# Type Theory (WIP)

Phu Sakulwongtana

2024

Notes from *Introduction to Homotopy Type Theory* by Egbert Rijke.

## 1 Introduction

Dependent type theory is a formal system to organize all mathematical objects, structure, and knowledge. There are some important different between type theory, set theory and the interpretation of types as sets has significant limitation.

**Remark 1.** *(Differences Between Type and Set Theory) There are various differences that we want to discuss between 2 foundations:*

1. ***(Every element comes with its type)**, where $a : A$ is the judgement that $a$ an element of type $A$.*
2. ***(Type Theory is its own formal system)**, while set theory is axiomatized in the formal system of first order logic. Types and their element are constructed by following the formal system rules, and only from previous constructed type*
   (a) *Thus, expression $a : A$ is therefore not considered to be proposition (something which one can assert), but a judgement (an assessment that is part of the construction of the element $a : A$).*
3. ***(Stronger focus on equality)** of element governed by the identity type, while classical set theory, where equality is a decidable proposition of first order logic.*
   (a) *For example, type $x =_A y$ of identification of 2 element $x, y : A$ is itself a type.*
   (b) *We need to know (1) how to construct an element of the type and (2) how to show that 2 element of the types are equal, in order to understand a type fully.*

**Remark 2.** *(Building Dependent Type Theory) To build it, we start with giving structural rule, which express the general theory of type dependency (no ambient deductive system of first order logic in type theory).*

- ***(Deductive System + Operations):** It has its own deductive system, and the structural rules are the heart of this system. The operation that are governed by the structural rules are substitution and weaken operation*
- ***(Forming Types):** The most fundamental class of types are dependency function type or $\Pi$-types, together with type of natural number (type-theoretic version of induction principle). The induction principle can be used to: (1) construct the operations (2) prove properties about those operations.*
- ***(Identity Type):** The identity type $x =_A y$ is an example of a dependent type and is inductively generated by the reflexive element of $\mathrm{refl}_x : x =_A x$ (beware that it can have many different element).*
- ***(Universe):** Type families that are closed under the operations of type theory: $\Pi$-types, $\Sigma$-types, identity types and so on. They can be used to define type families over inductive types via the induction properties.*
  – *For example, defining ordering relation $\leq$ and $<$ on the natural number, or showing that $\mathrm{succ}_{\mathbb{N}}$ is injective, and $0_{\mathbb{Z}}$ isn't successor from Peano axiom.*

# 2 Dependent Type Theory

Usually the goal is to construct an element of certain type. For example, an element is a function if the type of the constructed element is a function type; proof of property if the type is a proposition; identification if the type is an identity type, etc.

*A type is just a collection of mathematical object and constructing element of a type is that every mathematics task or challenge*, and the system of inference rules (type theory) offer a principled way to do it.

## 2.1 Judgement and Context in Type Theory

**Definition 1.** ***(Inference Rules)*** *Construction consists of sequence of deductive steps, using finitely many premises (judgement), such steps can be represented by inference rule written as:*

$$\frac{\mathcal{H}_1 \qquad \mathcal{H}_2 \qquad \cdots \qquad \mathcal{H}_n}{\mathcal{C}}$$

*We ended up getting the single judgement from the rule.*

**Definition 2.** ***(Judgement)*** *There are 4 types of judgement in the type theory:*

- *$A$ is a (well-formed) type in context $\Gamma$, expressed as:* $\Gamma \vdash A\,\mathrm{Type}$
- *$A$ and $B$ are judgmentally equal types in context $\Gamma$, expressed as* $\Gamma \vdash A \equiv B\,\mathrm{Type}$
- *$a$ is an element of type $A$ in context $\Gamma$, expressed as* $\Gamma \vdash a : A$
- *$a$ and $b$ are judgmentally equal elements of type $A$ in context $\Gamma$, expressed as* $\Gamma \vdash a \equiv b : A$

The role of a context is to declare what hypothetical elements (commonly called variables) are assumed, along with their types.

**Definition 3.** ***(Context)*** *It is an expression of the form:*

$$x_1 : A_1 \qquad x_2 : A_2(x_1) \qquad \cdots \qquad x_n : A_n(x_1, \ldots, x_{n-1})$$

*satisfying the condition that for each $1 \leq k \leq n$, we can derive them using the inference rules of type theory i.e:*

$$x_1 : A_1, x_2 : A_2(x_1), \cdots, x_{k-1} : A_{k-1}(x_1, \ldots, x_{k-2}) \vdash A_k(x_1, \ldots, x_{k-1})\,\mathrm{Type}$$

*There is a context of length $0$, the empty context, which declare no variable, and satisfies this condition. A list of variable declaration $x_1 : A_1$ of length $1$ is context iff $A_1$ is a type in the empty context.*

In dependent type theory, all judgement are context dependent, and the types of the variable in a context may depend on any previous declared variables.

**Definition 4.** ***(Type Family)*** *Consider a type $A$ in context $\Gamma$, a family of types over $A$ in context $\Gamma$ is a type $B(x)$ in context $\Gamma, x : A$ where $\Gamma, x : A \vdash B(x)\,\mathrm{Type}$. Alternatively, we can say that $B(x)$ is a type indexed by $x : A$ in context $\Gamma$*

The basic example of a type family occurs when we introduce identity types, introduced as, which asserts that given an element $a : A$ in context $\Gamma$, we may form the type $a = x$ in context $\Gamma, x : A$:

$$\frac{\Gamma \vdash a : A}{\Gamma, x : A \vdash a = x\,\mathrm{Type}}$$

**Definition 5.** ***(Section)*** *Consider a type family $B$ over $A$ in context $\Gamma$. A section of the family $B$ over $A$ in context $\Gamma$ is an element of type $B(x)$ in context $\Gamma, x : A$. In a judgement: $\Gamma, x : A \vdash b(x) : B(x)$, we say that $b$ is a section of family $B$ over $A$. Alternatively, we say that $b(x)$ is an element of type $B(x)$ indexed by $x : A$ in context $\Gamma$.*

## 2.2 Inference Rule

We will present the system of inference rules, known as structural rules of type theory. There are 5 sets of inference rule: (1) Judgemental equality is an equivalent relation (2) Variable conversion rules (3) Substitution rules (4) Weakening rules (5) Generic element.

**Definition 6.** ***(Judgemental Equality as Equivalent Relation)*** *These rules assert that the relation are reflexive, symmetric and transitive:*

$$\frac{\Gamma \vdash A\,\mathrm{Type}}{\Gamma \vdash A \equiv A\,\mathrm{Type}} \qquad \frac{\Gamma \vdash A' \equiv A\,\mathrm{Type}}{\Gamma \vdash A \equiv A'\,\mathrm{Type}} \qquad \frac{\Gamma \vdash A \equiv A'\,\mathrm{Type} \qquad \Gamma \vdash A' \equiv A''\,\mathrm{Type}}{\Gamma \vdash A \equiv A''\,\mathrm{Type}}$$

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash a \equiv a : A} \qquad \frac{\Gamma \vdash a \equiv a' : A}{\Gamma \vdash a' \equiv a : A} \qquad \frac{\Gamma \vdash a \equiv a' : A \qquad \Gamma \vdash a' \equiv a'' : A}{\Gamma \vdash a \equiv a'' : A}$$

**Definition 7.** ***(Variable Conversion Rule)*** *The rules assert that we can convert the type of a variable to judgmentally equal types:*

$$\frac{\Gamma \vdash A \equiv A'\,\mathrm{Type} \qquad \Gamma, x : A, \Delta \vdash \mathcal{J}}{\Gamma, x : A', \Delta \vdash \mathcal{J}}$$

*where $\mathcal{J}$ is generic judgment thesis as representing all kinds of judgement. For example, it can be $B(x)\,\mathrm{Type}$. We do this so that we don't have to write the rule 4 times.*

**Remark 3.** *(Type Theoretic Substitution) Suppose we have $\Gamma, x : A, y_1 : B_1, \ldots, y_n : B_n \vdash C\,\mathrm{Type}$ and an element $a : A$ in context $\Gamma$, we can substitute $a$ for all occurrences of $x$ in types $B_1, \ldots, B_n$ and $C$, to get:*

$$\Gamma, y_1 : B_1[a/x], \ldots, y_n : B_n[a/x] \vdash C[a/x]\,\mathrm{Type}$$

*The variable $y_1, \ldots, y_n$ are assigned new types. Also, we can substitute $a$ for $x$ in a element $c : C$ to obtain $c[a/x] : C[a/x]$. As this can be done on both sides together, even in the case with judgmental equality, we have the substitution rules.*

**Definition 8.** ***(Substitution Rules)*** *These rules assert that we can do the substitution on all kinds of judgement and on the whole context:*

$$\frac{\Gamma \vdash a : A \qquad \Gamma, x : A, \Delta \vdash \mathcal{J}}{\Gamma, \Delta[a/x] \vdash \mathcal{J}[a/x]} S$$

*Furthermore, we have to make sure that substitution of equal terms lead to equal substituted terms i.e congruent under both the judgemental equal kinds:*

$$\frac{\Gamma \vdash a \equiv a' : A \qquad \Gamma, x : A, \Delta \vdash B\,\mathrm{Type}}{\Gamma, \Delta[a/x] \vdash B[a/x] \equiv B[a'/x]\,\mathrm{Type}} \qquad \frac{\Gamma \vdash a \equiv a' : A \qquad \Gamma, x : A, \Delta \vdash b : B}{\Gamma, \Delta[a/x] \vdash b[a/x] \equiv b[a'/x] : B[a'/x]}$$

*Observe that both $B[a/x]$ and $B[a'/x]$ are types in context $\Delta[a/x]$ provided that $a \equiv a'$.*

**Definition 9.** ***(Fiber)*** *There are 2 cases, when consider the substitution on type family and its element. When $B$ is a family of types over $A$ in context $\Gamma$ and if $a : A$, then we say that $B[a/x]$ is a fiber of $B$ at $a$, denoted as $B(a)$. On the other hand, When $b$ is a section of the family $B$ over $A$ in context $\Gamma$, we call the element $b[a/x]$ the value of $b$ at $a$, denoted as $b(a)$*

**Remark 4.** *(Relationship between Section and Fiber)* ***(NOTE: Not sure)*** *Fiber is more of renaming, while section is more of like evaluation.*

*For example (not accurate but illustrate the points), the type family is $f(x, y) = x^2 + y^2$. Then the fiber of $f$ at $a$ is $f(x, y)[a/y] = f(x, a) = x^2 + a^2$. Its section (an element of type families) is $f(10, y) = 10^2 + y^2$, while fiber of section is $f[a/y](10, y)$. The indexed section can be $f(10, 3) = 10^2 + 3^2$.*

**Definition 10.** ***(Weaken Rule)*** *Weakening by a type $A$ in context preserves well-formedness and judgmental equality, where we have:*

$$\frac{\Gamma \vdash A\,\mathrm{Type} \qquad \Gamma, \Delta \vdash \mathcal{J}}{\Gamma, x : A, \Delta \vdash \mathcal{J}} W$$

*The process is expanding the context by a variable of type $A$ is called weakening (by $A$). In the simplest situation when weakening applies, we have 2 types of $A$ and $B$ in context $\Gamma$.*

**Definition 11.** ***(Constant/Trivial Family)*** *The simplest example of weakening is to weaken $B$ by $A$ as:*

$$\frac{\Gamma \vdash A\,\mathrm{Type} \qquad \Gamma \vdash B\,\mathrm{Type}}{\Gamma, x : A \vdash B\,\mathrm{Type}}$$

*The type $B$ in context $\Gamma, x : A$ is called constant family $B$ or the trivial family $B$.*

Given a type $A$ in context $\Gamma$, then we can weaken $A$ itself to obtain $A$ is a type in context $\Gamma, x : A$.

**Definition 12.** ***(Variable Rule/Generic Element)*** *Asserts that any element $x : A$ in the context $\Gamma, x : A$ is also an element of type $A$ in context $\Gamma, x : A$:*

$$\frac{\Gamma \vdash A\,\mathrm{Type}}{\Gamma, x : A \vdash x : A}\,\delta$$

*It is to make sure that the variable declared in a context are indeed elements and provides identifying function on the type $A$ in context $\Gamma$.*

## 2.3 Derivation

**Definition 13.** ***(Derivation)*** *Aa finite tree in which each node is a valid rule of inference. The root is the conclusion and the leaves of the tree we find the hypothesis. Given a derivation with hypothesis $\mathcal{H}_1, \ldots, \mathcal{H}_n$ and conclusion $\mathcal{C}$, we have a new inference rule as:*

$$\frac{\mathcal{H}_1, \ldots, \mathcal{H}_n}{\mathcal{C}}$$

*such a rule is called derivative. To make things short, we can use any derived rules in any future derivation.*

**Lemma 1.** ***(Changing Variable)*** *Variable can always be change to fresh variable, given as:*

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

*where $x'$ doesn't occurs in context $\Gamma, x : A, \Delta$*

*Proof.* Note that we have to do weakening rule is because we have to make sure that the context matches before performing the substitution

$$\frac{\dfrac{\Gamma \vdash A\,\mathrm{Type}}{\Gamma, x' : A \vdash x' : A}\,\delta \qquad \dfrac{\Gamma \vdash A\,\mathrm{Type} \qquad \Gamma, x : A, \Delta \vdash \mathcal{J}}{\Gamma, x' : A, x : A, \Delta \vdash \mathcal{J}}\,W}{\Gamma, x' : A, \Delta[x'/x] \vdash \mathcal{J}[x'/x]}\,S$$

□

**Lemma 2.** ***(Interchanging Variables)*** *If we have 2 types $A$ and $B$ in context $\Gamma$ and make a judgement in context $\Gamma, x : A, y : B, \Delta$ then we make that same judgement in context $\Gamma, y : B, x : A, \Delta$*

$$\frac{\Gamma \vdash B\,\mathrm{Type} \qquad \Gamma, x : A, y : B, \Delta \vdash \mathcal{J}}{\Gamma, y : B, x : A, \Delta \vdash \mathcal{J}}$$

*Proof.* The trick is that the substitution two times can leads to the "deletion" of the elements (this is how we delete $y' : B$ at the end, by replacing it again with $y$):

$$\frac{\dfrac{\dfrac{\Gamma \vdash B\,\mathrm{Type}}{\Gamma, y : B \vdash y : B}\,\delta}{\Gamma, y : B, x : A \vdash y : B}\,W \qquad \dfrac{\dfrac{\Gamma, x : A, y : B, \Delta \vdash \mathcal{J}}{\Gamma \vdash B\,\mathrm{Type} \qquad \Gamma, x : A, y' : B, \Delta[y'/y] \vdash \mathcal{J}[y'/y]}\,y'/y}{\Gamma, y : B, x : A, y' : B, \Delta[y'/y] \vdash \mathcal{J}[y'/y]}\,W}{\Gamma, y : B, x : A, \Delta \vdash \mathcal{J}}\,S$$

□

**Lemma 3.** ***(Element Conversion Rule)*** *If the types are judgmentally equals to each other then we can interchange the type of its element:*

$$\frac{\Gamma \vdash A \equiv A'\,\mathrm{Type} \qquad \Gamma \vdash a : A}{\Gamma \vdash a : A'} \qquad \frac{\Gamma \vdash A \equiv A'\,\mathrm{Type} \qquad \Gamma \vdash a \equiv b : A}{\Gamma \vdash a \equiv b : A'}$$

*On the RHS, we also have the congruence rule for element conversion.*

*Proof.* For the first part:

$$\dfrac{\dfrac{}{\Gamma \vdash a : A} \qquad \dfrac{\dfrac{\Gamma \vdash A \equiv A' \,\mathrm{Type}}{\Gamma \vdash A' \equiv A \,\mathrm{Type}} \qquad \dfrac{\Gamma \vdash A' \,\mathrm{Type}}{\Gamma, x : A' \vdash x : A'}\,\delta}{\Gamma, x : A \vdash x : A'}}{\Gamma \vdash a : A'}\, S$$

For the second part, we have used the congruence under substitution, at the last step.

$$\dfrac{\dfrac{}{\Gamma \vdash a \equiv b : A} \qquad \dfrac{\dfrac{\Gamma \vdash A \equiv A' \,\mathrm{Type}}{\Gamma \vdash A' \equiv A \,\mathrm{Type}} \qquad \dfrac{\Gamma \vdash A' \,\mathrm{Type}}{\Gamma, x : A' \vdash x : A'}\,\delta}{\Gamma, x : A \vdash x : A'}}{\Gamma \vdash a \equiv b : A'}$$

□

## 2.4 Dependent Function Types

**Definition 14.** ***(Dependent Function Types)*** *Consider section $b$ of family $B$ over $A$ in context $\Gamma$ i.e $\Gamma, x : A \vdash b(x) : B(x)$ such section $b$ is an operator that takes as input $x : A$ and produces a term $b(x) : B(x)$, which may depend on $x : A$. We will denote type of dependent functions as $\Pi_{(x:A)}B(x)$.*

**Remark 5.** ***(Principle Rules for $\Pi$-Types)*** *There are 4 principal rules for this: (1) The formulation rule (2) The introduction rule (3) The elimination rule (4) The computation rule.*

*Apart from that we also need rules, as part of the specification, that assert that all the constructors respect judgemental equality, called congruence rules*

**Definition 15.** ***($\Pi$-Formation Rule)*** *Tells us how we may form dependent function type. For any type family $B$ of types over $A$, $\Pi$-formation says (LHS), and it should respect judgmental equality (RHS):*

$$\dfrac{\Gamma, x : A \vdash B(x) \,\mathrm{Type}}{\Gamma \vdash \Pi_{(x:A)}B(x) \,\mathrm{Type}}\,\Pi \qquad \dfrac{\Gamma \vdash A \equiv A' \,\mathrm{Type} \qquad \Gamma, x : A \vdash B(x) \equiv B'(x) \,\mathrm{Type}}{\Gamma \vdash \Pi_{(x:A)}B(x) \equiv \Pi_{(x:A')}B'(x) \,\mathrm{Type}}\,\Pi\text{-}eq$$

*In order to form the type $\Pi_{(x:A)}B(x)$ in context $\Gamma$, we must have a type family $B$ over $A$ in context $\Gamma$.*

**Definition 16.** ***($\Pi$-Introduction Rule)*** *Tells us how to introduce new term of dependent function types. A dependent function $f : \Pi_{(x:A)}B(x)$ is an operation that takes an $x : A$ to $f(x) : B(x)$. Or, in order to construct a dependent function, one has to construct a term $b(x) : B(x)$ indexed by $x : A$ in context $\Gamma$ (LHS), and it should respect judgmental equality (RHS):*

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

*This is also called $\lambda$-abstraction rule, and say that the $\lambda x.b(x)$ binds the variable $x$ in $b$.*

