrightarrow{\gamma_1} & \Gamma_1 \\
 \gamma_0 \circ \gamma_1 \downarrow & \swarrow \gamma_0 & \\
 \Gamma_0 & &
 \end{array}$$

One is pulling back via red way, and another one is via blue way.  $a$  via red way is  $a[\gamma_0 \circ \gamma_1]$  in  $\Gamma_2$ , and via blue way is  $a[\gamma_0][\gamma_1]$  (Actually,  $a[\gamma_0]$  is  $a$  in  $\Gamma_1$  world. So we take it again into  $\Gamma_2$ .) However, how can we use variable in well-formed context  $\Gamma$  without own names? What is the meaning of De Bruijn index? Let’s see how can we distinguish variables in each ‘world’  $\Gamma$ .

Construction 1.3.7.

$$\frac{\Gamma \vdash A \text{ type}}{\Gamma.A \vdash \mathbf{q} : A[\mathbf{p}]}$$

We can see context as a stack.  $\mathbf{q}$  always means the top element of stack. However, stack  $\Gamma.A$  contains type  $A$  as top, which is type in  $\Gamma$  world. So the top variable's type in  $\Gamma.A$  is  $A[\mathbf{p}]$  which we take  $A$  from  $\Gamma$  into  $\Gamma.A$ . Similarly, we can access the any index of stack by following rule which is derivable from 1.3

Corollary 1.3.8.

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma.A \vdash B_1 \text{ type} \quad \cdots \Gamma.A.B_1 \cdots \vdash B_n \text{ type}}{\Gamma.A.B_1 \cdots .B_n \vdash \mathbf{q}[\mathbf{p}^n] : A[\mathbf{p}^{n+1}]}$$

Now, it's time to discuss our main theme in this subsection 1.3. Our ultimate goal is defining ‘valid traffic way from valid world to another valid world’. Let's discuss substitution judgement rules very intuitively. First, suppose that we only have rules for context judgements 1.2. The obvious thing that we must do first is, construct way from any  $\Gamma$  into  $\mathbf{1}$ . The rules are simple.

Construction 1.3.9.

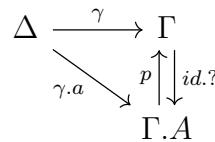
$$\frac{\vdash \Gamma \text{ cx}}{\Gamma \vdash ! : \mathbf{1}} \quad \frac{\Gamma \vdash \delta : \mathbf{1}}{\Gamma \vdash \delta =! : \mathbf{1}}$$

What does it mean? It is ‘way’ from  $\Gamma$  to  $\mathbf{1}$ . The role of this way is transfer terms and types in  $\mathbf{1}$  into  $\Gamma$  world. What terms and types are well-formed in  $\mathbf{1}$ ? Just closed terms and types (No Variables). It is obviously terms and types in  $\Gamma$  also. So we define the way is ‘unique’ here. With this substitution, we can always transfer closed terms(e.g. 1,  $(1+1)$ ) and closed types from  $\mathbf{1}$  to any well-formed context  $\Gamma$ . Let's think this as a base case of our definition of traffic ways. Following construction determines how can we extends substitutions.

Construction 1.3.10.

$$\frac{\Delta \vdash \gamma : \Gamma \quad \Gamma \vdash A \text{ type} \quad \Delta \vdash a : A[\gamma]}{\Delta \vdash \gamma.a : \Gamma.A}$$

Let's draw diagram.



Is it clear? It means that, suppose that  $\Gamma$  has  $n$  variables  $(x_1, \dots, x_n)$ . Here we denote names for each variables for representation. And then, in this context  $\Gamma$ ,  $A$  is well-formed type. Then automatically  $\Gamma.A$  is defined, where it has  $n + 1$  variables  $(x_1, \dots, x_n, x_{n+1})$ .

The key understanding is that, when we bring terms and types in  $\Gamma.A$  world, it can have  $x_1, \dots, x_{n+1}$  as symbol. When we meet  $x_{n+1}$ , then substitute it by  $a$ , and when we meet  $x_1, \dots, x_n$ , then substitute it by following  $\gamma$ . Then since  $a$  is well-formed term in  $\Delta$  and its type is  $A[\gamma]$ , which do roles of  $A$  in  $\Delta$  world, so such substitution also works well. This is very powerful intuition. Now, with above understanding, let's define more challenging and powerful rules for substitutions.

Construction 1.3.11.

$$\frac{\Delta \vdash \gamma : \Gamma \quad \Gamma \vdash A \text{ type} \quad \Delta \vdash a : A[\gamma]}{\Delta \vdash \mathbf{p} \circ (\gamma.a) = \gamma : \Gamma}$$

$$\frac{\Delta \vdash \gamma : \Gamma \quad \Gamma \vdash A \text{ type} \quad \Delta \vdash a : A[\gamma]}{\Delta \vdash \mathbf{q}[\gamma.a] = a : A[\gamma]}$$

$$\frac{\Gamma \vdash A \text{ type} \quad \Delta \vdash \gamma : \Gamma.A}{\Delta \vdash \gamma = (\mathbf{p} \circ \gamma).\mathbf{q}[\gamma] : \Gamma.A}$$

These rules are seemed to be very unclear. However, see following diagram. For first rule,

$$\begin{array}{ccc} \Delta & \xrightarrow{\gamma} & \Gamma \\ & \searrow \gamma.a & \uparrow p \quad \downarrow id.? \\ & & \Gamma.A \end{array}$$

We can represent the first rule as following way.

$$\begin{array}{ccc} \Delta & \xrightarrow{\gamma} & \Gamma \\ & \searrow \gamma.a & \uparrow p \quad \downarrow id.? \\ & & \Gamma.A \end{array}$$

So we can intuitively know that  $\gamma$  and  $\mathbf{p} \circ (\gamma.a)$  represent same traffic way from  $\Delta$  to  $\Gamma$ . For second rule, we already explained in front page. When we meet  $x_{n+1}$  (i.e. meet  $q$  in  $\Gamma.A$ ) then bring it from  $\Gamma.A$  into  $\Delta$  and substitute by  $a$ , term of  $\Delta$ . This rule explicitly claim this intuition. For third rule, let's draw slightly changed diagram.

$$\begin{array}{ccc} \Delta & & \Gamma \\ & \searrow \gamma & \uparrow p \quad \downarrow id.? \\ & & \Gamma.A \end{array}$$

Can we see what is  $(\mathbf{p} \circ \gamma)$  here? We can draw

$$\begin{array}{ccc} \Delta & \xrightarrow{p \circ \gamma} & \Gamma \\ & \searrow \gamma & \uparrow p \quad \downarrow id.? \\ & & \Gamma.A \end{array}$$

Now, see ???. When we make  $\mathbf{p} \circ \gamma : \Delta \rightarrow \Gamma$  to something that  $\Delta \rightarrow \Gamma.A$ , we append something to  $\mathbf{p} \circ \gamma$ . Here, actually the answer is already given. Suppose  $q$  in  $\Gamma.A$ , it has type  $A[\mathbf{p}]$  in  $\Gamma.A$ . We define  $\gamma$  by, how we take variable  $\mathbf{q}$  in  $\Gamma.A$  to  $\Delta$ ? Actually its answer is  $\mathbf{q}[\gamma]$ . It has type  $A[\mathbf{p}][\gamma] = A[\mathbf{p} \circ \gamma]$  in  $\Delta$ . So to make same substitution with  $\gamma$ , we must append  $\mathbf{q}[\gamma]$  at  $\mathbf{p} \circ \gamma$ . This intuition gives us  $\gamma = (\mathbf{p} \circ \gamma).\mathbf{q}[\gamma]$  in  $\Delta$ .

### Exercise 2.1

Prove that if  $\Gamma \vdash \gamma : \Gamma.A$ , then  $\mathbf{p} \circ \gamma = \mathbf{id}$

*Proof.* How can we define  $\gamma$ ? The intuitive meaning of  $\gamma$  is that, way to bring types and terms of  $\Gamma.A$  into  $\Gamma$ . Imagine that  $\Gamma.A$  contains types of  $x_1, \dots, x_{n+1}$  and  $\Gamma$  contains types of  $x_1, \dots, x_n$ . To bring types and terms in  $\Gamma.A$  into  $\Gamma$ , only to do is when we meet  $x_{n+1}$ , substitute it with well-formed term in  $\Gamma$  context. By this way we can delete the appearance of  $x_{n+1}$  and make types and terms in  $\Gamma.A$  become types and terms in  $\Gamma$ . So,  $\gamma$  is defined by following way. This is construction 1.3.10.

$$\frac{\Gamma \vdash \mathbf{id} : \Gamma \quad \Gamma \vdash A \text{ type} \quad \Gamma \vdash a : A}{\Gamma \vdash \underbrace{\mathbf{id}.a}_{\gamma} : \Gamma.A}$$

However, according to construction 1.3.11 we can know that

$$\mathbf{p} \circ (\mathbf{id}.a) = \mathbf{id}$$

□

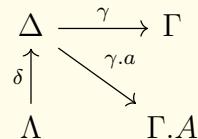
### Exercise 2.2

Show that  $(\gamma.a) \circ \delta = (\gamma \circ \delta).a[\delta]$

*Proof.* Recall third rule in construction 1.3.11.,

$$\frac{\Gamma \vdash A \text{ type} \quad \Delta \vdash \gamma^* : \Gamma.A}{\Delta \vdash \gamma^* = (\mathbf{p} \circ \gamma^*).\mathbf{q}[\gamma^*] : \Gamma.A}$$

Here, the pre-suppositions of problem is that  $\Delta \vdash \gamma : \Gamma$ ,  $\Delta \vdash a : A[\gamma]$ ,  $\Gamma \vdash A$  type and  $\delta$  be any substitution  $\delta : \Lambda \rightarrow \Delta$ . In diagram,



(Continue in Next Page . . .)

□

## Exercise 2.2

For intuition, we can draw following substitution.

$$\begin{array}{ccc} \Delta & \xrightarrow{\gamma} & \Gamma \\ \delta \uparrow & \searrow \gamma.a & \\ \Lambda & \xrightarrow[(\gamma.a) \circ \delta]{} & \Gamma.A \end{array}$$

And also,

$$\begin{array}{ccc} \Delta & \xrightarrow{\gamma} & \Gamma \\ \delta \uparrow & \nearrow \gamma \circ \delta & \\ \Lambda & \xrightarrow{\gamma \circ \delta} & \Gamma.A \end{array} \implies \begin{array}{ccc} \Delta & \xrightarrow{\gamma} & \Gamma \\ \delta \uparrow & \nearrow \gamma \circ \delta & \\ \Lambda & \xrightarrow[(\gamma \circ \delta).?]{} & \Gamma.A \end{array}$$

Here our claim is that ? part is  $a[\delta]$  and  $(\gamma.a) \circ \delta = (\gamma \circ \delta).a[\delta]$ . At first, let

$$\gamma^* = (\gamma.a) \circ \delta$$

Then  $\Lambda \vdash \gamma^* : \Gamma.A$  because

$$\frac{\Lambda \vdash \delta : \Delta \quad \Delta \vdash \gamma.a : \Gamma.A}{\Lambda \vdash \underbrace{(\gamma.a) \circ \delta}_{\gamma^*} : \Gamma.A}$$

So we can apply construction 1.3.11.

$$\frac{\Gamma \vdash A \text{ type} \quad \Lambda \vdash (\gamma.a) \circ \delta : \Gamma.A}{\Lambda \vdash (\gamma.a) \circ \delta = \underbrace{(\mathbf{p} \circ ((\gamma.a) \circ \delta)).\mathbf{q}[(\gamma.a) \circ \delta]}_{(*)} : \Gamma.A}$$

Here, our claim is that  $(*) = (\gamma \circ \delta).a[\delta]$ . This is because,

$$\begin{aligned} & (\mathbf{p} \circ ((\gamma.a) \circ \delta)).\mathbf{q}[(\gamma.a) \circ \delta] \\ &= ((\mathbf{p} \circ (\gamma.a)) \circ \delta).\mathbf{q}[(\gamma.a) \circ \delta] \\ &= (\gamma \circ \delta).\mathbf{q}[(\gamma.a) \circ \delta] \\ &= (\gamma \circ \delta).(\mathbf{q}[\gamma.a][\delta]) \\ &= (\gamma \circ \delta).a[\delta] \end{aligned}$$

This is end of proof.

## Exercise 2.3

Given  $\Delta \vdash \gamma : \Gamma$  and  $\Gamma \vdash A$  type, construct  $\gamma.A$  such that  $\Delta.A[\gamma] \vdash \gamma.A : \Gamma.A$ .

*Proof.* Actually, this is also very clear notion. Let's see following diagram.

$$\begin{array}{ccc} \Delta & \xrightarrow{\gamma} & \Gamma \\ p_\Delta \uparrow & & \\ \Delta.A[\gamma] & & \Gamma.A \end{array}$$

This implies that

$$\begin{array}{ccc} \Delta & \xrightarrow{\gamma} & \Gamma \\ p_\Delta \uparrow & \nearrow \gamma \circ p_\Delta & \\ \Delta.A[\gamma] & & \Gamma.A \end{array}$$

And our intuition for  $\gamma.A$  is that

$$\begin{array}{ccc} \Delta & \xrightarrow{\gamma} & \Gamma \\ p_\Delta \uparrow & \nearrow \gamma \circ p_\Delta & \\ \Delta.A[\gamma] & \xrightarrow[\text{(}\gamma \circ p_\Delta\text{).?}]^{\gamma.A} & \Gamma.A \end{array}$$

We can apply substitution extend rule here.

$$\frac{\Delta.A[\gamma] \vdash \gamma \circ p_\Delta : \Gamma \quad \Gamma \vdash A \text{ type} \quad \Delta.A[\gamma] \vdash \mathbf{q} : A[\gamma][\mathbf{p}_\Delta] = A[\gamma \circ \mathbf{p}_\Delta]}{\Delta.A[\gamma] \vdash (\gamma \circ \mathbf{p}_\Delta).\mathbf{q} : \Gamma.A}$$

Actually, this is our construction for  $\gamma.A$ .

$$\gamma.A := (\gamma \circ \mathbf{p}).\mathbf{q}, \quad \gamma.A : \Delta.A[\gamma] \rightarrow \Gamma.A$$

□

# Chapter 2

## Internalizing Judgemental Structure : $\Pi, \Sigma, \text{Eq}, \text{Unit}$

### 2.1 Basic Intuitions

We noted that Dependent Type Theory is similar with First Order Logic. In first order logic, there exists notion of assignment  $s$  in Interpretation  $\mathfrak{A} = \langle |\mathfrak{A}|, I \rangle$  and variants of assignments. Both are corresponds into context  $\Gamma$  and substitution  $\gamma$  in chapter 1. And then, what's the remaining things? Since dependent type is similar with FOL formulas, we need to construct intuition of correspondings between them. First, FOL terms are corresponds with ‘terms’ in our system. Second, predicates are corresponds with ‘type(dependent type)’ in our system. These are all what we talk about in previous chapter. Remain things are connectives, quantifiers, true and false. (And one specific predicate, equality judgement.) They are corresponds w.r.t. non-dependent connectives, dependent sum( $\exists$ ), dependent product( $\forall$ ), Unit and Void.

