Dependent Type Theory

About Extensional Type Systems

Hojune Lee

School of Computing, KAIST

Contents

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 it's 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) Γ, what terms of type A are well-formed ?
3. In this world (Context) Γ, what types are well-formed ?
4. Between 2 worlds (Context) Δ and Γ, is traffic way (substitution) γ 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. $\vdash \Gamma$ cx
2. $\Gamma \vdash a : A$
3. $\Gamma \vdash A$ type
4. $\Delta \vdash \gamma : \Gamma$
5. $\Gamma \vdash a = a' : A, \quad \Gamma \vdash A = A'$ 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.

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

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 γ is substitution from Δ to Γ. i.e.

$$\gamma : \Delta \to \Gamma$$

However, it's role is send types and terms of Γ into Δ. (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 Δ, Γ and mapping $\gamma : \Delta \to \Gamma$. Suppose that there is a function $g : \Gamma \to \mathbb{R}$, which is defined on set Γ. How can we 'use' this function in Δ set? One way is that,

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

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

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

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

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

Construction 1.3.2.

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

These rules give us that we can immigrate(pull-back) terms and types of Γ into Δ. 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 Γ 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 Γ 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.

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

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

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

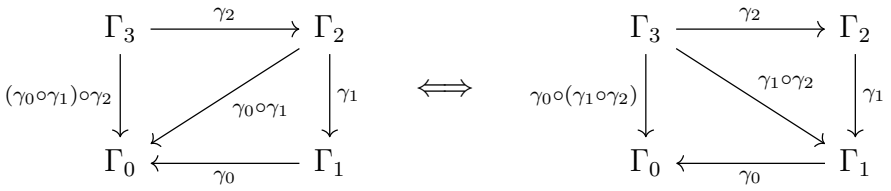
Then, above representation is very clear. It become same notation in our function calculus. Similarly,

Construction 1.3.5.

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

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

$$_{id}\circlearrowright \Delta \xrightarrow{\gamma} \Gamma \circlearrowright_{id}$$



Then we can imagine following equivalent rules clearly.

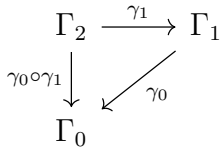
Construction 1.3.6.

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

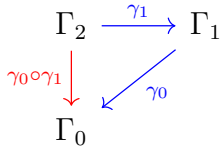
$$\frac{\Gamma_2 \vdash \gamma_1 : \Gamma_1 \quad \Gamma_1 \vdash \gamma_0 : \Gamma_0 \quad \Gamma_0 \vdash A \text{ type}}{\Gamma_2 \vdash A[\gamma_0 \circ \gamma_1] = A[\gamma_0][\gamma_1] \text{ type}}$$

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

These rules are very intuitable. When we draw diagram,



For example, look the last rule. a is term in the world Γ_0. We want to bring this term into Γ_2. Bring means that, we want to use 'function' a in Γ_2. i.e. such term in Γ_2 is working same roles with a in Γ_0. We've studied in box 1.3, there are 2 ways to bring a from Γ_0 into Γ_2.



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 Γ_2, and via blue way is $a[\gamma_0][\gamma_1]$ (Actually, $a[\gamma_0]$ is a in Γ_1 wolrd. So we take it again into Γ_2.) However, how can we use variable in well-formed context Γ without own names? What is the meaning of De Bruijn index? Let's see how can we distinguish variables in each 'world' Γ.

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 Γ wolrd. So the top variable's type in $\Gamma.A$ is $A[\mathbf{p}]$ which we take A from Γ into $\Gamma.A$. Similarly, we can access the any index of stack by following rule which is deribable from 1.3

Corolary 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 Γ into $\mathbf{1}$. The rules are simple.

Construction 1.3.9.

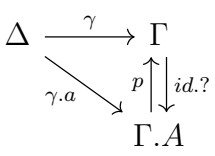
$$\frac{\vdash \Gamma \text{ cx}}{\Gamma \vdash ! : \mathbf{1}} \qquad \frac{\Gamma \vdash \delta : \mathbf{1}}{\Gamma \vdash \delta =! : \mathbf{1}}$$

What does it mean? It is 'way' from Γ to $\mathbf{1}$. The role of this way is transfer terms and types in $\mathbf{1}$ into Γ 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 Γ 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 Γ. 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 Γ has n variables $(x_1, \cdots, x_n)$. Here we denote names for each variables for representation. And then, in this context Γ, A is well-formed type. Then automatically $\Gamma.A$ is defined, where it has $n+1$ variables $(x_1, \cdots, x_n, x_{n+1})$.