**Definition 17.** ***($\Pi$-Elimination Rule)*** *Tells us how to use arbitrary terms of dependent function types. In other words, it is an evaluate rule (LHS) and it should respect judgmental equality (RHS):*

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

**Definition 18.** ***($\Pi$-Computation Rule)*** *Tells us how the introduction and elimination rule interact i.e guarantee that every term of a dependent function type is indeed a dependent function taking the values by which it is defined. On LHS, When we evaluate it at $x : A$, we obtain the value $b(x) : B(x)$ (called $\beta$-rule):*

$$\dfrac{\Gamma, x : A \vdash b(x) : B(x)}{\Gamma, x : A \vdash \big(\lambda y.b(y)\big)(x) \equiv b(x) : B(x)}\,\beta \qquad \dfrac{\Gamma \vdash f : \Pi_{(x:A)}B(x)}{\Gamma \vdash \lambda x.f(x) \equiv f : \Pi_{(x:A)}B(x)}\,\eta$$

*On RHS, we asserts that all elements of a $\Pi$-Type are dependent function (called $\eta$-rule). In other words, $\lambda$-abstraction rules and evaluate rules are mutual inverse*

**Definition 19.** ***(Ordinary Function Type)*** *Consider the case where both $A$ and $B$ are types in context $\Gamma$, and construct the ordinary function from $A$ to $B$:*

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

**Remark 6.** *We can make definition at the end of a derivation if the conclusion is a certain type/term in context. Suppose we have derivation of (LHS)*

$$\frac{\mathcal{D}}{\Gamma \vdash a : A} \qquad \frac{\dfrac{\mathcal{D}}{\Gamma \vdash a : A}}{\Gamma \vdash c := a : A}$$

*in which the derivation make use of the premises $\mathcal{H}_1, \ldots, \mathcal{H}_n$. If we wish to make a definition $c := a$, then we can extend the derivation tree (RHS). The effect is that we have extend our type theory with a new constant $c$ for which the inference rules are valid:*

$$\frac{\mathcal{H}_1 \quad \mathcal{H}_2 \quad \cdots \quad \mathcal{H}_n}{\Gamma \vdash c : A} \qquad \frac{\mathcal{H}_1 \quad \mathcal{H}_2 \quad \cdots \quad \mathcal{H}_n}{\Gamma \vdash c \equiv a : A}$$

*This would works with the definition of the ordinary function type that is we have $\Gamma \vdash A \to B \equiv \Pi_{(x:A)} B \,\mathrm{Type}$, at the end of the derivation.*

We can now use the term conversion rules, we have the corresponding formation, introduction, elimination and computation rules for the ordinary function type.

$$\frac{\Gamma \vdash A \,\mathrm{Type} \qquad \Gamma \vdash B \,\mathrm{Type}}{\Gamma \vdash A \to B \,\mathrm{Type}} \to \qquad \frac{\Gamma \vdash B \,\mathrm{Type} \qquad \Gamma, x : A \vdash b(x) : B \,\mathrm{Type}}{\Gamma \vdash \lambda x.b(x) : A \to B} \lambda$$

and so on. Now, we will consider identity function and the composition between functions.

**Definition 20.** ***(Identity Function)*** *For any type $A$ in context $\Gamma$, we define the identity function $\mathrm{id}_A : A \to A$ using the generic terms as:*

$$\frac{\dfrac{\dfrac{\Gamma \vdash A \,\mathrm{Type}}{\Gamma, x : A \vdash x : A}}{\Gamma \vdash \lambda x.x : A \to A} \lambda}{\Gamma \vdash \mathrm{id}_A := \lambda x.x : A \to A}$$

*And, satisfies the following inference rules as:*

$$\frac{\Gamma \vdash A \,\mathrm{Type}}{\Gamma \vdash \mathrm{id}_A : A \to A} \qquad \frac{\Gamma \vdash A \,\mathrm{Type}}{\Gamma \vdash \mathrm{id}_A \equiv \lambda x.x : A \to A}$$

For composition of function, we will introduce composition itself as a function comp that takes 2 arguments: $g : B \to C$ and $f : A \to B$. The output is $\mathrm{comp}(g, f) : A \to C$ for which we often write $g \circ f$.

**Remark 7.** *(Observation on Multiple-Valued Function) For multiple argument function, we can form it by iterating the $\Pi$-formation rule or the $\to$-formation rule. For example, with function $f : A \to (B \to C)$.*

*On the other hand, When $C(x, y)$ being a family of types indexed by $x : A$ and $y : B(x)$, then we can form the dependent type as: $\Pi_{(x:A)}\Pi_{(y:B(x))}C(x, y)$. In the special case where $C(x, y)$ is a family of types indexed by 2 elements $x, y : A$ of the same type, then we often write: $\Pi_{(x,y):A}C(x, y)$ for the type $\Pi_{(x:A)}\Pi_{(y:A)}C(x, y)$.*

**Definition 21.** *Given types $A, B$ and $C$ in context $\Gamma$, there is a composition operation:*

$$\mathrm{comp} : (B \to C) \to \big((A \to B) \to (A \to C)\big)$$

*It can also be defined as: $\mathrm{comp} := \lambda g.\lambda f.\lambda x.g(f(x))$, in which the construction is given by:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash A \,\mathrm{Type} \qquad \Gamma \vdash B \,\mathrm{Type}}{\Gamma, f : B^A, x : A \vdash f(x) : B}\,(1)}{\Gamma, g : C^B, f : B^A, x : A \vdash f(x) : B}\,(W) \qquad \dfrac{\dfrac{\Gamma \vdash B \,\mathrm{Type} \qquad \Gamma \vdash C \,\mathrm{Type}}{\Gamma, g : C^B, y : B \vdash g(y) : C}\,(2)}{\Gamma, g : C^B, x : A, f : B^A, y : B \vdash g(y) : C}\,(W)}{\Gamma, g : C^B, f : B^A, x : A, y : B \vdash g(y) : C}\,(W)}{\Gamma, g : C^B, f : B^A, x : A \vdash g(f(x)) : C}\,(E)}{\Gamma, g : C^B, f : B^A \vdash \lambda x.g\big(f(x)\big) : C^A}}{\Gamma, g : B \to C \vdash \lambda f.\lambda x.g\big(f(x)\big) : B^A \to C^A}}{\Gamma \vdash \mathrm{comp} := \lambda g.\lambda f.\lambda x.g\big(f(x)\big) : C^B \to \big(B^A \to C^A\big)}$$

*where the last 3 step we just used the lambda rule $(\lambda)$. For the derivation of $(1)$ and $(2)$, we have:*

$$\frac{\dfrac{\dfrac{\Gamma \vdash A \,\mathrm{Type} \qquad \Gamma \vdash B \,\mathrm{Type}}{\Gamma \vdash A \to B \,\mathrm{Type}}\,\delta}{\Gamma, f : B^A \vdash f : B^A}}{\Gamma, f : B^A, x : A \vdash f(x) : B}\,ev$$

And so we have that function composition is associative and that the identity function satisfies the unit laws.

**Lemma 4.** *The composition of function is associative:*

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

*Proof.* The idea: both $\big((h \circ g) \circ f\big)(x)$ and $\big(h \circ (g \circ f)\big)(x)$ evaluate to be $h(g(f(x)))$ and therefore $(h \circ g) \circ f$ and $h \circ (g \circ f)$ must be judgmentally equal.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash f : A \to B}{\Gamma, x : A \vdash f(x) : B}\,\text{ev} \qquad \dfrac{\dfrac{\Gamma \vdash g : B \to C}{\Gamma, y : B \vdash g(y) : C}\,\text{ev}}{\Gamma, x : A, y : B \vdash g(y) : C}\,W}{\Gamma, x : A \vdash g(f(x)) : C}\,E \qquad \dfrac{\dfrac{\Gamma \vdash h : C \to D}{\Gamma, z : C \vdash h(z) : D}\,\text{ev}}{\Gamma, x : A, z : C \vdash b(z) : D}\,W}{\Gamma, x : A \vdash h(g(f(x))) : D}\,E}{\Gamma, x : A \vdash h(g(f(x))) \equiv h(g(f(x))) : D}}{\Gamma, x : A \vdash (h \circ g)(f(x)) \equiv h((g \circ f)(x)) : D}}{\Gamma, x : A \vdash ((h \circ g) \circ f)(x) \equiv (h \circ (g \circ f))(x) : D}\,\Pi}{\Gamma \vdash ((h \circ g) \circ f)(x) \equiv (h \circ (g \circ f))(x) : A \to D}$$

Note that the last two steps were just the untangling of the definition of composition between functions. □

**Lemma 5.** *Composition of functions satisfy the left and right unit law i.e we can derive as:*

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

*Proof.* It suffices to derive that $\mathrm{id}_B(f(x)) \equiv f(x)$ in context $\Gamma, x : A$

$$\dfrac{\dfrac{\Gamma \vdash f : A \to B}{\Gamma, x : A \vdash f(x) : B}\,\text{ev} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash B\,\text{Type}}{\Gamma, y : B \vdash y : B}\,\delta}{\Gamma, y : B \vdash (\lambda x.x)(y) \equiv y}\,\beta \qquad \dfrac{\Gamma \vdash B\,\text{Type}}{\Gamma \vdash \mathrm{id}_B \equiv \lambda x.x : B \to B}}{\Gamma, y : B \vdash \mathrm{id}_B(y) \equiv \lambda x.x(y) : B}\,\text{ev-eq}}{\Gamma, y : B \vdash \mathrm{id}_B(y) \equiv y : B} \qquad \dfrac{\Gamma \vdash A\,\text{Type}}{}}{\Gamma, x : A, y : B \vdash \mathrm{id}_B(y) \equiv y : B}\,W}{\Gamma, x : A \vdash \mathrm{id}_B(f(x)) \equiv f(x) : B}\,E$$

Then we have that:

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma, x : A \vdash \mathrm{id}_B(f(x)) \equiv f(x) : B}}{\Gamma \vdash \lambda x.\,\mathrm{id}_B(f(x)) \equiv \lambda x.f(x) : A \to B}\,\lambda\text{-eq} \qquad \dfrac{\Gamma \vdash f : A \to B}{\Gamma \vdash \lambda x.f(x) \equiv f : A \to B}\,\eta}{\Gamma : \mathrm{id}_B \circ f \equiv f : A \to B}$$

□

**Lemma 6.** *If $f$ and $g$ are equal values, then they must be equal that is:*

$$\frac{\Gamma \vdash f : \Pi_{(x:A)}B(x) \qquad \Gamma \vdash g : \Pi_{(x:A)}B(x) \qquad \Gamma, x : A \vdash f(x) \equiv g(x) : B(x)}{\Gamma \vdash f \equiv g : \Pi_{(x:A)}B(x)}$$

*Proof.* We have that:

$$\dfrac{\dfrac{\Gamma \vdash f : \Pi_{(x:A)}B(x)}{\Gamma \vdash \lambda x.f(x) \equiv f : \Pi_{(x:A)}B(x)}\,\eta \qquad \dfrac{\Gamma, x : A \vdash f(x) \equiv g(x) : B(x)}{\Gamma \vdash \lambda x.f(x) \equiv \lambda x.g(x) : \Pi_{(x:A)}B(x)}\,\lambda\text{-eq}}{\Gamma \vdash f \equiv \lambda x.g(x) : \Pi_{(x:A)}B(x)} \qquad \dfrac{\Gamma \vdash g : \Pi_{(x:A)}B(x)}{\Gamma \vdash \lambda x.g(x) \equiv g : \Pi_{(x:A)}B(x)}\,\eta}{\Gamma \vdash f \equiv g : \Pi_{(x:A)}B(x)}$$

□

**Definition 22.** ***(Constant Function)*** *We can derive the constant function as follows:*

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash A\,\text{Type} \qquad \dfrac{\Gamma \vdash B\,\text{Type}}{\Gamma, y : B \vdash y : B}\,\delta}{\Gamma, y : B, x : A \vdash y : B}\,W}{\Gamma, y : B \vdash \lambda x.y : A \to B}\,\lambda}{\Gamma, y : B \vdash \mathrm{const}_y := \lambda x.y : A \to B} \quad \rightsquigarrow \quad \frac{\Gamma \vdash A\,\text{Type}}{\Gamma, y : B \vdash \mathrm{const}_y : A \to B}$$

**Lemma 7.** *The constant function behaves as expected that is:*

$$\frac{\Gamma \vdash f : A \to B}{\Gamma, z : C \vdash \mathrm{const}_z \circ f \equiv \mathrm{const}_z : A \to C} \qquad \frac{\Gamma \vdash A\,\mathrm{Type} \qquad \Gamma \vdash g : B \to C}{\Gamma, y : B \vdash g \circ \mathrm{const}_y \equiv \mathrm{const}_{g(y)} : A \to C}$$

*Proof.* **(First Part):** First, we are trying to perform the evaluation of the lambda function:

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash B\,\mathrm{Type} \qquad \Gamma \vdash C\,\mathrm{Type}}{\Gamma, z : C \vdash \mathrm{const}_z \equiv \lambda y.z : B \to C}}{\Gamma, y : B, z : C \vdash \mathrm{const}_z(y) \equiv \lambda y.z(y) : C}\,\text{ev-eq} \qquad \dfrac{\dfrac{\overline{\Gamma \vdash B\,\mathrm{Type}} \qquad \dfrac{\Gamma \vdash C\,\mathrm{Type}}{\Gamma, z : C \vdash z : C}\,\delta}{\Gamma, z : C, y : B \vdash z : C}\,W}{\Gamma, z : C, y : B \vdash (\lambda y.z)(y) \equiv z : C}\,\beta}{\Gamma, y : B, z : C \vdash \mathrm{const}_z(y) \equiv z : C}\,(\mathrm{swap} + E)$$

Then from this, we have that:

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash f : A \to B}{\Gamma, x : A \vdash f(x) : B}\,\mathrm{ev} \qquad \dfrac{\overline{\Gamma \vdash A\,\mathrm{Type}} \qquad \vdots \quad \overline{\Gamma, y : B, z : C \vdash \mathrm{const}_z(y) \equiv z : C}}{\Gamma, x : A, y : B, z : C \vdash \mathrm{const}_z(y) \equiv z : C}\,W}{\Gamma, x : A, z : C \vdash \mathrm{const}_z(f(x)) \equiv z : C}\,E}{\Gamma, z : C \vdash \lambda x.\,\mathrm{const}_z(f(x)) \equiv \lambda x.z : A \to C}\,(\mathrm{swap} + \lambda\text{-eq})$$

Finally, we have that:

$$\frac{\dfrac{\vdots}{\Gamma, z : C \vdash \lambda x.\,\mathrm{const}_z(f(x)) \equiv \lambda x.z : A \to C} \qquad \dfrac{\Gamma \vdash A\,\mathrm{Type}}{\Gamma, z : C \vdash \mathrm{const}_z \equiv \lambda x.z : A \to C}}{\Gamma, z : C \vdash \mathrm{const}_z \circ f \equiv \mathrm{const}_z : A \to C}$$

□

*Proof.* **(Second Part):** We have that:

$$\frac{\dfrac{\Gamma \vdash g : B \to C}{\Gamma, y : B \vdash g(y) : C}\,\mathrm{ev} \qquad \dfrac{\overline{\Gamma \vdash B\,\mathrm{Type}} \qquad \dfrac{\Gamma \vdash A\,\mathrm{Type}}{\Gamma, z : C \vdash \mathrm{const}_z \equiv \lambda x.z : A \to C}}{\Gamma, y : B, z : C \vdash \mathrm{const}_z \equiv \lambda x.z : A \to C}\,E}{\Gamma, y : B \vdash \mathrm{const}_{g(y)} \equiv \lambda x.g(y) : A \to C}$$

We aim to show that $\Gamma, y : B \vdash g \circ \mathrm{const}_y \equiv \lambda x.g(y)$. Thus, we have that (from the first part, together with the congruence of substitution rule, for the last step, assuming we can duplicate the context $\ldots, y : B, y : B \vdash \ldots$):

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma, x : A, y : B \vdash \mathrm{const}_y(x) \equiv y : B} \qquad \dfrac{\overline{\Gamma \vdash A\,\mathrm{Type}} \qquad \dfrac{\Gamma \vdash g : B \to C}{\Gamma, y : B \vdash g(y) : C}\,\mathrm{ev}}{\Gamma, x : A, y : B \vdash g(y) : C}\,W}{\Gamma, x : A, y : B \vdash g(\mathrm{const}_y(x)) \equiv g(y) : C}}{\Gamma, y : B \vdash \lambda x.g(\mathrm{const}_y(x)) \equiv \lambda x.g(y) : A \to C}\,\lambda\text{-eq}$$

So we have that:

$$\frac{\dfrac{\vdots}{\Gamma, y : B \vdash \lambda x.g(\mathrm{const}_y(x)) \equiv \lambda x.g(y) : A \to C} \qquad \dfrac{\vdots}{\Gamma, y : B \vdash \mathrm{const}_{g(y)} \equiv \lambda x.g(y) : A \to C}}{\Gamma, y : B \vdash (g \circ \mathrm{const}_y) \equiv \mathrm{const}_{g(y)} : C}$$

□

**Definition 23.** ***(Swap Function)*** *We can derive the swap function as follows:*

$$\frac{\Gamma \vdash A\,\mathrm{Type} \qquad \Gamma \vdash B\,\mathrm{Type} \qquad \Gamma, x : A, y : B \vdash C(x, y)\,\mathrm{Type}}{\Gamma \vdash \sigma : \left(\Pi_{(x:A)}\Pi_{(y:B)}C(x, y)\right) \to \left(\Pi_{(y:B)}\Pi_{(x:A)}C(x, y)\right)}$$

*We start by a creation of the evaluated elements (in order for us to prepare for the creation of lambda element):*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, x : A, y : B \vdash C(x,y)\,\mathrm{Type}}{\Gamma, x : A \vdash \Pi_{(y:B)} C(x,y)\,\mathrm{Type}}\,\Pi}{\Gamma \vdash \Pi_{(x:A)}\Pi_{(y:B)} C(x,y)\,\mathrm{Type}}\,\Pi}{\Gamma, f : \Pi_{(x:A)}\Pi_{(y:B)} C(x,y) \vdash f : \Pi_{(x:A)}\Pi_{(y:B)} C(x,y)}\,\delta}{\Gamma, f : \Pi_{(x:A)}\Pi_{(y:B)} C(x,y), x : A \vdash f(x) : \Pi_{(y:B)} C(x,y)}\,ev}{\Gamma, f : \Pi_{(x:A)}\Pi_{(y:B)} C(x,y), x : A, y : B \vdash f(x,y) : C(x,y)}\,ev}{\Gamma, f : \Pi_{(x:A)}\Pi_{(y:B)} C(x,y), y : B, x : A \vdash f(x,y) : C(x,y)}\,swap$$