I'll try to make more clear between those similarity, in each section. Before that, let's claim the slogan for (any) these connective or quantifier constructors.

#### Slogan

Any such connectives or quantifiers in our system is given by claiming followings :

1. Natural Type-forming Operation
2. Natural Isomorphism Relating that Type's terms to judgementally-determined structure.

First one means that for each newly introduced types in this chapter, there are explicit and meta-level clear construct operation for such type. i.e. We must have the recipe of type.

Second one means that, if we collect all terms that have types that introduced here, then the set must have isomorphism (meta-level meaning) with external (meta-level) structure.

## 2.2 Dependent Sums

First of all, let's make clear all the intuitions for dependent sum.

1. The type former of dependent sum is  $\Sigma$ . We'll define soon.
2. Dependent sum's judgementally-determined structure is 'dependent pair'.
3. Dependent sum is similar to  $\exists$ .

What is dependent pair? Imagine following 'type' in dependent type system.

$$(n : Nat, \text{Vec } A n)$$

Actually, this is not matched with our construction of De Bruijn index. Matched version is that

$$(Nat, \text{Vec } A \mathbf{q})$$

Then imagine the Curry-Howard corresponds here. The term of above type is like 'one' specific example of above pair, for example

$$(2, [a, a'])$$

We can understand that this is proof for

$$(\exists n : Nat)(\text{Vec } A n)$$

In this intuition, we say that dependent sum is similar to  $\exists$ . From now on, let's construct above intuitions explicitly by defining operations, isomorphisms, and rules.

**Construction 2.2.1. Natural Type Formal**

$$\Sigma_\Gamma : \left( \biguplus_{A \in Ty(\Gamma)} Ty(\Gamma.A) \right) \rightarrow Ty(\Gamma)$$

The domain space's meaning is (1) choose valid type  $A$  in  $\Gamma$  context and (2) Under previous choose, choose valid type in  $\Gamma.A$ . Then the codomain space's meaning is that, by this  $\Sigma_\Gamma$  operator, such pair become valid type in  $\Gamma$ . Then where is 'Natural'? It's intuitive meaning is that, we can feel free to choose context  $\Gamma$  here. i.e. freely substitute and then change context, the operator's working does not change. We can write above natural mapping as following inference rules.

**Construction 2.2.1. Natural Type Formal ( inference rule ver. )**

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma.A \vdash B \text{ type}}{\Gamma \vdash \Sigma(A, B) \text{ type}} \quad \frac{\Delta \vdash \gamma : \Gamma \quad \Gamma \vdash A \text{ type} \quad \Gamma.A \vdash B \text{ type}}{\Delta \vdash \Sigma(A, B)[\gamma] = \Sigma(A[\gamma], B[\gamma.A]) \text{ type}}$$

For the second rule, it can be not clear that why  $B[\gamma.A]$  introduced. Let's draw the diagram. See this construction for  $\gamma.A$

$$\Delta \xrightarrow{\gamma} \Gamma$$

$$\Delta.A[\gamma] \xrightarrow{\gamma.A} \Gamma.A$$

We can pull-back  $A$  from  $\Gamma$  to  $\Delta$  through  $\gamma$ . And note that  $B$  is type in the  $\Gamma.A$ . So if we pulling back this into  $\Delta.A[\gamma]$ , using  $\gamma.A$ . So, the pull-backed type is  $B[\gamma.A]$  in  $\Delta.A[\gamma]$ . It means that

$$\Delta \vdash A[\gamma] \text{ type}, \quad \Delta.A[\gamma] \vdash B[\gamma.A] \text{ type}$$

Then the type-formal implies that

$$\Delta \vdash \Sigma(A[\gamma], B[\gamma.A]) \text{ type}$$

And naturality means that, this type works same with  $\Sigma(A, B)$  in  $\Gamma$ . So construct identity with it's pull-backed version  $\Sigma(A, B)[\gamma]$  and above.

Then here, how can we define the isomorphism between ‘terms’ that have  $\Sigma(A, B)$  types and such meta-level notion of ‘pair’? See

$$\iota_{\Gamma, A, B} : \text{Tm}(\Gamma, \Sigma(A, B)) \xrightarrow{\cong} \biguplus_{a \in \text{Tm}(\Gamma, A)} \text{Tm}(\Gamma, B[\mathbf{id}.a])$$

Note that this is isomorphism on the ‘terms’. Right-hand side contain all dependent pairs (meta-level representation) and our goal is match our new type  $\Sigma$  into that notion. This setting is just done at abstraction level. Now our interests are (1) How can we unfold above isomorphism explicitly, using inference rules? (2) Can we define  $\iota_{\Gamma, A, B}$ ,  $\iota_{\Gamma, A, B}^{-1}$  and round-trip among them preserves structures? (3) Do such constructions have naturality?

These are remain works in this chapter.

### Exercise 2.16.

Construct non-dependent pair type, with dependent sum  $\Sigma$ .

*Proof.* TODO

□

Now let's construct above isomorphisms explicitly.

## Construction 2.2.2.

This is explicit un-packing of  $\iota_{\Gamma, A, B}$

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma.A \vdash B \text{ type} \quad \Gamma \vdash p : \Sigma(A, B)}{\Gamma \vdash \mathbf{fst}(p) : A}$$

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma.A \vdash B \text{ type} \quad \Gamma \vdash p : \Sigma(A, B)}{\Gamma \vdash \mathbf{snd}(p) : B[\mathbf{id}.fst(p)]}$$

Second construction is very genious. It gives us very good intuition. When  $\Gamma \vdash p : \Sigma(A, B)$  is given, we can take first element  $a$  of  $p$  easily. And then, when we meet  $\mathbf{q}$  in type  $B$ , automatically understand that it is  $a$  we taken. By this we can write

$$(2, [a_1, a_2]) : \Sigma(Nat, \mathbf{Vec} A \ \mathbf{q})$$

Actually above type generally represents following intuitive type without variable names.

$$(n : Nat, \mathbf{Vec} A \ \mathbf{n})$$

Then now, it's time to clarify the inverse isomorphism  $\iota_{\Gamma, A, B}^{-1}$ . Intuitive meaning of this isomorphism is that, connect each meta-level dependent pair terms into our type system.

## Construction 2.2.3.

$$\iota_{\Gamma, A, B}^{-1} : \uplus_{a \in \text{Tm}(\Gamma, A)} \text{Tm}(\Gamma, B[\mathbf{id}.a]) \longrightarrow \text{Tm}(\Gamma, \Sigma(A, B))$$

$$\frac{\Gamma \vdash a : A \quad \Gamma.A \vdash B \text{ type} \quad \Gamma \vdash b : B[\mathbf{id}.a]}{\Gamma \vdash \mathbf{pair}(a, b) : \Sigma(A, B)}$$

I want to write some understanding for this construction. I think that most important one is that  $b$  is absolutely term of  $\Gamma$ . So, we can't use exact term  $a$  via  $\mathbf{q}$  when express  $b$ . However, term  $a$  is used when express type of  $b$ , without writing  $a$  directly. This is very very similar with  $\exists$  quantifier. Imagine that we have set of premises  $\Gamma$ . Let's interpret above inference rule.

1. Under premises  $\Gamma$ , we have proof of  $A$  via  $a$ .
2. Under premises  $\Gamma$  with  $A$ , claim  $B$ .
3. Under premises  $\Gamma$ ,  $b$  is proof of  $B[\mathbf{id}.a]$ .

These three Curry-Howard statements give us intuition about the result of inference rule.

1. There exists proof  $a$  of  $A$ . By using this, we can prove  $B[\mathbf{id}.a]$  via  $b$ .
2.  $\iff (\exists x).(B[\mathbf{id}.x])$  satisfiable.

Now second job is that claiming these two isomorphisms are symmetric. It means that the correspondings between our type system and meta-level dependent pair is bijective. Actually proving is done by constructing  $\beta$ -rule and  $\eta$ -rule. To show bijective, we just show that round-trip between terms and meta-pairs always preserve original element. Formally, we want to make

$$\iota_{\Gamma,A,B} \circ \iota_{\Gamma,A,B}^{-1} = \mathbf{id}_{\mathbf{meta}}, \quad \iota_{\Gamma,A,B}^{-1} \circ \iota_{\Gamma,A,B} = \mathbf{id}_{\mathbf{term}}$$

This is guaranteed explicitly via following rules.

#### Construction 2.2.4.

$$\begin{array}{c} \frac{\Gamma \vdash A \text{ type} \quad \Gamma.A \vdash B \text{ type} \quad \Gamma \vdash p : \Sigma(A, B)}{\Gamma \vdash \mathbf{pair}(\mathbf{fst}(p), \mathbf{snd}(p)) = p : \Sigma(A, B)} \quad (\eta\text{-rule}) \\ \\ \frac{\Gamma \vdash a : A \quad \Gamma.A \vdash B \text{ type} \quad \Gamma \vdash b : B[\mathbf{id}.a]}{\Gamma \vdash \mathbf{fst}(\mathbf{pair}(a, b)) = a : A} \quad (\beta\text{-rule}) \\ \\ \frac{\Gamma \vdash a : A \quad \Gamma.A \vdash B \text{ type} \quad \Gamma \vdash b : B[\mathbf{id}.a]}{\Gamma \vdash \mathbf{snd}(\mathbf{pair}(a, b)) = b : B[\mathbf{id}.a]} \quad (\beta\text{-rule}) \end{array}$$

Recap our ultimate goal. The last job is clarify that this isomorphism has naturality. ‘Naturality’ means that we can freely use substitutions when apply this rule, or more easily, for all contexts, these pair structure is preserved. For this sake, we can make formal construction by rule also.

#### Construction 2.2.5.

$$\frac{\Delta \vdash \gamma : \Gamma \quad \Gamma \vdash a : A \quad \Gamma.A \vdash B \text{ type} \quad \Gamma \vdash b : B[\mathbf{id}.a]}{\Delta \vdash \mathbf{pair}(a, b)[\gamma] = \mathbf{pair}(a[\gamma], b[\gamma]) : \Sigma(A, B)[\gamma]}$$

This is very intuitive. When we make  $\mathbf{pair}(a, b)$  toy in  $\Gamma$ , we have 2 ways to take that toy into  $\Delta$  via  $\gamma$ . First way is that take the toy directly ( $\mathbf{pair}(a, b)[\gamma]$ ) and second way is that take two materials ( $a[\gamma], b[\gamma]$ ) and make in  $\Delta$  (then,  $\mathbf{pair}(a[\gamma], b[\gamma])$ ). This rule say that both are same. This is intuitive meaning of naturality. However, this is just naturality of  $\iota_{\Gamma,A,B}^{-1}$ . We need to show naturality of  $\iota_{\Gamma,A,B}$  also. But this is derivable.

$$\begin{aligned} \mathbf{fst}(p[\gamma]) &= \mathbf{fst}(\mathbf{pair}(\mathbf{fst}(p), \mathbf{snd}(p))[\gamma]) \\ &= \mathbf{fst}(\mathbf{pair}(\mathbf{fst}(p)[\gamma], \mathbf{snd}(p)[\gamma])) \\ &= \mathbf{fst}(p)[\gamma] \end{aligned}$$

## Exercise 2.17.

Show that well-typedness of  $b[\gamma]$  in Construction 2.2.5.

*Proof.* Our goal is showing that  $\Delta \vdash b[\gamma] : B[\gamma.A][\mathbf{id}.a[\gamma]]$  (And this is well-formed type in  $\Delta$  also). This is from pattern-matching with Construction 2.2.3. See following diagram first.

$$\begin{array}{ccc} \Delta & \xrightarrow{\gamma} & \Gamma \\ id.a[\gamma] \downarrow & \searrow \gamma.a[\gamma] & \downarrow id.a \\ \Delta.A[\gamma] & \xrightarrow{\gamma.A} & \Gamma.A \end{array}$$

Then see how can we take  $B$  from  $\Gamma.A$  to  $\Delta$  and  $b$  from  $\Gamma$  to  $\Delta$ .  $B$  is pull-backed through red way and  $b$  is pull-backed through blue way. So  $b[\gamma]$  is well-formed term in  $\Delta$ . And similarly,  $B[\gamma.A][\mathbf{id}.a[\gamma]]$  is well-formed type in  $\Delta$ . Last, we must show that  $b[\gamma]$  has such type. Since we know that  $\Gamma \vdash b : B[\mathbf{id}.a]$ , it is clear that  $\Delta \vdash b[\gamma] : B[\mathbf{id}.a][\gamma]$ . Then we can complete proof via showing  $\Delta \vdash B[\gamma.A][\mathbf{id}.a[\gamma]] = B[\mathbf{id}.a][\gamma]$  type. This holds because

$$\begin{aligned} [\gamma.A][\mathbf{id}.a[\gamma]] &= \gamma.A \circ (\mathbf{id}.a[\gamma]) \\ &= \gamma.a[\gamma] \quad \because \text{purple arrow above} \\ &= (\mathbf{id}.a) \circ \gamma \\ &= [\mathbf{id}.a][\gamma] \end{aligned}$$

□

## 2.3 Dependent Products

Here also, follow up the constructing of previous chapter. We'll clarify intuition for this type by :

1. The type former of dependent product is  $\Pi$ . We'll define soon.
2. Dependent product's judgementally-determined structure is 'dependent function'.
3. Dependent product is similar to  $\forall$ .

Here, what is dependent function? Imagin following type in dependent type system.

$$(n : Nat) \longrightarrow \mathbf{Vec} A n$$

The term of above type works like a function, whose input is natural number  $n$  and output is vector with length  $n$ . So this is very similar with  $\forall$ . Once we throw any natural number  $n$ , it returns proof of output type. i.e. this type is similar to claim that

$$(\forall n : Nat)(\mathbf{Vec} A n)$$

Recap the slogan. When we introduce types here, we'll define (1) Natural Type Former and (2) Natural Isomorphisms between terms and external structure. As we've studied in previous chapter, we can expect what are them.