The key understanding is that, when we bring terms and types in $\Gamma.A$ world, it can have $x_1, \cdots, x_{n+1}$ as symbol. When we meet x_{n+1}, then substitution it by a, and when we meet $x_1, \cdots, x_n$, then substitution it by following γ. Then since a is well-formed term in Δ and it's type is $A[\gamma]$, which do roles of A in Δ 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.

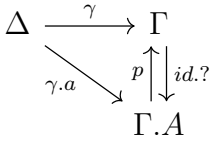
Construction 1.3.11.

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

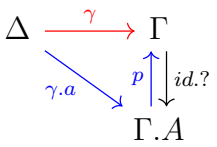
$$\frac{\Delta \vdash \gamma : \Gamma \quad \Gamma \vdash A \text{ type} \quad \Delta \vdash a : A[\gamma]}{\Delta \vdash \mathbf{q}[\gamma.a] = a : A[\gamma]}$$

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

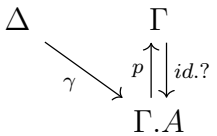
There rules are seemed to be very unclear. However, see following diagram. For first rule,



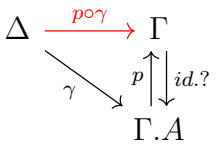
We can represent the first rule as following way.



So we can intuitively know that γ and $\mathbf{p} \circ (\gamma.a)$ represent same traffic way from Δ to Γ. 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 Δ and substitute by a, term of Δ. This rule explicitly claim this intuition. For third rule, let's draw slightly changed diagram.



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



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

Exercise 2.1

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

Proof. How can we define γ? The intuitive meaning of γ is that, way to bring types and terms of $\Gamma.A$ into Γ. Imagine that $\Gamma.A$ contains types of $x_1, \cdots, x_{n+1}$ and Γ contains types of $x_1, \cdots, x_n$. To bring types and terms in $\Gamma.A$ into Γ, only to do is when we meet x_{n+1}, substitute it with well-formed term in Γ 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 Γ. So, γ 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.,

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

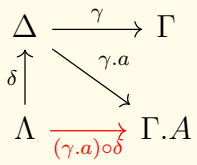
Here, the pre-suppostions of problem is that $\Delta \vdash \gamma : \Gamma$, $\Delta \vdash a : A[\gamma]$, $\Gamma \vdash A$ type and δ be any substitution $\delta : \Lambda \to \Delta$. In diagram,

$$\begin{array}{ccc} \Delta & \xrightarrow{\gamma} & \Gamma \\ {\scriptstyle\delta}\uparrow & \nearrow_{\gamma.a} & \\ \Lambda & & \Gamma.A \end{array}$$

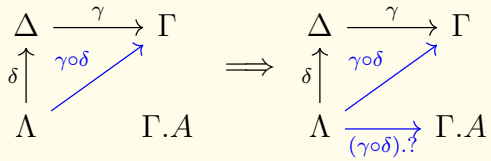
(Continue in Next Page . . .) □

Exercise 2.2

For intuition, we can draw following substitution.



And also,



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.

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

This implies that

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

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

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

We can apply substitution extend rule here.

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

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

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

□

Chapter 2

Internalizing Judgemental Structure : Π, Σ, Eq, 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 Γ and substitution γ 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 Σ. 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 imagin 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

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

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

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

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

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

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

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

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

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

We can pull-back A form Γ to Δ through γ. 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 Γ. 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

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

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 Σ 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 $\iota_{\Gamma,A,B}, \iota^{-1}_{\Gamma,A,B}$ 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 Σ.

Proof. TODO □

Now let's construct above isomorphisms explicitly.

Construction 2.2.2.

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

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

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

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

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

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

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

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

Construction 2.2.3.

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

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

I want to write some understanding for this construction. I think that most important one is that b is absolutely term of Γ. 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 Γ. Let's interpret above inference rule.