*Then we have to repeatedly used the lambda rule:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, f : \Pi_{(x:A)}\Pi_{(y:B)} C(x,y), y : B, x : A \vdash f(x,y) : C(x,y)}{\Gamma, f : \Pi_{(x:A)}\Pi_{(y:B)} C(x,y), y : B \vdash \lambda x.f(x,y) : \Pi_{(x:A)} C(x,y)}}{\Gamma, f : \Pi_{(x:A)}\Pi_{(y:B)} C(x,y) \vdash \lambda y.\lambda x.f(x,y) : \Pi_{(y:A)}\Pi_{(x:A)} C(x,y)}}{\Gamma \vdash \lambda f.\lambda y.\lambda x.f(x,y) : \big(\Pi_{(x:A)}\Pi_{(y:B)} C(x,y)\big) \to \big(\Pi_{(y:A)}\Pi_{(x:A)} C(x,y)\big)}}{\Gamma \vdash \sigma := \lambda f.\lambda y.\lambda x.f(x,y) : \big(\Pi_{(x:A)}\Pi_{(y:B)} C(x,y)\big) \to \big(\Pi_{(y:A)}\Pi_{(x:A)} C(x,y)\big)}$$

**Lemma 8.** *When we use the swap function 2 times, we ended up getting the same function:*

$$\dfrac{\Gamma \vdash A\,\mathrm{Type} \qquad \Gamma \vdash B\,\mathrm{Type} \qquad \Gamma, x : A, y : B \vdash C(x,y)\,\mathrm{Type}}{\Gamma \vdash \sigma \circ \sigma \equiv \mathrm{id} : \big(\Pi_{(x:A)}\Pi_{(y:B)} C(x,y)\big) \to \big(\Pi_{(x:A)}\Pi_{(y:B)} C(x,y)\big)}$$

*Proof.* Let's start with getting the lambda formulation of id as we repeated apply the $\eta$ rule (we will also stop using the type annotation because it is too long):

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash \Pi_{(x:A)}\Pi_{(y:B)} C(x,y)\,\mathrm{Type}}}{\Gamma \vdash \mathrm{id} \equiv \lambda g.g : \big(\Pi_{(x:A)}\Pi_{(y:B)} C(x,y)\big) \to \big(\Pi_{(x:A)}\Pi_{(y:B)} C(x,y)\big)}\,\text{ev-eq}}{\Gamma, f : \Pi_{(x:A)}\Pi_{(y:B)} C(x,y) \vdash \mathrm{id}(f) \equiv \lambda g.g(f)} \qquad \dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash \Pi_{(x:A)}\Pi_{(y:B)} C(x,y)\,\mathrm{Type}}}{\Gamma, f \vdash f}\,\delta}{\Gamma, f \vdash (\lambda g.g)(f) \equiv f}\,\beta}{\Gamma, f \vdash \mathrm{id}(f) \equiv f}$$

Note that for the $\beta$ rule on the RHS, we can rename $f$ to $g$ that is because when compare to the rule description $x$ is $f$ and $b(x)$ is $f$ i.e depends fully on the signature of $x$. On the other hand, we have that:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Gamma, f \vdash f \equiv \lambda x.\lambda y.f(x,y)} \qquad \dfrac{\vdots}{\Gamma, f \vdash \mathrm{id}(f) \equiv f}}{\Gamma, f \vdash \mathrm{id}(f) \equiv \lambda x.\lambda y.f(x,y)}\,\lambda\text{-eq}}{\Gamma \vdash \lambda f.\,\mathrm{id}(f) \equiv \lambda f.\lambda x.\lambda y.f(x,y)} \qquad \dfrac{\Gamma \vdash \mathrm{id}}{\Gamma \vdash \lambda f.\,\mathrm{id}(f) \equiv f}\,\eta}{\Gamma \vdash \mathrm{id} \equiv \lambda f.\lambda x.\lambda y.f(x,y)}$$

For $f \equiv \lambda x.\lambda y.f(x,y)$ we just used repeated application of $\eta$ rule. Now, we have finished the first side, let's move to the swap case, we will not to the proof tree here, but give the following judgemental equalities

$$\sigma(\sigma(f)) \equiv \sigma(\lambda y'.\lambda x'.f(x',y')) \equiv \lambda x.\lambda y.\big[(\lambda y'.\lambda x'.f(x',y'))(y,x)\big]$$
$$\sigma(\sigma(f))(x'',y'') \equiv \lambda x.\lambda y.\big[(\lambda y'.\lambda x'.f(x',y'))(y,x)\big](x'',y'') \equiv (\lambda y'.\lambda x'.f(x',y'))(y'',x'') \equiv f(x'',y'')$$

Note that the first line represents the unwrapping of $\lambda$-definition of $\sigma$, while the second line represents the repeated $\beta$ application. Thus we have that (with renaming)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Gamma, f, x : A, y : B \vdash \sigma(\sigma(f))(x,y) \equiv f(x,y)}}{\Gamma, f \vdash \lambda x.\lambda y.\sigma(\sigma(f))(x,y) \equiv \lambda x.\lambda y.f(x,y)}\,\lambda\text{-eq} \qquad \dfrac{\dfrac{\dfrac{\vdots}{\Gamma, f \vdash \sigma(\sigma(f))}}{\Gamma, f \vdash \lambda y.\sigma(\sigma(f))(y) \equiv \sigma(\sigma(f))}\,\eta}{\Gamma, f \vdash \lambda x.\lambda y.\sigma(\sigma(f))(y)(x) \equiv \sigma(\sigma(f))}\,\eta}{\Gamma, f \vdash \lambda x.\lambda y.\sigma(\sigma(f))(x,y) \equiv \sigma(\sigma(f))}}{\Gamma, f \vdash \sigma(\sigma(f)) \equiv \lambda x.\lambda y.f(x,y)}\,\lambda\text{-eq}}{\Gamma \vdash \lambda f.\sigma(\sigma(f)) \equiv \lambda f.\lambda x.\lambda y.f(x,y)}\,\lambda\text{-eq}}{\Gamma \vdash \lambda f.(\sigma \circ \sigma)(f) \equiv \lambda f.\lambda x.\lambda y.f(x,y)}$$

**(TO DO)** We have to check the order of the multi-variable function on the RHS. Thus, we have that:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash \mathrm{id} \equiv \lambda f.\lambda x.\lambda y.f(x,y)} \qquad \dfrac{\vdots}{\Gamma \vdash \lambda f.(\sigma \circ \sigma)(f) \equiv \lambda f.\lambda x.\lambda y.f(x,y)} \qquad \dfrac{\dfrac{\vdots}{\Gamma \vdash \sigma \circ \sigma}}{\Gamma \vdash \lambda f.(\sigma \circ \sigma)(f) \equiv \sigma \circ \sigma}\eta}{\Gamma \vdash \sigma \circ \sigma \equiv \mathrm{id}}$$

Note that the RHS we have created the $\sigma \circ \sigma$ from the composition of swap functions. Thus we have the equality as needed. □

## 2.5 Natural Number

Now, we are ready to consider the formal specification of the type of natural numbers $\mathbb{N}$, which is archetypal example of an inductive type. Similar to dependent function there are 4 rules (Formation, Introduction, Elimination (induction), Computation)

**Definition 24.** ***(Formation Rule of*** $\mathbb{N}$***)*** *Assert how the type* $\mathbb{N}$ *can be formed where* $\mathbb{N}$ *is to be a type in the empty context, as we have:*

$$\dfrac{}{\vdash \mathbb{N}\,\mathrm{Type}}\,\mathbb{N}\text{-}form$$

**Definition 25.** ***(Introduction Rules of*** $\mathbb{N}$***)*** *Peano' first axiom postulates that* $0$ *is a natural number. The introduction rule gives zero-element and successor function:*

$$\dfrac{}{\vdash 0_{\mathbb{N}} : \mathbb{N}} \qquad \dfrac{}{\vdash \mathrm{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}}$$

The classical induction principle of the natural number tells us what we have to do in order to show that $\forall(n \in \mathbb{N})P(n)$ holds for a predicate $P$ over $\mathbb{N}$.

**Definition 26.** ***(Induction Principle of*** $\mathbb{N}$***)*** *We think of type family* $P$ *over* $\mathbb{N}$ *as predicate over* $\mathbb{N}$ *as:*

$$\dfrac{\Gamma, n : \mathbb{N} \vdash P(n)\,\mathrm{Type} \qquad \Gamma \vdash p_0 : P(0_{\mathbb{N}}) \qquad \Gamma \vdash p_S : \Pi_{(n:\mathbb{N})}P(n) \to P\big(\mathrm{succ}_{\mathbb{N}}(n)\big)}{\Gamma \vdash \mathrm{ind}_{\mathbb{N}}(p_0, p_S) : \Pi_{(n:\mathbb{N})}P(n)}$$

*tells us what is need in order to construct a dependent function* $\Pi_{(n:\mathbb{N})}P(n)$*, which are base case i.e element* $p_0 : P(0_{\mathbb{N}})$ *and induction step i.e function of type of* $P(n) \to P(\mathrm{succ}_{\mathbb{N}}(n))$ *for all* $n : \mathbb{N}$*.*

**Remark 8.** ***(Alternative Induction Principle)*** *We just create a functions with both base case and induction step as the input:*

$$\dfrac{\Gamma, n : \mathbb{N} \vdash P(n)\,\mathrm{Type}}{\Gamma \vdash \mathrm{ind}_{\mathbb{N}} : P(0_{\mathbb{N}}) \to \Big(\big(\Pi_{(n:\mathbb{N})}P(n) \to P(\mathrm{succ}_{\mathbb{N}}(n))\big) \to \Pi_{(n:\mathbb{N})}P(n)\Big)}$$

*We can derive this alternative from the original formulation above. Starting with We can construct* $\Gamma \vdash P(0_{\mathbb{N}})\,\mathrm{Type}$ *as:*

$$\dfrac{\Gamma \vdash 0_{\mathbb{N}} : \mathbb{N} \qquad \Gamma, n : \mathbb{N} \vdash P(n)\,\mathrm{Type}}{\Gamma \vdash P(0_{\mathbb{N}})\,\mathrm{Type}}\,S$$

*Similarly, we can construct* $\Pi_{n:\mathbb{N}}\big(P(n) \to P(\mathrm{succ}_{\mathbb{N}}(n))\big)\,\mathrm{Type}$ *as:*

$$\dfrac{\dfrac{}{\Gamma, n : \mathbb{N} \vdash P(n)\,\mathrm{Type}} \qquad \dfrac{\dfrac{\Gamma \vdash \mathrm{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}}{\Gamma, n : \mathbb{N} \vdash \mathrm{succ}(n) : \mathbb{N}}\,ev \qquad \dfrac{\Gamma \vdash \mathbb{N}\,\mathrm{Type} \qquad \Gamma, n' : \mathbb{N} \vdash P(n')}{\Gamma, n : \mathbb{N}, n' : \mathbb{N} \vdash P(n')\,\mathrm{Type}}\,W}{\Gamma, n : \mathbb{N} \vdash P(\mathrm{succ}_{\mathbb{N}}(n))\,\mathrm{Type}}\,S}{\Gamma, n : \mathbb{N} \vdash P(n) \to P(\mathrm{succ}_{\mathbb{N}}(n))\,\mathrm{Type}}\,\to}{\Gamma \vdash \Pi_{n:\mathbb{N}}\big(P(n) \to P(\mathrm{succ}_{\mathbb{N}}(n))\big)\,\mathrm{Type}}\,\Pi$$

*By the weakening and variable laws, we have the following*

$$\Gamma, p_0 : P(0_{\mathbb{N}}), p_S : \Pi_{(n:\mathbb{N})}P(n) \to P\big(\mathrm{succ}_{\mathbb{N}}(n)\big) \vdash p_0 : P(0_{\mathbb{N}})$$
$$\Gamma, p_0 : P(0_{\mathbb{N}}), p_S : \Pi_{(n:\mathbb{N})}P(n) \to P\big(\mathrm{succ}_{\mathbb{N}}(n)\big) \vdash p_S : \Pi_{(n:\mathbb{N})}P(n) \to P\big(\mathrm{succ}_{\mathbb{N}}(n)\big)$$
$$\Gamma, p_0 : P(0_{\mathbb{N}}), p_S : \Pi_{(n:\mathbb{N})}P(n) \to P\big(\mathrm{succ}_{\mathbb{N}}(n)\big), n : \mathbb{N} \vdash P(n)\,\mathrm{Type}$$

*The original induction principle gives us the premise, and we can use the λ-abstraction twice to obtain a function (as the λ-abstraction removes the notion of element of the context).*

$$\frac{\Gamma, p_0 : P(0_{\mathbb{N}}), p_S : \Pi_{(n:\mathbb{N})} P(n) \to P\big(\mathrm{succ}_{\mathbb{N}}(n)\big) \vdash \mathrm{ind}_{\mathbb{N}}(p_0, p_S) : \Pi_{(n:\mathbb{N})} P(n)}{\Gamma \vdash \mathrm{ind}_{\mathbb{N}} : P(0_{\mathbb{N}}) \to \Big(\big(\Pi_{(n:\mathbb{N})} P(n) \to P(\mathrm{succ}_{\mathbb{N}}(n))\big) \to \Pi_{(n:\mathbb{N})} P(n)\Big)} (\lambda * 2)$$

*On the other hand, the original rule can be derived using the elimination rule and substitution rules (i.e application of the inputs to the given function) as, for example*

$$\frac{\dfrac{}{\Gamma \vdash p_0 : P(0_{\mathbb{N}})} \qquad \dfrac{\dfrac{\Gamma, n : \mathbb{N} \vdash P(n)\,\mathrm{Type}}{\Gamma \vdash \mathrm{ind}_{\mathbb{N}}}}{\Gamma, p_0' : P(0_{\mathbb{N}}) \vdash \mathrm{ind}_{\mathbb{N}}(p_0')}\, ev}{\Gamma \vdash \mathrm{ind}_{\mathbb{N}}(p_0) : \big(\Pi_{(n:\mathbb{N})} P(n) \to P(\mathrm{succ}_{\mathbb{N}}(n))\big) \to \Pi_{(n:\mathbb{N})} P(n)}\, S$$

**Definition 27.** ***(Computation Rule of*** $\mathbb{N}$***)*** *Asserts that the dependent function:* $\mathrm{ind}_{\mathbb{N}}(p_0, p_S)$ *behave as expected when it is applied to a natural number. We start with the base case i.e its behavior at* $0_{\mathbb{N}}$ *as:*

$$\frac{\Gamma, n : \mathbb{N} \vdash P(n)\,\mathrm{Type} \qquad \Gamma \vdash p_0 : P(0_{\mathbb{N}}) \qquad \Gamma \vdash p_S : \Pi_{(n:\mathbb{N})} P(n) \to P\big(\mathrm{succ}_{\mathbb{N}}(n)\big)}{\Gamma \vdash \mathrm{ind}_{\mathbb{N}}(p_0, p_S, 0_{\mathbb{N}}) \equiv p_0 : P(0_{\mathbb{N}})}$$

*On the other hand, the inductive step is given to be:*

$$\frac{\Gamma, n : \mathbb{N} \vdash P(n)\,\mathrm{Type} \qquad \Gamma \vdash p_0 : P(0_{\mathbb{N}}) \qquad \Gamma \vdash p_S : \Pi_{(n:\mathbb{N})} P(n) \to P\big(\mathrm{succ}_{\mathbb{N}}(n)\big)}{\Gamma \vdash \mathrm{ind}_{\mathbb{N}}(p_0, p_S, \mathrm{succ}_{\mathbb{N}}(n)) \equiv p_S(n, \mathrm{ind}_{\mathbb{N}}(p_0, p_S, n)) : P(\mathrm{succ}_{\mathbb{N}})}$$

*Note that* $\mathrm{ind}_{\mathbb{N}}(p_0, p_S, n) : P(n)$ *and* $p_S$ *can take* 2 *arguments.*

## 2.6 Addition on the Natural Number

The induction principle of $\mathbb{N}$ and be used to derive all the familiar properties about natural number. This requires a few more in ingredient of Martin-Lof's dependent type theory. We will need, for example, the identity type to specify all of forming types in Martin-Lof's dependent type.