Construction 2.3.1. Natural Type Formal

$$\Pi_\Gamma : \left( \biguplus_{A \in Ty(\Gamma)} Ty(\Gamma.A) \right) \rightarrow Ty(\Gamma)$$

And then, we can construct explicit rules for make this operator as same previous.

Construction 2.3.2.

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma.A \vdash B \text{ type}}{\Gamma \vdash \Pi(A, B) \text{ type}} \quad \frac{\Delta \vdash \gamma : \Gamma \quad \Gamma \vdash A \text{ type} \quad \Gamma.A \vdash B \text{ type}}{\Delta \vdash \Pi(A, B)[\gamma] = \Pi(A[\gamma], B[\gamma.A]) \text{ type}}$$

This is our 'type former' for dependent product. Actually for understanding type forming, it isn't difficult since we deal samely in previous chapter. Now our goal is that, find and define natural isomorphisms between 'terms' that have  $\Pi(A, B)$  type and meta-level notion of 'dependent function'. Imagine that following two sets are isomorphic.

Construction 2.3.3. Natural Isomorphism of Terms

$$Tm(\Gamma, \Pi(A, B)) \cong Tm(\Gamma.A, B)$$

And divide each isomorphism via

$$\iota_{\Gamma, A \rightarrow B} : Tm(\Gamma, \Pi(A, B)) \xrightarrow{\cong} Tm(\Gamma.A, B)$$

$$\iota_{\Gamma, A \rightarrow B}^{-1} : Tm(\Gamma.A, B) \xrightarrow{\cong} Tm(\Gamma, \Pi(A, B))$$

Now, let's do same construction with previous chapter. (1) How can we unfold above isomorphism explicitly, using inference rules? (2) Can we define  $\iota_{\Gamma, A \rightarrow B}$ ,  $\iota_{\Gamma, A \rightarrow B}^{-1}$  and round-trip among them preserves structures? (3) Do such constructions have naturality?

Construction 2.3.4.

Let's unpack  $\iota_{\Gamma, A \rightarrow B}$

$$\frac{\Gamma \vdash a : A \quad \Gamma.A \vdash B \text{ type} \quad \Gamma \vdash f : \Pi(A, B)}{\Gamma \vdash \mathbf{app}(f, a) : B[\mathbf{id}.a]}$$

Let's unpack  $\iota_{\Gamma, A \rightarrow B}^{-1}$

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma.A \vdash b : B}{\Gamma \vdash \lambda(b) : \Pi(A, B)}$$

Both are very intuitive. First one is application of function, and second one is lambda-definition of function without variable name. However, I want to note some slight difference between dependent sum and above. One difference is that, place that  $b$  exists. At dependent sum,  $b$  exists in  $\Gamma$  context. So we explain that we can't use **q** for express  $a$  in  $b$ . However, at above,  $b$  is term of  $\Gamma.A$  world. So we can use **q** for represent  $b$ . This is one of major difference. Further, by this, we can agree our abstractical notion for matching of  $\forall$  and  $\Pi$ . We can always take proof  $b$  of  $B$  under premises  $\Gamma$  appended with  $A$  by substitution **id**. $a$  for all  $a$ . It gives us that, with any proof  $x$  of  $A$ ,  $B[\mathbf{id}.x]$  is true. Formally, existance term of type  $\Pi(A, B)$  is proof of

$$(\forall x).(B[\mathbf{id}.x])$$

The next job is that claiming these two isomorphisms are symmetric. Similar to previous chapter, we need to define

$$\iota_{\Gamma, A \rightarrow B} \circ \iota_{\Gamma, A \rightarrow B}^{-1} = \mathbf{id}_{meta}, \quad \iota_{\Gamma, A \rightarrow B}^{-1} \circ \iota_{\Gamma, A \rightarrow B} = \mathbf{id}_{term}$$

This is guaranteed explicitly via following rules.

## Construction 2.3.5.

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma.A \vdash B \text{ type} \quad \Gamma \vdash f : \Pi(A, B)}{\Gamma \vdash \lambda(\mathbf{app}(f[\mathbf{p}], \mathbf{q})) = f : \Pi(A, B)} \quad (\eta\text{-rule})$$

$$\frac{\Gamma \vdash a : A \quad \Gamma.A \vdash b : B}{\Gamma \vdash \mathbf{app}(\lambda(b), a) = b[\mathbf{id}.a] : B[\mathbf{id}.a]} \quad (\beta\text{-rule})$$

Our final goal is state that this isomorphism has naturality. This can be holded by make following rules explicitly.

## Construction 2.3.6.

$$\frac{\Delta \vdash \gamma : \Gamma \quad \Gamma \vdash A \text{ type} \quad \Gamma.A \vdash B \text{ type}}{\Delta \vdash \Pi(A, B)[\gamma] = \Pi(A[\gamma], B[\gamma.A]) \text{ type}}$$

$$\frac{\Delta \vdash \gamma : \Gamma \quad \Gamma \vdash A \text{ type} \quad \Gamma.A \vdash b : B}{\Delta \vdash \lambda(b)[\gamma] = \lambda(b[\gamma.A]) : \Pi(A, B)[\gamma]}$$

$$\frac{\Delta \vdash \gamma : \Gamma \quad \Gamma \vdash a : A \quad \Gamma.A \vdash B \text{ type} \quad \Gamma \vdash f : \Pi(A, B)}{\Delta \vdash \mathbf{app}(f, a)[\gamma] = \mathbf{app}(f[\gamma], a[\gamma]) : B[\gamma.a[\gamma]]}$$

## Exercise 2.6.

At Construction 2.3.6. prove that  $\mathbf{app}(f, a)[\gamma]$  and  $\mathbf{app}(f[\gamma], a[\gamma])$  have the type  $B[\gamma.a[\gamma]]$ .

*Proof.* Let's draw diagram,

$$\begin{array}{ccc} \Delta & \xrightarrow{\gamma} & \Gamma \\ id.a[\gamma] \downarrow & \searrow \gamma.a[\gamma] & \downarrow id.a \\ \Delta.A[\gamma] & \xrightarrow{\gamma.A} & \Gamma.A \end{array}$$

We can take  $a$  from  $\Gamma$  to  $\Delta$ , type  $B$  from  $\Gamma.A$  to  $\Delta.A[\gamma]$ , and take  $f$  from  $\Gamma$  to  $\Delta$ . It gives us

$$\Delta \vdash a[\gamma] : A[\gamma] \quad \Delta.A[\gamma] \vdash B[\gamma.A] \text{ type} \quad \Delta \vdash f[\gamma] : \Pi(A, B)[\gamma]$$

Then by Construction 2.3.4. we can get

$$\Delta \vdash \mathbf{app}(f[\gamma], a[\gamma]) : B[\gamma.A][\mathbf{id}.a]$$

However, we know that  $B[\gamma.A][\mathbf{id}.a] = B[\gamma.a[\gamma]]$ , so

$$\Delta \vdash \mathbf{app}(f[\gamma], a[\gamma]) : B[\gamma.a[\gamma]]$$

And first one is easier, we can take  $\mathbf{app}(f, a) : B[\mathbf{id}.a]$  from  $\Gamma$  into  $\Delta$ .

$$\Delta \vdash \mathbf{app}(f, a)[\gamma] : B[\mathbf{id}.a][\gamma] = B[(\mathbf{id} \circ \gamma).a[\gamma]] = B[\gamma.a[\gamma]]$$

□

Exercise 2.7.

At Construction 2.3.5. prove that  $\lambda(\mathbf{app}(f[\mathbf{p}], \mathbf{q}))$  is well-formed term of  $\Pi(A, B)$  in  $\Gamma$ .

*Proof.* Our goal is showing

$$\Gamma \vdash \lambda(\mathbf{app}(f[\mathbf{p}], \mathbf{q})) : \Pi(A, B)$$

To this is valid, we need 2 premises

$$\Gamma \vdash A \text{ type}, \quad \Gamma.A \vdash \mathbf{app}(f[\mathbf{p}], \mathbf{q}) : B$$

First one is premised, so let's prove second one. Since  $f$  is term of  $\Gamma$ , so take it to  $\Gamma.A$  then  $f[\mathbf{p}]$ . So, we know that

$$\Gamma.A \vdash f[\mathbf{p}] : \Pi(A, B)[\mathbf{p}], \quad \Gamma.A \vdash \mathbf{q} : A[\mathbf{p}], \quad \Gamma.A.A[\mathbf{p}] \vdash B[\mathbf{p}.A] \text{ type}$$

Then with Construction 2.3.4. we can get

$$\Gamma.A \vdash \mathbf{app}(f[\mathbf{p}], \mathbf{q}) : B[\mathbf{p}.A][\mathbf{id}.\mathbf{q}]$$

Then remain thing to show is

$$\Gamma.A \vdash B = B[\mathbf{p}.A][\mathbf{id}.\mathbf{q}] \text{ type}$$

Actually, this is intuitive.  $\mathbf{p}.A$  is weakening from  $\Gamma.A$  into  $\Gamma.A.A[\mathbf{p}]$ . So it is just send  $B$  in  $\Gamma.A$  to  $\Gamma.A.A[\mathbf{p}]$  and take it again through  $\mathbf{id}.\mathbf{q}$ . Formal writing is that,

$$B[\mathbf{p}.A][\mathbf{id}.\mathbf{q}] = B[\mathbf{p}.A \circ \mathbf{id}.\mathbf{q}] = B[(\mathbf{p} \circ \mathbf{id}).\mathbf{q}] = B[\mathbf{p}.\mathbf{q}] = B$$

This is end of proof. □

Here, I'll add some intuition, introduced in page 41 of textbook. Recap the natural isomorphism, such that terms whose type is  $\Pi(A, B)$  in  $\Gamma$  and terms whose type is  $B$  in  $\Gamma.A$ .

$$Tm(\Gamma, \Pi(A, B)) \cong Tm(\Gamma.A, B)$$

Here, we can easily understand the inverse isomorphism  $\iota_{\Gamma, A \rightarrow B}^{-1}$ . It is similar to pull-back  $b$  which is term of type  $B$  in  $\Gamma.A$  into  $\Gamma$  context by unknown extending  $\mathbf{id}.\_?$ . It is denoted by  $\lambda(b)$ . However, when we see  $\iota_{\Gamma, A \rightarrow B}$ , the meaning of this map is send  $\Gamma \vdash f : \Pi(A, B)$  into term of type  $B$  in  $\Gamma.A$ . However, we introduced the forward map via const 2.3.4.,

$$\frac{\Gamma \vdash a : A \quad \Gamma.A \vdash B \text{ type} \quad \Gamma \vdash f : \Pi(A, B)}{\Gamma \vdash \mathbf{app}(f, a) : B[\mathbf{id}.a]}$$

It is quite un-expectable. Because in our intuition, forward isomorphism's range space is  $\Gamma.A$ . If we follow that, our  $\mathbf{app}$  maybe  $\Gamma.A \vdash \mathbf{app}^*(f[\mathbf{p}], \mathbf{q})$ . This is also valid

construction of elimination rule for  $\Pi$ -type. And also this is intuitively understandable, however this form of definition of application is not familiar to us. Most of time, we imagine application via  $f(a), \mathbf{f}(a)$  this form. This is reason why we use above form of **app** definition. When we move from context  $\Gamma$  to  $\Gamma.A$ , we always give  $\Gamma \vdash a : A$ , and **app**\* automatically has the intension that we'll use such ghost  $a$ . For convenience, our intension introducing here is that, externally mention such  $a$  we use. Then pull-back **app** $^*(f[\mathbf{p}], \mathbf{q})$  into  $\Gamma$  by **id**. $a$ , we can get above form of **app** definition.

## 2.4 Extensional Equality

Actually, the constructing for here is very simple and similar to previous sections. However, there are many remarkable stuffs about what ‘equality’ means. Because we already use a kind of equality many times. Recap

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma, A \vdash B \text{ type} \quad \Gamma \vdash p : \Sigma(A, B)}{\Gamma \vdash \mathbf{pair}(\mathbf{fst}(p), \mathbf{snd}(p)) = p : \Sigma(A, B)}$$

Here we already used ‘=’ symbol which means equality. This kind of equality is called definitional equality. This equality is part of constructing definition of pair. (Especially,  $\eta$ -rule for define validity of  $\iota_{\Gamma, A, B}^{-1}$ ) One good example to understand this is that,

**Listing 2.1:** Add in Peano Arithmetic

```
let rec Add(n : Nat, m : Nat) =
  match (n, m) with
  | (n, 0) -> n
  | (n, succ m') -> Succ(Add(n, m'))
```

In this case,  $Add(n, 0)$  and  $n$  are definitionally equal. Similarly,  $Add(n, 2)$  and  $Succ(Succ(n))$  also definitionally equal. However, what about  $Add(n, m)$  and  $Add(m, n)$ ? We(meta-level) know that they are equal, but we need some proof. It does not follows directly from definition. Such ‘provable’ equality is called propositional equality. (or called extensional equality) Main target of this section is bring those equality between terms into our type system. Recap the Curry-Howard corresponding, we’ll bring such ‘provable proposition’ as a **Eq** type in our system, and if there is a term  $p$  of type **Eq** exists, then it is proof for such proposition. Let’s get started.

Construction 2.4.1. Natural Type Formal

$$\mathbf{Eq}_{\Gamma} : \left( \biguplus_{A \in Ty(\Gamma)} Tm(\Gamma, A) \times Tm(\Gamma, A) \right) \rightarrow Ty(\Gamma)$$

The un-packed version of above intuition is that,

Construction 2.4.2.

$$\frac{\Gamma \vdash a, b : A}{\Gamma \vdash \mathbf{Eq}(A, a, b) \text{ type}} \qquad \frac{\Delta \vdash \gamma : \Gamma \quad \Gamma \vdash a, b : A}{\Delta \vdash \mathbf{Eq}(A, a, b)[\gamma] = \mathbf{Eq}(A[\gamma], a[\gamma], b[\gamma]) \text{ type}}$$

Thanks for the previous works, it is not hard to understand. This is our natural type formal **Eq**. However something is quite different for defining natural isomorphisms.

Construction 2.4.3.

$$Tm(\Gamma, \mathbf{Eq}(A, a, b)) \cong \mathbb{E}_{a \equiv b} := \begin{cases} \{\ast\} & a \equiv b \\ \emptyset & a \not\equiv b \end{cases}$$

And divide each isomorphism via

$$\iota_{\Gamma, A, a \equiv b} : Tm(\Gamma, \mathbf{Eq}(A, a, b)) \xrightarrow{\cong} \mathbb{E}_{a \equiv b}$$

$$\iota_{\Gamma, A, a \equiv b}^{-1} : \mathbb{E}_{a \equiv b} \xrightarrow{\cong} Tm(\Gamma, \mathbf{Eq}(A, a, b))$$

This isomorphic target is  $\mathbb{E}_{a \equiv b}$ , quite unclear now. Here, I wrote  $\equiv$  symbol to mean propositional equality. We can interpret above isomorphism in natural language, by ‘If two terms  $a, b$  of type  $A$  in  $\Gamma$  are propositionally equal, then there exists a unique proof (term) for such propositional equality  $\mathbf{Eq}(A, a, b)$  and if not, there aren’t any proof (term) for that.’ Then now let’s unpack above isomorphisms explicitly.