1. Under premises Γ, we have proof of A via a.
2. Under premises Γ with A, claim B.
3. Under premises Γ, 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 β-rule and η-rule. To show bijective, we just show that round-trip between terms and meta-pairs always preserve original element. Formally, we want to make

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

This is guaranteed explicitly via following rules.

Construction 2.2.4.

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

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

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

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.

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

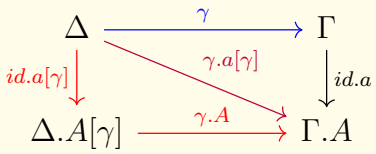
This is very intuitive. When we make $\mathbf{pair}(a,b)$ toy in Γ, we have 2 ways to take that toy into Δ via γ. First way is that take the toy directly ($\mathbf{pair}(a,b)[\gamma]$) and second way is that take two materials ($a[\gamma], b[\gamma]$) and make in Δ (then, $\mathbf{pair}(a[\gamma], b[\gamma])$). This rule say that both are same. This is intuitive meaning of naturality. However, this is just naturality of $\iota^{-1}_{\Gamma,A,B}$. 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 Δ also). This is from pattern-matching with Construction 2.2.3. See following diagram first.



Then see how can we take B from $\Gamma.A$ to Δ and b from Γ to Δ. B is pull-backed through red way and b is pull-backed through blue way. So $b[\gamma]$ is well-formed term in Δ. And similary, $B[\gamma.A][\mathbf{id}.a[\gamma]]$ is well-formed type in Δ. 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

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

□

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 Π. We'll define soon.
2. Dependent product's judgementrally-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

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

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

Construction 2.3.2.

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

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

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

And divide each isomorphism via

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

$$\iota^{-1}_{\Gamma, A\to B} : Tm(\Gamma.A, B) \xrightarrow{\cong} Tm(\Gamma, \Pi(A, B))$$

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

Construction 2.3.4.

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

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

Let's unpack $\iota^{-1}_{\Gamma,A\to B}$

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

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 Γ context. So we explain that we can't use $\mathbf{q}$ for express a in b. However, at above, b is term of $\Gamma.A$ world. So we can use $\mathbf{q}$ for represent b. This is one of major difference. Further, by this, we can agree our abstractical notion for matching of $\forall$ and Π. We can always take proof b of B under premises Γ appended with A by substitution $\mathbf{id}.a$ 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

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

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.

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

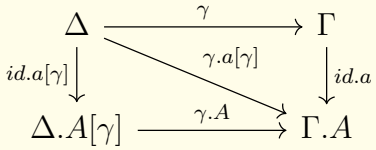
$$\frac{\Delta \vdash \gamma : \Gamma \quad \Gamma \vdash A \text{ type} \quad \Gamma.A \vdash b : B}{\Delta \vdash \lambda(b)[\gamma] = \lambda(b[\gamma.A]) : \Pi(A,B)[\gamma]}$$

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

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 drow diagram,



We can take a form Γ to Δ, type B from $\Gamma.A$ to $\Delta.A[\gamma]$, and take f from Γ to Δ. 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 Γ into Δ.

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

□

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 Γ.

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

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

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

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

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 Γ 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 $\iota^{-1}_{\Gamma, A \to B}$. It is similar to pull-back b which is term of type B in $\Gamma.A$ into Γ context by unknown extending $\mathbf{id}.?$. It is denoted by $\lambda(b)$. However, when we see $\iota_{\Gamma, A \to B}$, 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 Π-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 Γ to $\Gamma.A$, we always give $\Gamma \vdash a : A$, and $\mathbf{app}^*$ automatically has the intension that we'll use such ghost a. For convenience, our intension introducing here is that, externally mension such a we use. Then pull-back $\mathbf{app}^*(f[\mathbf{p}], \mathbf{q})$ into Γ by $\mathbf{id}.a$, 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, η-rule for define validity of $\iota^{-1}_{\Gamma,A,B}$) 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

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

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.

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

And divide each isomorphism via

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

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

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 Γ 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 $\iota_{\Gamma,A,a\equiv b}$:

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

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

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

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 cosntruct 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 decidablity 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** $\to$ .. We can always excute 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.

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

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

Chapter 3

Inductive Types
: Void, Bool, +, Nat

.