**Definition 28.** ***(Addition)*** *We have:* $\mathrm{add}_{\mathbb{N}} : \mathbb{N} \to \big(\mathbb{N} \to \mathbb{N}\big)$*, satisfying the specification:*

$$\mathrm{add}_{\mathbb{N}}(m, 0_{\mathbb{N}}) \equiv m$$
$$\mathrm{add}_{\mathbb{N}}(m, \mathrm{succ}_{\mathbb{N}}(n)) \equiv \mathrm{succ}_{\mathbb{N}}(\mathrm{add}_{\mathbb{N}}(m, n))$$

*We will write* $m + n$ *for* $\mathrm{add}_{\mathbb{N}}(m, n)$*. This can be constructed by induction on the second variables, follows* $m : \mathbb{N} \vdash \mathrm{add}_{\mathbb{N}}(m) : \mathbb{N} \to \mathbb{N}$*. The induction principle of* $\mathbb{N}$ *is used with the family of type* $P(n) := \mathbb{N}$ *indexed by* $n : \mathbb{N}$ *in context* $m : \mathbb{N}$*:*

$$\frac{\dfrac{\dfrac{\vdots}{m : \mathbb{N} \vdash \text{add-zero}_{\mathbb{N}}(m) := m : \mathbb{N}} \quad \dfrac{\vdots}{m : \mathbb{N} \vdash \text{add-succ}_{\mathbb{N}}(m) : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})}}{m : \mathbb{N} \vdash \mathrm{ind}_{\mathbb{N}}(\text{add-zero}_{\mathbb{N}}(m), \text{add-succ}_{\mathbb{N}}(m)) : \mathbb{N} \to \mathbb{N}}}{m : \mathbb{N} \vdash \mathrm{add}_{\mathbb{N}}(m) := \mathrm{ind}_{\mathbb{N}}(\text{add-zero}_{\mathbb{N}}(m), \text{add-succ}_{\mathbb{N}}(m)) : \mathbb{N} \to \mathbb{N}}$$

*We can use the specification above to define the functions, in which the element* $\text{add-zero}_{\mathbb{N}}(m) : \mathbb{N}$ *in context* $m : \mathbb{N}$ *is* $m : \mathbb{N}$*. And,* $\text{add-succ}_{\mathbb{N}}(m) : \mathbb{N} \to \big(\mathbb{N} \to \mathbb{N}\big)$ *is* $\text{add-succ}_{\mathbb{N}}(m, n, x) \equiv \mathrm{succ}_{\mathbb{N}}(x)$ *as we have:*

$$\begin{aligned}\mathrm{add}_{\mathbb{N}}(m, \mathrm{succ}_{\mathbb{N}}(n)) &\equiv \mathrm{ind}_{\mathbb{N}}\left(\text{add-zero}_{\mathbb{N}}(m), \text{add-succ}_{\mathbb{N}}(m), \mathrm{succ}_{\mathbb{N}}(n)\right) \\ &\equiv \text{add-succ}_{\mathbb{N}}(m, n, \mathrm{add}_{\mathbb{N}}(m, n)) \\ &\equiv \mathrm{succ}_{\mathbb{N}}(\mathrm{add}_{\mathbb{N}}(m, n))\end{aligned}$$

*Note that we have used the computation rule. The formal derivation for the construction of* $\text{add-succ}_{\mathbb{N}}$ *is:*

$$\frac{\dfrac{\dfrac{}{\vdash \mathbb{N}\,\mathrm{Type}} \quad \dfrac{\dfrac{}{\vdash \mathbb{N}\,\mathrm{Type}} \quad \dfrac{}{\vdash \mathrm{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}}}{n : \mathbb{N} \vdash \mathrm{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}}\, W}{m : \mathbb{N}, n : \mathbb{N} \vdash \mathrm{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}}\, W}{m : \mathbb{N} \vdash \lambda n.\, \mathrm{succ}_{\mathbb{N}} : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})}\, \lambda}{m : \mathbb{N} \vdash \text{add-succ}_{\mathbb{N}}(m) := \lambda n.\, \mathrm{succ}_{\mathbb{N}} : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})}$$

**Remark 9.** *We have a recursive definition of the addition. However, we can't show that* $0_{\mathbb{N}} + n \equiv n$ *or* $\text{succ}_{\mathbb{N}}(m) + n = \text{succ}_{\mathbb{N}}(n, m)$*. Dependent type theory with its inductive types does not provide any means to prove such judgmental equalities, and we will need identity type to do so.*

**Definition 29.** ***(Pattern Matching)*** *If we want to define a dependent function* $f : \Pi_{(n:\mathbb{N})} P(n)$ *by induction on* $n$ *using:* $p_0 : P(0_{\mathbb{N}})$ *and* $p_S : \Pi_{(n:\mathbb{N})} P(n) \to P(\text{succ}_{\mathbb{N}}(n))$*, then we can present its definition as:*

$$\begin{aligned} f(0_{\mathbb{N}}) &:= p_0 \\ f(\text{succ}_{\mathbb{N}}(n)) &:= p_S(n, f(n)) \end{aligned}$$

*To recover the dependent function* $p_S$ *from the expression* $p_S(n, f(n))$*, we can replace all occurrence of the term* $f(n)$ *in the expression with* $x : P(n)$*. In other words, when a sub-expression of* $p_S(n, f(n))$ *matches* $f(n)$*, we replace that sub-expression by* $x$*.*

**Remark 10.** *(Pattern Matchine in* $\text{add}(\cdot, \cdot)$*) We can observe that we have* $p_S := \text{add-succ}_{\mathbb{N}}(m, \cdot)$ *with* $f = \text{add}(m, \cdot)$*, then with expression* $f(\text{succ}_{\mathbb{N}}(n)) := p_S(n, f(n))$*, we have:*

$$\begin{aligned} \text{add}(m)(\text{succ}_{\mathbb{N}}(n)) &:= \text{add-succ}_{\mathbb{N}}(m)(n)(\text{add}(m)(n)) \\ &:= \text{succ}(\text{add}(m)(n)) \end{aligned}$$

*where the second line is the specification With* $f(n)$ *being* $\text{add}(m)(n)$*, we an define it by replacing it with* $x$*, that is we get* $\text{add-succ}_{\mathbb{N}}(m)(n)(x) \equiv \text{succ}(x)$*, as needed. The reason we can replace is follows from the computation rule (as the second parameter of* $p_S$ *gets ride of* succ*).*

**Definition 30.** ***(Fibonacci Function)*** *It is a function* $F : \mathbb{N} \to \mathbb{N}$ *that is defined as:*

$$\begin{aligned} F(0_{\mathbb{N}}) &:= 0_{\mathbb{N}} \\ F(1_{\mathbb{N}}) &:= 1_{\mathbb{N}} \\ F(\text{succ}_{\mathbb{N}}(\text{succ}_{\mathbb{N}}(n))) &:= F(\text{succ}_{\mathbb{N}}(n)) + F(n) \end{aligned}$$

*Since* $F(\text{succ}_{\mathbb{N}}(\text{succ}_{\mathbb{N}}(n)))$ *is defined using both* $F(\text{succ}_{\mathbb{N}}(n))$ *and* $F(n)$*, it isn't immediately clear how to present* $F$ *by the usual induction principle of* $\mathbb{N}$*. First we note that via the computational rule, we can unwrap the inductive function as:*

$$\begin{aligned} \text{ind}_{\mathbb{N}}(p_0, p_S, \text{succ}_{\mathbb{N}}(\text{succ}_{\mathbb{N}}(n))) &\equiv p_S(\text{succ}_{\mathbb{N}}(n), \text{ind}_{\mathbb{N}}(p_0, p_S, \text{succ}_{\mathbb{N}}(n))) \\ &\equiv p_S(\text{succ}_{\mathbb{N}}(n), p_S(n, \text{ind}_{\mathbb{N}}(p_0, p_S, n))) \end{aligned}$$

*Usually we define* $F \equiv \text{ind}_{\mathbb{N}}(p_0, p_S) : \mathbb{N} \to \mathbb{N}$*, in which the equation above is translated to the equation below:*

$$\begin{aligned} F(\text{succ}_{\mathbb{N}}(\text{succ}_{\mathbb{N}}(n))) &\equiv p_S(\text{succ}_{\mathbb{N}}(n), F(\text{succ}_{\mathbb{N}}(n))) \\ &\equiv p_S(\text{succ}_{\mathbb{N}}(n), p_S(n, F(n))) \end{aligned}$$

*If we let* $p_S(a, b) = b + F(\text{prec}_{\mathbb{N}}(a))$*, where* $\text{prec}_{\mathbb{N}}(x)$ *is defined as* $\text{prec}_{\mathbb{N}}(1) := 0$ *and* $\text{prec}_{\mathbb{N}}(\text{succ}_{\mathbb{N}}(x)) := x$

$$\begin{aligned} F(n+2) &= p_S(n+1, F(n+1)) = F(n+1) + F(n) \\ F(n+2) &= p_S(n+1, p_S(n, F(n))) = p_S(n, F(n)) + F(n) = F(n) + F(n-1) + F(n) \end{aligned}$$

**Remark 11.** *(Max Function) We can define the max/min function as follows* $\max_{\mathbb{N}}, \min_{\mathbb{N}} : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})$

$$\begin{aligned} \max_{\mathbb{N}}(0, b) &\equiv b & \min_{\mathbb{N}}(0, b) &\equiv 0 \\ \max_{\mathbb{N}}(a, 0) &\equiv a & \min_{\mathbb{N}}(a, 0) &\equiv 0 \\ \max_{\mathbb{N}}(a+1, b+1) &\equiv 1 + \max_{\mathbb{N}}(a, b) & \min_{\mathbb{N}}(a+1, b+1) &\equiv 1 + \min_{\mathbb{N}}(a, b) \end{aligned}$$

*To implement this, we would have to see the tuple* $(a, b)$ *as one number, in which the zero value is when either* $a$ *or* $b$ *is zero, while the* succ *is defined to be* $(a, b) \mapsto (a+1, b+1)$*.*

## 2.7 More Inductive Types

**Remark 12.** *(General Pattern) Just like the type of natural number, other inductive types are also specified by their constructors, and induction principles and their computation rules, which can be described as follows:*

- ***(Constructors):*** *tell what structure the inductive type comes equipped with. These may be any finite number of constructors, event no constructors at all.*

- ***(Induction Principle):*** *specifies the data that should be provided in order to construct a section of an arbitrary type family over the induction type.*
- ***(Computation Rules):*** *asserts that inductively defined section agrees on the constructors with the data that was used to define the section.*

*Since any inductively defined function is entirely determined by its behavior on the constructors, we can again present such inductive definition by pattern matching.*

**Definition 31.** ***(Unit Type)*** *A type* $1$ *equipped with a term:* $* : \mathbf{1}$ *satisfying the induction principle that for any family of types* $P(x)$ *indexed by* $x : \mathbf{1}$ *that is a function as:*

$$\text{ind}_1 : P(*) \to \Pi_{(x:\mathbf{1})} P(x)$$

*for which the computation rule is* $\text{ind}_1(p, *) \equiv p$ *holds. Alternatively, a definition* $f : \Pi_{(x:\mathbf{1})} P(x)$ *by induction using* $p : P(x)$ *can be presented by pattern matching* $f(*) := p$*:*

**Remark 13.** *(Special Case of Unit Type) Arises when* $P$ *doesn't actually depend on* $\mathbf{1}$*. Given a type* $A$*, then we can weaken it to obtain the constant family over* $\mathbf{1}$ *with* $A$*. Then the induction principle is a function of* $\text{ind}_{\mathbf{1}} : A \to (1 \to A)$*. That is for every* $x : A$*, we have a function* $\text{pt}_x := \text{ind}_1(x) : \mathbf{1} \to A$*. Given to be:*

$$\frac{\dfrac{\Gamma \vdash \mathbf{1}\,\text{Type} \qquad \Gamma \vdash A\,\text{Type}}{\Gamma, * : \mathbf{1} \vdash A\,\text{Type}}\, W}{\Gamma \vdash \text{ind}_1 : A \to 1 \to A\,\text{Type}}$$

The empty type is a degenerative example of an inductive type. It doesn't come equipped with any constructors and therefore, there are also no computation rules. Asserting that any type family has a section i.e we can prove anything.

**Definition 32.** ***(Empty Type)*** *A type* $\emptyset$ *satisfying to inductive principle that for any family of type* $P(x)$ *indexed by* $x : \emptyset$*, there is a term:*

$$\text{ind}_\emptyset : \Pi_{(x:\emptyset)} P(x)$$

**Remark 14.** *(Special Case of Empty Type) In the special case like above, we have a function* ex-falso $:= \text{ind}_\emptyset : \emptyset \to A$ *for any type* $A$*. To obtained this, we weaken* $A$ *to get constant family over* $\emptyset$ *with value* $A$*, and then the function follows from the induction principle.*

**Definition 33.** ***(Negation/Empty)*** *For any type* $A$*, we define negation of* $A$ *by:* $\neg A := A \to \emptyset$*. Type* $A$ *is empty if it comes equipped with an element of type* $\neg A$ *i.e* $\text{is-empty}(A) \equiv A \to \emptyset$

**Definition 34.** ***(Proof of Negation)*** *The proof of* $\neg A$ *is given by assuming that* $A$ *holds and then constructing an element of the empty type. That is we prove* $\neg A$ *by assuming* $A$ *and deriving a contradiction.*

**Remark 15.** *(Proof of Contradiction vs Negation) This is different from the proof by contradiction because for proof by contradiction of a property* $P$ *is an argument where we conclude that* $P$ *holds after showing that* $\neg P$ *implies a contradiction i.e the use of double negation elimination* $\neg\neg P \implies P$*.*
*In type theory, the type* $\neg\neg A$ *is type of function as* $(A \to \emptyset) \to \emptyset$*. This is different from* $A$ *itself as it isn't possible to construct* $\neg\neg A \to A$ *unless we know more about* $A$*.*

**Proposition 1.** *For any 2 types* $P$ *and* $Q$*, there is a function:*

$$(P \to Q) \to (\neg Q \to \neg P)$$

*Proof.* We will define the desired function as $\lambda$-abstraction. Assuming we have a function $f : P \to Q$, $\tilde{q} : \neg Q := Q \to \emptyset$ and $p : P$, then we have: $\lambda f.\lambda\tilde{q}.\lambda p.\tilde{q}(f(p)) : (P \to Q) \to (\neg Q \to \neg P)$. The natural deduction tree is similar to how we define the composition function. □

**Definition 35.** ***(Co-Product)*** *Let* $A$ *and* $B$ *be types, the co-product* $A + B$ *is a type that comes equipped with:* $\text{inl} : A \to A + B$ *and* $\text{inr} : B \to A + B$*. Satisfying the induction principle that for any family of types* $P(x)$ *index by* $x : A + B$*, there is a term (in the first line):*

$$\text{ind}_+ : \left(\Pi_{(x:A)} P(\text{inl}(x))\right) \to \left(\Pi_{(y:B)} P(\text{inr}(y))\right) \to \Pi_{(z:A+B)} P(z)$$
$$\text{ind}_+\left(f, g, \text{inl}(x)\right) \equiv f(x) \qquad \text{ind}_+\left(f, g, \text{inr}(x)\right) \equiv g(x)$$

*for which the computation rule can be given by, the second line.*

***(Pattern Matching):*** *Dependency function* $h : \Pi_{(x:A+B)}P(x)$ *defined by induction using* $f : \Pi_{(x:A)}P(\mathrm{inl}(x))$ *and* $g : \Pi_{(y:B)}P(\mathrm{inr}(y))$ *can be presented by pattern matching as:*

$$h(\mathrm{inl}(x)) := f(x) \qquad h(\mathrm{inr}(y)) := g(y)$$

*Sometimes, we write* $[f, g]$ *for the function* $\mathrm{ind}_+(f, g)$*. The co-product of 2 types is sometimes also called the disjoint sum.*

**Remark 16.** *(Special Case for Co-Product) By induction principle of co-product, we obtain a function:*

$$\mathrm{ind}_+ : (A \to X) \to \big((B \to X) \to (A + B \to X)\big)$$

*for any type* $X$*. This is very similar to the elimination rule of disjunction in the first order logic:* $A, P, P'$ *and* $Q$ *are proposition, which:* $(P \implies Q) \implies \big((P' \implies Q) \implies (P \vee P') \implies Q\big)$*. Under this interpretation of type theory the co-product is indeed the disjunction.*

**Remark 17.** *The induction principle for co-products gives us the map:* $f + g : A + B \to A' + B'$ *for every* $f : A \to A'$ *and* $g : B \to B'$*. Indeed, the map* $f + g$ *is defined to be:*

$$(f + g)\big(\mathrm{inl}(x)\big) := \mathrm{inl}(f(x)) \qquad (f + g)\big(\mathrm{inr}(x)\big) := \mathrm{inl}(g(y))$$

*where we have* $f + g := \mathrm{ind}_+(\mathrm{inl}_{A'+B'} \circ f, \mathrm{inr}_{A'+B'} \circ g)$ *note that we have the following signature:* $\mathrm{inl} \circ f : A \to A' \to A' + B'$ *and* $\mathrm{inr} \circ f : B \to B' \to A' + B'$*.*

**Proposition 2.** *Consider 2 types* $A$ *and* $B \equiv \emptyset$*. Then there is a function* $(A + B) \to A$

*Proof.* Since $B$ is empty, we have that $\tilde{b} : B \to \emptyset$ together with the function ex-falso : $\emptyset \to A$ note that we have ex-false $\circ \tilde{b} : B \to A$. On the other hand, we have the identity function $\mathrm{id}_A : A \to A$. Thus, we can construct the function $(A + B) \to A$ with the inductive properties of the co-product $A + B$ we have the function $\mathrm{ind}_+(\mathrm{id}_A, \text{ex-false} \circ \tilde{b}) : A + B \to B$, as needed. □

**Definition 36.** ***(Booleans)*** *Inductive type* bool *that comes equipped with* false : bool *and* true : bool*. The induction principle of the booleans asserts that for any family of types* $P(x)$ *indexed by* $x$ : bool *there is a term:*

$$\text{ind-bool} : P(\text{false}) \to (P(\text{true}) \to \Pi_{x:\text{bool}}P(x))$$

*With the computation rules of:* $\text{ind-bool}(p_0, p_1, \text{false}) := p_0$ *and* $\text{ind-bool}(p_0, p_1, \text{true}) := p_1$

**Proposition 3.** *We can construct the following functions:* neg-bool : bool → bool*, the boolean conjunction operation* $- \wedge - : \text{bool} \to (\text{bool} \to \text{bool})$ *and the boolean disjunction operation* $- \vee - : \text{bool} \to (\text{bool} \to \text{bool})$

*Proof.* We can define for eacgh of the following functions: (1) Negation neg-bool $\equiv$ ind-bool(true, false). (2) conjunction $\lambda a.\lambda b.\,\text{ind-bool}(\text{false}, a, b)$ (3) disjunction $\lambda a.\lambda b.\,\text{ind-bool}(a, \text{true}, b)$ □

**Remark 18.** *(Notes on Constructing Integer) The set of integers is usually defined as a quotient of the set* $\mathbb{N}\times\mathbb{N}$*, where* $\big((n, m) \sim (n', m')\big) := \big(n + m' = n' + m\big)$*. The identity type haven't been introduced yet. Furthermore, there are no quotient types in Martin-Lof's dependent type theory (much later on that we will see it).*

**Definition 37.** ***(Integer)*** *The integer is a type* $\mathbb{Z} :\equiv \mathbb{N} + (1 + \mathbb{N})$*, with the inclusion function of the positive and negative integer:* $\text{in-pos} :\equiv \mathrm{inr} \circ \mathrm{inr} : \mathbb{N} \to \mathbb{Z}$ *and* $\text{in-neg} :\equiv \mathrm{inl} : \mathbb{N} \to \mathbb{Z}$ *and the constants can be defined as:*

$$-1_{\mathbb{Z}} :\equiv \text{in-neg}(0) \qquad 0_{\mathbb{Z}} :\equiv \mathrm{inr}(\mathrm{inl}(*)) \qquad 1_{\mathbb{Z}} :\equiv \text{in-pos}(0)$$

*It is possible to derive an inductive principle, which asserts that for any type family* $P$ *over* $\mathbb{Z}$*. We have the dependent function* $f : \Pi_{(k:\mathbb{Z})}P(k)$ *in which:*

$$\begin{aligned}
f(-1_{\mathbb{Z}}) &\equiv p_{-1} & p_{-1} &: P(-1_{\mathbb{Z}}) \\
f(\text{in-neg}(\text{succ}_{\mathbb{N}}(n))) &\equiv p_{-S}(n, f(\text{in-neg}(n))) & p_{-S} &: \Pi_{(n:\mathbb{N})}P(\text{in-neg}(n)) \to P(\text{in-neg}(\text{succ}_{\mathbb{N}}(n))) \\
f(0_{\mathbb{Z}}) &\equiv p_0 & p_0 &: P(0_{\mathbb{Z}}) \\
f(1_{\mathbb{Z}}) &\equiv p_1 & p_1 &: P(1_{\mathbb{Z}}) \\
f(\text{in-pos}(\text{succ}_{\mathbb{N}}(n))) &\equiv p_S(n, f(\text{in-pos}(n))) & p_S &: \Pi_{(n:\mathbb{N})}P(\text{in-pos}(n)) \to P(\text{in-pos}(\text{succ}_{\mathbb{N}}(n)))
\end{aligned}$$