Construction 2.4.4.

Unpacking  $\iota_{\Gamma, A, a \equiv b}$  :

$$\frac{\Gamma \vdash a, b : A \quad \Gamma \vdash p : \mathbf{Eq}(A, a, b)}{\Gamma \vdash a = b : A} \quad \frac{\Gamma \vdash a, b : A \quad \Gamma \vdash p : \mathbf{Eq}(A, a, b)}{\Gamma \vdash \mathbf{refl} = p : \mathbf{Eq}(A, a, b)}$$

Unpacking  $\iota_{\Gamma, A, a \equiv b}^{-1}$  :

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash \mathbf{refl} : \mathbf{Eq}(A, a, a)}$$

First rule implies that, if there exists proof of equality, we can use that as definitional equality, i.e. rewrite anywhere, anytime. And second rule implies that such proof is unique. These two rules construct forward isomorphism. However, third rule is kind of unique term-generator. Note that all of basic definition equalities are included here. We can understand this rule as visualization of  $a$ -quotient in meta-space. Our textbook write down the power of this type : (1) If there exists a proof for  $a \equiv b$ , i.e.  $\Gamma \vdash p : \mathbf{Eq}(A, a, b)$  then we can always, anywhere rewrite  $a$  to  $b$  and  $b$  to  $a$ . (2) Proof  $p$  can be a variable. Imagine the context  $\Gamma, \mathbf{Eq}(A, a, b)$ .

However, this is dangerous since pull propositional equality into definitional equality. This is main feature of extensional type system. However, it can break the decidability of type-checker.

## 2.5 Unit Type

This is the last, and the most simple type in chapter 2. Intuitive meaning of this type is **true** in logic. Imagine some functions in OCaml, which have type **Unit** → .. We can always execute this function without any input ( premises ), so the functionality of unit type is true in logic. Since the constructions are simple, I'll write down without detailed explaining.

Construction 2.5.1.

$$\mathbf{Unit}_\Gamma \in Ty(\Gamma)$$

$$Tm(\Gamma, \mathbf{Unit}) \cong \{*\}$$

Construction 2.5.2.

$$\frac{\vdash \Gamma \text{ cx}}{\Gamma \vdash \mathbf{Unit} \text{ type}}$$

$$\frac{\Delta \vdash \gamma : \Gamma}{\Delta \vdash \mathbf{Unit}[\gamma] = \mathbf{Unit} \text{ type}}$$

$$\frac{\vdash \Gamma \text{ cx}}{\Gamma \vdash \mathbf{tt} : \mathbf{Unit}}$$

$$\frac{\Gamma \vdash a : \mathbf{Unit}}{\Gamma \vdash a = \mathbf{tt} : \mathbf{Unit}}$$

# Chapter 3

## Inductive Types : Void, Bool, Nat

### 3.1 Basic Intuitions

Inductive types are types defined via induction principles. Our previous discussion is ‘For given context  $\Gamma$ , let’s define natural type  $\Upsilon$ . Its structure is similar to some external meta-level structure. i.e.  $Tm(\Gamma, \Upsilon) \cong \dots$ ’ Our textbook introduce this kind of term constructing as ‘hard coded’ type formers. However, in this chapter, our discussion is entirely changed as ‘First, let’s define type former  $\Upsilon$ . Second, assume that our interesting context already has such type by  $\Gamma.\Upsilon$ . Third, in such context  $\Gamma.\Upsilon$ , what’s going on terms of well-defined type  $A$  in  $\Gamma.\Upsilon$ ?’ This flow is sufficiently seemed like inductive definitions. Although we get such kind of intuitions, it can be still not familiar for us that defining types and connecting them with its natural meta-level structures via  $Tm(\Gamma.\Upsilon, A) \cong \dots$  So, textbook also gives us some set-theoretical intuitions. I’ll introduce such thing in this chapter first.

Suppose that  $A, B$  are sets. Can you define the Cartesian product  $A \times B$  of  $A, B$ ? The answer might be  $A \times B := \{(a, b) | a \in A, b \in B\}$ . Correct, however there are alternative way to defining  $A \times B$ . Actually it is characterize of set  $A \times B$ . We now define  $A \times B$  via **for any set  $X$ , if any  $f : X \rightarrow A \times B$  is given, then  $f_1 : X \rightarrow A$  and  $f_2 : X \rightarrow B$  are also determined and vice versa.** Here, the symbol  $A \times B$  such that holds above bold **characteristic** is definition of Cartesian product of  $A, B$ . ( More rigorously, such isomorphic quotients are  $A \times B$ . )

Another example is characterize of single-ton set **1**. One way for definition is construct isomorphic quotient  $\mathbf{1} \cong \{\ast\}$ . However it can be characterize by following way : **For any set  $X$ , there are only one function  $X \rightarrow \mathbf{1}$ . i.e.  $\forall X, |Hom(X, \mathbf{1})| = 1$**  All these kind of construction is what we did in previous chapter. Look similarity between

$$Hom(X, \mathbf{1}) \cong Tm(\Gamma, \mathbf{Unit})$$

Now let’s define emptyset  $\emptyset$  like this kind of construction. One turns out that, the most elegant characterization of  $\emptyset$  is **For any set  $X$ , there are only one function  $\emptyset \rightarrow X$ .** This is story about, why we put our target in left blank.

## 3.2 Void : The empty type

Poetically, **Void** type's role is **false** in first order logic, and **emptyset** in set theory. When we prove something in first order logic, if we reach  $I \models \perp$  under assuming  $I \models \Gamma$ , then we say that  $\Gamma$  is false, or  $I \not\models \Gamma$ . Very similar role is done by **Void**. If we reach well-defined term  $\Gamma \vdash b : \mathbf{Void}$ , then it is proof that our context  $\Gamma$  is wrong. i.e. Un-reachable context. Means that, **Void** type works like false proposition.

Then now, the construction of type former is not difficult. It is same as **Unit** in previous chapter.

Construction 3.2.1. Natural Type Formal

$$\mathbf{Void}_\Gamma \in \text{Ty}(\Gamma)$$

And also, easily unpack this meta-level representation into our type system by explicit rules.

Construction 3.2.2.

$$\frac{\vdash \Gamma \text{ ex}}{\Gamma \vdash \mathbf{Void} \text{ type}} \quad \frac{\Delta \vdash \gamma : \Gamma}{\Delta \vdash \mathbf{Void}[\gamma] = \mathbf{Void} \text{ type}}$$

Defining natural type formal is similar to previous. If we are living in Chapter 2, the basic next step will be trying to construct following isomorphism :

$$Tm(\Gamma, \mathbf{Void}) \cong \emptyset \quad \Leftarrow (\text{This is Wrong !!!})$$

If we append this isomorphism into our system, it will be rejected. This isomorphism doesn't agree with us because 2 major reasons, First is that even wrong context  $\Gamma$  also couldn't have **Void** type term. Second one is that, our already existing rules generate **Void** type terms, so the isomorphism between emptyset will be broken. The broken cases are following :  $\mathbf{q} \in Tm(\Gamma, \mathbf{Void}, \mathbf{Void})$ , and  $\mathbf{app}(\mathbf{q}, \mathbf{tt}) \in Tm(\Gamma, \Pi(\mathbf{Unit}, \mathbf{Void}), \mathbf{Void})$ . By those reasons, setting above isomorphism is disgusting.

Then it's time to think like previous intuition section. **If Void type is existing our context already, then it can have terms for that type!! Moreover, in such context, Everything is Wrong !!** So does not try to construct as 'Void is type characterized by, for any context  $\Gamma$ , no terms... Uh! It broke everything!' and try to 'Void is type characterized by, if our context have Void already, then everything is false!' So let's imagine following isomorphism :

Construction 3.2.3. Natural Term Isomorphism

$$\rho_{\Gamma, A} : Tm(\Gamma, \mathbf{Void}, A) \cong \{*\}$$

This isomorphism has interesting meanings, first one is that if our context contains **Void**, then for every type  $A$  and terms of type  $A$  is single-ton, which means that ‘false’. Another one is that, recap the isomorphic structure of dependent product type  $\Pi$ .

$$Tm(\Gamma, \Pi(A, B)) \cong Tm(\Gamma.A, B)$$

Under this, we can state

$$Tm(\Gamma, \Pi(\mathbf{Void}, A)) \cong Tm(\Gamma.\mathbf{Void}, A) \cong \{*\}$$

All dependent function that domain type is **Void** is single-ton element. Now our goal is that : bring this notion into our type system explicitly via rules. Approach one, bring above isomorphism ‘directly’ , means that

$$\frac{\vdash \Gamma \text{ cx } \Gamma.\mathbf{Void} \vdash A \text{ type}}{\Gamma.\mathbf{Void} \vdash \mathbf{absurd}' : A} \quad \frac{\vdash \Gamma \text{ cx } \Gamma.A \vdash a : A}{\Gamma.\mathbf{Void} \vdash \mathbf{absurd}' = a : A}$$

We bring the isomorphism literally. However, recap what we did when handle **app** construction, we’ve liked represent the constructed term in  $\Gamma$  context. However, above rules represents the constructed terms are in  $\Gamma.\mathbf{Void}$ . So, we’ll axiomatization that  $\Gamma \xrightarrow{id.b} \Gamma.\mathbf{Void}$  so get  $\mathbf{absurd}(b) := \mathbf{absurd}'[\mathbf{id}.b]$  where  $\Gamma \vdash b : \mathbf{Void}$  So we write cut-in version of term (proof for false) construction rules : ( These rules automatically infer the above rules, checking will be proved later as exercise. )

#### Construction 3.2.4.

$$\begin{array}{c} \frac{\Gamma \vdash b : \mathbf{Void} \quad \Gamma.\mathbf{Void} \vdash A \text{ type}}{\Gamma \vdash \mathbf{absurd}(b) : A[\mathbf{id}.b]} \\[10pt] \frac{\Delta \vdash \gamma : \Gamma \quad \Gamma \vdash b : \mathbf{Void} \quad \Gamma.\mathbf{Void} \vdash A \text{ type}}{\Delta \vdash \mathbf{absurd}(b)[\gamma] = \mathbf{absurd}(b[\gamma]) : A[\gamma.b[\gamma]]} \\[10pt] \frac{\Gamma \vdash b : \mathbf{Void} \quad \Gamma.\mathbf{Void} \vdash a : A}{\Gamma \vdash \mathbf{absurd}(b) = a[\mathbf{id}.b] : A[\mathbf{id}.b]} \end{array}$$

Understanding above is not difficult. Note that the meaning ‘inductive’ is well-expressed in here. We used type **Void** for premise of term constructor. So it is called inductive type. First rule tells us that, if we have a proof of false, then if we assume this as premise, then everything can be derived. Just the symbol  $\mathbf{absurd}(b)$  is proof of ‘every statements is false when we assume  $b$ ’. Second one gives naturality for the term constructor. Third rule guarantees the single-toness of proof of  $A$  in  $\Gamma.\mathbf{Void}$ .

Now, with Exercises we’ll make stronger understanding.

## Exercise 3.2.5.

Show that if  $\Gamma \vdash b_0, b_1 : \mathbf{Void}$  then  $\Gamma \vdash b_0 = b_1 : \mathbf{Void}$ .

*Proof.* First, we can get

$$\frac{\Gamma. \mathbf{Void} \vdash \mathbf{p} : \Gamma \quad \Gamma \vdash b_0, b_1 : \mathbf{Void}}{\Gamma. \mathbf{Void} \vdash b_0[\mathbf{p}], b_1[\mathbf{p}] : \underbrace{\mathbf{Void}[\mathbf{p}]}_{=\mathbf{Void}}}$$

Then,  $\eta$ -rule in const 3.2.4. implies that

$$\frac{\Gamma \vdash b_0, b_1 : \mathbf{Void} \quad \Gamma. \mathbf{Void} \vdash b_0[\mathbf{p}], b_1[\mathbf{p}] : \mathbf{Void}}{\begin{aligned} \Gamma \vdash \mathbf{absurd}(b_0) &= b_0[\mathbf{p}][\mathbf{id}.b_0] : \mathbf{Void}[\mathbf{id}.b_0] \\ \Gamma \vdash \mathbf{absurd}(b_0) &= b_1[\mathbf{p}][\mathbf{id}.b_0] : \mathbf{Void}[\mathbf{id}.b_0] \\ \Gamma \vdash \mathbf{absurd}(b_1) &= b_0[\mathbf{p}][\mathbf{id}.b_1] : \mathbf{Void}[\mathbf{id}.b_1] \\ \Gamma \vdash \mathbf{absurd}(b_1) &= b_1[\mathbf{p}][\mathbf{id}.b_1] : \mathbf{Void}[\mathbf{id}.b_1] \end{aligned}}$$

However, Exer 2.1. implies that  $\Gamma \vdash \mathbf{p} \circ (\mathbf{id}.b_0) = \mathbf{p} \circ (\mathbf{id}.b_1) = \mathbf{id}$ , and since  $\Gamma \vdash \mathbf{Void}[\mathbf{id}.b_0] = \mathbf{Void}[\mathbf{id}.b_1] = \mathbf{Void}$  We can get

$$\Gamma \vdash \mathbf{absurd}(b_0) = b_0 = \mathbf{absurd}(b_1) = b_1 : \mathbf{Void}$$

This implies that

$$\Gamma \vdash b_0 = b_1 : \mathbf{Void}$$

So, this intuitively gives us that **Void** term is always single-ton if it exists.  $\square$

## Exercise 3.2.6.

Fixing  $\Delta \vdash \gamma : \Gamma$ , prove that there is at most one substitution  $\Delta \vdash \bar{\gamma} : \Gamma. \mathbf{Void}$  such that satisfying  $\mathbf{p} \circ \bar{\gamma} = \gamma$ .

*Proof.* I'll draw diagram for good understanding.

$$\begin{array}{ccc} \Delta & \xrightarrow{\gamma} & \Gamma \\ & \searrow \bar{\gamma} & \uparrow p \\ & & \Gamma. \mathbf{Void} \end{array}$$

We'll use following rule for substitution :

$$\frac{\Gamma \vdash A \text{ type} \quad \Delta \vdash \gamma : \Gamma.A}{\Delta \vdash \gamma = (\mathbf{p} \circ \gamma). \mathbf{q}[\gamma] : \Gamma.A} (*)$$

Suppose that  $\bar{\gamma}_i, i = 1, 2$  such that  $\Delta \vdash \bar{\gamma}_i : \Gamma. \mathbf{Void}, i = 1, 2$  and  $(\mathbf{p} \circ \bar{\gamma}_i) = \gamma, i = 1, 2$  exists. Then we can apply (\*) by

$$\frac{\Gamma \vdash \mathbf{Void} \text{ type} \quad \Delta \vdash \bar{\gamma}_i : \Gamma. \mathbf{Void}}{\Delta \vdash \bar{\gamma}_i = (\mathbf{p} \circ \bar{\gamma}_i). \mathbf{q}[\bar{\gamma}_i] : \Gamma. \mathbf{Void}}$$

By assumption, we can directly rewrite above result by

$$\Delta \vdash \bar{\gamma}_i = \gamma \cdot \mathbf{q}[\bar{\gamma}_i] : \Gamma \cdot \mathbf{Void} \quad (**)$$

However,  $\Gamma \cdot \mathbf{Void} \vdash \mathbf{q} : \mathbf{Void}$ . So, we can pull-back  $\mathbf{q}$  through  $\bar{\gamma}_i$ .

$$\frac{\Delta \vdash \bar{\gamma}_i : \Gamma \cdot \mathbf{Void} \quad \Gamma \cdot \mathbf{Void} \vdash \mathbf{q} : \mathbf{Void}}{\Delta \vdash \mathbf{q}[\bar{\gamma}_i] : \mathbf{Void}[\bar{\gamma}_i] = \mathbf{Void}}$$

However, the result of Exercise 2.23. implies that,

$$\Delta \vdash \mathbf{q}[\bar{\gamma}_1] = \mathbf{q}[\bar{\gamma}_2] : \mathbf{Void}$$

We can use this result for (\*\*).

$$\Delta \vdash \bar{\gamma}_1 = \gamma \cdot \mathbf{q}[\bar{\gamma}_1] = \gamma \cdot \mathbf{q}[\bar{\gamma}_2] = \bar{\gamma}_2 : \Gamma \cdot \mathbf{Void}$$

This implies that,

$$\Delta \vdash \bar{\gamma}_1 = \bar{\gamma}_2 : \Gamma \cdot \mathbf{Void}$$

We proved that, if such  $\bar{\gamma}$  exists, then it is unique. This is sufficient to prove that there is at most one such  $\bar{\gamma}$ .

□

### Exercise 3.2.7.

Let  $\Gamma \cdot \mathbf{Void} \vdash A$  type and  $\Gamma \vdash a : A[\mathbf{id}.b]$ . Show that  $\Gamma \cdot \mathbf{Void} \vdash A[(\mathbf{id}.b) \circ \mathbf{p}] = A$  type, and therefore that  $\Gamma \cdot \mathbf{Void} \vdash a[\mathbf{p}] : A$ .

*Proof.* We'll use following previous result.

$$(\gamma.a) \circ \delta = (\gamma \circ \delta).a[\delta]$$

In this exercises, we can write as

$$(\mathbf{id}.b) \circ \mathbf{p} = (\mathbf{id} \circ \mathbf{p}).b[\mathbf{p}]$$

However, by property of  $\mathbf{id}$ ,

$$(\mathbf{id}.b) \circ \mathbf{p} = (\mathbf{id} \circ \mathbf{p}).b[\mathbf{p}] = \mathbf{p}.b[\mathbf{p}] \quad (*)$$

However, our problem gives some pre-supposition :

$$\Gamma \vdash A[\mathbf{id}.b] \text{ type}$$

Which means that

$$\Gamma \vdash \mathbf{id}.b : \Gamma \cdot \mathbf{Void}$$

This also gives pre-supposition :

$$\Gamma \vdash b : \mathbf{Void}$$

So we can pull-back such  $b$  through  $\mathbf{p}$  :

$$\Gamma.\mathbf{Void} \vdash b[\mathbf{p}] : \mathbf{Void}[\mathbf{p}] = \mathbf{Void}$$

However,  $\Gamma.\mathbf{Void} \vdash \mathbf{q} : \mathbf{Void}$  and Exercise 2.23. implies that

$$\Gamma.\mathbf{Void} \vdash b[\mathbf{p}] = \mathbf{q} : \mathbf{Void}$$

Then we can rewrite (\*) by

$$(\mathbf{id}.b) \circ \mathbf{p} = (\mathbf{id} \circ \mathbf{p}).b[\mathbf{p}] = \mathbf{p}.b[\mathbf{p}] = \mathbf{p}.\mathbf{q} = \mathbf{id}$$

in context  $\Gamma.\mathbf{Void}$ . So we can prove our original claim,

$$\Gamma.\mathbf{Void} \vdash A[(\mathbf{id}.b) \circ \mathbf{p}] = A[\mathbf{id}] = A \text{ type}$$

Then the remaining part is easy. Since  $\Gamma \vdash a : A[\mathbf{id}.b]$ , we can pull-back  $a$  into  $\Gamma.\mathbf{Void}$  through  $p$  :

$$\Gamma.\mathbf{Void} \vdash a[\mathbf{p}] : A[\mathbf{id}.b][\mathbf{p}] = A[(\mathbf{id}.b) \circ \mathbf{p}] = A$$

□

### Exercise 3.2.8.

Derive the following rule, using the previous exercise and the  $\eta$ -rule.

$$\frac{\Gamma \vdash b : \mathbf{Void} \quad \Gamma.\mathbf{Void} \vdash A \text{ type} \quad \Gamma \vdash a : A[\mathbf{id}.b]}{\Gamma \vdash a = \mathbf{absurd}(b) : A[\mathbf{id}.b]}$$

*Proof.* We'll use above Exercise 2.25. as lemma. We proved that

$$\frac{\Gamma \vdash b : \mathbf{Void} \quad \Gamma.\mathbf{Void} \vdash A \text{ type} \quad \Gamma \vdash a : A[\mathbf{id}.b]}{\Gamma.\mathbf{Void} \vdash a[\mathbf{p}] : A} \quad (*)$$

And the original  $\eta$ -rule is that

$$\frac{\Gamma \vdash b : \mathbf{Void} \quad \Gamma.\mathbf{Void} \vdash a : A}{\Gamma \vdash \mathbf{absurd}(b) = a[\mathbf{id}.b] : A[\mathbf{id}.b]}$$

To apply this, let's write that

$$\frac{\Gamma \vdash b : \mathbf{Void} \quad \Gamma.\mathbf{Void} \vdash a[\mathbf{p}] : A}{\Gamma \vdash \mathbf{absurd}(b) = a[\mathbf{p}][\mathbf{id}.b] : A[\mathbf{id}.b]}$$

Since (\*) gives the premises, we get

$$\Gamma \vdash \mathbf{absurd}(b) = a[\mathbf{p}][\mathbf{id}.b] : A[\mathbf{id}.b]$$

However,  $a[\mathbf{p}][\mathbf{id}.b] = a[\mathbf{p} \circ (\mathbf{id}.b)] = a[\mathbf{id}] = a$ , we can get

$$\Gamma \vdash \mathbf{absurd}(b) = a : A[\mathbf{id}.b]$$

This is End of Proof. □

### Exercise 3.2.9.

We have included the rule  $\Delta \vdash \mathbf{absurd}(b)[\gamma] = \mathbf{absurd}(b[\gamma]) : A[\gamma.b[\gamma]]$  but it is in fact derivable using the  $\eta$ -rule. Prove this.

*Proof.* Our assumptions here is that :  $\Delta \vdash \gamma : \Gamma$ ,  $\Gamma \vdash b : \mathbf{Void}$ ,  $\Gamma.\mathbf{Void} \vdash A$  type. Let's draw the diagram first.

$$\begin{array}{ccc} \Delta & \xrightarrow{\gamma} & \Gamma \\ & \uparrow p & \\ \Delta.\mathbf{Void} & \xrightarrow{\gamma.\mathbf{Void}} & \Gamma.\mathbf{Void} \end{array}$$

Then, since  $\Gamma \vdash b : \mathbf{Void}$ , we can pull-back into  $\Delta$  :

$$\Delta \vdash b[\gamma] : \mathbf{Void}$$

and since we've defined that  $\Gamma \vdash \mathbf{absurd}(b) : A[\mathbf{id}.b]$ , we can pull-back this into  $\Gamma.\mathbf{Void}$  and again into  $\Delta.\mathbf{Void}$  through given substitutions :

$$\Delta.\mathbf{Void} \vdash \mathbf{absurd}(b)[\mathbf{p}][\gamma.\mathbf{Void}] : A[\gamma.\mathbf{Void}]$$

Now we can use that  $\eta$ -rule.

$$\frac{\Delta \vdash b[\gamma] : \mathbf{Void} \quad \Delta.\mathbf{Void} \vdash \mathbf{absurd}(b)[\mathbf{p}][\gamma.\mathbf{Void}] : A[\gamma.\mathbf{Void}]}{\Delta \vdash \mathbf{absurd}(b[\gamma]) = \mathbf{absurd}(b)[\mathbf{p}][\gamma.\mathbf{Void}][\mathbf{id}.b[\gamma]] : A[\gamma.\mathbf{Void}][\mathbf{id}.b[\gamma]]} (*)$$

However here,

$$\mathbf{p} \circ \gamma.\mathbf{Void} \circ (\mathbf{id}.b[\gamma]) = \mathbf{p} \circ \gamma.b[\gamma] = \gamma$$

and

$$\gamma.\mathbf{Void} \circ \mathbf{id}.b[\gamma] = \gamma.b[\gamma]$$

We can clean up the result of (\*) as :

$$\Delta \vdash \mathbf{absurd}(b[\gamma]) = \mathbf{absurd}(b)[\gamma] : A[\gamma.b[\gamma]]$$

This is end of proof. □

Here, I can construct further realize about this type. As programmer's perspective, we can't reach  $\Gamma.\mathbf{Void}$  context during the run-time. So one function of **Void** is that, represent the unreachable branch (or pattern matching). The another one is, representing negation. For  $\neg A$ , we can construct type  $A \rightarrow \mathbf{Void}$ . This means that, if we have both

term of  $A$  type and  $A \rightarrow \mathbf{Void}$ , then such context is wrong.

### 3.3 Booleans

Almost type system contains Boolean type, which is characterized by 2 elements : true and false. However, we've already talk about true and false. It was **Unit** and **Void** type. Here are somewhat differences between such true, false and boolean's true and false. With **tt**, we can always prove 'true' in any context. With **absurd(–)**, we can prove that current context is wrong. However, **false** term of type **Bool** does not means current context is wrong. **Bool** just give us 'case analysis tool'. And with this 2-ary case analysis tool, we can generate n-ary case analysis tool. (if, else if, else if, ... , else) With general notion of Boolean type, the basic construction is straight-forward and not difficult.

Construction 3.3.1.

$$\frac{\vdash \Gamma \text{ cx}}{\Gamma \vdash \mathbf{Bool} \text{ type}} \quad \frac{\Delta \vdash \gamma : \Gamma}{\Delta \vdash \mathbf{Bool}[\gamma] = \mathbf{Bool} \text{ type}}$$

Construction 3.3.2.

$$\frac{\vdash \Gamma \text{ cx}}{\Gamma \vdash \mathbf{true} : \mathbf{Bool}} \quad \frac{\vdash \Gamma \text{ cx}}{\Gamma \vdash \mathbf{false} : \mathbf{Bool}}$$

$$\frac{\Delta \vdash \gamma : \Gamma}{\Delta \vdash \mathbf{true} = \mathbf{true}[\gamma] : \mathbf{Bool}} \quad \frac{\Delta \vdash \gamma : \Gamma}{\Delta \vdash \mathbf{false} = \mathbf{false}[\gamma] : \mathbf{Bool}}$$

However, does **Bool** actually have 'only' 2 terms? More rigorous question is that, our isomorphism structures can be following?

$$Tm(\Gamma, \mathbf{Bool}) \cong \{\star, \star'\} \quad (\text{THIS IS WRONG !!})$$

Can you imagine that why this is wrong? If we choice this isomorphism to construct, then every **Bool** terms in claim context  $\Gamma$  must be determined by **true** or **false**. i.e. we must always have definitional equality that such claim of **Bool** type term and one of **true**, **false**. However, there are many un-deterministic possible boolean terms. For example, let  $b : \mathbf{Bool}$  is true when Riemann Conjecture is true, otherwise false. If we choice above form for isomorphism, we must determine that  $b$  is true or false. This is why we define **Bool** as inductive type. So as **Void**, we characterize **Bool** via mapping out property. Suppose following characterize target.

$$Tm(\Gamma.\mathbf{Bool}, A) \cong ?$$

If we have term  $a \in Tm(\Gamma.\mathbf{Bool}, A)$ , it works like 'Lazy Evaluation of Boolean value **q**' With respect to our intension of Boolean, there must be exactly 2 ways to 'case analysis' or 'Lazy Evaluate' such **q** both in  $a$  and  $A$ . Just that is role of true and false. So our construction scheme now is 'there are only 2 cases for interpret **q** in  $Tm(\Gamma.\mathbf{Bool}, A)$ .

So one pull-back (evaluation) way for  $a$  is  $\Gamma \vdash a[\text{id.true}] = A[\text{id.true}]$  and another one is that  $\Gamma \vdash a[\text{id.false}] : A[\text{id.false}]$ . We can represent this scheme via following mathematical construction :

Construction 3.3.3.

$$\mathbf{Bool}_\Gamma \in Ty(\Gamma)$$

$$\mathbf{true}_\Gamma, \mathbf{false}_\Gamma \in Tm(\Gamma, \mathbf{Bool})$$

$$((\text{id.true})^*, (\text{id.false})^*) : Tm(\Gamma.\mathbf{Bool}, A) \cong Tm(\Gamma, A[\text{id.true}]) \times Tm(\Gamma, A[\text{id.false}])$$

This isomorphism structure immediately give us that we have only 2 way to evaluating **q** symbols in  $Tm(\Gamma.\mathbf{Bool}, A)$  into previous context  $\Gamma$ . Understanding this story-line is quite important. This also means that, when we consider inductive type's term natural isomorphism, it works like ‘Lazy Evaluation’ of symbol **q** and so, the way to pull-back (evaluate) it is defined via term-constructors for that inductive type. In this case, there was 2 term-constructor for each context, true and false.