*Note that, semantically, we have* $\text{in-neg}(n) \equiv -n - 1$ *and* $\text{in-pos}(n) \equiv n + 1$

We have the following successor function to be defined as:

**Definition 38.** ***(Successor/Predecessor Function on Integer)*** *Both* $\text{succ}_{\mathbb{Z}} : \mathbb{Z} \to \mathbb{Z}$ *and* $\text{pred}_{\mathbb{Z}} : \mathbb{Z} \to \mathbb{Z}$ *can be defined inductively as*

$$\begin{aligned}
\text{succ}_{\mathbb{Z}}(-1_{\mathbb{Z}}) &\equiv 0_{\mathbb{Z}} & \text{pred}_{\mathbb{Z}}(-1_{\mathbb{Z}}) &\equiv \text{in-neg}(1_{\mathbb{N}}) \\
\text{succ}_{\mathbb{Z}}(0_{\mathbb{Z}}) &\equiv 1_{\mathbb{Z}} & \text{pred}_{\mathbb{Z}}(0_{\mathbb{Z}}) &\equiv -1_{\mathbb{Z}} \\
\text{succ}_{\mathbb{Z}}(1_{\mathbb{Z}}) &\equiv \text{in-pos}(1_{\mathbb{N}}) & \text{pred}_{\mathbb{Z}}(1_{\mathbb{Z}}) &\equiv 0_{\mathbb{Z}} \\
\text{succ}_{\mathbb{Z}}(\text{in-neg}(\text{succ}_{\mathbb{N}}(n))) &\equiv \text{in-neg}(n) & \text{pred}_{\mathbb{Z}}(\text{in-neg}(\text{succ}_{\mathbb{N}}(n))) &\equiv \text{in-neg}(\text{succ}_{\mathbb{N}}(\text{succ}_{\mathbb{N}}(n))) \\
\text{succ}_{\mathbb{Z}}(\text{in-pos}(\text{succ}_{\mathbb{N}}(n))) &\equiv \text{in-pos}(\text{succ}_{\mathbb{N}}(\text{succ}_{\mathbb{N}}(n))) & \text{pred}_{\mathbb{Z}}(\text{in-pos}(\text{succ}_{\mathbb{N}}(n))) &\equiv \text{in-pos}(n)
\end{aligned}$$

*For the case of* $\text{succ}_{\mathbb{Z}}(\text{in-neg}(\text{succ}_{\mathbb{N}}(n)))$*, so we have* $\text{in-neg}(n+1) \equiv -n-1-1$ *and so* $\text{succ}_{\mathbb{Z}}(-n-2) \equiv -n-1$*.*

**Definition 39.** ***(Addition)*** *We can define* $\text{add}_{\mathbb{Z}} : \mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})$ *in similar manners to the natural number case that is we define* $\text{add}_{\mathbb{Z}}(m)$ *(this is added by* $m$*) and* $\text{add}_{\mathbb{Z}}(\text{succ}_{\mathbb{Z}}(\text{in-neg}(m)))$ *(this is added by* $-m$*)*

$$\begin{aligned}
\text{add}_{\mathbb{Z}}(m)(-1_{\mathbb{Z}}) &\equiv \text{pred}_{\mathbb{Z}}(m) & \text{add}_{\mathbb{Z}}(m)(\text{in-neg}(\text{succ}_{\mathbb{N}}(n))) &\equiv \text{add}_{\mathbb{Z}}(\text{prec}_{\mathbb{Z}}(m))(\text{in-neg}(n)) \\
\text{add}_{\mathbb{Z}}(m)(0_{\mathbb{Z}}) &\equiv m & \text{add}_{\mathbb{Z}}(m)(\text{in-pos}(\text{succ}_{\mathbb{N}}(n))) &\equiv \text{add}_{\mathbb{Z}}(\text{succ}_{\mathbb{Z}}(m))(\text{in-pos}(n)) \\
\text{add}_{\mathbb{Z}}(m)(1_{\mathbb{Z}}) &\equiv \text{succ}_{\mathbb{N}}(m)
\end{aligned}$$

*For the first case of the second column, we have* $(m-1)+(-n-1) = m-n-2$*, similar for the second case, we have that* $m+n+2 = (m+1)+(n+1)$

**Definition 40.** ***(Inverse)*** *We can define* $\text{neg}_{\mathbb{Z}} : \mathbb{Z} \to \mathbb{Z}$ *as:*

$$\begin{aligned}
\text{neg}_{\mathbb{Z}}(-1_{\mathbb{Z}}) &\equiv 1_{\mathbb{Z}} & \text{neg}_{\mathbb{Z}}(\text{in-neg}(\text{succ}_{\mathbb{N}}(n))) &\equiv \text{in-pos}(\text{succ}_{\mathbb{N}}(n)) \\
\text{neg}_{\mathbb{Z}}(0_{\mathbb{Z}}) &\equiv 0_{\mathbb{Z}} & \text{neg}_{\mathbb{Z}}(\text{in-pos}(\text{succ}_{\mathbb{N}}(n))) &\equiv \text{in-neg}(\text{succ}_{\mathbb{N}}(n)) \\
\text{neg}_{\mathbb{Z}}(1_{\mathbb{Z}}) &\equiv -1_{\mathbb{Z}}
\end{aligned}$$

Now, we have defined the group operation for the integer.

**Definition 41.** ***(Dependent Pair Type)*** *Given a type family* $B$ *over* $A$*, we can define the dependent pair type (or* $\Sigma$*-Type) to be inductive type* $\Sigma_{(x:A)}B(x)$ *equipped with a pairing function:* $\text{pair} : \Pi_{(x:A)}\big(B(x) \to \Sigma_{(y:A)}B(y)\big)$*. The induction principle* $\Sigma_{(x:A)}B(x)$ *asserts that for any family of types* $P(p)$ *indexed by* $p : \Sigma_{(x:A)}B(x)$*, we have:*

$$\text{ind}_{\Sigma} : \Big(\Pi_{(x:A)}\Pi_{(y:B(x))}P(\text{pair}(x,y))\Big) \to \Big(\Pi_{(z:\Sigma_{(x:A)}B(x))}P(z)\Big)$$

*satisfy the following computation rules of* $\text{ind}_{\Sigma}(g, \text{pair}(x,y)) \equiv g(x,y)$*, where* $g : \prod_{x:A} B(x) \to P(\text{pair}(x,y))$ *and note that* $\text{pair}(x,y) \equiv \Sigma_{x:A}B(x)$*. Alternatively, a definition of a dependent function* $f : \Pi_{(z:\Sigma_{(x:A)}B(x))}P(z)$ *by induction using a function* $g : \Pi_{(x:A)}\Pi_{(y:B(x))}P((x,y))$ *can be presented as* $f(\text{pair}(x,y)) \equiv g(x,y)$

The induction principle of $\Sigma$-type can be used to define the projection function:

**Definition 42.** ***(Projection)*** *Consider a type* $A$ *and a type family* $B$ *over* $A$*, then we have: (1) The first projection map:* $\text{pr}_1 : (\Sigma_{(x:A)}B(x)) \to A$ *is defined by induction as* $\text{pr}_1(x,y) :\equiv x$*. (2) The second projection map:* $\text{pr}_2 : \Pi_{(p:\Sigma_{(x:A)}B(x))}B(\text{pr}_1(p))$ *is defined by induction as:* $\text{pr}_2 : (x,y) \equiv y$*.*

**Remark 19.** *(More Details on Projection Map) For the first projection map, define the map* $\Pi_{x:A}B(x) \to A$ *(the family of types in this case is* $A$*) to return only the first element* $A$ *i.e* $f_1 := \lambda a \lambda b.a$*, then we set* $\text{pr}_1 \equiv \text{ind}_{\Sigma}(f_1)$*. The case for the second projection map is the same.*

**Remark 20.** ***(Currying + Uncurrying)*** *If we want to construct a function:* $f : \Pi_{(z:\Sigma_{(x:A)}B(x))}P(z)$*, by* $\Sigma$*-induction, then we get to assume a pair* $(x,y)$ *consisting of* $x : A$ *and* $y : B(x)$ *and our goal will be to construct an element of type* $P(x,y)$*. Thus it is an converse of currying operation, which given to be:*

$$\text{ev-pair} : \Big(\Pi_{(z:\Sigma_{(x:A)}B(x))}P(z)\Big) \to \Big(\Pi_{(x:A)}\Pi_{y:B(x)}P(x,y)\Big)$$

*given by* $f \mapsto \lambda x.\lambda y.f(x,y)$*. Thus it is known as the uncurrying operation.*

Similar to other cases, we can consider a special case of $\Sigma$-type, which happens when $B$ is constant family over $A$. Thus, $\Sigma_{(x:A)}B$ is a type of an ordinary pair $(x,y)$ where $x : A$ and $y : B$.

**Definition 43.** ***(Cartesian Product)*** *Given* 2 *types* $A$ *and* $B$*. Then we define the cartesian product* $A \times B$ *of* $A$ *and* $B$ *by:* $A \times B := \Sigma_{(x:A)}B$*. It satisfies the induction principle of* $\Sigma$*-types, in which for any type family* $P$ *over* $A \times B$ *there is a function:*

$$\text{ind}_{\times} : \Big(\Pi_{(x:A)}\Pi_{(y:B)}P(x,y)\Big) \to \Big(\Pi_{(z:A\times B)}P(z)\Big)$$

*Satisfies the computation rule as:* $\text{ind}_{\times}(g,(x,y)) \equiv g(x,y)$*. The projection mapping is defined in similar way to* $\Sigma$*-type, above. Finally, when think of types as proposition,* $A \times B$ *is interpreted as conjunction of* $A$ *and* $B$*.*

**Proposition 4.** *$\neg(P \times \neg P)$ and $\neg(P \leftrightarrow \neg P)$*

*Proof.* We will consider the use of proof of negation, in which we will construct element of $\emptyset$ from the inner statement. **(Part 1):** Given the terms $a : P \times \neg P$, we can see that $\text{proj}_2(a)(\text{proj}_1(a)) : \emptyset$. Note that $a \equiv (p : P, f : P \to \emptyset)$.
**(Part 2):** Note that $P \leftrightarrow \neg P :\equiv (P \to \neg P) \times (\neg P \to P)$, given a term $a : P \leftrightarrow \neg P$, then $(p_1, p_2) \equiv a$. Then we have $p_1 : P \to (P \to \emptyset)$ and $p_2 : (P \to \emptyset) \to P$. We can define the function $f \equiv \lambda x.p_2(x, \cdot) : P \to P$, in which we can apply $p_1(f) : \emptyset$. Thus, we have created the term of empty type. □

**Proposition 5.** *We have the double negation monad, in which: $P \to \neg\neg P$ and $(P \to Q) \to (\neg\neg P \to \neg\neg Q)$ and $(P \to \neg\neg Q) \to (\neg\neg P \to \neg\neg Q)$*

*Proof.* For each of them we can make the following construction:

- We can construct $P \to ((P \to \emptyset) \to \emptyset)$ by setting it to be $\lambda x.\lambda f.f(x)$.
- We can construct $\big(P \to Q\big) \to \big(((P \to \emptyset) \to \emptyset) \to ((Q \to \emptyset) \to \emptyset)\big)$. Suppose we have $f : P \to Q, g : (P \to \emptyset) \to \emptyset$ and $h : Q \to \emptyset$. Then we can set $\lambda f.\lambda g.\lambda h.g((h \circ f))$.
- We have that: $\big(P \to ((Q \to \emptyset) \to \emptyset)\big) \to \big(((P \to \emptyset) \to \emptyset) \to ((Q \to \emptyset) \to \emptyset)\big)$. Suppose that we have $f : (Q \to \emptyset) \to \emptyset$ with $g : (P \to \emptyset) \to \emptyset$ and $h : Q \to \emptyset$ with $p : P$. We have: $\lambda p.\lambda f.\lambda g.\lambda h.g(p, f(h))$.

□

**Definition 44.** ***(List Type)*** *For any type $A$, we can define a type $\text{list}(A)$ of lists of elements of $A$ as inductive type with constructors of $\text{nil} : \text{list}(A)$ and $\text{cons} : A \to (\text{list}(A) \to \text{list}(A))$. Then the induction principle is:*

$$\text{ind}_{\text{list}(A)} : P(\text{nil}) \to \Big(\Pi_{a:A}\Pi_{l:P(\text{list}(A))}P(\text{con}(a, l))\Big) \to \big(\Pi_{l:\text{list}(A)}P(l)\big)$$

*The computation rule is that if we want to define $f : \Pi_{l:\text{list}(A)}P(l)$, we can define $f(\text{nil}) \equiv p_{\text{nil}}$ where $p_{\text{nil}} : P(\text{Nil})$ and $f(\text{cons}(a, l)) \equiv g(a, f(l))$ where $g : \Pi_{a:A}\Pi_{l:P(\text{list}(A))}P(\text{con}(a, l))$*

**Definition 45.** ***(Fold/Map/Length/Reverse)*** *These are the usual function on the list type: given $A$ and $B$ be types:*

- *(Fold): Suppose that $b : B$ and consider binary operation $\mu : A \to (B \to B)$. We define a function $\text{fold-list}(\mu) : \text{list}(A) \to B$ as (following from the induction principle):*

$$\text{fold-list}(\mu, \text{nil}) := b \qquad \text{fold-list}(\mu, \text{cons}(a, l)) := \mu(a, \text{fold-list}(\mu, l))$$

- *(Map): We can also define the operation: $\text{map-list} : (A \to B) \to (\text{list}(A) \to \text{list}(B))$ in which (following from the induction principle):*

$$\text{map-list}(f, \text{nil}) := \text{nil} \qquad \text{map-list}(f, \text{cons}(a, l)) := \text{cons}(f(a), \text{map-list}(f, l))$$

- *(Length): Finally, we have $\text{length-list} : \text{list}(A) \to \mathbb{N}$, where:*

$$\text{length-list}(\text{nil}) := 0 \qquad \text{length-list}(\text{cons}(a, l)) := 1 + \text{length-list}(l)$$

- *(Reverse): $\text{revese} := \text{fold-list}(\text{cons})$, note that that cons is append at the back with the base case of nil. That is we have the following sequence:*

$$\begin{aligned}\text{reverse}([1, 2, 3]) &= \text{cons}(1, \text{cons}(2, \text{cons}(3, [\cdot])))\\ &= \text{cons}(1, \text{cons}(2, [3])) = \text{cons}(1, [3, 2]) = [3, 2, 1]\end{aligned}$$

## 2.8 Identity Type

**Remark 21.** *(Some Notes on Identity) Given a type $A$, and 2 element $x, y : A$, $x = y$ should again be a type that depends on $x, y : A$. An element $p : x = y$ witness (identity) that $x$ and $y$ are equal elements of type $A$:*

- *There might be many element of type $x = y$, many identification of $x$ and $y$. And, since $x = y$ is itself a type, we can form the type $p = q$ for any 2 identification $p, q : x = y$.*
- *The situation is analogous to the situation in homotopy theory, where 2 points of a space can be connected by possible more than 1 path. And, between any 2 path from $x$ to $y$, there is a space of homotopies between then and so on.*

**Definition 46.** ***(Identity Type)*** *Given a type $A$ and let $a : A$, an identity type of $A$ at $a$ is an inductive family of types $a =_A x$ indexed by $x : A$ for which the constructor is $\text{refl}_a : a =_A a$. The induction principle postulates that for any family of types $P(x, p)$ indexed by $x : A$ and $p : a =_A x$ there is a function:*

$$\text{ind-eq}_a : P(a, \text{refl}_a) \to \Pi_{(x:A)}\Pi_{(p:a=_A x)}P(x, p)$$

*That satisfies the computation rule of $\text{ind-eq}_a(u, a, \text{refl}_a) \equiv u$, where $u : P(a, \text{refl}_a)$. An element of type $a =_A x$ is also called identification of $a$ with $x$ or path from $a$ to $x$.*

**Remark 22.** *(Observation on The Definition) (1) We have a type $a =_A x$ for any $x : A$, the constructor only provides an element $\text{refl}_a : a =_A a$, identifying $a$ with itself. (2) The injunction principle shows that to prove something about all identification of of $a$ with some $x : A$, it suffices to proof an assertion about $\text{refl}_a$ only.*

**Definition 47.** ***(Formal Rule for Identity Type)*** *We have following formation, induction, elimination and computation rule given here:*

$$\frac{\Gamma \vdash a : A}{\Gamma, x : A \vdash a =_A x \,\text{Type}} \qquad \frac{\Gamma \vdash a : A}{\Gamma \vdash \text{refl}_a : a =_A a} \qquad \frac{\Gamma \vdash a : A \qquad \Gamma, x : A, p : a =_A x \vdash P(x, p)\,\text{Type}}{\Gamma \vdash \text{ind-eq}_a : P(a, \text{refl}_a) \to \Pi_{(x:A)}\Pi_{(p:a=_A x)}P(x, p)}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma, x : A, p : a =_A x \vdash P(x, p)\,\text{Type}}{\Gamma \vdash \text{ind-eq}_a(p, a, \text{refl}_a) \equiv p : P(a, \text{refl}_a)}$$

**Remark 23.** *(Observation on Variable Type) It is also possible to form the identity type at a variable type $A$, rather than at an element. Since we can form the identity type in any context, we can form the identity type at variable $x : A$ as:*

$$\frac{\Gamma, x : A \vdash x : A}{\Gamma, x : A, y : A \vdash x =_A y\,\text{Type}} \qquad \frac{\Gamma, x : A \vdash x : A}{\Gamma, x : A \vdash \text{refl}_x : x =_A x}$$

*Or we can index its constructor in which we have the following induction rule, see RHS.*

Now we will show the groupoidal structure of types, that is we can define the concatenated and inverted operator, which corresponds to the transitivity and symmetry of the identity type.