Now our job is that, unfold such isomorphism structures into our type system explicitly. For the forward map

$$\iota : Tm(\Gamma.\mathbf{Bool}, A) \xrightarrow{\cong} Tm(\Gamma, A[\text{id.true}]) \times Tm(\Gamma, A[\text{id.false}])$$

Actually we do not need anything new. This direction just be written as :

$$\frac{\Gamma.\mathbf{Bool} \vdash A \text{ type} \quad \Gamma.\mathbf{Bool} \vdash a : A}{\Gamma \vdash a[\text{id.true}] : A[\text{id.true}]} \quad \text{and vice versa}$$

This is already derivable with term-constructing rules. Important one is reverse map.

$$\iota^{-1} : Tm(\Gamma, A[\text{id.true}]) \times Tm(\Gamma, A[\text{id.false}]) \xrightarrow{\cong} Tm(\Gamma.\mathbf{Bool}, A)$$

Here, this has very important intuitive meaning. We'll give 2 terms exists in context  $\Gamma$  each type is  $A[\text{id.true}]$  and  $A[\text{id.false}]$ . Then, our target term is in  $\Gamma.\mathbf{Bool}$  with lazy-evaluate target **q** is also ghostly given from  $\Gamma$ , if such **q** is true then evaluated as first one, if false, then second one. As I wrote above, the main intuition of this direction is ‘dependent if’ type. **Note that, to agree with our usage of if syntax, we'll built-in cut rule here. This will be explained in Exercise of this section.**

Construction 3.3.4.

$$\frac{\Gamma.\mathbf{Bool} \vdash A \text{ type} \quad \Gamma \vdash a_t : A[\text{id.true}] \quad \Gamma \vdash a_f : A[\text{id.false}] \quad \Gamma \vdash b : \mathbf{Bool}}{\Gamma \vdash \mathbf{if}(a_t, a_f, b) : A[\text{id.b}]}$$

$$\Delta \vdash \gamma : \Gamma$$

$$\frac{\Gamma.\mathbf{Bool} \vdash A \text{ type} \quad \Gamma \vdash a_t : A[\text{id.true}] \quad \Gamma \vdash a_f : A[\text{id.false}] \quad \Gamma \vdash b : \mathbf{Bool}}{\Delta \vdash \mathbf{if}(a_t, a_f, b)[\gamma] = \mathbf{if}(a_t[\gamma], a_f[\gamma], b[\gamma]) : A[\gamma.b[\gamma]]}$$

$$\begin{array}{c}
 \frac{\Gamma.\text{Bool} \vdash A \text{ type} \quad \Gamma \vdash a_t : A[\text{id.true}] \quad \Gamma \vdash a_f : A[\text{id.false}]}{\Gamma \vdash \text{if}(a_t, a_f, \text{true}) = a_t : A[\text{id.true}] \quad \Gamma \vdash \text{if}(a_t, a_f, \text{false}) = a_f : A[\text{id.false}]} \\
 \frac{\Gamma.\text{Bool} \vdash A \text{ type} \quad \Gamma.\text{Bool} \vdash a : A \quad \Gamma \vdash b : \text{Bool}}{\Gamma \vdash \text{if}(a[\text{id.true}], a[\text{id.false}], b) = a[\text{id.b}] : A[\text{id.b}]} \tag{*}
 \end{array}$$

We'll see (\*) in post section again.

Exercise 3.3.5.

**Exercise 2.29.** Give rules axiomatizing the boolean analogue of **absurd'**, and prove that these rules are interderivable with our rules for  $\text{if}(a_t, a_f, b)$ .

*Proof.* This is also question about ‘cut-rule applied’ version of our backward homomorphism  $\iota^{-1}$ . Since

$$\iota^{-1} : Tm(\Gamma, A[\text{id.true}]) \times Tm(\Gamma, A[\text{id.false}]) \rightarrow Tm(\Gamma.\text{Bool}, A)$$

Then when we directly construct this homomorphism, it will be

$$\begin{array}{c}
 \frac{\Gamma.\text{Bool} \vdash A \text{ type} \quad \Gamma \vdash a_t : A[\text{id.true}] \quad \Gamma \vdash a_f : A[\text{id.false}]}{\Gamma.\text{Bool} \vdash \text{if}'(a_t, a_f) : A}
 \end{array}$$

With above definition, the remaining modified rules are simply constructed :

$$\begin{array}{c}
 \frac{\Delta \vdash \gamma : \Gamma \quad \Gamma.\text{Bool} : A \text{ type} \quad \Gamma \vdash a_t : A[\text{id.true}] \quad \Gamma \vdash a_f : A[\text{id.false}]}{\Delta.\text{Bool} \vdash \text{if}'(a_t, a_f)[\gamma.\text{Bool}] = \text{if}'(a_t[\gamma], a_f[\gamma]) : A[\gamma.\text{Bool}]}
 \end{array}$$

$$\begin{array}{c}
 \frac{\Gamma.\text{Bool} \vdash A \text{ type} \quad \Gamma \vdash a_t : A[\text{id.true}] \quad \Gamma \vdash a_f : A[\text{id.false}]}{\Gamma \vdash \text{if}'(a_t, a_f)[\text{id.true}] = a_t : A[\text{id.true}]}
 \end{array}$$

$$\begin{array}{c}
 \frac{\Gamma.\text{Bool} \vdash A \text{ type} \quad \Gamma \vdash a_t : A[\text{id.true}] \quad \Gamma \vdash a_f : A[\text{id.false}]}{\Gamma \vdash \text{if}'(a_t, a_f)[\text{id.false}] = a_f : A[\text{id.false}]}
 \end{array}$$

$$\begin{array}{c}
 \frac{\Gamma.\text{Bool} \vdash A \text{ type} \quad \Gamma.\text{Bool} \vdash a : A}{\Gamma.\text{Bool} \vdash \text{if}'(a[\text{id.true}], a[\text{id.false}]) = a : A}
 \end{array}$$

However, our rules for **if** is constructed on context  $\Gamma$ . Actually, as same we discussed, it can be imagined as pull-backed **if'** into  $\Gamma$ . Here, we can write

$$\begin{array}{c}
 \frac{\Gamma.\text{Bool} \vdash A \text{ type} \quad \Gamma \vdash a_t : A[\text{id.true}] \quad \Gamma \vdash a_f : A[\text{id.false}] \quad \Gamma \vdash b : \text{Bool}}{\Gamma \vdash \text{if}'(a_t, a_f)[\text{id.b}] := \text{if}(a_t, a_f, b) : A[\text{id.b}]}
 \end{array}$$

Now, let's prove our original rules for **if** implies above  $\beta, \eta$ -rules. ( Here, I'll omit some premises when they are clear ) We have

$$\Gamma \vdash \mathbf{if}(a_t, a_f, \mathbf{true}) = a_t : A[\mathbf{id.true}]$$

Rewrite this :

$$\Gamma \vdash \mathbf{if}'(a_t, a_f)[\mathbf{id.true}] = a_t : A[\mathbf{id.true}]$$

Similarly,

$$\Gamma \vdash \mathbf{if}'(a_t, a_f)[\mathbf{id.false}] = a_f : A[\mathbf{id.false}]$$

This directly proves above  $\beta$ -rules. For  $\eta$ -rule, we can imagine following diagram :

$$\Gamma \xleftarrow[p]{} \Gamma.\mathbf{Bool} \xrightleftharpoons[p.\mathbf{Bool}]{id.q} \Gamma.\mathbf{Bool.Bool}$$

How can we use above intuition? Let's see. The original  $\eta$ -rule was

$$\frac{\Gamma.\mathbf{Bool} \vdash A \text{ type} \quad \Gamma.\mathbf{Bool} \vdash a : A \quad \Gamma \vdash b : \mathbf{Bool}}{\Gamma \vdash \mathbf{if}(a[\mathbf{id.true}], a[\mathbf{id.false}], b) = a[\mathbf{id.b}] : A[\mathbf{id.b}]}$$

To prove the  $\eta$ -rule for  $\mathbf{if}'$  of us, we'll use that

$$\frac{\Delta \vdash \gamma : \Gamma \quad \Gamma.\mathbf{Bool} : A \text{ type} \quad \Gamma \vdash a_t : A[\mathbf{id.true}] \quad \Gamma \vdash a_f : A[\mathbf{id.false}]}{\Delta.\mathbf{Bool} \vdash \mathbf{if}'(a_t, a_f)[\gamma.\mathbf{Bool}] = \mathbf{if}'(a_t[\gamma], a_f[\gamma]) : A[\gamma.\mathbf{Bool}]}$$

We can write this as

$$\frac{\begin{array}{c} \Gamma.\mathbf{Bool} \vdash p : \Gamma \quad \Gamma.\mathbf{Bool} \vdash a : A \\ \Gamma \vdash a[\mathbf{id.true}] : A[\mathbf{id.true}] \quad \Gamma \vdash a[\mathbf{id.false}] : A[\mathbf{id.false}] \end{array}}{\begin{array}{c} \Gamma.\mathbf{Bool.Bool} \vdash \mathbf{if}'(a[\mathbf{id.true}], a[\mathbf{id.false}])[p.\mathbf{Bool}] \\ = \mathbf{if}'(a[\mathbf{id.true}][p], a[\mathbf{id.false}][p]) : A[p.\mathbf{Bool}] \end{array}}$$

We can pull-back the result using  $\Gamma.\mathbf{Bool} \vdash q : \mathbf{Bool}$ .

$$\begin{aligned} & \Gamma.\mathbf{Bool} \vdash \mathbf{if}'(a[\mathbf{id.true}], a[\mathbf{id.false}])[p.\mathbf{Bool}][\mathbf{id.q}] \\ &= \mathbf{if}'(a[\mathbf{id.true}][p], a[\mathbf{id.false}][p])[\mathbf{id.q}] : A[p.\mathbf{Bool}][\mathbf{id.q}] \end{aligned}$$

Rewrite this as

$$\Gamma.\mathbf{Bool} \vdash \mathbf{if}'(a[\mathbf{id.true}], a[\mathbf{id.false}]) = \mathbf{if}(a[\mathbf{id.true}][p], a[\mathbf{id.false}][p], q) : A$$

However, we already know that  $\mathbf{id.true} \circ p = p.A \circ \mathbf{id.true}'$  where  $\mathbf{id.true}$  is arrow  $\Gamma.\mathbf{Bool} \rightarrow \Gamma.\mathbf{Bool.Bool}$ . Then ,

$$\Gamma.\mathbf{Bool} \vdash \mathbf{if}'(a[\mathbf{id.true}], a[\mathbf{id.false}]) = \mathbf{if}(a[p.A][\mathbf{id.true}'], a[p.A][\mathbf{id.false}'], q) : A$$

However, the  $\eta$ -rule of given **if** on  $\Gamma.\text{Bool}$  context implies that

$$\begin{aligned} & \mathbf{if}'(a[\mathbf{id}.\mathbf{true}], a[\mathbf{id}.\mathbf{false}]) \\ &= \mathbf{if}(a[\mathbf{p}.A][\mathbf{id}.\mathbf{true}'], a[\mathbf{p}.A][\mathbf{id}.\mathbf{false}'], \mathbf{q}) \\ &= a[\mathbf{p}.A][\mathbf{id}.\mathbf{q}] : A \end{aligned}$$

Then finally, we can holds

$$\Gamma.\text{Bool} \vdash \mathbf{if}'(a[\mathbf{id}.\mathbf{true}], a[\mathbf{id}.\mathbf{false}]) = a : A$$

This is proof of  $\eta$ -rule for **if'**. □

## 3.4 Natural Number

Natural number is actually, most important and intuitive type that we need. However, I think that this also most challenging one to understand. Before start construction, I'll introduce the notion of natural number in ‘logic’ definitions. In the logic courses, they introduce natural number as the **least** recursively defined set  $\mathbb{N}$ , with **zero** and  $\text{suc}(-) : \mathbb{N} \rightarrow \mathbb{N}$ . We'll try to characterize this set in our type system. (as we've seen before, we internalize the intuitive external structure into our type system via follow-up the characterization of external structure into type system. ) First, without the **least** characterization, the characterization of such recursively defined set is easy.

Construction 3.4.1.

$$\frac{}{\Gamma \vdash \mathbf{Nat} \text{ type}} \quad \frac{}{\Gamma \vdash \mathbf{zero} : \mathbf{Nat}} \quad \frac{\Gamma \vdash n : \mathbf{Nat}}{\Gamma \vdash \mathbf{suc}(n) : \mathbf{Nat}}$$

(Actually, this is similar to define  $\mathbb{N}$  is a set, such that  $\mathbf{zero} \in \mathbb{N}, \forall n \in \mathbb{N}, \mathbf{suc}(n) \in \mathbb{N}$ . )

$$\frac{\Delta \vdash \gamma : \Gamma}{\Delta \vdash \mathbf{Nat}[\gamma] = \mathbf{Nat} : \Gamma} \quad \frac{\Delta \vdash \gamma : \Gamma}{\Delta \vdash \mathbf{zero}[\gamma] = \mathbf{zero} : \mathbf{Nat}}$$

$$\frac{\Delta \vdash \gamma : \Gamma \quad \Gamma \vdash n : \mathbf{Nat}}{\Delta \vdash \mathbf{suc}(n)[\gamma] = \mathbf{suc}(n[\gamma]) : \mathbf{Nat}}$$

However, everything is now starting. As we've seen in previous section **Bool**, reader will immediately capture that we need Lazy Evaluation. It's because of decidability, if we try to capture natural number as  $Tm(\Gamma, \mathbf{Nat}) \cong \mathbb{N}$ , then our type checker can detect ‘divide by zero’ which is undecidable problem. So to ensure our type checker decidable, we also choose Lazy Evaluation scheme for **Nat** also. In this point, as same with before, our first trial will be following.

Wrong Construction 3.4.2.

Oh, as same with **Bool**, the term **q** in  $\Gamma.\mathbf{Nat}$  can be interpreted into  $\Gamma$  via **zero** or **suc** of something. Then construct each mapping as :

$$(\mathbf{id.zero}^*) : Tm(\Gamma.\mathbf{Nat}, A) \rightarrow Tm(\Gamma, A[\mathbf{id.zero}])$$

$$(\mathbf{p.suc(q)}^*) : Tm(\Gamma.\mathbf{Nat}, A) \rightarrow Tm(\Gamma.\mathbf{Nat}, A[\mathbf{p.suc(q)}])$$

Below one change all appearance of **q** in original  $A$  into **suc(q)**. So the context does not be changed. Then finally our natural isomorphism structure is :

$$((\mathbf{id.zero})^*, (\mathbf{p.suc(q)})^*) :$$

$$Tm(\Gamma.\mathbf{Nat}, A) \cong Tm(\Gamma, A[\mathbf{id.zero}]) \times Tm(\Gamma.\mathbf{Nat}, A[\mathbf{p.suc(q)}])$$

However, it is not enough to characterize the **least** property. It means that, can you conclude that above **Nat** is exactly same with  $\mathbb{N} = \{0, 1, 2, \dots\}$ ? Actually, you can't. Many other larger sets can holds above isomorphism. Suppose that our **Nat** is  $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$ , above isomorphism also holds. Moreover,  $\mathbb{R}_{\geq 0}$  also holds. To match **Nat** with  $\mathbb{N}$ , one we must do is obtain **least** property. It can be understood in Algebraic Perspective. First, we can write **equation** that determine our **Nat** type as follow.

$$N \cong \{\star\} \sqcup N$$

The form of solution of this equation is triple :  $(N, \text{zero}, \text{suc}(-))$ . Can you see that this equation is Algebraic presentation of wrong const 3.4.2.? As I wrote, there can be many solutions for above algebraic equation. Our intended job is distinguish the solution  $(\mathbb{N}, 0, \_ + 1)$ , which is meta-level natural number with another solutions and internalize it into our type system. See following Algebraic observations.

#### Observation 3.4.3.

Let  $(\mathbb{N}, 0, \_ + 1)$  be our targeted solution. And there are many another solutions, we represents  $(N, z, s)$ . Then one can imagine the *homomorphism* from  $(\mathbb{N}, 0, \_ + 1)$  to  $(N, z, s)$ . Which intuitively means that, Algebra interpreter from  $\mathbb{N}$  to  $N$ . Suppose a map  $f : \mathbb{N} \rightarrow N$  such that  $f(0) = z$ ,  $f(n + 1) = s(f(n))$ . Then here, you know that  $f$  preserves the Algebraic property, i.e. give meaning 0 to  $z$  and  $\_ + 1$  to  $s$ . In mathematics' terminology,  $(N, z, s)$  triple is called algebras for the signature  $N \rightarrowtail \{\star\} \sqcup N$ . And such structure preserving function between algebras  $f$  is called *algebra homomorphism*. And such **least** algebra  $(\mathbb{N}, 0, \_ + 1)$  is called *initial algebra*. One important fact is that, we can **characterize such initial algebra via following fact** : There are unique *homomorphism* from initial algebra to another algebras.

What can we do here is that, (1) extends this into dependent Algebra, (2) internalize the unique homomorphism characterization into our type system. To understand this, first we must know what is **Dependent Algebra**? and why we need them? **Dependent Algebra** is very similar to our dependent type system. Suppose the **Vec A n** example again, this algebra structure is represented by ‘indexed family’ of **Nat**.

$$\mathbf{Vec}\ A\ 0 = \emptyset$$

$$\mathbf{Vec}\ A\ 1 = \{a \mid a \in Tm(\Gamma, A)\}$$

⋮

$$\mathbf{Vec}\ A\ \mathbf{suc}(n) = \{v :: a \mid v \in Tm(\Gamma, \mathbf{Vec}\ A\ n), a \in Tm(\Gamma, A)\}$$

Here, the algebraic structure of  $(\mathbf{Nat}, 0, \mathbf{suc}(-))$  is hidden. This is example of dependent algebra, and we call that *displayed algebra over*  $(\mathbf{Nat}, 0, \mathbf{suc}(-))$ . More formally, the definition is following.

**Definition 3.4.4.**

The *displayed algebra over*  $(N, z, s)$  (here, this is algebra of the signature  $N \rightarrow \{\star\} \sqcup N$ ) is defined by triple :

$$(\{\tilde{X}_n\}_{n \in N}, \tilde{z} \in \tilde{X}_n, \tilde{s} : (n : N) \rightarrow \tilde{X}_n \rightarrow \tilde{X}_{s(n)})$$

Suppose the initial algebra  $(\mathbb{N}, 0, - + 1)$ . And we construct any displayed algebra

$$(\{\tilde{X}_n\}_{n \in \mathbb{N}}, \tilde{z} \in \tilde{X}_0, \tilde{s} : (n : \mathbb{N}) \rightarrow \tilde{X}_n \rightarrow \tilde{X}_{n+1})$$

The **initial** condition here is that, there is unique algebra homomorphism

$$f : (n : \mathbb{N}) \rightarrow \tilde{X}_n$$

such that  $f(0) = \tilde{z}$  and  $f(n + 1) = \tilde{s}(n, f(n))$

Actually, the role of  $f$  here is directly proof of for all  $n \in \mathbb{N}$  statements. Because, we can imagine that such displayed algebra is given by propositions indexed by  $\mathbb{N}$ .

Now it's time to internalize this notion into our type system. We'll axiomatize above initiality which is described in defn 3.4.4. It means that, our natural number algebra  $(\mathbf{Nat}, \mathbf{zero}, \mathbf{suc}(-))$  in our type system holds that, for all displayed algebra over  $(\mathbf{Nat}, \mathbf{zero}, \mathbf{suc}(-))$ , there exists unique homomorphism from  $(\mathbf{Nat}, \mathbf{zero}, \mathbf{suc}(-))$ . Now, very challenging notions is upcoming. We'll imagine the ‘displayed algebra’ is

$$A \in Ty(\Gamma.\mathbf{Nat})$$

We can understand that this is a kind of ‘family of terms with type  $A$ ,  $\mathbf{q}$  in  $A$  is indexed by  $\mathbf{Nat}$ ’. Then for any context  $\Gamma$ , we can construct displayed algebra over  $(\mathbf{Nat}, \mathbf{zero}, \mathbf{suc}(-))$  by

$$(A \in Ty(\Gamma.\mathbf{Nat}), a_z \in Tm(\Gamma, A[\mathbf{id.zero}]), a_s \in Tm(\Gamma.\mathbf{Nat}.A, A[p^2.\mathbf{suc}(\mathbf{q}[p])]))$$

We can understand that above as propositional perspective.  $A$  is  $\mathbf{q}$ -indexed proposition. And  $a_z$  is zero-index instance(proof). challenging one is  $a_s$ , why this represents successor? See following diagram first.

$$\begin{array}{ccc} \Gamma & \xleftarrow{p} & \Gamma.\mathbf{Nat} \\ \uparrow p^2 & \nearrow p & \\ \Gamma.\mathbf{Nat}.A & & \end{array} \implies \begin{array}{ccc} \Gamma & & \Gamma.\mathbf{Nat} \\ \uparrow p^2 & & \\ \Gamma.\mathbf{Nat}.A & \nearrow p^2.\mathbf{suc}(\mathbf{q}[p]) & \end{array}$$

Above arrow is constructed in this way. Let's unpack the meaning of  $a_s$  from now on. First, see  $A$  type in  $\Gamma.\mathbf{Nat}$  context. We already see that it's like a indexed proposition by  $\mathbf{q}$ . However we first extend context into  $\Gamma.\mathbf{Nat}.A$ , this situation means that we have  $\mathbf{q}[p] : \mathbf{Nat}$ , which is fixed index  $n : \mathbf{Nat}$  and **proof  $\mathbf{q}$  for  $A$  indexed by  $\mathbf{q}[p]$** (or,  $n$ ). (More exactly, the  $\mathbf{q}$  appeared in  $A$  is actually  $\mathbf{q}[p]$ ). Important one is understanding  $p^2.\mathbf{suc}(\mathbf{q}[p])$ . It is constructed via substitution extending.

$$\frac{\Gamma.\mathbf{Nat}.A \vdash p^2 : \Gamma \quad \Gamma \vdash \mathbf{Nat} \text{ type} \quad \Gamma.\mathbf{Nat}.A \vdash \mathbf{suc}(\mathbf{q}[p]) : \mathbf{Nat}}{\Gamma.\mathbf{Nat}.A \vdash p^2.\mathbf{suc}(\mathbf{q}[p]) : \Gamma.\mathbf{Nat}}$$

So this arrow directly means that, when we pull-back from  $\Gamma.\mathbf{Nat}$  world to  $\Gamma.\mathbf{Nat}.A$ , all the appearance of  $\mathbf{q}$  in  $\Gamma.\mathbf{Nat}$ , we'll take it into  $\mathbf{suc}(\mathbf{q}[\mathbf{p}])$ . Then finally,  $A[\mathbf{p}^2.\mathbf{suc}(\mathbf{q}[\mathbf{p}])]$  is proposition  $A$ , which is indexed by  $\mathbf{suc}(\mathbf{q}[\mathbf{p}])$ . So if the term  $a_s$  of such type exists, then it works like dependent function whose type is following :

$$\tilde{s} : (n : \mathbf{Nat}) \rightarrow \tilde{A}_n \rightarrow \tilde{A}_{\mathbf{suc}(n)}$$

In this understanding, we can realize that  $a_s$  itself works like dependent function  $\tilde{s}$ . Our final job is claim explicitly  $(\mathbf{Nat}, 0, \mathbf{suc}(-))$  is initial algebra. In the textbook, such initial property is described by

$$\rho_{\Gamma, A, a_z, a_s} : \{a \in Tm(\Gamma.\mathbf{Nat}, A) | a_z = a[\mathbf{id.zero}] \wedge a_s[\mathbf{p.q.a}] = a[\mathbf{p.suc(q)}]\} \cong \{\star\}$$

In our language, I think that ‘Every proof by mathematical induction is unique’. Poetically, natural number is origin(initial algebra) of whole mathematical induction and so homomorphism is unique.

Let's take a look the explicit rule version.

Construction 3.4.5.

**rec** is pull-backed above singleton  $a$ .

$$\frac{\begin{array}{c} \Gamma \vdash n : \mathbf{Nat} \\ \Gamma.\mathbf{Nat} \vdash A \text{ type} \quad \Gamma \vdash a_z : A[\mathbf{id.zero}] \quad \Gamma.\mathbf{Nat}.A \vdash a_s : A[\mathbf{p}^2.\mathbf{suc}(\mathbf{q}[\mathbf{p}])] \end{array}}{\Gamma \vdash \mathbf{rec}(a_z, a_s, n) : A[\mathbf{id.n}]}$$

For naturalism,

$$\frac{\begin{array}{c} \Delta \vdash \gamma : \Gamma \quad \Gamma \vdash n : \mathbf{Nat} \\ \Gamma.\mathbf{Nat} \vdash A \text{ type} \quad \Gamma \vdash a_z : A[\mathbf{id.zero}] \quad \Gamma.\mathbf{Nat}.A \vdash a_s : A[\mathbf{p}^2.\mathbf{suc}(\mathbf{q}[\mathbf{p}])] \end{array}}{\Delta \vdash \mathbf{rec}(a_z, a_s, n)[\gamma] = \mathbf{rec}(a_z[\gamma], a_s[\gamma.\mathbf{Nat}.A], n[\gamma]) : A[\gamma.n[\gamma]]}$$

And we now gives the explicit description about this displayed algebraic type by rules.

$$\frac{\Gamma.\mathbf{Nat} \vdash A \text{ type} \quad \Gamma \vdash a_z : A[\mathbf{id.zero}] \quad \Gamma.\mathbf{Nat}.A \vdash a_s : A[\mathbf{p}^2.\mathbf{suc}(\mathbf{q}[\mathbf{p}])]}{\Gamma \vdash \mathbf{rec}(a_z, a_s, \mathbf{zero}) = a_z : A[\mathbf{id.zero}]}$$

For successor,

$$\frac{\begin{array}{c} \Gamma \vdash n : \mathbf{Nat} \\ \Gamma.\mathbf{Nat} \vdash A \text{ type} \quad \Gamma \vdash a_z : A[\mathbf{id.zero}] \quad \Gamma.\mathbf{Nat}.A \vdash a_s : A[\mathbf{p}^2.\mathbf{suc}(\mathbf{q}[\mathbf{p}])] \end{array}}{\Gamma \vdash \mathbf{rec}(a_z, a_s, \mathbf{suc}(n)) = a_s[\mathbf{id.n.rec}(a_z, a_s, n)[\mathbf{p}]] : A[\mathbf{id.suc}(n)]}$$

One fun thing is that, the last rule directly show us that how the term  $a_s$  works like proof successor here. That is proof of  $A[\mathbf{p}^2.\mathbf{suc}(\mathbf{q}[\mathbf{p}])]$  in context  $\Gamma.\mathbf{Nat}.A$ , so when we take it

back to  $\Gamma$  with substitution  $\mathbf{id}.n.\mathbf{rec}(a_z, a_s, n)[\mathbf{p}]$ , it become a proof of

$$\begin{aligned}
 & A \underbrace{[\mathbf{p}^2.\mathbf{suc}(\mathbf{q}[\mathbf{p}])]}_{\gamma.a} \underbrace{[\mathbf{id}.n.\mathbf{rec}(a_z, a_s, n)[\mathbf{p}]]}_{\delta} \\
 &= A[(\gamma \circ \delta).a[\delta]] \\
 &= A[(\mathbf{p}^2 \circ \mathbf{id}.n.\mathbf{rec}(a_z, a_s, n)[\mathbf{p}]).\mathbf{suc}(\mathbf{q}[\mathbf{p}])[\mathbf{id}.n.\mathbf{rec}(a_z, a_s, n)[\mathbf{p}]]] \\
 &= A[\mathbf{id}.\mathbf{suc}(n)]
 \end{aligned}$$

I hope that this is helpful for understanding the last rule and working of  $a_s$ .

Finally, we need  $\eta$ -rule for the uniqueness property. However, our textbook said that it is typically omitted. The rule-form is that,

$$\frac{\dots, \Gamma.Nat \vdash a_s[\mathbf{p}.\mathbf{q}.a] = a[\mathbf{p}.\mathbf{suc}(\mathbf{q})] : A[\mathbf{p}.\mathbf{suc}(\mathbf{q})]}{\Gamma \vdash \mathbf{rec}(a_z, a_s, n) = a[\mathbf{id}.n] : A[\mathbf{id}.n]}$$

Exercise 3.4.1.

Argue the ‘Algebraic’ perspective for the **Bool** type we’ve learned.