**Definition 48.** ***(Concatenation Operation)*** *Let $A$ be a type. We define the concatenation operation, as:*

$$\text{concat} : \Pi_{(x,y,z:A)}(x = y) \to \big((y = z) \to (x = z)\big)$$

*where we denote $p \bullet q$ for $\text{concat}(p, q)$.*

***(Construction):*** *We start with function, for any $a : A$, see LHS. Then, by induction principle, it is enough to construct them as:*

$$f(x) : \Pi_{(y:A)}(x = y) \to \Pi_{(z:A)}(y = z) \to (x = z) \qquad \rightsquigarrow \qquad f(x, x, \text{refl}_x) : \Pi_{(z:A)}(x = z) \to (x = z)$$

*On RHS, we can have $\lambda z.\,\text{id}_{(x=z)}$. With the identity elimination, we can set $P(y, p) \equiv \Pi_{(z:A)}(y = z) \to (x = z)$ where $p : x =_A y$, but this means that $P(x, \text{refl}_x) \equiv \Pi_{(z:A)}(x = z) \to (x = z)$*

$$\text{ind-eq}_x : P(x, \text{refl}_x) \to \Pi_{y:A}\Pi_{p:x=_A y}P(y, p)$$
$$f(x) := \text{ind-eq}_x(\lambda z.\,\text{id}_{(x=z)}) : \Pi_{y:A}(x = y) \to \Pi_{(z:A)}(y = z) \to (x = z)$$

*we use swap function for the third and forth variable of $f$, where $\text{concat}_{x,y,z}(p, q) := f(x, y, p, z, q)$.*

**Remark 24.** *(Trick with Identity Type) Note that by defining the only on the $\text{refl}_x$, by the identity elimination, we can consider a more complex set of equality, for example $x =_A y$ for any $y$ i.e $P(y, p)$. That is, we consider the behavior of the function on $\text{refl}_x$ and then with that result, plug it to the $\text{ind-eq}_x(\cdot)$ to get the desired function. This is shown more clearly in different formulation of path induction (HoTT books showed that they are equal to each other):*

$$\text{path-ind} : \Big(\Pi_{x:A}P(x, x, \text{refl}_x)\Big) \to \Big(\Pi_{x,y:A}\Pi_{p:x=_A y}P(x, y, p)\Big)$$

**Definition 49.** ***(Inverse)*** *Let $A$ be a type. We denote the inverse operation as:*

$$\text{inv} : \Pi_{(x,y:A)}(x = y) \to (y = x)$$

***(Construction):*** *We note that we have:* $\text{inv}(x, x, \text{refl}_x) = \text{refl}_x$. *And so we can define* $P(y, p) \equiv (y = x)$, *then we have that* $P(x, \text{refl}_x) \equiv (x = x)$: $f(x) := \text{ind-eq}_x(\text{refl}_x) : \Pi_{y:A}(x = y) \to (y = x)$

**Remark 25.** *(Notes on Operation on Identification Type) The next question is whether the concatenation and inverting operations on identification behave as expected i.e Is concatenation of identification associative ? Does it satisfy the unit laws ? and Is inverse of an identification indeed a two-sided inverse ?*

*For example, in the case of associativity, we want to compare* $(p \bullet q) \bullet r$ *with* $p \bullet (q \bullet r)$, *given* $p : x = y, q : y = z$ *and* $r : z = w$ *in a type* $A$. *The composition rule aren't strong enough to conclude that* $(p \bullet q) \bullet r$ *and* $p \bullet (q \bullet r)$ *are judgmentally equal*

Since both $(p \bullet q) \bullet r$ and $p \bullet (q \bullet r)$ are element of the same type i.e $x = w$. we ask the question where there is an identification of: $(p \bullet q) \bullet r = p \bullet (q \bullet r)$. And the answer is yes, as we have:

**Definition 50.** ***(Associator)*** *Let $A$ be a type and consider 3 consecutive path as:*

$$x \overset{p}{=\!=} y \overset{q}{=\!=} z \overset{r}{=\!=} w$$

*in $A$. We define the associator as* $\text{assoc}(p, q, r) : (p \bullet q) \bullet r = p \bullet (q \bullet r)$.

***(Construction):*** *By the induction principle for identity types it suffices to show that:*

$$\Pi_{(z:A)}\Pi_{(q:x=z)}\Pi_{(w:A)}\Pi_{(r:z=w)}\ (\text{refl}_x \bullet q) \bullet r = \text{refl}_x \bullet (q \bullet r)$$

*Note that by the definition of concatenation, we have that* $\text{refl}_x \bullet q \equiv q$, *we have that:* $\big(\text{refl}_x \bullet q\big) \bullet r \equiv q \bullet r$. *And same for* $\text{refl}_x \bullet (q \bullet r) \equiv q \bullet r$, *which we have that* $\big(\text{refl}_x \bullet q\big) \bullet r \equiv \text{refl}_x \bullet (q \bullet r)$. *Then we can define the function above to be* $\text{assoc}(\text{refl}_x, q, r) := \text{refl}_{q \bullet r}$

**Definition 51.** ***(Unit Law Operation)*** *Given a type $A$, the left and right unit law operation which assigns for each $p : x = y$ as:*

$$\text{left-unit}(p) : \text{refl}_x \bullet p = p \qquad \text{right-unit}(p) : p \bullet \text{refl}_y = p$$

***(Construction):*** *It is suffices to construct:*

$$\text{left-unit}(\text{refl}_x) : \text{refl}_x \bullet \text{refl}_x = \text{refl}_x$$
$$\text{right-unit}(\text{refl}_x) : \text{refl}_x \bullet \text{refl}_x = \text{refl}_x$$

*Then we take* $\text{refl}_{\text{refl}_x}$.

**Definition 52.** ***(Inverse Law Opeartion)*** *Given a type $A$, we can define left and right inverse law operation, where we have:*

$$\text{left-inp}(p) : p^{-1} \bullet p = \text{refl}_y$$
$$\text{right-inp}(p) : p \bullet p^{-1} = \text{refl}_x$$

***(Construction):*** *It is suffices to construct as:*

$$\text{left-inp}(\text{refl}_x) : \text{refl}_x^{-1} \bullet \text{refl}_x = \text{refl}_x$$
$$\text{right-inp}(\text{refl}_x) : \text{refl}_x \bullet \text{refl}_x^{-1} = \text{refl}_x$$

*Via the computation rule, we have that:*

$$\text{refl}_x^{-1} \bullet \text{refl}_x \equiv \text{refl}_x \bullet \text{refl}_x \equiv \text{refl}_x \qquad \text{refl}_x \bullet \text{refl}_x^{-1} \equiv \text{refl}_x \bullet \text{refl}_x \equiv \text{refl}_x$$

*And so we can define* $\text{left-inv}(\text{refl}_x) \equiv \text{refl}_{\text{refl}_x}$ *and* $\text{right-inv}(\text{refl}_x) = \text{refl}_{\text{refl}_x}$

Note that the differences between the unit law and the inverse law is the $P(x, p)$

**Definition 53.** ***(Distributive Inverse Concatenation)*** *The operation inverting identifications distributes over the concatenation operation that is we have:*

$$\text{distributive-inv-concat}(p, q) : (p \bullet q)^{-1} = q^{-1} \bullet p^{-1}$$

*for any $p : x = y$ and $q : y = z$.*

***(Construction):*** *It is suffices to consider the construct as follows:*

$$\text{distributive-inv-concat}(\text{refl}_x, q) : (\text{refl}_x \bullet q)^{-1} = q^{-1} \bullet \text{refl}_x^{-1}$$

*In which by the computation rule, we have that:*

$$(\text{refl}_x \bullet q)^{-1} \equiv (q)^{-1} \qquad q^{-1} \bullet \text{refl}_x^{-1} \equiv q^{-1} \bullet \text{refl}_x \equiv q^{-1}$$

*Thus, we have that* $\text{distributive-inv-concat}(\text{refl}_x, q) = \text{refl}_{\text{inv}(q)}$

**Definition 54.** ***(Inverse Concatenation)*** *For any $p : x = y, q : y = z$ and $r : x = z$, construct the map:*

$$\text{inv-con}(p, q, r) : (p \bullet q = r) \to (q = p^{-1} \bullet r)$$
$$\text{con-inv}(p, q, r) : (p \bullet q = r) \to (q = r \bullet p^{-1})$$

***(Construction):*** *It is suffices to consider the construct as follows (together with the computation rule on the RHS as follows from inverse and concatenation):*

$$\text{inv-con}(\text{refl}_x, q, r) : (\text{refl}_x \bullet q = r) \to (q = \text{refl}_x^{-1} \bullet r) \equiv (q = r) \to (q = r)$$
$$\text{con-inv}(\text{refl}_x, q, r) : (\text{refl}_x \bullet q = r) \to (q = r \bullet \text{refl}_x^{-1}) \equiv (q = r) \to (q = r)$$

*Then we can set* $\text{inv-con}(\text{refl}_x, q, r) = \text{id}_{q=r}$ *and* $\text{con-inv}(\text{refl}_x, q, r) = \text{id}_{q=r}$.

## 2.9 Action on Path and Transport

**Remark 26.** *(Action on Identifications of Functions) We can show that every function preserves identification. In other words, every function sends identified element to identified elements. This is a form of continuity for function in type theory.*

**Definition 55.** ***(Action on Path)*** *Let $f : A \to B$ be a map. We define the action of paths of $f$ as an operation, as the first one, and other more. With the construction happens on the RHS:*

$$\text{ap}_f : \Pi_{(x,y:A)}(x = y) \to \big(f(x) = f(y)\big) \qquad \text{ap}_f(\text{refl}_x) := \text{refl}_{f(x)}$$
$$\text{ap-id}_A : \Pi_{(x,y:A)}\Pi_{(p:x=y)}\ p = \text{ap}_{\text{id}_A}(p) \qquad \text{ap-id}_A(\text{refl}_x) := \text{refl}_{\text{refl}_x}$$
$$\text{ap-comp}_A(f, g) : \Pi_{(x,y:A)}\Pi_{(p:x=y)}\ \text{ap}_g(\text{ap}_f(p)) = \text{ap}_{g\circ f}(p) \qquad \text{ap-comp}_A(\text{refl}_x) := \text{refl}_{\text{refl}_{g(f(x))}}$$

***(Construction):*** *We derive* $\text{ap}_f$ *by the induction principle of identity types, and the the rest follows.*

**Definition 56.** ***(Identifications for Action of Path)*** *Let $f : A \to B$ be a map. Then there are identification, with the following construction on the RHS:*

$$\text{ap-refl}(f, x) : \text{ap}_f(\text{refl}_x) = \text{refl}_{f(x)} \qquad \text{ap-refl}(f, x) \equiv \text{refl}_{\text{refl}_{f(x)}}$$
$$\text{ap-inv}(f, p) : \text{ap}_f(p^{-1}) = \text{ap}_f(p)^{-1} \qquad \text{ap-inv}(f, \text{refl}_x) := \text{refl}_{\text{refl}_{f(x)}}$$
$$\text{ap-concat}(f, p, q) : \text{ap}_f(p \bullet q) = \text{ap}_f(p) \bullet \text{ap}_f(q) \qquad \text{ap-concat}(f, \text{refl}_x, q) := \text{refl}_{\text{ap}_f(q)}$$

***(Construction):*** *The first one following from the computation rule of the action on path. For the second (LHS) and third (RHS) identification, we can see that, following from the computation rule:*

$$\text{ap}_f(\text{refl}_x^{-1}) \equiv \text{ap}_f(\text{refl}_x) \equiv \text{refl}_{f(x)} \qquad \text{ap}_f(\text{refl}_x \bullet q) \equiv \text{ap}_f(q)$$
$$\text{ap}_f(\text{refl}_x)^{-1} \equiv \text{refl}_{f(x)}^{-1} \equiv \text{refl}_{f(x)} \qquad \text{ap}_f(\text{refl}_x) \bullet \text{ap}_f(q) \equiv \text{refl}_{f(x)} \bullet \text{ap}_f(q) \equiv \text{ap}_f(q)$$

**Definition 57.** ***(Transport)*** *Let $A$ be a type and let $B$ be a type family over $A$. The transport operation is defined as:*

$$\text{tr}_B : \Pi_{(x,y:A)}(x = y) \to \big(B(x) \to B(y)\big)$$

*where we construct* $\text{tr}_B(p)$ *by as* $\text{tr}_B(\text{refl}_x) := \text{id}_{B(x)}$.

**Remark 27.** *(Some Notes on Transport Function) Dependent type also come with an action on identification: the transport functions. Given an identification $p : x = y$ in the base type $A$, we can transport any element $b : B(x)$ to the fibre $B(y)$.*

- *In type theory, we can't distinguish between identified elements $x$ and $y$ because for any type family $B$ over $A$ one obtains an element of $B(y)$ from the elements of $B(x)$.*

**Remark 28.** *(Some Application of Transport Function) We have the following scenario: consider the dependent function $f : \Pi_{(x:A)}B(x)$, with an identification $p : x =_A y$, it doesn't make sense to directly compare $f(x) : B(x)$ and $f(y) : B(y)$. But we can transport $f(x)$ along $p$, to get element $\mathrm{tr}_B(p, f(x)) : B(y)$. Then we can show that:*

**Definition 58.** ***(Identification of Dependent Function)*** *Given a dependent function $f : \Pi_{(a:A)}B(a)$ and an identification $p : x = y$ in, we can construct an identification as:*

$$\mathrm{apd}_f(p) : \mathrm{tr}_B(p, f(x)) = f(y)$$

*where the construction is given as: $\mathrm{apd}_f(\mathrm{refl}_x) := \mathrm{refl}_{f(x)}$, where $\mathrm{tr}_B(\mathrm{refl}_x, f(x)) \equiv \mathrm{id}_{B(x)}\, f(x) \equiv f(x)$*

**Remark 29.** *(Notes on Reflection Element) The identity type is an indices **family** of types. While the type $a = x$ is inductively generated by $\mathrm{refl}_a$, type $a = a$ isn't inductively generated by $\mathrm{refl}_a$. We can't use the inductive principle of identity type to show that $p = \mathrm{refl}_a$ for any $p : a = a$ because the end point of $p : a = a$ isn't free.*

Therefore, it is interesting to see how the reflexivity identification is unique. Note that the identification works as follows: we identify an element $a$ by giving the endpoint $x$ with which we seek to identify $a$, and then given the identification $p : a = x$. Thus the pair $(a, \mathrm{refl}_a)$ is unique in the type of all pairs as: $(x, p) : \Sigma_{(x:A)}a = x$ i.e

**Proposition 6.** *Consider an element $a : A$, then we can show that there is an identification, in the type $\Sigma_{x:A}a = x$ for every $y : \Sigma_{(x:A)}a = x$:*

$$(a, \mathrm{refl}_a) = y$$

*Proof.* By $\Sigma$-induction (as we set $g$ to be $\lambda y.(\mathrm{pr}_1(y), \mathrm{pr}_2(y))$), it suffices to show that there is an identification as: $(a, \mathrm{refl}_a) = \mathrm{pair}(x, p)$ for any $x : A$ and $p : a = x$. Then we can use the induction principle of identity types, where $(a, \mathrm{refl}_a) = (a, \mathrm{refl}_a)$, which is obtained by reflexivity. □

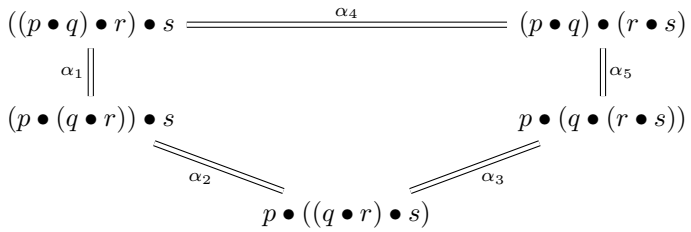
The proposition showed that, up to identification, there is only one element in $\Sigma$-type of the identity type. Such types are called contractible.

**Definition 59.** ***(Base Type Lift)*** *Let $B$ be a type family over $A$ and consider identification $p : a = x$ in $A$. For any $b : B(a)$ we have the identification of:*

$$\mathrm{lift}_B(p, b) : (a, b) = (x, \mathrm{tr}_B(p, b))$$

*The identification $p : x = y$ in the base type $A$ lifts to an identification in $\Sigma_{(x:A)}B(x)$ for all element in $B(x)$ (analogous to the path lifting property). We consider $\mathrm{tr}_B(\mathrm{refl}_a, b) \equiv b$ and so $\mathrm{lift}_B(\mathrm{refl}_a, b) := \mathrm{refl}_{(a,b)}$, as needed.*

**Definition 60.** ***(Mac Lane Pentagon)*** *Given the four consecutive identifications, with the construction on the RHS, we have*




$$\begin{aligned}
\alpha_1(p, q, r, \mathrm{refl}_x) &:= \mathrm{assoc}(p, q, r)\\
\alpha_2(p, q, \mathrm{refl}_x, s) &:= \mathrm{assoc}(p, q, s)\\
\alpha_3(\mathrm{refl}_x, q, r, s) &:= \mathrm{assoc}(q, r, s)\\
\alpha_4(p, \mathrm{refl}_x, r, s) &:= \mathrm{assoc}(p, r, s)\\
\alpha_5(p, q, r, \mathrm{refl}_x) &:= \mathrm{assoc}(p, q, r)
\end{aligned}$$

*Then we can show that $(\alpha_1 \bullet \alpha_2) \bullet \alpha_3 = \alpha_4 \bullet \alpha_5$. We are required to do double induction. Starting with assigning $\alpha_4 \equiv \mathrm{refl}_{(p\bullet q)\bullet(r\bullet s)}$, then we have to show that:*

$$\begin{aligned}
&\big[(\alpha_1 \bullet \alpha_2) \bullet \alpha_3\big](p, q, r, \mathrm{refl}_x) = \alpha_5(p, q, r, \mathrm{refl}_x)\\
\equiv\ &\mathrm{assoc}(p, q, r) = \mathrm{assoc}(p, q, r)
\end{aligned}$$

*The second line follows from the observation that: $\alpha_1$ behaves slightly difference as "domain" are now differences and thats all since $\alpha_2, \alpha_3$ doesn't make any changes as they are associator of $p, q, s$, and the witness on type on the RHS:*

$$(p \bullet q) \bullet (r \bullet \mathrm{refl}_x) \equiv (p \bullet q) \bullet r \xrightarrow{\mathrm{assoc}(p,q,r)} p \bullet (q \bullet r) \qquad \mathrm{refl}_{\text{ind-eq}(\mathrm{refl}_{\mathrm{assoc}(p,q,r)})}$$

## 2.10 Law of addition on $\mathbb{N}$

Now we have the have introduced the identity type, then we can proof the following identification:

$$\begin{aligned} 0 + n &= n & m + \mathrm{succ}_{\mathbb{N}}(n) &= \mathrm{succ}_{\mathbb{N}}(m + n) \\ m + 0 &= m & (m + n) + k &= m + (n + k) \\ \mathrm{succ}_{\mathbb{N}}(m) + n &= \mathrm{succ}_{\mathbb{N}}(m + n) & m + n &= n + m \end{aligned}$$

where we recall that the natural number can be defined as: $m + 0 \equiv m$ and $m + \mathrm{succ}_{\mathbb{N}}(n) \equiv \mathrm{succ}_{\mathbb{N}}(m + n)$. we will have to find way to apply these in our proof, as:

**Proposition 7.** *For any natural number $n$, there are identification:*

$$\begin{aligned} \text{left-unit-law-add}_{\mathbb{N}}(n) &: 0 + n = n \\ \text{right-unit-law-add}_{\mathbb{N}}(n) &: n + 0 = n \end{aligned}$$

*Proof.* Note that the right-unit laws follows directly from the fact that $\text{right-unit-law-add}_{\mathbb{N}}(n) := \mathrm{refl}_n$, as the rule on the natural number suggests. So, we are left with the left unit law. This can be done by induction on $n$:

- Base Case: This is simple we have that $\text{left-unit-law-add}_{\mathbb{N}}(0) : \mathrm{refl}_0$
- Step Case: The IH is that we have an identification on $p : 0 + n = n$, then we want to show that $0 + \mathrm{succ}_{\mathbb{N}}(n) = \mathrm{succ}_{\mathbb{N}}(n)$, then we use the action on path

  $$\mathrm{ap}_{\mathrm{succ}_{\mathbb{N}}}(p) : 0 + \mathrm{succ}_{\mathbb{N}}(n) \equiv \mathrm{succ}_{\mathbb{N}}(0 + n) = \mathrm{succ}_{\mathbb{N}}(n)$$

  This means that the left unit law can be defined as: $\text{left-unit-law-add} : \mathrm{ind}_{\mathbb{N}}(\mathrm{refl}_0, \lambda p.\, \mathrm{ap}_{\mathrm{succ}_{\mathbb{N}}}(p))$

□

**Proposition 8.** *For any natural number $n$ and $m$, there are identification as:*

$$\begin{aligned} \text{left-successor-law-add}_{\mathbb{N}}(m, n) &: \mathrm{succ}_{\mathbb{N}}(m) + n = \mathrm{succ}_{\mathbb{N}}(m + n) \\ \text{right-successor-law-add}_{\mathbb{N}}(m, n) &: m + \mathrm{succ}_{\mathbb{N}}(n) = \mathrm{succ}_{\mathbb{N}}(m + n) \end{aligned}$$

*Proof.* The right successor law follows directly from the rule of natural number so we don't have to do anything more. On the other hand, we will do the induction on the second variable to define left successor law:

- Base Case, we have the following construction

  $$\mathrm{ap}_{\mathrm{succ}_{\mathbb{N}}}(\text{right-unit-law-add}(m)) \bullet \mathrm{inv}\left(\text{right-unit-law-add}(\mathrm{succ}_{\mathbb{N}}(m))\right) : \mathrm{succ}_{\mathbb{N}}(m) + 0 = \mathrm{succ}_{\mathbb{N}}(m + 0)$$

  where $\mathrm{ap}_{\mathrm{succ}_{\mathbb{N}}}(\text{right-unit-law-add}(m)) : \mathrm{succ}_{\mathbb{N}}(m + 0) = \mathrm{succ}_{\mathbb{N}}(m)$ and $\text{right-unit-law-add}(\mathrm{succ}_{\mathbb{N}}(m)) : \mathrm{succ}_{\mathbb{N}}(m) + 0 = \mathrm{succ}_{\mathbb{N}}(m)$
- Step Case: we assume that $p : \mathrm{succ}_{\mathbb{N}}(m) + n = \mathrm{succ}_{\mathbb{N}}(m + n)$ as the IH and we want to show that $\mathrm{succ}_{\mathbb{N}}(m) + \mathrm{succ}_{\mathbb{N}}(n) = \mathrm{succ}_{\mathbb{N}}(m + \mathrm{succ}_{\mathbb{N}}(n))$, we have that:

  $$\begin{aligned} \mathrm{succ}_{\mathbb{N}}(m) + \mathrm{succ}_{\mathbb{N}}(n) &\equiv \mathrm{succ}_{\mathbb{N}}(\mathrm{succ}_{\mathbb{N}}(m) + n) \\ \mathrm{ap}_{\mathrm{succ}_{\mathbb{N}}}\left(\text{right-successor-law-add}_{\mathbb{N}}(m, n)\right) &: \mathrm{succ}_{\mathbb{N}}\left(m + \mathrm{succ}_{\mathbb{N}}(n)\right) = \mathrm{succ}_{\mathbb{N}}\left(\mathrm{succ}_{\mathbb{N}}(m + n)\right) \\ \mathrm{inv}(\mathrm{ap}_{\mathrm{succ}_{\mathbb{N}}}(p)) &: \mathrm{succ}_{\mathbb{N}}(\mathrm{succ}_{\mathbb{N}}(m + n)) = \mathrm{succ}_{\mathbb{N}}(\mathrm{succ}_{\mathbb{N}}(m) + n) \end{aligned}$$

  Then we have that:

  $$\begin{aligned} \text{right-successor-law-add} : \mathrm{ind}\Big( & \mathrm{ap}_{\mathrm{succ}_{\mathbb{N}}}(\text{right-unit-law-add}(m)) \bullet \mathrm{inv}\left(\text{right-unit-law-add}(\mathrm{succ}_{\mathbb{N}}(m))\right), \\ & \lambda p.\, \mathrm{ap}_{\mathrm{succ}_{\mathbb{N}}}\left(\text{right-successor-law-add}_{\mathbb{N}}(m, n)\right) \bullet \mathrm{inv}(\mathrm{ap}_{\mathrm{succ}_{\mathbb{N}}}(p))\Big) \end{aligned}$$

□

**Proposition 9.** *Addition on the natural number is associative, given 3 natural numbers $m, n$ and $k$. There is an identification of:*

$$\text{associative-add}_{\mathbb{N}}(m, n, k) : (m + n) + k = m + (n + k)$$

*Proof.* We will consider the proving this as an induction on $k$, in which we have:

- Base Case: $(m + n) + 0 = m + n$ and $m + (n + 0) = m + n$ following from the right unit law.
- Step Case, We will assume that $p : (m + n) + k = m + (n + k)$ and will show that: $(m + n) + \text{succ}_{\mathbb{N}}(k) = m + (n + \text{succ}_{\mathbb{N}}(k))$, we have following equalities:

$$\begin{aligned}(m + n) + \text{succ}_{\mathbb{N}}(k) &= \text{succ}_{\mathbb{N}}((m + n) + k) = \text{succ}_{\mathbb{N}}(m + (n + k)) \\ &= m + \text{succ}_{\mathbb{N}}(n + k) = m + (n + \text{succ}_{\mathbb{N}}(k))\end{aligned}$$

  Note that $\text{ap}_{\text{succ}_{\mathbb{N}}}(p) : \text{succ}_{\mathbb{N}}((m + n) + k) = \text{succ}_{\mathbb{N}}(m + (n + k))$ is used for the second equality.

□

**Proposition 10.** *Addition on the natural number is commutative i.e for any 2 natural number $n$ and $m$ there is an identification of:*

$$\text{commutative-add}_{\mathbb{N}}(m, n) : m + n = n + m$$

*Proof.* We will consider the induction on $m$, in which we have:

- Base Case: We simply have $0 + n = n + 0$, which is the unit law.
- Step Case: We will assume that $m + n = n + m$ and then will show that $\text{succ}_{\mathbb{N}}(m) + n = n + \text{succ}_{\mathbb{N}}(m)$, this follows from the following equalities:

$$\text{succ}_{\mathbb{N}}(m) + n = \text{succ}_{\mathbb{N}}(m + n) = \text{succ}_{\mathbb{N}}(n + m) = n + \text{succ}_{\mathbb{N}}(m)$$

  A combination of left and right successor law, and in the middle we have used action on path $\text{ap}_{\text{succ}_{\mathbb{N}}}(p) : \text{succ}_{\mathbb{N}}(m + n) = \text{succ}_{\mathbb{N}}(n + m)$

□

**Proposition 11.** *Before we proof the result of predecessor function and successor function, we can show that, for any natural number $n$:*

$$\text{succ}_{\mathbb{Z}}(\text{in-pos}(n)) \equiv \text{in-pos}(\text{succ}_{\mathbb{N}}(n)) \qquad \text{pred}_{\mathbb{Z}}(\text{in-neg}(n)) \equiv \text{in-neg}(\text{succ}_{\mathbb{N}}(k))$$

*Proof.* **(Part 1):** We will perform the induction on $n$, in which:

- Base Case: $\text{succ}_{\mathbb{Z}}(\text{in-pos}(0)) \equiv \text{in-pos}(\text{succ}_{\mathbb{N}}(0))$, where

$$\begin{aligned}&\text{succ}_{\mathbb{Z}}(\text{in-pos}(0)) \equiv \text{succ}_{\mathbb{Z}}(1_{\mathbb{Z}}) \equiv \text{in-pos}(1_{\mathbb{N}}) \\ &\text{in-pos}(\text{succ}_{\mathbb{N}}(0)) \equiv \text{in-pos}(1_{\mathbb{N}})\end{aligned}$$

- Step Case: We the rule that we have defined defined:

$$\text{succ}_{\mathbb{Z}}(\text{in-pos}(\text{succ}_{\mathbb{N}}(n))) \equiv \text{in-pos}(\text{succ}_{\mathbb{N}}(\text{succ}_{\mathbb{N}}(n)))$$

**(Part 2):** Using the induction on $n$, we have:

- Base Case: $\text{pred}_{\mathbb{Z}}(\text{in-neg}(0_{\mathbb{N}})) \equiv \text{in-neg}(\text{succ}_{\mathbb{N}}(0_{\mathbb{N}}))$, where we have that:

$$\begin{aligned}&\text{pred}_{\mathbb{Z}}(\text{in-neg}(0_{\mathbb{N}})) \equiv \text{pred}_{\mathbb{Z}}(-1_{\mathbb{Z}}) \\ &\text{in-neg}(\text{succ}_{\mathbb{N}}(0_{\mathbb{N}})) \equiv \text{in-neg}(1_{\mathbb{N}}) \equiv \text{pred}_{\mathbb{Z}}(-1_{\mathbb{Z}})\end{aligned}$$

- Step Case: We the rule that we have defined defined:

$$\text{pred}_{\mathbb{Z}}(\text{in-neg}(\text{succ}_{\mathbb{N}}(n))) \equiv \text{in-neg}(\text{succ}_{\mathbb{N}}(\text{succ}_{\mathbb{N}}(n)))$$

□

**Proposition 12.** *Given the predecessor defined in definition 38, we will show that:*

$$\text{succ}_{\mathbb{Z}}(\text{pred}_{\mathbb{Z}}(k)) = k \qquad \text{pred}_{\mathbb{Z}}(\text{succ}_{\mathbb{Z}}(k)) = k$$

*Proof.* **(Part 1):** Let's start with the first one, where we will perform an induction on $k$, we have the following:

- Base Case: We have that $\text{succ}_{\mathbb{Z}}(\text{pred}_{\mathbb{Z}}(0_{\mathbb{Z}})) \equiv \text{succ}_{\mathbb{Z}}(-1_{\mathbb{Z}}) \equiv 0_{\mathbb{Z}}$
- Step Case: since there are 2 kinds of number (positive and negative), we have to consider them both, that is given $\text{succ}_{\mathbb{Z}}(\text{pred}_{\mathbb{Z}}(\text{in-neg}(n))) = \text{in-neg}(n)$ and $\text{succ}_{\mathbb{Z}}(\text{pred}_{\mathbb{Z}}(\text{in-pos}(n))) = \text{in-pos}(n)$, we have that:

$$\begin{aligned}\text{succ}_{\mathbb{Z}}(\text{pred}_{\mathbb{Z}}(\text{in-neg}(\text{succ}_{\mathbb{N}}(n)))) &\equiv \text{succ}_{\mathbb{Z}}\left(\text{in-neg}(\text{succ}_{\mathbb{N}}(\text{succ}_{\mathbb{N}}(n)))\right)\\ &\equiv \text{in-neg}(\text{succ}_{\mathbb{N}}(n))\end{aligned}$$

$$\begin{aligned}\text{succ}_{\mathbb{Z}}(\text{pred}_{\mathbb{Z}}(\text{in-pos}(\text{succ}_{\mathbb{N}}(n)))) &\equiv \text{succ}_{\mathbb{Z}}(\text{in-pos}(n))\\ &\equiv \text{in-pos}(\text{succ}_{\mathbb{N}}(n))\end{aligned}$$

  Note that for the last equation we have used the proposition above.

**(Part 2):** On the other hand, we will, similarly, perform the induction on $k$, which we have the following:

- Base Case: We have that $\text{pred}_{\mathbb{Z}}(\text{succ}_{\mathbb{Z}}(0_{\mathbb{Z}})) \equiv \text{pred}_{\mathbb{Z}}(1_{\mathbb{Z}}) \equiv 0_{\mathbb{Z}}$
- Step Case: We have that:

$$\begin{aligned}\text{pred}_{\mathbb{Z}}(\text{succ}_{\mathbb{Z}}(\text{in-neg}(\text{succ}_{\mathbb{N}}(n)))) &\equiv \text{pred}_{\mathbb{Z}}(\text{in-neg}(n))\\ &\equiv \text{in-neg}(\text{succ}_{\mathbb{N}}(n))\end{aligned}$$

$$\begin{aligned}\text{pred}_{\mathbb{Z}}(\text{succ}_{\mathbb{Z}}(\text{in-pos}(\text{succ}_{\mathbb{N}}(n)))) &\equiv \text{pred}_{\mathbb{Z}}(\text{in-pos}(\text{succ}_{\mathbb{N}}(\text{succ}_{\mathbb{N}}(n))))\\ &\equiv \text{in-pos}(\text{succ}_{\mathbb{N}}(n))\end{aligned}$$

  Note that for the last equation we have used the proposition above.

□

# 3 Universes

We can think of universe as the types that consists of types.

**Definition 61.** ***(Universe Type Family)*** *Given element $X : \mathcal{U}$, $\mathcal{T}(X)$ is thought as the type of element $X$. The family $\mathcal{T}$ is called a universal type family.*

**Remark 30.** *(Usefulness of Universe) We need universe because:*

- *Define a new type family over inductive type via inductive principles. For example, we can used to define ordering relation $\leq$ and $<$ of $\mathbb{N}$:*
  - *Allow us to define observation equality, denoted as $\text{Eq}_{\mathbb{N}}$ on $\mathbb{N}$, which can be used to show that $0_{\mathbb{N}} \neq 1_{\mathbb{N}}$. This helps us to think about identity type, as it can be used to characterized the identity type of $\mathbb{N}$*
  - *Characterizing identity type is one of the main themes in HoTT*
- *Universe also allows us to define many types of types equipped with structure. For example type of groups, where we can have the operation satisfying the group law, with underlying type is a set.*

*Finally, universe got the useful feature as they are defined to be closed under all types constructors, as we will see below, in the definition. However, it is inconsistent to assume that universe is contained in itself (as we might have Russell's paradox).*

**Definition 62.** ***(Universe/Universe Family)*** *A universe is a type $\mathcal{U}$ in the empty context, equipped with a type family $\mathcal{T}$ over $\mathcal{U}$ called a universal family:*

- *$\mathcal{U}$ is closed under $\Pi$, that is it is equipped with the function: $\check{\Pi} : \Pi_{(X:\mathcal{U})}\big(\mathcal{T}(X) \to \mathcal{U}\big) \to \mathcal{U}$ with the judgmental equality of: $\mathcal{T}\big(\check{\Pi}(X, Y)\big) \equiv \Pi_{(x:\mathcal{T}(X))}\mathcal{T}(Y(x))$*
- *$\mathcal{U}$ is closed under $\Sigma$, that is it is equipped with the function: $\check{\Sigma} : \Pi_{(X:\mathcal{U})}\big(\mathcal{T}(X) \to \mathcal{U}\big) \to \mathcal{U}$ with the judgmental equality of: $\mathcal{T}\big(\check{\Sigma}(X, Y)\big) \equiv \Sigma_{(x:\mathcal{T}(X))}\mathcal{T}(Y(x))$*

- $\mathcal{U}$ *is closed under identity type, that is it is equipped with the function:* $\check{I} : \Pi_{(X:\mathcal{U})}\mathcal{T}(X) \to \big(\mathcal{T}(X) \to \mathcal{U}\big)$ *with the judgmental equality of:* $\mathcal{T}\big(\check{I}(X,x,y)\big) \equiv (x = y)$
- $\mathcal{U}$ *is closed under co-products, that is it is equipped with function:* $\check{+} : \mathcal{U} \to \big(\mathcal{U} \to \mathcal{U}\big)$ *with the judgmental equality of:* $\mathcal{T}(X \check{+} Y) \equiv \mathcal{T}(X) + \mathcal{T}(Y)$
- $\mathcal{U}$ *contains elements* $\check{\emptyset}, \check{\mathbb{1}}, \check{\mathbb{N}} : \mathcal{U}$ *with the judgmental equality of:*

$$\mathcal{T}(\check{\emptyset}) \equiv \emptyset \qquad \mathcal{T}(\check{\mathbb{1}}) \equiv \mathbf{1} \qquad \mathcal{T}(\check{\mathbb{N}}) \equiv \mathbb{N}$$

**Definition 63.** ***(Contains)*** *Given a type* $A$ *and universe* $\mathcal{U}$*, we say that* $A$ *is a type in* $\mathcal{U}$ *or* $\mathcal{U}$ *contains* $A$ *if* $\mathcal{U}$ *comes equipped with element* $\check{A} : \mathcal{U}$*, in context for which:*

$$\Gamma \vdash \mathcal{T}(\check{A}) \equiv A\,\mathrm{Type}$$

*holds. If* $A$ *is a type in* $\mathcal{U}$*, we usually write* $A$ *for* $\check{A}$ *and* $\mathcal{T}(\check{A})$*.*

Although we can't have universe within itself, nor getting by only a single universe, it is convenient if every type, including any universe, is in some universe. That is, we will assume that there are sufficiently many universes.

**Postulate 1.** *For every finite list of types in context as we may have:*

$$\Gamma_1 \vdash A_1\,\mathrm{Type} \quad \cdots \quad \Gamma_n \vdash A_n\,\mathrm{Type}$$

*There is a universe that contains each* $A_i$ *that is* $\mathcal{U}$ *equipped with* $\Gamma_i \vdash \check{A}_i : \mathcal{U}$ *for which the judgment of* $\Gamma_i \vdash \mathcal{T}(\check{A}_i) \equiv A\,\mathrm{Type}$ *holds.*

With this, we will rarely need to work with more than one universe at the same time. We can obtain many specific universe as:

**Definition 64.** ***(Base Universe)*** *The base universe* $\mathcal{U}_0$ *is the universe that we obtain using postulate above with the empty list of type in context.*

That is it is closed under all the ways of forming types, but it isn't specified to contain any future types.