*Proof.* The target initial algebra is 2-ton set. It can be described by

$$\text{Initial Algebra} : (\mathbb{B} = \{\text{true}, \text{false}\}, \text{true}, \text{false})$$

Then, by same perspective, the any displayed algebra will be

$$\text{Displayed Algebra} : (\{\tilde{X}_t, \tilde{X}_f\}, \tilde{t} \in \tilde{X}_t, \tilde{f} \in \tilde{X}_f)$$

And the Homomorphism will be :

$$Hom : (b : \mathbb{B}) \rightarrow \tilde{X}_b$$

Then all things for argue is that, such Homomorphism is unique.

$$\rho_{\Gamma, A, a_t, a_f} := \{a \in Tm(\Gamma.\mathbf{Bool}, A) | a[\mathbf{id}.\mathbf{true}] = a_t, a[\mathbf{id}.\mathbf{false}] = a_f\} \cong \{\star\}$$

□

# Chapter 4

## Further topics of Chapter 3

### 4.1 Uniqueness and Extensional Equality

I'll try to note the ‘story’ for this section clearly. First, we introduced  $\eta$ -rule for dependent type when we **explicitly mention the uniqueness of homomorphism**. Following two facts are turns out that, (1) Such uniqueness is too strong (2) Such uniqueness on inductive type is derivable with extensional equality, i.e. **Eq** type. First one implies that, it can make our proof system ‘inconsistent’ or our prover ‘undecidable’. Second one implies that, we actually doesn’t need  $\eta$ -rules for ‘inductive types’. So, our story line is : In this chapter, we’ll show why (2) holds. And in post chapters, we’ll show implementation perspective, why (1) holds and what is the alternative way ( Intensional Type Theory ). Intuitively, think that if we accept extensional equality, then the quotient of expression will be too large, and it leads us to inconsistent problems. Let’s see how (2) : Uniqueness (  $\eta$ -rules ) for inductive types can be derived from extensional equalities ?

Theorem 4.1.1.

Uniqueness of **Void**, i.e.  $\eta$ -rule for **Void** can be derived with rules for **Eq** type.

$$\frac{\Gamma \vdash b : \mathbf{Void} \quad \Gamma. \mathbf{Void} \vdash a : A}{\Gamma \vdash \mathbf{absurd}(b) = a[\mathbf{id}.b] : A[\mathbf{id}.b]}$$

*Proof.* First,

$$\frac{\Gamma \vdash b : \mathbf{Void} \quad \Gamma. \mathbf{Void} \vdash A \text{ type}}{\Gamma \vdash \mathbf{absurd}(b) : A[\mathbf{id}.b]}$$

Second,

$$\frac{\Gamma \vdash b : \mathbf{Void} \quad \Gamma. \mathbf{Void} \vdash a : A}{\Gamma \vdash a[\mathbf{id}.b] : A[\mathbf{id}.b]}$$

Third, above 2 results implies that

$$\Gamma \vdash \mathbf{Eq}(A[\mathbf{id}.b], \mathbf{absurd}(b), a[\mathbf{id}.b]) \text{ type}$$

This is similar job with ‘claim’ a proving statements. However,

$$\Gamma. \mathbf{Void} \vdash \mathbf{Eq}(A[\mathbf{id}.b], \mathbf{absurd}(b), a[\mathbf{id}.b])[p] \text{ type}$$

Then, this again implies that

$$\frac{\Gamma \vdash b : \mathbf{Void} \quad \Gamma.\mathbf{Void} \vdash \mathbf{Eq}(A[\mathbf{id}.b], \mathbf{absurd}(b), a[\mathbf{id}.b])[p] \text{ type}}{\Gamma \vdash \mathbf{absurd}(b) : \mathbf{Eq}(A[\mathbf{id}.b], \mathbf{absurd}(b), a[\mathbf{id}.b])}$$

This is similar with **sorry** in Lean 4, with  $b$  we can prove any statements here. Finally, this Extensional equality directly implies that

$$\Gamma \vdash \mathbf{absurd}(b) = a[\mathbf{id}.b] : A[\mathbf{id}.b]$$

□

The after stories are what I mentioned. Sometimes, the  $\eta$ -rule and external equality are too strong. So during remain chapters, our job will be alternative ways. One example is Intensional Type Theory, where they does not explicitly contain such ‘uniqueness’ of inductive type. In other words, in there, we remove any  $\eta$ -rules for inductive types so we can’t argue our defined type should be target Initial Algebra.

**Exercise 4.1.2.**

Prove that  $\eta$ -rule for **Bool** type can be derived with extensional equalities.

*Proof.* The  $\eta$ -rule for **Bool** type is,

$$\frac{\Gamma \vdash b : \mathbf{Bool} \quad \Gamma.\mathbf{Bool} \vdash a : A}{\Gamma \vdash \mathbf{if}(a[\mathbf{id}.true], a[\mathbf{id}.false], b) = a[\mathbf{id}.b] : A[\mathbf{id}.b]}$$

First, understanding following diagram.

$$\Gamma \xleftarrow[p]{} \Gamma.\mathbf{Bool} \xrightleftharpoons[p.\mathbf{Bool}]{} \Gamma.\mathbf{Bool}.\mathbf{Bool}$$

And here, claim that

$$\Gamma \vdash \mathbf{Eq}(A[\mathbf{id}.b], \mathbf{if}(a[\mathbf{id}.true], a[\mathbf{id}.false], b), a[\mathbf{id}.b]) \text{ type}$$

We can easily know that this is well-defined type. However, the  $\beta$ -rules for **Bool** type gives us

$$\Gamma \vdash \mathbf{rfl} : \mathbf{Eq}(A[\mathbf{id}.true], \mathbf{if}(a[\mathbf{id}.true], a[\mathbf{id}.false], \mathbf{true}), a[\mathbf{id}.true])$$

$$\Gamma \vdash \mathbf{rfl}' : \mathbf{Eq}(A[\mathbf{id}.false], \mathbf{if}(a[\mathbf{id}.true], a[\mathbf{id}.false], \mathbf{false}), a[\mathbf{id}.false])$$

Then, we’ll show that

$$\Gamma.\mathbf{Bool} \vdash \underbrace{\mathbf{Eq}(A, \mathbf{if}(a[\mathbf{id}.true][p], a[\mathbf{id}.false][p], q), a)}_B \text{ type}$$

By the way, if this holds, then

$$\Gamma \vdash \mathbf{if}(\mathbf{rfl}, \mathbf{rfl}', b) : B[\mathbf{id}.b] = \mathbf{Eq}(A[\mathbf{id}.b], \mathbf{if}(a[\mathbf{id}.true], a[\mathbf{id}.false], b), a[\mathbf{id}.b])$$

So it proves our original claim. Then the remaining is that,  $B$  is well-formed type of context  $\Gamma.\text{Bool}$ . For the sake, since  $a : A$  is given, we'll show that

$$\Gamma.\text{Bool} \vdash \text{if}(a[\text{id.true}][\mathbf{p}], a[\text{id.false}][\mathbf{p}], \mathbf{q}) : A$$

It is not difficult to show because,

$$\begin{aligned}\Gamma.\text{Bool} &\vdash a[\text{id.true}][\mathbf{p}] \\ &: A[\text{id.true}][\mathbf{p}] \\ &= A[\mathbf{p.true}] \\ &= A[\mathbf{p.Bool}][\text{id.true}]\end{aligned}$$

Similarly,

$$\Gamma.\text{Bool} \vdash a[\text{id.false}][\mathbf{p}] = A[\mathbf{p.Bool}][\text{id.false}]$$

Then, by elimination rule of **Bool**,

$$\Gamma.\text{Bool} \vdash \text{if}(a[\text{id.true}][\mathbf{p}], a[\text{id.false}][\mathbf{p}], \mathbf{q}) : A[\mathbf{p.Bool}][\text{id.q}] = A$$

This is end of proof. □

## 4.2 Large Elimination

By the way, we learned pretty lots of things about dependent types. Is it really satisfiable? Can we do something else for dependent types? The answer is trivially yes. Although we made types depends on terms, we did not make anythings for dependency of type constructors. For example, one can be imagined :

$$\mathbf{UnitOrBool}_{\Gamma}(b) = \begin{cases} \mathbf{Unit} & b = \mathbf{true} \\ \mathbf{Bool} & b = \mathbf{false} \end{cases}$$

With rules in chapter 3, we can't implement such type formers. This kind of dependency is called full-spectrum dependency. Textbook said that there are 2 way to achieve such full-spectrum dependency.

1. Large Elimination : Not popular because it make type system too strong.
2. Universes : Popular scheme for these days.

In this section, let's learn about what Large Elimination is. It is implementation of what I mentioned.

### Intro 4.2.1.

Each inductive types can equip Large Elimination. We lazy evaluate (or lazy pull-back) terms in context  $\Gamma.X$  into  $\Gamma$ . Large Elimination is just implementation this for types. We'll take types in  $\Gamma.X$  into  $\Gamma$ . In this section, we'll focus on Large Elimination for **Bool** types. The idea is that, take terms in  $\Gamma.\mathbf{Bool}$  with  $\Gamma \vdash b : \mathbf{Bool}$ , which can be **true** or **false**!

### Construction 4.2.2.

$$\frac{\Gamma \vdash A_t \text{ type} \quad \Gamma \vdash A_f \text{ type} \quad \Gamma \vdash b : \mathbf{Bool}}{\Gamma \vdash \mathbf{If}(A_t, A_f, b) \text{ type}}$$

With Natural condition :

$$\frac{\Delta \vdash \gamma : \Gamma \quad \Gamma \vdash A_t \text{ type} \quad \Gamma \vdash A_f \text{ type} \quad \Gamma \vdash b : \mathbf{Bool}}{\Delta \vdash \mathbf{If}(A_t, A_f, b)[\gamma] = \mathbf{If}(A_t[\gamma], A_f[\gamma], b[\gamma]) \text{ type}}$$

With  $\beta$ -rules :

$$\frac{\Gamma \vdash A_t \text{ type} \quad \Gamma \vdash A_f \text{ type}}{\Gamma \vdash \mathbf{If}(A_t, A_f, \mathbf{true}) = A_t \text{ type} \quad \Gamma \vdash \mathbf{If}(A_t, A_f, \mathbf{false}) = A_f \text{ type}}$$

However, there are also  $\eta$ -rule here. But we show that such inductive  $\eta$ -rules almost always omitted since it make our type system too strong, and so undecidable. Same for here. Following  $\eta$ -rule for **If** almost always non-included in our system.

$$\frac{\Gamma.\mathbf{Bool} \vdash A \text{ type} \quad \Gamma \vdash b : \mathbf{Bool}}{\Gamma \vdash \mathbf{If}(A[\mathbf{id.true}], A[\mathbf{id.false}], b) = A[\mathbf{id.b}] \text{ type}}$$

By the way, we can easily imagine that this structure tried to implement following isomorphism structure :

$$Ty(\Gamma.\mathbf{Bool}) \cong Ty(\Gamma) \times Ty(\Gamma)$$

The  $\beta$ -rule gives us forward homomorphism, and  $\eta$ -rule gives us backward homomorphism. Now, we'll enjoy 2 examples and realize that what can we do with this Large Elimination.

### Example 4.2.3.

One things that we can do is, generalize the eliminator for **Bool** type, which is ‘small’ **if**. Formally, we can ‘derive’ the following.

$$\frac{\Gamma \vdash a_t : A_t \quad \Gamma \vdash a_f : A_f \quad \Gamma \vdash b : \mathbf{Bool}}{\Gamma \vdash \mathbf{if}(a_t, a_f, b) : \mathbf{If}(A_t, A_f, b)}$$

*Proof.* First, by pre-assumptions,  $A_t, A_f$  are well-typed in  $\Gamma$ . It implies that

$$\Gamma.\mathbf{Bool} \vdash A_t[\mathbf{p}], A_f[\mathbf{p}] \text{ type}$$

And in this context, one is clear that

$$\Gamma.\mathbf{Bool} \vdash \mathbf{q} : \mathbf{Bool}$$

So, by introduction rule for **If**,

$$\Gamma.\mathbf{Bool} \vdash \underbrace{\mathbf{If}(A_t[\mathbf{p}], A_f[\mathbf{p}], \mathbf{q})}_B \text{ type}$$

Then, by the  $\beta$ -rule of **If**,

$$\Gamma \vdash B[\mathbf{id.true}] = A_t \quad \Gamma \vdash B[\mathbf{id.false}] = A_f$$

So we can write

$$\Gamma \vdash a_t : B[\mathbf{id.true}] \quad \Gamma \vdash a_f : B[\mathbf{id.false}]$$

Then by introduction rule for **if**,

$$\Gamma \vdash \mathbf{if}(a_t, a_f, b) : B[\mathbf{id.b}] = \mathbf{If}(A_t[\mathbf{p}], A_f[\mathbf{p}], \mathbf{q})[\mathbf{id.b}] = \mathbf{If}(A_t, A_f, b)$$

□

### Theorem 4.2.4.

Now we'll see one tricky theorem. The statement is really tricky, just **true**  $\neq$  **false** in our type system. One interesting thing is that, our textbook said that without Large Elimination or Universes, proving this is impossible.

*Proof.* Proving **true**  $\neq$  **false** is same with find following term disjoint.

$$\mathbf{1} \vdash \text{disjoint} : \Pi(\mathbf{Eq}(\mathbf{Bool}, \mathbf{true}, \mathbf{false}), \mathbf{Void})$$

And again, find such term is same with find following term.

$$\frac{1.\text{Eq}(\text{Bool}, \text{true}, \text{false}) \vdash v : \text{Void}}{\Gamma}$$

Imagine  $\Gamma.\text{Bool}$  now. In here,

$$\Gamma.\text{Bool} \vdash \text{Unit type} \quad \Gamma.\text{Bool} \vdash \text{Void type}$$

So by the introduction rule for **If**,

$$\frac{\Gamma.\text{Bool} \vdash \text{If}(\text{Unit}, \text{Void}, b) \text{ type}}{P}$$

However, the  $\beta$ -rule for **If** also implies that

$$\Gamma \vdash P[\text{id.true}] = \text{Unit}[\text{id.true}] = \text{Unit type}$$

and vice versa,

$$\Gamma \vdash P[\text{id.false}] = \text{Void type}$$

However,

$$\Gamma \vdash q : \text{Eq}(\text{Bool}, \text{true}, \text{false})$$

So extensional equality gives us

$$\text{true} = \text{false} \implies P[\text{id.false}] = \text{Void} = \text{Unit} = P[\text{id.true}]$$

So,

$$\Gamma \vdash tt : \text{Unit} = \text{Void}$$

This implies that there are term  $v : \text{Void}$  in  $\Gamma$ , means that disjoint =  $\lambda(tt)$ . So proof is done.  $\square$

However, there are some limitations when we imagine Large Elimination for **Nat** types. By mirroring, we reach following rule :

$$\frac{\Gamma \vdash n : \text{Nat} \quad \Gamma.\text{Nat} \vdash A_z \text{ type} \quad \Gamma.\text{Nat.type} \vdash A_s \text{ type}}{\Gamma \vdash \text{Rec}(A_z, A_s, n) \text{ type}}$$

Which is very non-trivial.

temp