**Definition 65.** ***(Successor Universe)*** *The successor universe of a universe* $\mathcal{U}$ *is the universe obtained by a postulate above with the finite list of.*

$$\vdash \mathcal{U}\,\mathrm{Type}$$
$$X : \mathcal{U} \vdash \mathcal{T}(X)\,\mathrm{Type}$$

*It is usually denoted as* $\mathcal{U}^+$*.*

**Remark 31.** *(What successor universe contains): The successor universes* $\mathcal{U}^+$*, therefore, contain type* $\mathcal{U}$ *as well as any types in* $\mathcal{U}$ *in the sense that:*

$$\vdash \check{\mathcal{U}} : \mathcal{U}^+ \qquad \vdash \mathcal{T}^+(\check{\mathcal{U}}) \equiv \mathcal{U}\,\mathrm{Type}$$
$$X : \mathcal{U} \vdash \check{\mathcal{T}}(X) : \mathcal{U}^+ \qquad X : \mathcal{U} \vdash \mathcal{T}^+(\check{\mathcal{T}}(X)) \equiv \mathcal{T}(X)\,\mathrm{Type}$$

*We also get the function* $i : \mathcal{U} \to \mathcal{U}^+$ *that includes the type in* $\mathcal{U}$ *into* $\mathcal{U}^+$ *and it is defined as* $i := \lambda X.\check{\mathcal{T}}(X)$*, and with this successor universe, we can create an infinite tower:* $\mathcal{U}, \mathcal{U}^+, \mathcal{U}^{++}, \ldots$ *of universes.*

The tower of universe need not to be exhaustive i.e every types is contained in a universe in this tower.

**Definition 66.** ***(Join)*** *The join of 2 universes* $\mathcal{U}$ *and* $\mathcal{V}$ *is the universes* $\mathcal{U} \sqcup \mathcal{V}$ *obtained using postulate above, where we have:*

$$X : \mathcal{U} \vdash \mathcal{T}_{\mathcal{U}}(X)\,\mathrm{Type}$$
$$Y : \mathcal{V} \vdash \mathcal{T}_{\mathcal{V}}(Y)\,\mathrm{Type}$$

**Remark 32.** *(Inclusion Function of Join Universe) Since the join* $\mathcal{U} \sqcup \mathcal{V}$ *contains all the types in* $\mathcal{U}$ *and* $\mathcal{V}$*, there is a maps of* $i : \mathcal{U} \to \mathcal{U} \sqcup \mathcal{V}$ *and* $j : \mathcal{V} \to \mathcal{U} \sqcup \mathcal{V}$*. Note that we don't postulate any relations between universes and in general, it is the case that* $(\mathcal{U} \sqcup \mathcal{V}) \sqcup \mathcal{W}$ *and* $\mathcal{U} \sqcup (\mathcal{V} \sqcup \mathcal{W})$ *will be unrelated.*

## 3.1 Observational Equality of the Natural Numbers

**Remark 33.** *(Intuition Behind Observational Equality) Via the notion of universe, we can define many relation on natural number, where we will start with observational equality of* $\mathbb{N}$*. To proof that* $m$ *and* $n$ *are observational equal, we can look of how they are constructed.*

**Definition 67.** ***(Observational Equality)*** *The observational equality of* $\mathbb{N}$ *is a binary relation with the type of* $\mathrm{Eq}_{\mathbb{N}} : \mathbb{N} \to \left(\mathbb{N} \to \mathcal{U}_0\right)$ *such that:*

$$\begin{aligned} \mathrm{Eq}_{\mathbb{N}}(0_{\mathbb{N}}, 0_{\mathbb{N}}) &\equiv \mathbf{1} & \mathrm{Eq}_{\mathbb{N}}(\mathrm{succ}_{\mathbb{N}}(n), 0_{\mathbb{N}}) &\equiv \emptyset \\ \mathrm{Eq}_{\mathbb{N}}(0_{\mathbb{N}}, \mathrm{succ}_{\mathbb{N}}(n)) &\equiv \emptyset & \mathrm{Eq}_{\mathbb{N}}(\mathrm{succ}_{\mathbb{N}}(n), \mathrm{succ}_{\mathbb{N}}(m)) &\equiv \mathrm{Eq}_{\mathbb{N}}(n, m) \end{aligned}$$

***(Construction):*** *We define* $\mathrm{Eq}_{\mathbb{N}}$ *by double induction on* $\mathbb{N}$*, that is it is suffices to provide:*

$$\begin{aligned} E_0 &: \mathbb{N} \to \mathcal{U}_0 \\ E_S &: \mathbb{N} \to \Big((\mathbb{N} \to \mathcal{U}_0) \to (\mathbb{N} \to \mathcal{U}_0)\Big) \end{aligned}$$

*Both function can be defined inductively as, furthermore, the observational equality itself can be constructed based on these two types (see the right most side).*

$$\begin{aligned} E_0(0_{\mathbb{N}}) &\equiv \mathbf{1} & E_S(n, X, 0_{\mathbb{N}}) &\equiv \emptyset & \mathrm{Eq}_{\mathbb{N}}(0_{\mathbb{N}}, m) &\equiv E_0(m) \\ E_0(\mathrm{succ}_{\mathbb{N}}(n)) &\equiv \emptyset & E_S(n, X, \mathrm{succ}_{\mathbb{N}}(m)) &\equiv X(m) & \mathrm{Eq}_{\mathbb{N}}(\mathrm{succ}_{\mathbb{N}}(n), m) &\equiv E_S(n, \mathrm{Eq}_{\mathbb{N}}(n), m) \end{aligned}$$

**Remark 34.** *(On the need of Universe) We can see here that by having a universe of types, we can "bootstrap" the type* $\mathrm{Eq}(n)$ *as we have:*

$$\mathrm{Eq}_{\mathbb{N}}(\mathrm{succ}_{\mathbb{N}}(n), \mathrm{succ}_{\mathbb{N}}(m)) \equiv E_S(n, \mathrm{Eq}_{\mathbb{N}}(n), \mathrm{succ}_{\mathbb{N}}(m)) \equiv \mathrm{Eq}_{\mathbb{N}}(n)(m)$$

The observational equality of the natural number is important because it can be used to prove equalities and negations of equalities (see proposition below). Consider:

**Lemma 9.** *Observational equality of* $\mathbb{N}$ *is a reflexive relation:*

$$\text{refl-}\mathrm{Eq}_{\mathbb{N}} : \Pi_{(n:\mathbb{N})} \mathrm{Eq}_{\mathbb{N}}(n, n)$$

*Proof.* The function $\text{refl-}\mathrm{Eq}_{\mathbb{N}}$ is defined inductively on $n$ as follows:

$$\text{refl-}\mathrm{Eq}_{\mathbb{N}}(0_{\mathbb{N}}) \equiv * \qquad \text{refl-}\mathrm{Eq}_{\mathbb{N}}(\mathrm{succ}_{\mathbb{N}}(n)) \equiv \text{refl-}\mathrm{Eq}_{\mathbb{N}}(n)$$

□

*Proof.* (Alternative By Me) That is we need to show that:

$$\frac{\Gamma, n : \mathbb{N} \vdash * : \mathbf{1} \equiv \mathrm{Eq}_{\mathbb{N}}(n, n)}{\Gamma \vdash \text{refl-}\mathrm{Eq}_{\mathbb{N}} \equiv \lambda n.* : \Pi_{(n:\mathbb{N})} \mathrm{Eq}_{\mathbb{N}}(n, n)}\ \lambda$$

We will claim that $\mathrm{Eq}_{\mathbb{N}}(n, n) \equiv \mathbf{1}$. This is done by induction on $n$. That is: Base Case: $\mathrm{Eq}_{\mathbb{N}}(0_{\mathbb{N}}, 0_{\mathbb{N}}) \equiv \mathbf{1}$ per definition. On the other hand, for step case, with IH to be $\mathrm{Eq}_{\mathbb{N}}(n, n) \equiv \mathbf{1}$, then we can show that $\mathrm{Eq}_{\mathbb{N}}(\mathrm{succ}_{\mathbb{N}}(n), \mathrm{succ}_{\mathbb{N}}(n)) \equiv \mathbf{1}$ where:

$$\mathrm{Eq}_{\mathbb{N}}(\mathrm{succ}_{\mathbb{N}}(n), \mathrm{succ}_{\mathbb{N}}(n)) \equiv \mathrm{Eq}_{\mathbb{N}}(n, n) \equiv \mathbf{1}$$

□

**Proposition 13.** *For any 2 natural number* $m$ *and* $n$*, we have that:*

$$(m = n) \leftrightarrow \mathrm{Eq}_{\mathbb{N}}(m, n)$$

*Proof.* We can define $f \equiv \lambda x.* : (m = m) \to \mathbf{1}$, where we have $f(\mathrm{refl}_m) = *$. On the other hand, the function $\mathrm{Eq}_{\mathbb{N}}(m, n) \to (m = n)$ can be defined by induction on $m$ and $n$, where we have the following cases:

- Base Case: $\mathrm{Eq}_{\mathbb{N}}(0_{\mathbb{N}}, 0_{\mathbb{N}}) \equiv \mathbf{1}$, then we have $\mathbf{1} \to \mathrm{refl}_{0_{\mathbb{N}}}$. Furthermore, $\mathrm{Eq}_{\mathbb{N}}(0_{\mathbb{N}}, \mathrm{succ}_{\mathbb{N}}(n)) \equiv \emptyset$, then we have the $\emptyset \to (m = n)$, in which by induction principle, we have a term. We can define the terms in similar manners if the second argument is $0_{\mathbb{N}}$.

- Step case, the IH is that we assume to have $f : \mathrm{Eq}_{\mathbb{N}}(m,n) \to (m = n)$, in which, we want to define $f : \mathrm{Eq}_{\mathbb{N}}(\mathrm{succ}_{\mathbb{N}}(m), \mathrm{succ}_{\mathbb{N}}(n)) \to (\mathrm{succ}_{\mathbb{N}}(m) = \mathrm{succ}_{\mathbb{N}}(n))$, this can be done as follows:

$$\begin{array}{ccc} \mathrm{Eq}_{\mathbb{N}}(\mathrm{succ}_{\mathbb{N}}(m), \mathrm{succ}_{\mathbb{N}}(n)) & \dashrightarrow & \mathrm{succ}_{\mathbb{N}}(m) = \mathrm{succ}_{\mathbb{N}}(n) \\ \downarrow{\scriptstyle \mathrm{id}} & & \uparrow{\scriptstyle \mathrm{ap}_{\mathrm{succ}_{\mathbb{N}}}} \\ \mathrm{Eq}_{\mathbb{N}}(m,n) & \xrightarrow[f]{} & m = n \end{array}$$

  The map on the left is identity as we have the judgmental equality

□

Using the observational equality of $\mathbb{N}$, we can prove Peano's seventh and eighths axioms, which is stated and proved in the next 2 theorems.

**Theorem 1.** *For any two $m, n \in \mathbb{N}$, we have:*

$$(m = n) \leftrightarrow (\mathrm{succ}_{\mathbb{N}}(m) = \mathrm{succ}_{\mathbb{N}}(n))$$

*Proof.* Starting with the function $\mathrm{ap}_{\mathrm{succ}_{\mathbb{N}}} : (m = n) \to (\mathrm{succ}_{\mathbb{N}}(m) = \mathrm{succ}_{\mathbb{N}}(n))$. On the other hand, we want to construct the function $(\mathrm{succ}_{\mathbb{N}}(m) = \mathrm{succ}_{\mathbb{N}}(n)) \to (m = n)$. Note that we can define it as:

$$\begin{array}{ccc} (\mathrm{succ}_{\mathbb{N}}(m) = \mathrm{succ}_{\mathbb{N}}(n)) & \dashrightarrow & (m = n) \\ \downarrow & & \uparrow \\ \mathrm{Eq}_{\mathbb{N}}(\mathrm{succ}_{\mathbb{N}}(m), \mathrm{succ}_{\mathbb{N}}(n)) & \xrightarrow[\mathrm{id}]{} & \mathrm{Eq}_{\mathbb{N}}(m,n) \end{array}$$

where the left and right arrows comes from the proposition above □

**Theorem 2.** *For any $n \in \mathbb{N}$, we have $0_{\mathbb{N}} \neq \mathrm{succ}_{\mathbb{N}}(n)$*

*Proof.* Please note that, if $0_{\mathbb{N}} = \mathrm{succ}_{\mathbb{N}}(n)$, then by proposition above, there is a function $(0_{\mathbb{N}} = \mathrm{succ}_{\mathbb{N}}(n)) \to \mathrm{Eq}_{\mathbb{N}}(0_{\mathbb{N}} = \mathrm{succ}_{\mathbb{N}}(n)) \equiv \emptyset$. This implies that $0_{\mathbb{N}} = \mathrm{succ}_{\mathbb{N}}(n) \equiv \emptyset$, therefore $0_{\mathbb{N}} \neq \mathrm{succ}_{\mathbb{N}}(n)$, as needed. □

**Proposition 14.** *Given $m, n, k \in \mathbb{N}$ where: $(m = n) \leftrightarrow (m + k = n + k)$. That is, adding $k$ is an injective function.*

*Proof.* Starting with the function $(m = n) \to (m + k = n + k)$, then we can consider induction on identity type, that is $(m = m) \to (m + k = m + k)$ to define this function, we will perform an induction on $m$:

- Base Case, We want to show that $(0 = 0) \to (0 + k = 0 + k) \equiv (k = k)$, again this can be proven by induction on $k$ as we have:
  - Base Case: $\mathrm{id}_{\mathrm{refl}_0} : (0 = 0) \to (0 = 0)$
  - Step Case: Given $p : (0 = 0) \to (k = k)$, then $\mathrm{ap}_{\mathrm{succ}_{\mathbb{N}}} \circ p : (0 = 0) \to (\mathrm{succ}_{\mathbb{N}}(k) = \mathrm{succ}_{\mathbb{N}}(k))$, as needed.
- Step Case, we assume that $p : (m = m) \to (m + k = m + k)$, then we want to show that $(\mathrm{succ}_{\mathbb{N}}(m) = \mathrm{succ}_{\mathbb{N}}(m)) \to (\mathrm{succ}_{\mathbb{N}}(m) + k = \mathrm{succ}_{\mathbb{N}}(m) + k)$, in which, we have that:

$$\begin{array}{ccccc} (\mathrm{succ}_{\mathbb{N}}(m) = \mathrm{succ}_{\mathbb{N}}(m)) & & \dashrightarrow & & (\mathrm{succ}_{\mathbb{N}}(m) + k = \mathrm{succ}_{\mathbb{N}}(m) + k) \\ \downarrow & & & & \uparrow{\scriptstyle \mathrm{id}} \\ \mathrm{Eq}_{\mathbb{N}}(\mathrm{succ}_{\mathbb{N}}(m), \mathrm{succ}_{\mathbb{N}}(m)) & & & & (\mathrm{succ}_{\mathbb{N}}(m + k) = \mathrm{succ}_{\mathbb{N}}(m + k)) \\ \downarrow{\scriptstyle \mathrm{id}} & & & & \uparrow{\scriptstyle \mathrm{ap}_{\mathrm{succ}_{\mathbb{N}}}} \\ \mathrm{Eq}_{\mathbb{N}}(m,m) & \longrightarrow & (m = m) & \xrightarrow[p]{} & (m + k = m + k) \end{array}$$

□

**Proposition 15.** *Given $m, n \in \mathbb{N}$, then $(m + n = 0) \leftrightarrow (m = 0) \times (n = 0)$*

*Proof.* Let's start with $(m+n=0) \to (m=0) \times (n=0)$, note that if we can define $g_1 : (m+n=0) \to (m=0)$ and $g_2 : (m+n=0) \to (n=0)$, then we can define (that is using a pair function):

$$\lambda.x(g_1(x), g_2(x)) : (m+n=0) \to (m=0) \times (n=0)$$

as needed. We will provide the construction just for the $g_1$ case because the other is done similarly. We will perform induction on $n$:

- Base Case: We have that $(m+0=0) \to (m=0)$, suppose we are given a term $a : (m+0=0)$, then we can concatenate with $\text{inv}(\text{right-unit-law-add}_{\mathbb{N}}(m)) : m = m+0$ to get $m=0$. That is the function of

$$\lambda x.\, \text{right-unit-law-add}_{\mathbb{N}}(m) \bullet x : (m+0=0) \to (m=0)$$

- Step Case: Given the IH of $p : (m+n=0) \to (m=0)$, we want to construct $(m+\text{succ}_{\mathbb{N}}(n)=0) \to (m=0)$, which is created by the following composition:

$$\begin{array}{ccc} (m+\text{succ}_{\mathbb{N}}(n)=0) \equiv (\text{succ}_{\mathbb{N}}(m+n)=0) & & \\ \Big\downarrow & & \\ \text{Eq}_{\mathbb{N}}(\text{succ}_{\mathbb{N}}(m+n),0) \equiv \emptyset & \xrightarrow[\text{ex-falso}]{} & (m=0) \end{array}$$

On the other hand, we want to proof that $(m=0) \times (n=0) \to (m+n=0)$. First, we can define the type $f : (m=0) \to (m+n=0+n)$ via induction:

- Base Case: Starting with the helper function add-zero $= \lambda x.x+0$, with the action of path we can define the term $\text{ap}_{\text{add-zero}} : (m=0) \to (m+0=0+0)$.
- Step Case: we will assume that that is a type $p : (m=0) \to (m+n=0+n)$, then we want to find a term for the type $(m=0) \to (m+\text{succ}_{\mathbb{N}}(n)=0+\text{succ}_{\mathbb{N}}(n))$. Then, we have:

$$\begin{array}{ccc} (m=0) & \dashrightarrow & (m+\text{succ}_{\mathbb{N}}(n)=0+\text{succ}_{\mathbb{N}}(n)) \\ {\scriptstyle p}\big\downarrow & & \big\uparrow \\ (m+n=0+n) & \xrightarrow[\text{ap}_{\text{succ}_{\mathbb{N}}}]{} & (\text{succ}_{\mathbb{N}}(m+n)=\text{succ}_{\mathbb{N}}(0+n)) \end{array}$$

  The RHS arrow is just for the concatenation of various equality.

Then we can construct the function as follows: given $a : (m=0)$ and $b : (n=0)$, and so

$$\lambda(a,b).f(a) \bullet \text{left-unit-law-add}_{\mathbb{N}}(n) \bullet b : (m=0) \times (n=0) \to (m+n=0)$$

where note that $f(a) : (m+n=0+n)$ and with the application of concatenation of identity types, as follows (note that $l$ is the left unit law that we have proved):

$$m+n \overset{f(a)}{=\!=} 0+n \overset{l}{=\!=} n \overset{b}{=\!=} 0$$

□